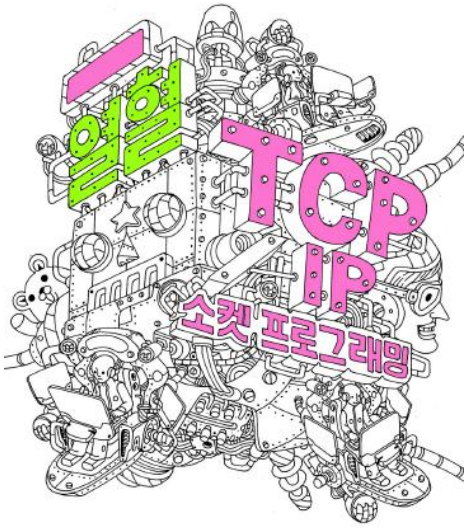
윤성우의 열혈 TCP/IP 소켓 프로그래밍

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Chapter 06. UDP 기반 서버/클라이언트

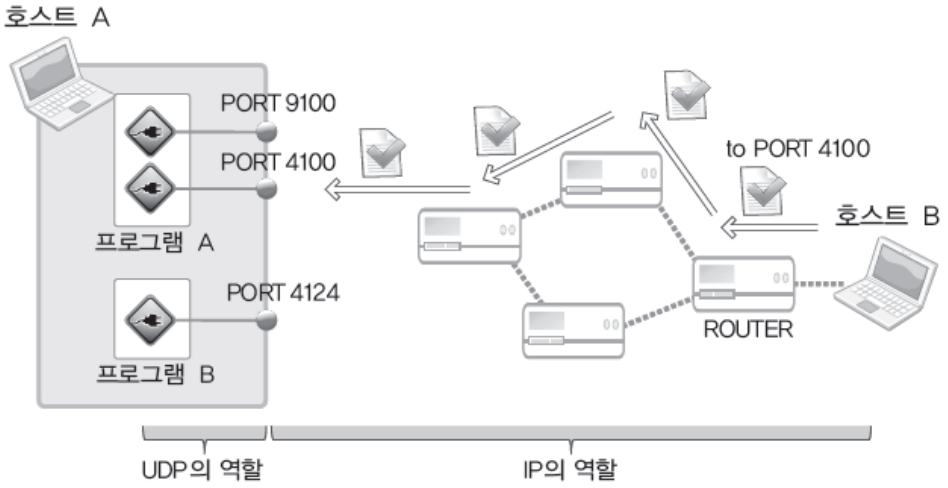


Chapter 06-1. UDP에 대한 이해

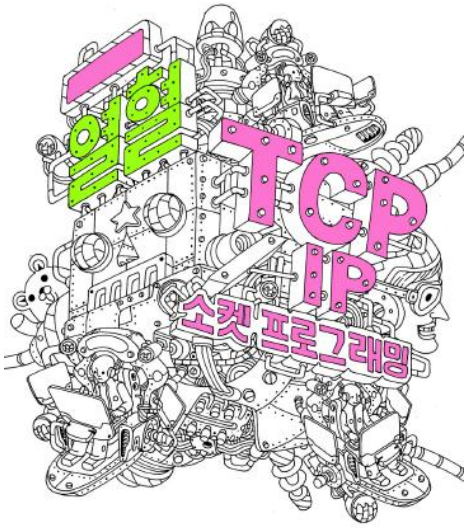
윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

UDP 소켓의 특성과 동작원리

- UDP 소켓과 TCP 소켓의 데이터 송수신 비교
  - UDP 소켓은 SEQ,ACK와 같은 메시지 전달을 하지 않는다(Flow Control 없음).
  - 그리고 연결의 설정과 해제의 과정도 존재하지 않는다.
  - 때문에 데이터의 분실 및 손실의 위험이 있다.
  - 그러나 확인의 과정이 존재하지 않기 때문에 데이터의 전송이 빠르다.
  - 따라서 안전성보다 성능이 중요시 될 때에는 UDP를 사용한다.



▶ 그림 06-1 : 패킷 전송에 있어서의 UDP와 IP의 역할

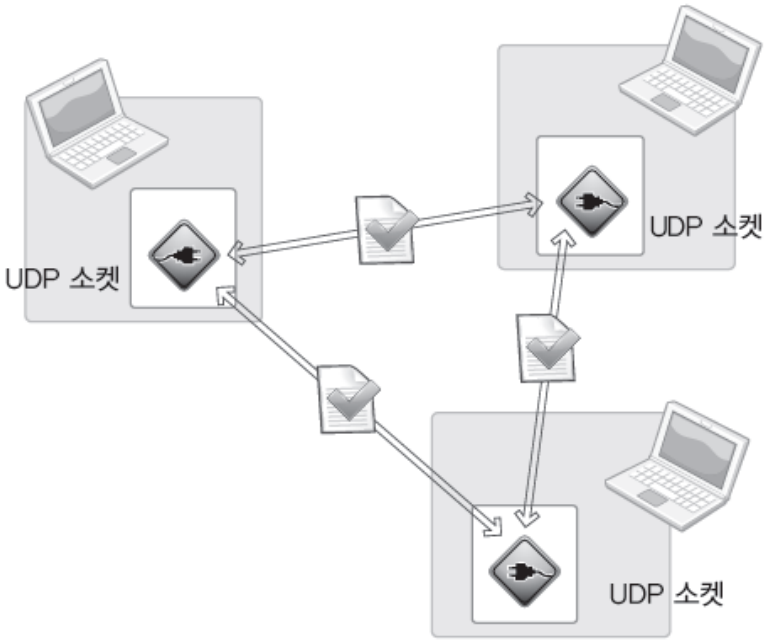


Chapter 06-2. UDP 기반 서버/클라이언트의 구현

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

UDP 소켓은 연결이라는 개념이 존재하지 않는다.

- UDP의 데이터 송수신
  - TCP는 1대 1의 연결을 필요로 하지만, UDP는 연결의 개념이 존재하지 않는다.
  - 따라서 서버 소켓과 클라이언트 소켓의 구분이 없다.
  - 연결의 개념이 존재하지 않으므로, 하나의 소켓으로 둘 이상의 영역과 데이터 송수신이 가능하다.



UDP 기반의 데이터 입출력 함수


```
#include <sys/socket.h>

ssize_t sendto(int sock, void *buff, size_t nbytes, int flags,
                                    struct sockaddr *to, socklen_t addrlen);
```

➡ 성공 시 전송된 바이트 수, 실패 시 -1 반환

- sock — 데이터 전송에 사용될 UDP 소켓의 파일 디스크립터를 인자로 전달.
- buff — 전송할 데이터를 저장하고 있는 버퍼의 주소 값 전달.
- nbytes — 전송할 데이터 크기를 바이트 단위로 전달.
- flags — 옵션 지정에 사용되는 매개변수, 지정할 옵션이 없다면 0 전달.
- to — 목적지 주소정보를 담고 있는 sockaddr 구조체 변수의 주소 값 전달.
- addrlen — 매개변수 to로 전달된 주소 값의 구조체 변수 크기 전달.

UDP 소켓은 연결의 개념이 있지 않으므로, 데이터를 전송할 때마다 목적지에 대한 정보를 전달해야 한다.

UDP 소켓은 연결의 개념이 있지 않으므로, 데이터의 전송지가 둘 이상이 될 수 있다. 따라서 데이터 수신 후 전송지가 어디인지 확인할 필요가 있다.

```
#include <sys/socket.h>

ssize_t recvfrom(int sock, void *buff, size_t nbytes, int flags,
                                    struct sockaddr *from, socklen_t *addrlen);
```

➡ 성공 시 수신한 바이트 수, 실패 시 -1 반환

- sock — 데이터 수신에 사용될 UDP 소켓의 파일 디스크립터를 인자로 전달.
- buff — 데이터 수신에 사용될 버퍼의 주소 값 전달.
- nbytes — 수신할 최대 바이트 수 전달, 때문에 매개변수 buff가 가리키는 버퍼의 크기를 넘을 수 없다.
- flags — 옵션 지정에 사용되는 매개변수, 지정할 옵션이 없다면 0 전달.
- from — 발신지 정보를 채워 넣을 sockaddr 구조체 변수의 주소 값 전달.
- addrlen — 매개변수 from으로 전달된 주소 값의 구조체 변수의 크기 전달.

UDP 기반의 에코 서버와 클라이언트


```
if(bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1)
    error_handling("bind() error");
while(1)
{
    clnt_adr_sz=sizeof(clnt_adr);
    str_len=recvfrom(serv_sock, message, BUF_SIZE, 0,
            (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
    sendto(serv_sock, message, str_len, 0,
            (struct sockaddr*)&clnt_adr, clnt_adr_sz);
}
```

UDP 에코 서버의 일부

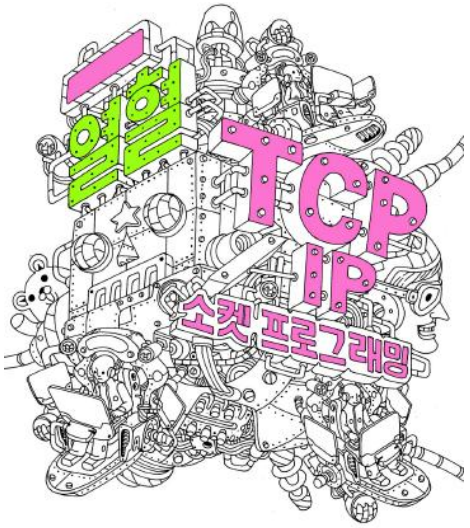
UDP 에코 서버 코드에서는 수신한 데이터의 전송지 정보를 참조하여 데이터를 에코 함에 주의하자!

UDP 에코 클라이언트의 일부

UDP는 데이터의 경계가 존재하기 때문에 한번의 recvfrom 함수호출을 통해서 하나의 메시지를 완전히 읽어 들인다. 그리고 **sendto 함수호출 시 IP와 PORT 번호가 자동으로 할당**되기 때문에 일반적으로 UDP의 클라이언트 프로그램에서는 주소정보를 할당하는 별도의 과정이 불필요하다.

```
while(1)
{
    fputs("Insert message(q to quit): ", stdout);
    fgets(message, sizeof(message), stdin);
    if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
        break;

    sendto(sock, message, strlen(message), 0,
        (struct sockaddr*)&serv_adr, sizeof(serv_adr));
    adr_sz=sizeof(from_adr);
    str_len=recvfrom(sock, message, BUF_SIZE, 0,
            (struct sockaddr*)&from_adr, &adr_sz);
    message[str_len]=0;
    printf("Message from server: %s", message);
}
```



Chapter 06-3. UDP의 데이터 송수신 특성과 UDP에서의 connect 함수호출

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

데이터의 경계가 존재하는 UDP 소켓

```
if(bind(sock, (struct sockaddr*)&my_adr, sizeof(my_adr))==-1)
    error_handling("bind() error");

for(i=0; i<3; i++)
{
    sleep(5);       // delay 5 sec.
    adr_sz=sizeof(your_adr);
    str_len=recvfrom(sock, message, BUF_SIZE, 0,
            (struct sockaddr*)&your_adr, &adr_sz);

    printf("Message %d: %s \n", i+1, message);
}
```

bound_host1.c

의 데이터 수신부분

데이터의 경계가 존재하지 않기 때문에 5초간의 delay를 삽입해도 총 3개의 메시지를 3번의 recvfrom 함수호출을 통해서 수신한다.

데이터의 전송에 있어서 TCP와의 유일한 차이점은 사용하는 함수가 다르고 전달할 목적지 정보를 매 호출 시마다 지정한다는 점이다.

bound_host2.c

의 데이터 전송부분

```
sendto(sock, msg1, sizeof(msg1), 0,
    (struct sockaddr*)&your_adr, sizeof(your_adr));
sendto(sock, msg2, sizeof(msg2), 0,
    (struct sockaddr*)&your_adr, sizeof(your_adr));
sendto(sock, msg3, sizeof(msg3), 0,
    (struct sockaddr*)&your_adr, sizeof(your_adr));
```

connected UDP 소켓

unconnected UDP 소켓의 sendto 함수 호출과정

- 1단계 UDP 소켓에 목적지의 IP와 PORT번호 등록
- 2단계 데이터 전송
- 3단계 UDP 소켓에 등록된 목적지 정보 삭제

connected UDP 소켓의 경우 1단계와 3단계의 과정을 매회 거치지 않는다.

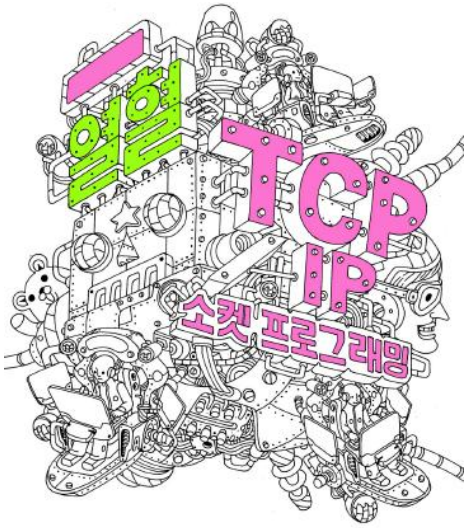
connected UDP 소켓의 생성과정

connected UDP 소켓은 TCP와 같이 상대 소켓과의 연결을 의미하지는 않는다. 그러나 소켓에 목적지에 대한 정보는 등록이 된다. 그리고 connected UDP 소켓을 대상으로는 read, write 함수의 호출이 가능하다.

```
sock=socket(PF_INET, SOCK_DGRAM, 0);
if(sock==-1)
    error_handling("socket() error");

memset(&serv_adr, 0, sizeof(serv_adr));
serv_adr.sin_family=AF_INET;
serv_adr.sin_addr.s_addr=inet_addr(argv[1]);
serv_adr.sin_port=htons(atoi(argv[2]));

connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr));
```



Chapter 06-4. 윈도우 기반으로 구현하기

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

윈도우 기반 sendto, recvfrom 함수

```
#include <winsock2.h>

int sendto(SOCKET s, const char* buf, int len, int flags, const struct sockaddr* to, int tolen);
```

➜ 성공 시 전송된 바이트 수, 실패 시 SOCKET_ERROR 반환

리눅스의 sendto 함수와 사실상 차이가 없다.

```
#include <winsock2.h>

int recvfrom(SOCKET s, char* buf, int len, int flags, struct sockaddr* from, int* fromlen);
```

➜ 성공 시 수신한 바이트 수, 실패 시 SOCKET_ERROR 반환

리눅스의 recvfrom 함수와 사실상 차이가 없다.

윈도우 기반 connected UDP 소켓

```
sock=socket(PF_INET, SOCK_DGRAM, 0);
if(sock==INVALID_SOCKET)
    ErrorHandling("socket() error");

memset(&servAdr, 0, sizeof(servAdr));
servAdr.sin_family=AF_INET;
servAdr.sin_addr.s_addr=inet_addr(argv[1]);
servAdr.sin_port=htons(atoi(argv[2]));
connect(sock, (SOCKADDR*)&servAdr, sizeof(servAdr));
```

connected UDP 소켓의 생성과정

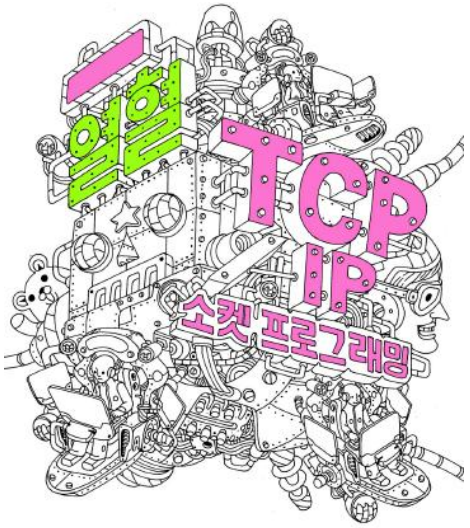
connected UDP 소켓의 생성방법은 리눅스와 동일하다.

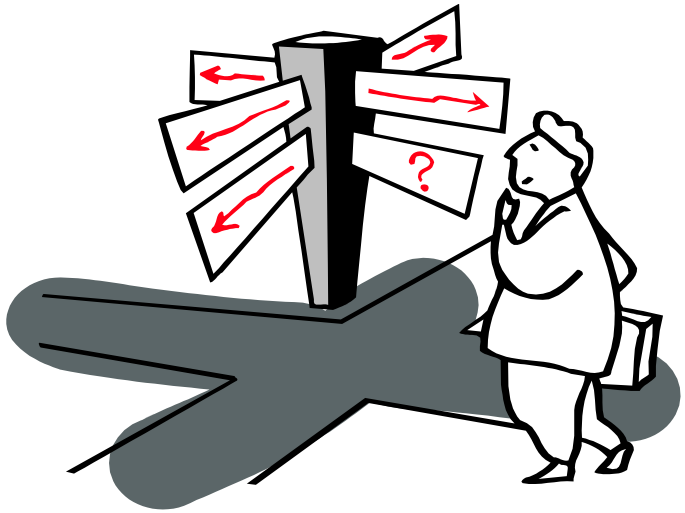
connected UDP 소켓 대상의 데이터 송수신

TCP 기반에서 사용한 입출력 함수인 send와 recv 함수를 호출하여 데이터를 송수신 한다.

```
while(1)
{
    fputs("Insert message(q to quit): ", stdout);
    fgets(message, sizeof(message), stdin);
    if(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
        break;

    send(sock, message, strlen(message), 0);
    strLen=recv(sock, message, sizeof(message)-1, 0);
    message[strLen]=0;
    printf("Message from server: %s", message);
}
```





Chapter 06이 끝났습니다. 질문 있으신지요?