

Chapter 09 소켓 옵션

- TCP/IP 응용 프로그램에 적용 가능한 소켓 옵션을 이해하고 활용할 수 있다.
- 멀티캐스팅의 개념과 동작 원리를 이해하고 UDP를 이용하여 구현할 수 있다.
- 소켓 옵션을 조합해서 적용하여 데이터 통신의 기능과 효율을 보완할 수 있다.

목차

01 소켓 옵션의 종류와 관련 함수

02 SOL_SOCKET 레벨 옵션

03 IPPROTO_IP, IPPROTO_IPV6 레벨 옵션

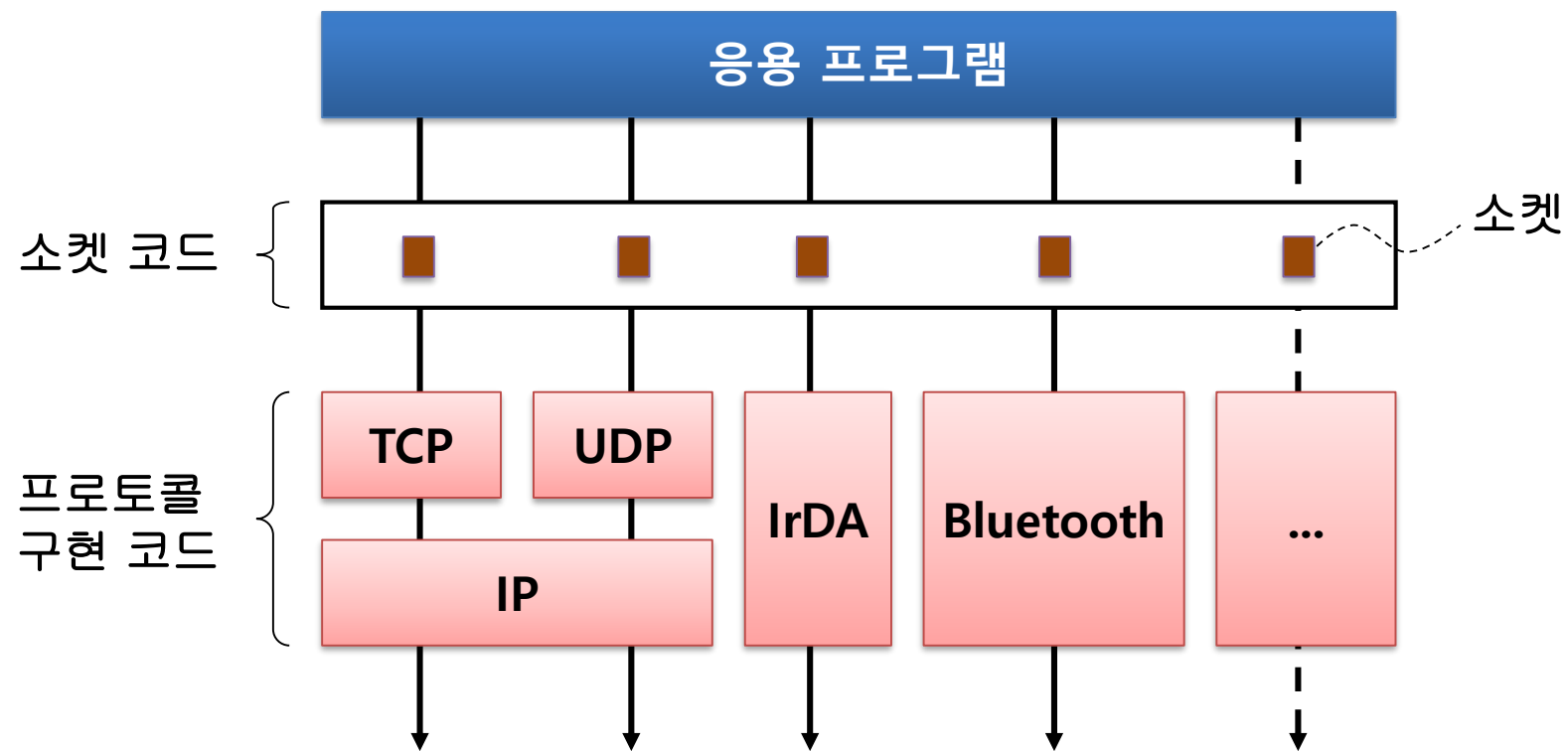
04 IPPROTO_TCP 레벨 옵션

01 소켓 옵션의 종류와 관련 함수



개요 (1)

■ 소켓 프로그래밍 모델



개요 (2)

■ 소켓 옵션

- 소켓 함수의 기본 동작을 변경
 - 소켓 코드와 프로토콜 구현 코드를 세부적으로 제어

■ 처리 주체에 따른 소켓 옵션의 종류

- ① 소켓 코드가 처리하는 옵션
 - 옵션을 설정하면 소켓 코드에서 해석하고 처리
- ② 프로토콜 구현 코드가 처리하는 옵션
 - 옵션을 설정하면 프로토콜 구현 코드에서 해석하고 처리

개요 (3)

■ 소켓 옵션 설정(=변경)하기

윈도우

```
#include <winsock2.h>
int setsockopt(
    ❶ SOCKET sock,
    ❷ int level,
    ❸ int optname,
    ❹ const char *optval,
    ❺ int optlen
);
```

성공: 0, 실패: SOCKET_ERROR

리눅스

```
#include <sys/types.h>
#include <sys/socket.h>
int setsockopt(
    ❶ int sock,
    ❷ int level,
    ❸ int optname,
    ❹ const void *optval,
    ❺ socklen_t optlen
);
```

성공: 0, 실패: -1

개요 (4)

■ 현재 설정된 소켓 옵션 값 얻기

윈도우

```
#include <winsock2.h>
int getsockopt(
    ❶ SOCKET sock,
    ❷ int level,
    ❸ int optname,
    ❹ char *optval,
    ❺ int optlen
);
```

성공: 0, 실패: SOCKET_ERROR

리눅스

```
#include <sys/types.h>
#include <sys/socket.h>
int getsockopt(
    ❶ int sock,
    ❷ int level,
    ❸ int optname,
    ❹ void *optval,
    ❺ socklen_t *optlen
);
```

성공: 0, 실패: -1

소켓 옵션의 종류 (1)

■ level = SOL_SOCKET

표 9-1 level = SOL_SOCKET

optname 값	optval 타입	get	set	설명
SO_BROADCAST	DWORD (boolean)	o	o	브로드캐스트 패킷 전송 가능 여부
SO_KEEPALIVE	DWORD (boolean)	o	o	주기적으로 연결 상태 확인 여부
SO_LINGER	linger{ }	o	o	소켓 송신 버퍼에 미전송 데이터가 있을 때 closesocket() ^[윈도우] /close() ^[리눅스] 함수의 리턴 지연 시간 설정
SO_SNDBUF SO_RCVBUF	DWORD	o	o	소켓 송/수신 버퍼의 크기 설정
SO_SNDTIMEO SO_RCVTIMEO	DWORD ^[윈도우] timeval{ } ^[리눅스]	o	o	데이터 송/수신 함수 호출 시 타임아웃 설정
SO_REUSEADDR	DWORD (boolean)	o	o	지역 주소(IP 주소, 포트 번호) 재사용 여부

소켓 옵션의 종류 (2)

■ level = IPPROTO_IP

표 9-2 level = IPPROTO_IP

optname 값	optval 타입	get	set	설명
IP_TTL	DWORD	o	o	IP 패킷의 TTL ^{Time-To-Live} 값 설정
IP_ADD_MEMBERSHIP IP_DROP_MEMBERSHIP	ip_mreq{}	x	o	멀티캐스트 그룹 가입과 탈퇴
IP_MULTICAST_IF	DWORD ^[윈도우] in_addr{ } ^[리눅스]	o	o	멀티캐스트 패킷 송신 인터페이스 선택 (인터페이스 IP 주소 전달, 기본값은 0)
IP_MULTICAST_LOOP	DWORD (boolean) ^[윈도우] u_char ^[리눅스]	o	o	멀티캐스트 패킷의 루프백 여부
IP_MULTICAST_TTL	DWORD ^[윈도우] u_char ^[리눅스]	o	o	멀티캐스트 패킷의 TTL 값 설정

소켓 옵션의 종류 (3)

■ level = IPPROTO_IPV6

표 9-3 level = IPPROTO_IPV6

optname 값	optval 타입	get	set	설명
IPV6_UNICAST_HOPS	DWORD	o	o	IP 패킷의 TTL ^{Time-To-Live} 값 설정
IPV6_JOIN_GROUP IPV6_LEAVE_GROUP	ipv6_mreq{}	x	o	멀티캐스트 그룹 가입과 탈퇴
IPV6_MULTICAST_IF	DWORD ^[윈도우] u_int ^[리눅스]	o	o	멀티캐스트 패킷 송신 인터페이스 선택 (인터페이스 인덱스 전달, 기본값은 0)
IPV6_MULTICAST_LOOP	DWORD (boolean) ^[윈도우] u_int ^[리눅스]	o	o	멀티캐스트 패킷의 루프백 여부
IPV6_MULTICAST_HOPS	DWORD	o	o	멀티캐스트 패킷의 TTL 값 설정

소켓 옵션의 종류 (4)

■ level = IPPROTO_TCP

표 9-4 level = IPPROTO_TCP

optname 값	optval 타입	get	set	설명
TCP_NODELAY	DWORD (boolean)	0	0	Nagle 알고리즘 비활성화 여부 (기본값은 0; Nagle 알고리즘 활성 상태)

02 SOL_SOCKET 레벨 옵션



SO_BROADCAST 옵션

■ 용도

- 브로드캐스트 데이터 전송 기능 활성화
- TCP 소켓에는 사용할 수 없고 UDP 소켓에만 사용 가능

■ 사용 예

윈도우

```
// 브로드캐스팅 활성화
DWORD bEnable = 1;
retval = setsockopt(sock, SOL_SOCKET, SO_BROADCAST,
    (const char *)&bEnable, sizeof(bEnable));
if (retval == SOCKET_ERROR) err_quit("setsockopt()");
```

리눅스

```
// 브로드캐스팅 활성화
int bEnable = 1;
retval = setsockopt(sock, SOL_SOCKET, SO_BROADCAST,
    &bEnable, sizeof(bEnable));
if (retval == SOCKET_ERROR) err_quit("setsockopt()");
```

SO_KEEPALIVE 옵션 (1)

■ 용도

- TCP 프로토콜 수준에서 연결 여부를 확인하기 위해 상대 TCP에 주기적으로(기본 2시간 간격) TCP 패킷을 보냄

■ 상대 TCP의 반응에 따른 동작

- 상대 TCP가 정해진 시간 안에 응답하는 경우
 - 응용 프로그램은 정상적으로 동작
- 상대 TCP가 정해진 시간 안에 응답하지 않는 경우
 - 자동으로 소켓을 닫아 시스템 자원 소모를 막음
- 상대 TCP가 RST 패킷으로 응답하는 경우
 - 자동으로 소켓을 닫아 시스템 자원 소모를 막음

SO_KEEPALIVE 옵션 (2)

■ 사용 예

윈도우

```
DWORD bEnable = 1;  
retval = setsockopt(sock, SOL_SOCKET, SO_KEEPALIVE,  
    (const char *)&bEnable, sizeof(bEnable));  
if (retval == SOCKET_ERROR) err_quit("setsockopt());
```

리눅스

```
int bEnable = 1;  
retval = setsockopt(sock, SOL_SOCKET, SO_KEEPALIVE,  
    &bEnable, sizeof(bEnable));  
if (retval == SOCKET_ERROR) err_quit("setsockopt());
```


SO_LINGER 옵션 (1)

■ 용도

- closesocket()/close() 함수의 동작 변경

```
send(sock, ...);    // 데이터를 보낸다.  
closesocket(sock);  // 소켓을 닫는다. [윈도우]  
또는 close(sock);   // 소켓을 닫는다. [리눅스]
```

■ closesocket()/close() 함수의 두 가지 기능

- 소켓을 닫고 할당된 운영체제 자원을 반환
- TCP 프로토콜 수준에서 연결 종료 절차를 시작

SO_LINGER 옵션 (2)

■ 옵션 값

윈도우

```
struct linger {  
    ❶ u_short l_onoff; /* 0=off, nonzero=on */  
    ❷ u_short l_linger; /* linger time (seconds) */  
};
```

리눅스

```
struct linger {  
    ❶ int l_onoff; /* 0=off, nonzero=on */  
    ❷ int l_linger; /* linger time (seconds) */  
};
```

SO_LINGER 옵션 (3)

■ 사용 예

원도우

```
struct linger optval;  
optval.l_onoff = 1;    /* linger on */  
optval.l_linger = 10; /* linger time = 10초 */  
retval = setsockopt(sock, SOL_SOCKET, SO_LINGER,  
    (const char *)&optval, sizeof(optval));  
if (retval == SOCKET_ERROR) err_quit("setsockopt()");
```

리눅스

```
struct linger optval;  
optval.l_onoff = 1;    /* linger on */  
optval.l_linger = 10; /* linger time = 10초 */  
retval = setsockopt(sock, SOL_SOCKET, SO_LINGER,  
    &optval, sizeof(optval));  
if (retval == SOCKET_ERROR) err_quit("setsockopt()");
```

SO_LINGER 옵션 (4)

■ 옵션 값에 따른 closesocket()/close() 함수 동작

표 9-5 SO_LINGER 옵션값에 따른 closesocket()/close() 함수 동작

linger 구조체		closesocket()/close() 함수 동작	추가 설명
l_onoff	l_linger		
0	사용 안 함	설명 ❶과 동일	closesocket()/close()의 기본 동작 방식
1	0	설명 ❷과 동일	
1	양수	설명 ❸과 동일	

SO_SNDBUF, SO_RCVBUF 옵션 (1)

■ 용도

- 소켓의 송신 버퍼와 수신 버퍼 크기 변경

■ 사용 예(윈도우)

윈도우

```
DWORD optval; int optlen;
// 수신 버퍼의 크기를 얻는다.
optlen = sizeof(optval);
retval = getsockopt(listen_sock, SOL_SOCKET, SO_RCVBUF,
    (char *)&optval, &optlen);
if (retval == SOCKET_ERROR) err_quit("getsockopt()");
printf("수신 버퍼 크기(old) = %d바이트\n", optval);
// 수신 버퍼의 크기를 두 배로 늘린다.
optval *= 2;
retval = setsockopt(listen_sock, SOL_SOCKET, SO_RCVBUF,
    (const char *)&optval, sizeof(optval));
if (retval == SOCKET_ERROR) err_quit("setsockopt()");
// 수신 버퍼의 크기를 얻는다.
optlen = sizeof(optval);
retval = getsockopt(listen_sock, SOL_SOCKET, SO_RCVBUF,
    (char *)&optval, &optlen);
if (retval == SOCKET_ERROR) err_quit("getsockopt()");
printf("수신 버퍼 크기(new) = %d바이트\n", optval);
```

SO_SNDBUF, SO_RCVBUF 옵션 (2)

■ 사용 예(리눅스)

리눅스

```
int optval; socklen_t optlen;
// 수신 버퍼의 크기를 얻는다.
optlen = sizeof(optval);
retval = getsockopt(listen_sock, SOL_SOCKET, SO_RCVBUF,
    &optval, &optlen);
if (retval == SOCKET_ERROR) err_quit("getsockopt()");
printf("수신 버퍼 크기(old) = %d바이트\n", optval);
// 수신 버퍼의 크기를 두 배로 늘린다.
optval *= 2;
retval = setsockopt(listen_sock, SOL_SOCKET, SO_RCVBUF,
    &optval, sizeof(optval));
if (retval == SOCKET_ERROR) err_quit("setsockopt()");
// 수신 버퍼의 크기를 얻는다.
optlen = sizeof(optval);
retval = getsockopt(listen_sock, SOL_SOCKET, SO_RCVBUF,
    &optval, &optlen);
if (retval == SOCKET_ERROR) err_quit("getsockopt()");
printf("수신 버퍼 크기(new) = %d바이트\n", optval);
```

SO_SNDBUF, SO_RCVBUF 옵션 (3)

■ 코드를 윈도우 10에서 실행한 결과



그림 9-2 소켓 수신 버퍼 크기 변경

SO_SNDTIMEO, SO_RCVTIMEO 옵션 (1)

■ 용도

- 데이터 전송 함수가 작업 완료와 상관없이 일정 시간 후 리턴하게 함

■ 사용 예

윈도우

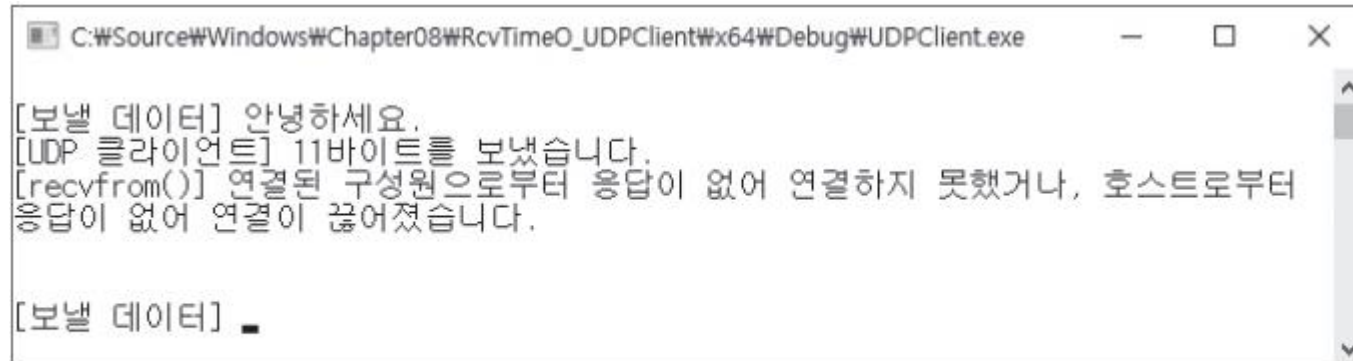
```
// 수신 타임아웃을 3.5초(=3500밀리초)로 지정한다.  
DWORD optval = 3500;  
retval = setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO,  
    (const char *)&optval, sizeof(optval));  
if (retval == SOCKET_ERROR) err_quit("setsockopt()");
```

리눅스

```
// 수신 타임아웃을 3.5초(=3초+500000마이크로초)로 지정한다.  
struct timeval optval = { 3, 500000 };  
retval = setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO,  
    &optval, sizeof(optval));  
if (retval == SOCKET_ERROR) err_quit("setsockopt()");
```


SO_SNDTIMEO, SO_RCVTIMEO 옵션 (2)

■ SO_RCVTIMEO 옵션을 사용해 수신 타임아웃을 지정한 경우



C:\Source\Windows\Chapter08\RcvTimeO_UDPClient\wx64\Debug\UDPClient.exe

[보낼 데이터] 안녕하세요.
[UDP 클라이언트] 11바이트를 보냈습니다.
[recvfrom()] 연결된 구성원으로부터 응답이 없어 연결하지 못했거나, 호스트로부터
응답이 없어 연결이 끊어졌습니다.

[보낼 데이터] _

그림 9-3 SO_RCVTIMEO 옵션을 사용하여 수신 타임아웃을 지정한 경우

SO_REUSEADDR 옵션 (1)

■ 용도

- 현재 사용 중인 IP 주소와 포트 번호를 재사용
 - 현재 사용 중인 IP 주소와 포트 번호를 이용해 bind() 함수를 (성공적으로) 호출할 수 있음

■ 목적

- TCP 서버 종료 후 재실행 시 bind() 함수에서 오류가 발생하는 일을 방지
- 여러 IP 주소를 보유한 호스트에서 같은 기능의 서버를 IP 주소별로 따로 운용할 수 있게 함
- 멀티캐스팅 응용 프로그램이 같은 포트 번호를 사용할 수 있게 함

SO_REUSEADDR 옵션 (2)

■ 실습 9-1 SO_REUSEADDR 옵션 테스트



C:\Source\Windows\Chapter08\TCPServer\x64\Debug\TCPServer.exe

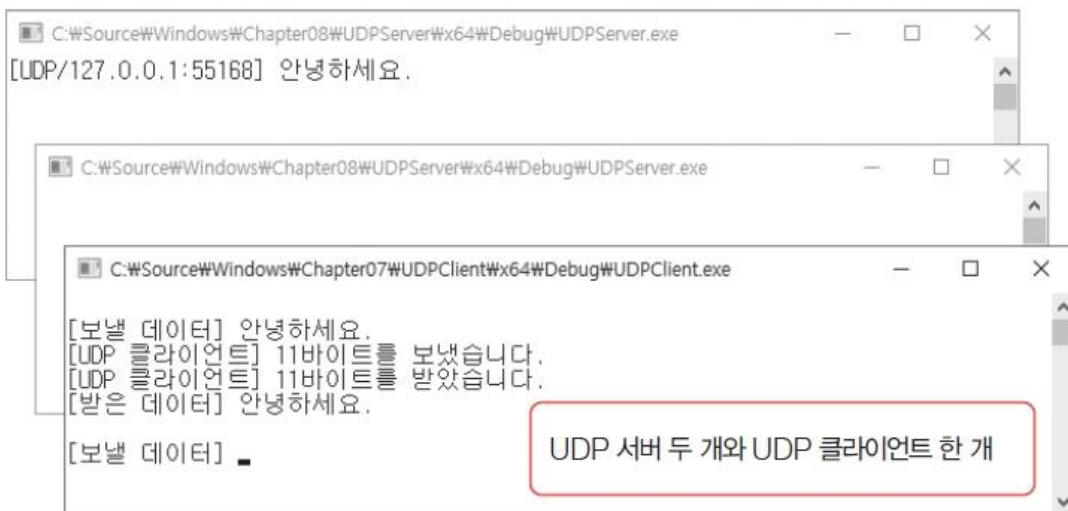
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=4468
[TCP/127.0.0.1:4468] 안녕하세요.

C:\Source\Windows\Chapter08\TCPServer\x64\Debug\TCPServer.exe

C:\Source\Windows\Chapter04\TCPClient\x64\Debug\TCPClient.exe

[보낼 데이터] 안녕하세요.
[TCP 클라이언트] 11바이트를 보냈습니다.
[TCP 클라이언트] 11바이트를 받았습니다.
[받은 데이터] 안녕하세요.
[보낼 데이터] .

원도우 TCP 서버 두 개와 TCP 클라이언트 한 개



C:\Source\Windows\Chapter08\UDPServer\x64\Debug\UDPServer.exe

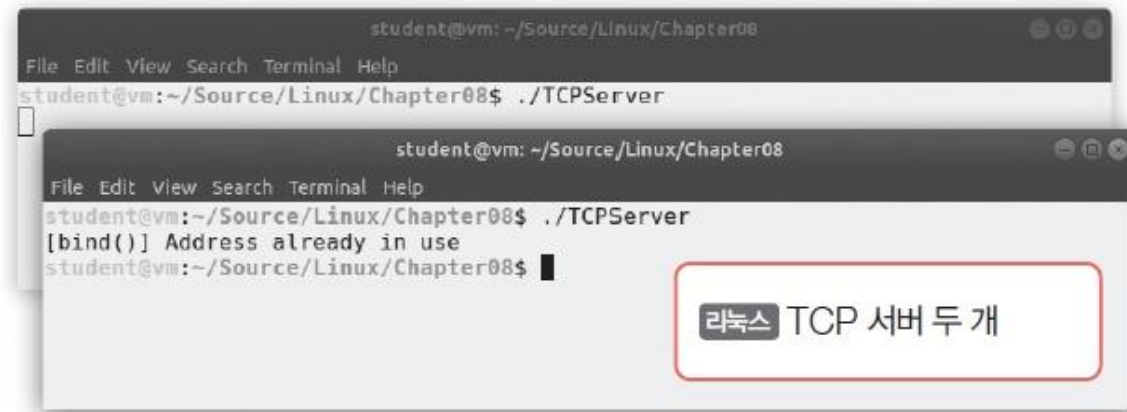
[UDP/127.0.0.1:55168] 안녕하세요.

C:\Source\Windows\Chapter08\UDPServer\x64\Debug\UDPServer.exe

C:\Source\Windows\Chapter07\UDPClient\x64\Debug\UDPClient.exe

[보낼 데이터] 안녕하세요.
[UDP 클라이언트] 11바이트를 보냈습니다.
[UDP 클라이언트] 11바이트를 받았습니다.
[받은 데이터] 안녕하세요.
[보낼 데이터] .

UDP 서버 두 개와 UDP 클라이언트 한 개



```
student@vm: ~/Source/Linux/Chapter08
File Edit View Search Terminal Help
student@vm:~/Source/Linux/Chapter08$ ./TCPServer

student@vm: ~/Source/Linux/Chapter08
File Edit View Search Terminal Help
student@vm:~/Source/Linux/Chapter08$ ./TCPServer
[bind()] Address already in use
student@vm:~/Source/Linux/Chapter08$
```

리눅스 TCP 서버 두 개

SO_REUSEADDR 옵션 (3)

■ 실습 9-1 SO_REUSEADDR 옵션 테스트

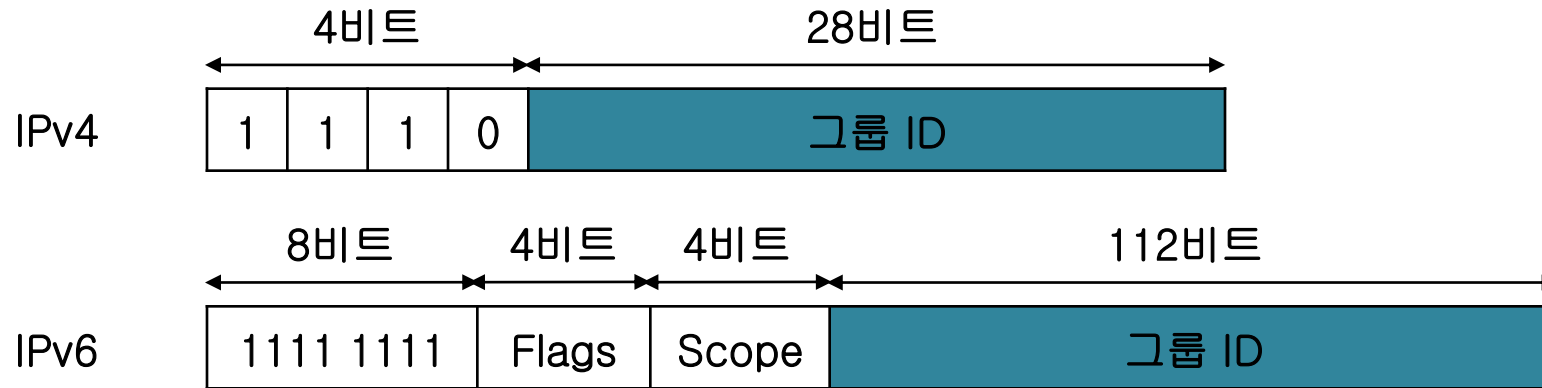
- TCPServer.cpp
 - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter09/TCPServer/TCPServer.cpp>
 - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter09/TCPServer.cpp>
- UDPServer.cpp
 - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter09/UDPServer/UDPServer.cpp>
 - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter09/UDPServer.cpp>

03 IPPROTO_IP, IPPROTO_IPV6 레벨 옵션



멀티캐스팅 (1)

■ 멀티캐스트 주소



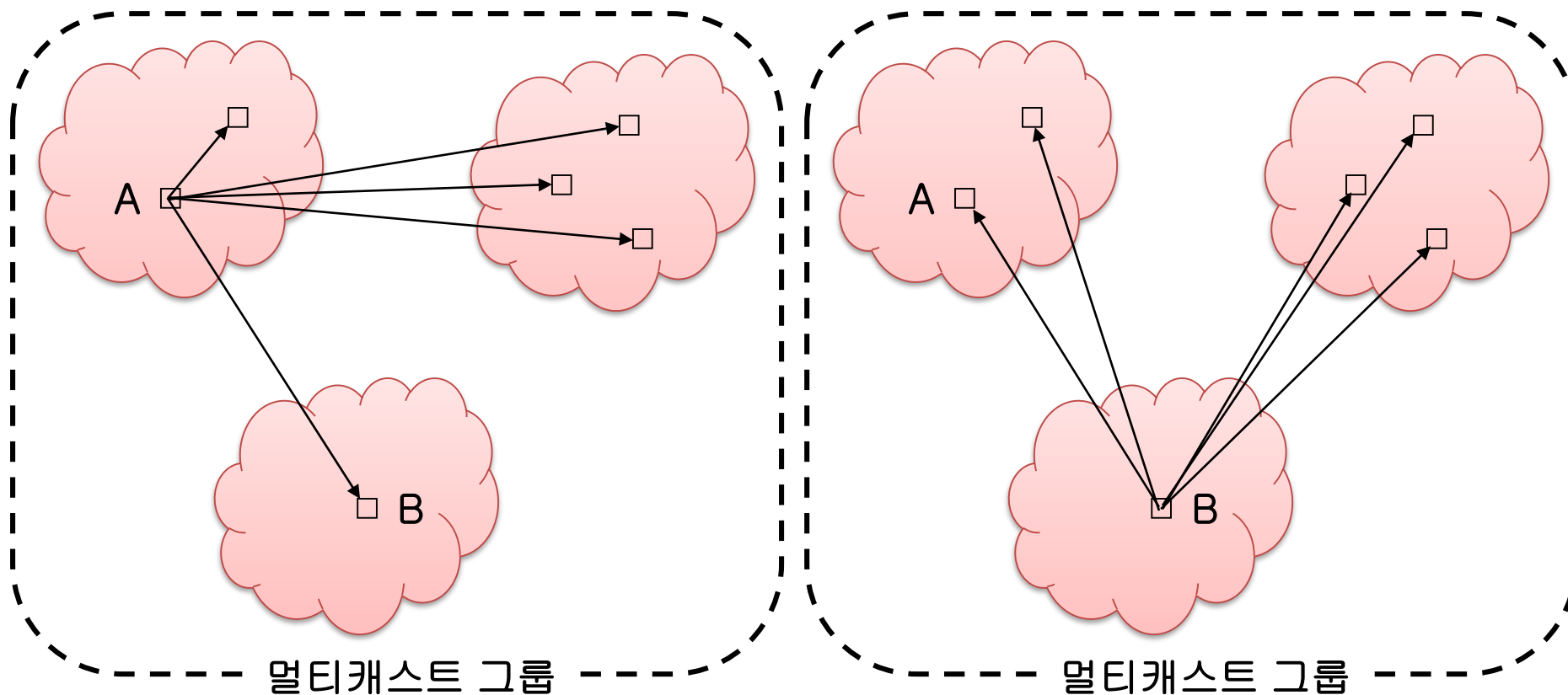
■ 특징

- 그룹 가입과 탈퇴가 자유롭고 그룹 구성원 모두가 평등
- 멀티캐스트 데이터를 받으려면 그룹에 가입해야 함
- 멀티캐스트 데이터를 보내려고 그룹에 가입할 필요 없음

멀티캐스팅 (2)

■ 멀티캐스트 데이터 전송

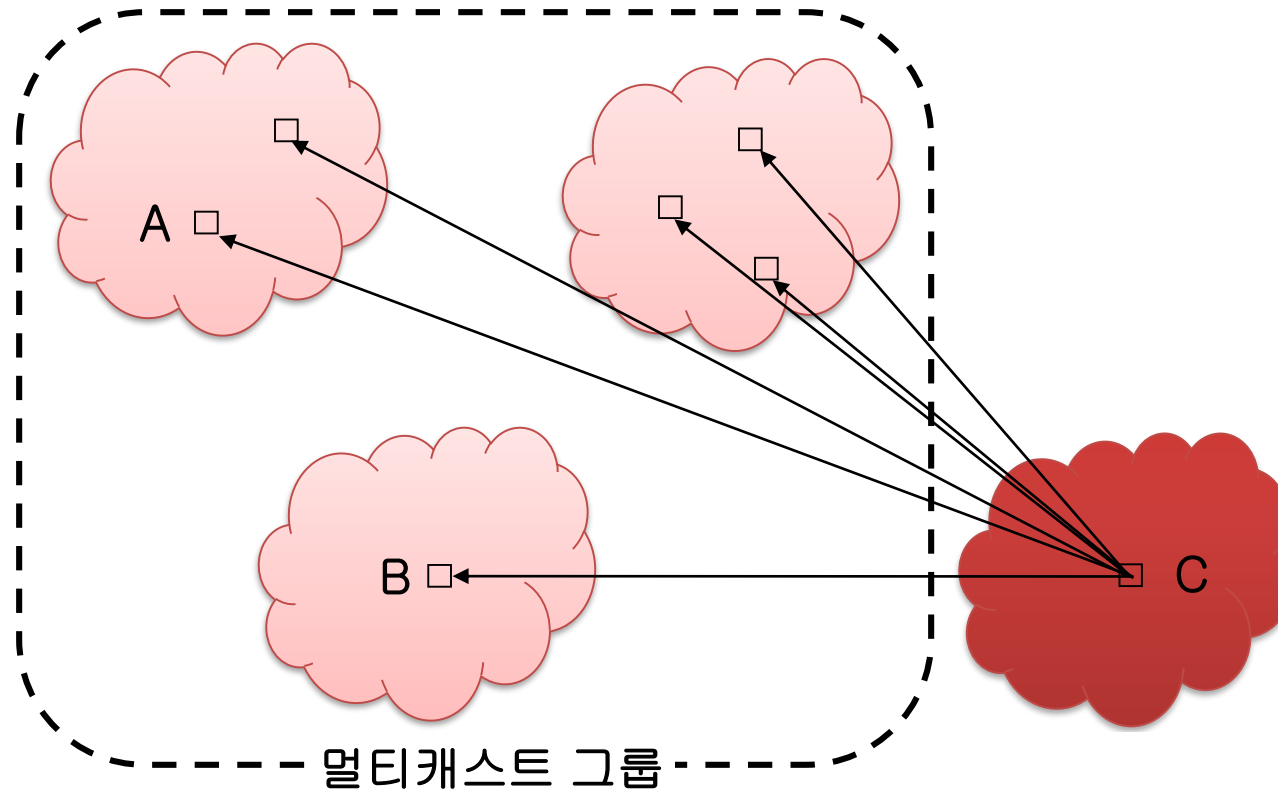
- 그룹에 속한 구성원이 데이터를 보내는 경우



멀티캐스팅 (3)

■ 멀티캐스트 데이터 전송

- 그룹에 속하지 않은 구성원이 데이터를 보내는 경우



멀티캐스팅 (4)

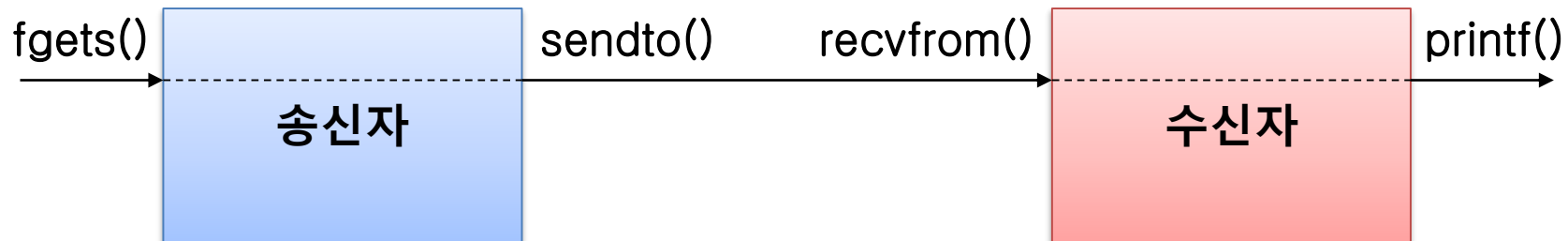
■ 멀티캐스팅 예제 동작

■ 송신자

- 사용자가 키보드로 입력한(fgets) 문자열을 멀티캐스트 주소로 보냄(sendto)

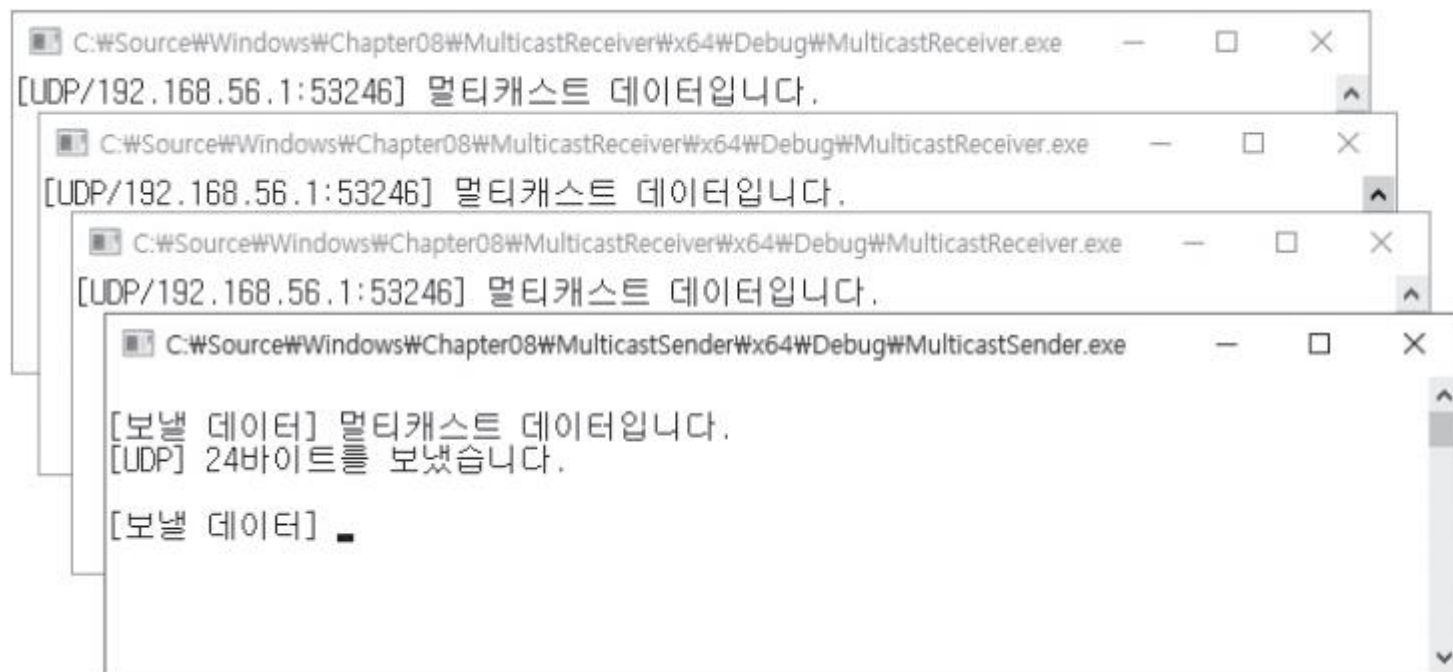
■ 수신자

- 멀티캐스트 그룹에 가입
- 멀티캐스트 데이터를 받고(recvfrom), 이를 문자열로 간주하여 무조건 화면에 출력(sprintf)



멀티캐스팅 (5)

■ 실습 9-2 멀티캐스팅(IPv4) 예제 작성과 테스트



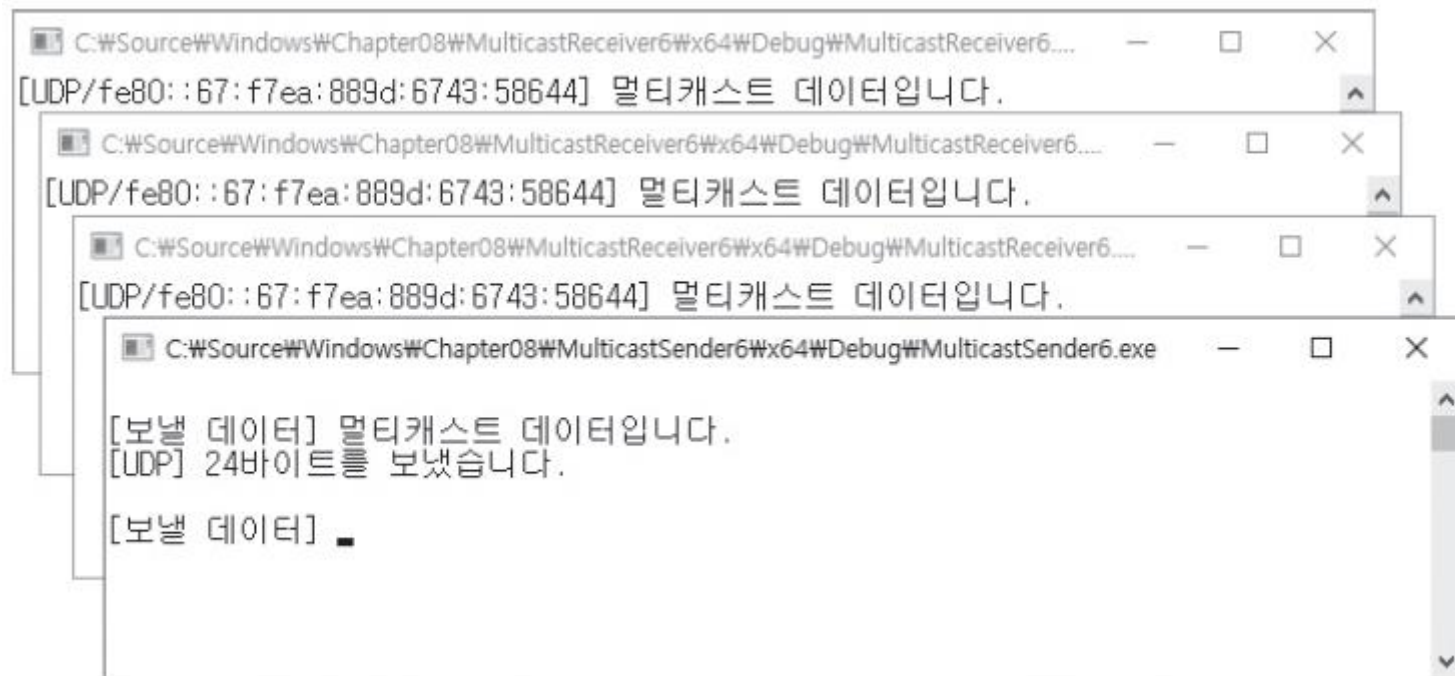
멀티캐스팅 (6)

■ 실습 9-2 멀티캐스팅(IPv4) 예제 작성과 테스트

- MulticastReceiver.cpp
 - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter09/MulticastReceiver/MulticastReceiver.cpp>
 - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter09/MulticastReceiver.cpp>
- MulticastSender.cpp
 - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter09/MulticastSender/MulticastSender.cpp>
 - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter09/MulticastSender.cpp>

멀티캐스팅 (7)

■ 실습 9-3 멀티캐스팅(IPv6) 예제 작성과 테스트



멀티캐스팅 (8)

■ 실습 9-3 멀티캐스팅(IPv6) 예제 작성과 테스트

- MulticastReceiver6.cpp
 - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter09/MulticastReceiver6/MulticastReceiver6.cpp>
 - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter09/MulticastReceiver6.cpp>
- MulticastSender6.cpp
 - [윈도우] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter09/MulticastSender6/MulticastSender6.cpp>
 - [리눅스] <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Linux/Chapter09/MulticastSender6.cpp>

IP_{or}IPV6_MULTICAST_IF 옵션 (1)

■ 용도

- IP 주소를 두 개 이상 보유한 호스트에서 멀티캐스트 데이터를 보낼 네트워크 인터페이스를 선택

■ 사용 예

윈도우

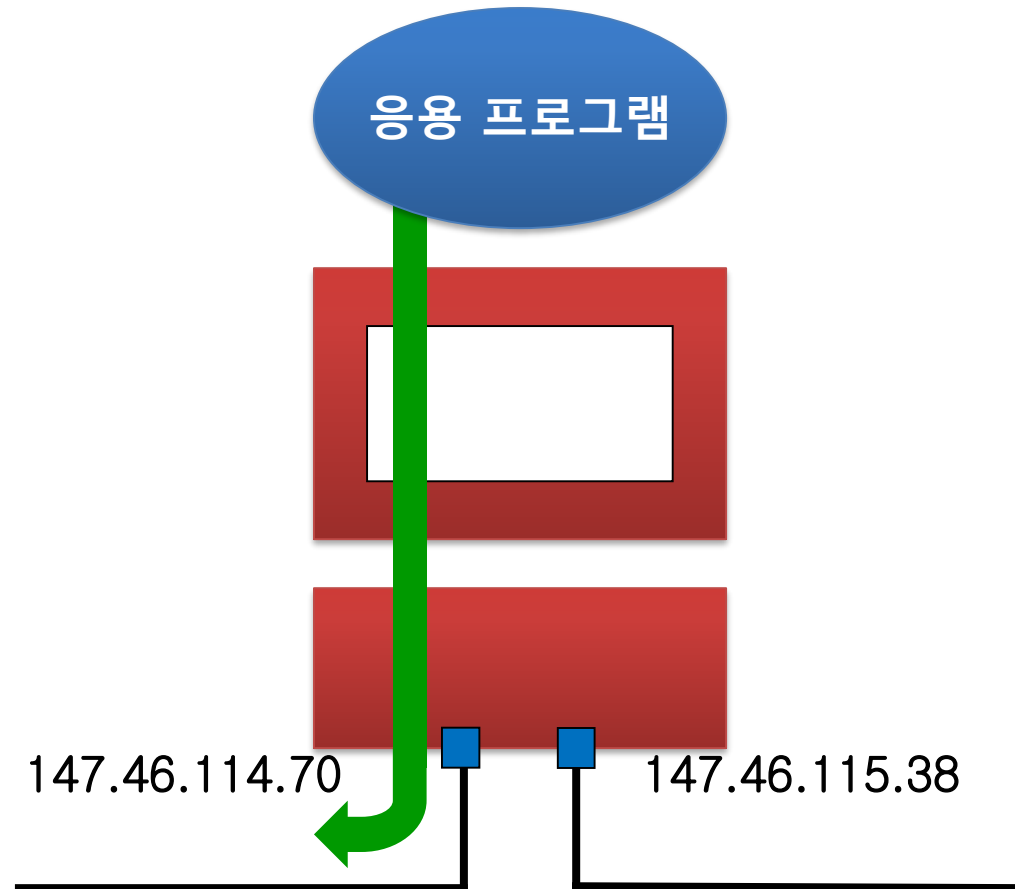
```
DWORD localaddr;  
inet_pton(AF_INET, "147.46.114.70", &localaddr);  
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_IF,  
            (const char *)&localaddr, sizeof(localaddr));
```

리눅스

```
struct in_addr localaddr;  
inet_pton(AF_INET, "147.46.114.70", &localaddr);  
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_IF, &localaddr, sizeof(localaddr));
```

IP_{or}IPV6_MULTICAST_IF 옵션 (2)

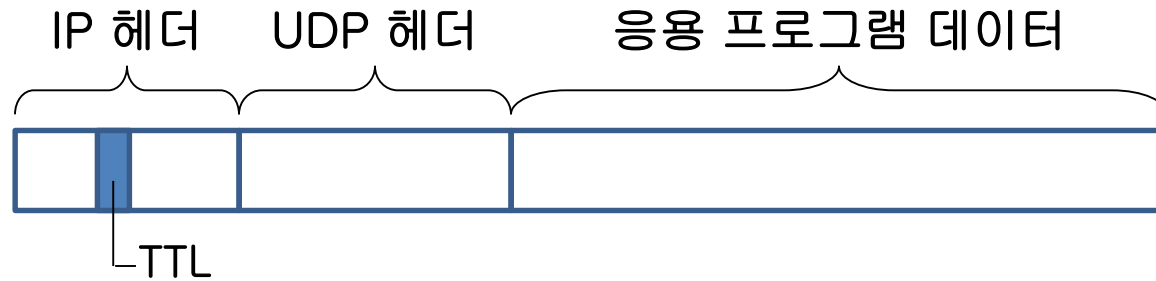
■ IP_MULTICAST_IF 옵션 설정 결과



IP_{or}IPV6_MULTICAST_TTL_{or}HOPS 옵션 (1)

■ 용도

- IP 헤더의 TTL(또는 Hop Limit) 값을 변경



- 멀티캐스트 패킷이 생성될 때 IP 헤더의 TTL 필드는 기본값 1로 설정됨
- TTL은 라우터를 통과할 때마다 1씩 감소하는데, 0이 되면 패킷이 버려짐

IP_{or}IPV6_MULTICAST_TTL_{or}HOPS 옵션 (2)

■ 사용 예#1

```
21  DWORD ttl = 2;
22  retval = setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL,
23      (const char *)&ttl, sizeof(ttl));
24  if (retval == SOCKET_ERROR) err_quit("setsockopt());
```

■ 사용 예#2

```
21  DWORD ttl = 2;
22  retval = setsockopt(sock, IPPROTO_IPV6, IPV6_MULTICAST_HOPS,
23      (const char *)&ttl, sizeof(ttl));
24  if (retval == SOCKET_ERROR) err_quit("setsockopt());
```

IP_{or}IPV6_MULTICAST_LOOP 옵션

■ 용도

- 멀티캐스트 그룹에 가입한 응용 프로그램이 자신의 그룹에 멀티캐스트 데이터를 보낼 때 자신도 받을지를 결정

■ 사용 예

윈도우

```
DWORD optval = 0;    // 자신이 보낸 멀티캐스트 데이터는 받지 않는다.  
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_LOOP,  
            (const char *)&optval, sizeof(optval));
```

리눅스

```
u_char optval = 0;    // 자신이 보낸 멀티캐스트 데이터는 받지 않는다.  
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_LOOP, &optval, sizeof(optval));
```

IP_{or}IPV6_ADD_MEMBERSHIP, IP_{or}IPV6_DROP_MEMBERSHIP 옵션 (1)

■ 용도

- 멀티캐스트 그룹에 가입 또는 탈퇴

■ 옵션값

```
struct ip_mreq {  
    ❶ struct in_addr  imr_multiaddr;    // IPv4 멀티캐스트 주소  
    ❷ struct in_addr  imr_interface;    // 로컬 인터페이스의 IPv4 주소(기본값은 0)  
};  
  
struct ipv6_mreq {  
    ❸ struct in6_addr ipv6mr_multiaddr; // IPv6 멀티캐스트 주소  
    ❹ unsigned int    ipv6mr_interface; // 로컬 인터페이스의 인덱스(기본값은 0)  
};
```

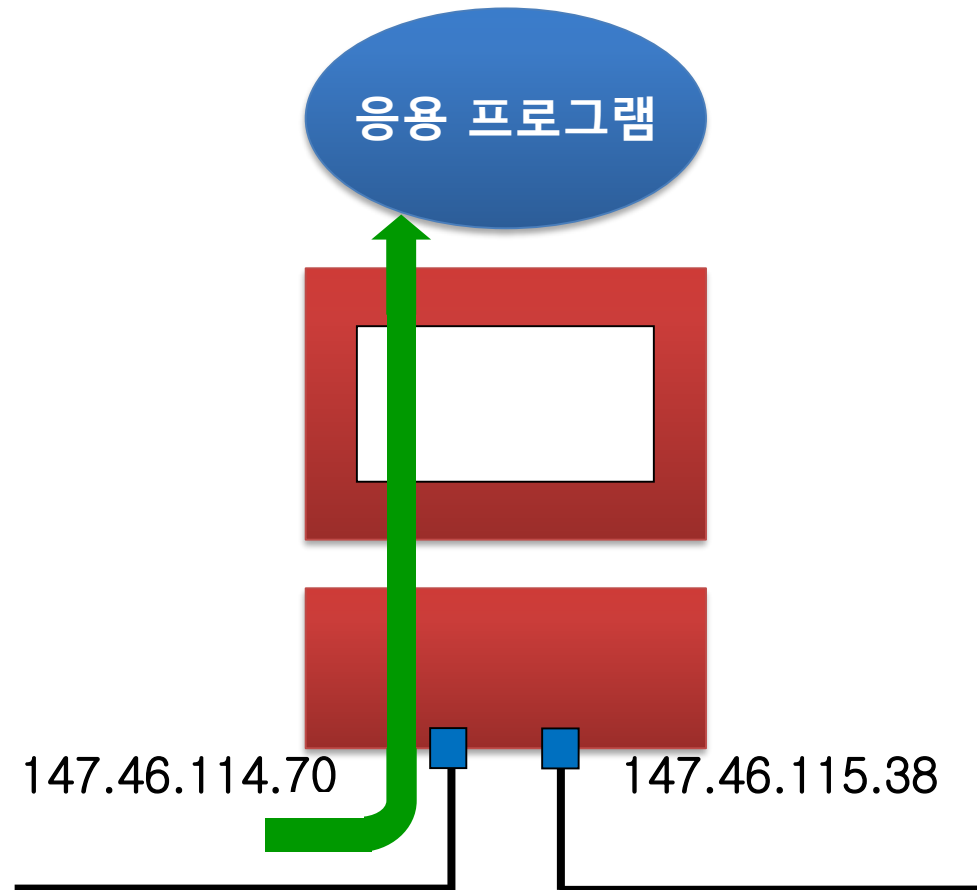
IP_{or}IPV6_ADD_MEMBERSHIP, IP_{or}IPV6_DROP_MEMBERSHIP 옵션 (2)

■ 사용 예

```
struct ip_mreq mreq;  
inet_pton(AF_INET, "235.7.8.9", &mreq.imr_multiaddr);  
inet_pton(AF_INET, "147.46.114.70", &mreq.imr_interface);  
setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,  
           (const char *)&mreq, sizeof(mreq));
```

IP_{or}IPV6_ADD_MEMBERSHIP, IP_{or}IPV6_DROP_MEMBERSHIP 옵션 (3)

■ IP_ADD_MEMBERSHIP 옵션 설정 결과



IP_{or}IPV6_ADD_MEMBERSHIP, IP_{or}IPV6_DROP_MEMBERSHIP 옵션 (4)

■ MulticastReceiver에서 멀티캐스트 그룹에 가입하고 탈퇴하는 코드

```
3 #define MULTICASTIP "235.7.8.9"
...
35 // 멀티캐스트 그룹 가입
36 struct ip_mreq mreq;
37 inet_pton(AF_INET, MULTICASTIP, &mreq.imr_multiaddr);
38 mreq.imr_interface.s_addr = htonl(INADDR_ANY);
39 retval = setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
40     (const char *)&mreq, sizeof(mreq));
41 if (retval == SOCKET_ERROR) err_quit("setsockopt()");
...
66 // 멀티캐스트 그룹 탈퇴
67 retval = setsockopt(sock, IPPROTO_IP, IP_DROP_MEMBERSHIP,
68     (const char *)&mreq, sizeof(mreq));
69 if (retval == SOCKET_ERROR) err_quit("setsockopt()");
```

IP_{or}IPV6_ADD_MEMBERSHIP, IP_{or}IPV6_DROP_MEMBERSHIP 옵션 (5)

■ MulticastReceiver6에서 멀티캐스트 그룹에 가입하고 탈퇴하는 코드

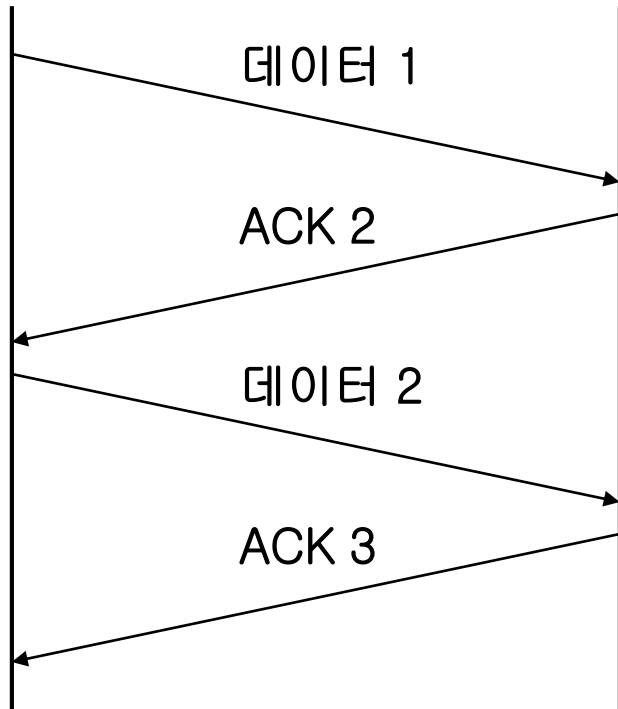
```
3 #define MULTICASTIP "FF12::1:2:3:4"
...
35 // 멀티캐스트 그룹 가입
36 struct ipv6_mreq mreq;
37 inet_pton(AF_INET6, MULTICASTIP, &mreq.ipv6mr_multiaddr);
38 mreq.ipv6mr_interface = 0;
39 retval = setsockopt(sock, IPPROTO_IPV6, IPV6_JOIN_GROUP,
40     (const char *)&mreq, sizeof(mreq));
41 if (retval == SOCKET_ERROR) err_quit("setsockopt()");
...
66 // 멀티캐스트 그룹 탈퇴
67 retval = setsockopt(sock, IPPROTO_IPV6, IPV6_LEAVE_GROUP,
68     (const char *)&mreq, sizeof(mreq));
69 if (retval == SOCKET_ERROR) err_quit("setsockopt()");
```


04 IPPROTO_TCP 레벨 옵션

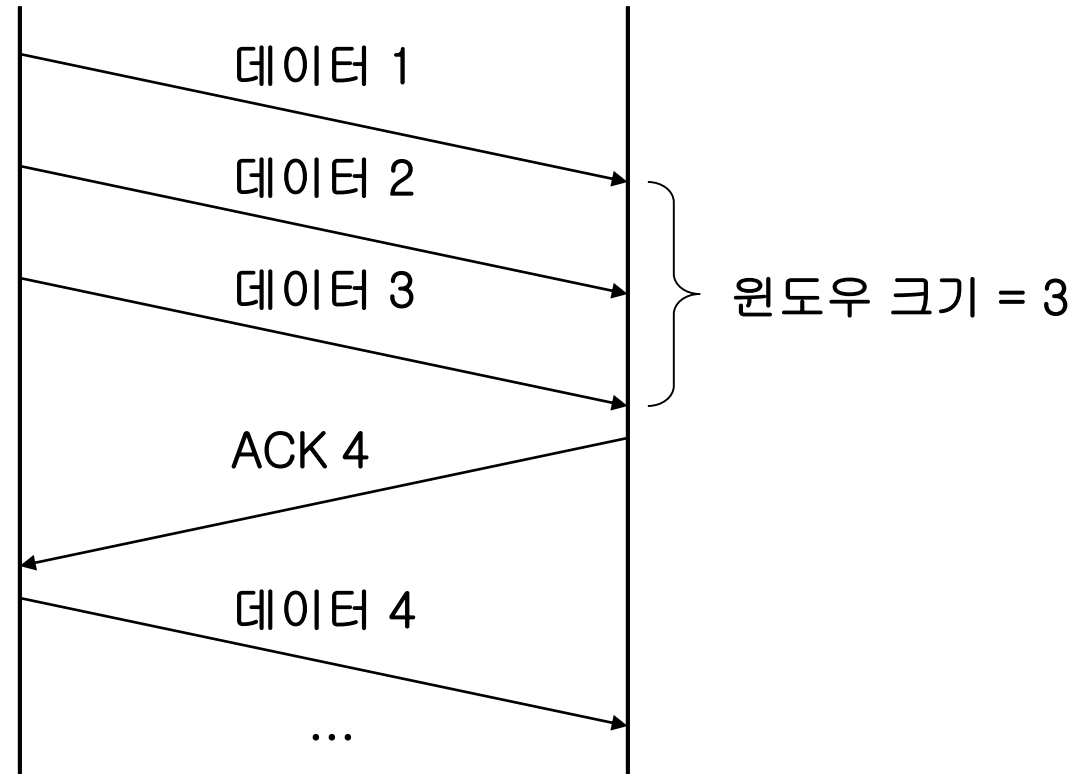


TCP_NODELAY 옵션 (1)

■ TCP 데이터 전송 원리



ACK를 이용하여 데이터 수신 확인



슬라이딩 윈도우를 이용하여 전송 효율 높임

TCP_NODELAY 옵션 (2)

■ 용도

- Nagle(네이글) 알고리즘 작동 여부 결정

■ Nagle 알고리즘의 동작 방식

- 보낼 데이터가 MSS로 정의된 크기만큼 쌓이면 상대방에 무조건 보냄
- 보낼 데이터가 MSS보다 작으면 이전에 보낸 데이터에 대한 ACK가 오기를 기다림
ACK가 도착하면 보낼 데이터가 MSS보다 작더라도 상대방에 보냄

TCP_NODELAY 옵션 (3)

■ Nagle 알고리즘의 장단점

- 장점 : 작은 패킷이 불필요하게 많이 생성되는 일을 방지해 네트워크 트래픽을 감소시킴
- 단점 : 데이터가 충분히 쌓일 때까지 또는 ACK가 도달할 때까지 대기하는 시간 때문에 응용 프로그램의 반응 시간이 길어질 수 있음

■ 사용 예

윈도우

```
DWORD optval = 1; // Nagle 알고리즘 중지
setsockopt(sock, IPPROTO_TCP, TCP_NODELAY,
    (const char *)&optval, sizeof(optval));
```

리눅스

```
#include <netinet/tcp.h> /* TCP_NODELAY 상수 정의 */
...
int optval = 1; // Nagle 알고리즘 중지
setsockopt(sock, IPPROTO_TCP, TCP_NODELAY,
    &optval, sizeof(optval));
```