

Global KPU!

글로벌 경쟁력을 갖춘 산업기술 명문대학
세계를 향해 더 큰 미래를 펼쳐갑니다

8장. 소켓 옵션

네트워크 게임 프로그래밍

❖ TCP/IP 응용 프로그램에 적용 가능한 다양한 소켓 옵션을 이해하고 활용한다.

- 소켓 옵션의 종류와 관련 함수
- SOL_SOCKET 레벨 옵션
- IPPROTO_IP, IPPROTO_IPV6 레벨 옵션
- IPPROTO_TCP 레벨 옵션



❖ 소켓은 표준화되어 쉽게 사용할 수 있음

- 하지만 특별한 상황에서는 기능이나 성능을 제대로 발휘하지 못하다는 단점이 있음
- 소켓하나에 다양한 인터넷이나 프로토콜이 존재하며, 서로 연결하는데 문제가 있을 수 있음

❖ 다양한 인터넷 환경 존재

❖ 다양한 인터넷 프로토콜 존재

- 빈번한 접속 혹은 지속적인 접속 필요 혹은 불필요 (게임의 경우 지속적인 접속이 요구 됨)
- 대용량 송/수신 필요 혹은 불필요 (게임의 경우 대용량 보다는 작은 용량을 빈번히 주고 받는 경우가 많음)

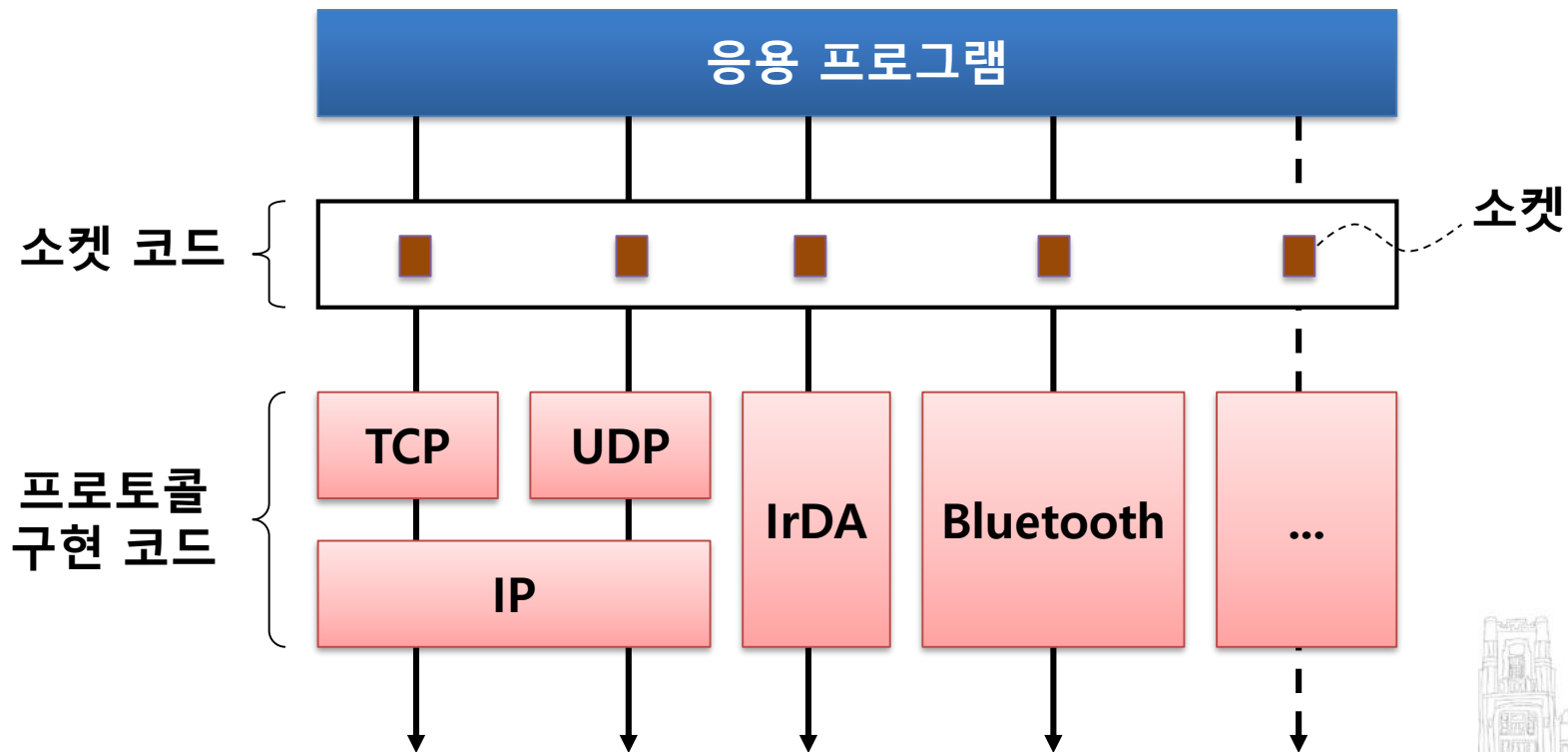


- ❖ 소켓을 이용하여 요구되는 기능을 최적화
- ❖ 송/수신되는 데이터 크기에 대한 제어
- ❖ 소켓 레벨, TCP/IP 프로토콜 구현 레벨에서의 기능 제어



❖ 소켓 프로그래밍 모델

- 응용 프로그램은 소켓 코드가 제공하는 인터페이스인 소켓 함수를 사용하므로, 통신 프로토콜을 변경하더라도 기존 코드 수정을 최소화할 수 있다는 장점이 있다.



❖ 소켓 옵션

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

❖ 참고: 7장까지는 소켓 동작의 속성을 변경하지 않고 그대로 사용하였음

❖ 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



❖ 소켓 옵션 설정(=변경)하기

```
int setsockopt (  
    SOCKET s,  
    int level,  
    int optname,  
    const char *optval,  
    int optlen  
);
```

성공: 0, 실패: SOCKET_ERROR

- s : 옵션을 적용할 대상 소켓
- level : 옵션을 해석하고 처리할 주체를 지시
- optname: 설정할 옵션의 이름
- optval : 설정할 옵션 값을 담고 있는 버퍼의 주소
- optlen: optval 이 가리키는 버퍼의 크기(byte)



❖ 현재 설정된 소켓 옵션 값 얻기

```
int getsockopt (  
    SOCKET s,  
    int level,  
    int optname,  
    char *optval,  
    int *optlen  
);
```

성공: 0, 실패: SOCKET_ERROR

- s : 옵션 값을 얻을 대상 소켓
- level : 옵션을 해석하고 처리할 주체를 지시. 예로 소켓 코드가 처리하면 SOL_*, 프로토콜 코드가 처리하면 IPPROTO_*. optname이 정해지면 자동으로 결정.
- optname: 값을 얻을 옵션의 이름
- optval : 얻은 옵션 값을 저장할 버퍼의 주소
- optlen: 값-결과(value-result) 인자로 사용



소켓 옵션의 종류 (1)

❖ level = SOL_SOCKET

optname	optval	get	set	설명
SO_BROADCAST	BOOL	.	.	브로드캐스팅 데이터 전송 허용 여부
SO_KEEPALIVE	BOOL	.	.	주기적으로 연결 상태 확인 여부
SO_LINGER	linger{}	.	.	소켓 송신 버퍼에 미전송 데이터가 있을 때 closesocket() 함수의 리턴 지연 시간 설정
SO_SNDBUF SO_RCVBUF	int	.	.	소켓 송/수신 버퍼의 크기 설정
SO_SNDTIMEO SO_RCVTIMEO	int	.	.	데이터 송/수신 함수 호출 시 타임아웃 값 설정
SO_REUSEADDR	BOOL	.	.	지역 주소(IP 주소, 포트 번호) 재사용 허용 여부



소켓 옵션의 종류 (2)

❖ level = IPPROTO_IP

optname	optval	get	set	설명
IP_HDRINCL	BOOL	.	.	데이터 송신 시 IP 헤더 직접 포함 여부
IP_TTL	int	.	.	IP 패킷의 TTL(time-to-live) 값 설정
IP_MULTICAST_IF	IN_ADDR{}	.	.	멀티캐스트 패킷을 보낼 인터페이스 선택
IP_MULTICAST_TTL	int	.	.	멀티캐스트 패킷의 TTL 값 설정
IP_MULTICAST_LOOP	BOOL	.	.	멀티캐스트 패킷의 루프백 여부
IP_ADD_MEMBERSHIP IP_DROP_MEMBERSHIP	ip_mreq{}		.	멀티캐스트 그룹 가입과 탈퇴



소켓 옵션의 종류 (3)

❖ level = IPPROTO_IPV6

optname	optval	get	set	설명
IPV6_HDRINCL	BOOL	.	.	데이터 송신 시 IP 헤더 직접 포함 여부
IPV6_UNICAST_HOPS	int	.	.	IP 패킷의 TTL(time-to-live) 값 설정
IPV6_MULTICAST_IF	DWORD	.	.	멀티캐스트 패킷을 보낼 인터페이스 선택
IPV6_MULTICAST_HOPS	int	.	.	멀티캐스트 패킷의 TTL 값 설정
IPV6_MULTICAST_LOOP	BOOL	.	.	멀티캐스트 패킷의 루프백 여부
IPV6_ADD_MEMBERSHIP IPV6_DROP_MEMBERSHIP	ipv6_mreq{}		.	멀티캐스트 그룹 가입과 탈퇴



소켓 옵션의 종류 (4)

❖ level = IPPROTO_TCP

optname	optval	get	set	설명
TCP_NODELAY	BOOL	.	.	Nagle 알고리즘 작동 여부



❖ 용도

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

❖ 사용 예

- 브로드캐스트가 기본값이 아니라 옵션인 이유?

```
BOOL bEnable = TRUE;  
retval = setsockopt(sock, SOL_SOCKET, SO_BROADCAST,  
    (char *)&bEnable, sizeof(bEnable));  
if(retval == SOCKET_ERROR) err_quit("setsockopt());
```



❖ 용도

- TCP 프로토콜 수준에서 연결 여부를 확인하려고 상대 TCP에 주기적으로(약 2시간 간격) TCP 패킷을 보냄
- TCP에서만 사용 가능

❖ 사용 예

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

- 상대 TCP가 정해진 시간 안에 응답하는 경우 : 정상 간주
- 상대 TCP가 정해진 시간 안에 응답하지 않은 경우 : 일시적인 문제가 있는 상태로 간주
- 상대 TCP가 RST(reset) 패킷으로 응답하는 경우 : 비정상적인 상태로 간주
- TCP 프로토콜 수준에서 주기적으로 끊어진 연결을 감지해 해당 소켓을 닫으므로 불필요한 시스템 자원 소모를 방지



❖ 용도

- 소켓 송신 버퍼에 미전송 데이터가 있을때, closesocket() 함수의 동작 변경

```
send(sock, ...);           // 데이터를 보낸다.  
closesocket(sock);        // 소켓을 닫는다.
```

- TCP 소켓을 사용할 경우, closesocket() 함수는 두가지 기능을 한다
 - 소켓을 닫고 할당된 운영체제 자원을 반환
 - TCP 프로토콜 수준에서 연결 종료 절차를 시작

❖ 옵션 값

- closesocket() 함수의 두가지 기능을 세부적으로 제어하는데 사용

```
struct linger {  
    u_short l_onoff;           /* option on/off*/ 값이 0이면 closesocket() 함수 바로 호출  
    u_short l_linger;         /* linger time*/ closesocket() 함수가 리턴하지 않고 대기시간 부여  
};  
typedef struct linger LINGER;
```



❖ 사용 예

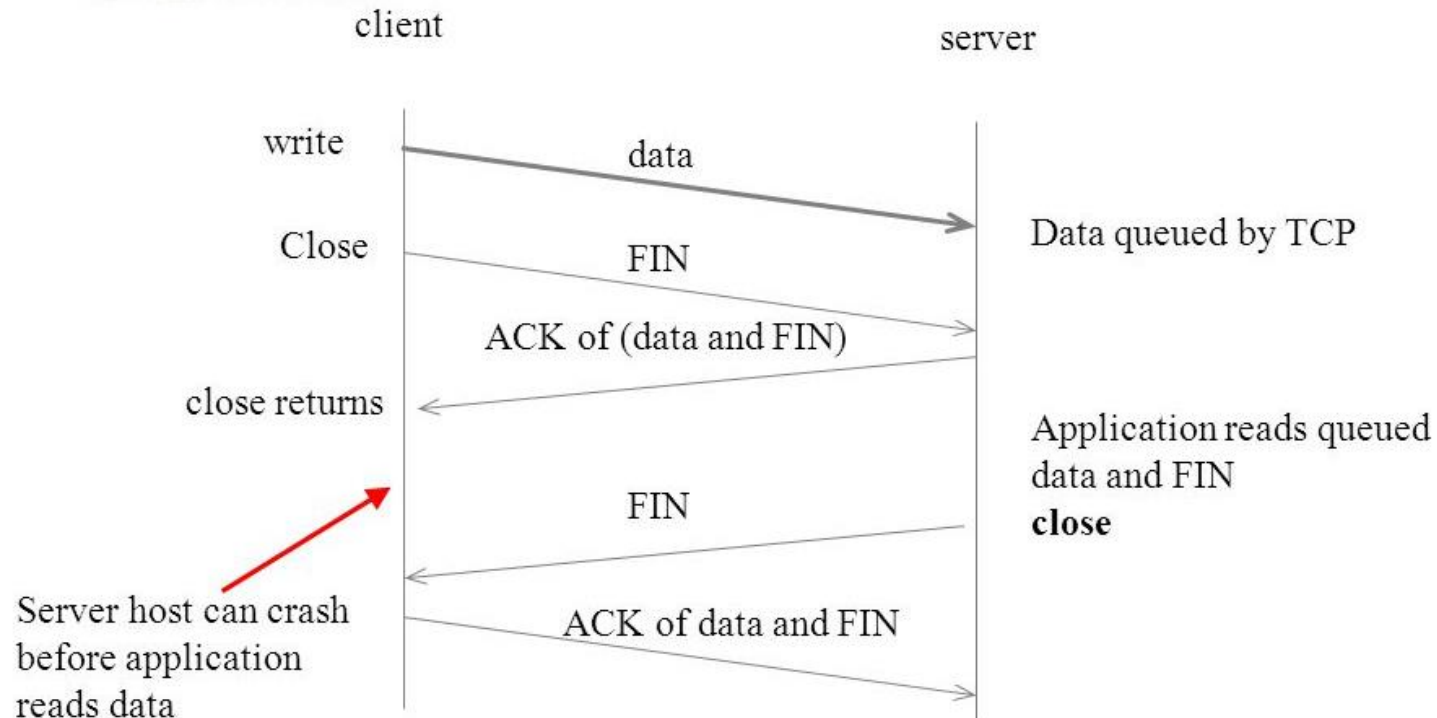
```
LINGER optval;  
optval.l_onoff = 1; /* linger on */  
optval.l_linger = 10; /* linger time = 10초 */  
retval = setsockopt(sock, SOL_SOCKET, SO_LINGER,  
    (char *)&optval, sizeof(optval));  
if(retval == SOCKET_ERROR) err_quit("setsockopt()");
```

- 송신버퍼의 데이터를 모두 보내고 TCP 연결을 정상 종료한 후 closesocket() 함수가 리턴하며, 10초 이내에 송신 버퍼의 데이터를 모두 보내지 못하면 TCP 연결을 강제 종료한 후 closesocket() 함수 리턴
- 이 경우 송신 버퍼에 남은 데이터는 삭제.



❖ 기본적인 절차

•SO_LINGER



Close with `SO_LINGER` socket option set and `l_linger` a positive value



❖ 옵션 값에 따른 closesocket() 함수 동작

struct linger{		closesocket() 함수 동작	추가 설명
l_onoff	l_linger		
0	사용 안함	①과 동일	closesocket() 함수의 기본 동작
1	0	②와 동일	
1	양수	③과 동일	

- ① closesocket() 함수는 곧바로 리턴하되, 송신 버퍼의 데이터를 백그라운드로 모두 보내고 TCP 연결을 정상 종료
- ② closesocket() 함수는 곧바로 리턴하되, 송신 버퍼의 데이터를 삭제하고 TCP 연결을 강제 종료
- ③ 송신 버퍼의 데이터를 모두 보내고 TCP 연결을 정상 종료한 후 closesocket() 함수가 리턴. 일정 시간 안에 송신 버퍼의 데이터를 모두 보내지 못하면 TCP 연결을 강제 종료한 후 closesocket() 함수가 리턴. 이때 송신 버퍼에 남은 데이터는 삭제



❖ 용도

- 운영체제가 소켓에 할당하는 송신 버퍼와 수신 버퍼 크기 변경
- 소켓이 연결되기 이전에 변경 (listen(), connect() 함수전에 옵션 설정)

❖ 사용 예

```
int optval, 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()");
```

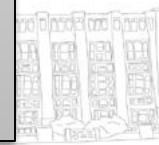
```
printf("수신 버퍼 크기(old) = %d바이트\n", optval);
```

```
// 수신 버퍼의 크기를 두 배로 늘린다.
```

```
optval *= 2;
```

```
retval = setsockopt(listen_sock, SOL_SOCKET, SO_RCVBUF,  
    (char *)&optval, sizeof(optval));
```

```
if(retval == SOCKET_ERROR) err_quit("setsockopt()");
```



❖ 실습

- buffer.cpp



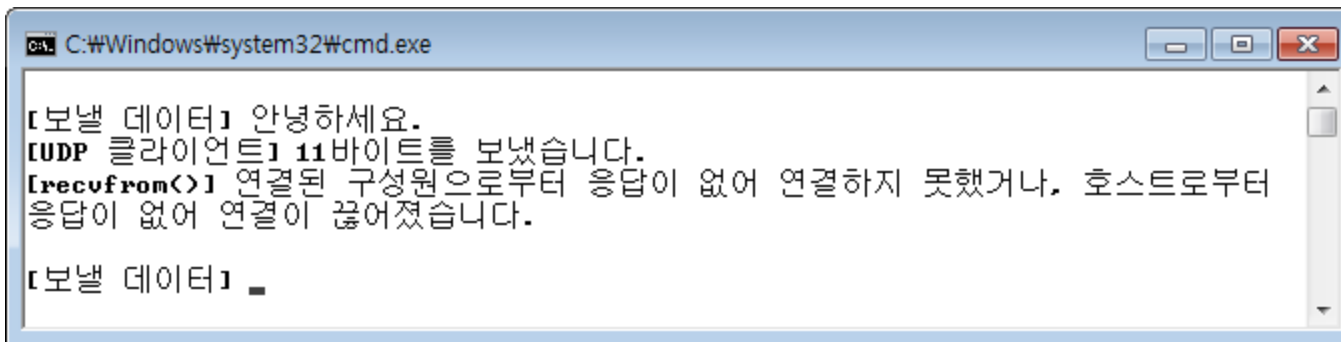
❖ 용도

- 소켓함수(socket())는 블로킹 소켓이므로데이터 전송함수 호출 시 조건이 충족하지 않으면 무한정 블록되어 교착상태 발생 (linux에서는 fork() 함수 호출시 에러 발생)
- 데이터 전송 함수가 작업 완료와 상관없이 일정 시간 후 리턴하게 함

❖ 사용 예

- TCP와 UDP 소켓에 모두 사용 가능
- 교착상태 방지 외에 다른 목적으로도 사용 가능 예)12장 추가 설명 – Ping 응용 프로그램

```
int optval = 3000;  
retval = setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO,  
    (char *)&optval, sizeof(optval));  
if(retval == SOCKET_ERROR) err_quit("setsockopt());
```



```
C:\Windows\system32\cmd.exe  
[보낼 데이터] 안녕하세요.  
[UDP 클라이언트] 11바이트를 보냈습니다.  
[recvfrom()] 연결된 구성원으로부터 응답이 없어 연결하지 못했거나, 호스트로부터  
응답이 없어 연결이 끊어졌습니다.  
[보낼 데이터] _
```



❖ 용도

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

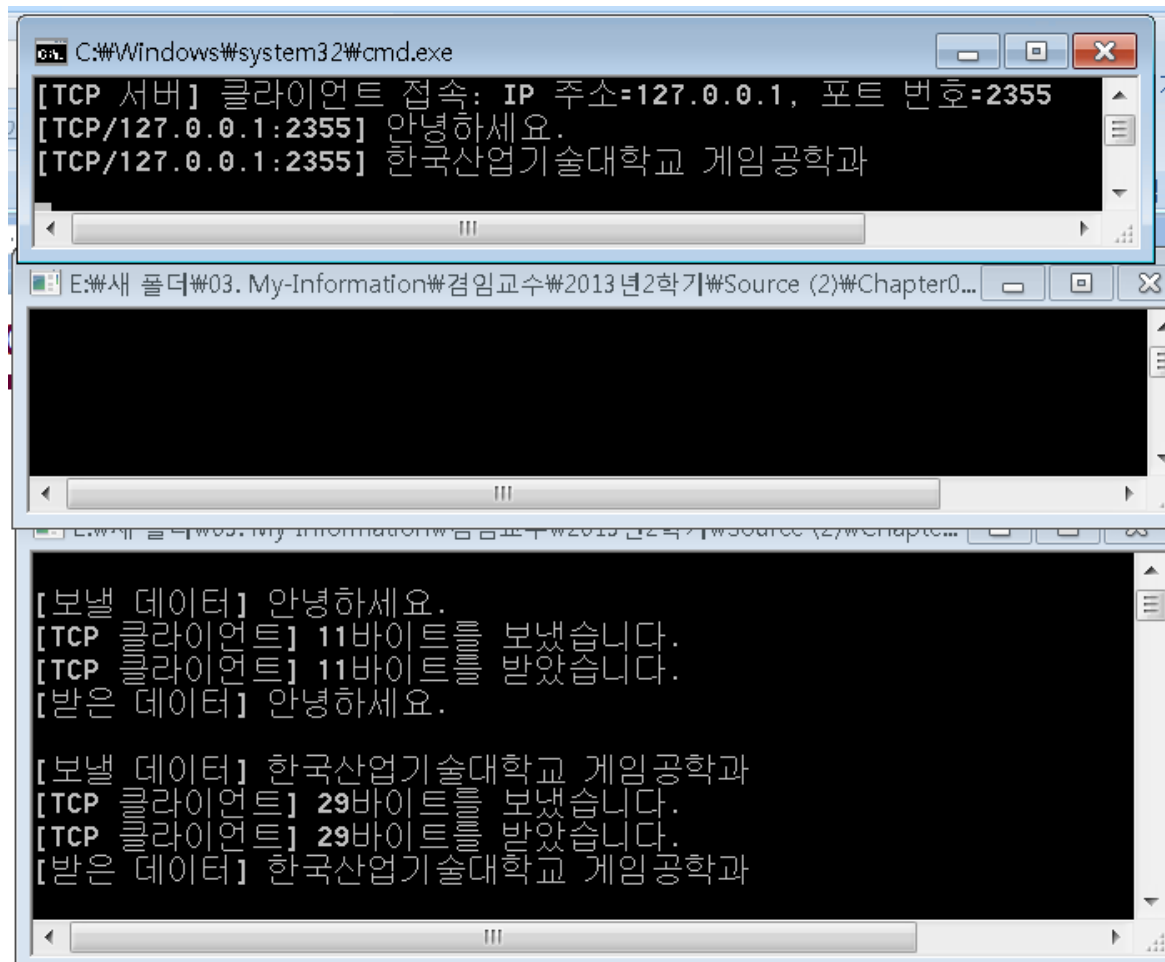
❖ 목적

- 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



실습 8-1

P271~



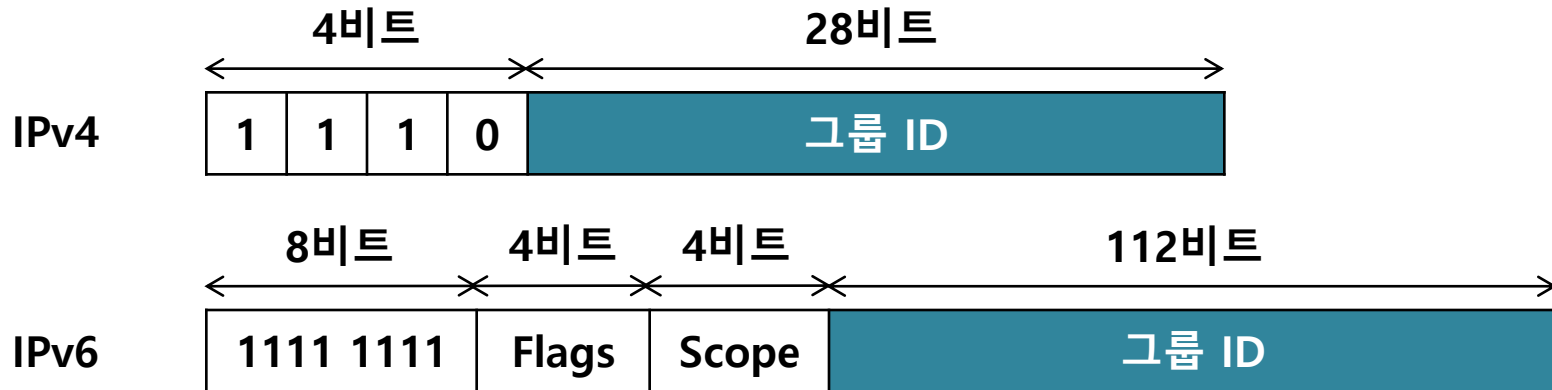
The screenshot shows three overlapping Windows command prompt windows. The top window, titled 'C:\Windows\system32\cmd.exe', displays the following text:
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=2355
[TCP/127.0.0.1:2355] 안녕하세요.
[TCP/127.0.0.1:2355] 한국산업기술대학교 게임공학과
The middle window, titled 'E:\새 폴더\03. My-Information\검임교수\2013년2학기\Source (2)\Chapter0...', is empty. The bottom window, titled 'E:\새 폴더\03. My-Information\검임교수\2013년2학기\Source (2)\Chapter0...', displays the following text:
[보낼 데이터] 안녕하세요.
[TCP 클라이언트] 11바이트를 보냈습니다.
[TCP 클라이언트] 11바이트를 받았습니다.
[받은 데이터] 안녕하세요.

[보낼 데이터] 한국산업기술대학교 게임공학과
[TCP 클라이언트] 29바이트를 보냈습니다.
[TCP 클라이언트] 29바이트를 받았습니다.
[받은 데이터] 한국산업기술대학교 게임공학과



❖ 멀티캐스트 주소

- 고정된 상위비트를 제외한 나머지 부분이 그룹ID이고, 그룹ID가 통신할 대상 멀티캐스트 그룹임



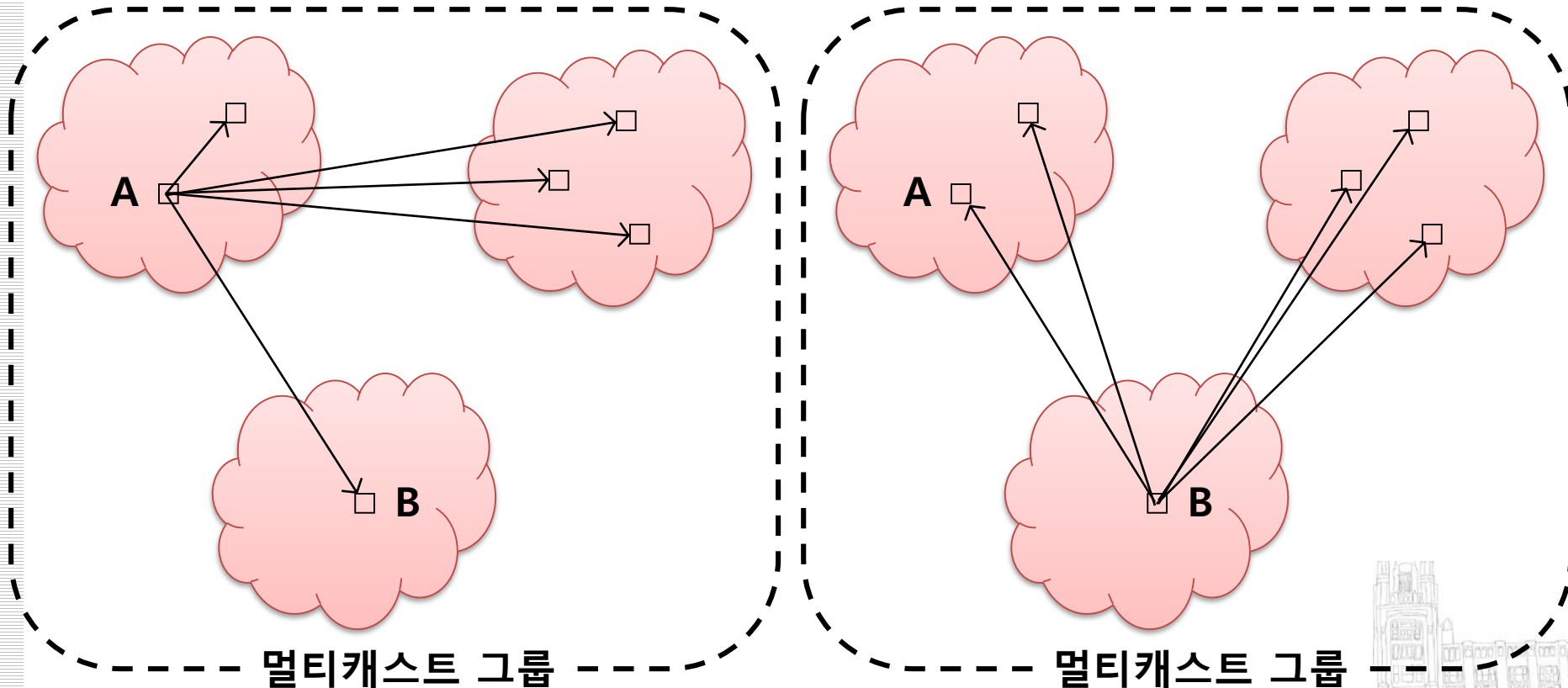
❖ 특징

- 그룹 가입과 탈퇴가 자유롭고 그룹 구성원 모두가 평등
- 멀티캐스트 데이터를 받으려면 그룹에 가입해야 함
- 멀티캐스트 데이터를 보내려고 그룹에 가입할 필요 없음
- 멀티캐스트용 IP 주소는 IPv4의 경우 224.0.0.0~239.255.255.255 임.



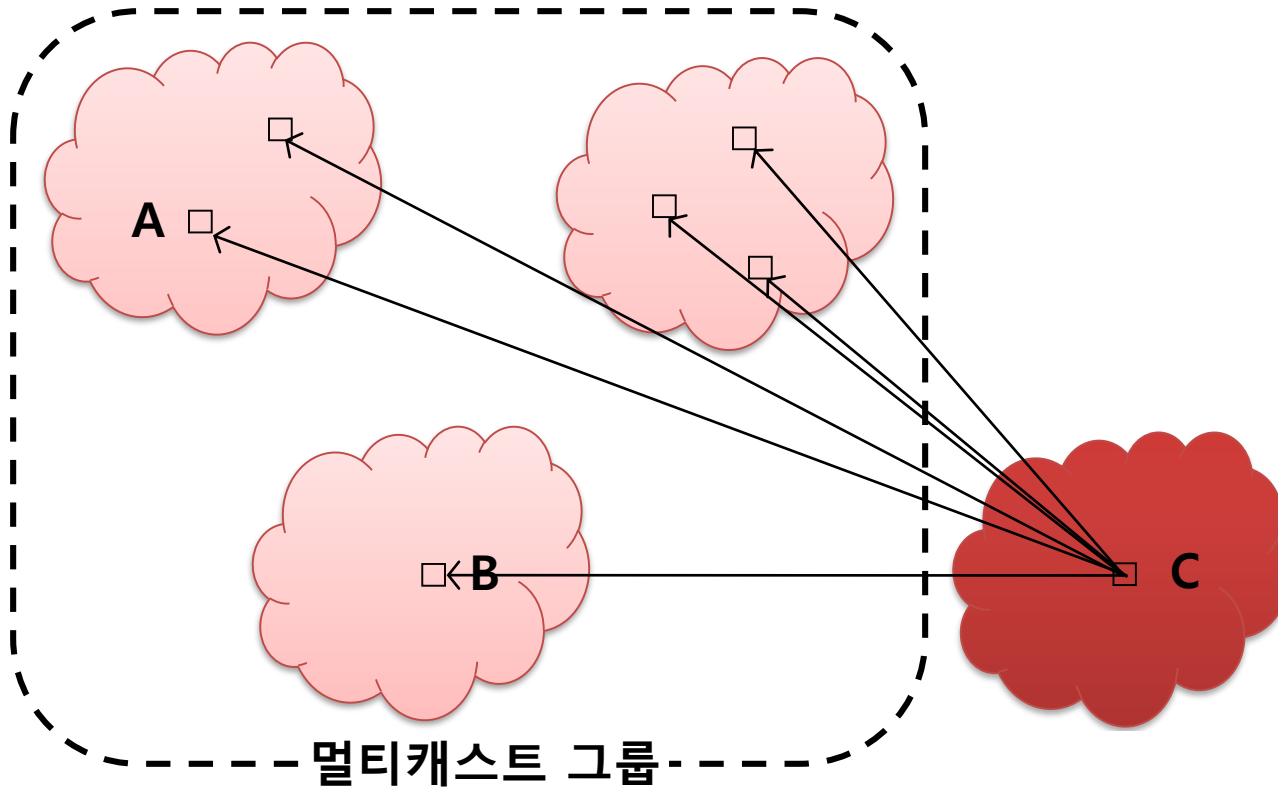
❖ 멀티캐스트 데이터 전송

- 그룹 구성원이 데이터를 보내는 경우



❖ 멀티캐스트 데이터 전송

- 그룹 비구성원이 데이터를 보내는 경우



❖ 용도

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

❖ 사용 예

- 특정 IP 주소로 멀티캐스트 데이터를 보내도록 설정한 예
- 147.46.114.70 인 네트워크 인터페이스를 통해 멀티캐스트 데이터가 전송
- 옵션을 별도로 설정하지 않으면 IP 라우팅 기능에 따라 각 패킷별로 적절한 네트워크 인터페이스가 선택되어 전송

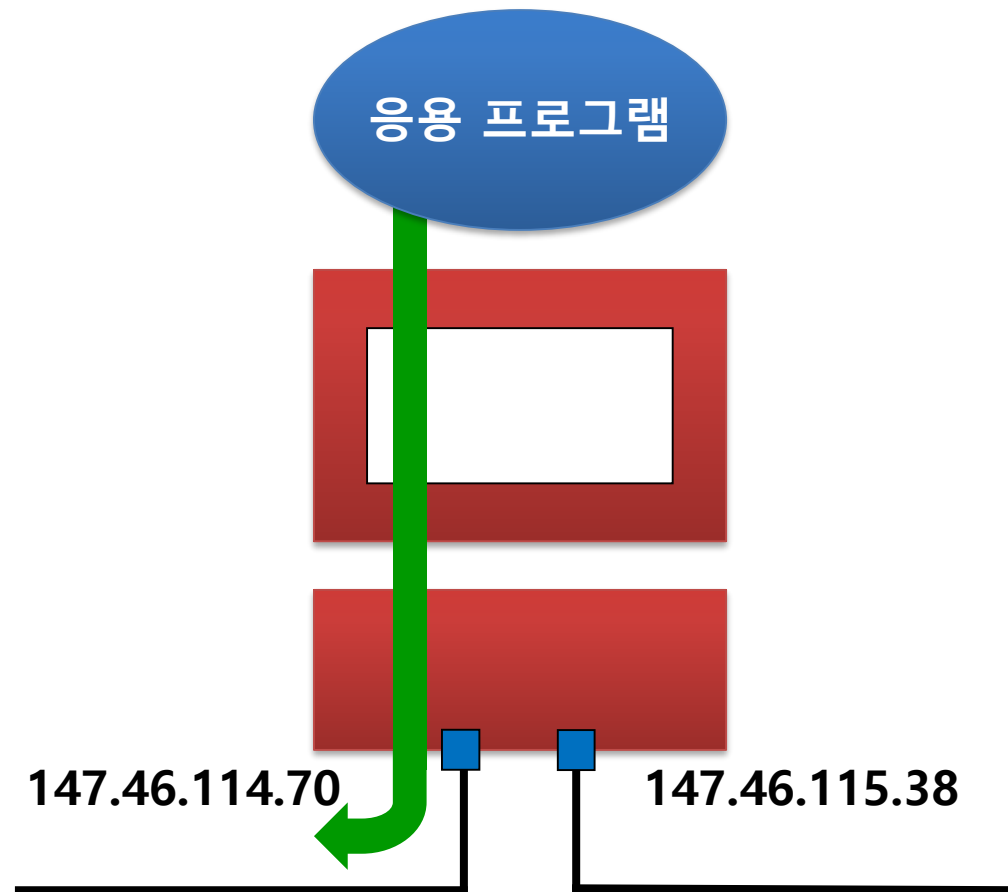
```
IN_ADDR localaddr;
```

```
localaddr.s_addr = inet_addr("147.46.114.70");
```

```
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_IF,  
(char *)&localaddr, sizeof(localaddr));
```

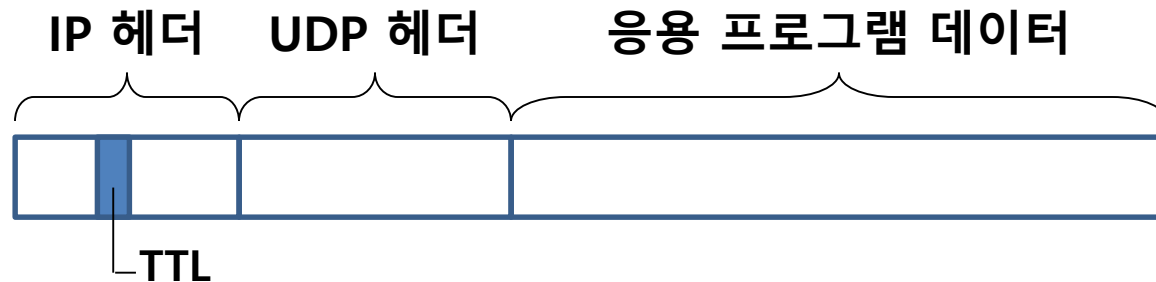


❖ IP_MULTICAST_IF 옵션 설정 결과



❖ 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



❖ 사용 예#1

```
int ttl = 2;  
retval = setsockopt(sock, IPPROTO_IP, IP_MULTICAST_TTL,  
    (char *)&ttl, sizeof(ttl));  
if(retval == SOCKET_ERROR) err_quit("setsockopt()");
```

❖ 사용 예#2

```
int ttl = 2;  
retval = setsockopt(sock, IPPROTO_IPV6, IPV6_MULTICAST_HOPS,  
    (char *)&ttl, sizeof(ttl));  
if(retval == SOCKET_ERROR) err_quit("setsockopt()");
```



❖ 용도

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

❖ 사용 예

// 자신이 보낸 멀티캐스트 데이터는 받지 않는다.

```
BOOL optval = FALSE;
```

```
setsockopt(sock, IPPROTO_IP, IP_MULTICAST_LOOP,  
           (char *)&optval, sizeof(optval));
```



❖ 용도

- 멀티캐스트 그룹에 가입 또는 탈퇴

❖ 옵션값

```
#include <ws2tcpip.h>
```

```
typedef struct ip_mreq {  
    IN_ADDR    imr_multiaddr;  
    IN_ADDR    imr_interface;  
} IP_MREQ;
```

// IPv4 multicast address

// Local IP address of interface

```
typedef struct ipv6_mreq {  
    IN6_ADDR    ipv6mr_multiaddr;  
    ULONG       ipv6mr_interface;  
} IPV6_MREQ;
```

// IPv6 multicast address

// Interface index

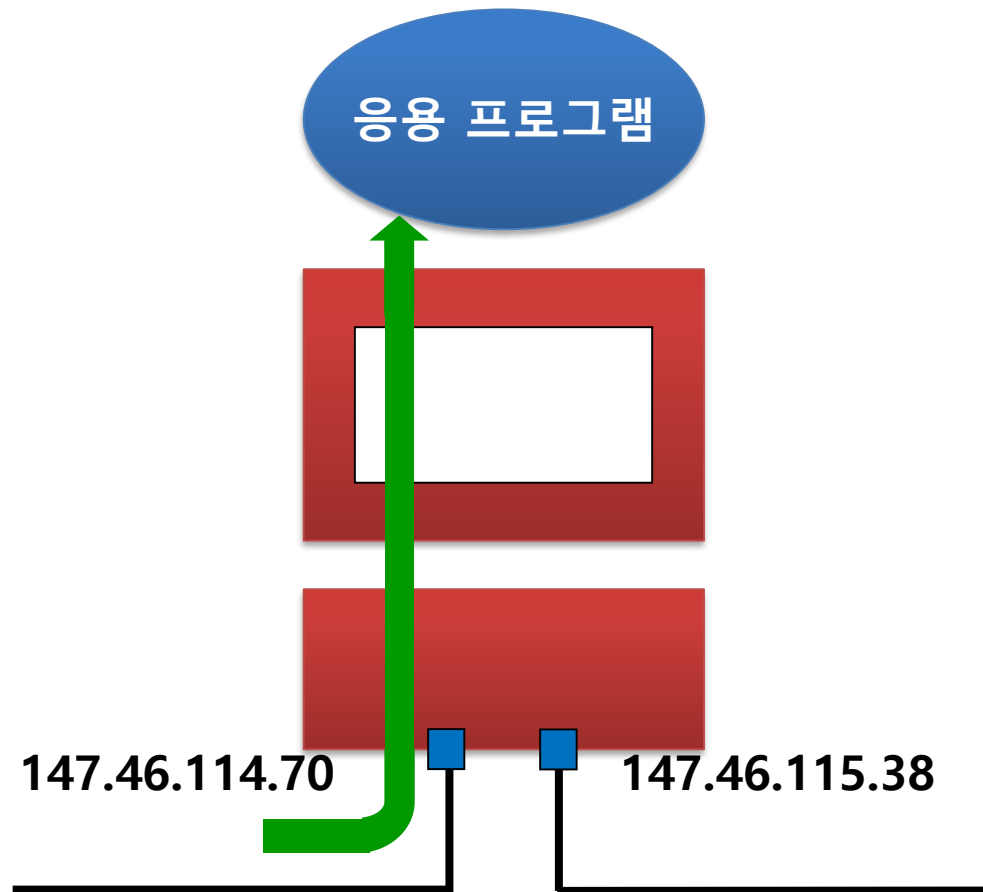
❖ 사용 예

```
struct ip_mreq mreq;  
mreq.imr_multiaddr.s_addr = inet_addr("235.7.8.9");  
mreq.imr_interface.s_addr = inet_addr("147.46.114.70");  
setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,  
           (char *)&mreq, sizeof(mreq));
```

- 특정 네트워크 인터페이스(147.46.114.70)를 멀티캐스트 그룹(235.7.8.9)에 가입
- 주소가 247.46.114.70 인 네트워크 인터페이스가 멀티캐스트 데이터를 수신



❖ IP_ADD_MEMBERSHIP 옵션 설정 결과



실습 8-2

P277~

The image displays four terminal windows arranged in a 2x2 grid, illustrating a UDP multicast session. The top-left window shows the client's perspective, receiving messages from the server. The top-right window shows the server's perspective, sending messages to the clients. The bottom-left window shows the client's perspective, receiving messages from the server. The bottom-right window shows the server's perspective, sending messages to the clients. A file explorer window is also visible in the bottom-right terminal, showing the location of the source code files.

Client Terminal (Top Left): C:\Windows\system32\cmd.exe

```
[UDP/10.1.90.1:59603] 안녕하세요.  
[UDP/10.1.90.1:54609] 산업기술대학교 게임공학과  
[UDP/10.1.90.1:54609] 게임공학과  
[UDP/10.1.90.1:59603] 오랜만..
```

Server Terminal (Top Right): C:\Windows\system32\cmd.exe

```
[보낼 데이터] 안녕하세요.  
[UDP] 11바이트를 보냈습니다.  
  
[보낼 데이터] 오랜만..  
[UDP] 8바이트를 보냈습니다.  
  
[보낼 데이터]
```

Client Terminal (Bottom Left): E:\새 폴더\03. My-Information\검임교수\2013년2학기\Source (2)\Chapter08 -

```
[UDP/10.1.90.1:54609] 산업기술대학교 게임공학과  
[UDP/10.1.90.1:54609] 게임공학과  
[UDP/10.1.90.1:59603] 오랜만..
```

Server Terminal (Bottom Right): E:\새 폴더\03. My-Information\검임교수\2013년2학기\Source (2)\Chapter08 -

```
[보낼 데이터] 산업기술대학교 게임공학과  
[UDP] 25바이트를 보냈습니다.  
  
[보낼 데이터] 게임공학과  
[UDP] 10바이트를 보냈습니다.  
  
[보낼 데이터]
```

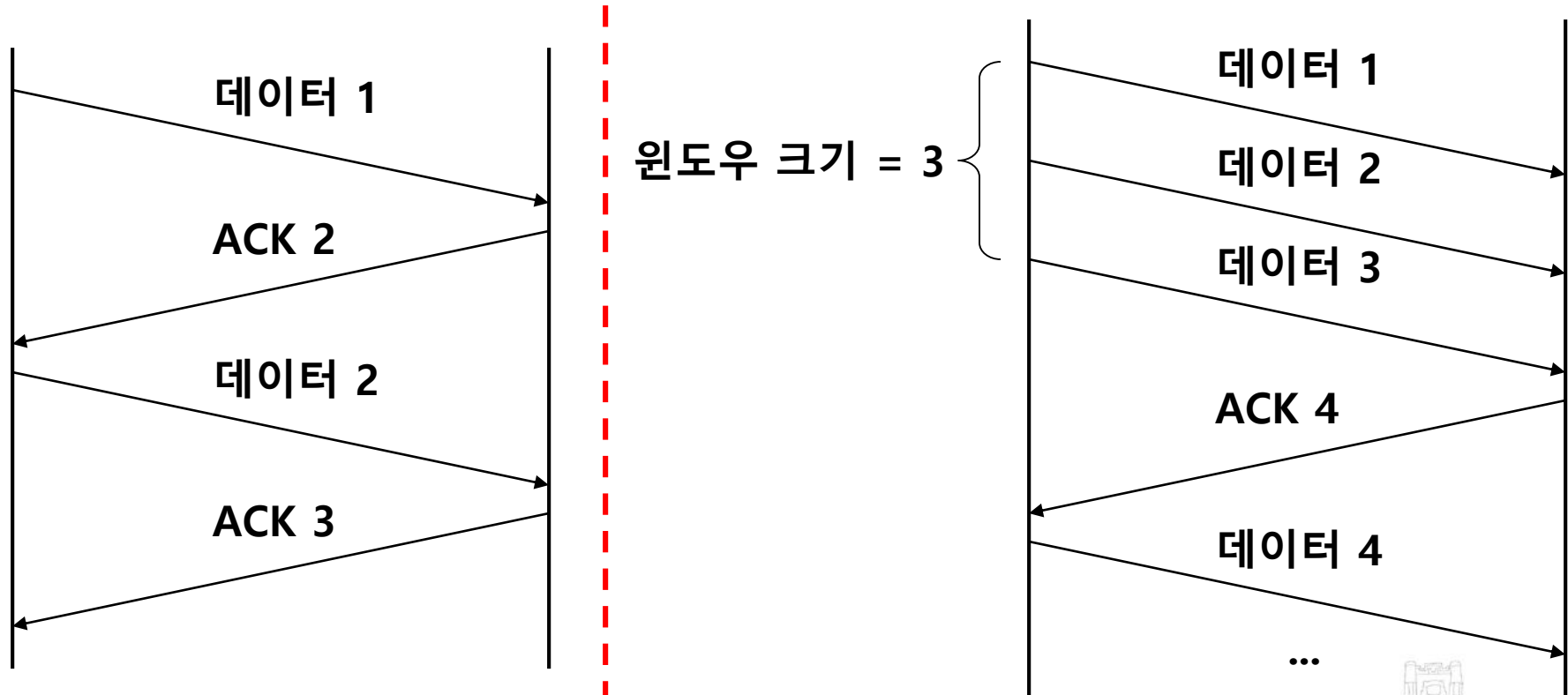
File Explorer (Bottom Right):

- 유형: 파일 폴더
- 수정한 날짜: 2013-10-02 오후 2:02
- 크기: 444KB
- 폴더: AsyncSelectTCPServer, EventSelectTCPServer, ...



TCP_NODELAY 옵션 (1)

❖ TCP 데이터 전송 원리



ACK를 이용한 데이터 수신 확인

슬라이딩 윈도우를 이용한 전송 효율 높임

❖ 용도

- Nagle(네이글) 알고리즘 작동 여부 결정

❖ Nagle 알고리즘

- 보낼 데이터가 MSS(Maximum Segment Size)로 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
- 결국, 데이터가 충분히 크면 곧바로 보내고, 그렇지 않으면 데이터가 쌓일 때까지 대기한다. 단, 데이터가 충분히 쌓이지 않았더라도 이전에 보낸 데이터를 상대방이 받았다면 다음 데이터를 보낸다



❖ 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

- Nagle 알고리즘의 장점을 포기하는 대신 응용 프로그램의 반응 속도를 빠르게 할 때 사용

```
BOOL optval = TRUE; // Nagle 알고리즘 중지
setsockopt(sock, IPPROTO_TCP, TCP_NODELAY,
(char *)&optval, sizeof(optval));
```





Thank You !

oasis01@gmail.com