5.4 Multitsaking
(한번에 여러 개의 클라이언트 다루기)

- **5.4.1 Per-Client Process**
  - 클라이언트별 접속을 프로세스로 담당하기
- **5.4.2 Per-Client Thread**
  - 클라이언트별 접속을 쓰레드로 담당하기
- **5.4.3 Constrained Multitasking**
  - 프로세스·쓰레드의 수를 제한하여 시스템 부하 줄이기

5.4.1 Per-Client Processes (클라이언트 당 프로세스들)

(1) 클라이언트로부터 들어오는 연결을 받아들인다.
(2) 그 연결을 다루기 위한 새 프로세서를 생성한다.

* 프로세서들은 ***서버 호스트의 자원들을*** 공유하며 자신의 클라이언트를 담당할 프로세서 복사본을 생성하여 각각 병행적으로 서비스한다.

단점: 자원을 많이소모하는 단점이있다.

fork()　　자식프로세서를 만든다.

1.설명

- 자신만의 PID를 가지게 되며, PPID는 부모프로세스의 PID를 가진다.
- 그밖에 PGID PGID, SID 를 상속받으며, 파일지시자, 시그널등을 상속받는다.

  **단 파일잠금(lock)와 시그널 팬딩은 상속받지 않는다.*

2.반환값

- 성공할 경우 *자식 프로세스의 PID*가 부모에게 리턴
  0 이 자식에게 리턴
- 실패할 경우　　-1 이 리턴, errno (errornumber)값 설정.

  - EAGAIN
    - 자식프로세스를 위한 테스크 구조체를 할당할 수 없을 경우
    - 주로 메모리 문제(시스템 자원)와 연관.

3. 사용법

```
#include <unistd.h>

pid_t fork(void);
```

```
pid = fork();
If( pid == 0 )
{
/* 자식 프로세서가 할 일 */
}
else
{
/* 부모 프로세서가 할 일*/
}
```

waitpid() 프로세스의 종료를 기다린다.

1.설명

- waitpid 함수는 인수로 주어진 pid 번호의 자식프로세스가 종료되거나, 시그널 함수를 호출하는 신호가 전달될때까지 waitpid 호출한 영역에서 일시 중지 된다.
- 만일 pid 로 지정된 자식이 *waitpid()* 호출전에 이미 종료되었다면, 함수는 즉시 리턴하고 자식프로세스는 "**좀비프로세스**"로 남는다.

** pid 값*

- < -1 프로세서 그룹 ID가 pid 의 절대값과 같은 자식 프로세스를 기다린다.
- *-1* *임의의 자식프로세스를 기다린다. 이것은 wait() 와 동일하다.*
- 0 프로세스 그룹 ID가 호출 프로세스의 ID와 같은 기다린다.

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t waitpid(pid_t pid, int *status, int options);
```

- options
  - WNOHANG
    - 어떠한 자식도 종료되지 않았더라도 리턴
  - WUNTRACED
    - 멈추거나 상태가 보고되지 않은 자식들을 위해 리턴
- status
  - **NULL**이 아닐경우에 status가 가리키는 곳에 프로세스의 상태를 저장

5.4.2 Per-Client Thread (클라이언트 당 쓰레드)

- 프로세스내의 멀티태스킹을 이용하여 클라이언트의 비용을 줄인다.
- 새로 생성된 쓰레드는 부모와 같은 주소 공간(코드 및 데이터)을 공유하고, 부모의 상태를 복제할 필요성을 배제한다.

pthread_creat() 새로운 쓰레드를 생성

- **1.2절. 설명**
  - pthread_exit() 을 호출하거나 또는 start_rutine 에서 return 할 경우 제거할 수 있다.
- attr 아규먼트는 쓰레드와 관련된 특성을 지정하기 위한용도로 사용attr 을 NULL 로 할경우 기본 특성으로 지정
  - 리눅스에서의 쓰레드는 joinable 과 non real-time 스케쥴 정책을 기본특성으로 한다.
- **1.3절. 반환값**
- 성공할 경우 쓰레드식별자인 thread에 쓰레드 식별번호를 저장하고, 0을 리턴
- 실패했을경우 0 이 아닌 에러코드 값을 리턴
- **1.4절. 에러**
- EAGAIN
  - 쓰레드 생성을 위한 자원이 부족하거나, PTHREAD_THREADS_MAX를 초과해서 쓰레드 생성을 요청할 경우

사용법

```
#include <pthread.h>

int pthread_create(pthread_t * thread, pthread_attr_t *
 attr, void * (*start_routine)(void *), void * arg);
```

5.4.3 Constrained Multitasking (제한 멀티태스킹)

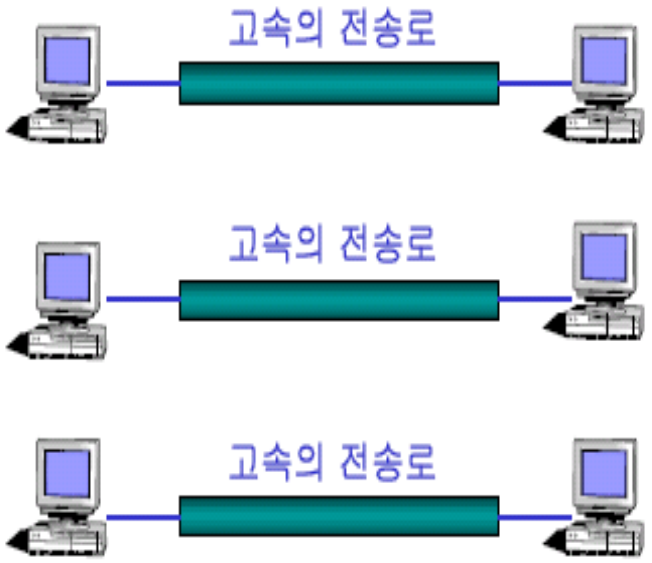
- 프로세스나 쓰레드의 생성숫자를 사용자가 임의의 제한을 두어 시스템 성능을 유지.
- Line23

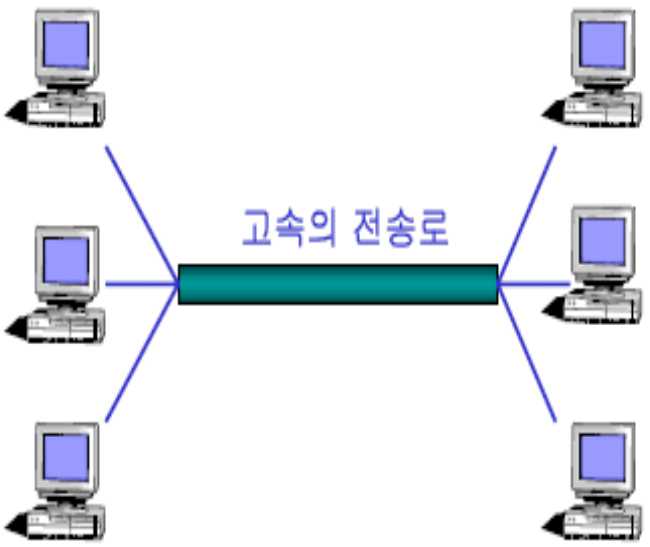
```
For(processCt=0; processCt < processLimit; processCt++)
```

사용자가 지정

5.5 멀티플렉싱(Multiplexing)

하나의 전송로를 여러 사용자가 동시에 사용해서 효율성을 극대화 함





1. 개념

하나의 통신 회선을 통하여 여러개의 입력이 결합된 형태로서 데이터 신호를 전송한 후 수신측에서는 전송된 원래 형태의 데이터 신호로 분해하여 주는 것이다. 이것은 음성과 데이터, 그리고 영상 데이터를 동시에 안전하고 신속하게 서비스해 주는데 있다.

2. 기능

- 1) 가입자측에서 다양한 속도로 접속되는 인터페이스 기능
- 2) 음성, 데이터, 영상 서비스 기능
- 3) 시스템 사이의 전송선로 자동 우회 기능
- 4) 데이터의 멀티드롭기능
- 5) 대용량 고속 전송기능

3. 특징

- 여러 채널의 신호를 한 채널에 보내야 하므로 하나의 전송 채널은 일반적으로 n배의 넓은 대역폭을 가져야한다. 여기서 대역폭이라는 것은 단위가 시간
  다중화에서는 bits per second(bps)가 되고, 주파수 다중화에서는 Hz가 된다. --
  보통 대역폭이라고 하면 주파수의 대역폭을 말하지만, 시분할 다중화에서는 데이터 전송속도 즉, bps를 대역폭이라고도 한다. 이것은 통신을 처음 공부하는 사람에게 상당한 혼동을 일으킬 수도 있는 점이다.

4. 종류

- 1)
- 시분할 다중화
- (Time Division Multiplexing, TDM) :
- 디지털 신호에 대해서 베이스 밴드에서 사용한다.
-
- 2)
- 주파수 분할 다중화
- (Frequency Division Multiplexing, FDM):
- 아날로그 신호에 대해서 브로드 밴드에서 사용한다.
-

시분할 다중화

- 시간을 여러 개의 조각(타임 슬롯)으로 나누어서 거기에 각 채널의 데이터를 넣어서 보내는 방식이다.
- 디지털 신호로 전송되며, 아날로그 데이터에는 직접적으로 사용하지 못하므로 양자화하고 코드화해서 디지털 신호로 만든 후에 사용할 수 있다.

1) 동기식(Synchronous) TDM

이 방식은 고전적인 회선 교환(circuit-switching)에서 사용되는 것으로 각 입력에 버퍼를 두지 않고, 타임 슬롯을 모든 이용자에게 규칙적으로 할당한다. 즉, 각 채널에서 데이터를 보내던 안 보내던 타임 슬롯을 미리 할당하고 고정시킨다.

장점: 구현이 간단하다.
단점: 입력이 없는 채널이 있어도 계속 대역폭은 낭비되어야 한다.

적용 예: DS-1 프레임 포맷, ISDN, SONET/SDH

- 2) 비동기식(Asynchronous) 또는 통계적(Statistical) TDM

이 방식은 인터넷에서 사용되는 패킷 교환에 적용되는 방식으로 각 입력에 버퍼를 두고 입력이 있는 채널에만 타임 슬롯을 할당하는 동적 할당 방식이다.

장점: 입력이 있는 채널에만 타임 슬롯을 할당하기 때문에 대역폭이 남을 수 있고, 그 남는 대역폭 동안 다른 일을 할 수 있다.

단점: 각 채널마다 헤드를 삽입하기 때문에 구현이 다소 복잡하다.
적용 예: HDLC 프레임 포멧

주파수 분할 다중화

- **1. 개념**
- 주파수를 여러개로 나누어서 동시에 한 채널로 전송하는 방식으로 아날로그 신호를 변조해서 각 대역폭을 합해서 보내다. 흔히 볼 수 있는 예가 방송국에서 보내는 전파이다. 이 전파에는 TV나 라디오의 각 채널에 있는 변조된 신호들이 공중에서 동시에 전송된다. 여기서는 공중이 바로 하나의 채널이 되는 것이다.

select 를 사용한 서버 설계

Select 함수의 소개

- 프로세스가 kernel에게 여러 사건 중에서 하나이상이 발생할 때까지 기다리다가 하나 이상의 사건(accpet,recve등)이 발생하거나 또는 지정된 시간이 경과할 때만 프로세스를 깨우라고 지시
- 서버가 어떤 일을 하는 동안 블록 실행을 하는것을 허락해 줌

```
#include <sys/select.h>
#include <sys/time.h>
Int select(int maxfdp1, fd_set *readset , fd_set *writeset ,
        fd_set *exceptset, const struct timeval * timeout);
```

Select 함수를 사용한 서버 설계

Select 인수

1. 최대 숫자(n)파일기술자에서 테스트.
2. 파일 기술자 집합(readfds) 읽는 자료로부터 테스트
3. 파일 기술자 집합(writefds) 쓰기 가능 으로부터 테스트
4. 파일 기술자 집합(exceptfds) 예외에 관한 테스트.
5. 포인터 timeout 요구는 함수 호출을 선정,
   포인터가 null 이면 no timeout 을 지시한다.

Select 함수를 사용한 서버 설계

select 함수 리턴값

-1 함수 호출에 에러 발생

0 아무것도 발생하지 않고 timeout 시

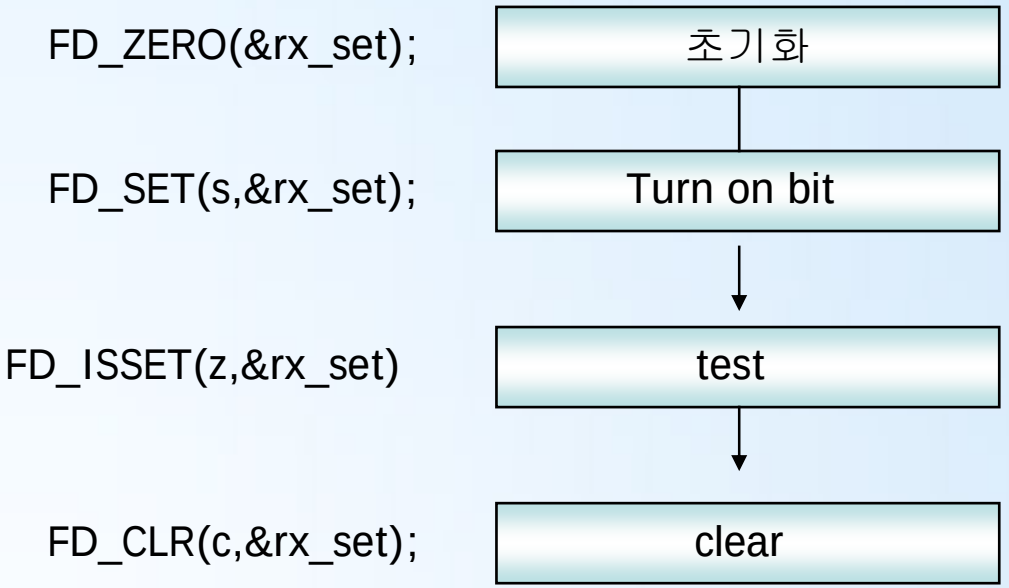
0 보다 큰 숫자는 파일기술자가 무언가를 생성

Timeval 구조

```
struct timeval {
    long tv_sec;
    long tv_use;
    };
```

Select 함수를 사용한 서버 설계

Int select(int maxfdp1, &rx_set , fd_set*writeset ,
fd_set *exceptset, const struct timeval * timeout);



Select()의 개요

- **select()**

```
int select(int nfds, fd_set *readfds,
                  fd_set *writefds,
                  fd_set *exceptfds,
                  struct timeval *timeout);
```

  - select() waits for a number of file descriptors to change status
- **Macros for manipulating fd_set**
  - **FD_ZERO(fd_set *set);**
  - **FD_SET(int fd, fd_set *set);**
  - **FD_CLEAR(int fd, fd_set *set);**
  - **FD_ISSET(int fd, fd_set *set);**

5.6 여러 수신자들(Multiple Recipients)

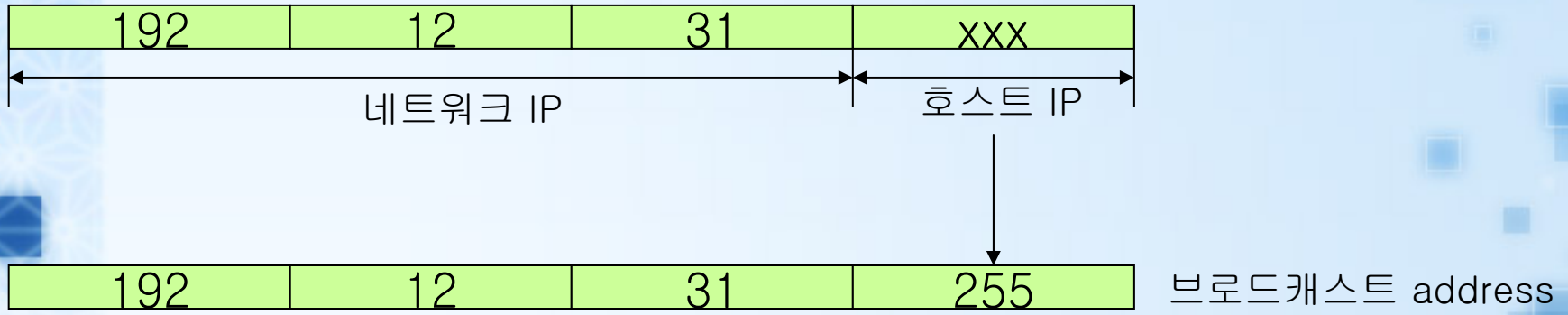
- 5.6.1 브로드캐스트
- 5.6.2 멀티캐스트

– 5.6.1브로드캐스트(Broadcast)

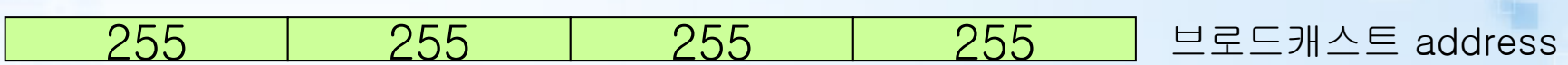
- 전송 방식.
  - **UDP**를 기반으로 하는 전송 방식**(**멀티캐스트와 같다**).**
  - 일반적인 **UDP** 패킷과의 차이점은 전송 목적지 **IP**주소 뿐이다**.**
  - 동일 네트워크에 속하는 모든 호스트에 동시 전송**(**멀티캐스트와의 차이점**).**
  - 인터넷상에서는 지역 네트워크내에서만 브로드캐스트를 허용한다**(**네트워크의 부하를 고려**).**

❒ 주소선택에 따른 브로드캐스트 방식의 구분.

- 지정된 브로드캐스트 : 예 **192.12.31.255**



지역적 브로드캐스트 : 예 255.255.255.255



❑ 브로드캐스트 Sender 예제

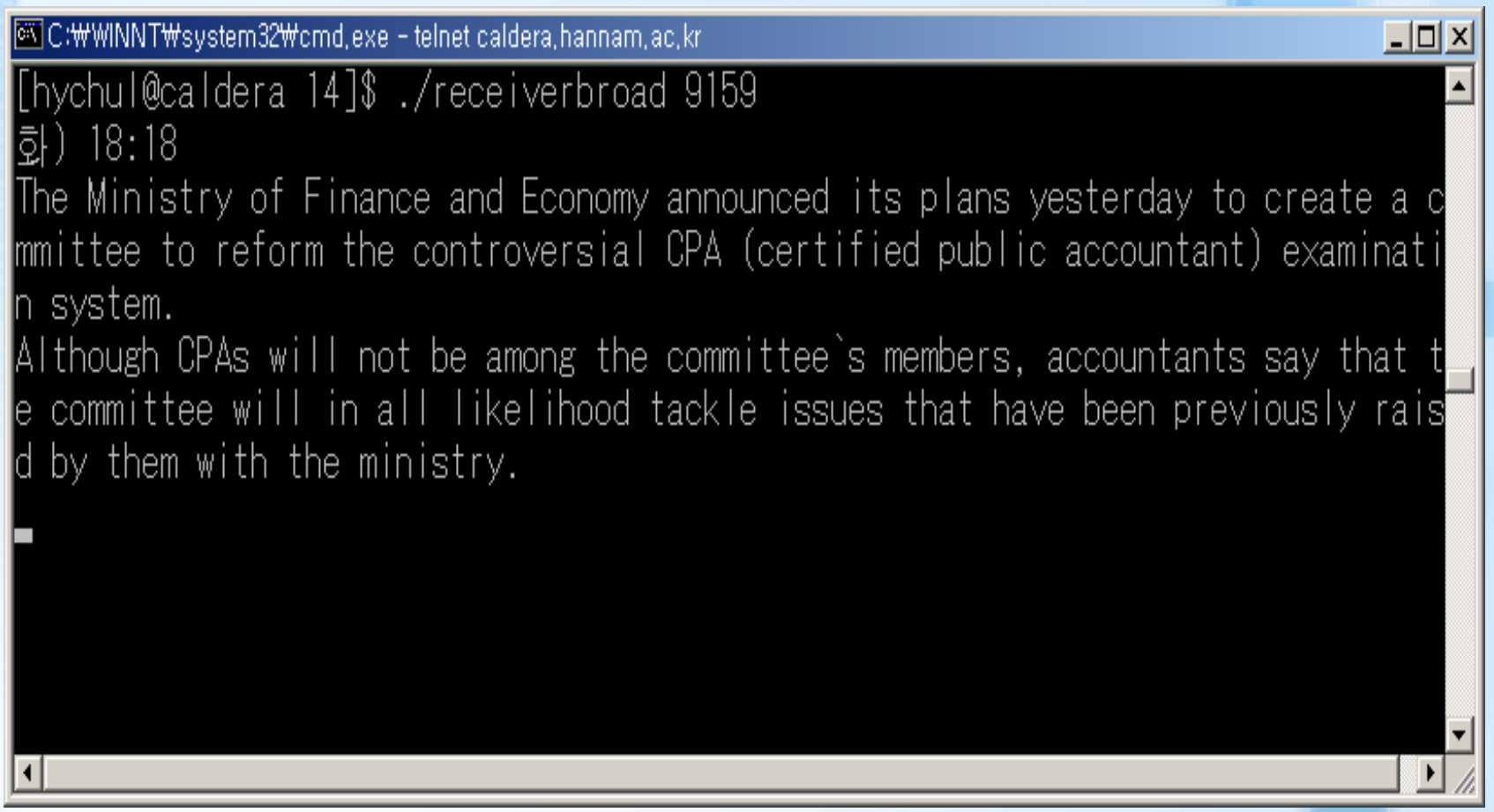
```
        :
send_sock=socket(PF_INET, SOCK_DGRAM, 0);
                                    /*브로드캐스트를 위해 UDP소켓 생성 */ :
state=setsockopt(send_sock, SOL_SOCKET, SO_BROADCAST,
                    (void*)&so_broadcast, sizeof(so_broadcast));
        :
if((fp=fopen("News.txt", "r"))==NULL) /* news 파일 오픈 */
    error_handling("fopen() error");

while(!feof(fp)){    /* news 방송 */
    fgets(buf, BUFSIZE, fp);
    sendto(send_sock, buf, strlen(buf), 0,
                    (struct sockaddr*)&broad_addr, sizeof(broad_addr));
                    :
```

❑ 브로드캐스트 Sender의 실행결과

❑ 브로드캐스트 Receiver 예제

```
        :
recv_sock=socket(PF_INET, SOCK_DGRAM, 0);
                                /*브로드캐스트를 위한 UDP소켓 생성 */
        :
memset(&addr, 0, sizeof(addr));
addr.sin_family=AF_INET;
addr.sin_addr.s_addr=htonl(INADDR_ANY);
addr.sin_port=htons(atoi(argv[1]));     /* 브로드캐스트 port 설정 */

if(bind(recv_sock, (struct sockaddr*)&addr, sizeof(addr))==-1)
          error_handling("bind() error");
        :
```

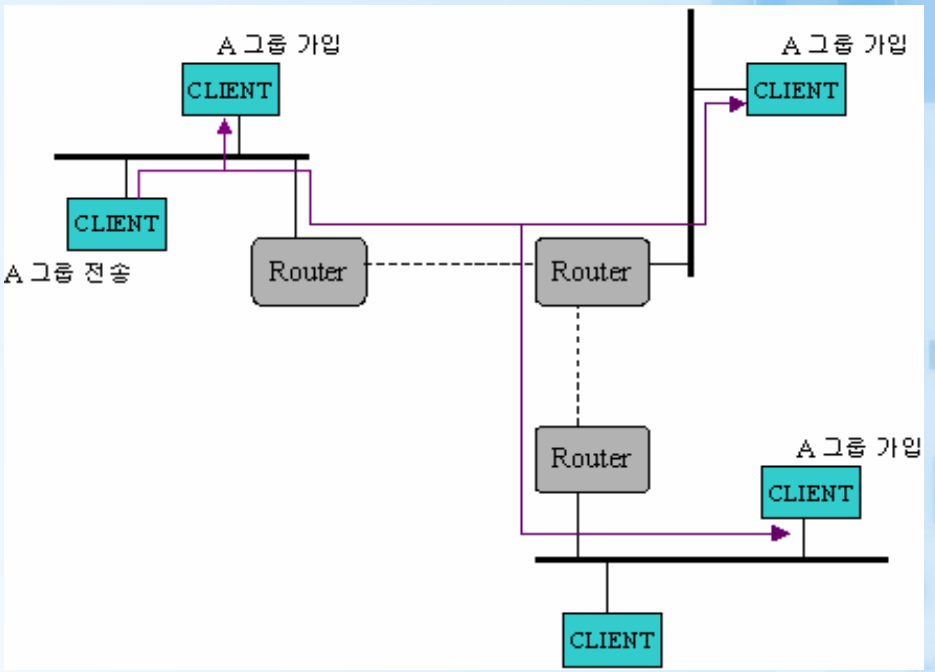
❑ 브로드캐스트 Receiver 의 실행결과



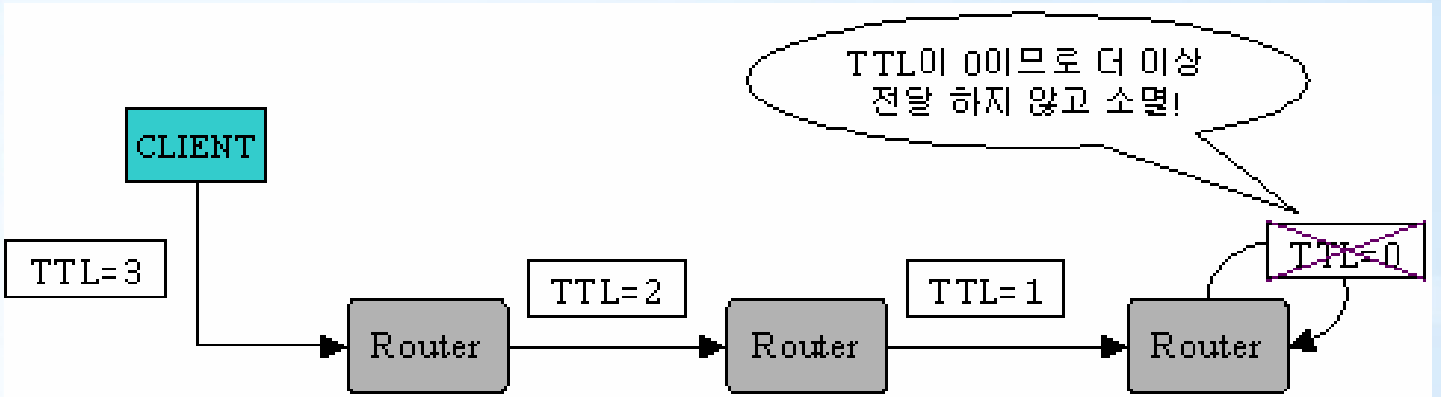
- 5.6.2 멀티캐스트(Multicast)

❑ 전송방식

- **UDP**를 기반으로 하는 전송 방식.
- 멀티캐스트 그룹을 기반으로 멀티캐스트 패킷을 주고 받음.
- 하나의 멀티캐스트 패킷은 라우터를 통해서 다수의 호스트에 전송.



- 라우팅(Routing)과 TTL(Time To Live)
  - 라우터에 의해서 패킷이 경로를 찾는 과정을 라우팅이라 한다.
  - 멀티캐스트 패킷 내에는 **TTL** 정보가 포함된다. **TTL**은 거쳐 갈 수 있는 라우터의 수를 의미한다



❒ 멀티캐스트 Sender와 Receiver.

- **Sender :** 임의의 멀티캐스트 그룹에 데이터를 전송하는 호스트
- **Receiver :** 임의의 멀티캐스트 그룹으로부터 데이터를 수신하는 호스트

❒ 멀티캐스트 Sender와 Receiver의 구현

Sender	Receiver
• UDP 소켓 생성. • TTL 설정(소켓 옵션 설정). • 멀티캐스트 그룹으로 데이터 전송.	• UDP 소켓 생성. • 멀티캐스트 그룹 지정 (ip_mreq 구조체). • 멀티캐스트 그룹 가입 (소켓 옵션 설정).

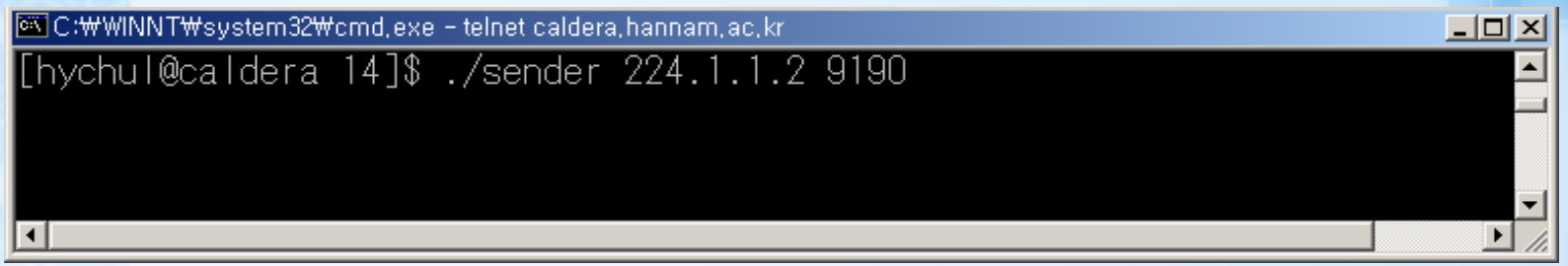
❑ 멀티캐스트 Sender 예제

```
        :
#define TTL 64
        :
send_sock=socket(PF_INET, SOCK_DGRAM, 0);
                                /*멀티캐스트를 위한 UDP소켓 생성 */
 if(send_sock == -1)
   error_handling("socket() error");

 memset(&multi_addr, 0, sizeof(multi_addr));
 multi_addr.sin_family=AF_INET;
 multi_addr.sin_addr.s_addr=inet_addr(argv[1]); /* 멀티캐스트 IP 설정 */
 multi_addr.sin_port=htons(atoi(argv[2]));              /* 멀티캐스트 port 설정 */

 state=setsockopt(send_sock, IPPROTO_IP, IP_MULTICAST_TTL, (void*)
    &multi_TTL, sizeof(multi_TTL));
        :
```

❑ 멀티캐스트 Sender 예제

```
            :
if((fp=fopen("News.txt", "r"))==NULL) /* news 파일 오픈 */
        error_handling("fopen() error");

  while(!feof(fp)){   /* news 방송 */
        fgets(buf, BUFSIZE, fp);
        sendto(send_sock, buf, strlen(buf), 0,
                (struct sockaddr*)&multi_addr, sizeof(multi_addr));
        sleep(2);
  }
            :
```

❑ 멀티캐스트 Sender의 실행결과



❑ 멀티캐스트 Reciver 예제

```
            :
int recv_sock;
  struct sockaddr_in addr;
  int state, str_len;
  char buf[BUFSIZE];
  struct ip_mreq join_addr;
            :
recv_sock=socket(PF_INET, SOCK_DGRAM, 0); /*멀티캐스트를 위해 UDP소켓 생성
     */
  if(recv_sock == -1)
    error_handling("socket() error");
            :
  addr.sin_port=htons(atoi(argv[2]));          /* 멀티캐스트 port 설정 */
            :
```

멀티캐스트 Receiver 예제

```
        :
join_addr.imr_multiaddr.s_addr=inet_addr(argv[1]);
join_addr.imr_interface.s_addr=htonl(INADDR_ANY);

state=setsockopt(recv_sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
    (void*)&join_addr, sizeof(join_addr));
        :
 while(1){
    str_len=recvfrom(recv_sock, buf, BUFSIZE-1, 0, NULL, 0);
    if(str_len<0) break;
    buf[str_len]=0;
    fputs(buf, stdout);
 }
        :
```

❑ 멀티캐스트 Receiver의 실행결과