

Global KPU!

글로벌 경쟁력을 갖춘 산업기술 명문대학  
세계를 향해 더 큰 미래를 펼쳐갑니다

## 10장. 소켓 입출력 모델(I)

네트워크 게임 프로그래밍

- ❖ 블로킹과 논블로킹 소켓의 특징을 이해한다.
- ❖ Select 소켓 입출력 모델을 이해하고 활용한다.
- ❖ WSAAsyncSelect 소켓 입출력 모델을 이해하고 활용한다.
- ❖ WSAEventSelect 소켓 입출력 모델을 이해하고 활용한다.



## ❖ 소켓 모드

- 블로킹 소켓과 년블로킹 소켓으로 구분

## ❖ 블로킹 소켓

- 소켓 함수 호출 시 조건이 만족되지 않으면 함수가 리턴하지 않고 스레드 실행이 정지
- 조건이 만족되면 소켓 함수가 리턴하면서 정지된 스레드가 깨어나 실행을 재개

| 소켓 함수              | 리턴 조건                                                    |
|--------------------|----------------------------------------------------------|
| accept()           | 접속한 클라이언트가 있을 때                                          |
| connect()          | 서버에 접속이 성공했을 때                                           |
| send(), sendto()   | 응용 프로그램이 전송을 요청한 데이터를 소켓 송신 버퍼에 모두 복사했을 때                |
| recv(), recvfrom() | 소켓 수신 버퍼에 도착한 데이터가 1바이트 이상 있고 이를 응용 프로그램이 제공한 버퍼에 복사했을 때 |



## ❖ 년블로킹 소켓

- 소켓 함수 호출 시 조건이 만족되지 않더라도 함수가 리턴하므로 스레드가 중단 없이 다음 코드를 수행
- socket() 함수는 기본적으로 블로킹 소켓을 생성하므로 **ioctlsocket()** 함수를 호출해 년블로킹 소켓으로 전환

// 블로킹 소켓 생성

```
SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);  
if(sock == INVALID_SOCKET) err_quit("socket()");
```

// 년블로킹 소켓으로 전환

```
u_long on = 1;  
retval = ioctlsocket(sock, FIONBIO, &on);  
if(retval == SOCKET_ERROR) err_quit("ioctlsocket()");
```



## ❖ 년블로킹 소켓과 소켓 함수

- 년블로킹 소켓에 대해 소켓 함수를 호출할 때 조건이 만족되지 않으면 소켓 함수는 오류를 리턴
- WSAGetLastError() 함수를 이용해 오류 코드를 확인
- 대개 오류 코드는 **WSAEWOULDBLOCK**
  - 조건이 만족되지 않았음을 나타내므로 나중에 다시 소켓 함수를 호출하면 됨
  - 실제 오류가 아님을 주의

## ❖ 교착상태 발생 (블로킹) 해결 방안

- 멀티쓰레딩 혹은 멀티프로세스
- 년블로킹 I / O
- Select 함수 – 이상 리눅스 mac등 지원
- 기타 소켓 입출력 모델(IOCP등) – 이상 윈도우즈



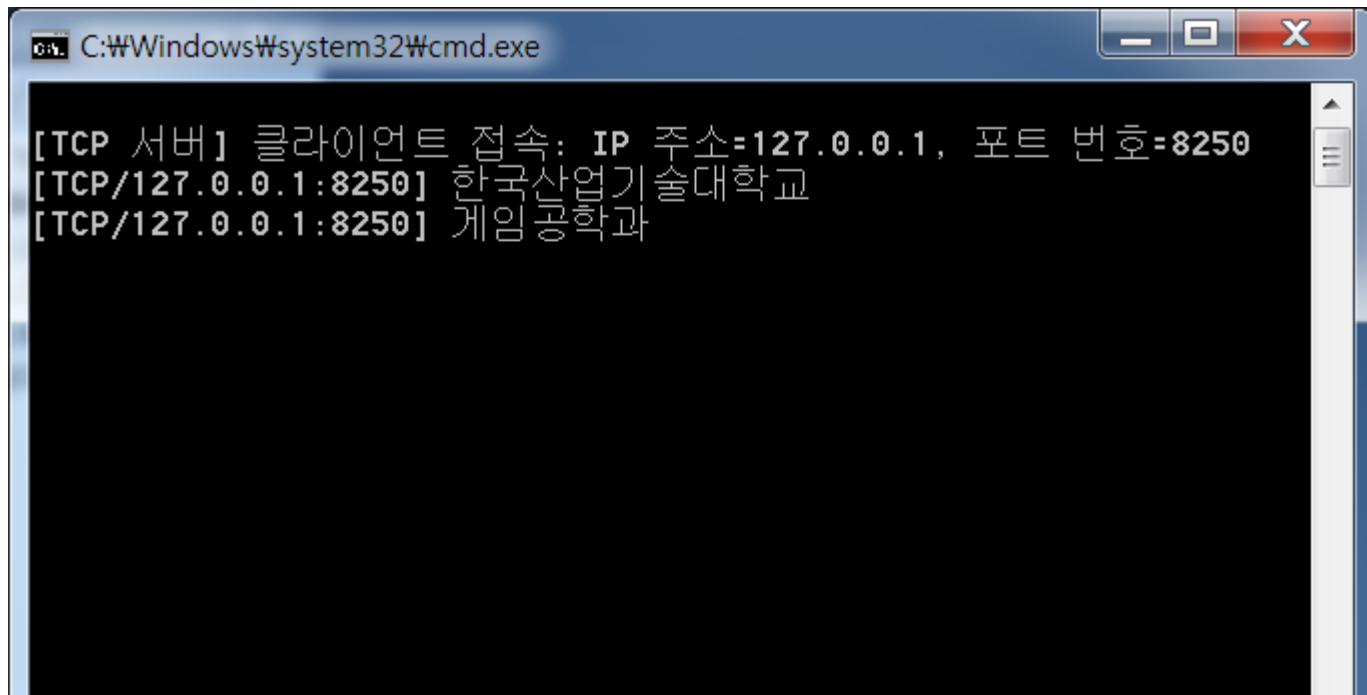
## ❖ 년블로킹 소켓의 특징

|    |                                                                                                                                                                                                                          |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 장점 | <ul style="list-style-type: none"><li>■ 소켓 함수 호출 시 항상 리턴하므로 조건이 만족되지 않아 스레드가 오랜 시간 정지하는 상황, 즉 교착 상태가 생기지 않는다.</li><li>■ 멀티스레드를 사용하지 않고도 여러 소켓에 대해 돌아가면서 입출력을 처리할 수 있다. 필요하다면 중간에 소켓과 직접 관계가 없는 다른 작업을 할 수도 있다.</li></ul> |
| 단점 | <ul style="list-style-type: none"><li>■ 소켓 함수를 호출할 때마다 WSAEWOULDBLOCK과 같은 오류 코드를 확인하고 처리해야 하므로 프로그램 구조가 복잡해진다.</li><li>■ 블로킹 소켓을 사용한 경우보다 CPU 사용률이 높다.</li></ul>                                                         |



## 실습 10-1 P361~

기존 TCP SERVER 예제와 기능은 같지만 CPU 점유율(사용율)이 높다.  
이는 접속한 클라이언트가 없어도 accept() 함수가 리턴하여 계속 수행하기 때문이다.



```
C:\Windows\system32\cmd.exe

[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=8250
[TCP/127.0.0.1:8250] 한국산업기술대학교
[TCP/127.0.0.1:8250] 게임공학과
```

## ❖ 반복 서버

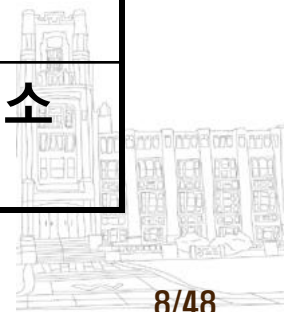
- 여러 클라이언트를 한 번에 하나씩 처리
- UDP 서버에 적합

|    |                                           |
|----|-------------------------------------------|
| 장점 | 스레드 한 개만으로 구현하므로 시스템 자원 소모가 적음            |
| 단점 | 한 클라이언트의 처리 시간이 길어지면 다른 클라이언트의 대기 시간이 길어짐 |

## ❖ 병행 서버

- 여러 클라이언트를 동시에 처리 (예. 6장 멀티쓰레드)
- TCP 서버에 적합

|    |                                            |
|----|--------------------------------------------|
| 장점 | 한 클라이언트의 처리 시간이 길어지더라도 다른 클라이언트에 영향을 주지 않음 |
| 단점 | 스레드를 여러 개 생성하여 구현하므로 시스템 자원 소모가 많음         |





## ❖ 이상적인 서버의 기능

- 가능한 많은 클라이언트가 접속 가능
- 서버는 각 클라이언트의 서비스 요청에 빠르게 반응하며 고속으로 데이터를 전송
- 시스템 자원(CPU, Memory 등) 사용량을 최소화

## ❖ 이상적인 소켓 입출력 모델의 특징

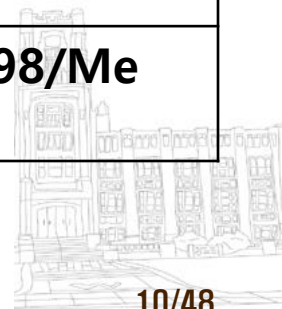
- 소켓 함수 호출 시 블로킹을 최소화
- 스레드 개수를 일정 수준으로 유지
- CPU 명령 수행과 입출력 작업을 병행
- 유저 모드와 커널 모드 전환 횟수를 최소화



## ❖ 윈도우 운영체제의 소켓 입출력 모델 지원

- WINDOWS NT 계열 서버와 윈도우 7/10등은 소켓 입출력 모델 모두 지원

| 소켓 입출력 모델              | 운영체제 버전        |                                |                |
|------------------------|----------------|--------------------------------|----------------|
|                        | 윈도우 CE         | 윈도우<br>(클라이언트 버전)              | 윈도우<br>(서버 버전) |
| <b>Select</b>          | CE 1.0 이상      | 윈도우 95 이상                      | 윈도우 NT 이상      |
| <b>WSAAsyncSelect</b>  | x              | 윈도우 95 이상                      | 윈도우 NT 이상      |
| <b>WSAEventSelect</b>  | CE .NET 4.0 이상 | 윈도우 95 이상                      | 윈도우 NT 3.51 이상 |
| <b>Overlapped</b>      | CE .NET 4.0 이상 | 윈도우 95 이상                      | 윈도우 NT 3.51 이상 |
| <b>Completion Port</b> | x              | 윈도우 NT 3.5 이상(윈도우 95/98/Me 제외) |                |



## ❖ Select 모델

- select() 함수가 핵심적인 역할을 함
- 소켓 모드(블로킹, 년블로킹)와 관계없이 여러 소켓을 한 스레드로 처리 가능

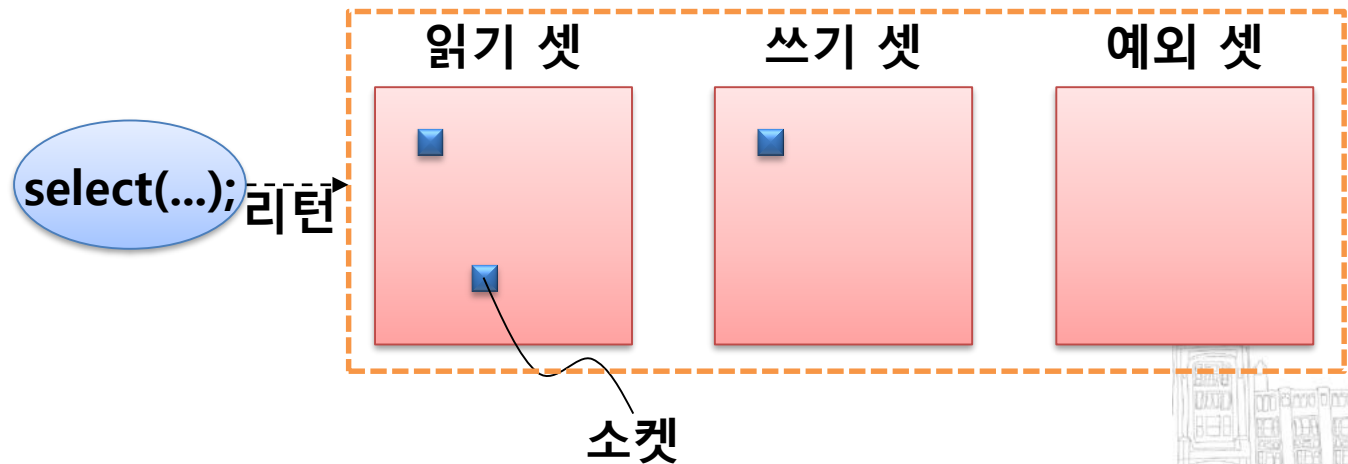
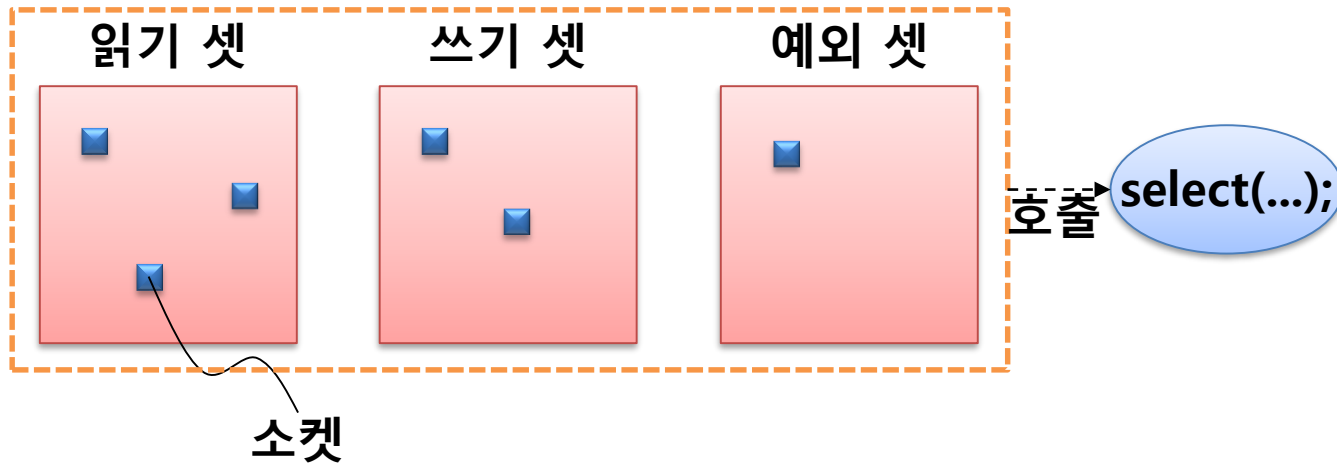
## ❖ 핵심 원리

- 소켓 함수 호출이 성공할 수 있는 시점을 미리 알 수 있어서 소켓 함수 호출 시 조건이 만족되지 않아 생기는 문제를 해결할 수 있음
  - 블로킹 소켓: 소켓 함수 호출 시 조건이 만족되지 않아 블로킹되는 상황을 방지
  - 년블로킹 소켓: 소켓 함수 호출 시 조건이 만족되지 않아 나중에 다시 호출해야 하는 상황을 방지



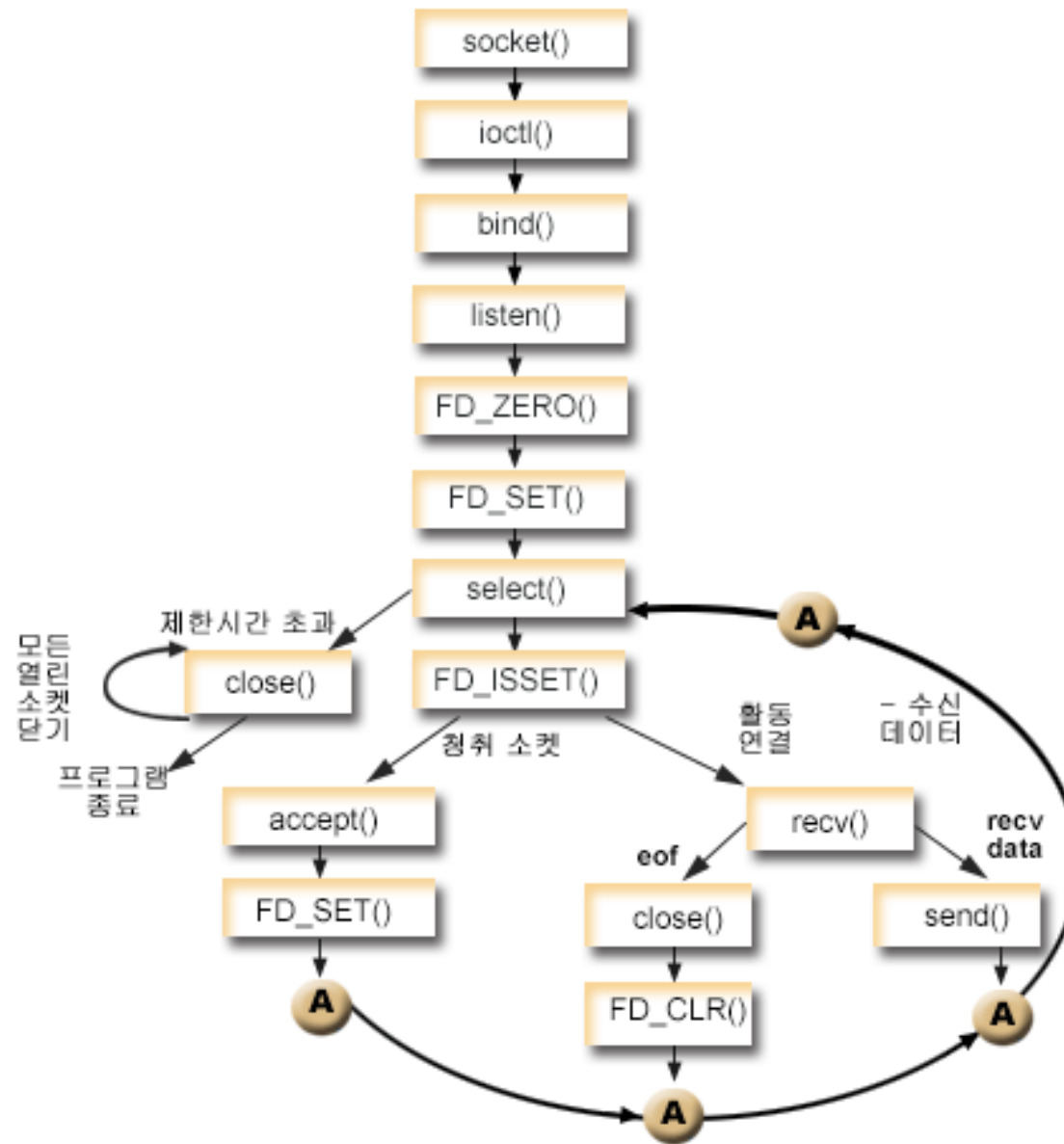
# Select 모델 (2)

## ❖ 동작 원리



# Select 모델 (2)

## ❖ 동작 원리



## ❖ 소켓 셋의 역할 - 각각 읽기, 쓰기, 예외 셋

|          |                                                                                                                                                                                                                                                  |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 함수 호출 시점 | <ul style="list-style-type: none"> <li>■ 접속한 클라이언트가 있으므로 accept() 함수를 호출할 수 있음</li> <li>■ 소켓 수신 버퍼에 도착한 데이터가 있으므로 recv(), recvfrom() 등의 함수를 호출해 데이터를 읽을 수 있음</li> <li>■ TCP 연결이 종료되었으므로 recv(), recvfrom() 등의 함수를 호출해 연결 종료를 감지할 수 있음</li> </ul> |
| 함수 호출 시점 | <ul style="list-style-type: none"> <li>■ 소켓 송신 버퍼의 여유 공간이 충분하므로 send(), sendto() 등의 함수를 호출하여 데이터를 보낼 수 있음</li> </ul>                                                                                                                             |
| 함수 호출 결과 | <ul style="list-style-type: none"> <li>■ non블로킹 소켓을 사용한 connect() 함수 호출이 성공함</li> </ul>                                                                                                                                                          |
| 함수 호출 시점 | <ul style="list-style-type: none"> <li>■ OOB(Out-Of-Band) 데이터가 도착했으므로 recv(), recvfrom() 등의 함수를 호출하여 OOB 데이터를 받을 수 있음</li> </ul>                                                                                                                 |
| 함수 호출 결과 | <ul style="list-style-type: none"> <li>■ non블로킹 소켓을 사용한 connect() 함수 호출이 실패함</li> </ul>                                                                                                                                                          |

## ❖ select() 함수

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

성공: 조건을 만족하는 소켓의 개수 또는 0(타임아웃),  
실패: **SOCKET\_ERROR**

- nfds: 유닉스/리눅스와의 호환성을 위해 존재하며 윈도우에서는 무시.
- readfds, writefds, exceptfds: 각각 읽기 셋, 쓰기 셋, 예외 셋을 나타냄. 사용하지 않으면 모두 NULL 값이 될 수 있음
- Timeout: 초(seconds)와 마이크로초(microseconds) 단위로 타임아웃을 지정. 이 시간이 지나면 select() 함수는 무조건 리턴. NULL이면 조건이 만족될때까지 무한히 대기.



## ❖ select() 함수를 이용한 소켓 입출력 절차

- ① 소켓 셋을 비움(초기화).
- ② 소켓 셋에 소켓을 넣음. 넣을 수 있는 소켓의 최대 개수는 FD\_SETSIZE(=64)로 정의되어 있음
- ③ select() 함수 호출. 타임아웃이 NULL이면 select() 함수는 조건을 만족하는 소켓이 있을 때까지 리턴하지 않음
- ④ select() 함수가 리턴하면 소켓 셋에 남아 있는 모든 소켓에 대해 적절한 소켓 함수를 호출하여 처리
- ⑤ ①~④를 반복



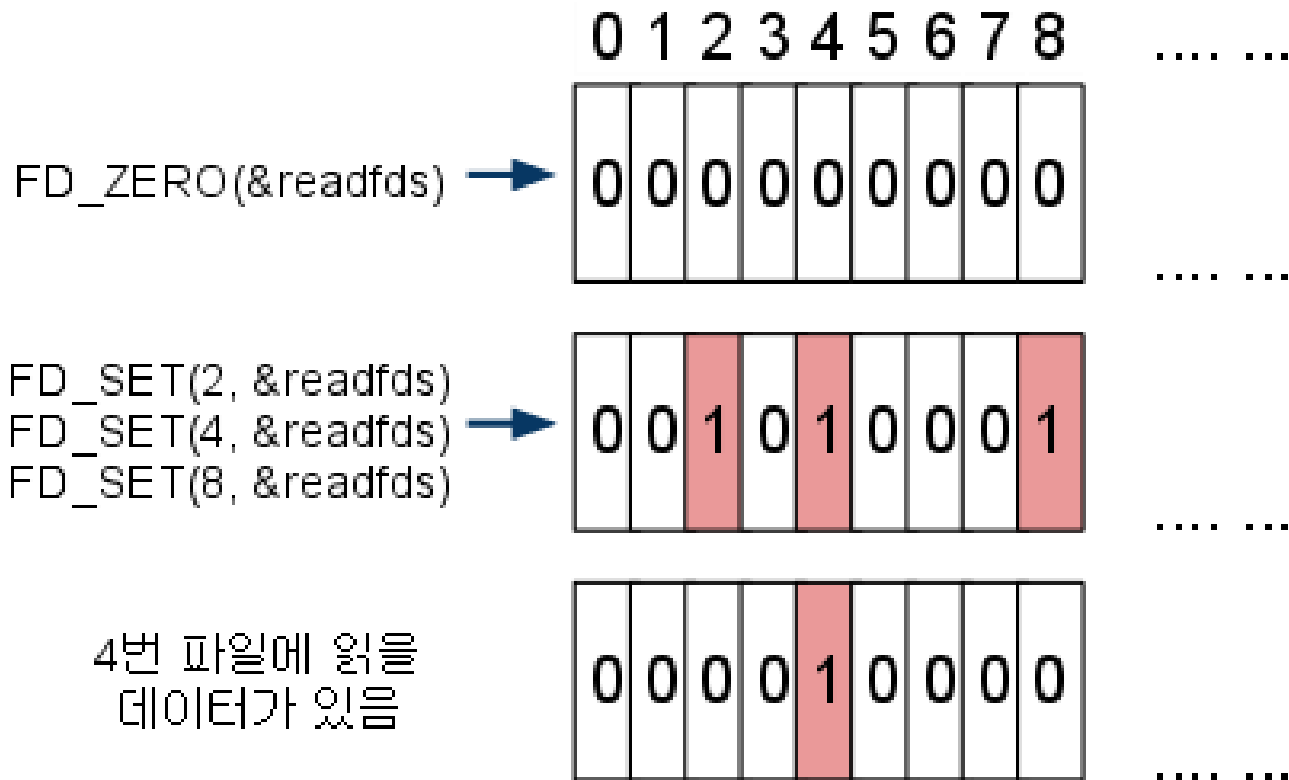


## ❖ 소켓 셋을 조작하는 매크로 함수

| 매크로 함수                          | 기능                                         |
|---------------------------------|--------------------------------------------|
| FD_ZERO(fd_set *set)            | 셋을 비움(초기화)                                 |
| FD_SET(SOCKET s, fd_set *set)   | 셋에 소켓 s를 넣음                                |
| FD_CLR(SOCKET s, fd_set *set)   | 셋에서 소켓 s를 제거                               |
| FD_ISSET(SOCKET s, fd_set *set) | 소켓 s가 셋에 들어 있으면 0이 아닌 값을 리턴. 그렇지 않으면 0을 리턴 |



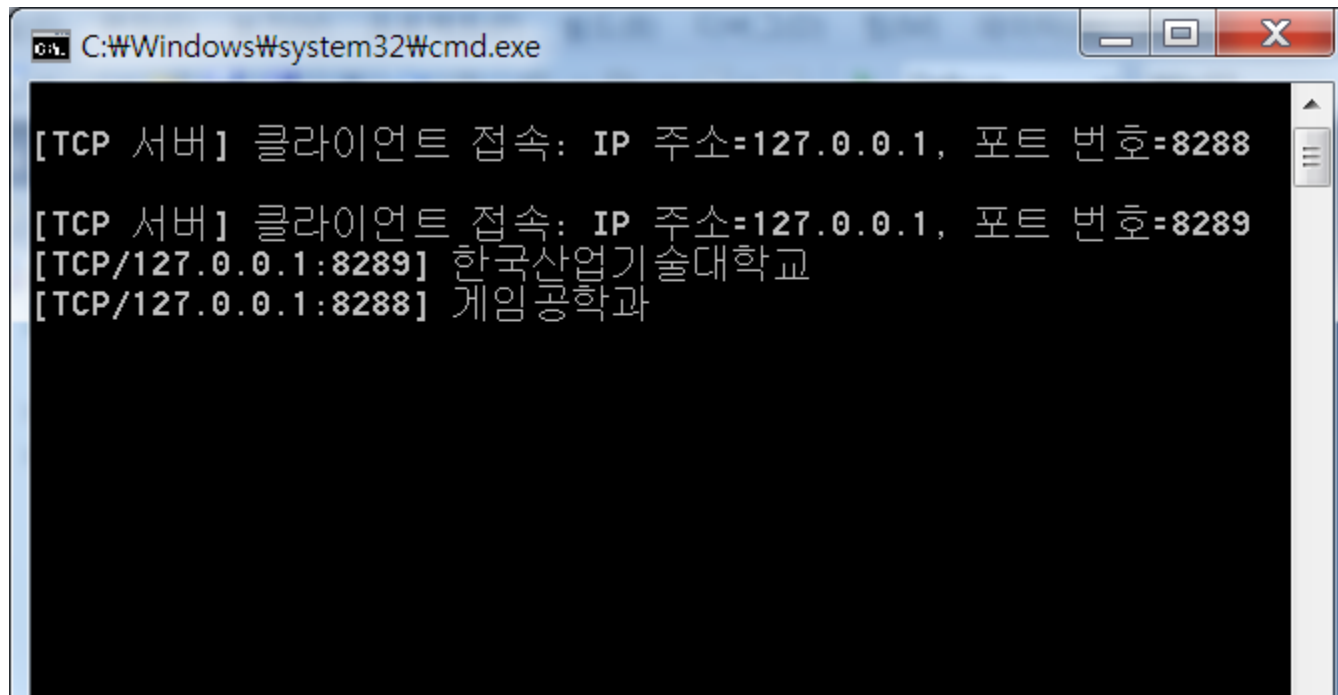
# Select 모델 (6)



1. FD\_ZERO로 readfds를 초기화
  2. 파일 지정 번호 2, 4, 8을 readfds에 추가한다. 대응되는 필드의 값을 1로 설정
  3. 4번 파일에 읽을 데이터가 있다면, 4번을 1로 채운다음 반환
- 다만, fd\_set이 이전 상태를 기억하지 못함. 그러므로 select함수를 호출하기 전에 이전 fd\_set의 값을 저장해 두어야 함. 매번 fd\_set 정보를 복사해야 한다는게 select함수의 단점임

## 실습 10-2 P371~

- 멀티스레드를 사용하지 않고도 여러 소켓을 처리할 수 있음.
- 년블록킹 소켓이지만 CPU 사용률이 낮음
- Select 모델은 여러 소켓에 대해 함수 호출 시점(혹은 호출 결과)을 알려주는 역할을 할 뿐 소켓 정보를 관리해주지는 않음. 따라서 각 소켓에 필요한 정보(응용 프로그램 버퍼, 송/수신 바이트 정보등)를 관리하는 기능은 응용 프로그램이 구현

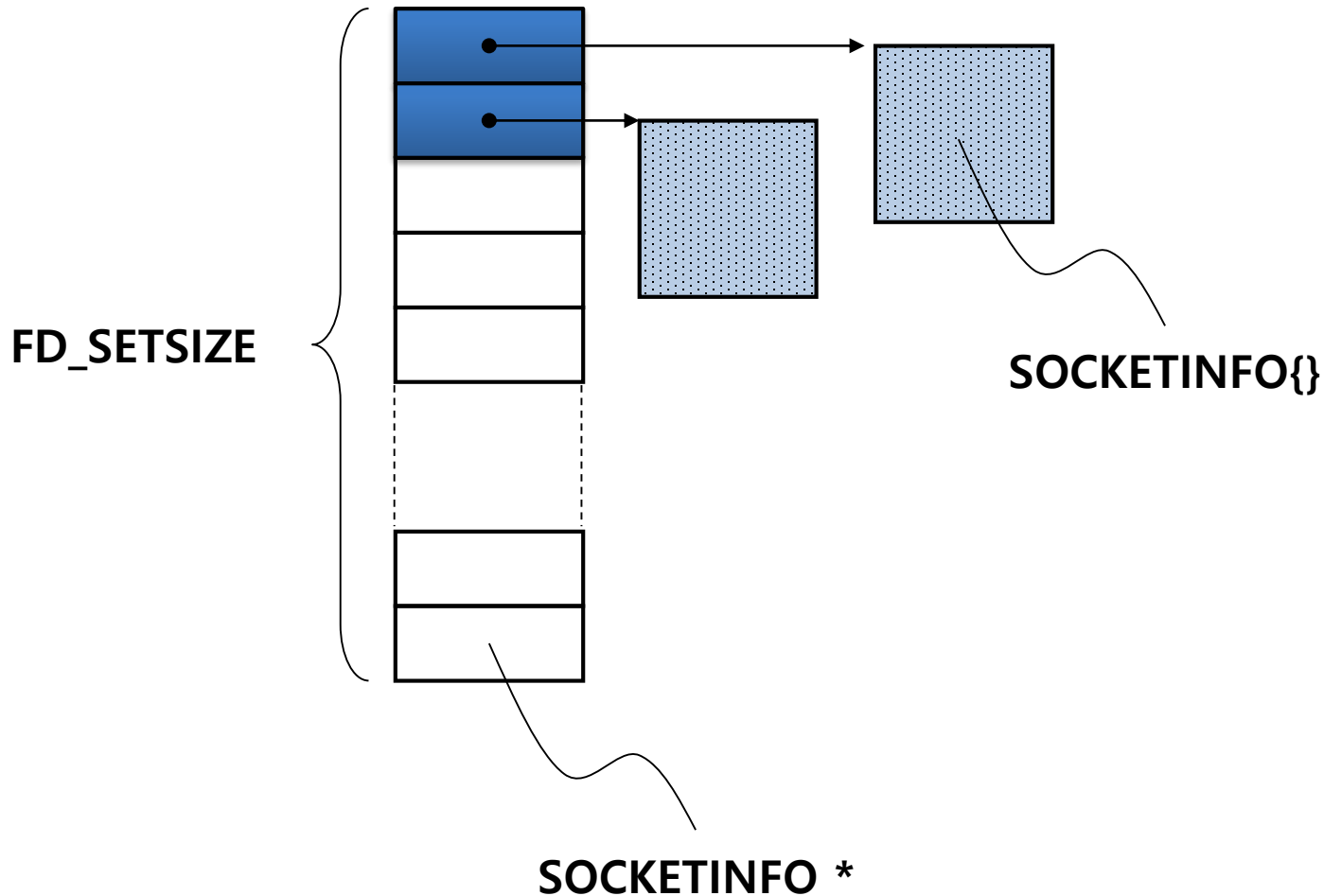


```
C:\Windows\system32\cmd.exe

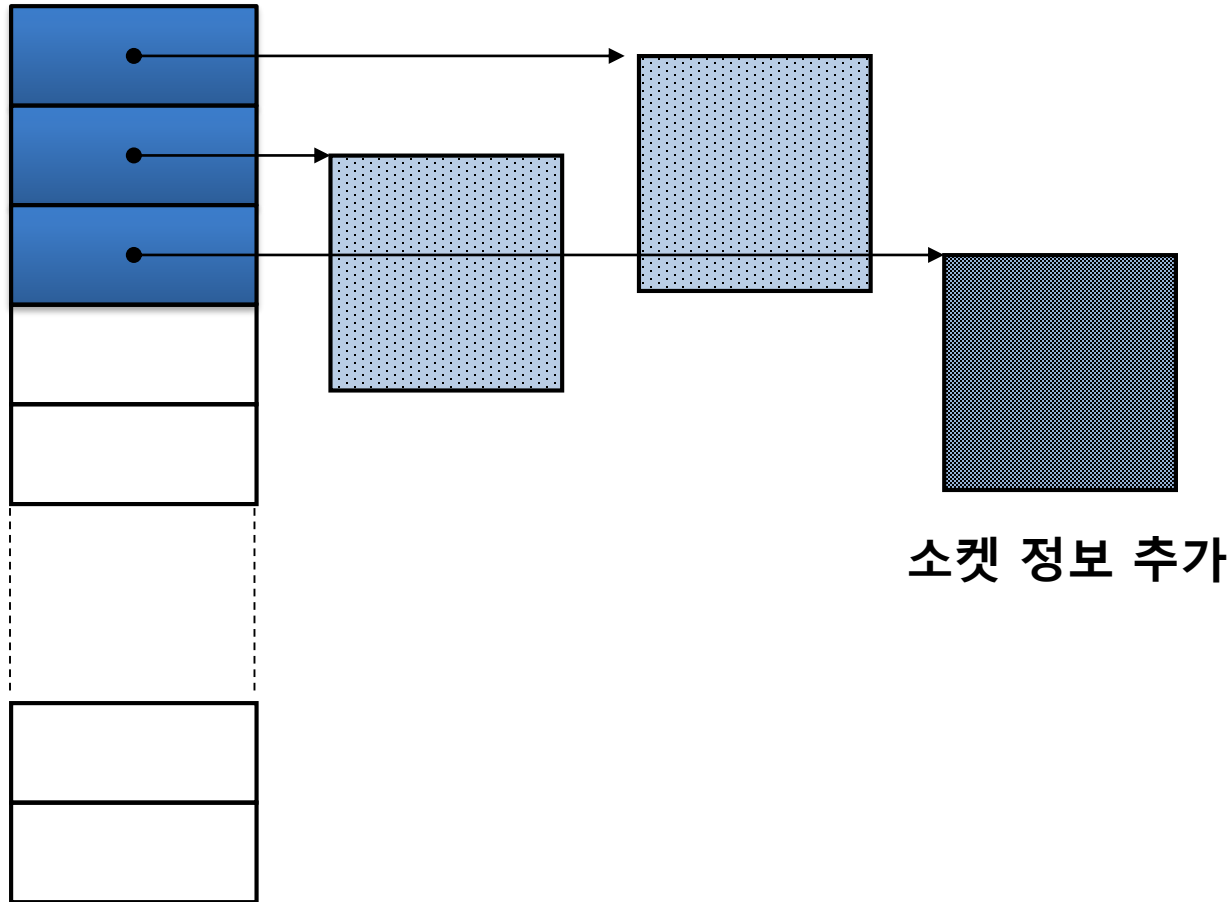
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=8288
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=8289
[TCP/127.0.0.1:8289] 한국산업기술대학교
[TCP/127.0.0.1:8288] 게임공학과
```



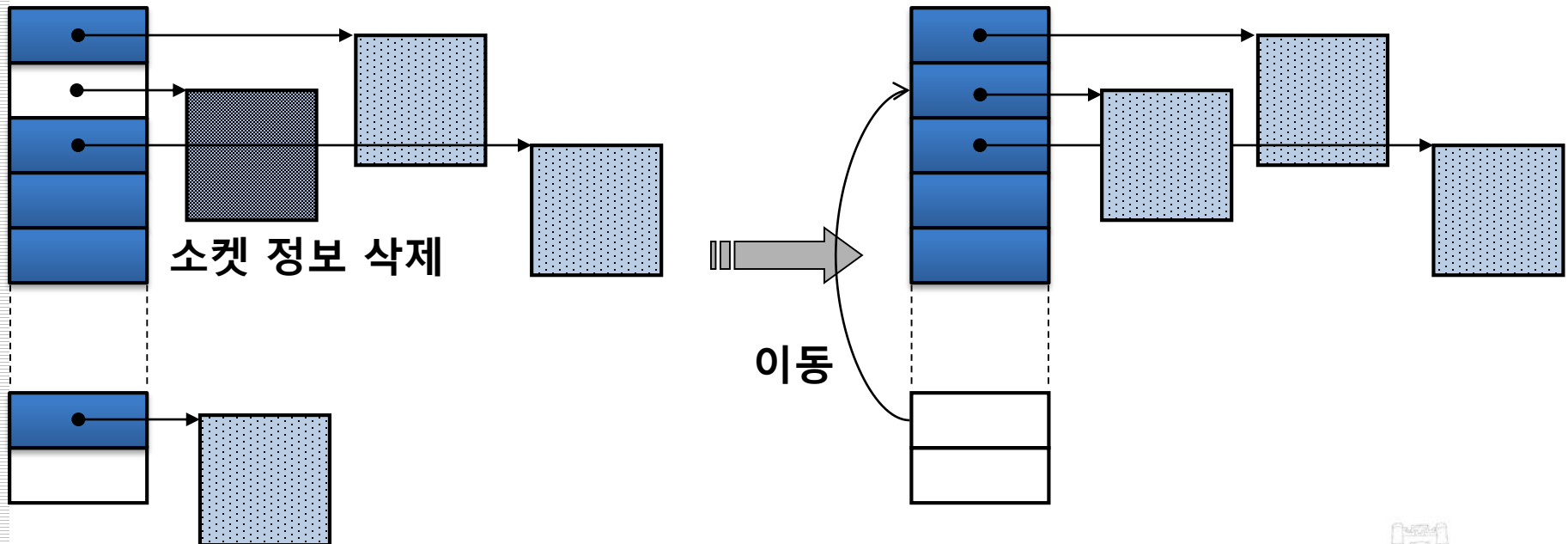
## ❖ 소켓 정보 관리를 위한 구조



## ❖ 소켓 정보 추가하기



## ❖ 소켓 정보 삭제하기



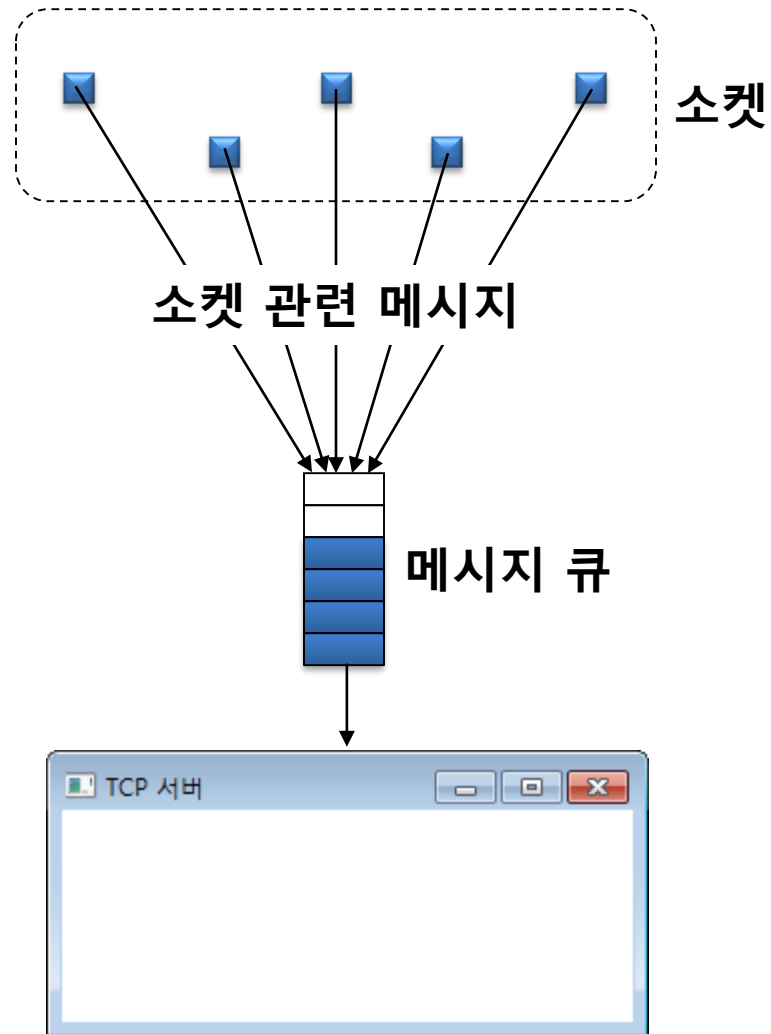
## ❖ WSAAsyncSelect 모델

- WSAAsyncSelect() 함수가 핵심적인 역할을 함
- 소켓과 관련된 네트워크 이벤트를 윈도우 메시지로 받음
  - 모든 소켓과 관련된 메시지가 한 윈도우, 즉 한 윈도우 프로시저에 전달되므로 멀티스레드를 사용하지 않고도 여러 소켓을 처리 가능
- 윈도우가 없는 콘솔 프로그램은 사용불가. 즉, GUI 어플리케이션에서 사용.



# WSAAsyncSelect 모델 (2)

## ❖ 동작 원리





- ❖ WSAAsyncSelect 모델을 사용하면 소켓 함수 호출이 성공할 수 있는 시점을 윈도우 메시지 수신으로 알 수 있음. 따라서 소켓 함수 호출 시 조건이 만족되지 않아 발생하는 문제를 해결할 수 있음.
- ❖ WSAAsyncSelect 모델을 이용한 소켓 입출력 절차
  - ① WSAAsyncSelect( ) 함수를 호출하여 소켓 이벤트를 알려줄 윈도우 메시지와 관심 있는 네트워크 이벤트를 등록
  - ② 등록된 네트워크 이벤트가 발생하면 윈도우 메시지가 발생하여 윈도우 프로시저가 호출됨
  - ③ 윈도우 프로시저에서는 받은 메시지의 종류에 따라 적절한 소켓 함수를 호출하여 처리



## ❖ WSAAsyncSelect() 함수

```
int WSAAsyncSelect (  
    SOCKET s,                // 네트워크 이벤트를 처리하고자 하는 소켓  
    HWND hWnd,               // 네트워크 이벤트가 발생하면 메시지를 받을 윈도우의 핸들  
    unsigned int wMsg,       // 네트워크 이벤트가 발생하면 윈도우가 받을 메시지  
    long lEvent              // 관심 있는 네트워크 이벤트를 비트 마스크 조합으로 나타냄  
);  
  
    성공: 0, 실패: SOCKET_ERROR
```

## ❖ 사용 예

```
#define WM_SOCKET (WM_USER+1)    // 사용자 정의 윈도우 메시지  
...  
WSAAsyncSelect(s, hWnd, WM_SOCKET, FD_READ|FD_WRITE);
```



## ❖ 네트워크 이벤트를 나타내는 상수

| 네트워크 이벤트   | 의미                 |
|------------|--------------------|
| FD_ACCEPT  | 접속한 클라이언트가 있다.     |
| FD_READ    | 데이터 수신이 가능하다.      |
| FD_WRITE   | 데이터 송신이 가능하다.      |
| FD_CLOSE   | 상대가 접속을 종료했다.      |
| FD_CONNECT | 통신을 위한 연결 절차가 끝났다. |
| FD_OOB     | OOB 데이터가 도착했다.     |



## ❖ 네트워크 이벤트와 대응 함수

- 윈도우 메시지를 받았을때 적절한 소켓 함수를 호출하지 않으면, 다음 번에는 같은 윈도우 메시지가 발생하지 않음. 따라서, 윈도우 메시지가 발생하면 대응 함수를 호출해야 하며, 그렇지 않을 경우 응용 프로그램이 나중에 직접 메시지를 발생시켜야 함.

| 네트워크 이벤트   | 의미                 |
|------------|--------------------|
| FD_ACCEPT  | accept()           |
| FD_READ    | recv(), recvfrom() |
| FD_WRITE   | send(), sendto()   |
| FD_CLOSE   | 없음                 |
| FD_CONNECT | 없음                 |
| FD_OOB     | recv(), recvfrom() |



## ❖ 윈도우 프로시저

- 네트워크 이벤트 발생 시 윈도우 프로시저에 전달되는 내용

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,  
    WPARAM wParam, LPARAM lParam)  
{  
    ....  
}
```

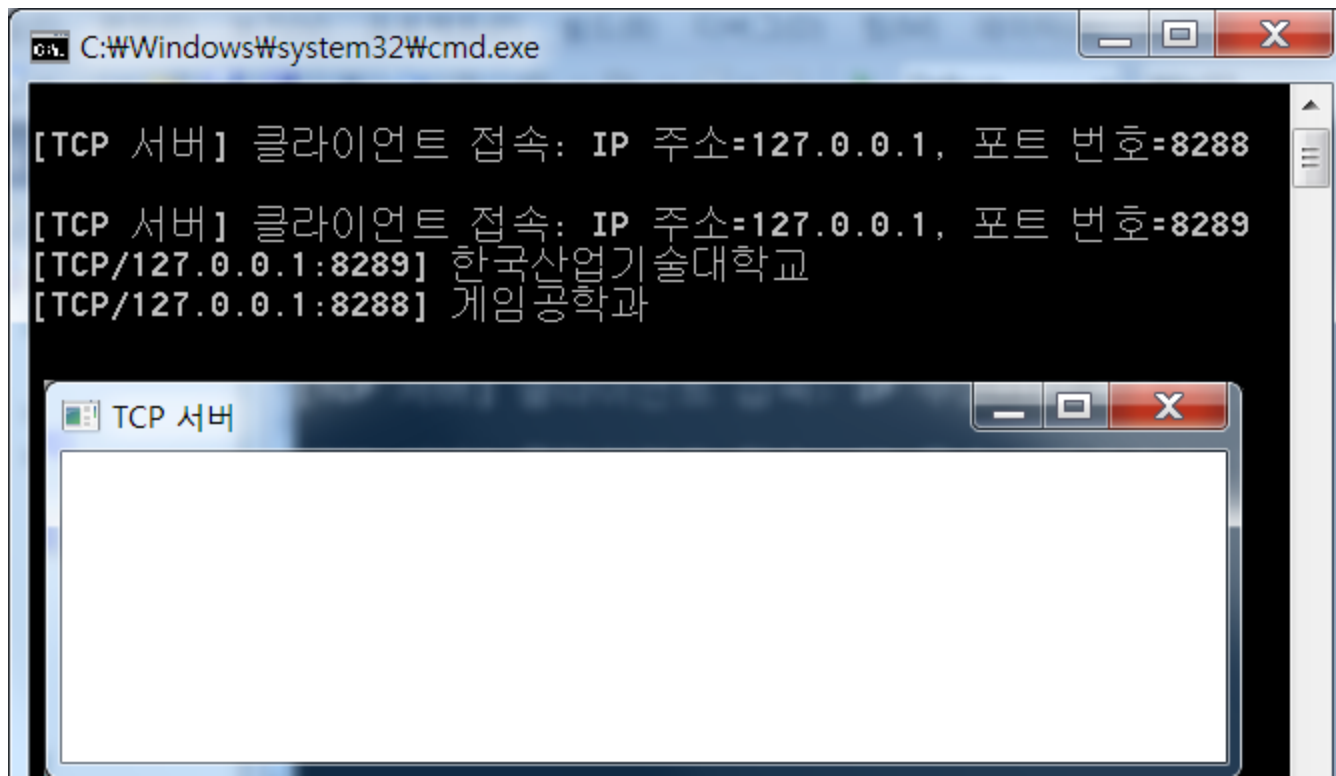
- hwnd : 메시지가 발생한 윈도우의 핸들
- msg : WSAAsyncSelect() 함수 호출 시 등록했던 사용자 정의 메시지
- wParam : 네트워크 이벤트가 발생한 소켓
- lParam : 하위 16비트는 발생한 네트워크 이벤트 상위 16비트는 오류 코드. 오류코드를 먼저 확인한 후 네트워크 이벤트를 처리해야 함.



## 실습 10-3

P386~

- 실행 결과는 SelectTCPServer() 예제와 같음.
- 차이점은 윈도우 기반의 소켓 프로그램임
- 스레드를 한 개만 사용하고 년블럭킹 소켓을 사용.
- CPU 사용률이 낮음.



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe" with the following output:

```
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=8288  
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=8289  
[TCP/127.0.0.1:8289] 한국산업기술대학교  
[TCP/127.0.0.1:8288] 게임공학과
```

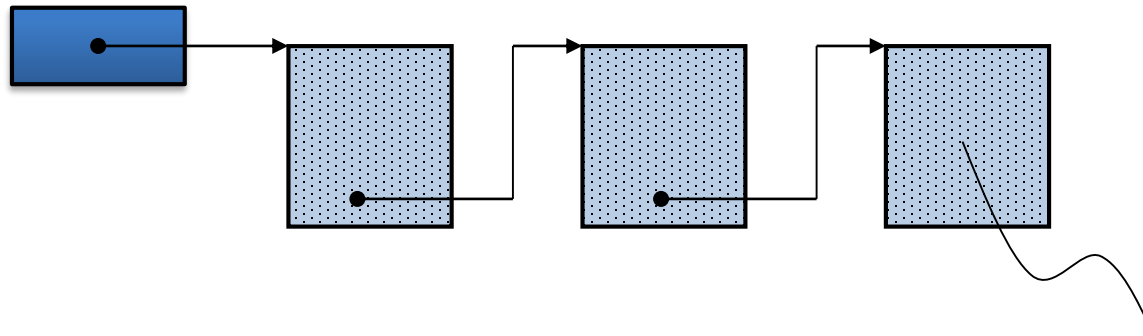
Below the command prompt, there is a smaller window titled "TCP 서버" which is currently empty.



## ❖ 소켓 정보 관리를 위한 구조

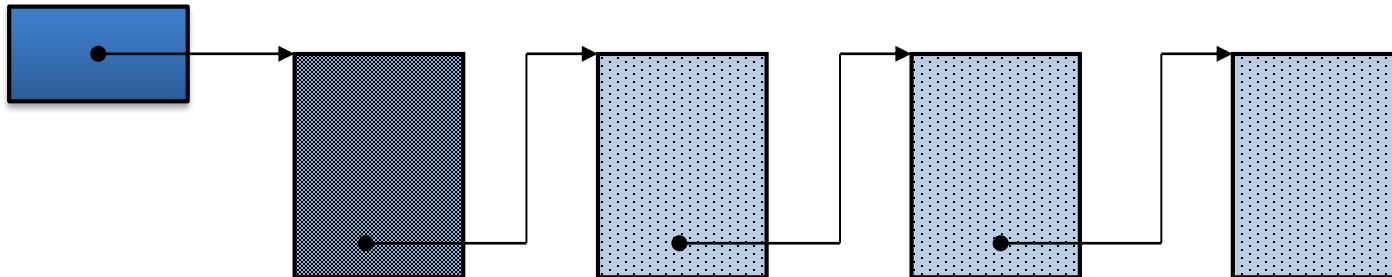
- SocketInfoList 변수: 연결 리스트의 시작점을 나타내는 SOCKETINFO형 포인터

SocketInfoList



## ❖ 소켓 정보 추가하기

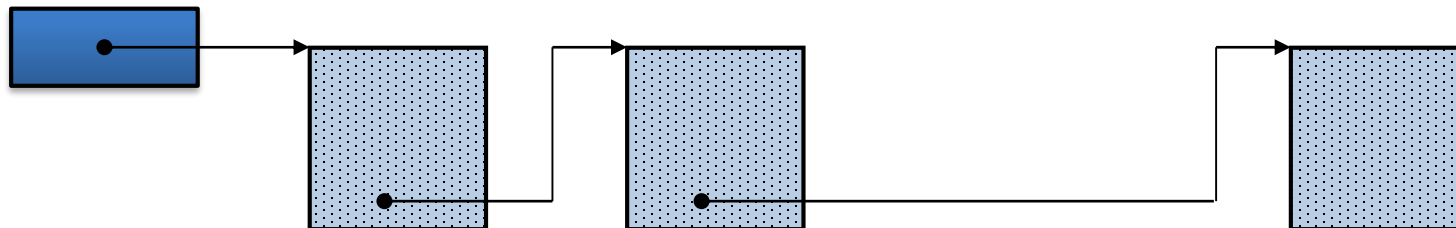
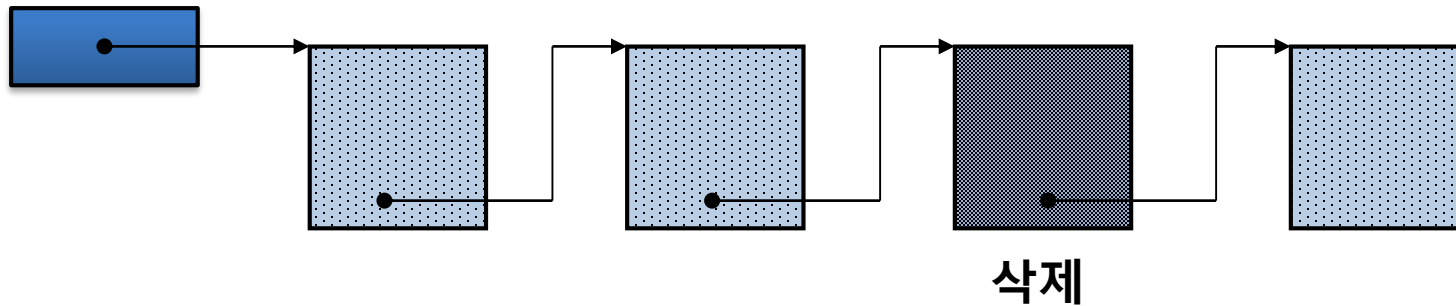
SOCKETINFO{}



추가



## ❖ 소켓 정보 삭제하기



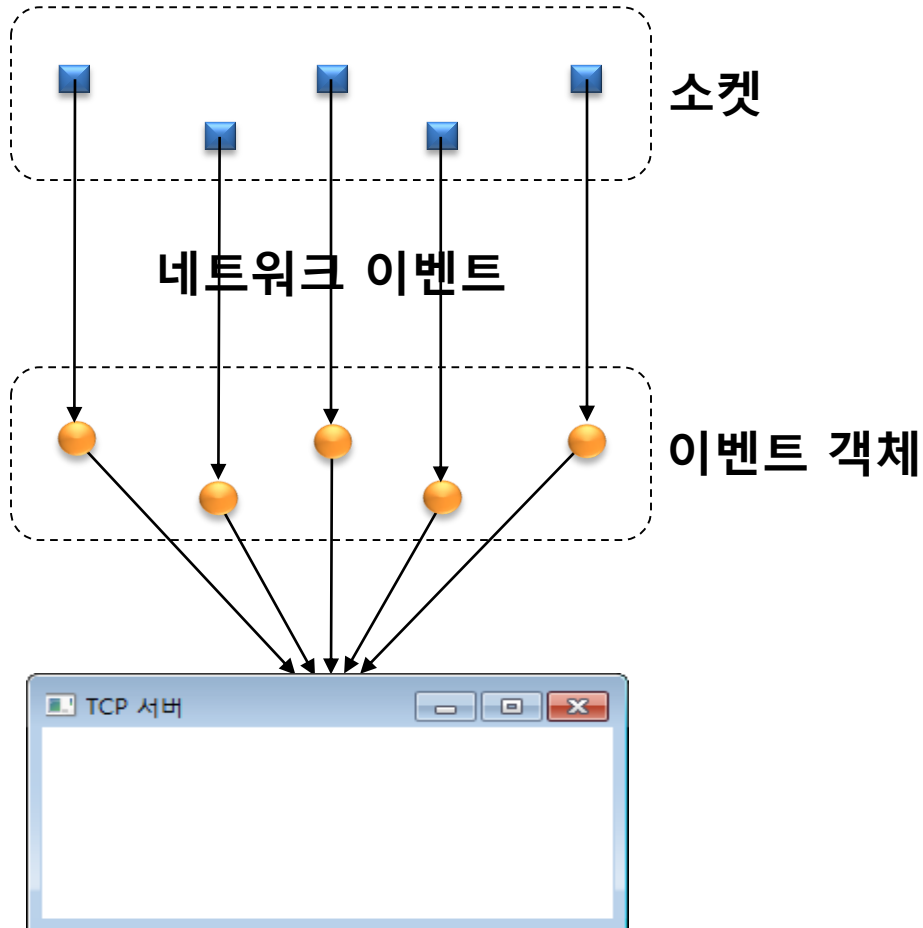


## ❖ WSAEventSelect 모델

- WSAEventSelect() 함수가 핵심적인 역할을 함
- 소켓 함수 호출이 성공할 수 있는 시점을 이벤트 객체를 통해 감지
  - 소켓에 대해 이벤트 객체를 생성하여 짝지어두면 네트워크 이벤트가 발생할 때마다 이벤트 객체가 신호 상태가 됨
  - 이벤트 객체의 상태 변화를 관찰함으로써 네트워크 이벤트 발생을 감지



## ❖ 동작 원리



## ❖ WSAEventSelect 모델의 필요 기능과 관련 함수

- 이벤트 객체 생성과 제거
  - WSACreateEvent(), WSACloseEvent()
- 소켓과 이벤트 객체 짝짓기
  - WSAEventSelect()
- 이벤트 객체의 신호 상태 감지하기
  - WSAWaitForMultipleEvents()
- 구체적인 네트워크 이벤트 알아내기
  - WSAEnumNetworkEvents()



## ❖ WSAEventSelect 모델을 이용한 소켓 입출력 절차

- ① 소켓을 생성할 때마다 WSACreateEvent() 함수를 이용해 이벤트 객체를 생성
- ② WSAEventSelect() 함수를 이용해 소켓과 이벤트 객체를 짝지음과 동시에 처리할 네트워크 이벤트를 등록
- ③ WSAWaitForMultipleEvents() 함수를 호출해 이벤트 객체가 신호 상태가 되기를 기다림. 등록된 네트워크 이벤트가 발생하면 해당 소켓과 연관된 이벤트 객체가 신호 상태가 됨
- ④ WSAEnumNetworkEvents() 함수를 호출해 발생한 네트워크 이벤트를 알아내고 적절한 소켓 함수를 호출해 처리



## ❖ 이벤트 객체 생성과 제거하기

```
WSAEVENT WSACreateEvent ( ) ;  
성공: 이벤트 객체 핸들, 실패: WSA_INVALID_EVENT
```

```
BOOL WSACloseEvent (WSAEVENT hEvent) ;  
성공: TRUE, 실패: FALSE
```



## ❖ 소켓과 이벤트 객체 짝짓기

```
int WSAEventSelect (  
    SOCKET s,  
    WSAEVENT hEventObject,  
    long lNetworkEvents  
);
```

성공: 0, 실패: SOCKET\_ERROR

- s: 네트워크 이벤트를 처리하고자 하는 소켓
- hEventObject: 소켓과 연관시킬 이벤트 객체의 핸들
- lNetworkEvents: 관심 있는 네트워크 이벤트를 비트 마스크 조합으로 나타냄



## ❖ 네트워크 이벤트를 나타내는 상수

| 네트워크 이벤트   | 의미                 |
|------------|--------------------|
| FD_ACCEPT  | 접속한 클라이언트가 있다.     |
| FD_READ    | 데이터 수신이 가능하다.      |
| FD_WRITE   | 데이터 송신이 가능하다.      |
| FD_CLOSE   | 상대가 접속을 종료했다.      |
| FD_CONNECT | 통신을 위한 연결 절차가 끝났다. |
| FD_OOB     | OOB 데이터가 도착했다.     |



## ❖ 이벤트 객체의 신호 상태 감지하기

```
DWORD WSAWaitForMultipleEvents (  
  DWORD cEvents,  
  const WSAEVENT *lphEvents,  
  BOOL fWaitAll,  
  DWORD dwTimeout,  
  BOOL fAlertable  
);
```

**성공: WSA\_WAIT\_EVENT\_0 ~ WSA\_WAIT\_EVENT\_0+cEvents-1**  
 **또는 WSA\_WAIT\_TIMEOUT**  
**실패: WSA\_WAIT\_FAILED**

- cEvents, lphEvents: 배열 원소 개수, 배열의 시작 주소
- fWaitAll: TRUE면 모든 이벤트 객체가 신호 상태가 될 때까지 대기. FALSE면 이벤트 객체 중 하나가 신호 상태가 되는 즉시 리턴
- dwTimeout: 대기시간으로 밀리초 단위를 사용. INFINITE면 무한히 기다림.
- fAlertable: 입출력 완료 루틴과 관련된 부분. WSAEvent 항상 FALSE





## ❖ 구체적인 네트워크 이벤트 알아내기

```
int WSAEnumNetworkEvents (  
    SOCKET s,  
    WSAEVENT hEventObject,  
    LPWSANETWORKEVENTS lpNetworkEvents  
);
```

성공: 0, 실패: SOCKET\_ERROR

- s: 대상 소켓
- hEventObject: 대상 소켓s와 짝지어둔 이벤트 객체 핸들을 넘겨주면 이벤트 객체가 자동으로 비신호 상태로 됨. 선택사항이므로 사용하지 않으려면 NULL 값을 넘겨주면 됨
- lpNetworkEvents: WSANETWORKEVENTS 구조체 변수의 주소 값을 전달하면, 발생한 네트워크 이벤트와 오류 정보가 이 변수에 저장



## ❖ 예제 코드

- WSAEnumNetworkEvents() 함수 사용 예

```
SOCKET s;  
WSAEVENT hEvent;  
WSANETWORKEVENTS NetworkEvents;  
...  
WSAEnumNetworkEvents(s, hEvent, &NetworkEvents);  
// FD_ACCEPT 이벤트 처리  
if(NetworkEvents.lNetworkEvents & FD_ACCEPT){  
    if(NetworkEvents.iErrorCode[FD_ACCEPT_BIT] != 0){  
        printf("오류 코드 = %d\n",  
            NetworkEvents.iErrorCode[FD_ACCEPT_BIT]);  
    }  
    else{  
        // accept() 함수 호출  
    }  
}
```

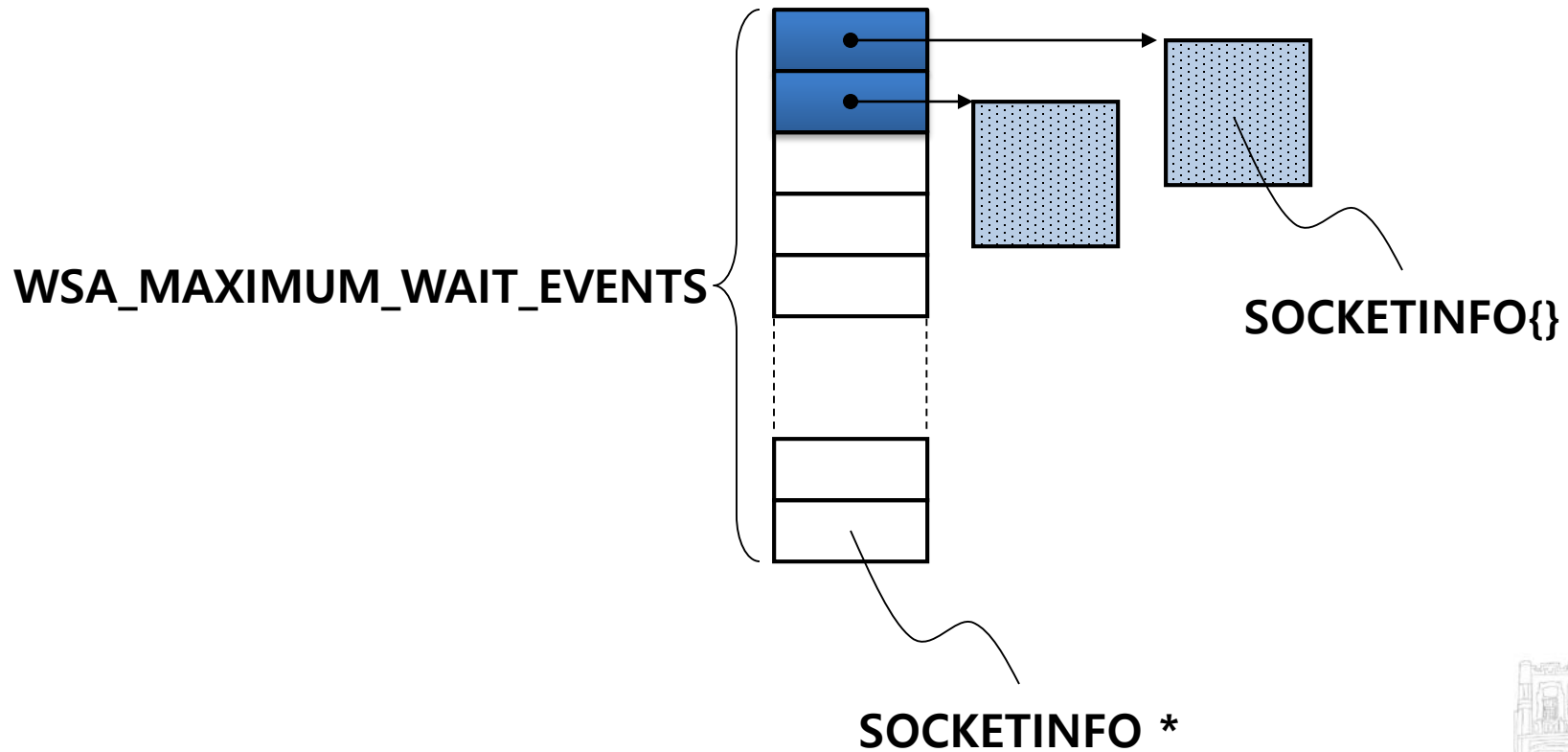


## ❖ 예제 코드(계속)

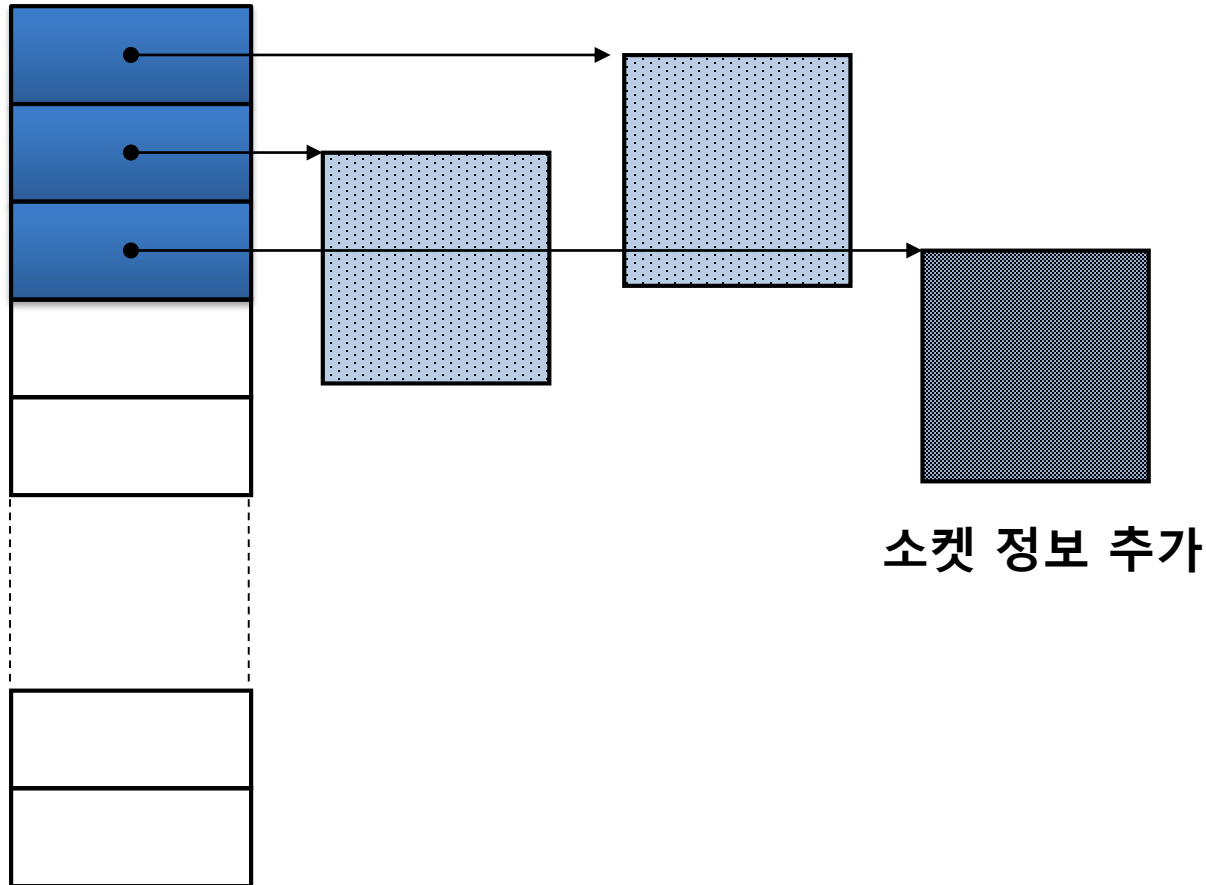
```
// FD_READ 이벤트 처리
if(NetworkEvents.lNetworkEvents & FD_READ){
    if(NetworkEvents.iErrorCode[FD_READ_BIT] != 0){
        printf("오류 코드 = %d\n",
            NetworkEvents.iErrorCode[FD_READ_BIT]);
    }
    else{
        // recv() 함수 호출
    }
}
```



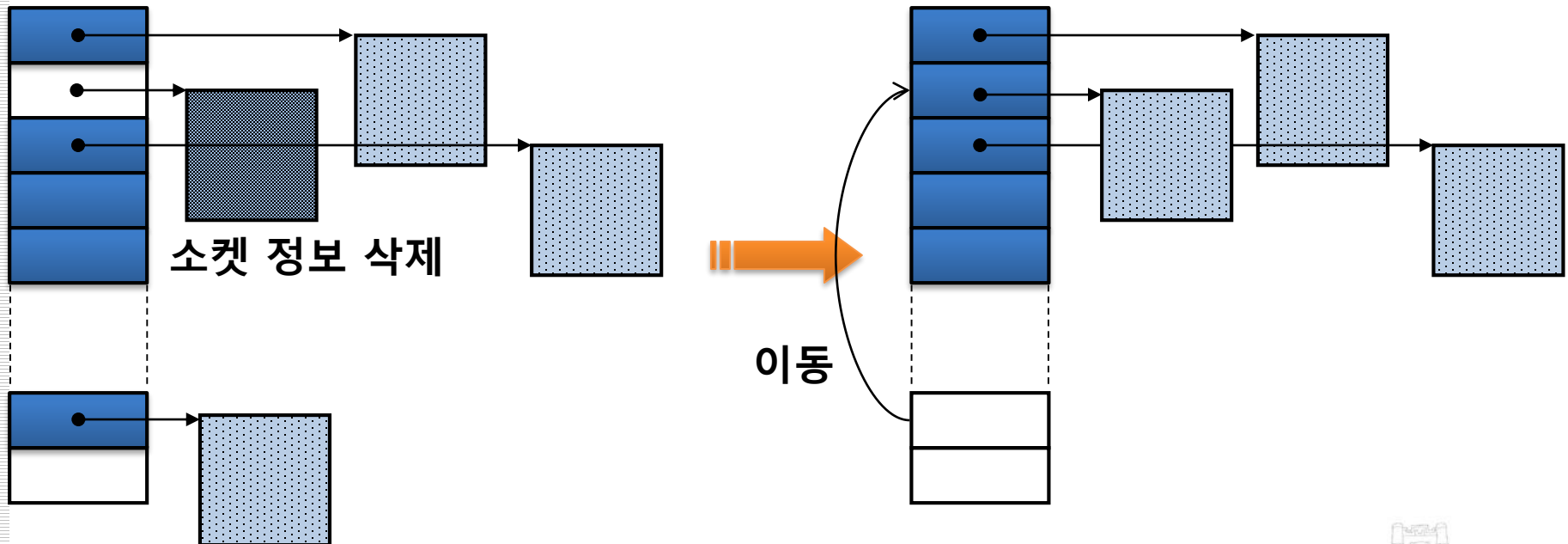
## ❖ 소켓 정보 관리를 위한 구조



## ❖ 소켓 정보 추가하기

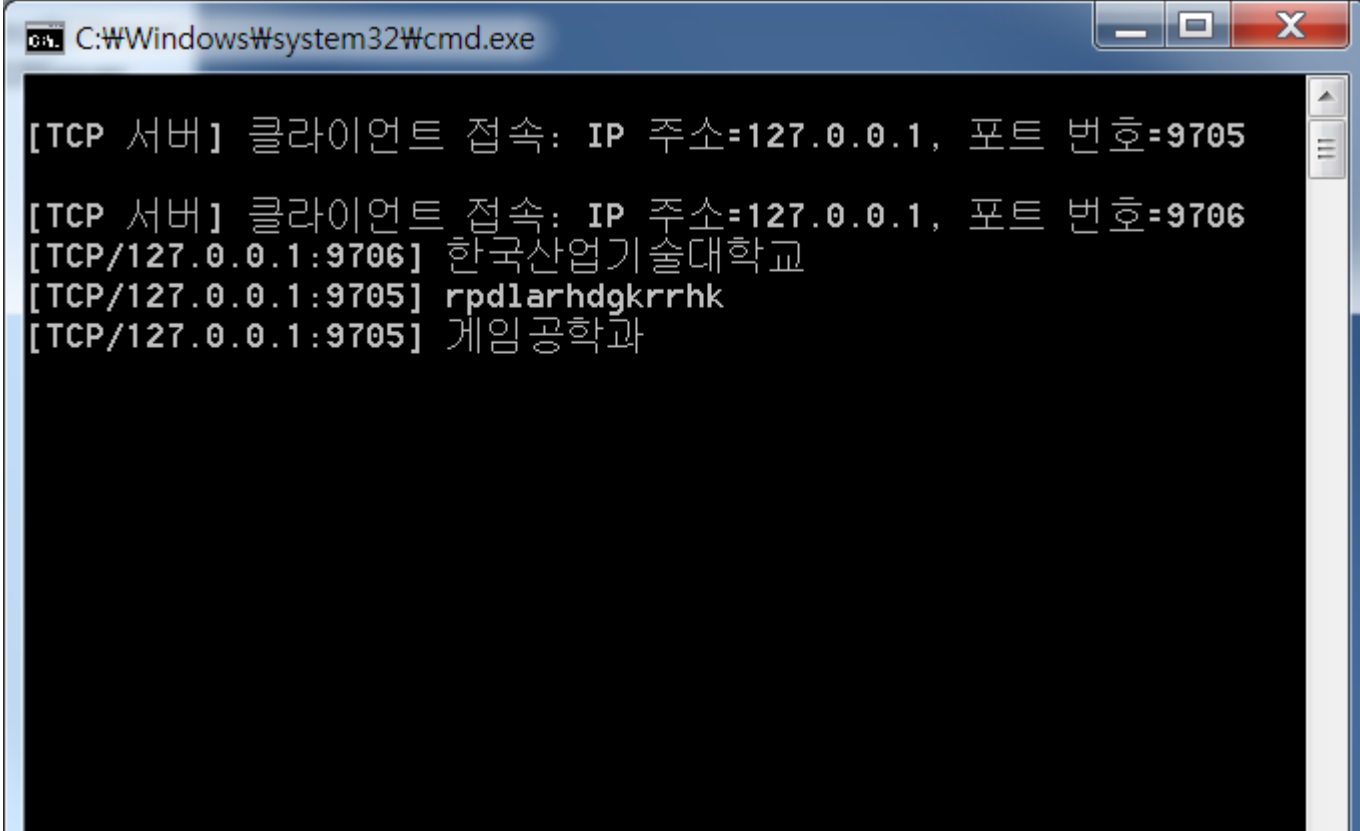


## ❖ 소켓 정보 삭제하기



## 실습 10-4 P408~

실행 결과는 기존 예제와 같음.  
스레드를 한 개만 사용하고 년블럭킹 소켓을 사용.  
CPU 사용률이 낮음.



```
C:\Windows\system32\cmd.exe

[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=9705
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=9706
[TCP/127.0.0.1:9706] 한국산업기술대학교
[TCP/127.0.0.1:9705] rpd1arhdgkrrhk
[TCP/127.0.0.1:9705] 게임공학과
```



# Thank You !

[oasis01@gmail.com](mailto:oasis01@gmail.com) / [rhqudtn75@nate.com](mailto:rhqudtn75@nate.com)