

Chapter 11 소켓 입출력 모델: 윈도우

학습목표

- 블로킹과 논블로킹 소켓의 특징을 이해한다.
- Select 소켓 입출력 모델을 이해하고 활용한다.
- WSAAsyncSelect 소켓 입출력 모델을 이해하고 활용한다.
- Completion Port 소켓 입출력 모델을 이해하고 활용한다.

IOCP

목차

- 01 소켓 입출력 모델 개요
- 02 Select 모델
- 03 WSAAsyncSelect 모델
- 04 Completion Port 모델
- 05 소켓 입출력 모델 비교

01 소켓 입출력 모델 개요



소켓 입출력 모델 개요

■ 소켓 입출력 모델(Socket I/O model)이란?

- 다수의 소켓을 관리하고 소켓에 대한 입출력을 처리하는 방식
- 소켓 입출력 모델을 사용하면 프로그래밍 복잡도는 높아지지만, 시스템 자원을 적게 사용하면서도 다수의 클라이언트를 효율적으로 처리하는 서버를 만들 수 있음

소켓 모드의 종류 (1)

■ 소켓 모드

- 블로킹과 논블로킹 소켓으로 구분

■ 블로킹 소켓

- 소켓 함수 호출 시 조건을 만족하지 않으면 함수가 리턴하지 않고 스레드 실행이 정지
- 조건을 만족하면 소켓 함수가 리턴하고 정지된 스레드가 실행을 재개

표 11-1 주요 소켓 함수와 리턴 조건

소켓 함수	리턴 조건
accept()	TCP 소켓 접속한 클라이언트가 있을 때
connect()	TCP 소켓 서버에 접속이 성공했을 때
send(), sendto()	TCP 소켓, UDP 소켓 응용 프로그램이 전송을 요청한 데이터를 소켓 송신 버퍼에 모두 복사했을 때
recv(), recvfrom()	TCP 소켓 소켓 수신 버퍼에 도착한 데이터가 1바이트 이상 있어서 이를 응용 프로그램이 제공한 버퍼에 복사했을 때 UDP 소켓 UDP 패킷이 완전히 도착해서 소켓 수신 버퍼의 UDP 데이터를 응용 프로그램이 제공한 버퍼에 복사했을 때

소켓 모드의 종류 (2)

■ 년블로킹 소켓

- 소켓 함수 호출 시 조건을 만족하지 않더라도 함수가 리턴하므로 스레드가 중단 없이 다음 코드를 수행
- socket() 함수는 기본적으로 블로킹 소켓을 생성함
- 년블로킹 소켓이 필요하다면 ioctlsocket() 함수를 호출하여 소켓 모드를 변경해야 함

```
// 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()");
```

IO 컨트롤 소켓

flag ↓

← true 값

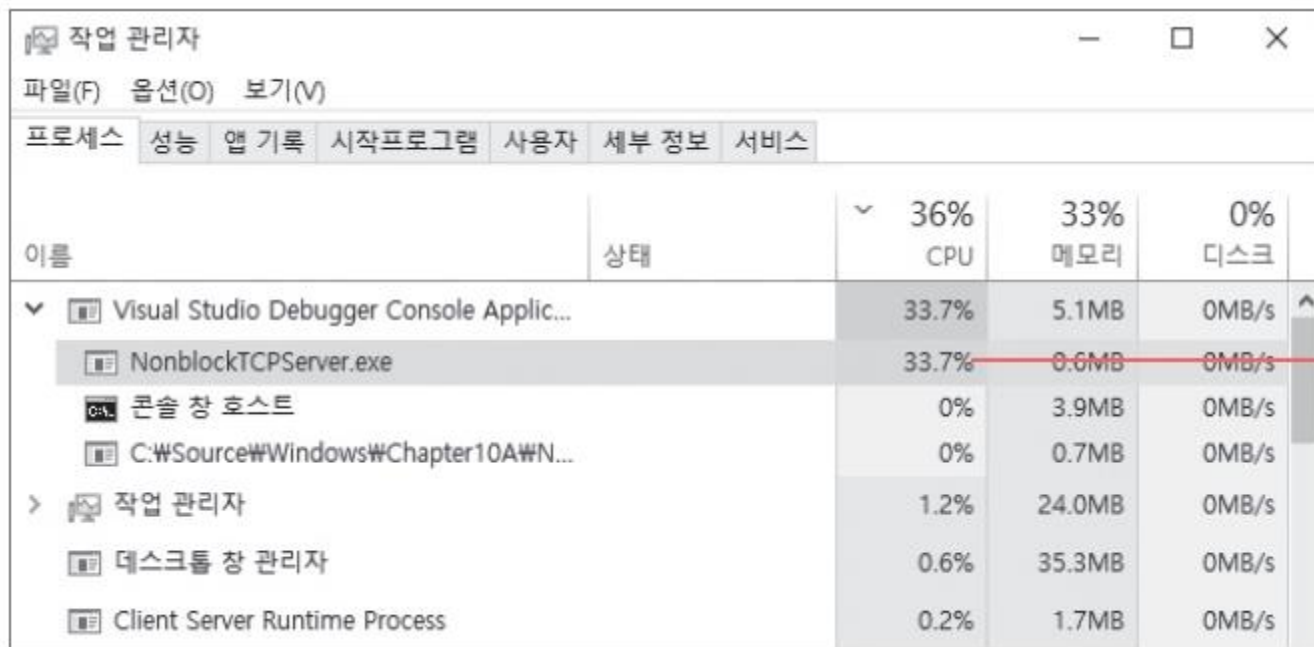
소켓 모드의 종류 (3)

■ 년블로킹 소켓과 소켓 함수

- 년블로킹 소켓에 대해 소켓 함수 호출 시 조건을 만족하지 않으면 소켓 함수는 오류를 리턴
- WSAGetLastError() 함수를 이용하여 오류 코드를 확인
- 대개 오류 코드는 `WSAEWOULDBLOCK`
 - 조건을 만족하지 않았음을 나타내므로 나중에 다시 소켓 함수를 호출하면 됨

소켓 모드의 종류 (4)

■ 실습 11-1 년블로킹 소켓 연습



이름		상태	36% CPU	33% 메모리	0% 디스크
Visual Studio Debugger Console Applic...			33.7%	5.1MB	0MB/s
NonblockTCPServer.exe			33.7%	0.6MB	0MB/s
콘솔 창 호스트			0%	3.9MB	0MB/s
C:\Source\Windows\Chapter10A\N...			0%	0.7MB	0MB/s
작업 관리자			1.2%	24.0MB	0MB/s
데스크톱 창 관리자			0.6%	35.3MB	0MB/s
Client Server Runtime Process			0.2%	1.7MB	0MB/s

NonblockTCPServer의
CPU 사용률

소켓 모드의 종류 (5)

■ 실습 11-1 년블로킹 소켓 연습

- NonblockTCPServer.cpp
- <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter11/NonblockTCPServer/NonblockTCPServer.cpp>

소켓 모드의 종류 (6)

■ 년블로킹 소켓의 특징

표 11-2 년블로킹 소켓의 특징

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

서버 작성 모델의 종류

■ 반복 서버

- 여러 클라이언트를 한 번에 하나씩 처리

표 11-3 반복 서버의 특징

장점	한 개의 스레드로 구현하므로 시스템 자원 소모가 적다.
단점	한 클라이언트의 처리 시간이 길어지면 다른 클라이언트의 대기 시간이 길어진다.

■ 병행 서버

- 여러 클라이언트를 동시에 처리

표 11-4 병행 서버의 특징

장점	한 클라이언트의 처리 시간이 길어지더라도 다른 클라이언트의 대기 시간에 영향을 주지 않는다.
단점	스레드를 여러 개 생성하여 구현하므로 시스템 자원 소모가 크다.

이상적인 소켓 입출력 모델의 특징

■ 서버가 제공해야 할 이상적인 기능

- 가능한 많은 클라이언트가 접속할 수 있음
- 서버는 각 클라이언트의 서비스 요청에 빠르게 반응하며 고속으로 데이터를 전송
- 시스템 자원 사용량(CPU 사용률, 메모리 사용량 등)을 최소화

■ 소켓 입출력 모델에 요구되는 사항

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

소켓 입출력 모델의 종류

■ 소켓 입출력 모델 지원

표 11-5 윈도우 운영체제의 소켓 입출력 모델 지원

소켓 입출력 모델	운영체제 버전	
	윈도우(클라이언트 버전)	윈도우(서버 버전)
Select	윈도우 95 이상	윈도우 NT 이상
WSAAsyncSelect	윈도우 95 이상	윈도우 NT 이상
WSAEventSelect	윈도우 95 이상	윈도우 NT 3.51 이상
Overlapped	윈도우 95 이상	윈도우 NT 3.51 이상
Completion Port	윈도우 NT 3.5 이상(윈도우 95/98/Me 제외)	

02 Select 모델



Select 모델의 동작 원리 (1)

■ Select 모델

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

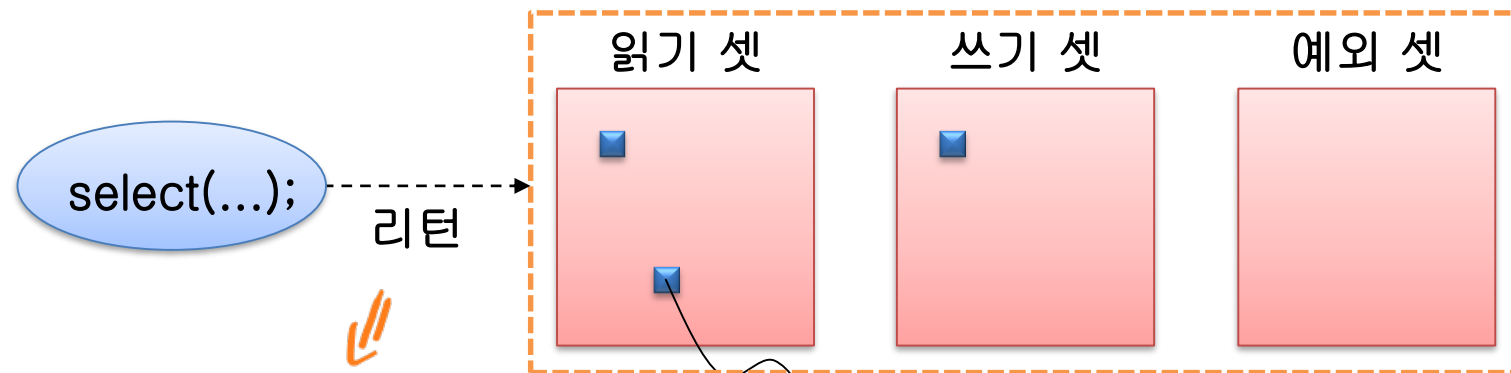
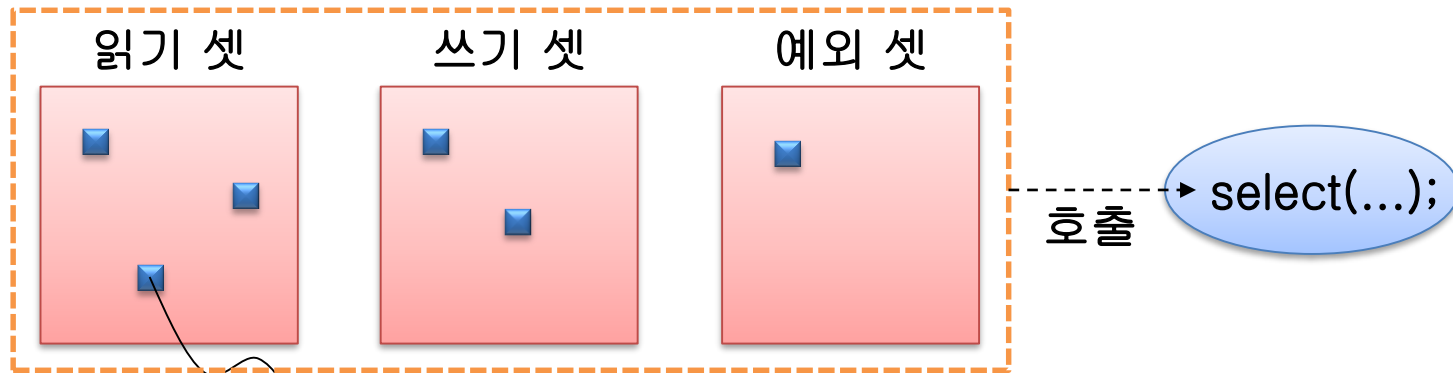
■ 핵심 원리

응답자의 응답을 받아
↓

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

Select 모델의 동작 원리 (2)

■ Select 모델의 동작 원리



리턴 가능한 소켓들만 반환한다.

Select 모델의 동작 원리 (3)

■ 소켓 셋의 역할

표 11-6 소켓 셋의 역할

읽기 셋 read set	
함수 호출 시점	<ul style="list-style-type: none">• 접속한 클라이언트가 있으므로 <u>accept()</u> 함수를 호출할 수 있다.• 소켓 수신 버퍼에 도착한 데이터가 있으므로 <u>recv()</u>, <u>recvfrom()</u> 등의 함수를 호출하여 데이터를 읽을 수 있다.• TCP 연결을 종료했으므로 <u>recv()</u>, <u>recvfrom()</u> 등의 함수를 호출하여 리턴값(0)으로 연결 종료를 알 수 있다.
쓰기 셋 write set	
함수 호출 시점	<ul style="list-style-type: none">• 소켓 송신 버퍼의 여유 공간이 충분하므로 <u>send()</u>, <u>sendto()</u> 등의 함수를 호출하여 데이터를 보낼 수 있다.
함수 호출 결과	<ul style="list-style-type: none">• <u>non-blocking</u> 소켓을 사용한 <u>connect()</u> 함수 호출이 성공했다.
예외 셋 exception set	
함수 호출 시점	<ul style="list-style-type: none">• OOB^{Out-Of-Band} 데이터가 도착했으므로 <u>recv()</u>, <u>recvfrom()</u> 등의 함수를 호출하여 OOB 데이터를 받을 수 있다.
함수 호출 결과	<ul style="list-style-type: none">• <u>non-blocking</u> 소켓을 사용한 <u>connect()</u> 함수 호출이 실패했다.

Select 모델의 동작 원리 (4)

■ select() 함수

```
#include <winsock2.h>
int select(
    ① int nfds,
    ② fd_set *readfds,
    fd_set *writefds,
    fd_set *exceptfds,
    ③ const struct timeval *timeout
);
```



성공: 조건을 만족하는 소켓의 개수 또는 0(타임아웃), 실패: SOCKET_ERROR

■ timeval 구조체

```
struct timeval {
    long tv_sec; /* seconds */
    long tv_usec; /* microseconds */
};
```

Select 모델의 동작 원리 (5)

■ 타임아웃값에 따른 select() 함수의 동작

■ NULL

- 적어도 한 소켓이 조건을 만족할 때까지 무한히 기다림
- 리턴값은 조건을 만족하는 소켓의 개수가 됨

■ {0, 0}

- 소켓 셋에 포함된 모든 소켓을 검사한 후 곧바로 리턴
- 리턴값은 조건을 만족하는 소켓의 개수(0도 가능)가 됨

■ 양수

- 적어도 한 소켓이 조건을 만족하거나 타임아웃으로 지정한 시간이 지나면 리턴
- 리턴값은 조건을 만족하는 소켓의 개수 또는 0(타임아웃)이 됨

Select 모델의 동작 원리 (6)

■ Select 모델을 이용한 소켓 입출력 절차

- ① 소켓 셋을 비움(초기화)
- ② 소켓 셋에 소켓을 넣음. 넣을 수 있는 소켓의 최대 개수는 FD_SETSIZE(64)로 정의
→ 64개 넣으면 쓰레드 동작은 생략
- ③ select() 함수를 호출하여 소켓 이벤트를 기다림. 타임아웃이 NULL이면 select() 함수는 조건을 만족하는 소켓이 하나라도 있을 때까지 리턴하지 않음
- ④ select() 함수가 리턴하면 소켓 셋에 남아 있는 모든 소켓에 대해 적절한 소켓 함수를 호출하여 처리
- ⑤ ①~④를 반복

Select 모델의 동작 원리 (7)

■ 소켓 셋을 조작하는 매크로 함수

표 11-7 소켓 셋을 조작하는 매크로 함수

매크로 함수	기능
FD_ZERO(fd_set *set)	셋을 비운다(초기화).
FD_SET(SOCKET s, fd_set *set)	소켓 s를 셋에 넣는다.
FD_CLR(SOCKET s, fd_set *set)	소켓 s를 셋에서 제거한다.
FD_ISSET(SOCKET s, fd_set *set)	소켓 s가 셋에 들어 있으면 0이 아닌 값을 리턴한다. 그렇지 않으면 0을 리턴한다.

field

Select 모델 서버 작성 (1)

■ 실습 11-2 Select 모델 TCP 서버 작성과 테스트



이름		상태	9% CPU	30% 메모리	0% 디스크
Visual Studio Debugger Console Applic...			0%	1.3MB	0MB/s
C:\Source\Windows\Chapter10\AWS...			0%	0.6MB	0MB/s
SelectTCPServer.exe			0%	0.6MB	0MB/s
원격 데스크톱 연결			0.1%	63.1MB	0MB/s
작업 관리자			0.8%	22.0MB	0MB/s

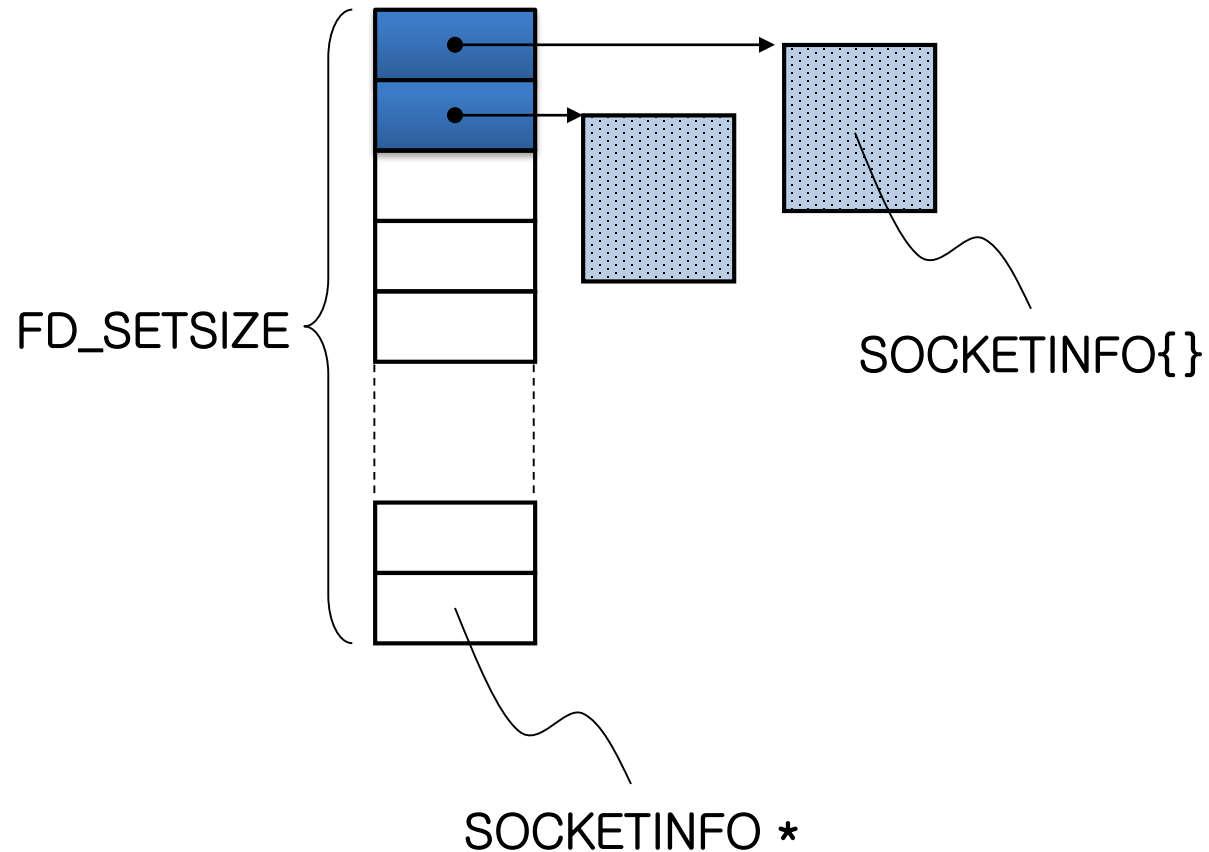
SelectTCPServer의
CPU 사용률

Select 모델 서버 작성 (2)

- 실습 11-2 Select 모델 TCP 서버 작성과 테스트
 - SelectTCPServer.cpp
 - <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter11/SelectTCPServer/SelectTCPServer.cpp>

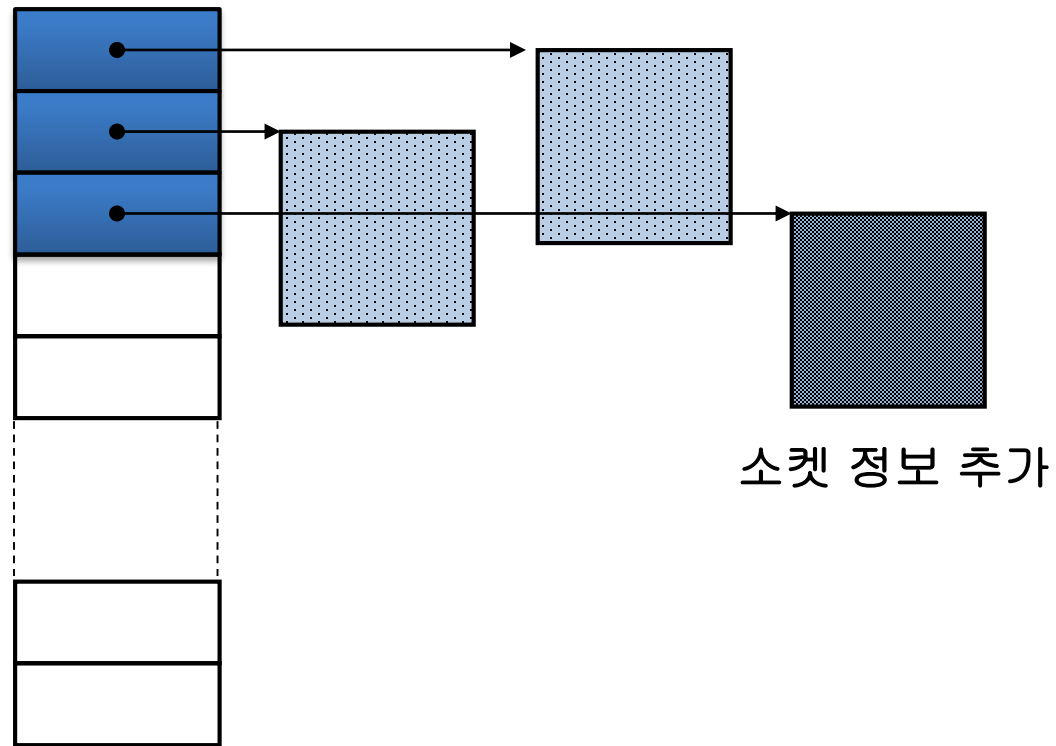
SelectTCPServer 예제의 소켓 정보 관리 (1)

■ 소켓 정보 관리를 위한 구조



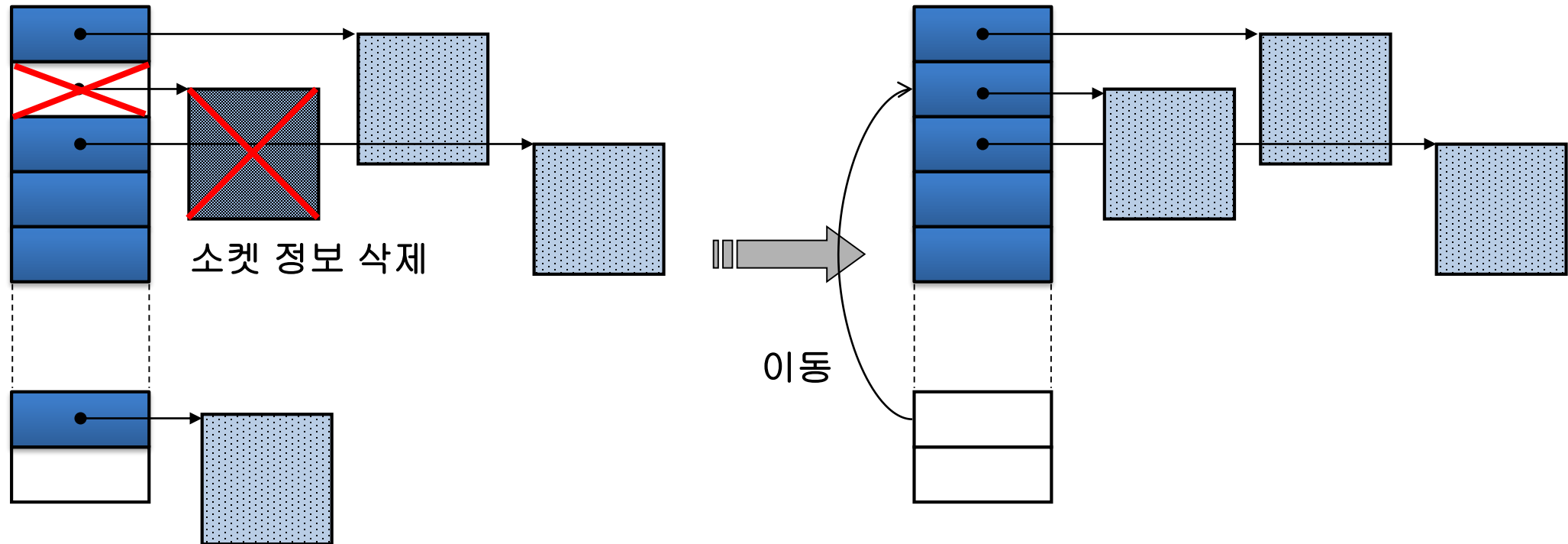
SelectTCPServer 예제의 소켓 정보 관리 (2)

■ 소켓 정보 추가하기



SelectTCPServer 예제의 소켓 정보 관리 (3)

■ 소켓 정보 삭제하기



03 WSAAsyncSelect 모델

GUI에서 적용하기 쉬운 것

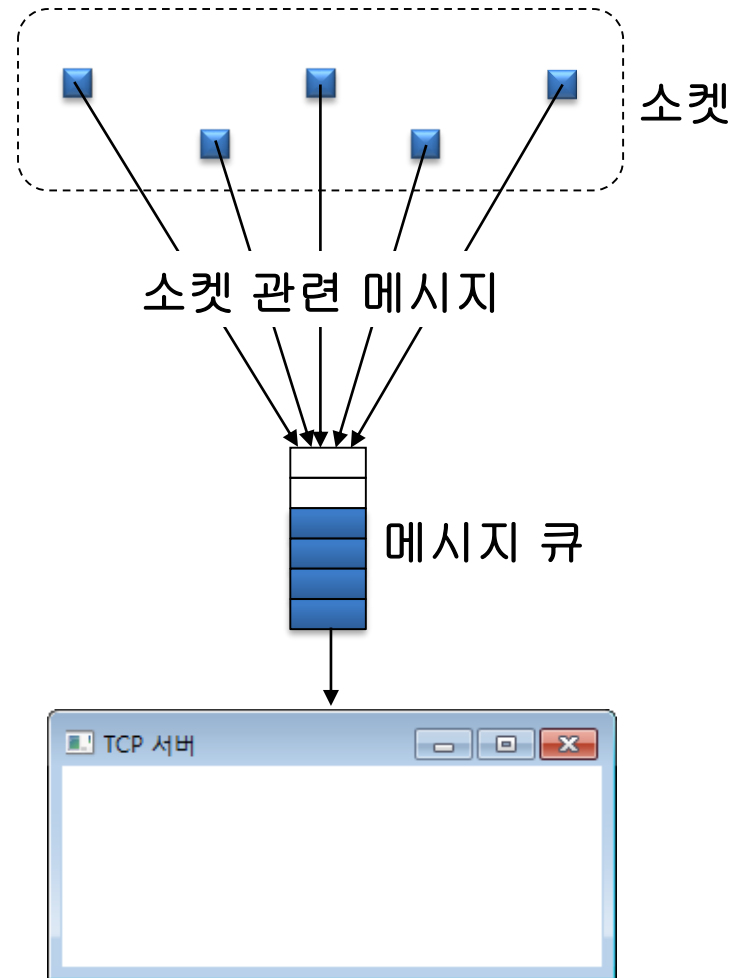
↳ 콘스 플랫폼에 적용하기 쉽다.



WSAAsyncSelect 모델

■ WSAAsyncSelect 모델

- WSAAsyncSelect 모델은 소켓과 관련된 네트워크 이벤트를 윈도우 메시지 형태로 받음



WSAAsyncSelect 모델의 동작 원리 (1)

■ WSAAsyncSelect 모델을 이용한 소켓 입출력 절차

- ❶ WSAAsyncSelect() 함수를 호출하여 소켓 이벤트를 알려줄 윈도우 메시지와 함께 관심있는 네트워크 이벤트를 등록
- ❷ 등록된 네트워크 이벤트가 발생하면, 윈도우 메시지가 발생해 윈도우 프로시저가 호출
- ❸ 윈도우 프로시저는 받은 메시지의 종류에 따라 적절한 소켓 함수를 호출하여 처리

■ WSAAsyncSelect() 함수

```
#include <winsock2.h>
int WSAAsyncSelect(
    ❶ SOCKET sock,
    ❷ HWND hWnd,
    ❸ unsigned int wMsg,
    ❹ long lEvent
);
```

socket 에 대한 통보를 윈도우 메시지로 받게 하는 의미

← 해당 윈도우에게
메시지를 보낸다.

메시지를 hWnd인 윈도우를 쿼백하여 wMsg 와

| Event 이 따라 처리해준다.

← 네트워크 이벤트 (조건)을 만족할 때

성공: 0, 실패: SOCKET_ERROR

WSAAsyncSelect 모델의 동작 원리 (2)

- 관심 있는 네트워크 이벤트를 비트 마스크([표 11-8] 조합으로 나타냄)

표 11-8 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.

- 소켓 s에 대해 FD_READ와 FD_WRITE 이벤트를 등록하는 예

```
#define WM_SOCKET (WM_USER+1) // 사용자 정의 윈도우 메시지
```

```
...
```

```
WSAAsyncSelect(s, hWnd, WM_SOCKET, FD_READ|FD_WRITE);
```

여기 hWnd의 윈도우 프로시저에서
처리해 주면 된다.

→ 여기서 선택받은 이벤트

WSAAsyncSelect 모델의 동작 원리 (3)

■ WSAAsyncSelect() 함수 사용 시 유의 사항

- WSAAsyncSelect() 함수를 호출하면 해당 소켓은 자동으로 넌블로킹 모드로 전환
- accept() 함수가 리턴하는 소켓은 연결 대기 소켓과 같은 속성을 가짐
- 윈도우 메시지에 대응해서 소켓 함수를 호출하면 대부분 성공하지만, WSAEWOULDBLOCK 오류 코드가 발생하는 경우가 드물게 있음
- 윈도우 메시지를 받았을 때 적절한 소켓 함수를 호출하지 않으면, 다음에는 같은 윈도우 메시지가 발생하지 않음
→ 윈프록에서 반드시 처리 해주라

WSAAsyncSelect 모델의 동작 원리 (4)

■ 네트워크 이벤트와 대응 함수

표 11-9 네트워크 이벤트와 대응 함수

네트워크 이벤트	대응 함수
FD_ACCEPT	accept()
FD_READ	recv(), recvfrom()
FD_WRITE	send(), sendto()
FD_CLOSE	없음
FD_CONNECT	없음
FD_OOB	recv(), recvfrom()

■ 네트워크 이벤트 발생 시 윈도우 프로시저에 전달되는 내용

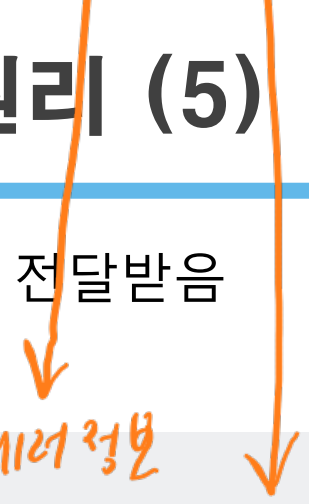
```
LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    ...
}
```

4byte ('상위 2, 하위 2')

WSAAsyncSelect 모델의 동작 원리 (5)

- 윈도우 프로시저는 총 네 개의 인수를 통해 데이터를 전달받음
 - hWnd, uMsg, wParam, lParam

```
#define WSAGETSELECTERROR(lParam) HIWORD(lParam) → 에러 정보
#define WSAGETSELECTEVENT(lParam) LOWORD(lParam) → 네트워크 이벤트
```



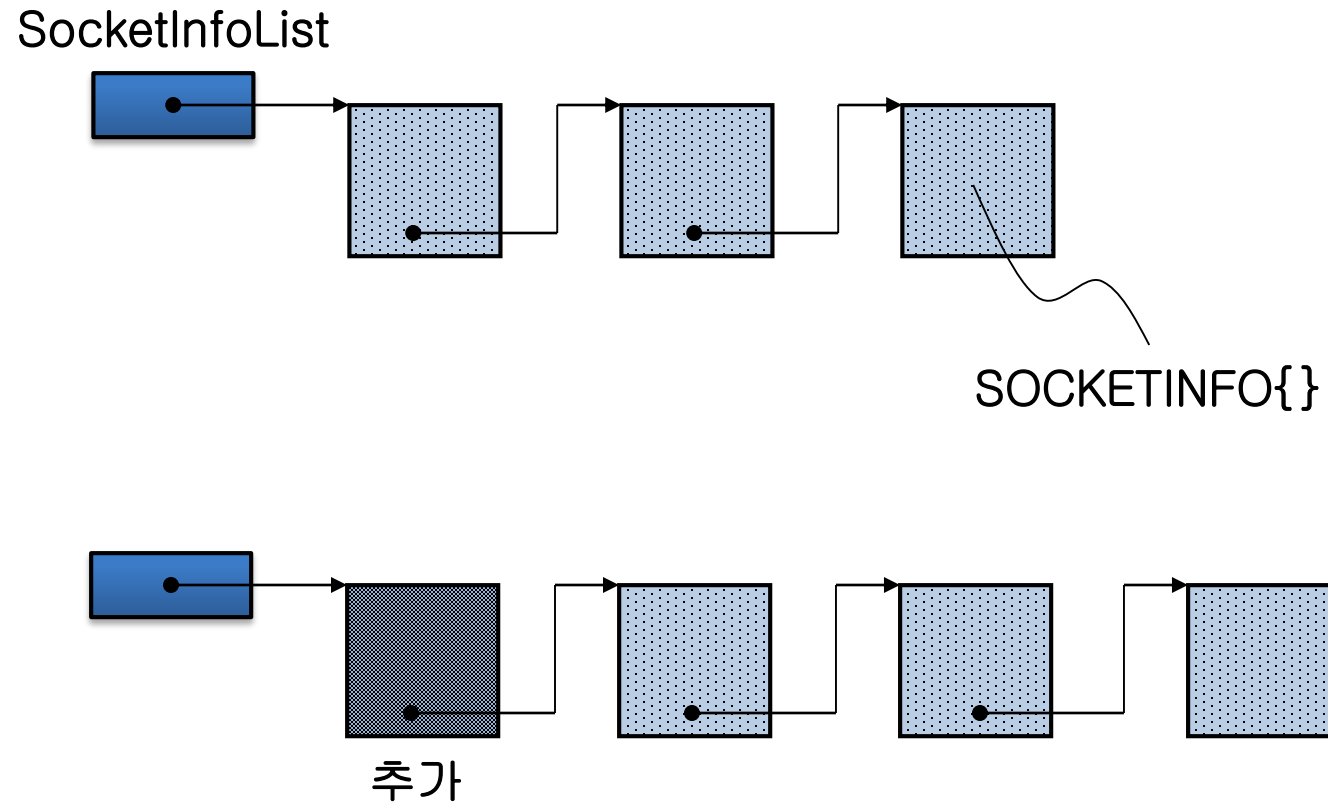
WSAAsyncSelect 모델 서버 작성

■ 실습 11-3 WSAAsyncSelect 모델 TCP 서버 작성과 테스트

- AsyncSelectTCPServer.cpp
- <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter11/AsyncSelectTCPServer/AsyncSelectTCPServer.cpp>

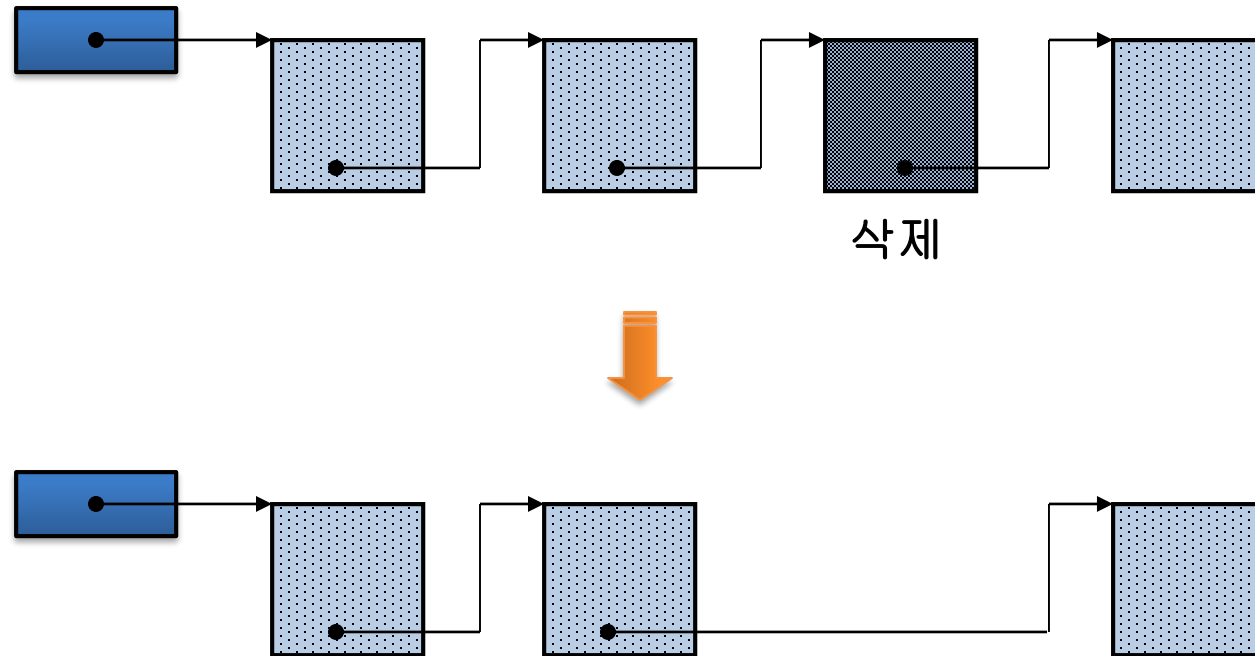
AsyncSelectTCPServer 예제의 소켓 정보 관리 (1)

■ 소켓 정보 관리를 위한 구조와 소켓 정보 추가



AsyncSelectTCPServer 예제의 소켓 정보 관리 (2)

■ 소켓 정보 삭제하기



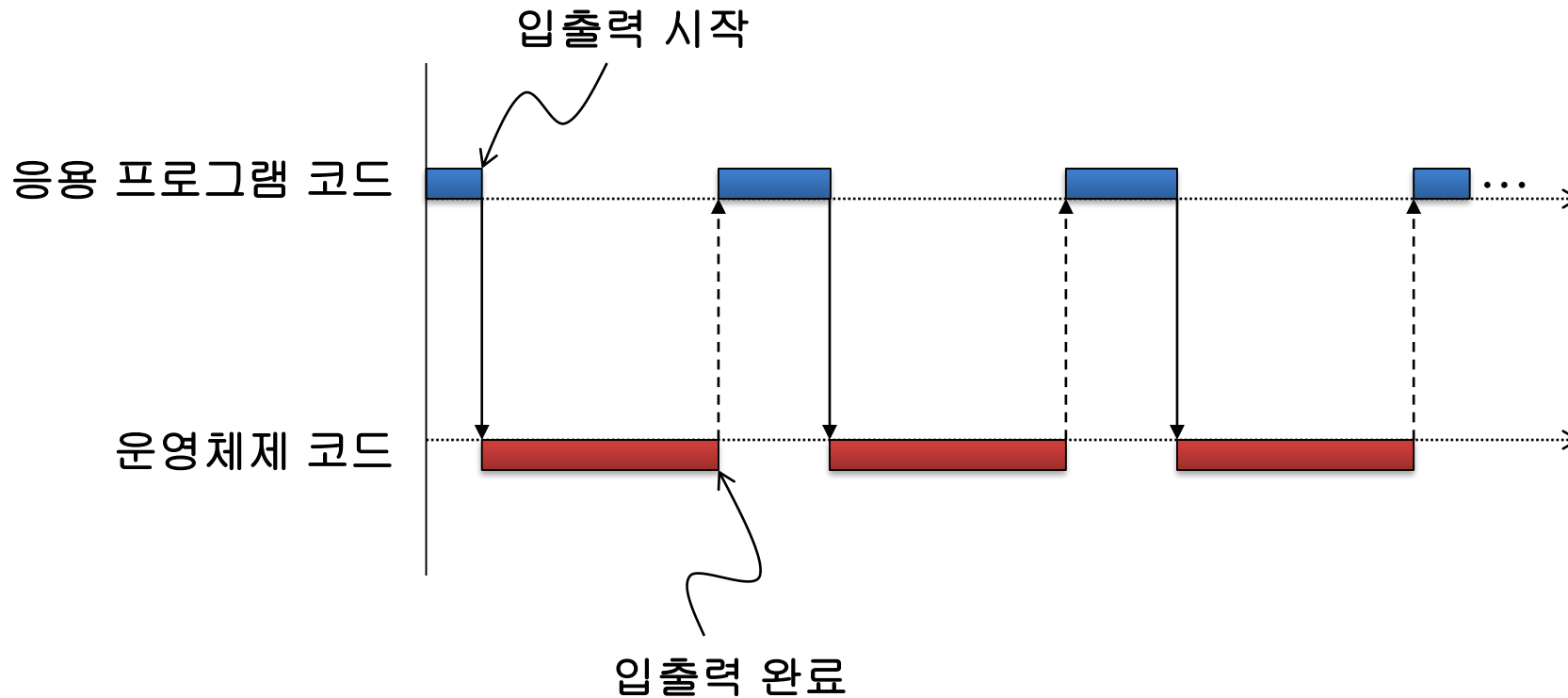
04 Completion Port 모델



동기 입출력과 비동기 입출력 (1)

■ 동기 입출력

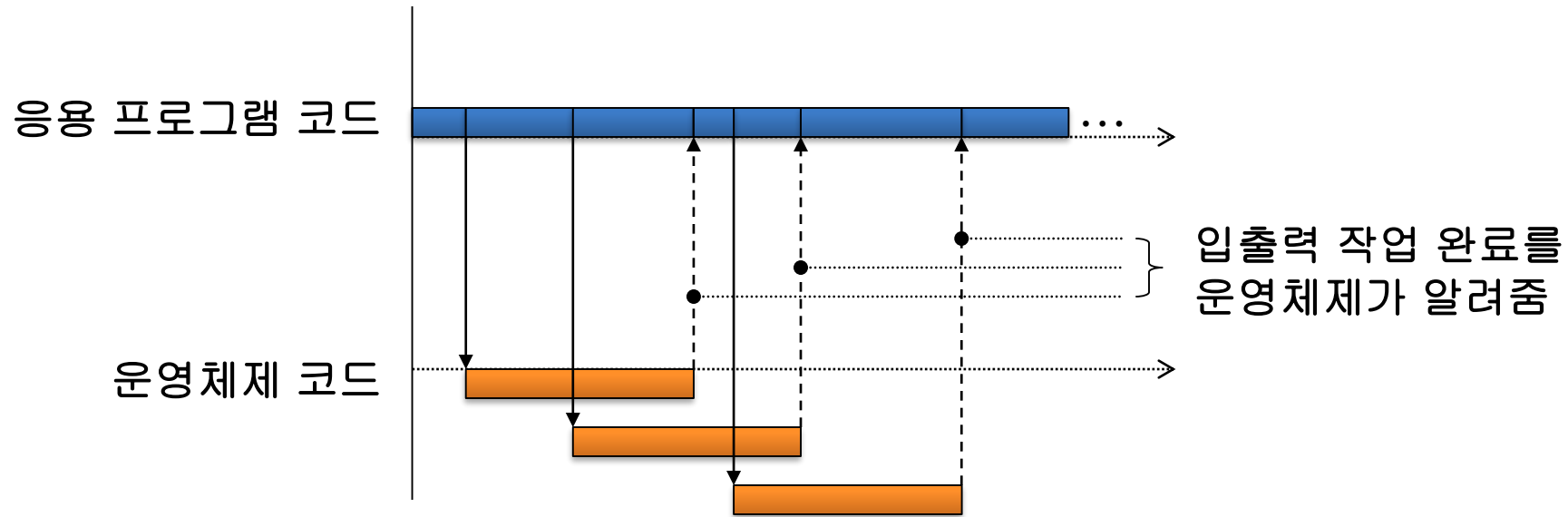
- 전형적인 입출력 방식
- 소켓 입출력 모델(Select, WSAAsyncSelect)은 모두 동기 입출력 방식으로 소켓 입출력을 처리



동기 입출력과 비동기 입출력 (2)

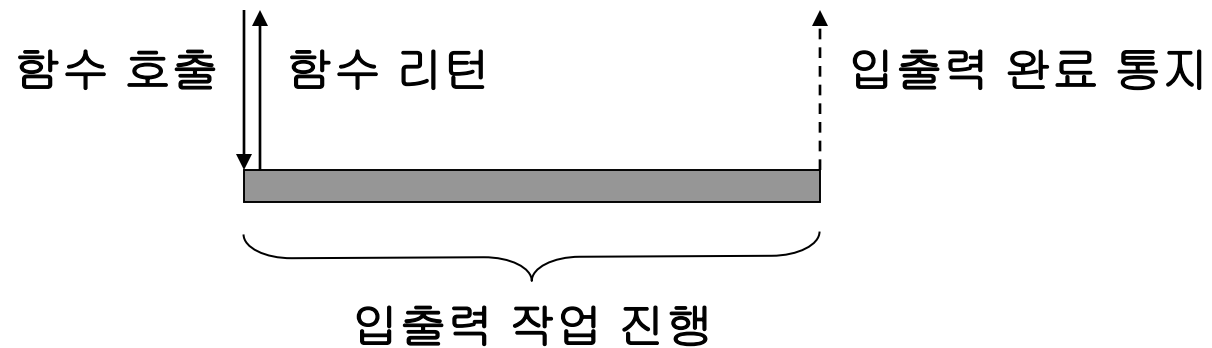
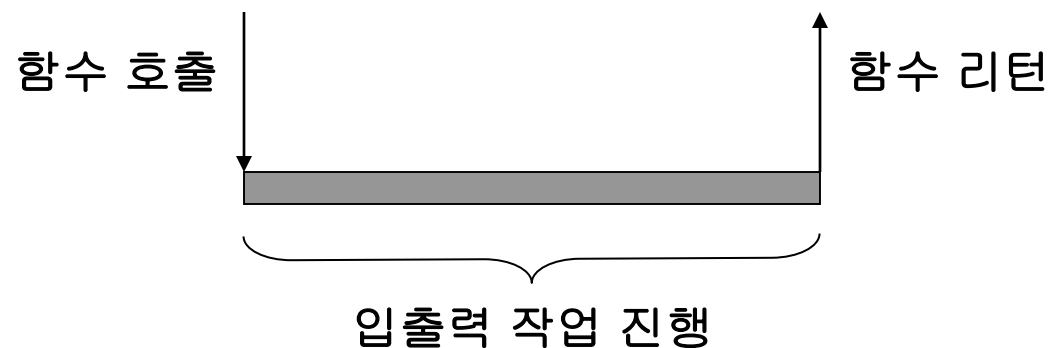
■ 비동기 입출력(중첩 입출력)

- Completion Port 모델은 비동기 입출력 방식으로 소켓 입출력을 처리



동기 입출력과 비동기 입출력 (3)

■ 동기와 비동기 개념



동기 입출력과 비동기 입출력 (4)

■ 비동기 통지의 개념



→ 2가지 역할 :

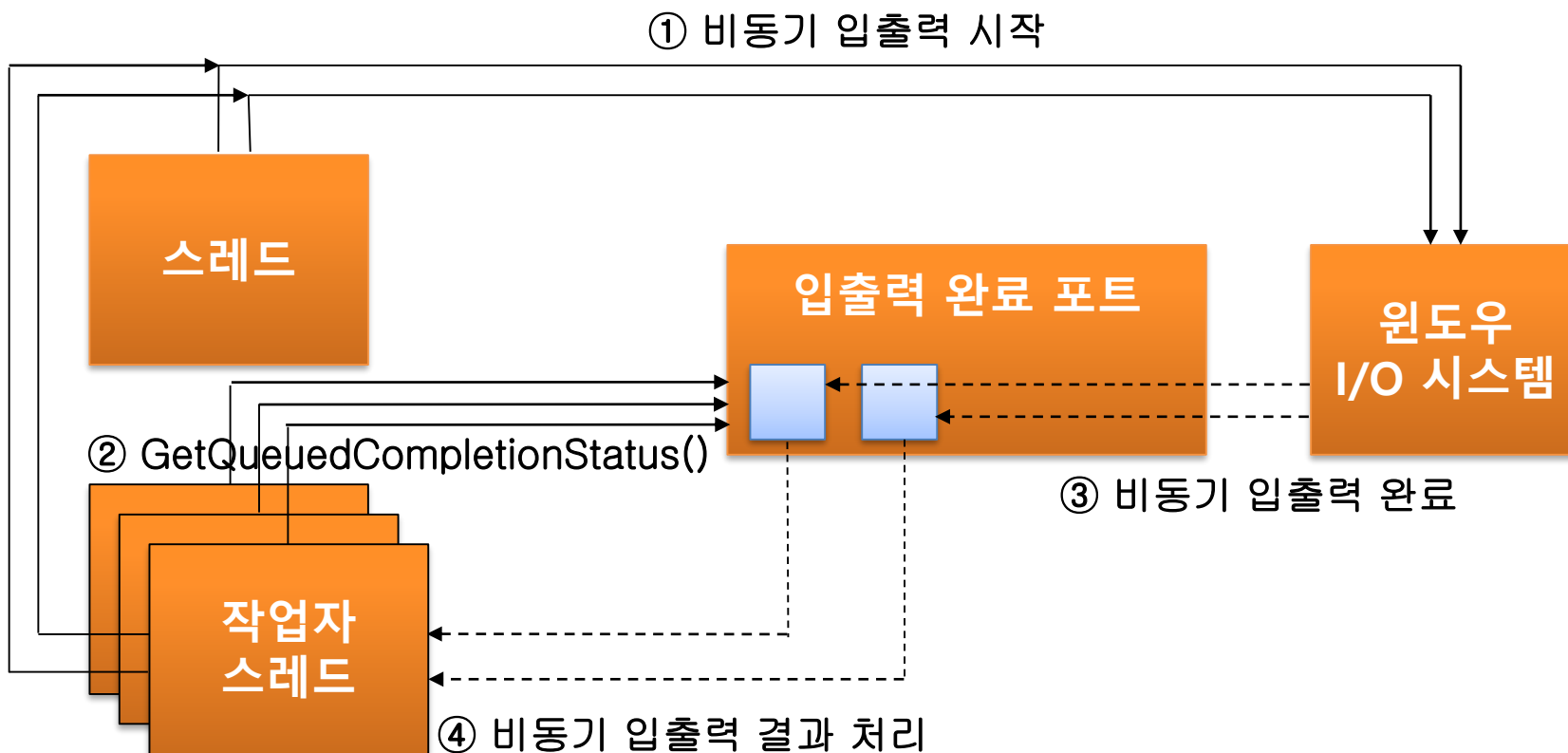
- ① 입력 2개 산출 1개를 만드는 것
- ② 비동기 입력 2개, 1개는 연결해서 처리

Completion Port 모델의 동작 원리 (1)

■ 특징

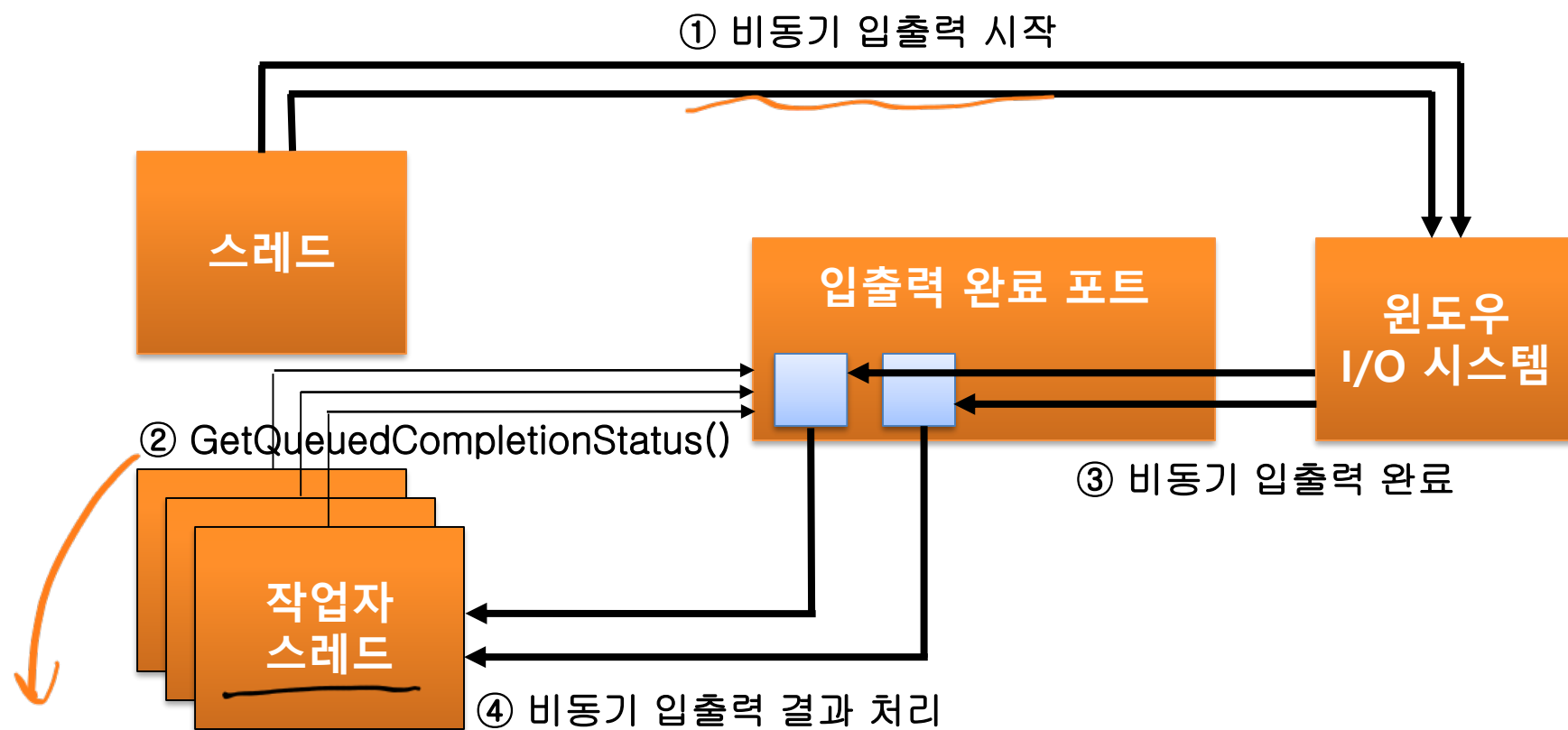
- 생성과 파괴, 스레드 접근 제약, 비동기 입출력 처리 방법

■ Completion Port 모델을 이용한 입출력 과정



Completion Port 모델의 동작 원리 (2)

■ Completion Port 모델의 실행 순서 (1)

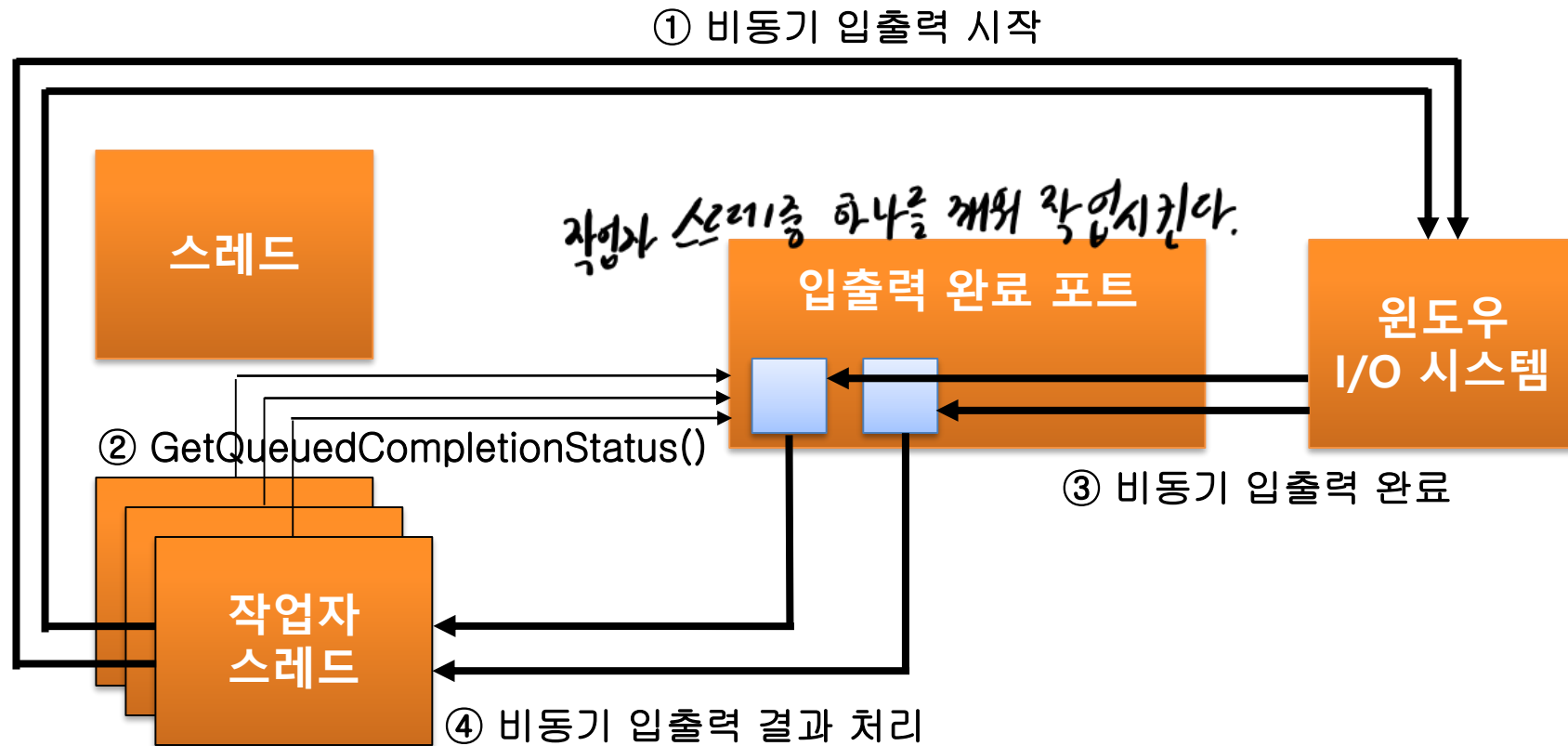


입력이 완료되지 않을 때 처리를 한다

1. 스레드를 대기시켜 주고 바쁘게

Completion Port 모델의 동작 원리 (3)

■ Completion Port 모델의 실행 순서 (2)



Completion Port 모델에서 새롭게 등장하는 함수 (1)

■ 입출력 완료 포트 생성

■ CreateIoCompletionPort() 함수의 역할

- ① 입출력 완료 포트를 새로 생성
- ② 소켓과 입출력 완료 포트를 연결

```
#include <windows.h>
```

```
HANDLE CreateIoCompletionPort(
```

```
    ① HANDLE FileHandle,
```

```
    ② HANDLE ExistingCompletionPort,
```

```
    ③ ULONG_PTR CompletionKey,
```

```
    ④ DWORD NumberOfConcurrentThreads
```

```
);
```

→ 완료 포트의 핸들 값
→ 작업 스레드

성공: 입출력 완료 포트 핸들, 실패: NULL

0을 넣으면 CPU 개수 만큼 만들어준다.

• CreateIoCompletionPort() 함수 사용 예

```
① 예제 HANDLE hcp = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 0);  
    if (hcp == NULL) return 1;
```

Completion Port 모델에서 새롭게 등장하는 함수 (2)

- 기존 입출력 완료 포트와 소켓을 연결하는 코드

② 여기

```
SOCKET sock;
```

```
...
```

```
HANDLE hResult = CreateIoCompletionPort((HANDLE)sock, hcp, (ULONG_PTR)sock, 0);
```

```
if (hResult == NULL) return 1;
```

Completion Port 모델에서 새롭게 등장하는 함수 (3)

■ 비동기 입출력(데이터 송/수신)

- send() 함수와 recv() 함수 대신 WSASend() 함수와 WSARecv() 함수를 사용해야 비동기 입출력을 할 수 있음

```
#include <winsock2.h>
int WSASend(
    ❶ SOCKET sock,
    ❷ LPWSABUF lpBuffers,
    DWORD dwBufferCount,
    ❸ LPDWORD lpNumberOfBytesSent,
    ❹ DWORD dwFlags,
    ❺ LPWSAOVERLAPPED lpOverlapped,
    ❻ LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

성공: 0, 실패: SOCKET_ERROR

Completion Port 모델에서 새롭게 등장하는 함수 (4)

```
#include <winsock2.h>
int WSARecv(
    ❶ SOCKET sock,
    ❷ LPWSABUF lpBuffers,
    DWORD dwBufferCount,
    ❸ LPDWORD lpNumberOfBytesRecv,
    ❹ LPDWORD lpFlags,
    ❺ LPWSAOVERLAPPED lpOverlapped,
    ❻ LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
);
```

성공: 0, 실패: SOCKET_ERROR

- 각 배열 요소(WSABUF 타입)는 버퍼의 시작 주소와 길이(바이트 단위)를 담음

```
typedef struct __WSABUF {
    u_long len; // 길이(바이트 단위)
    char *buf; // 버퍼 시작 주소
} WSABUF, *LPWSABUF;
```

Completion Port 모델에서 새롭게 등장하는 함수 (5)

- WSAOVERLAPPED 구조체 필드 중 처음 네 개는 운영체제가 내부적으로만 사용
- 마지막 필드인 hEvent는 이벤트 객체의 핸들값으로, 입출력 작업을 완료하면 hEvent가 가리키는 이벤트 객체가 신호 상태가 됨

```
typedef struct _WSAOVERLAPPED {  
    DWORD Internal;  
    DWORD InternalHigh;  
    DWORD Offset;  
    DWORD OffsetHigh;  
    WSAEVENT hEvent;  
} WSAOVERLAPPED, *LPWSAOVERLAPPED;
```

Completion Port 모델에서 새롭게 등장하는 함수 (6)

■ WSASend() 함수와 WSARecv() 함수의 특징

① Scatter/Gather 입출력을 지원

```
// 송신 측 코드
char buf1[128];
char buf2[256];
WSABUF wsabuf[2];
wsabuf[0].buf = buf1;
wsabuf[0].len = 128;
wsabuf[1].buf = buf2;
wsabuf[1].len = 256;
WSASend(sock, wsabuf, 2, ...);
```

```
// 수신 측 코드
char buf1[128];
char buf2[256];
WSABUF wsabuf[2];
wsabuf[0].buf = buf1;
wsabuf[0].len = 128;
wsabuf[1].buf = buf2;
wsabuf[1].len = 256;
WSARecv(sock, wsabuf, 2, ...);
```

- ② 마지막 두 인수에 NULL 값을 사용하면 send() 함수, recv() 함수처럼 동기 함수로 동작
- ③ 본문에서 소개하지 않은 Overlapped 모델에서는 WSAOVERLAPPED 구조체의 hEvent 필드 또는 lpCompletionRoutine 인수를 사용

Completion Port 모델에서 새롭게 등장하는 함수 (7)

■ 비동기 입출력 결과 확인

- 작업자 스레드는 GetQueuedCompletionStatus() 함수를 호출함으로써 입출력 완료 포트에 입출력 완료 패킷이 들어올 때까지 대기

```
#include <windows.h>

BOOL GetQueuedCompletionStatus(
    ① HANDLE CompletionPort,
    ② LPDWORD lpNumberOfBytes,
    ③ PULONG_PTR lpCompletionKey,
    ④ LPOVERLAPPED *lpOverlapped,
    ⑤ DWORD dwMilliseconds
);
```

성공: 0이 아닌 값, 실패: 0

Completion Port 모델 서버 작성

■ 실습 11-4 Completion Port 모델 TCP 서버 작성과 테스트

- CompletionPortTCPServer.cpp
- <https://github.com/promche/TCP-IP-Socket-Prog-Book-2nd/blob/Source/Windows/Chapter11/CompletionPortTCPServer/CompletionPortTCPServer.cpp>

05 소켓 입출력 모델 비교



소켓 입출력 모델 비교 (1)

■ 소켓 입출력 모델의 장점

- Select 모델
 - 모든 윈도우 운영체제는 물론, 유닉스/리눅스 운영체제에서도 사용할 수 있으므로 이식성이 높음
- WSAAsyncSelect 모델
 - 소켓 이벤트를 윈도우 메시지 형태로 처리하므로 GUI 응용 프로그램과 잘 결합할 수 있음
- Completion Port 모델
 - 비동기 입출력과 완료 포트를 통해 가장 뛰어난 성능을 제공

소켓 입출력 모델 비교 (2)

■ 각 입출력 모델의 단점

- Select 모델
 - 스레드당 처리할 수 있는 소켓의 개수가 64개로 제한되어 있음
- WSAAsyncSelect 모델
 - 단일 윈도우 프로시저에서 일반 윈도우 메시지+소켓 메시지를 처리해야 하므로 성능 저하 요인이 됨
- Completion Port 모델
 - 가장 단순한 소켓 입출력 방식(블로킹 소켓+스레드)과 비교하면 코딩이 복잡하지만, 성능 면에서 특별한 단점은 없음

소켓 입출력 모델 비교 (3)

■ 이상적인 소켓 입출력 모델에 요구되는 사항을 각 소켓 입출력 모델과 비교

- 소켓 함수 호출 시 블로킹을 최소화
 - 모든 모델이 만족
- 스레드 개수를 일정 수준으로 유지
 - 모든 모델이 만족
- CPU 명령 수행과 입출력 작업을 병행
 - 비동기 입출력 방식을 사용하는 Completion Port 모델만 만족
- 유저 모드와 커널 모드 전환 횟수를 최소화
 - 비동기 입출력 방식을 사용하는 Completion Port 모델만 만족