

Global KPU!

글로벌 경쟁력을 갖춘 산업기술 명문대학
세계를 향해 더 큰 미래를 펼쳐갑니다

11장. 소켓 입출력 모델(II)

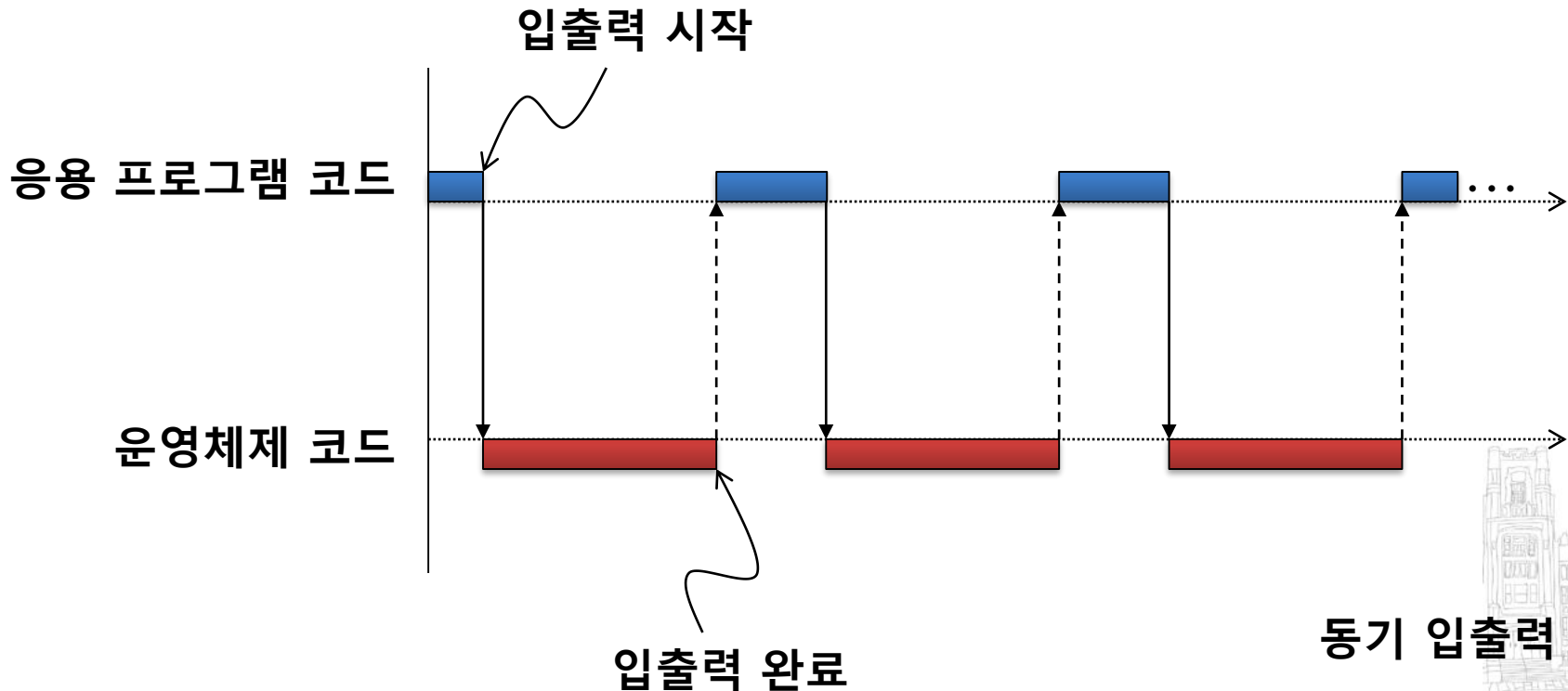
네트워크 게임 프로그래밍

- ❖ Overlapped 소켓 입출력 모델의 두 가지 형태를 이해하고 활용한다.
- ❖ Completion Port 소켓 입출력 모델을 이해하고 활용한다.



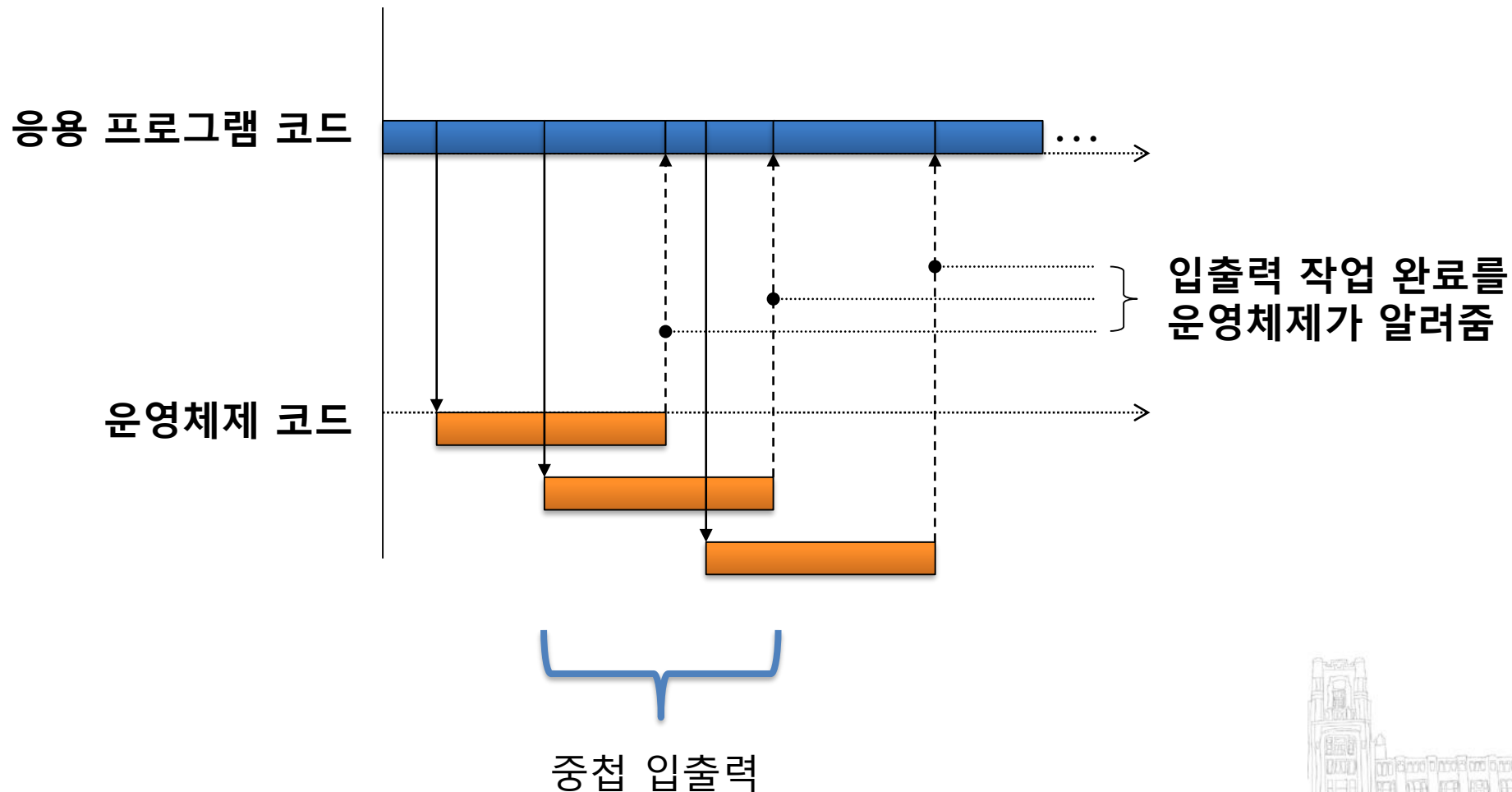
❖ 동기 vs. 비동기 입출력

- 동기 입출력: 결과값이 오기 전까지 블럭킹 상태가 되며, 블럭킹된 동안에는 다른 작업이 불가
- 비동기 입출력: 입출력 요청 후에도 다른 작업이 가능(중첩 입출력 포함)하며, 오버헤드가 줄고 효율적임.
- 10장에서 배운 소켓 입출력 모델은 동기 입출력 형태임.



입출력 방식 (2)

❖ 동기 vs. 비동기 입출력(계속) - 비동기 입출력



❖ 동기 입출력

- 응용 프로그램은 입출력 함수를 호출한 후 입출력 작업이 끝날 때까지 대기
- 입출력 작업이 끝나면 입출력 함수가 리턴하고 응용 프로그램은 입출력 결과를 처리하거나 다른 작업을 진행
- 운영체제가 함수 호출 시점을 알려주는 개념을 비동기 통지(asynchronous notification)이라고 함.

❖ 비동기 입출력 or 중첩 입출력

- 응용 프로그램은 입출력 함수를 호출(바로 리턴)한 후 입출력 작업의 완료 여부와 무관하게 다른 작업을 진행
- 입출력 작업이 끝나면 운영체제는 작업 완료를 응용 프로그램에 알려줌. 응용 프로그램은 하던 작업을 중단하고 입출력 결과를 처리
- 입출력 완료를 운영체제가 알려주는 개념이 반드시 필요하므로 비동기 통지도 사용



❖ 입출력 방식에 따른 소켓 입출력 모델 분류

- 동기 입출력 + 비동기 통지
 - Select 모델
 - WSAAsyncSelect 모델
 - WSAEventSelect 모델
- 비동기 입출력 + 비동기 통지
 - Overlapped 모델
 - Completion Port 모델



❖ Overlapped 모델 사용 절차

① 비동기 입출력을 지원하는 소켓 생성

socket() 함수로 생성한 소켓은 기본적으로 비동기 입출력을 지원

② 비동기 입출력을 지원하는 소켓 함수 호출

- AcceptEx(), ConnectEx(), DisconnectEx(), TransmitFile(), TransmitPackets(), WSALoctl(), WSANSPloctl(), WSAProviderConfigChange(), [WSARecv\(\)](#), [WSARecvFrom\(\)](#), WSARecvMsg(), [WSASend\(\)](#), [WSASendTo\(\)](#)

③ 운영체제가 입출력 작업 완료를 응용 프로그램에 알려주면(=비동기 통지), 응용 프로그램은 입출력 결과를 처리



❖ Overlapped 모델의 종류

- 운영체제의 비동기 통지 방식에 따라 두 종류로 구분.
- 어떤것을 사용하는가에 따라 응용 프로그램의 구조가 달라짐

종류	설명
Overlapped 모델(I)	소켓 입출력 작업이 완료되면 운영체제는 응용 프로그램이 등록해둔 이벤트 객체 를 신호 상태로 바꾼다. 응용 프로그램은 이벤트 객체를 관찰함으로써 입출력 작업 완료를 감지할 수 있다.
Overlapped 모델(II)	소켓 입출력 작업이 완료되면 운영체제는 응용 프로그램이 등록해둔 함수를 자동으로 호출한다. 일반적으로 운영체제가 호출하는 응용 프로그램 함수를 콜백 함수라 하는데 특별히 Overlapped 모델에서는 완료 루틴 이라 부른다.



❖ 입출력 함수

- 접속된 상대방 소켓에게 지정한 데이터를 보내는 함수

```
int WSASend (  
    SOCKET s,  
    LPWSABUF lpBuffers,  
    DWORD dwBufferCount,  
    LPDWORD lpNumberOfBytesSent,  
    DWORD dwFlags,  
    LPWSAOVERLAPPED lpOverlapped,  
    LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine  
);
```

성공: 0, 실패: SOCKET_ERROR

- s: 접속된 소켓을 가리키는 소켓 기술자(디스크립터)
- lpBuffers: WSABUF 구조체 배열을 가리키는 포인터. WSABUF 구조체는 버퍼의 포인터와 길이를 나타내는 함수로 구성되는 구조체로 이 구조체의 배열을 포인팅해서 넘겨줌(뒷장에 설명)
- dwBufferCount: lpBuffers가 가리키는 WSABUF 구조체 배열의 크기
- lpNumberOfBytesSent: 전송된 바이트 수. 함수 호출이 성공하면 보내거나 받은 바이트 수를 저장.
- dwFlags: 어떠한 방식으로 전송을 수행할 것인지 설정하는 플래그. 대부분 0을 사용.
- lpOverlapped: WSAOVERLAPPED 구조체(뒷장에 설명)의 포인터. 중첩된 출력이 아닌 경우 전달된 인자는 무시.
- lpCompletionRoutine: 전송연산이 완료되었을때 호출되는 전송 완료 루틴에 대한 포인터. 입출력 작업이 완료되면 운영체제가 자동으로 호출할 완료 루틴(콜백 함수)의 주소 값.



❖ 입출력 함수(계속)

- Overlapped 모델(I)에서는 WSAOVERLAPPED 구조체의 hEvent 변수를, Overlapped 모델(II)에서는 IpCompletionRoutine 인자를 사용
- IpCompletionRoutine 인자의 우선순위가 높으므로 이 값이 NULL 이 아니면 WSAOVERLAPPED 구조체의 hEvent 변수는 사용되지 않음

```
int WSARecv (  
    SOCKET s,  
    LPWSABUF IpBuffers,  
    DWORD dwBufferCount,  
    LPDWORD IpNumberOfBytesRecv,  
    LPDWORD IpFlags,  
    LPWSAOVERLAPPED IpOverlapped,  
    LPWSAOVERLAPPED_COMPLETION_ROUTINE IpCompletionRoutine  
);
```

성공: 0, 실패: SOCKET_ERROR



❖ 관련 구조체

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

```
typedef struct _WSAOVERLAPPED {  
    DWORD Internal;  
    DWORD InternalHigh;  
    DWORD Offset;  
    DWORD OffsetHigh;  
    WSAEVENT hEvent;    // 작업에 대한 이벤트 등록  
} WSAOVERLAPPED, *LPWSAOVERLAPPED;
```

- hEvent는 이벤트 객체의 핸들 값. 이외의 인자는 운영체제가 내부적으로만 사용.



❖ Overlapped 모델(I)을 이용한 소켓 입출력 절차

- ① 비동기 입출력을 지원하는 소켓을 생성. 이때 **WSACreateEvent()** 함수를 호출하여 대응하는 이벤트 객체도 같이 생성
- ② 비동기 입출력을 지원하는 소켓 함수를 호출. 이때 **WSAOVERLAPPED** 구조체의 **hEvent** 변수에 이벤트 객체의 핸들 값을 넣어서 전달
- ③ **WSAWaitForMultipleEvents()** 함수를 호출해 이벤트 객체가 신호 상태가 되기를 기다림
- ④ 비동기 입출력 작업이 완료하여 **WSAWaitForMultipleEvents()** 함수가 리턴하면 **WSAGetOverlappedResult()** 함수를 호출해 비동기 입출력 결과를 확인하고 데이터를 처리
- ⑤ 새로운 소켓을 생성하면 ①~④를, 그렇지 않으면 ②~④를 반복



❖ WSAGetOverlappedResult() 함수

- 매개 변수로 지정한 소켓에 대한 오버랩 연산의 결과를 반환
 - 입출력이 성공 혹은 실패했는지, 성공했다면 몇 바이트나 입출력이 되었는지에 대한 결과를 확인

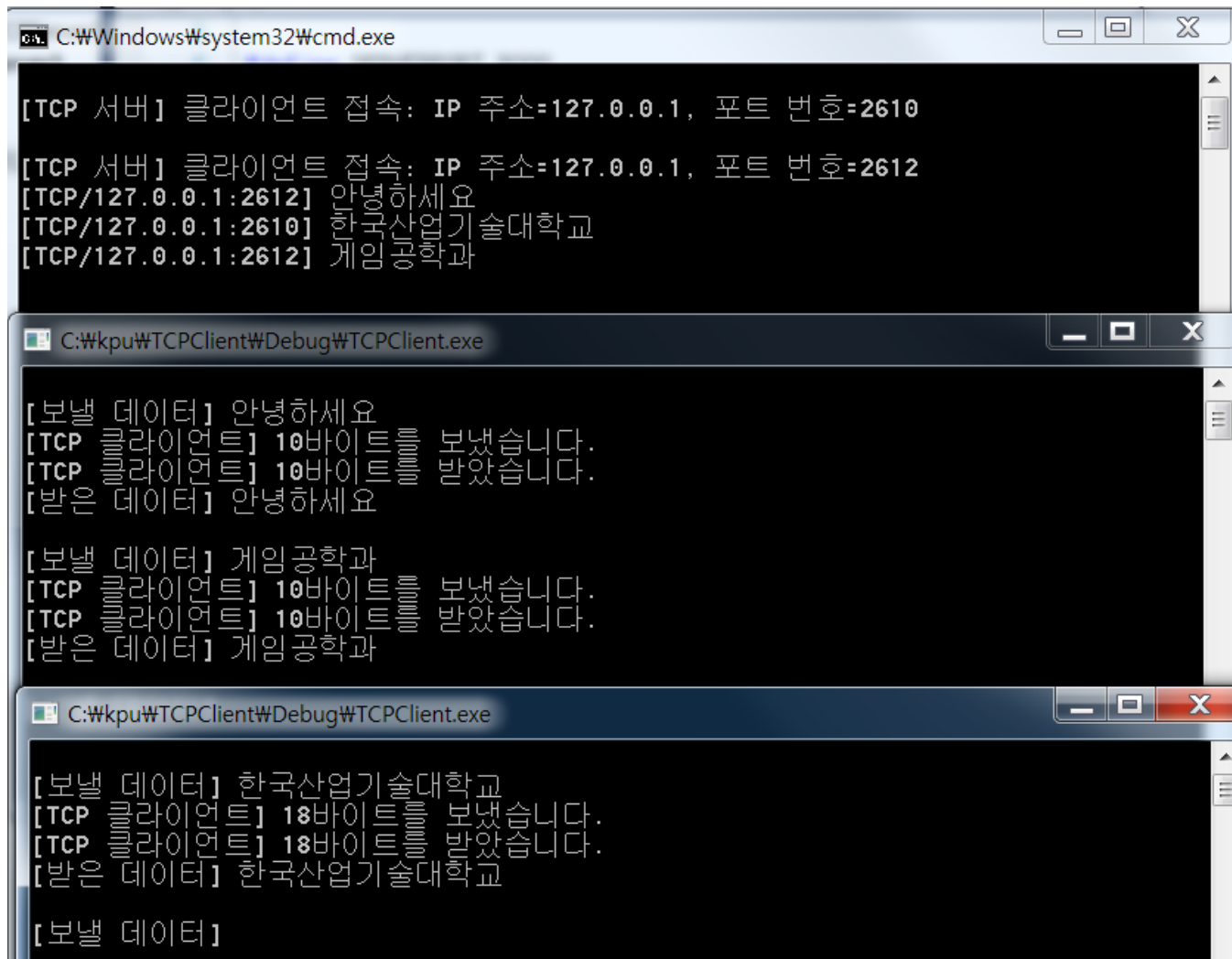
```
BOOL WSAGetOverlappedResult (  
    SOCKET s,  
    LPWSAOVERLAPPED lpOverlapped,  
    LPDWORD lpcbTransfer,  
    BOOL fWait,  
    LPDWORD lpdwFlags  
);
```

성공: TRUE, 실패: FALSE

- s: 소켓을 식별하는 기술자. 비동기 입출력 함수 호출에 사용했던 소켓을 넣음.
- lpOverlapped: 오버랩 연산이 시작될 때 지정된 WSAOVERLAPPED 구조체에 대한 포인터
- lpcbTransfer: 이 변수를 통해 실제로 전송된 바이트를 얻음. 전송된 바이트 수가 여기에 저장
- fWait: 오버랩 연산을 완료하기 위해 대기할 것인지를 결정하는 플래그. 비동기 입출력 작업이 끝날때까지 대기하려면 TRUE, 그렇지 않으면 FALSE를 사용
- lpdwFlags: WSARecv 함수가 호출되었을 경우 부수적인 정보를 얻기 위해 사용. 거의 사용하지 않음



실습 11-1 P432~



The image displays three overlapping Windows command prompt windows. The top window, titled 'C:\Windows\system32\cmd.exe', shows a TCP server listening on port 2610 and 2612, receiving connections from 127.0.0.1, and responding with '안녕하세요' and '한국산업기술대학교 게임공학과'. The middle window, titled 'C:\wpu\TCPClient\Debug\TCPClient.exe', shows a client sending and receiving 10-byte data packets with the message '안녕하세요'. The bottom window, also titled 'C:\wpu\TCPClient\Debug\TCPClient.exe', shows a client sending and receiving 18-byte data packets with the message '한국산업기술대학교'.

```
C:\Windows\system32\cmd.exe
[ TCP 서버 ] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=2610
[ TCP 서버 ] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=2612
[ TCP/127.0.0.1:2612 ] 안녕하세요
[ TCP/127.0.0.1:2610 ] 한국산업기술대학교
[ TCP/127.0.0.1:2612 ] 게임공학과

C:\wpu\TCPClient\Debug\TCPClient.exe
[ 보낼 데이터 ] 안녕하세요
[ TCP 클라이언트 ] 10바이트를 보냈습니다.
[ TCP 클라이언트 ] 10바이트를 받았습니다.
[ 받은 데이터 ] 안녕하세요

[ 보낼 데이터 ] 게임공학과
[ TCP 클라이언트 ] 10바이트를 보냈습니다.
[ TCP 클라이언트 ] 10바이트를 받았습니다.
[ 받은 데이터 ] 게임공학과

C:\wpu\TCPClient\Debug\TCPClient.exe
[ 보낼 데이터 ] 한국산업기술대학교
[ TCP 클라이언트 ] 18바이트를 보냈습니다.
[ TCP 클라이언트 ] 18바이트를 받았습니다.
[ 받은 데이터 ] 한국산업기술대학교

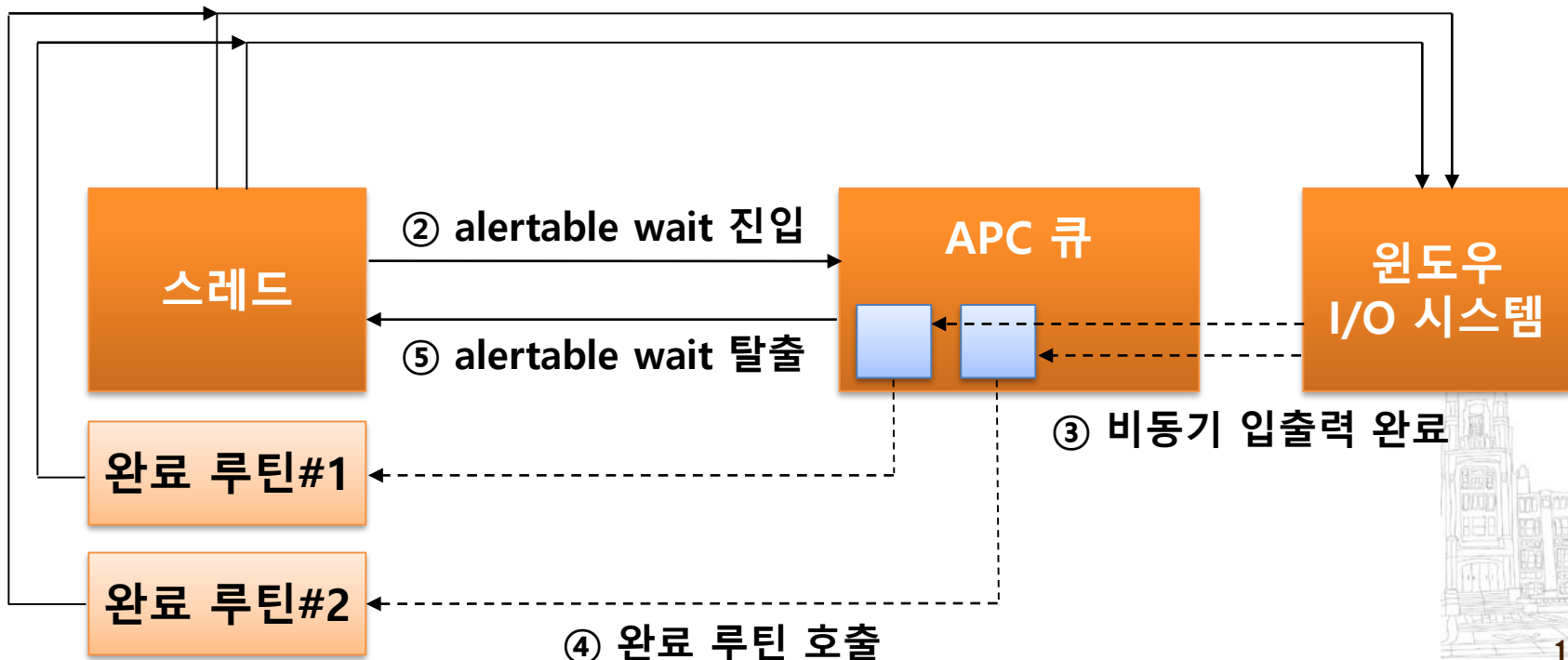
[ 보낼 데이터 ]
```



❖ Overlapped 모델(II)의 동작 원리

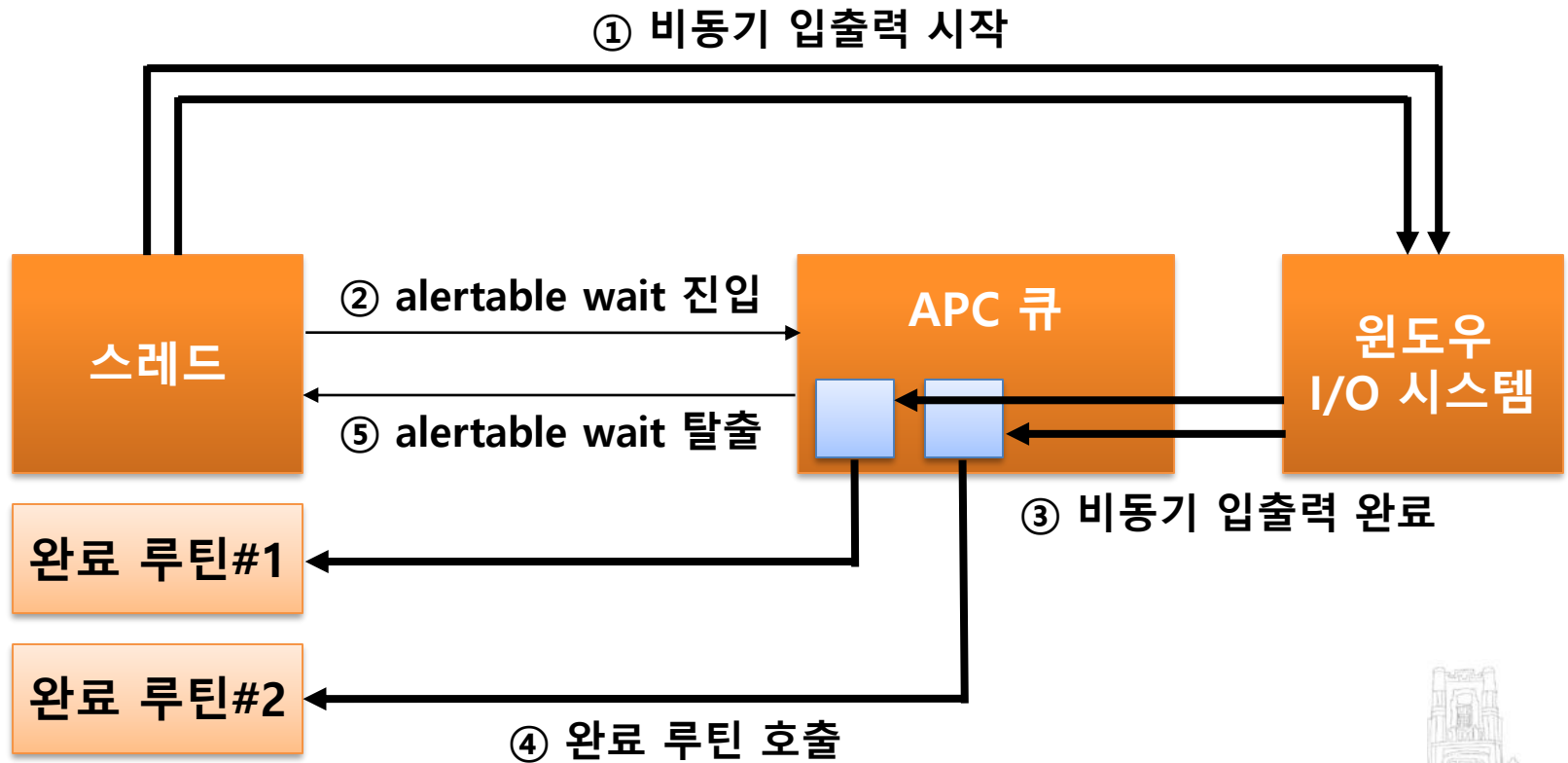
- 비동기 입출력 함수를 호출함으로써 운영체제에 입출력 작업을 요청
- 해당 스레드는 곧바로 alertable wait 상태에 진입
- 비동기 입출력 작업이 완료되면, 운영체제는 스레드의 APC 큐에 결과를 저장
- 비동기 입출력 함수를 호출한 스레드가 alertable wait 상태에 있으면 운영체제는 APC큐에 저장된 정보(완료 루틴의 주소)를 참조하여 완료 루틴 호출
- APC 큐에 저장된 정보를 토대로 모든 완료 루틴 호출이 끝나면 스레드는 alertable wait 상태에서 빠져 나옴

① 비동기 입출력 시작



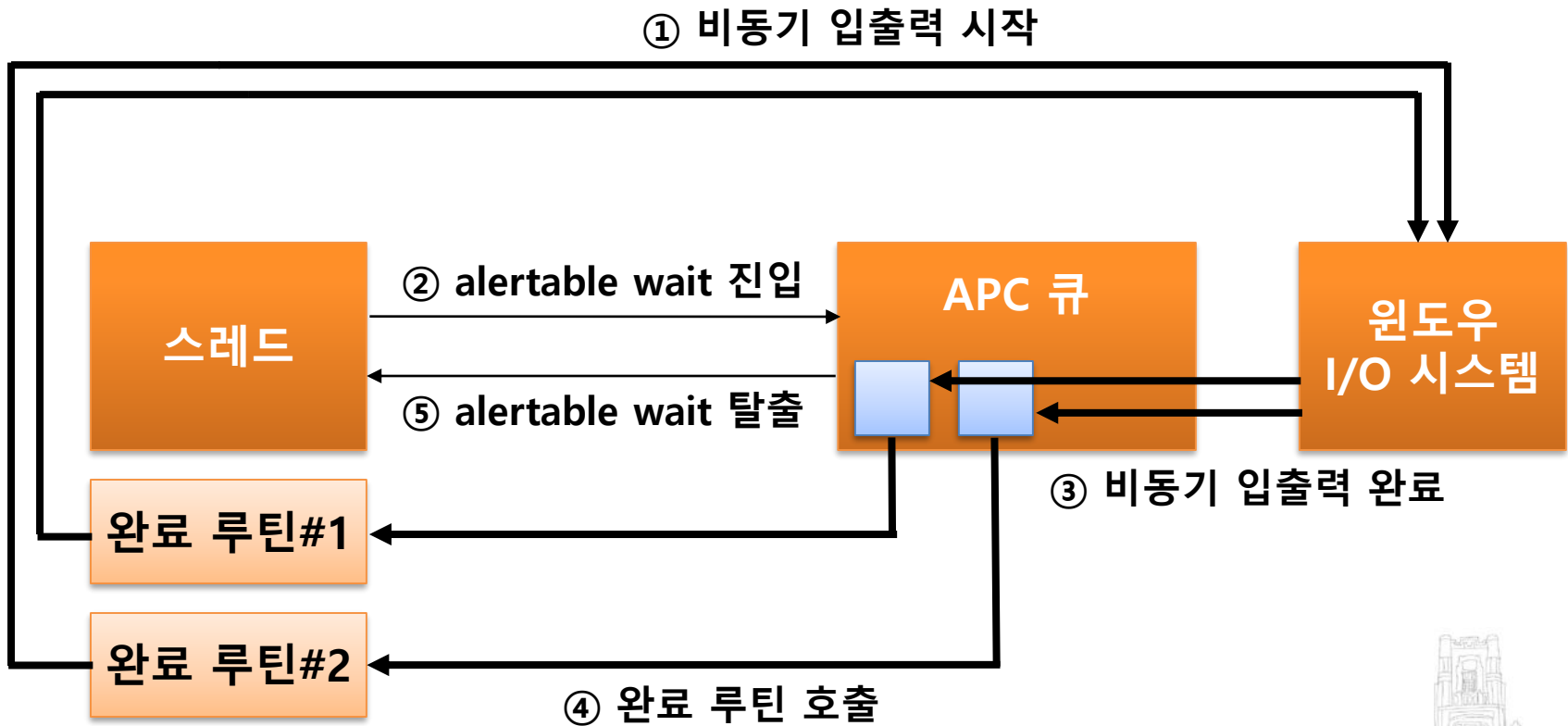
Overlapped 모델 II (2)

❖ Overlapped 모델(II)의 동작 원리(계속)



Overlapped 모델 II (3)

❖ Overlapped 모델(II)의 동작 원리(계속)



❖ Overlapped 모델(II)를 이용한 소켓 입출력 절차

- ① 비동기 입출력을 지원하는 소켓을 생성
- ② 비동기 입출력을 지원하는 소켓 함수를 호출. 이때 완료 루틴의 시작 주소를 함수 인자로 전달
- ③ 비동기 입출력 함수를 호출한 스레드를 alertable wait 상태로 만듦
- ④ 비동기 입출력 작업이 완료되면 운영체제는 완료 루틴을 호출.
완료 루틴에서는 비동기 입출력 결과를 확인하고 후속 처리를 함
- ⑤ 완료 루틴 호출이 모두 끝나면 스레드는 alertable wait 상태에서 빠져나옴
- ⑥ 새로운 소켓을 생성하면 ①~⑤를, 그렇지 않으면 ②~⑤를 반복



❖ 완료 루틴

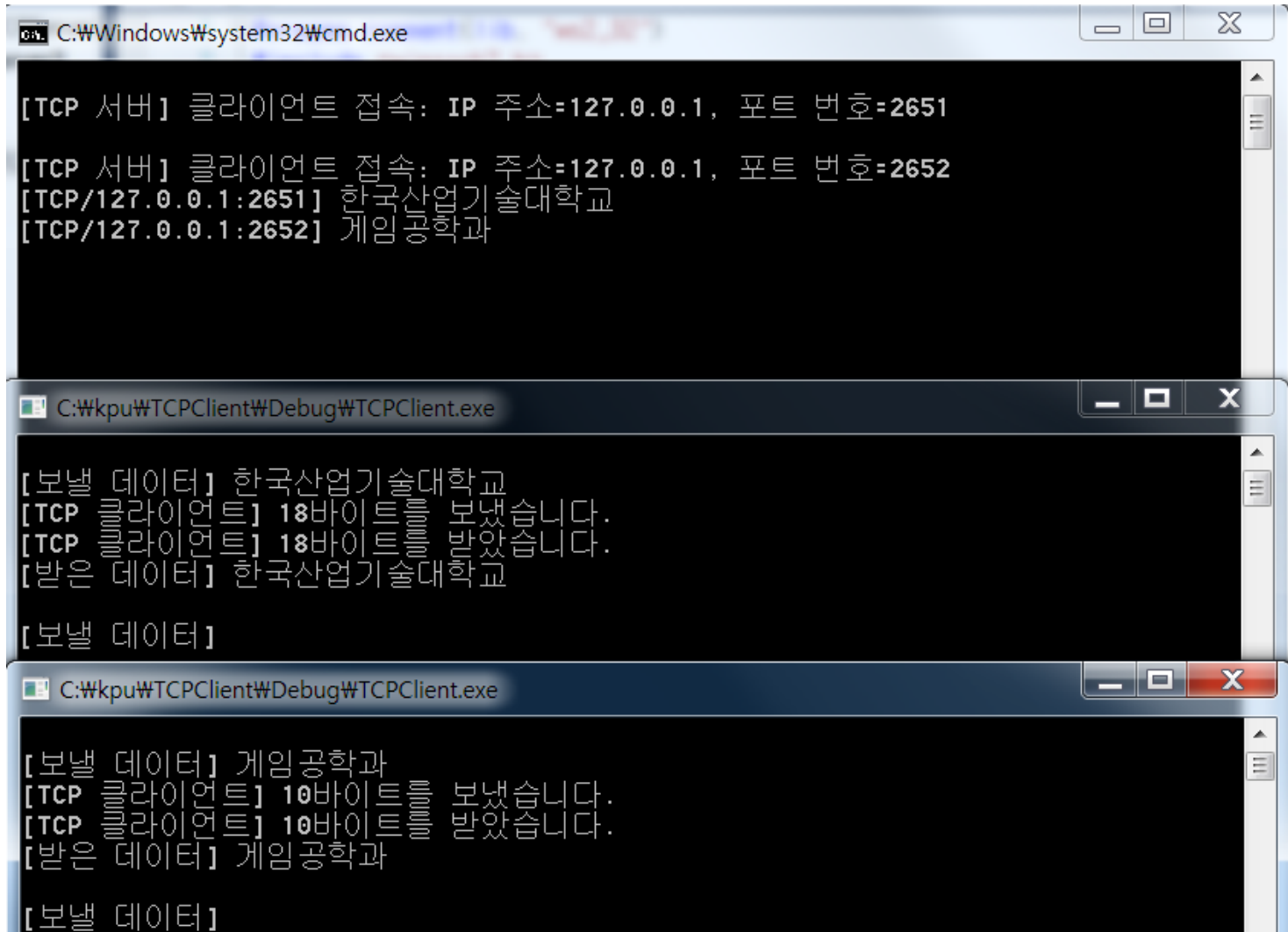
- I/O가 완료되었을 때, 자동으로 호출될 함수 (**WSASend, WSARecv** 함수의 마지막 전달 인자)

```
void CALLBACK CompletionRoutine (  
    DWORD dwError,  
    DWORD cbTransferred,  
    LPWSAOVERLAPPED lpOverlapped,  
    DWORD dwFlags  
);
```

- dwError: 오류정보 전달. 비동기 입출력 결과. 오류가 발생하면 0이 아닌 값이 됨
- cbTransferred: 전송된 바이트 수가 전달. 통신 상대가 접속을 종료하면 값은 0이 됨
- lpOverlapped: WSARecv 함수 호출 시 전달한 구조체 변수. 비동기 입출력 함수 호출 시 넘겨준 WSAOVERLAPPED 구조체의 주소값이 이 인자를 통해 다시 응용 프로그램에 넘겨옴. Overlapped 모델(II)에서는 이벤트 객체를 사용하지 않으므로 WSAOVERLAPPED 구조체를 완료 루틴 내부에서 직접 사용할 일은 없음.
- dwFlags: 옵션. 대부분 0임.



실습 11-2 P444~



The image shows three overlapping Windows command prompt windows. The top window is titled 'C:\Windows\system32\cmd.exe' and displays logs for a TCP server. The middle and bottom windows are titled 'C:\Wkpu\TCPClient\Debug\TCPClient.exe' and display logs for a TCP client. The logs show the client connecting to the server at IP 127.0.0.1 on ports 2651 and 2652, and the server responding with data.

```
C:\Windows\system32\cmd.exe
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=2651
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=2652
[TCP/127.0.0.1:2651] 한국산업기술대학교
[TCP/127.0.0.1:2652] 게임공학과

C:\Wkpu\TCPClient\Debug\TCPClient.exe
[보낼 데이터] 한국산업기술대학교
[TCP 클라이언트] 18바이트를 보냈습니다.
[TCP 클라이언트] 18바이트를 받았습니다.
[받은 데이터] 한국산업기술대학교

[보낼 데이터]

C:\Wkpu\TCPClient\Debug\TCPClient.exe
[보낼 데이터] 게임공학과
[TCP 클라이언트] 10바이트를 보냈습니다.
[TCP 클라이언트] 10바이트를 받았습니다.
[받은 데이터] 게임공학과

[보낼 데이터]
```

❖ 입출력 완료 포트(IOCP)

- 비동기 입출력 결과와 결과를 처리할 스레드에 관한 정보를 담고 있는 구조
 - Overlapped 모델(II)에서 소개한 APC 큐와 비슷한 개념

❖ APC 큐 vs. 입출력 완료 포트

- 생성과 파괴
 - APC 큐는 각 스레드마다 자동으로 생성되고 파괴. 입출력 완료 포트는 **CreateIoCompletionPort()** 함수를 호출하여 생성하고 **CloseHandle()** 함수를 호출하여 파괴
- 접근 제약
 - APC 큐에 저장된 결과는 APC 큐를 소유한 스레드만 확인할 수 있지만 입출력 완료 포트에는 이런 제약이 없음



❖ APC 큐 vs. 입출력 완료 포트(계속)

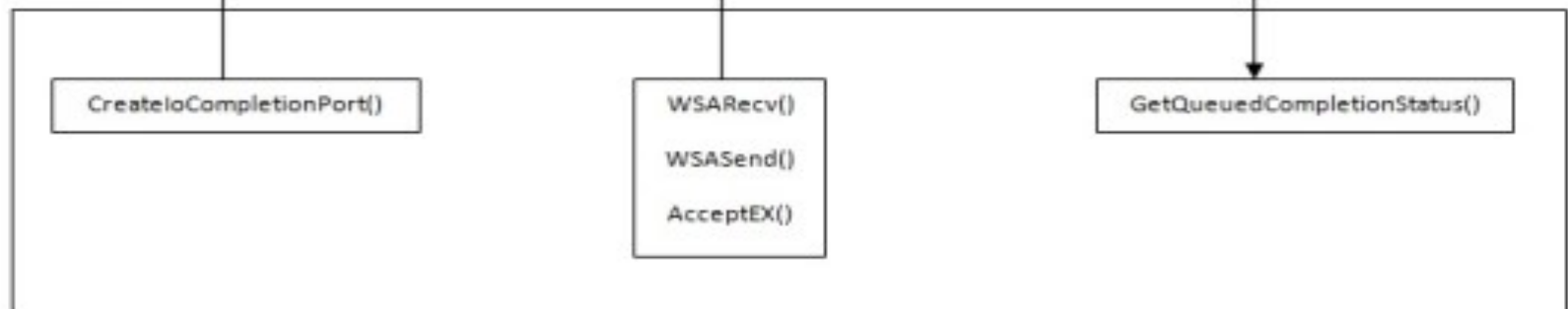
■ 비동기 입출력 처리 방법

- APC 큐에 저장된 결과를 처리하려면 해당 스레드는 alertable wait 상태에 진입해야 함. 입출력 완료 포트에 저장된 결과를 처리하려면 작업자 스레드는 **GetQueuedCompletionStatus()** 함수를 호출해야 함

커널 모드

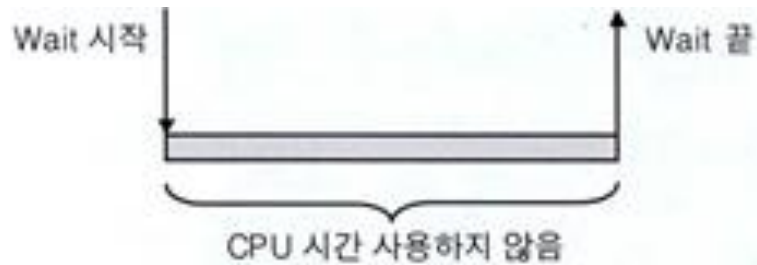


유저 모드

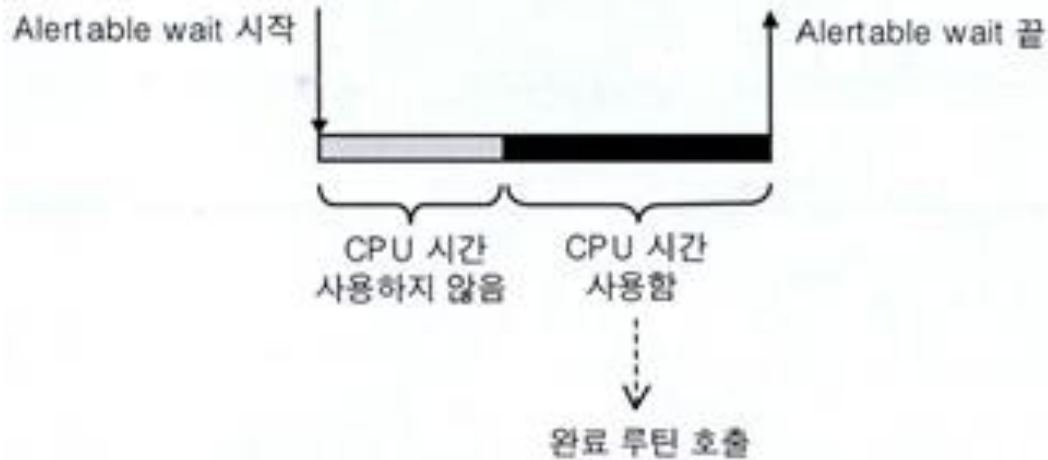


Completion Port 모델 (2)

❖ Alertable wait



▶ Wait

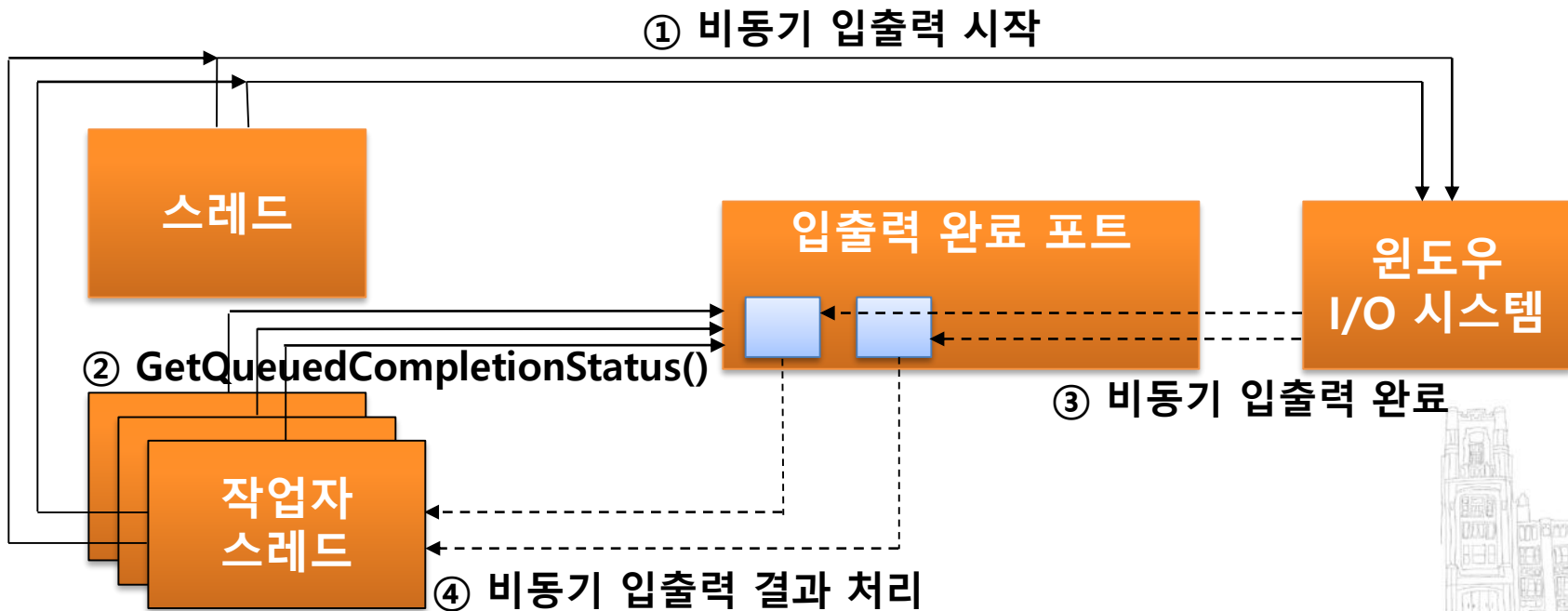


▶ Alertable wait



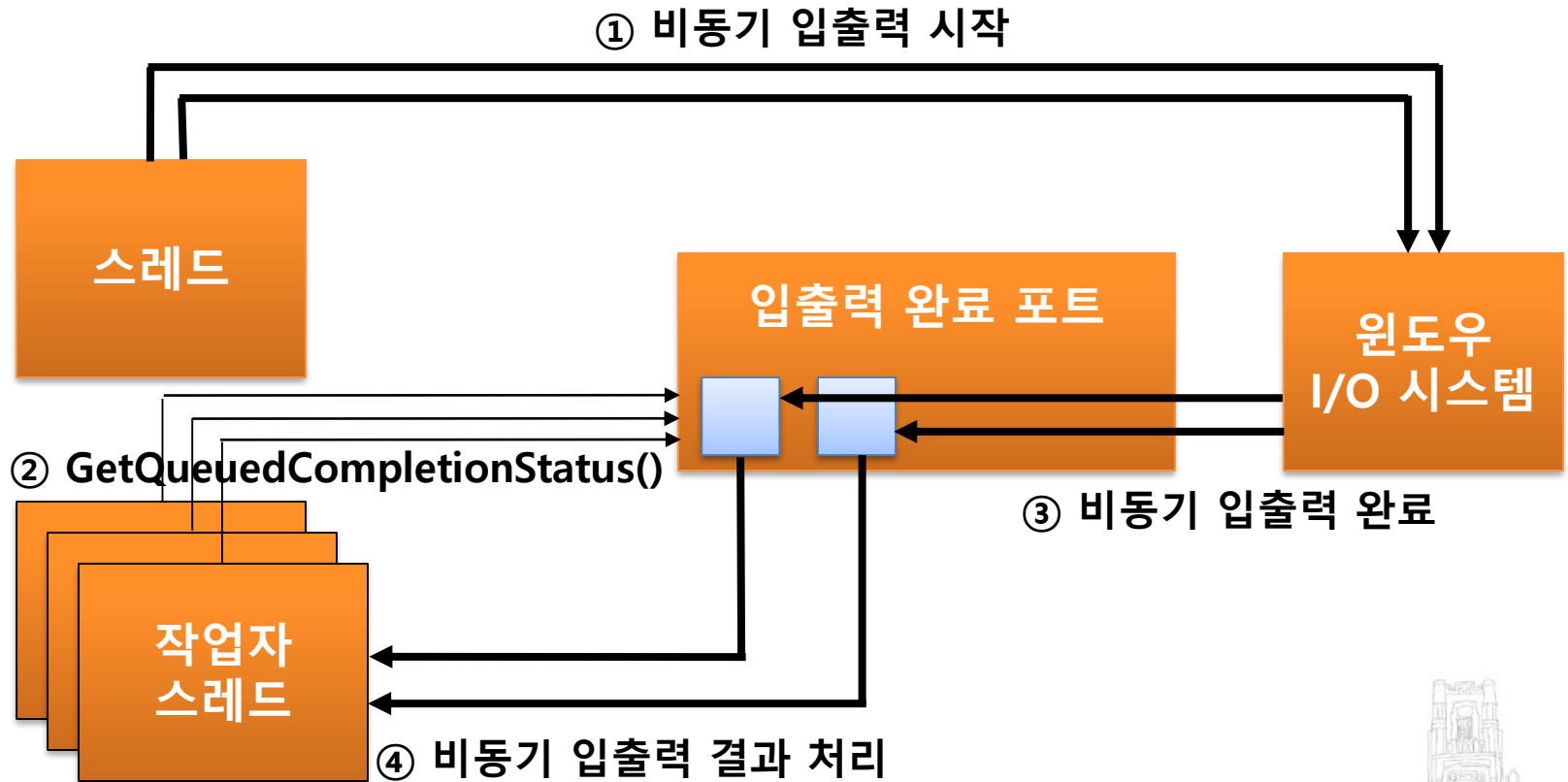
❖ Completion Port 모델 동작 원리

- 응용 프로그램을 구성하는 임의의 스레드에서 비동기 입출력 함수를 호출함으로써 운영체제에 입출력 작업을 요청
- 모든 작업자 스레드는 GetQueuedCompletionStatus() 함수를 호출해 입출력 완료 포트를 감시
- 비동기 입출력 작업이 완료되면 운영체제는 입출력 완료 포트에 결과를 저장. 입출력 완료 패킷
- 운영체제는 입출력 완료 포트에 저장된 작업자 스레드 목록에서 하나를 선택하여 깨움. 대기 상태에서 깨어난 작업자 스레드는 비동기 입출력 결과를 처리.



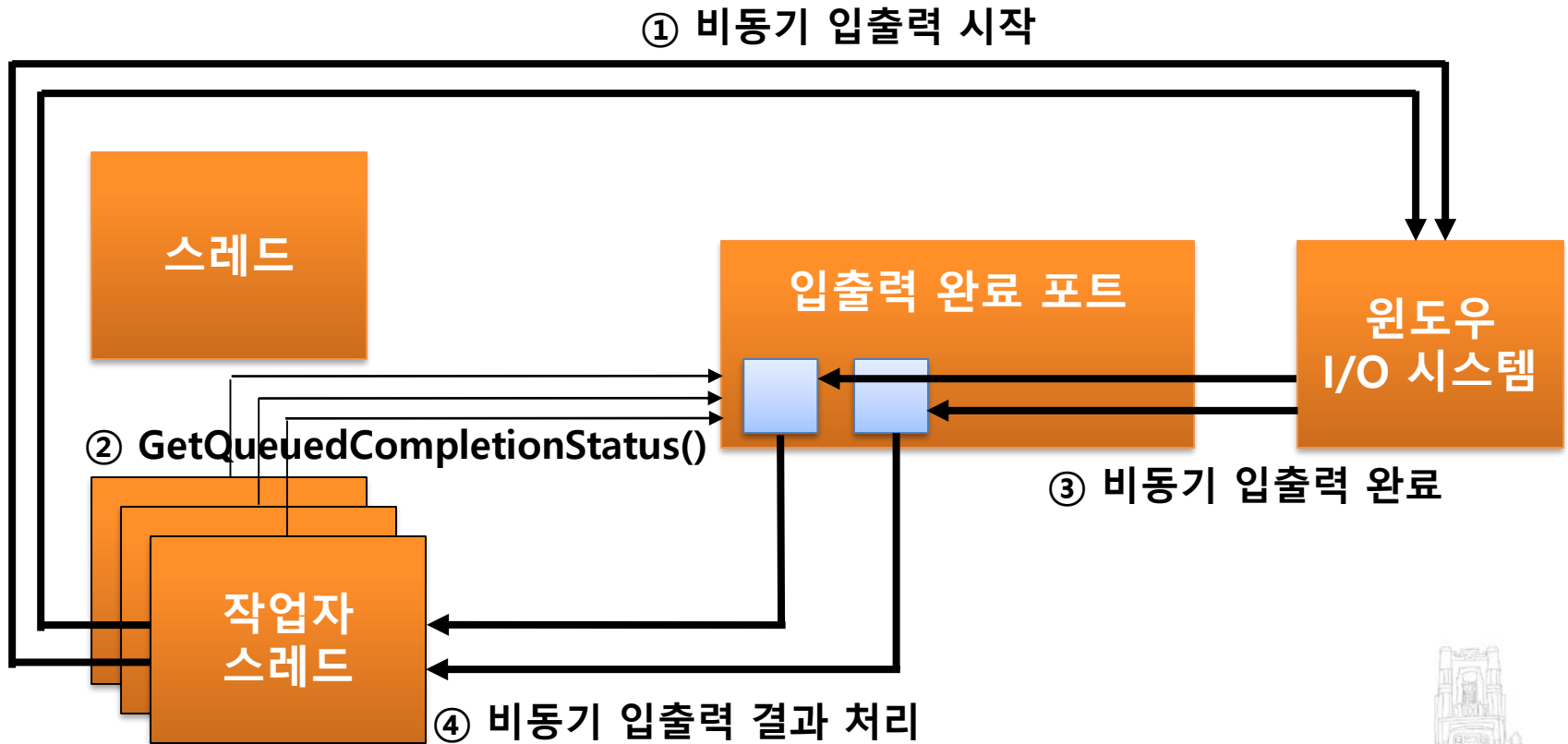
Completion Port 모델 (4)

❖ Completion Port 모델 동작 원리(계속)



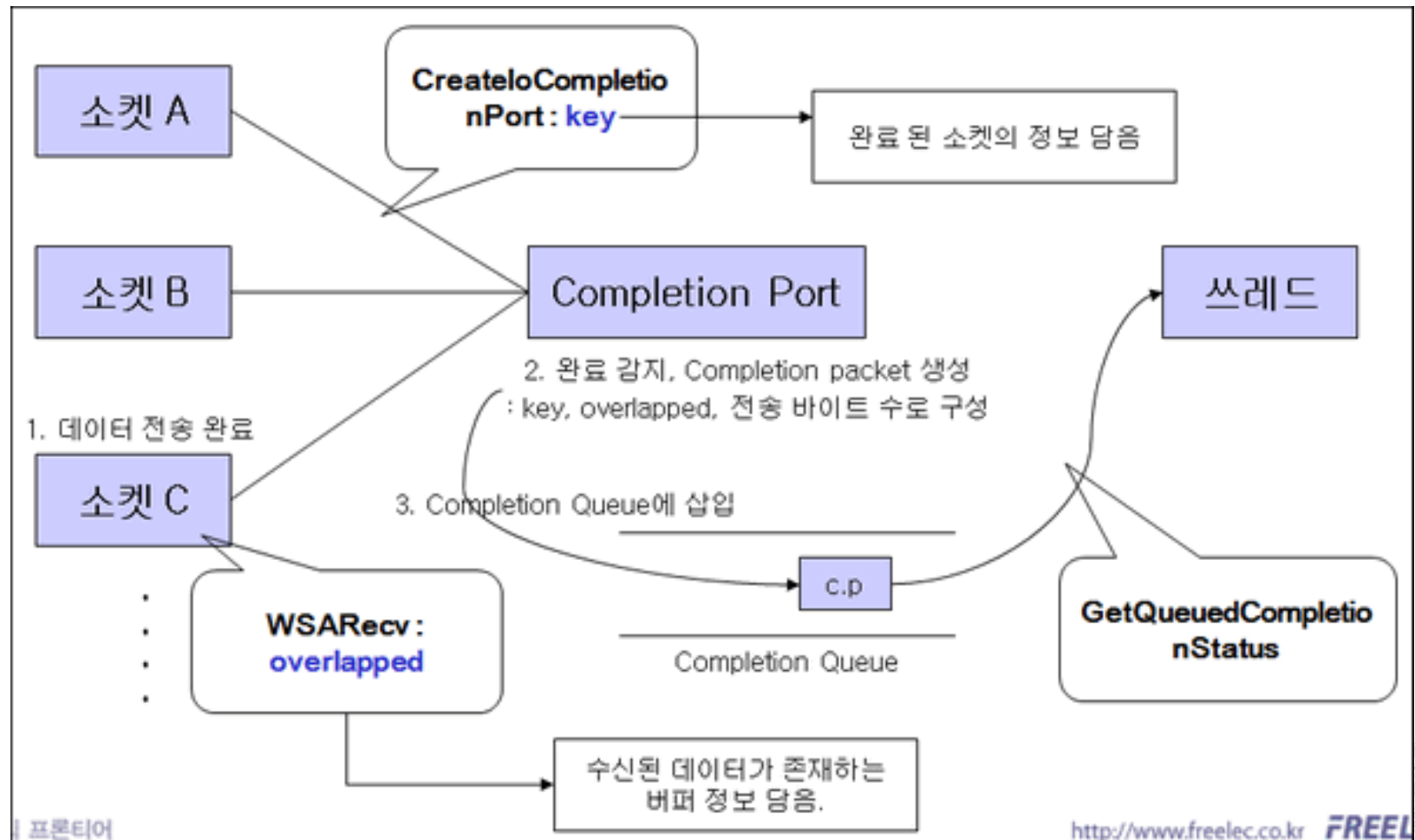
Completion Port 모델 (5)

❖ Completion Port 모델 동작 원리(계속)



Completion Port 모델 (5)

❖ Completion Port 모델 동작 원리(계속)



❖ Completion Port 모델을 이용한 소켓 입출력 절차

- ① **CreateloCompletionPort()** 함수를 호출해 입출력 완료 포트 생성
- ② CPU 개수에 비례하여 작업자 스레드를 생성. 모든 작업자 스레드는 **GetQueuedCompletionStatus()** 함수를 호출하여 대기 상태가 됨
- ③ 비동기 입출력을 지원하는 소켓을 생성. 소켓에 대한 비동기 입출력 결과가 입출력 완료 포트에 저장되려면 **CreateloCompletionPort()** 함수를 호출해 소켓과 입출력 완료 포트를 연결해야 함
- ④ 비동기 입출력 함수를 호출. 비동기 입출력 작업이 곧바로 완료되지 않으면 소켓 함수는 **SOCKET_ERROR**를 리턴하고 오류 코드는 **WSA_IO_PENDING**으로 설정됨
- ⑤ 비동기 입출력 작업이 완료되면 운영체제는 입출력 완료 포트에 결과를 저장하고 대기 중인 스레드 하나를 깨움. 대기 상태에서 깨어난 작업자 스레드는 비동기 입출력 결과를 처리
- ⑥ 새로운 소켓을 생성하면 ③~⑤를, 그렇지 않으면 ④~⑤를 반복



❖ 입출력 완료 포트 생성

- 새로 IOCP 핸들을 만들거나 이미 만들어져 있는 IOCP에 파일 핸들을 연결할 때 사용
- Completion Port 오브젝트를 생성하기 위해서는 HANDLE이나 Key 값은 불필요하고, NumberOfConcurrentThreads 값만 제대로 입력해 주면 된다.

```
HANDLE CreateIoCompletionPort (  
    HANDLE FileHandle,  
    HANDLE ExistingCompletionPort,  
    ULONG CompletionKey,  
    DWORD NumberOfConcurrentThreads  
);
```

성공: 입출력 완료 포트 핸들, 실패: NULL

- FileHandle: CompletionPort에 연결하고자 하는 Overlapped 소켓 핸들. INVALID_HANDLE_VALUE 를 지정하면 새로운 IOCP 핸들을 생성.
- ExistingCompletionPort: 이미 생성된 CompletionPort 핸들
- CompletionKey: 연결되는 소켓에 관한 정보를 스레드에 전달하는 용도
- NumberOfConcurrentThreads: 동시 실행 가능한 스레드 수를 입력, 0을 입력하면 시스템 CPU 수와 일치하게 됨



❖ 비동기 입출력 결과 확인

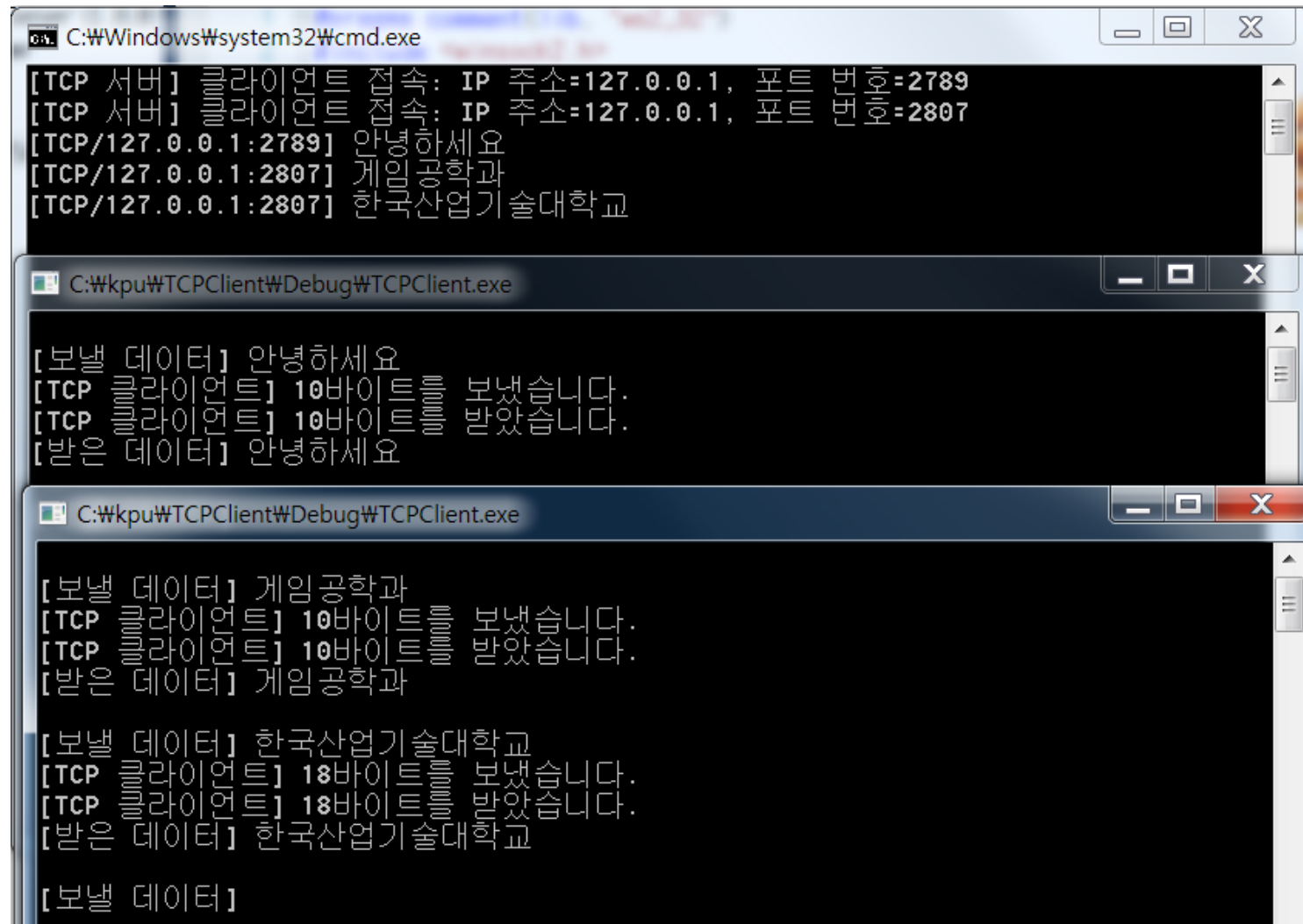
```
BOOL GetQueuedCompletionStatus (  
    HANDLE CompletionPort,  
    LPDWORD lpNumberOfBytes,  
    LPDWORD lpCompletionKey,  
    LPOVERLAPPED *lpOverlapped,  
    DWORD dwMilliseconds  
);
```

성공: 0이 아닌 값, 실패: 0

- CompletionPort: 입출력 완료 포트 핸들
- lpNumberOfBytes: 비동기 입출력 작업으로 전송된 바이트 수가 여기에 저장
- lpCompletionKey: CreateCompletionPort() 함수 호출시 전달한 세 번째 인자(32비트)가 여기에 저장
- lpOverlapped: 비동기 입출력 함수 호출시 전달한 OVERLAPPED 구조체의 주소값이 여기에 저장
- dwMilliseconds: 작업자 스레드가 대기할 시간을 설정. 예) INFINITE



실습 11-3 P457~



The image shows three overlapping Windows command prompt windows. The top window is titled 'C:\Windows\system32\cmd.exe' and displays the following text:

```
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=2789  
[TCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=2807  
[TCP/127.0.0.1:2789] 안녕하세요  
[TCP/127.0.0.1:2807] 게임공학과  
[TCP/127.0.0.1:2807] 한국산업기술대학교
```

The middle window is titled 'C:\wkpu\TCPClient\Debug\TCPClient.exe' and displays the following text:

```
[보낼 데이터] 안녕하세요  
[TCP 클라이언트] 10바이트를 보냈습니다.  
[TCP 클라이언트] 10바이트를 받았습니다.  
[받은 데이터] 안녕하세요
```

The bottom window is also titled 'C:\wkpu\TCPClient\Debug\TCPClient.exe' and displays the following text:

```
[보낼 데이터] 게임공학과  
[TCP 클라이언트] 10바이트를 보냈습니다.  
[TCP 클라이언트] 10바이트를 받았습니다.  
[받은 데이터] 게임공학과  
  
[보낼 데이터] 한국산업기술대학교  
[TCP 클라이언트] 18바이트를 보냈습니다.  
[TCP 클라이언트] 18바이트를 받았습니다.  
[받은 데이터] 한국산업기술대학교  
  
[보낼 데이터]
```

❖ Select 모델

- 모든 윈도우 버전은 물론 유닉스에서도 사용할 수 있으므로 이식성이 높다.

❖ WSAAsyncSelect 모델

- 소켓 이벤트를 윈도우 메시지 형태로 처리하므로, GUI 애플리케이션과 잘 결합 할 수 있다.

❖ WSAEventSelect 모델

- Select 모델과 WSAAsyncSelect 모델의 특성을 혼합한 형태로, 비교적 뛰어난 성능을 제공하면서 윈도우를 필요로 하지 않는다.

❖ Overlapped 모델(I)

- WSAEventSelect 모델과 비슷하지만 비동기 입출력을 통해 성능이 뛰어나다.

❖ Overlapped 모델(II)

- 비동기 입출력을 통해 성능이 뛰어나다. (APC Queue)

❖ Completion Port 모델 (IOCP)

- 비동기 입출력과 완료포트를 통해 가장 뛰어난 성능을 제공한다.



❖ Select 모델

- 하위 호환성을 위해 존재하며, 성능은 여섯 가지 모델 중 가장 떨어진다
- 64개 이상의 소켓을 처리하려면 여러개의 스레드를 사용해야 한다.

❖ WSAAsyncSelect 모델

- 하나의 윈도우 프로시저에서 일반 윈도우 메시지와 소켓 메시지를 처리해야 하므로 성능저하의 요인이 된다.

❖ WSAEventSelect 모델

- 64개 이상의 소켓을 처리하려면 여러 개의 스레드를 사용해야 한다.

❖ Overlapped 모델(I)

- 64개 이상의 소켓을 처리하려면 여러 개의 스레드를 사용해야 한다.

❖ Overlapped 모델(II)

- 모든 비동기 소켓 함수에 대해 완료 루틴을 사용 할 수 있는 것은 아니다.

❖ Completion Port 모델

- 윈도우 가장 단순한 소켓 입출력 방식(블로킹 소켓 + 스레드)와 비교하면 코딩이 복잡하지만 성능면에서 특별한 단점이 없다.
- NT계열에서만 사용할 수 있다.



❖ 소켓 함수 호출 시 블로킹을 최소화한다.

- 여섯가지 모델 모두 만족한다.

❖ 입출력 작업을 다른 작업과 병행한다.

- 비동기 입출력 방식을 사용하는 Overlapped 모델(I), Overlapped 모델(II), Completion Port 모델만 만족한다.

❖ 스레드 개수를 최소화한다.

- 여섯가지 모델 모두 어느정도 만족한다.
- 64개 이상의 소켓을 만들시에는 WSAAsyncSelect 모델, Overlapped 모델(II), Completion Port 모델을 제외하고는 스레드를 추가로 생성한다.
- 하지만 WSAAsyncSelect 모델은 스레드 개수가 늘어나면 성능이 상당히 떨어져서 Overlapped 모델(II), Completion Port 모델만이 조건을 만족한다. (IOCP는 CPU 개수에 비례하여 작업자 스레드를 생성할 수 있으므로 가장 이상적이다.)

❖ 유저모드와 커널모드 전환 횟수와 데이터 복사를 최소화한다.

- 비동기 입출력 방식을 사용하는 모델만 이 조건을 만족한다.
- 비동기 입출력을 할때 송신버퍼나 수신버퍼가 가득차면, 윈도우 운영체제는 애플리케이션 버퍼를 잠근후(lock), 이 메모리 영역을 직접 접근한다.
- 따라서 유저영역 ↔ 커널 영역 복사가 불필요하며 모드 전환없이 입출력 작업을 곧바로 이루므로 효율적이다.



경우에 따라서는 다른 소켓 입출력 모델을 사용하는 경우가 있겠지만, 게임의 경우 대부분 IOCP를 사용한다





Thank You !

oasis01@gmail.com / rhqudtn75@nate.com