WINDOW NETWORK -CHAPTER 5-

SOULSEEK

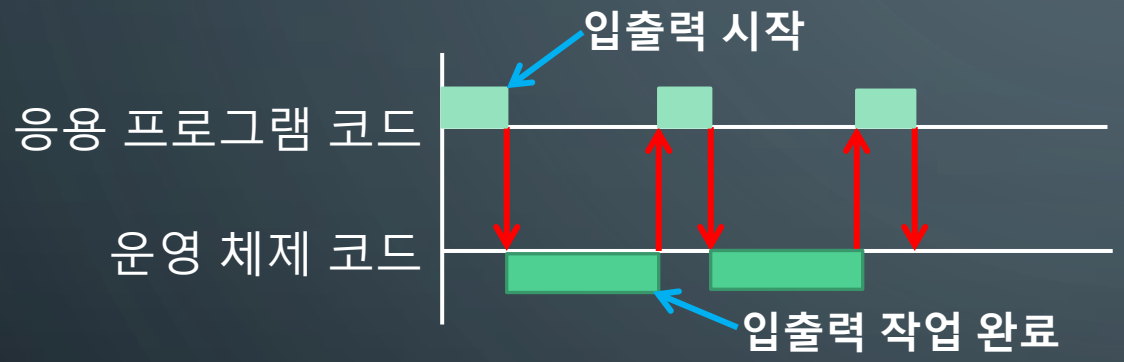
목차

1. OVERLAPPED(1)

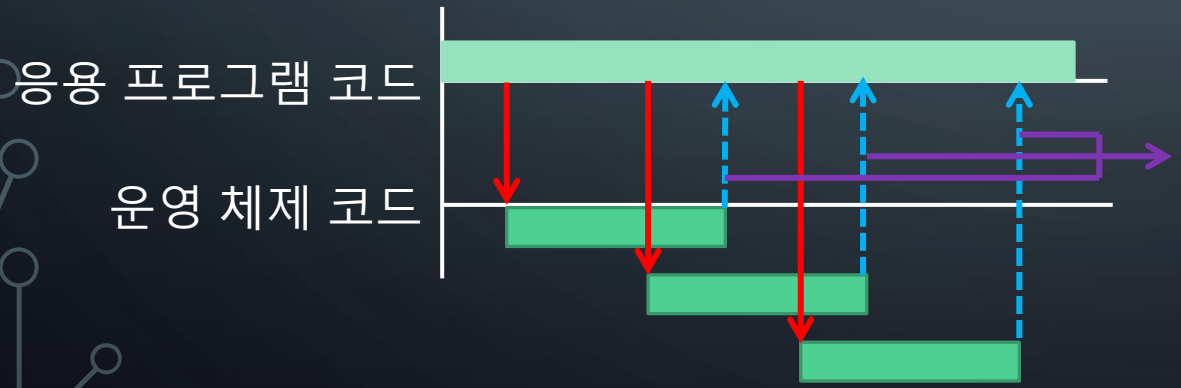
1. OVERLAPPED(1)

- **Window OS**에서 고성능 파일 입출력을 위해 제공하는 방식을 소켓 입출력에서도 사용할 수 있게 만든 것. ➔ 비동기 입출력
- 다른 소켓 모델의 입출력과 다른 모습이며 확실한 성능이 보장된다.

동기 입출력 & 비동기 입출력



동기 입출력
(syschronous I/O)



입출력 작업 완료를
운영체제가 알려줌

비동기 입출력
(asynchronous I/O)

1. OVERLAPPED(1)

동기 입출력(synchronous I/O)

- 응용 프로그램은 입출력 함수를 호출한 후 입출력 작업이 끝날 때까지 대기
- 입출력 작업이 완료되면 입출력 함수 리턴
- 입출력 결과 처리 또는 다른 작업 진행
- **WSAAsyncSelect**가 동기 입출력에 속하고 이전까지 배웠던 것들은 모두 동기 입출력이다.
  - **OS**가 입출력 시점을 운영체제가 알려주기 때문에 조금 쉽게 접근.
  - 이를 비동기 통지**(asynchronous notification)**라고 부른다

비동기 입출력(asynchronous I/O)

- 응용 프로그램은 입출력 함수를 호출한 후 입출력 작업의 완료 여부와 무관하게 다른 작업을 진행
- 입출력 작업이 끝나면 **OS**는 작업 완료를 응용 프로그램에 알려준다.
- 응용 프로그램은 통지가 오면 하던 작업을 잠시 멈추고 결과처리를 진행한다.
- **Overlapped, Completion Port(IOCP)**가 비동기 입출력에 속한다.
  - 통지를 받기 때문에 비동기 통지 형태이다.

1. OVERLAPPED(1)

Overlapped 모델의 공통 절차

- 비동기 입출력을 지원하는 소켓을 생성한다.
  - **socket()**함수로 생성한 소켓은 기본적으로 비동기 입출력을 지원한다.

- 비동기 입출력을 지원하는 소켓 함수를 호출한다.
  - **AcceptEx(), ConnectEx(), DisconnectEx(), TransmitFile(), TransmitPackets(), WSAIoctl(), WSANSPIoctl(), WSAProviderConfigChange(), WSARecv(), WSARecvFrom(), WSARecvMsg(), WSASend(), WSASendTo()**의 **13**개의 함수를 지원한다.

- **OS**가 입출력 작업 완료를 응용 프로그램에 알려주면**(=** 비동기 통지**),** 응용 프로그램은 입출력 결과를 처리한다.

종류	설명
Overlapped(1)	소켓 입출력 작업이 완료되면**, OS**는 응용 프로그램이 등록해둔 이벤트 객체를 신호 상태로 바꾼다**.** 응용 프로그램은 이벤트 객체를 관찰함으로써 입출력 작업완료를 감지할 수 있다**.**
Overlapped(2)	소켓 입출력 작업이 완료 되면**, OS**는 응용 프로그램이 등록해둔 함수를 자동으로 호출한다**.** 일반적으로 운영체제가 호출하는 응용 프로그램 함수를 호출하는데 **Overlapped**에서는 완료 루틴**(completion routine)**이라 부른다**.**

1. OVERLAPPED(1)

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

int WSARecv(SOCKET s, LPWSABUF lpBuffers, DWORD dwBufferCount, LPDWORD lpNumberOfBytesRecvd, LPDWORD lpFlags, LPWSAOVERLAPPED lpOverlapped, LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine)

- 성공 : **0,** 실패 : **SOCKET_ERROR**

  - **첫 번째 인자 :** 비동기 입출력을 할 대상 소켓
  - **두 번째 인자 : WSABUF** 구조체 배열의 시작 주소와 배열의 원소 개수**,** 각 배열 원소는 버퍼의 시작 주소와 길이를 담고 있다**.**
  - **세 번째 인자 :** 함수 호출이 성공하면 보내거나 받은 바이트 수 저장**.**
  - **네 번째 인자 :** 옵션으로 **MSG_*** 형태의 상수를 전달할 수 있다**,** 각각 **send()**와 **recv()**함수의 마지막 인자와 같은 기능을 한다**.** 대부분 **0**을 보냄
  - **다섯 번째 인자 : WSAOVERLAPPED** 구조체의 주소 값
  - **여섯 번째 인자 :** 입출력 작업이 완료 되면 **OS**가 자동으로 호출할 완료 루틴**(**콜백함수**)**의 주소 값이다**.**

1. OVERLAPPED(1)

WSABUF 구조체

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

WSAOVERLAPPED 구조체

- 비동기 입출력을 위한 정보를 **OS**에 전달하거나**, OS**가 비동기 입출력 결과를 응용 프로그램에 알려줄때 사용한다**.**
- **Overlapped** 입출력을 사용한다고 알려주는 역활

```
Typedef struct __WSAOVERLAPPED{
    DWORD Internal;       //OS가 내부적으로 사용
    DWORD InternalHigh;   //OS가 내부적으로 사용
    DWORD Offset;         //OS가 내부적으로 사용
    DWORD OffsetHigh;     //OS가 내부적으로 사용
    WSAEVENT hEvent;      // 이벤트 객체의 핸들 값, Overlapped(1)에서만 사용.
} WSAOVERLAPPED, *LPWSAOVERLAPPED;
```

1. OVERLAPPED(1)

WSASend()와 WSARecv()함수의 특징

1. 송수신 측에서 **WSABUF** 구조체 배열을 사용하면, 여러 버퍼에 저장된 데이터를 모아서 보내거나 받을 수 있다

```
//송신 측 코드
Char buf1[128];
Char buf2[256];
WSABUF wsabuf[2];

Wsabuf[0].buf = buf1;
Wsabuf[1].len = 128;
Wsabuf[1].buf = buf2;
Wsabuf[1].len = 256;

WSASend(sock, wsabuf, 2, ...);

//수신 측 코드
//선언부분은 송신 측과 동일

WSARecv(sock, wsabuf, 2, ...);
```

1. OVERLAPPED(1)

WSASend()와 WSARecv()함수의 특징

2. 마지막 두 인자에 모두 **NULL**값을 사용하면 **send()/recv()**함수처럼 동기 함수로 동작한다**.**

3. Overlapped(1)에서는 **WSAOVERLAPPED** 구조체인 **hEvent** 변수를**, Overlapped(2)**에서는 **lpCompletionRoutine**인자를 사용한다**.** 단**, lpCompletionRoutine**의 우선순위가 높으므로 이 값이 **NULL**이 아니면 **WSAOVERLAPPED** 구조체는 **hEvent** 변수는 사용되지 않는다**.**

Overlapped(1)의 구현 절차

1. 비동기 입출력을 지원하는 소켓을 생성한다**.**
   - **WSACreateEvent()**함수를 호출하여 대응하는 이벤트 객체도 같이 생성한다**.**

2. 비동기 입출력을 지원하는 소켓 함수를 호출한다**.**
   - **WSAOVERLAPPED** 구조체의 **hEvent** 변수에 이벤트 객체의 핸들 값을 넣어서 전달한다**.** 비동기 입출력 작업이 곧바로 완료되지 않으면**,** 소켓 함수는 오류를 리턴하고 오류 코드는 **WSA_IO_PENDING**으로 설정되고 작업 완료 후 **OS**는 이벤트 객체를 신호 상태로 만들어 이 사실을 응용 프로그램에 알린다**.**

3. WSAWaitForMulitpleEvents()함수를 호출하여 이벤트 객체가 신호 상태가 되기를 기다린다**.**
4. 비동기 입출력 작업이 완료하여 **WSAWaitForMultipleEvents()**함수가 리턴하면**,** **WSAGetOverlappedResult()**함수를 호출해 비동기 입출력 결과를 확인하고 데이터를 처리한다**.**
5. 새로운 소켓을 생성하면 **1~4**단계를**,** 그렇지 않으면 **2~4**단계를 반복한다**.**

1. OVERLAPPED(1)

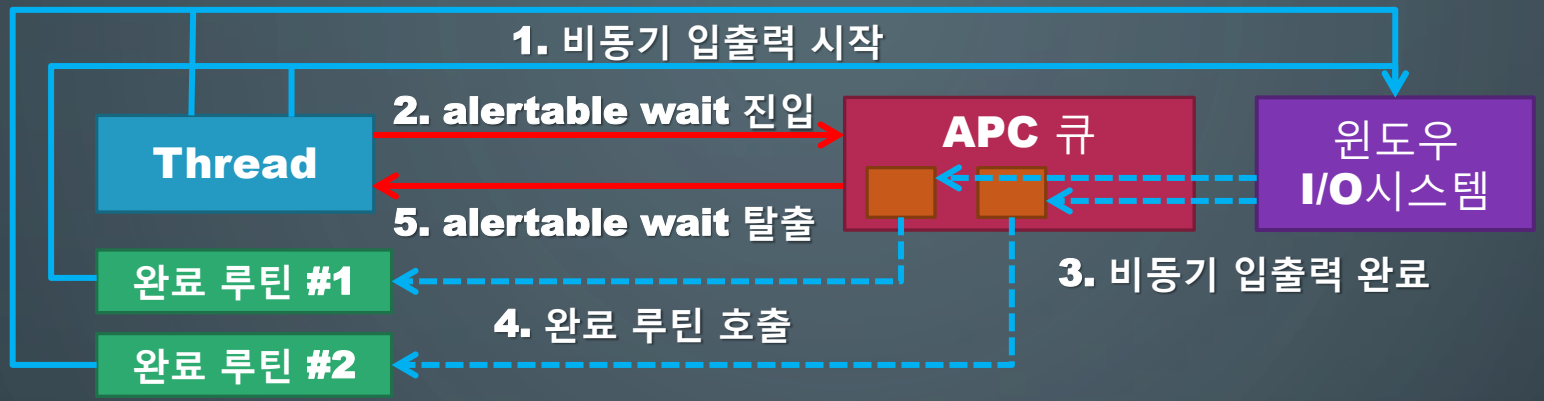
DWORD WSAWaitForMultipleEvent(DWORD cEvents, const WSAEVENT* lphEvents, BOOL dwTimeout, DWORD dwTimeout, BOOL fAlertable);
성공 : **WSA_WAIT_EVENT_0~WSA_WAIT_EVENT_0+cEvent-1** 또는 **WSA_WAIT_TIMEOUT**
실패 : **WSA_WAIT_FAILED**
첫 번째 인자 : 이벤트 객체 핸들의 배열
두 번째 인자 : 이벤트 객체의 수
세 번째 인자 : **TRUE**면 모든 이벤트 객체가 신호 상태가 될 때까지 기다린다. **FALSE**면 이벤트 객체 중 하나가 신호 상태가 되는 즉시 리턴한다.
네 번째 인자 : 대기 시간으로 밀리초 단위를 사용한다. **WSA_INFINITE** 값을 사용하면 무한히 기다린다.
다섯 번째 인자 : 입출력 완료 루틴을 사용할 것인지를 설정한다.

BOOL WSAGetOverlappedResult(SOCKET s, LPWSAOVERLAPPED lpOverlapped, LPDWORD lpcbTransfer, BOOL fWait, LPDWORD lpdwFlags);
성공 : **TRUE**, 실패 : **FALSE**
첫 번째 인자 : 비동기 입출력 함수 호출에 사용했던 소켓
두 번째 인자 : 비동기 입출력 함수 호출에 사용했던 **WSAOVERLAPPED** 구조체
세 번째 인자 : 전송된 바이트 수
네 번째 인자 : 비동기 입출력 작업이 끝날 때까지 대기하려면 **TRUE**, 그렇지 않으면 **FALSE**사용, **WSAWaitForMultipleEvent()** 함수를 이전에 호출해서 리턴했다면 비동기 입출력이 끝났다는 뜻이므로 **FALSE**를 사용.
다섯 번째 인자 : 비동기 입출력 작업과 관련된 부가적인 정보가 저장, 사용하지 않기 때문에 무시해도 된다.

2. OVERLAPPED(2)

2. OVERLAPPED(2)

- 완료 루틴(**Completion routine**)통해 입출력 처리를 한다 ➔ **Overlapped(1)**은 이벤트
- **OS**가 적절한 시점에 자동으로 호출하는 사용자 정의 함수



1. 비동기 입출력 함수를 호출함으로써 **OS**에 입출력 작업을 요청한다.
2. 해당 **Thread**는 곧바로 **alertable wait** 상태에 진입한다.
   - **alertable wait** 상태는 비동기 입출력을 위한 대기상태, 비동기 입출력 함수를 호출한 **Thread**가 이 상태로 되어 있어야 완료 루틴이 호출 될 수 있다. **Thread** 대기 함수들에 약간 변형이 있는 파생함수로 **alertable wait** 사용 여부인자가 추가된 **WaitForSingleObjectEx(), WaitForMultipleObjectEx(), SleepEx(), WSAWaitForMultipleEvnets()**등이 있다. 마지막 인자를 **TRUE**로 하면 **alertable wait** 상태가 된다.

3. 비동기 입출력 작업이 완료되면 **OS**는 **Thread**의 **APC** 큐에 결과를 저장한다.
   - **APC** 큐**(asynchronous procedure call queue)**는 비동기 입출력 결과 저장을 위해 **OS**가 각 **Thread**에 할당하는 메모리.

4. 비동기 입출력 함수를 호출한 **Thread**가 **wait** 상태에 있으면, **OS**는 **APC**큐에 저장된 정보를 참조해서 루틴을 호출
5. **APC**큐에 저장된 정보를 토대로 모든 완료 루틴호출이 끝나면, **Thread**는 **alertable wait**상태에서 빠져 나오고 지속적인 처리를 위해서 다시 **alertable wait** 상태로 진입해야 한다.

2. OVERLAPPED(2)

Overlapped(2) 소켓 입출력 절차

1. 비동기 입출력을 지원하는 소켓을 생성한다.
   - **sock()** 함수로 생성한 소켓은 기본적으로 비동기 입출력을 지원한다.

2. 비동기 입출력을 지원하는 소켓 함수를 호출한다.
   - **Overlapped(1)**에서 사용하지 않던 완료 루틴의 시작 주소(함수이므로)인자에 완료 루틴을 전달한다. 이때, 입출력 작업이 곧바로 완료되지 않으면, 소켓 함수는 **SOCKET_ERROR**를 리턴하고 오류코드는 **WSA_IO_PENDING**으로 설정된다.

3. 비동기 입출력 작업이 완료되면, **OS**는 완료 루틴을 호출한다. 완료 루틴에 서는 비동기 입출력 결과를 확인하고 후속 처리를 한다.
4. 완료 루틴 호출이 모두 끝나면, **Thread**는 **alertable wait**상태에서 빠져나온다.
5. 새로운 소켓을 생성하면 **1~5**단계를, 그렇지 않으면 **2~5**단계를 반복한다.

void CALLBACK CompletionRoutine(DWORD dwError, DWORD cbTransferred, LPWSAOVERLAPPED lpOverlapped, DWORD dwFlags);
첫 번째 인자 : 비동기 입출력 결과, 오류가 발생하면 **0**이 아닌 값이 된다.
두 번째 인자 : 전송 바이트 수, 통신 상대가 접속을 종료하면 이 값은 **0**이 된다.
세 번째 인자 : 비동기 입출력 함수 호출 시 넘겨준 **WSAOVERLAPPED** 구조체의 주소 값
네 번째 인자 : 항상 **0**이므로 적어도 현재까지는 사용하지 않는다.

3. COMPLETION PORT(IOCP)

3. COMPLETION PORT(IOCP)

Completion Port의 동작원리

- **OS**가 제공하는 입출력 완료 포트를 이용한다.
- 입출력 완료 포트**(I/O Completion Port)**는 비동기 입출력 결과와 이 결과를 처리할 **Thread**에 관한 정보를 담고 있는 구조
- **Overlapped(2)**의 **APC** 큐와 비슷한 개념.

ACP 큐와 다른점

생성과 파괴

- **ACP** 큐는 각 **Thread**마다 자동으로 생성되고 파괴된다. 입출력 완료 포트 는 **CreateCompletionPort()** 함수를 호출하여 생성하고 **CloseHandle()** 함수를 호출하여 파괴한다.

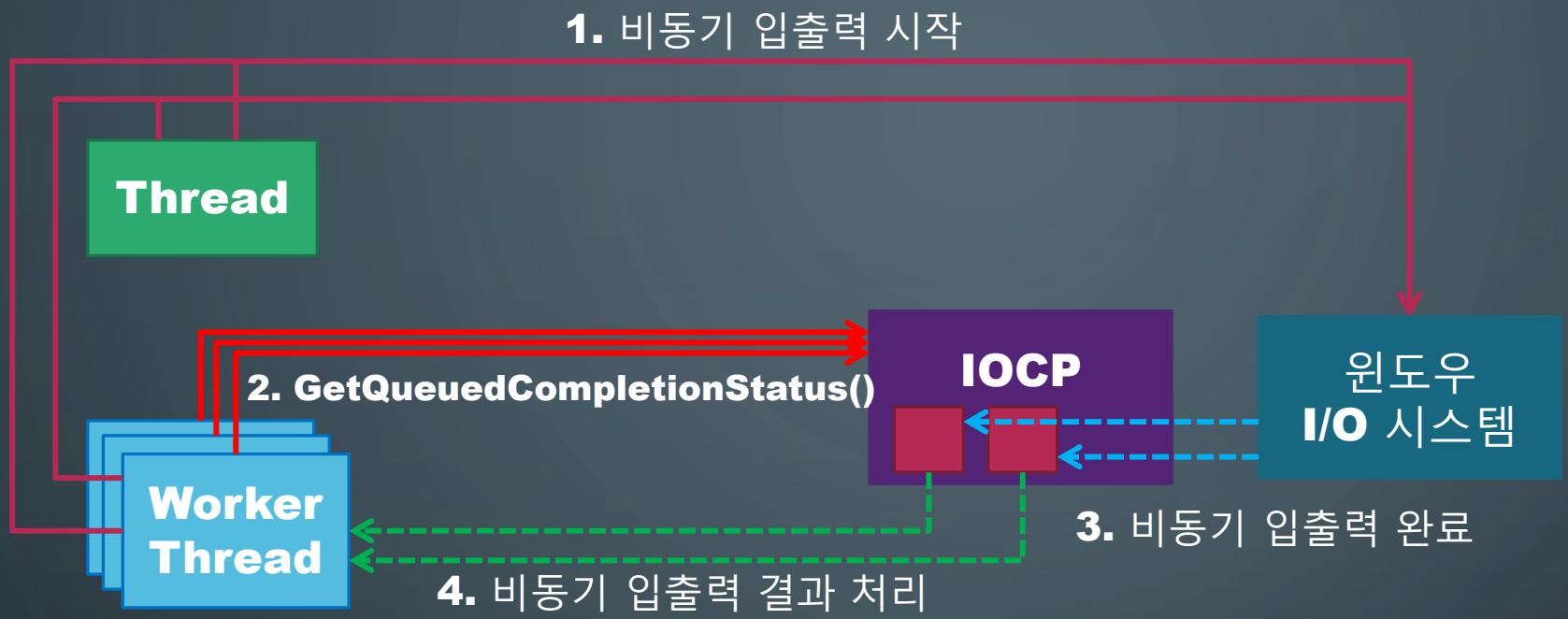
접근제약

- **APC** 큐에 저장된 결과는 **APC** 큐를 소유한 **Thread**만 확인할 수 있지만, 입출력 완료 포트 에는 이런 제약이 없다.
- 입출력 완료 포트 에 접근하는 **Thread**를 별도로 두고, 이를 **Worker Thread**라 부른다
- 이상적인 **Worker Thread** 생성 개수는 **CPU** 개수 * **n**개**(n**의 최소값은 **1)**

비동기 입출력 처리 방법

- **APC** 큐에 저장된 결과를 처리하려면 해당 **Thread**는 **alertable wait** 상태에 진입해야 한다.
- 입출력 완료 포트 에 저장된 결과를 처리하려면 **Worker Thread**는 **GetQueuedCompletionStatus()** 함수를 로출해야 한다.

3. COMPLETION PORT(IOCP)



1. 응용 프로그램을 구성하는 임의의 **Thread**에서 비동기 입출력 함수를 호출함으로써 **OS**에 입출력 작업을 요청한다.
2. 모든 **Worker Thread**는 **GetQueueCompletionStatus()** 함수를 호출에 입출력 완료 포트를 감시한다.
   - 완료한 비동기 입출력 작업이 아직 없다면 모든 **Worker Thread**는 대기 상태가 되고 이때 대기중인 **Worker Thread** 목록은 입출력 완료 포트 내부에 저장된다.

3. 비동기 입출력 작업이 완료되면 **OS**는 입출력 완료 포트에 결과를 저장한다.
   - 저장되는 정보를 입출력 완료 패킷**(I/O completion packet - IOCP)**이라 부른다.

4. **OS**는 입출력 완료 포트에 저장된 **Worker Thread** 목록에서 하나를 선택하여 깨운다.
   - 대기 상태에서 깨어난 **Worker Thread**는 비동기 입출력 결과를 처리한다. 이후 필요에 따라 다시 비동기 입출력 함수를 호출 할 수 있다.

3. COMPLETION PORT(IOCP)

IOCP의 소켓 입출력의 절차

1. **CreateCompletionPort()**함수를 호출하여 입출력 완료 포트 를 생성한다.

2. **CPU** 개수에 비례하여 **Worker Thread**를 생성한다.
   - 모든 **Worker Thread**는 **GetQueuedCompletionStatus()**함수를 호출하여 대기 상태가 된다

3. 비동기 입출력을 지원하는 소켓을 생성한다.
   - 이 소켓에 대한 비동기 입출력 결과가 입출력 완료 포트에 저장되려면, **CreateCompletionPort()** 함수를 호출하여 입출력 완료 포트 를 연결해야 한다.

4. 비동기 입출력 함수를 호출한다.
   - 비동기 입출력 작업이 곧바로 완료되지 않으면, 소켓 함수는 **SOCKET_ERROR**를 리턴하고 오류 코드는 **WSA_IO_PENDING**으로 설정된다.

5. 비동기 입출력 작업이 완료되면, **OS**는 입출력 완료 포트 에 결과를 저장하고, 대기 중인 **Thread** 하나를 깨운다.
   - 대기 상태에서 깨어난 **WorkerThread**는 비동기 입출력 결과를 처리한다.

6. 새로운 소켓을 생성하면 **3~5**단계를, 그렇지 않으면 **4~5**단계를 반복한다.

3. COMPLETION PORT(IOCP)

HANDLE CreateIOCompletionPort(HANDLE FileHandle, HANDLE ExistingCompletionPort, ULONG CompletionKey, DWORD NumberOfConcurrentThreads);

- 성공 : 연결할 입출력 완료 포트 핸들, 실패 : **NULL**
- **첫 번째 인자 :** 연결할 입출력 완료 포트 와 연결할 파일 핸들
  - 소켓 프로그래밍에서는 소켓 디스트립터를 넣어주면 된다. 새로운 연결할 입출력 완료 포트 를 생성할 때는 유효한 핸들 대신 **INVALID_HANDLE_VALUE**값을 사용해도 된다.
- **두 번째 인자 :** 파일 또는 소켓과 연결할 입출력 완료 포트 핸들.
  - 이 값이 **NULL**이면 새로운 입출력 완료 포트를 생성한다.
- **세 번째 인자 :** 입출력 완료 패킷**(IOCP)**에 들어갈 부가 정보로 **32**비트 값을 줄 수 있다.
  - **IOCP**은 비동기 입출력 작업이 완료될 때마다 생성되어 입출력 완료 포트에 저장되는 정보다.
- **네 번째 인자 :** 동시에 실행할 수 있는 **Worker Thread**의 개수.
  - **0**을 사용하면 자동으로 **CPU** 개수와 같은 수로 설정된다. **OS**는 실행 중인 **Worker Thread** 개수가 여기서 설정한 값을 넘지 않도록 관리해준다.

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

- 성공 : **0**이 아닌 값, 실패 : **0**
- **첫 번째 인자 :** 입출력 완료 포트 핸들
- **두 번째 인자 :** 비동기 입출력 작업으로 전송된 바이트 수가 저장.
- **세 번째 인자 : CreateIoCompletionPort()**함수 호출 시 전달한 세 번째 인자가 여기에 들어간다.
- **네 번째 인자 :** 비동기 입출력 함수 호출 시 전달한 **OVERLAPPED** 구조체의 주소 값이 저장된다.
- **다섯 번째 인자 : Worker Thread**가 대기할 시간을 밀리초 단위로 저장한다.
  - **INFINITE** 값을 넣으면 **IOCP**이 생성되어 **OS**가 깨울 때까지 무한히 대기한다.