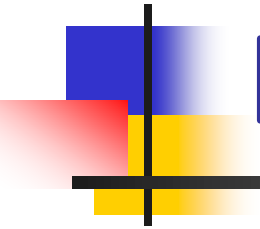


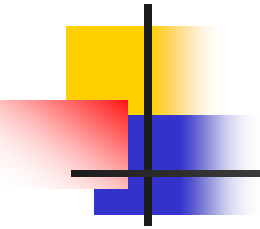
IOCP

편집: 홍익대학교(세종)
김혜영

동작 원리

IOCP의 목적

IO 작업에서 동시 수행되는 스레드 개수의 상한을 설정하자! 그래서 CPU 자원을 최대한 효율적으로 사용하자!

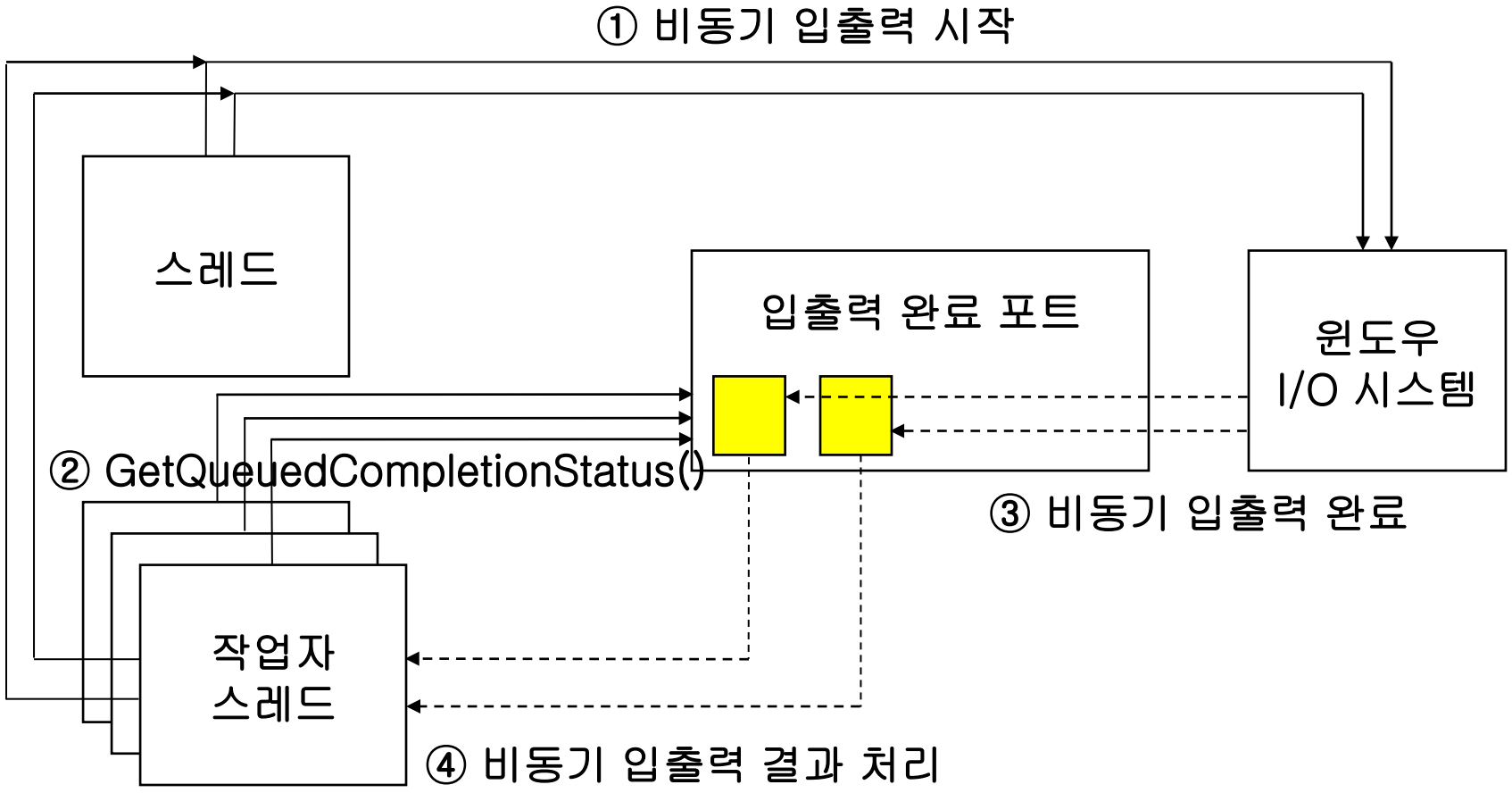


Completion Port 모델 (1/7)

- 입출력 완료 포트(I/O completion port)
  - 비동기 입출력 결과와 이 결과를 처리할 스레드에 대한 정보를 담고 있는 구조
- 입출력 완료 포트 vs. APC 큐
  - 생성과 파괴
  - 접근 제약
  - 비동기 입출력 처리 방법

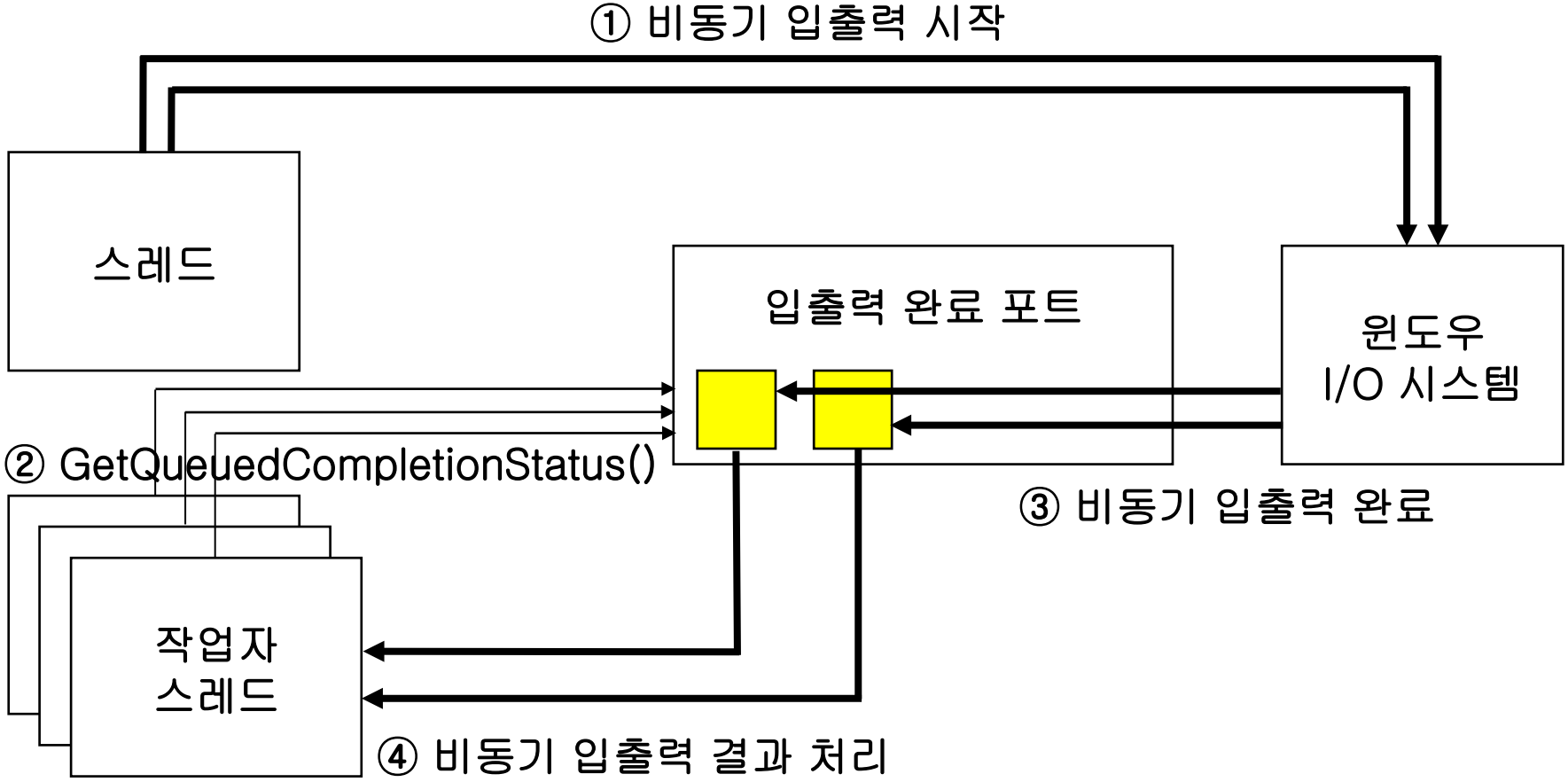
Completion Port 모델 (2/7)

- Completion Port 모델 동작 원리



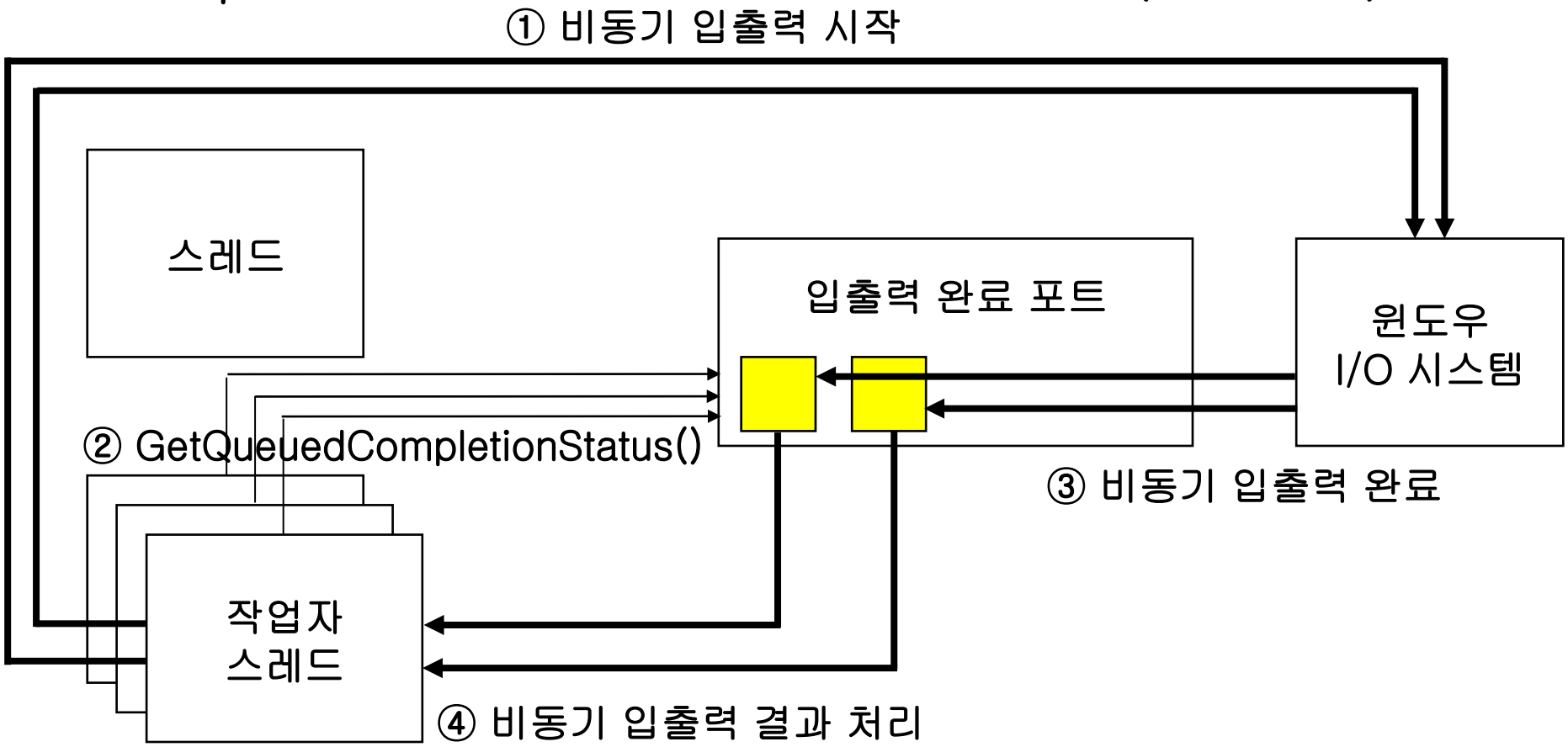
Completion Port 모델 (3/7)

- Completion Port 모델 동작 원리 (cont'd)



Completion Port 모델 (4/7)

- Completion Port 모델 동작 원리 (cont'd)



Completion Port 모델 (5/7)

- Completion Port 모델을 이용한 소켓 입출력 절차
  ① CreateIoCompletionPort() 함수를 호출하여 입출력 완료 포트를 생성한다.
  ② CPU 개수에 비례하여 작업자 스레드를 생성한다. 모든 작업자 스레드는 GetQueuedCompletionStatus() 함수를 호출하여 대기 상태가 된다.
  ③ 비동기 입출력을 지원하는 소켓을 생성한다. 이 소켓에 대한 비동기 입출력 결과가 입출력 완료 포트에 저장되려면, CreateIoCompletionPort() 함수를 호출하여 소켓과 입출력 완료 포트를 연결시켜야 한다.
  ④ 비동기 입출력 함수를 호출한다. 비동기 입출력 작업이 곧바로 완료되지 않으면, 소켓 함수는 오류를 리턴하고, 오류 코드는 WSA_IO_PENDING으로 설정된다.
  ⑤ 비동기 입출력 작업이 완료되면, 운영체제는 입출력 완료 포트에 결과를 저장하고, 대기 중인 스레드 하나를 깨운다. 대기 상태에서 깨어난 작업자 스레드는 비동기 입출력 결과를 처리한다.
  ⑥ 새로운 소켓을 생성하면 ③~⑤를, 그렇지 않으면 ④~⑤를 반복한다.

Completion Port 모델 (6/7)

- 입출력 완료 포트 생성

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

성공: 입출력 완료 포트 핸들, 실패: NULL

동작 원리

IOCP 생성

당연히 IOCP를 쓰려면 먼저 IOCP를 만들어야한다.

원형

```
CreateIoCompletionPort(                    // IOCP와 연결할 핸들. 첫 생성시는
HANDLE fileHandle,
                                           // INVALID_HANDLE_VALUE 넘김
  HANDLE ExistingCompletionPort, //IOCP 핸들. 역시 첫 생성시는 NULL.
  ULONG_PTR CompletionKey,  //IO 완료시 넘어갈 값. 사용자가 넘기고 싶은 값 넘김.
  DWORD NumberOfConcurrentThreads //한 번에 동작할 수 있는 최대 스레드 개수.
  //0 넘기면 프로세서 숫자로 자동 지정됨.
```

동작 원리

IO 장치와 IOCP 연결 - 마찬가지로 CreateIoCompletionPort를 쓴다.

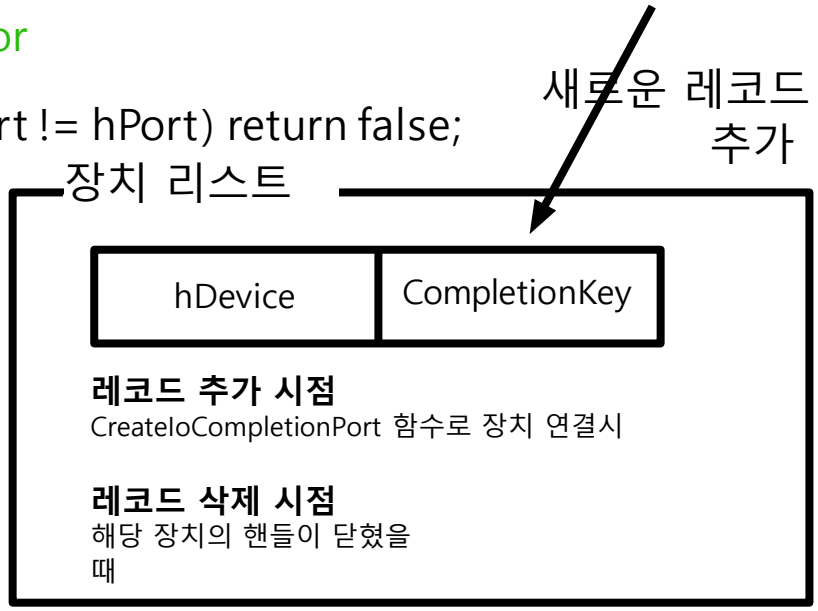
HANDLE

HANDLE hPort = CreateIoCompletionPort(INVALID_HANDLE_VALUE,NULL,0,0);

HANDLE port = CreateIoCompletionPort(socket,hPort,(ULONG_PTR)id,0);

//error

if(port != hPort) return false;



CreateIoCompletionPort를 호출하면 내부적으로 IOCP와 연결된 여러 장 치들을 관리하기 위한 장치 리스트에 새 로운 레코드를 추가한다.

동작 원리

장치와 연결이 끝나면 IO 작업이 완료된 후 완료된 IO에 대한 처리를 수행 할 스레드 풀 구성. 스레드 풀의 크기는 프로세서 개수의 2배 정도 할당

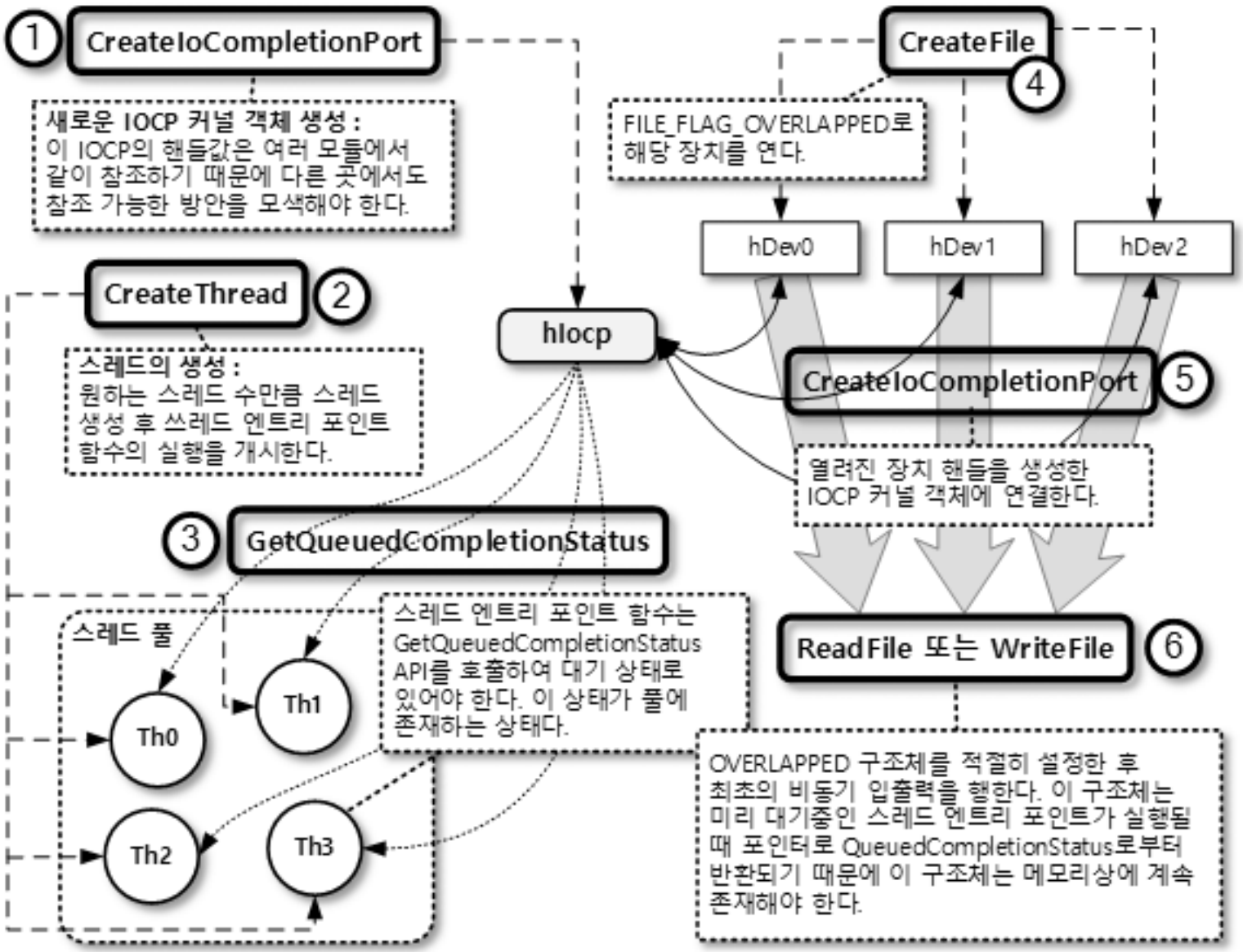
```
for(int i = 0; i < ioThreadCount; i++)
{
DWORD ThreadId;
HANDLE hThread = (HANDLE)_beginthreadex(NULL,0, workerThread,
(LPVOID)i,0,(unsinged int)&dwThreadId);
}
```

Completion Port 모델 (7/7)

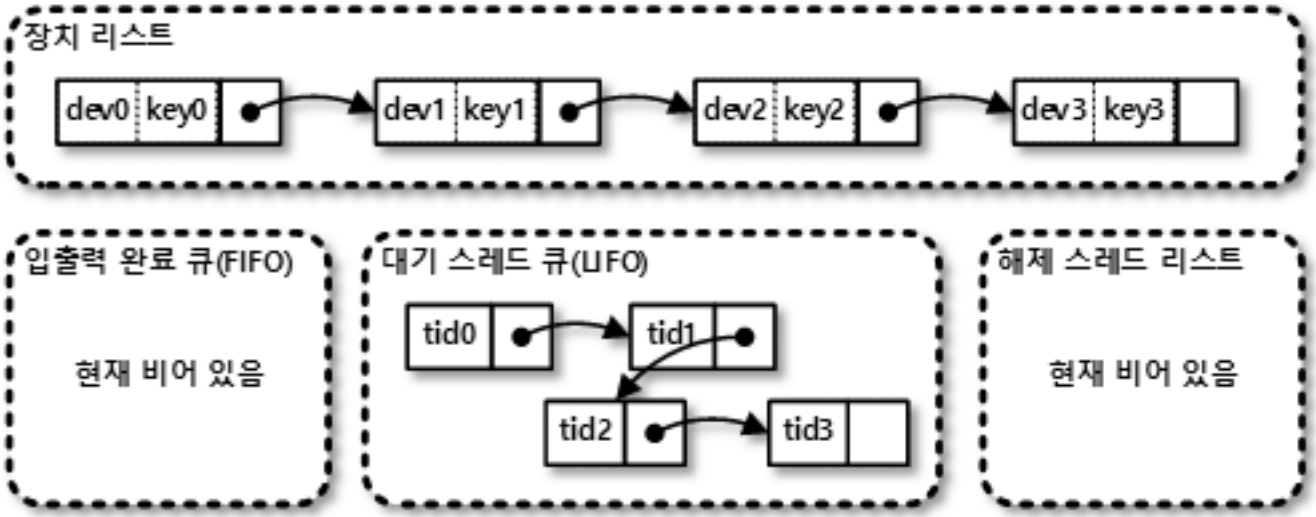
- 비동기 입출력 결과 확인

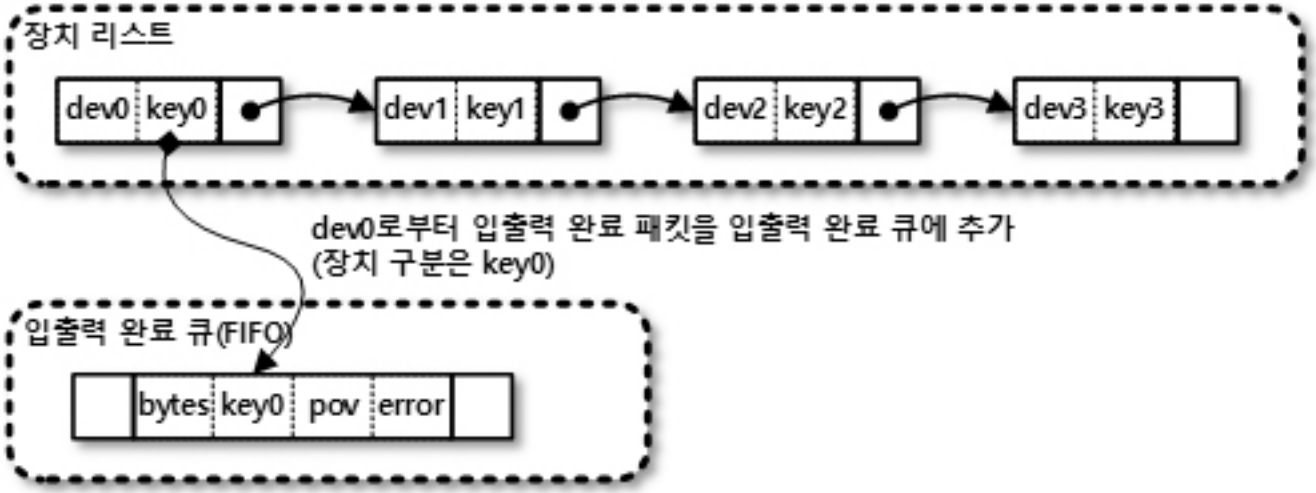
```
BOOL GetQueuedCompletionStatus (
    HANDLE CompletionPort,
    LPDWORD lpNumberOfBytes,
    LPDWORD lpCompletionKey,
    LPOVERLAPPED* lpOverlapped,
    DWORD dwMilliseconds
) ;
```

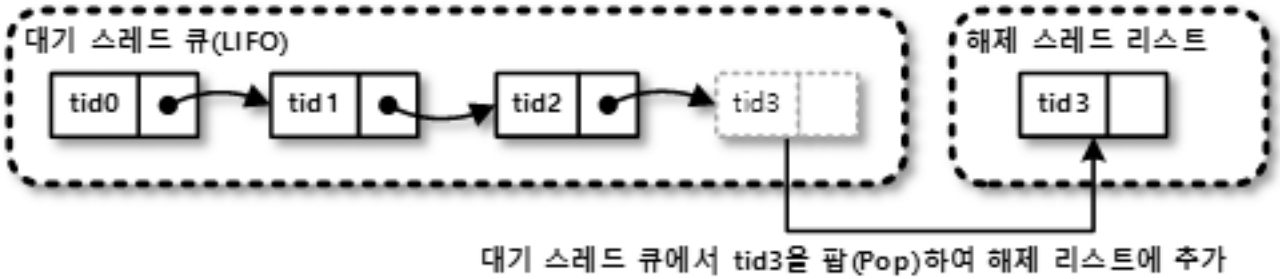
성공: 0이 아닌 값, 실패: 0

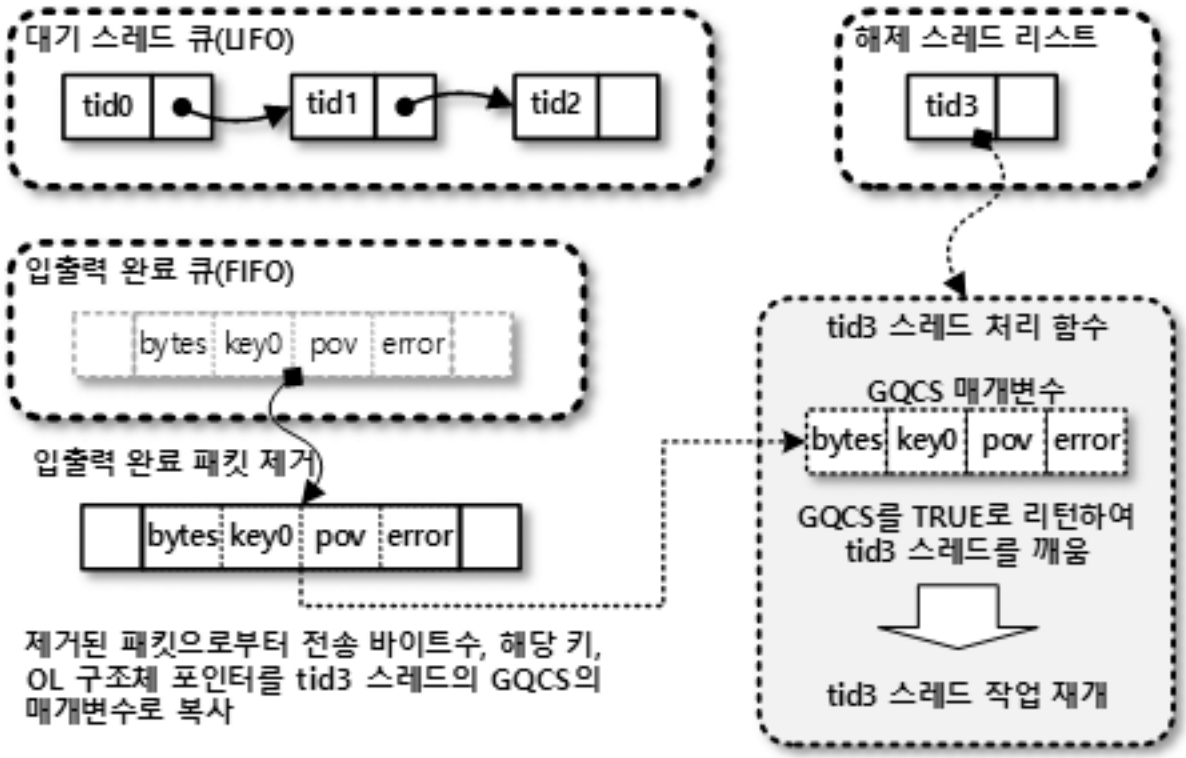

1
CreateIoCompletionPort
새로운 IOCP 커널 객체 생성 :
이 IOCP의 핸들값은 여러 모듈에서
같이 참조하기 때문에 다른 곳에서도
참조 가능한 방안을 모색해야 한다.
CreateFile
4
FILE_FLAG_OVERLAPPED로
해당 장치를 연다.
hDev0
hDev1
hDev2
CreateThread
2
스레드의 생성 :
원하는 스레드 수만큼 스레드
생성 후 쓰레드 엔트리 포인트
함수의 실행을 개시한다.
hIocp
CreateIoCompletionPort
5
열려진 장치 핸들을 생성한
IOCP 커널 객체에 연결한다.
3
GetQueuedCompletionStatus
스레드 풀
Th0
Th1
Th2
Th3
스레드 엔트리 포인트 함수는
GetQueuedCompletionStatus
API를 호출하여 대기 상태로
있어야 한다. 이 상태가 풀에
존재하는 상태다.
ReadFile 또는 WriteFile
6
OVERLAPPED 구조체를 적절히 설정한 후
최초의 비동기 입출력을 행한다. 이 구조체는
미리 대기중인 스레드 엔트리 포인트가 실행될
때 포인터로 QueuedCompletionStatus로부터
반환되기 때문에 이 구조체는 메모리상에 계속
존재해야 한다.

IOCP 초기 상태

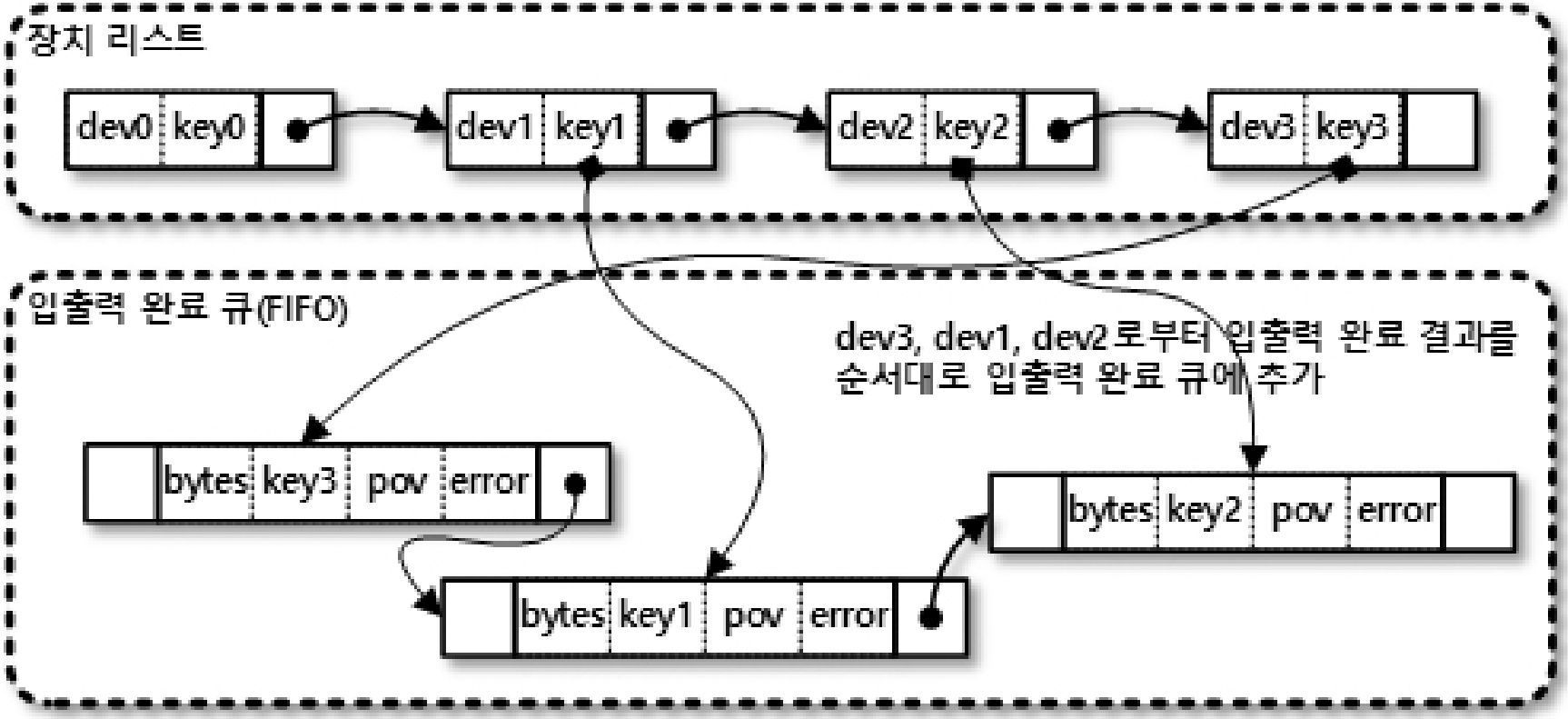

장치 리스트
dev0
key0
dev1
key1
dev2
key2
dev3
key3
입출력 완료 큐(FIFO)
현재 비어 있음
대기 스레드 큐(LIFO)
tid0
tid1
tid2
tid3
해제 스레드 리스트
현재 비어 있음

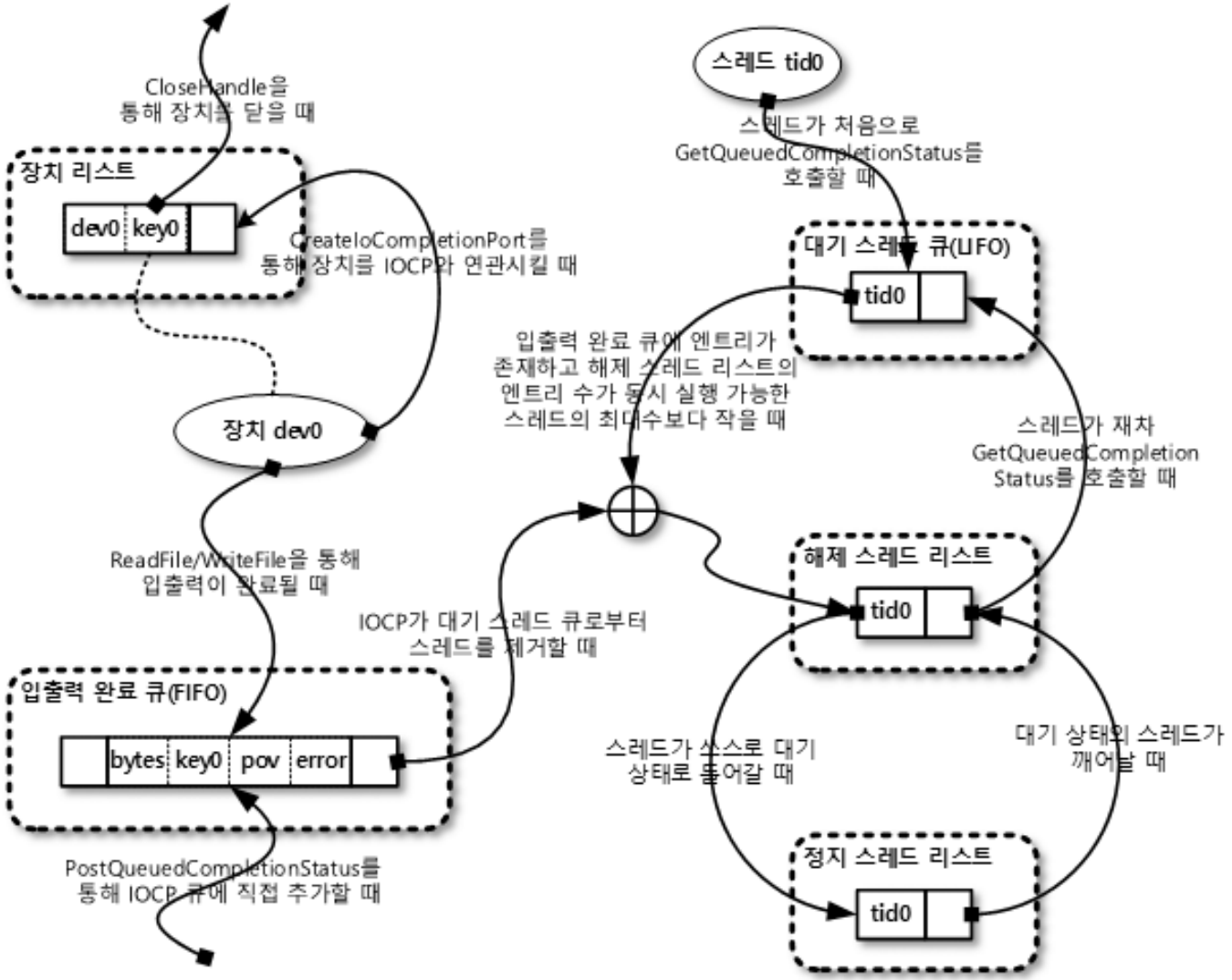

장치 리스트
dev0
key0
dev1
key1
dev2
key2
dev3
key3
dev0로부터 입출력 완료 패킷을 입출력 완료 큐에 추가
(장치 구분은 key0)
입출력 완료 큐(FIFO)
bytes
key0
pov
error


대기 스레드 큐(LIFO)
tid0
tid1
tid2
tid3
해제 스레드 리스트
tid3
대기 스레드 큐에서 tid3을 팝(Pop)하여 해제 리스트에 추가


대기 스레드 큐(LIFO)
tid0
tid1
tid2
해제 스레드 리스트
tid3
입출력 완료 큐(FIFO)
bytes
key0
pov
error
tid3 스레드 처리 함수
GQCS 매개변수
bytes
key0
pov
error
입출력 완료 패킷 제거
bytes
key0
pov
error
GQCS를 TRUE로 리턴하여
tid3 스레드를 깨움
제거된 패킷으로부터 전송 바이트수, 해당 키,
OL 구조체 포인터를 tid3 스레드의 GQCS의
매개변수로 복사
tid3 스레드 작업 재개

여러 장치에서의 입출력

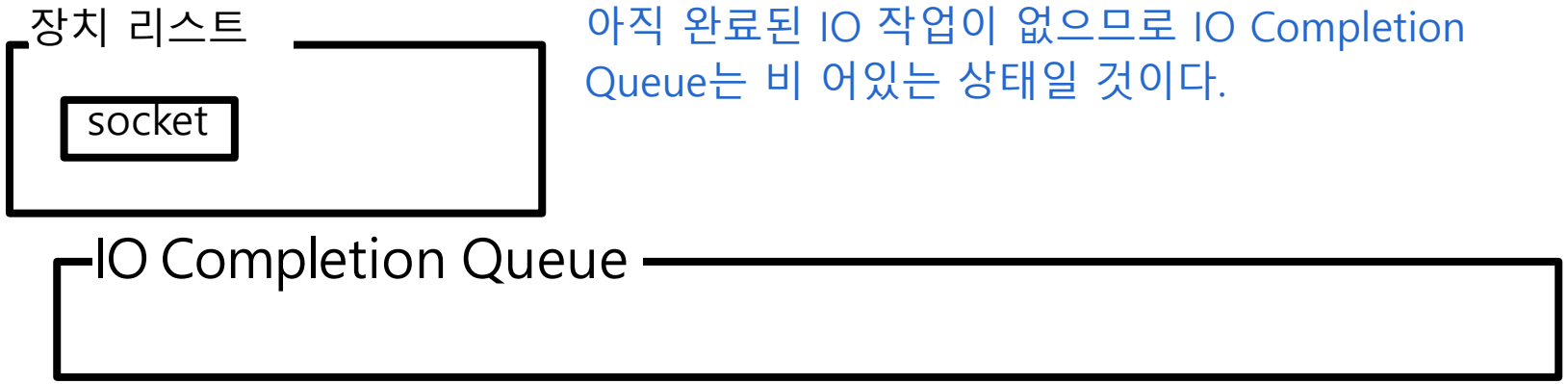

장치 리스트
dev0
key0
dev1
key1
dev2
key2
dev3
key3
입출력 완료 큐(FIFO)
dev3, dev1, dev2로부터 입출력 완료 결과를
순서대로 입출력 완료 큐에 추가
bytes
key3
pov
error
bytes
key1
pov
error
bytes
key2
pov
error


CloseHandle을
통해 장치를 닫을 때
장치 리스트
dev0
key0
CreateIoCompletionPort를
통해 장치를 IOCP와 연관시킬 때
장치 dev0
ReadFile/WriteFile을 통해
입출력이 완료될 때
입출력 완료 큐(FIFO)
bytes
key0
pov
error
PostQueuedCompletionStatus를
통해 IOCP 큐에 직접 추가할 때
IOCP가 대기 스레드 큐로부터
스레드를 제거할 때
스레드 tid0
스레드가 처음으로
GetQueuedCompletionStatus를
호출할 때
대기 스레드 큐(LIFO)
tid0
입출력 완료 큐에 엔트리가
존재하고 해제 스레드 리스트의
엔트리 수가 동시 실행 가능한
스레드의 최대수보다 작을 때
스레드가 재차
GetQueuedCompletion
Status를 호출할 때
해제 스레드 리스트
tid0
스레드가 스스로 대기
상태로 들어갈 때
대기 상태의 스레드가
깨어날 때
정지 스레드 리스트
tid0

예시

프로세서가 2개인 컴퓨터에서 socket과 연결된 IOCP가 하나 있 다고 가정할 때 IOCP의 동작을 통해 구조를 이해해보자.

예시

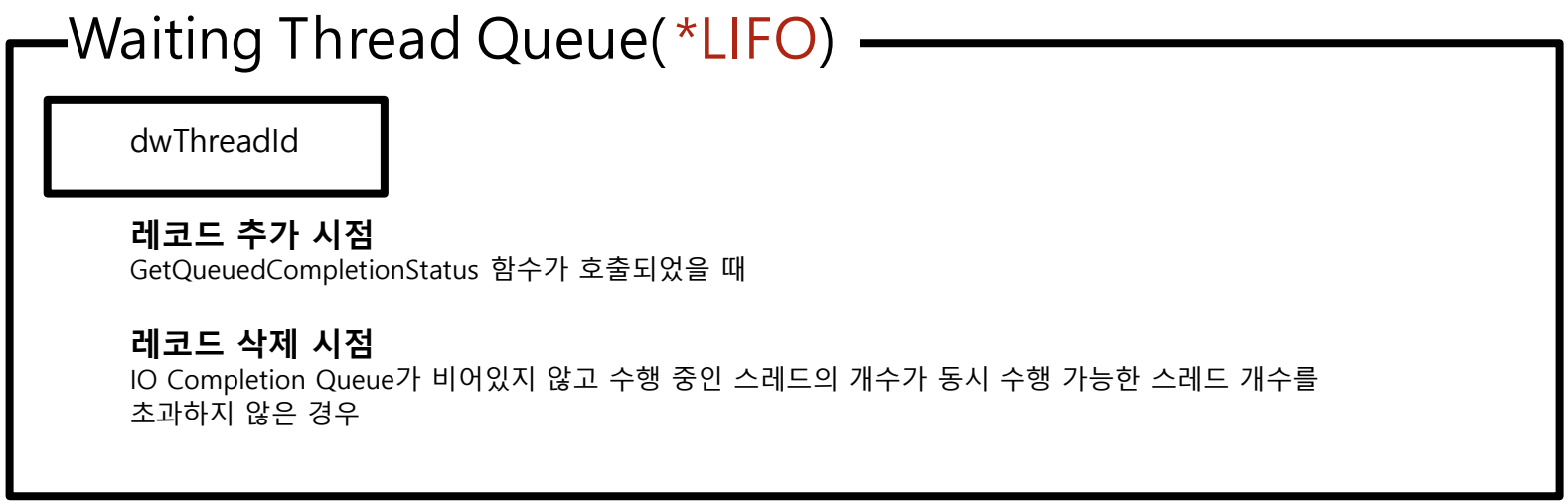


아직 완료된 IO 작업이 없으므로 IO Completion Queue는 비 어있는 상태일 것이다.

동작 원리

즉 이 큐는 IOCP와 연결한 device의 IO 작업이 끝났음을 알려주는 큐이다. 방금 전 생성한 스레드 풀에서 적절한 스레드들이 IO Completion Queue 에서 작업거리를 꺼내 IO 완료에 따른 처리를 수행하게 된다.

동작 원리



이 큐는 IO 작업의 완료를 처리하기 위해 대기하고 있는 스레드들의 큐이다. GetQueuedCompletionStatus 함수를 호출했을 때 당장 IO Completion Queue에 항목이 없거나 동시 수행 가능한 스레드 개수를 초과한 경우 대기 큐에 들어가 대기하고 있다가 적당한 상황이 오면 깨어나서 IO 완료에 대한 처리를 수행하게 된다.

동작 원리

Release Thread List

dwThreadId

레코드 추가 시점

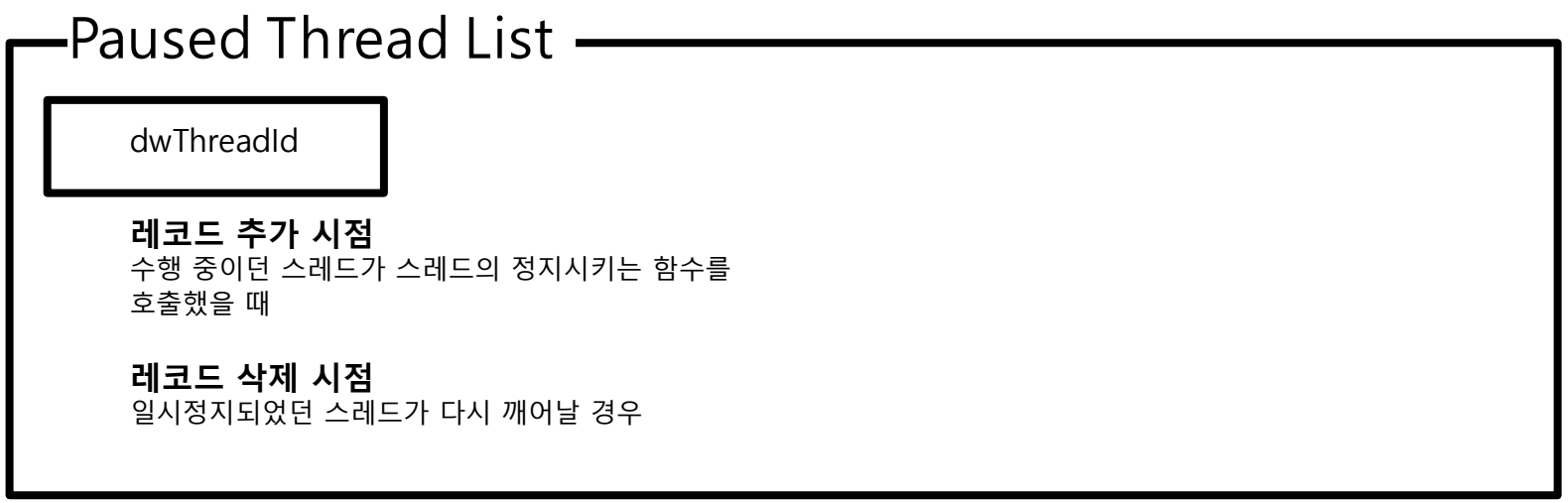
IO Completion Port가 Waiting Thread Queue에 있는 스레드를 깨우는 경우 일시정지되었던 스레드가 다시 깨어났을 경우

레코드 삭제 시점

스레드가 다시 GetQueuedCompletionStatus 함수를 호출했을 때 스레드가 정지되는 함수를 호출했을 때

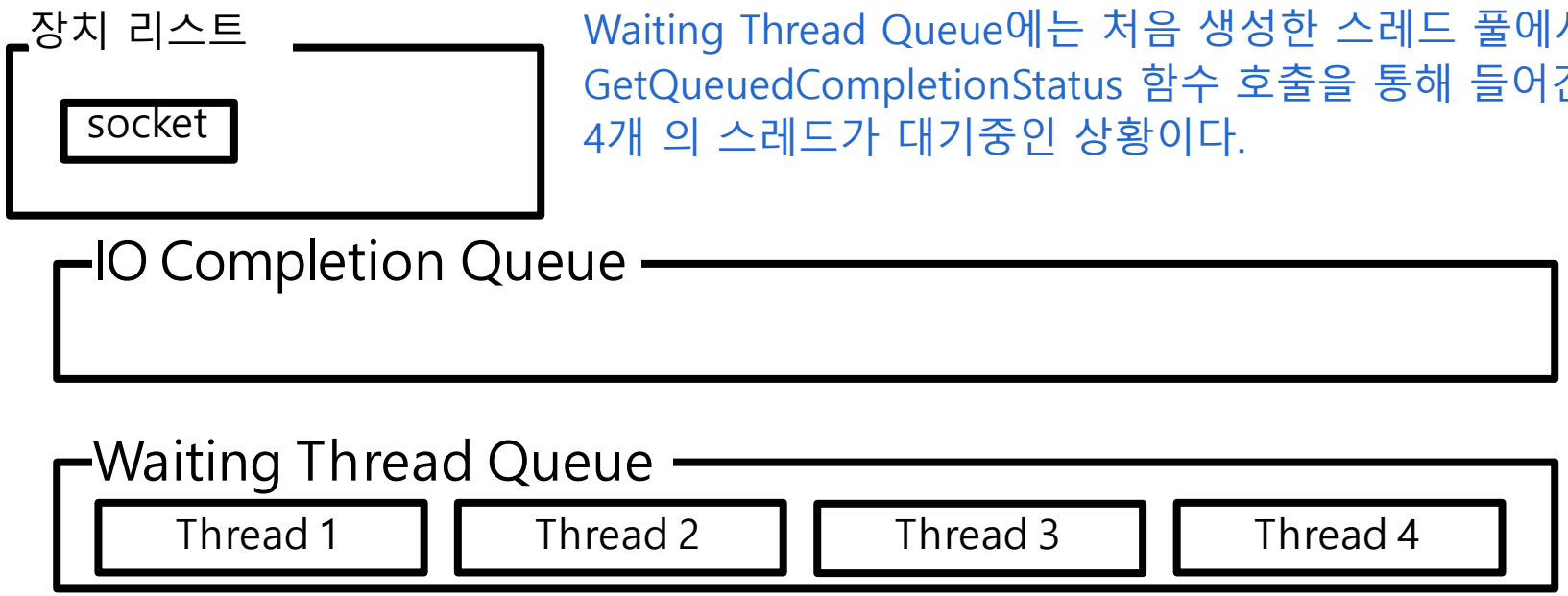
- 실제 IO 완료에 따른 처리를 수행하기 위해 깨어나서 돌고 있는 Thread 들의 리스트이다. IOCP에서 지정한 개수만큼의 스레드만 이 리스트에 속할 수 있다(그 만큼만 깨어있을 수 있다).

동작 원리



작업을 처리하던 스레드가 특정 함수의 호출(스레드를 블락시키는 함수 등) 했을 때 이 스레드를 Paused Thread List로 보낸다.

예시



Waiting Thread Queue에는 처음 생성한 스레드 풀에서 GetQueuedCompletionStatus 함수 호출을 통해 들어간 4개 의 스레드가 대기중인 상황이다.

예시

장치 리스트

socket

아직 완료된 IO 작업이 없으니 Release Thread List와 Pause Thread List도 비어있는 상태.

IO Completion Queue

Waiting Thread Queue

Thread 1 | Thread 2 | Thread 3 | Thread 4

Release Thread List

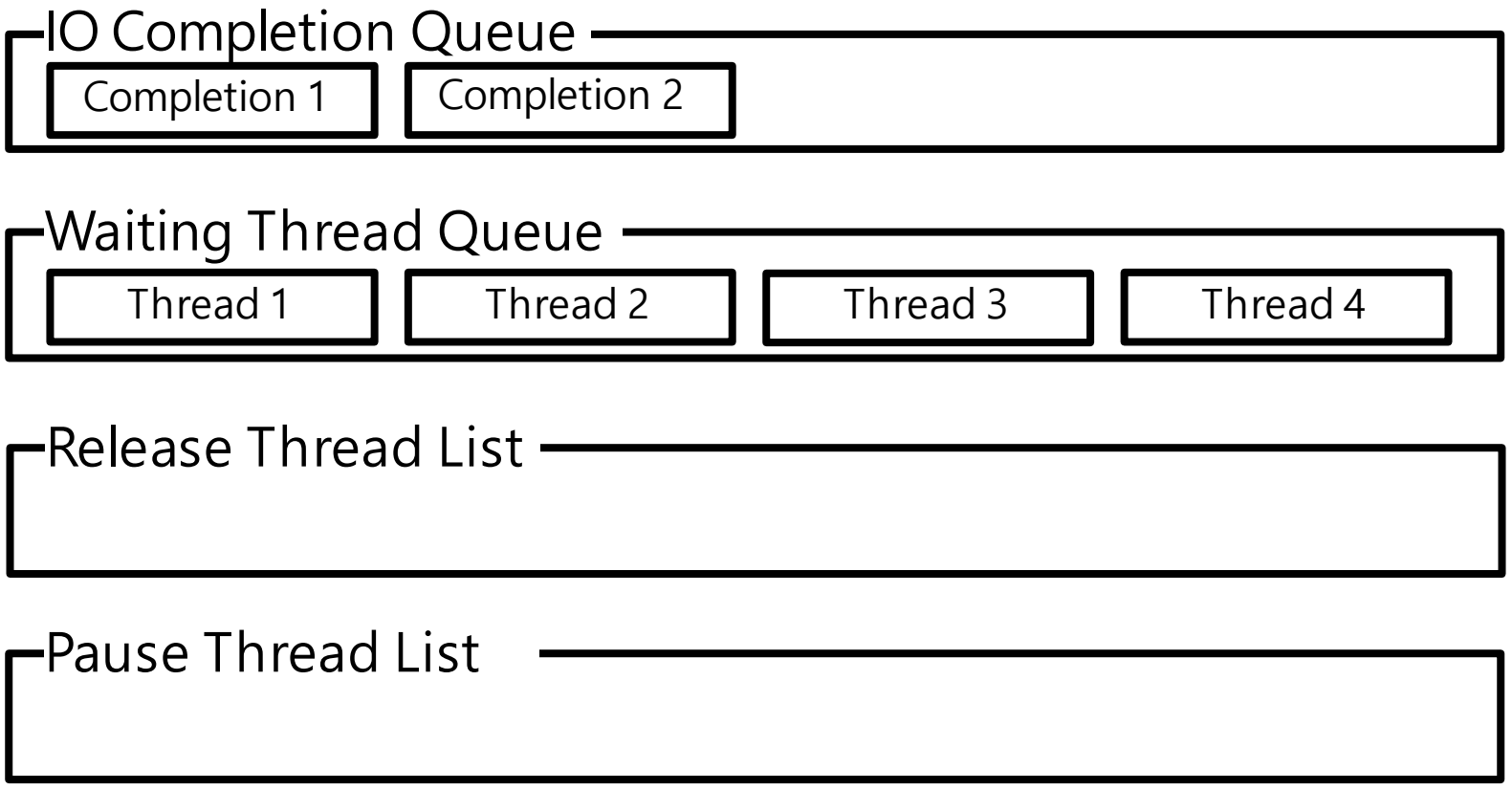
Pause Thread List

예시

장치 리스트

socket

이 때 2개의 IO 작업이 완료되었다고 하자.



예시

장치 리스트

socket

Release Thread List에 2개가 돌고 있을 때, 또 다시 새로운 IO 작업이 완료되었다고 하자. 이 때 이미 Relase Thread List에 2 개의 Thread가 돌고 있기 때문에 새로운 스레드는 깨우지 않는 다.

IO Completion Queue

Completion 3 | Completion4

Waiting Thread Queue

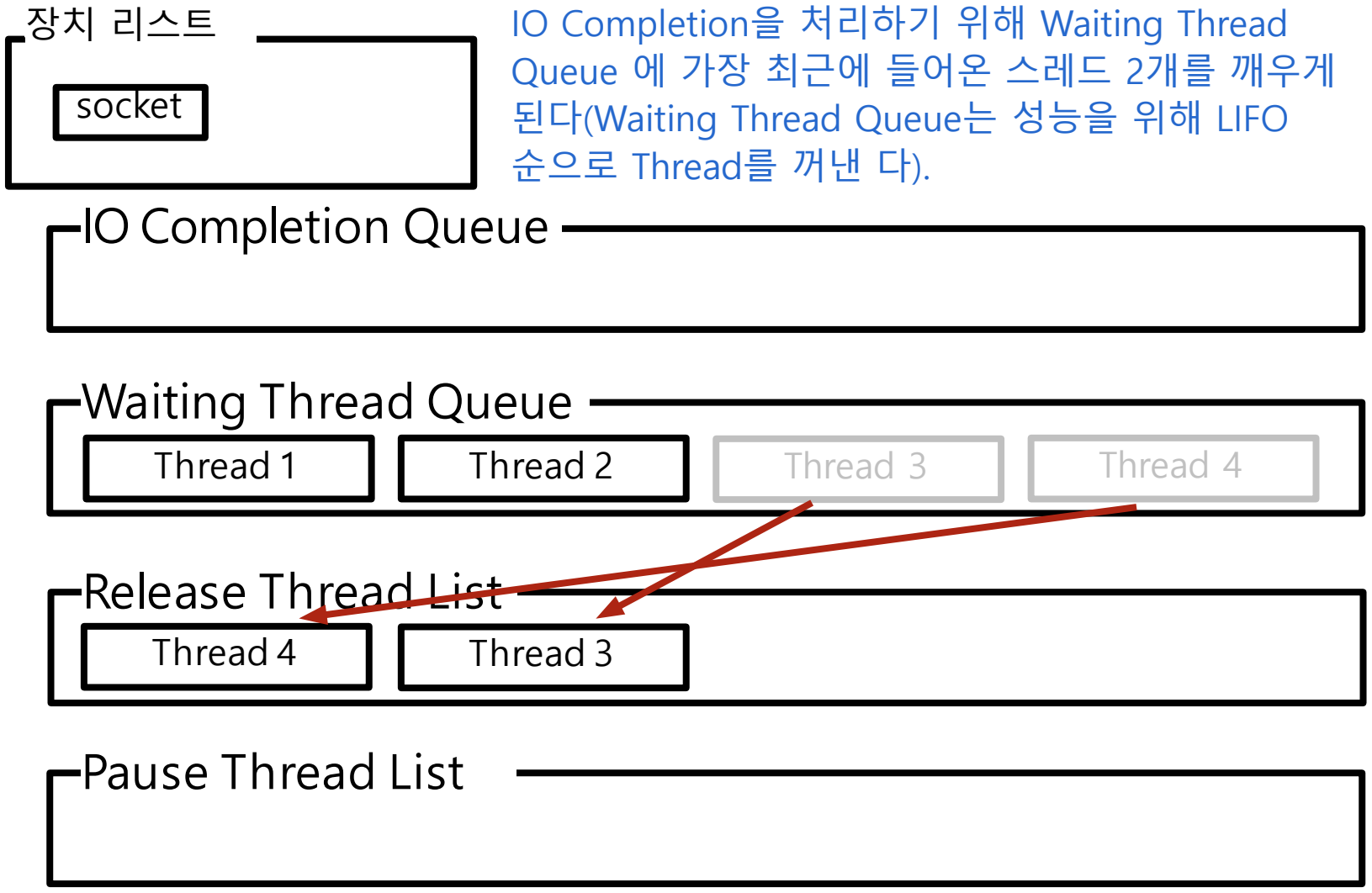
Thread 1 | Thread 2

Release Thread List

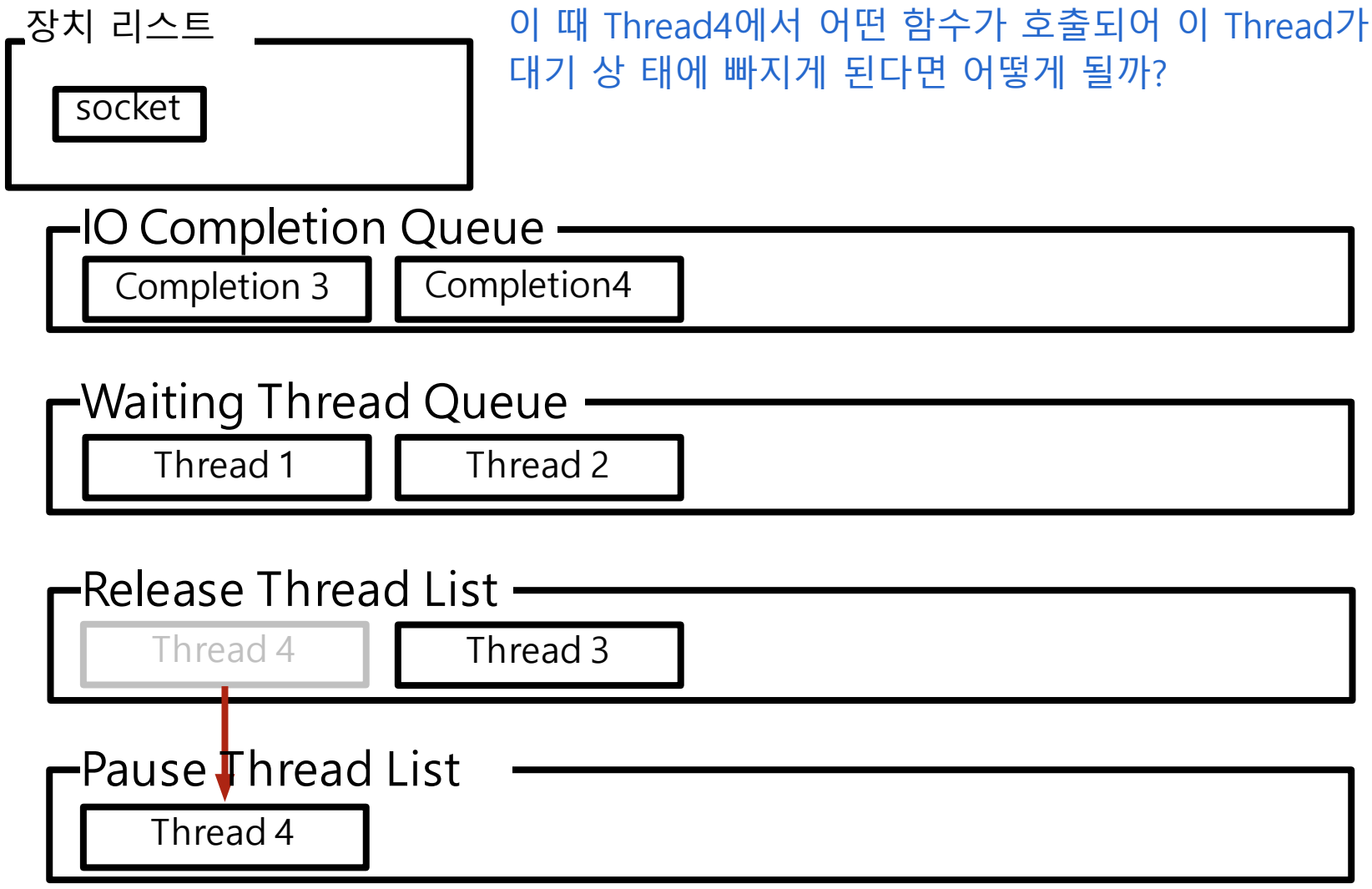
Thread 4 | Thread 3

Pause Thread List

예시

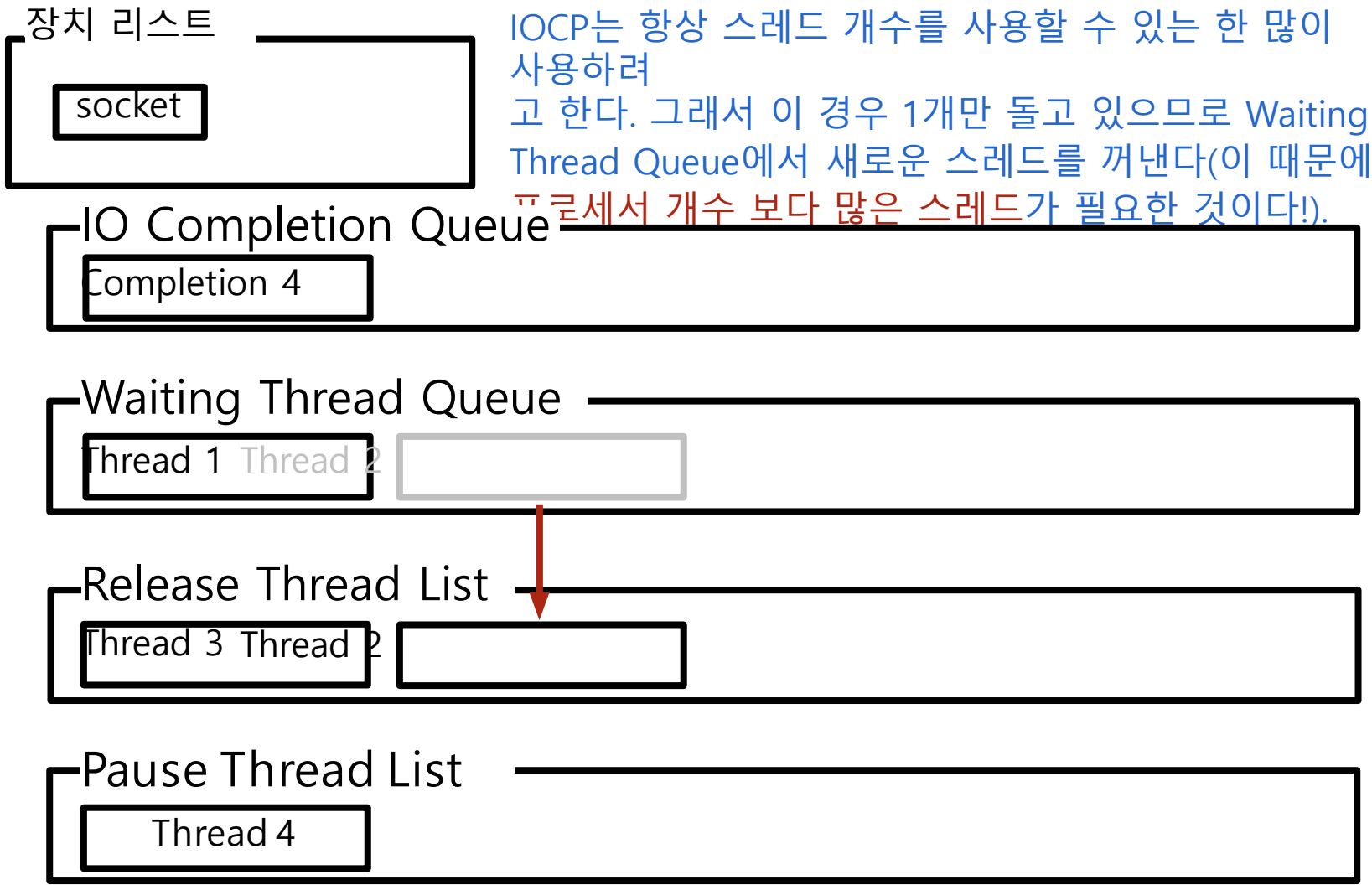

장치 리스트
socket
IO Completion을 처리하기 위해 Waiting Thread Queue 에 가장 최근에 들어온 스레드 2개를 깨우게 된다(Waiting Thread Queue는 성능을 위해 LIFO 순으로 Thread를 꺼낸 다).
IO Completion Queue
Waiting Thread Queue
Thread 1
Thread 2
Thread 3
Thread 4
Release Thread List
Thread 4
Thread 3
Pause Thread List

예시



이 때 Thread4에서 어떤 함수가 호출되어 이 Thread가 대기 상 태에 빠지게 된다면 어떻게 될까?

예시


장치 리스트
socket
IOCP는 항상 스레드 개수를 사용할 수 있는 한 많이 사용하려
고 한다. 그래서 이 경우 1개만 돌고 있으므로 Waiting Thread Queue에서 새로운 스레드를 꺼낸다(이 때문에 프로세서 개수 보다 많은 스레드가 필요한 것이다!).
IO Completion Queue
Completion 4
Waiting Thread Queue
Thread 1
Thread 2
Release Thread List
Thread 3
Thread 2
Pause Thread List
Thread 4

예시

장치 리스트

socket

그런데 이 상태에서 Pause Thread List의 스레드가 대기 상태 의 작업이 끝난다면? 이 걸 그냥 깨우면 Release Thread List의 최대 Thread 개수를 초과하게 되어버린다.

IO Completion Queue

Completion 4

Waiting Thread Queue

Thread 1

Release Thread List

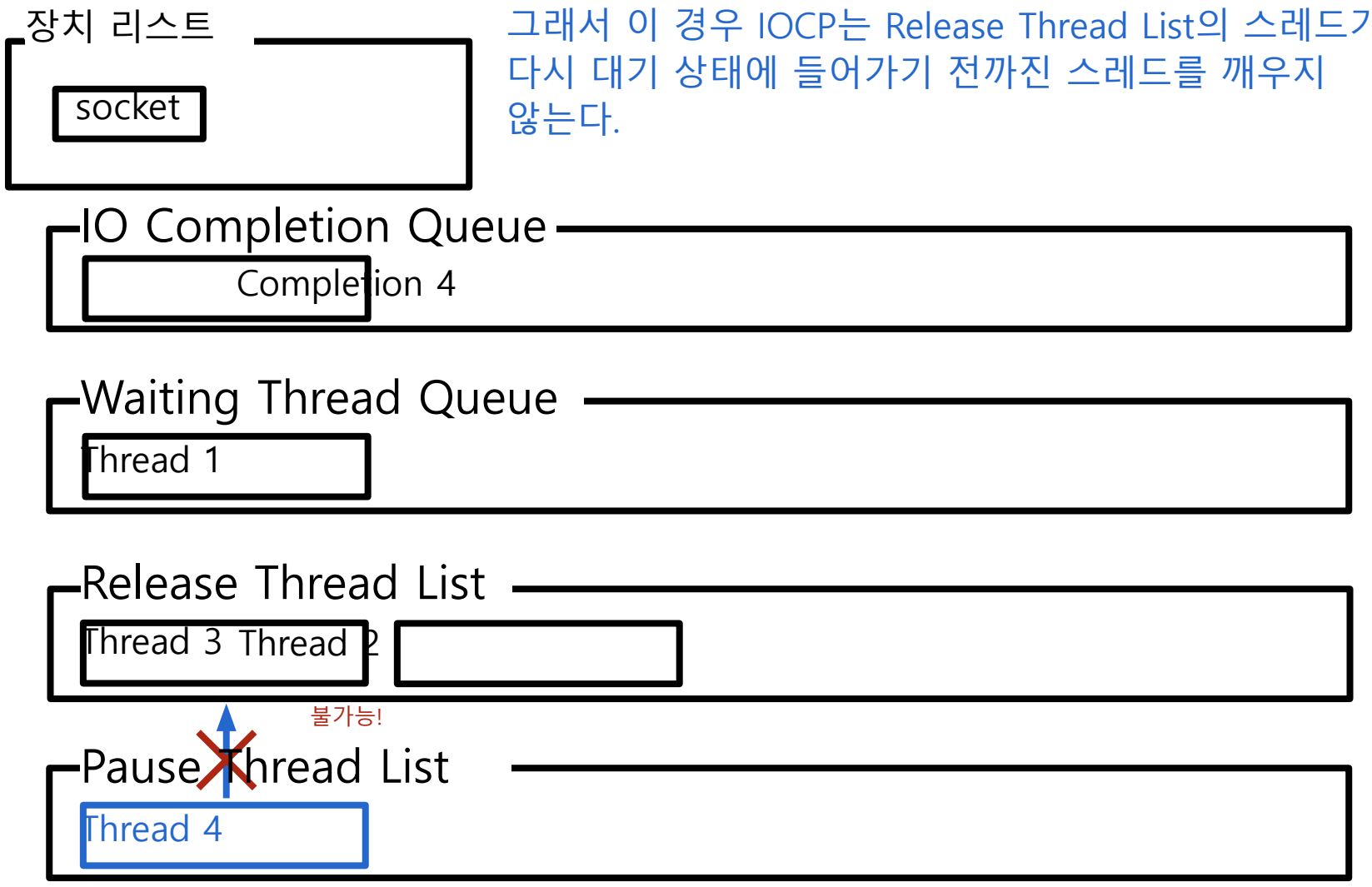
Thread 3 Thread 2

Pause Thread List

Thread 4

이제 대기 상태 끝!

예시



그래서 이 경우 IOCP는 Release Thread List의 스레드가 다시 대기 상태에 들어가기 전까진 스레드를 깨우지 않는다.

코드(GQCS/PQCS)

각각의 IO Worker Thread에서는 아래와 같은 코드를 수행해서 IO 완료에 따른 처리를 하게 된다.

```
while(true)
{
int ret = GetQueuedCompletionStatus(port,&numBytes,&completionKey,&overlapped,TIME_OUT);

if(ret == 0 && GetLastError() == WAIT_TIMEOUT) continue;

//그 외 경우 에러 처리는 생략. overlapped가 nullptr이라든가 numBytes가 이상하다든가 하는 경우들이 있음. 보통 연결 종료.

//completionKey가 종료된  IO작업 종류를 담고 있다고 가정
if(completionKey == IO_RECV)
{
                                                    //recv 관련 작업 수행
}
else if(completionKey == IO_SEND)
{
                                                    //send 관련 작업 수행
}
}
```

PQCS는 IO Completion Queue에 새 요소를 집어넣을 수 있게 해준다. 다른 스레드에 뭔가 완료 통지를 보내고 싶을 때 유용. 인자는 GQCS와 거의 동일(GQCS 호출시 필요한 데이터 거의 그대로 넘김)

```
//example. 대기중인 다른 스레드 종료시키기(THREAD_RELEASE가 스레드 종료 이벤트라고 가정).
PostQueuedCompletionStatus(port, 100, THREAD_RELEASE , overlapped);
```

코드(SEND/RECV)

send, recv를 호출하는 부분에서는 아래와 같은 방식으로 코드를 작성하게 된다.

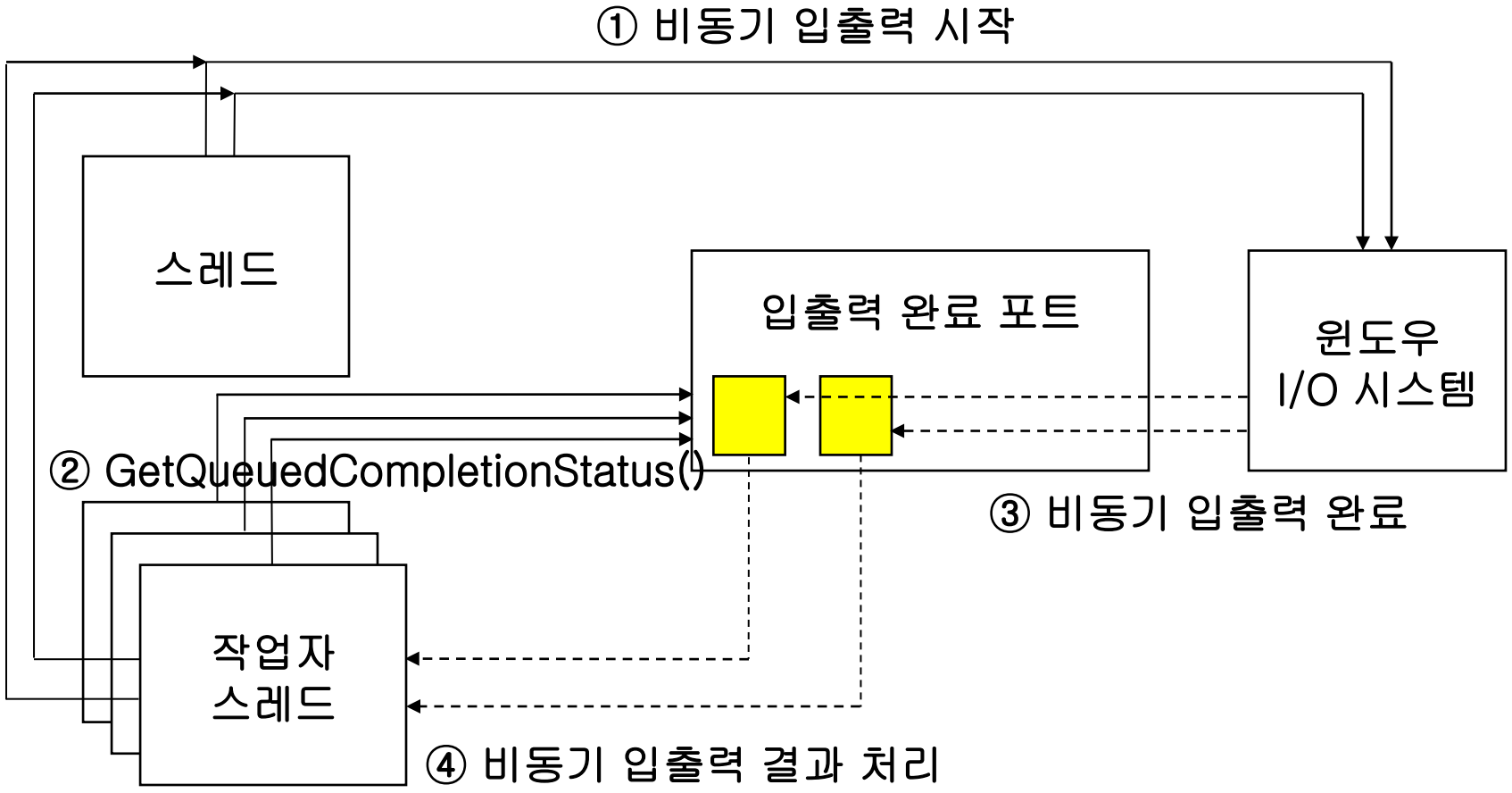
```
        WSARecv(clntSock,                   //클라이언트 소켓.
&ioContext->wsaBuf, 1,                      //읽을 데이터 버퍼의 포인터.
nullptr,                                    //데이터 입력 버퍼의 개수.
&flags,                                     //recv 결과 읽은 바이트 수. IOCP에서는 비동기 방식으로 사용하지 않으므로 nullptr 넘겨도
&ioContext->overlapped),                    무방.
nullptr);                                   //recv에 사용될 플래그.
                                            //OVERLAPPED 구조체의 포인터
                                            //completion routine 함수의 포인터. IOCP에서는 사용하지 않으므로 nullptr 넘겨도 무방.


WSARecv(clntSock,
&ioContext->wsaBuf, 1,
nullptr, 0,
&ioContext->overlapped), nullptr);
```

recv나 send나 거의 똑같다. io 작업을 위한 버퍼와 OVERLAPPED 구조체의 포인터를 넘겨준다.

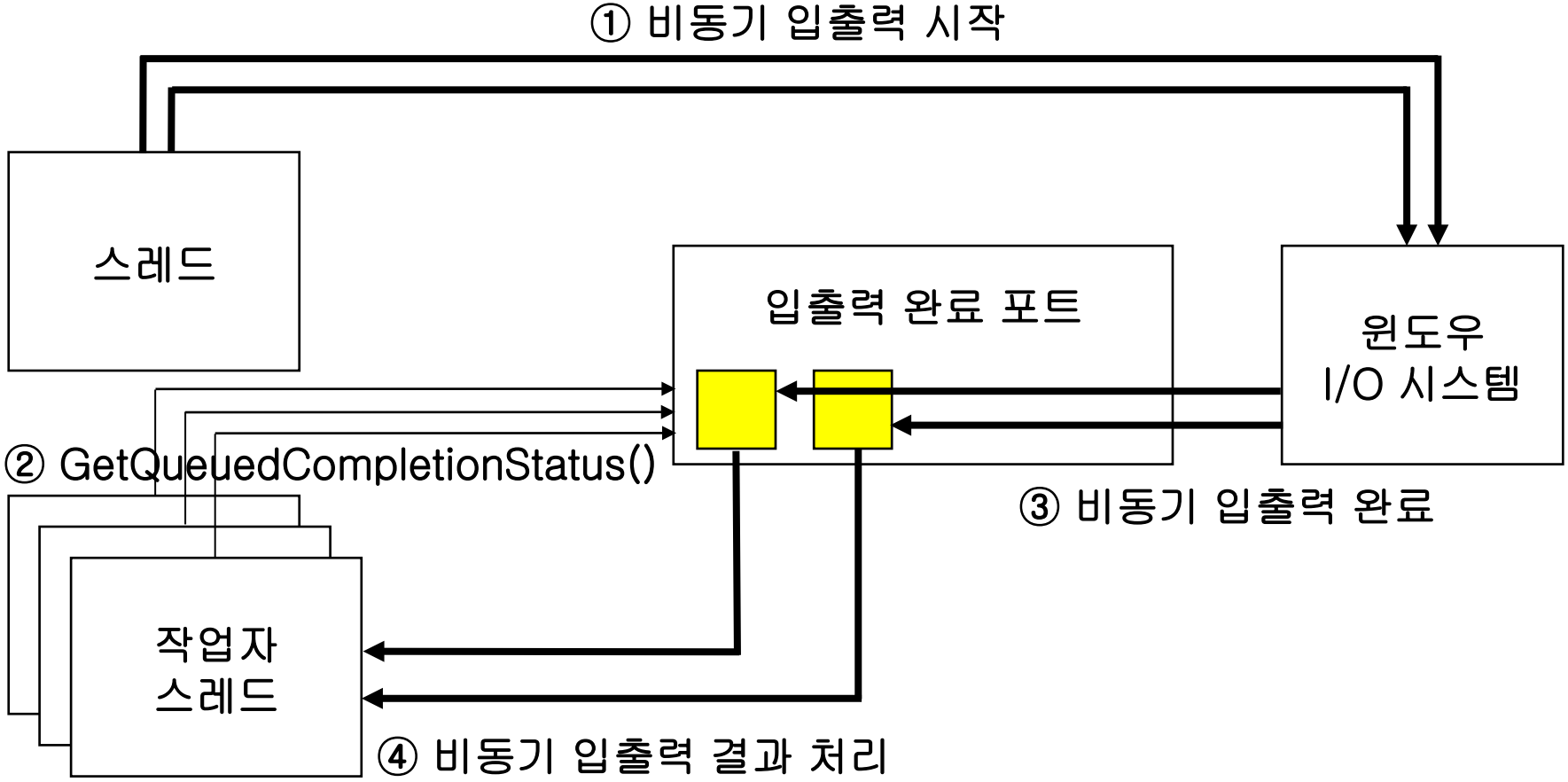
Completion Port 모델 (2/7)

- Completion Port 모델 동작 원리



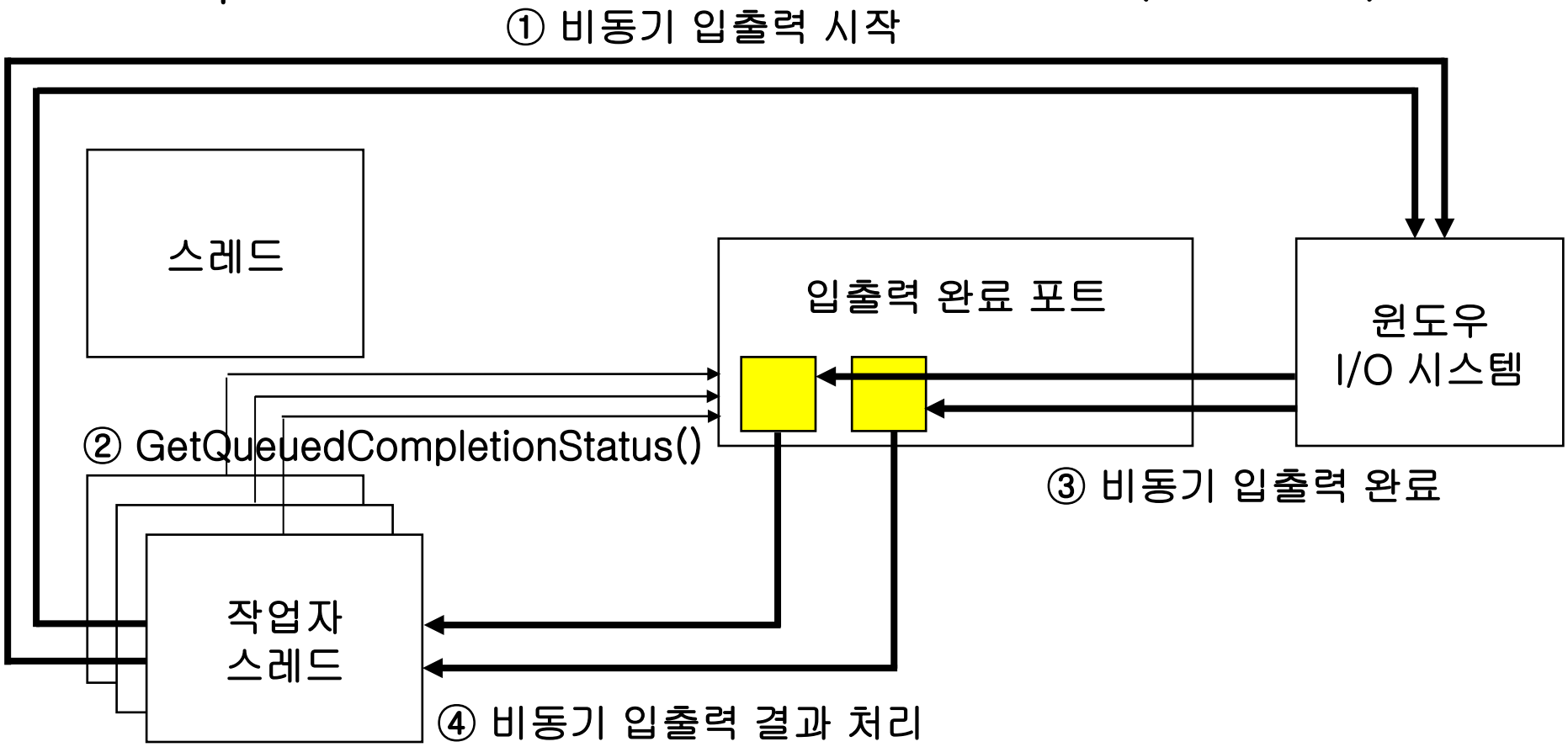
Completion Port 모델 (3/7)

- Completion Port 모델 동작 원리 (cont'd)



Completion Port 모델 (4/7)

- Completion Port 모델 동작 원리 (cont'd)



Completion Port 모델 (5/7)

- Completion Port 모델을 이용한 소켓 입출력 절차
  ① CreateIoCompletionPort() 함수를 호출하여 입출력 완료 포트를 생성한다.
  ② CPU 개수에 비례하여 작업자 스레드를 생성한다. 모든 작업자 스레드는 GetQueuedCompletionStatus() 함수를 호출하여 대기 상태가 된다.
  ③ 비동기 입출력을 지원하는 소켓을 생성한다. 이 소켓에 대한 비동기 입출력 결과가 입출력 완료 포트에 저장되려면, CreateIoCompletionPort() 함수를 호출하여 소켓과 입출력 완료 포트를 연결시켜야 한다.
  ④ 비동기 입출력 함수를 호출한다. 비동기 입출력 작업이 곧바로 완료되지 않으면, 소켓 함수는 오류를 리턴하고, 오류 코드는 WSA_IO_PENDING으로 설정된다.
  ⑤ 비동기 입출력 작업이 완료되면, 운영체제는 입출력 완료 포트에 결과를 저장하고, 대기 중인 스레드 하나를 깨운다. 대기 상태에서 깨어난 작업자 스레드는 비동기 입출력 결과를 처리한다.
  ⑥ 새로운 소켓을 생성하면 ③~⑤를, 그렇지 않으면 ④~⑤를 반복한다.

Completion Port 모델 (6/7)

- 입출력 완료 포트 생성

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

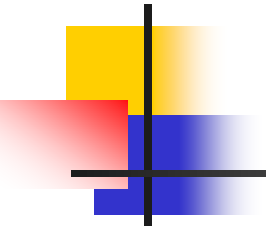
성공: 입출력 완료 포트 핸들, 실패: NULL

Completion Port 모델 (7/7)

- 비동기 입출력 결과 확인

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

성공: 0이 아닌 값, 실패: 0



실습

IOCP 서버 모델로 다음을 위한 서버를 구현하시오.

클라이언트가 접속하면 현재 날짜와 시간을 “2021년 XX월 XX시 XX분 XX초” 형식으로 서버가 클라이언트에게 전송한다.