Overlapped IO와 IOCP 이야기 - 2번째

이 기탁 (Microsoft 2002 Asia MVP)
E-Mail: snaiper80@korea.com ,
nsnaiper@hotmail.com(MSN ID)
Devpia ID: snaiper

자자 두 번째 강좌네요. 처음 강좌에서는 모자람이 많았을 텐데 좋았다고 평가해주신 점 감사 드립니다. 그런데 틀린 점이나 모자란 부분을 지적해주시는 분들은 없더군요. 모자란 부분을 지적해주시면 좋을 텐데…… 다들 바쁘신 듯 ^^ 뭐 이제 앞 잡담(?)은 그만하고 시작하죠.

◆ Overlapped IO와 Winsock

이제 Overlapped IO 와 Winsock을 얘기해 볼 차례입니다. 저번 강좌에서는 Overlapped IO가 뭔지, 뭐할라고 쓰는지 등등을 알아봤습니다. 또한 장점도 알아보았죠. 이전에 언급했던 장점 외에도 Receive시의 버퍼 풀을 미리 방지함으로써 속도 느려짐을 방지하는 것도 있다는 점이 있습니다.(사실은 빼먹었습니다 ^^:) 이것은 버퍼를 우리가 제공함으로써 연쇄적으로 얻어지는 장점이라고 볼 수 있겠네요. 일단 이 내용은 TCP에서의 Rate-throttling 얘기입니다. 시간 나시면 이 쪽을 찾아보세요. 그리고 Overlapped IO 는 ReadFile, WriteFile등의 File IO에도 적용된다는 사실을 까먹지 마시길 바랍니다. 물론 우리가 가장 많이 보는 건 WinSock이고, 또 네트워크 분야에서 관심을 가장 많이 가지는 주제이기도 하지만요.

♣ File IO시의 Overlapped IO를 하면서 나오는 유용한 점이 전역적인 성격을 띄고 있는 파일 포인터의 사용을 피할 수 있다는 점입니다. 그래서 파일 포인터의 사용에 답답함을 느끼시고 계시다면 이 Overlapped IO를 이용하는 것도 좋으리라 생각합니다.

이제 WinSock에 대하여 얘기해보죠. WinSock 에서 Overlapped IO를 이용해서 얻는 점은 뭘까요? 이전 강좌에서도 얘기했든 카피시간을 줄이는 등 성능상의 이점이겠죠. 그리고 전에는 말하지 않았는데 Disk IO에서의 Device-level Parallelism은 이 Winsock에 들어와서는 Multi-homed 그러니까 랜 카드 여러 개 꼽히는 걸 병렬적으로 처리할 수 있습니다.
뭐 네트워크 IO에 적용된다는 점 외에는 크게 차이 없다고 할 수 있습니다.
일단 기본적인 얘기는 다 했으니 간단히 이런 Overlapped IO를 어떻게 소켓에 적용해야

하는지 알아봅시다. Overlapped IO를 소켓이 하려면 일단 이걸 쓴다는 옵션을 줘야 합니다. 이 옵션이 뭐냐 면 바로 WSA_FLAG_OVERLAPPED 라는 겁니다. 아마 읽어보시는 분들 중에서도 이거 넣어보신 분들 많으리라 봅니다. 이 옵션을 소켓 생성 시에 줌으로써 소켓이 Overlapped IO를 할 수 있도록 선언해 주는 거죠. 그리고 한가지 참고하실 점은 socket() 함수는 이것을 기본적으로 잡고 들어간다는 겁니다. 반대로 얘기하자면 2.X 계열 소켓 생성 함수인 WSASocket은 이걸 기본적인 옵션에 넣지 않는다는 얘기이죠. 그래서 Overlapped IO를 하려면 반드시 넣어주세요. 그런데 제가 저번에 MSDN 뒤져보다가 본 얘기인데 WSASocket 사용시에 Overlapped IO를 사용하지 않더라도 반드시 WSA_FLAG_OVERLAPPED를 넣어주라고 되어 있더군요. 뭐 버그 때문이라고 하던데 혹시나 WSASocket써서 문제가 생긴다면 이런 점도 한번 가만해 보시길 바랍니다.

그럼 간단히 소켓 생성하는 코드를 한번 써 볼까요?

```
SOCKET sock=WSASocket(AF_INET,
                      SOCK_STREAM,
                      IPPROTO_TCP,
                      NULL,
                      0,
                      WSA_FLAG_OVERLAPPED);
```

자 Overlapped IO 소켓을 생성하는 코드입니다. 간단하죠? 이 정도는 해보신 분들이야 많으실 겁니다. 뭐 생성하는 거는 문제가 아니고 Send, Recv 하는 것이 중요하겠죠? 일단 Send하는 함수부터 봅시다.

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

MSDN 에 나와 있는 WinSock 2.2 함수인 WSASend 함수 입니다. 참고로 말씀 드리자면 Winsock 1.1 함수인 send, recv 함수는 이 Overlapped IO 를 지원 못합니다. 아예 그 인자가 없죠. 모르시는 분들은 MSDN 에서 찾아보세요. 그래서 당근 DLL 초기화 시도 2.2 로 해야 되겠죠? 2.2 함수를 써야 하니까요.

자 다른 얘기는 그만하고 위 함수를 보죠. 뭐 send 에 비해서는 복잡합니다. 그래도 Overlapped IO 에 관한 인자는 뭔지 아실 수 있겠죠? 아래 두 개입니다. LPWSAOVERLAPPED 가 바로 OVERLAPPED 구조체를 집어넣는 겁니다. 지금 저기에서는 WSAOVERLAPPED 라고 되어 있지만 OVERLAPPED 가 WSAOVERLAPPED 나 그 놈이 그 놈입니다. 그래서 여기에 Overlapped 구조체에 대한 포인터를 넣습니다. 넣는 거야 new OVERLAPPED 를 하시든 OVERLAPPED ov; 해서 &ov 요렇게 하든 자유입니다만…… 한 가지 주의하실 점은 이 OVERLAPPED 구조체라는 것이 IO 가 끝나기 전에 절대 메모리에서 사라지면 안 된다는 점입니다. 이미 아시는 분들도 있겠지만 IO 를 하면서 이 OVERLAPPED 구조체의 멤버를 계속 쓰기 때문에 없으면 런타임 에러를 발생시킵니다. 물론 저도 첨 Overlapped IO 를 할 때 이거 모르고 하다가 에러 먹고 열심히 디버깅 해본 기억이 있었죠. 이것을 첨 하시는 분들께서 자주 하실 실수 중에 하나이니 주의하시길 바랍니다. 뭐 그럼 위에서 어떻게 해야 하는지 답은 나왔네요. 그 답은 아실 거라고 보고 넘어가겠습니다.

그리고 LPWSAOVERLAPPED_COMPLETION_ROUTINE 요것도 궁금하시겠죠? 이건 Overlapped IO 를 이용한 콜백 처리 방식 때 사용하는 겁니다. 제가 일전에 얘기했던 몇 가지 방법 중에 하나입니다. 간단히 말씀 드리자면 여기에 콜백 함수를 하나 지정해줍니다. 그리고 Overlapped 구조체를 이용하여 IO 를 하나 요청해주고(Send 든 Recv 든 둘 중에 하나겠지요?) 그 쓰레드를 Alertable 하게 만듭니다.(이 Alertable 이라는 게 골 때립니다. 완전히 Suspend 도 아니고 그렇다고 완전히 Active 하지 않는 그런 요상한 상태라고 보시면 되겠네요.) 이렇게 되면 IO 가 완료되면 이 콜백 함수를 호출해서 다 됐다고 얘기를 해주는 겁니다. 그럼 이 콜백 함수에서 처리해서 IO 를 더 요청하던지 아님 소켓을 닫던지 하는 거죠. 콜백 함수의 원형을 보시면 대충 아실 거라고 봅니다. 이것에 대한 예제가 필요하시다면 올려드리겠습니다. 하지만 그게 버그가 있는 관계로 예제 수준으로 만족하셔야만 하는 코드이니 그 점 참고하시길 ^^:

그럼 두 가지 인자를 살펴봤습니다. 다 끝났다고 생각하시겠죠? 아닙니다 더 있습니다. 실제로 Overlapped IO 프로그래밍하면서 주의 깊게 처리하실 부분 중에 하나가 리턴값에 따른 에러 처리입니다. 제가 저번에도 강조해 드린 바가 있습니다. 리턴 값은 체크하셔야 하는데 뭐 대충 이렇다고 보시면 됩니다.

```
Int ret=WSASend();
If(ret==SOCKET_ERROR)
{
    if(WSAGetLastError()!=ERROR_IO_PENDING)
    {
        // 에러 처리
     }
```

}

이렇게 ERROR_IO_PENDING 에러를 만나시면 무시해주세요. 이것은 IO 진행 중이라는 얘기입니다. 이것은 에러로 처리하지 마세요. 에러라기 보다는 하나의 정보로 봐주시면 되겠습니다.

자 이제 조금은 지루했을 함수 설명은 마치고 IOCP 에 대한 쪽으로 슬슬 넘어가보죠.

◆ IO Completion Port!

드디어 IOCP 에 대한 얘기를 해보는 군요. 그 전에 빼먹은 것들이 있긴 하지만(나중에 한다고 넘긴 것들 있죠?) 그건 나중에 하는 게 나을 것 같습니다. 솔직히 말해서 이 주제에는 별로 맞지 않는 얘기이기도 합니다.

그럼 IOCP 란 뭔가? 이런 주제를 가지고 얘기를 해보죠.

◆ IOCP 란?

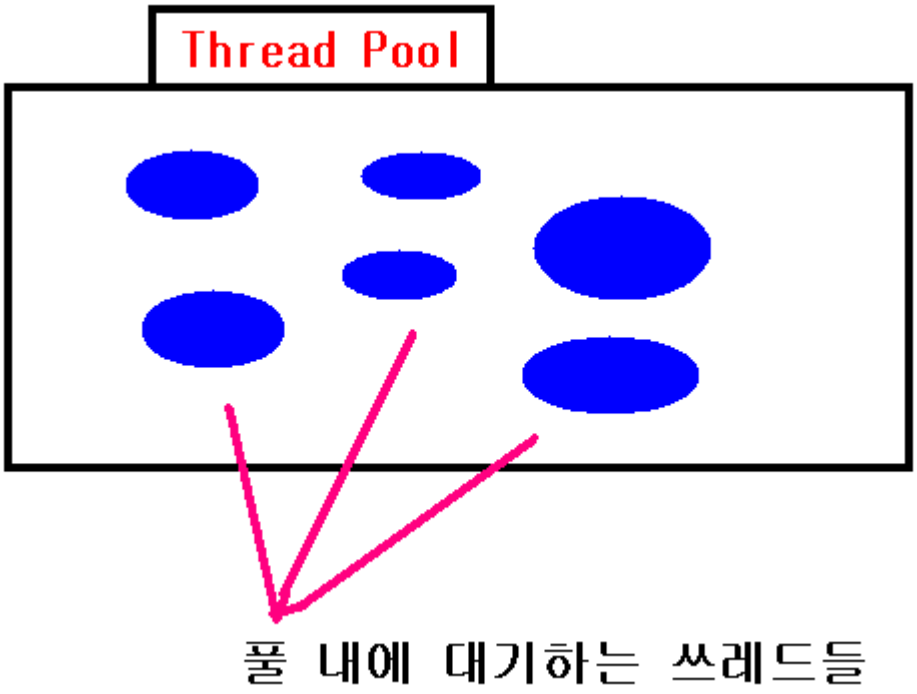
그럼 IOCP 란 뭘까요? 일단은 운영체제 적인 측면에서 따져보면 Win32 에서 제공하는 커널 오브젝트 중에 하나입니다. 즉 말해서 MUTEX 라던지 Thread Kernel Object 라던지 이런 것과 비슷하다는 얘기이죠. 하지만 그 작용이 조금 특별 나다라는 점이 우리가 IOCP 에 관심을 갖는 이유일 겁니다. 그리고 다른 측면에서 생각해볼까요? IO Completion Port 라는 용어에서 그 의미를 찾아본다면 Port 라는 것은 어떤 것이 드나드는 것을 말하죠. 뭐 컴퓨터에서 가장 쉽게 볼 수 있는 것이 시리얼 포트 같은 것이겠죠? 그럼 Completion 단어도 있고 IO 라는 단어도 있으니까 이걸 붙여보죠. IO Completion 즉 IO 완료라는 겁니다. 그러니까 IO 가 완료되면 어떤 정보 단위가 만들어 지고 이것이 port 즉 이것을 통해 왔다 갔다 한다는 겁니다. 흠…… 조금 더 생각한다면 IO 를 감시한다는 역할도 있을 수 있겠죠? 일단 이렇게 정의를 해보고 넘어가죠. IOCP 라는 놈이 정확한 정의보다는 그것이 하는 일이 더 초점이 맞춰져 있는 놈이라서 책이나 제가 가지고 있는 자료나 다들 정의를 두리뭉실하군요. 뭐 그래서 저도 두리뭉실하겠지만 그렇게 넘어가야겠네요.

◆ IOCP 가 하는 일

자 이것이 하는 일이 뭔지를 살펴보죠.(이것이 가장 중요하겠죠? ^^:)

1. 재사용이 가능한 쓰레드 풀을 유지합니다.

   말을 이해하시겠나요? 일단 쓰레드 풀이라는 것이 생소하신 분들도 계시리라고 생각합니다. 쓰레드 풀이라는 말을 설명하는 것이 바로 재사용이 가능하다 라는 말입니다. 일단 풀이라는 것부터 해보죠. 수영장에 있는 그 풀장을 생각해보세요. 물들이 모여서 갇혀있습니다. 물이 지 아무리 용 써봐야 그 안을 못 빠져나갑니다. 이런 것을 머리 속에 그림을 그리면서 생각해보세요.



(허접하지만 그림도 그려봤습니다. 별로 이해하는데 도움 될 것 같지는 않겠지만요 ^^)

여기 풀에서 물 대신 쓰레드들이 들었다고 생각하세요. 쓰레드들이 풀에 모여 있겠네요. 뭐 둥둥 쓰레드들이 둥둥 떠 다닌다고 생각하셔도 좋을 듯 합니다. 자 그럼 이 쓰레드들은 여기서 더 못 나가고 계속 거기에 있겠네요. 그럼 우리가 풀장에서 그 물에서 계속 놀듯이 그 물들이 어딜 안 가고 계속 쓰이겠죠? 강처럼 어디 안 흘러가잖아요. 이와 같이 쓰레드도 한번 쓰고 버리는 게 아니고 계속 쓰는 겁니다. 이게 재 사용 가능하다는 말입니다. 한번 쓰고 그 쓰레드 없애버리는 것이 아니라 도로 그 쓰레드 모임 속으로 돌아간다는 얘기죠. 이런 개념이 바로 쓰레드 풀입니다. 이런 비슷한 풀 개념들이 많이 쓰입니다. Memory Pooling, Connection Pooling, Thread Pooling, Object Pooling(이건 COM+ 에서 나오는 얘기던데, 자세히는 몰라도 Component 를 바로 죽이지 않고 다음에 요청하면 풀 중에서 또 꺼내와서 그 객체를 COM 클라이언트에게 붙여주는 겁니다.) 등등 참 풀(Pool)로 만들어 버리는 것도 많습니다. 그럼 이런 풀링(풀 만드는 걸 말합니다.)을 해서 나오는 장점이 뭔가가 궁금하시겠죠? 이렇게 풀링하는 놈들을 생각해보세요. 메모리, 커넥션(연결되는 소켓을 말합니다.), 쓰레드, 오브젝트자 생각해보자 전부 리소스 들이죠? 그리고 만들 때 시간도 잡아먹고요. 조금 어렵게 말하면 만드는데 오버헤드가 있는뎅! ^^: 이란 겁니다. 그래서 현재 말하고 있는 쓰레드 또한 만드는데 그 오버헤드가 있습니다. 그런데 서버가 돌아가는 데에서 이런 오버헤드가 존재한다는 말은 즉 성능과 바로 직결되는 사항입니다. 빨리 빨리 처리해야 하는데 쓰레드 만드느라 시간 먹고

있어보세요. 클라이언트들만 미치죠. 그래서 이런 것을 해결할려고 쓰레드 풀링을 하는 겁니다. 즉 서버 시작할 때 왕창 만들어 두는 거죠. 그럼 중간에 안 만들어도 되겠죠? 그리고 쓰고 집어넣고 필요하면 또 빼서 쓰고요. 그렇게 한다는 겁니다.

자 그럼 그렇게나 좋은(?) 쓰레드 풀링을 IOCP 를 얘기하는 도중에 나오는 이유가 뭐냐? 라고 궁금하실 텐데…… 이 IOCP 가 쓰레드 풀링을 지원한다는 겁니다. 조금 바꿔 말하자면 IOCP 를 쓰면 쓰레드 풀링을 쉽게 할 수 있다는 겁니다. 물론 IOCP 를 쓰지 않고도 얼마든지 쓰레드 풀링이 가능합니다. 그런데 쉽게 만들어보려면 IOCP 를 이용하는 게 빠르고 좋다는 말입니다. 그래서 잘만하면 IOCP 가지고 IO 만 하는 게 아니고 이 쓰레드 풀링이 필요할 때 이거 가지고 만들어도 되는 겁니다. 뭐 요즘 닷넷에 들어와서 MFC 도 업그레이드 되면 CThreadPool 이던가 하는 클래스도 생겼더군요. 닷넷 프레임웍에도 이 쓰레드 풀이 클래스로 지원되는 듯 하고요.

2. CPU 에 Dispatch 되는 쓰레드를 조절합니다.

참 말 어렵게 해놨네요. 제가 적어놓고도 왜 이렇게 적어놨는지 모르겠네요. ^^: 짧게 줄이려다 보니 어려운 말만큼 좋은 것도 없군요. 이게 무슨 말인고 하니 운영체제를 공부하셨거나 학교에서 수강하신 분들은 아시겠지만 쓰레드가 돌아가려면 반드시 CPU 사용권을 가지고 있어야 합니다. 좀 쉽게 말하자면 쓰레드가 CPU 를 가지고 있어야 한다는 거죠. 당연하겠죠? 쓰레드라는 것이 실행 흐름인데 이 실행이라는 것이 CPU 가 없으면 말짱 헛거죠. 그래서 쓰레드들은 서로 이 CPU 를 얻으려고 합니다. 당연히 이걸 조정해야 하겠죠? 그래서 스케쥴러가 필요한 거고요. 쓰레드에 대한 스케쥴러를 Short Term Schduler 라고도 합니다. 뭐 이런 배경에서 생각해봅시다. CPU 를 가지려는 쓰레드가 많이 있다고 보세요. 그럼 스케쥴러는 이 쓰레드 실행하다가 저 쓰레드로 넘기고 하는 작업을 할 겁니다. 그런데 스케쥴러는 이런 작업을 공정하게 해야겠죠? 이 놈만 계속 주고 저 쪽 놈은 안 주고 하면 한 놈은 삐지잖아요.(^^::) 그럼 사용자가 보는 입장에서는 이 프로그램을 잘 되는데 저 프로그램은 왜 이렇게 잘 안되지 하는 상황이 되는 겁니다. 그럼 아무리 생각해봐도 별로 안 좋을 것 같다는 것은 아시겠죠? 이런 상황을 해결하기 위해 스케쥴러는 아주 공정하게 해야 합니다. 여러 가지 기준에 따라서 처리를 하겠죠? 뭐 쓰레드에 따르는 우선 순위라던 지 아니면 실행하는 시간이라던지 등등 뭐 여러 가지 기준들이요. 물론 이런 여러 가지 기준에 따라 처리하다 보면 위에서 얘기한 한 쪽은 안 주는 상황이 생길 수도 있습니다. (이 현상을 가리켜 starvation 이라고 합니다.) 그래서 이걸 최소화 해야 하는데 스케쥴러가 해야 하는 일 중에서 가장 중요한 거죠 여튼 스케줄러는 공정하게 줄려고 한다고 합시다. 동시에 돌아가는 쓰레드가 많으면 많을수록 CPU 를 차지해야 하는 쓰레드는 많아질 것이고 이 쓰레드를 차지할 때 필요한 작업 또한 많아질 겁니다. 여튼 이런 CPU 를 차지하는 쓰레드 교환작업을 쓰레드 컨텍스트 스위칭이라고 하는데 몇 개 스위치 하는거야 별거 아니지만 많은 쓰레드에서는 오히려 이 쓰레드 컨텍스트 스위칭하는게 더 시간이

걸리는 경우가 생겨버립니다. 아무리 CPU 좋더라도 쓰레드 많아 버리면 문제되는 거는 마찬가지이죠. 뭐 그래도 낫기야 하겠지만요. 뭐 예를 들자면 하는 일이 5 초인데 그 바꾸는 일이 10 초가 걸려버리는 상황이 발생한다는 겁니다. 그럼 이런 상황이 되 버리면 쓰레드가 아무리 많아 봐야 성능은 안 올라가고 컴퓨터는 버벅거리기만 하는 겁니다. 그래서 서버 모델에서 아주 많은 사용자를 받아들이는 상황이라면 1 user 1 thread 가 안 좋다는 이유가 이겁니다. 이 쓰레드 컨텍스트 스위칭 오버헤드가 커지기 때문에 하지 말라는 거죠. 그런데 IOCP 는 쓰레드를 효율적으로 관리함으로써 이런 상황을 미연에 방지하는 겁니다. 즉 어떻게 하느냐면 돌아가는 쓰레드 수를 제한시켜버리는 겁니다. 즉 최대 쓰레드 숫자를 정해버리고 그것을 지키면서 쓰레드를 처리하는 거죠. 물론 완전히 지키는 것은 아니고 어느 정도 융통성 있게 합니다. 이게 IOCP 의 좋은 점이라고 할 수 있겠죠.

◆ IOCP 를 한번 만들어 볼까?

제목 그대로 IOCP 를 생성하는 함수를 살펴봅시다. 이 IOCP Kernel Object 를 생성하는 함수 이름은 CreateIoCompletionPort 입니다. 정말 깁니다. 물론 더 긴 API 도 있습니다. 한번 찾아보시길…… 힌트를 드리자면 COM 관련 함수에 있습니다. 정말 정말 깁니다. 왜 그렇게 길 게 만들어 놨는지 --;

```
HANDLE CreateIoCompletionPort (
HANDLE FileHandle,                  // handle to file
HANDLE ExistingCompletionPort,   // handle to I/O completion port
ULONG_PTR CompletionKey,           // completion key
DWORD NumberOfConcurrentThreads // number of threads to execute concurrently
);
```

MSDN 에 있는걸 그대로 가지고 왔습니다. 인자를 하나씩 살펴보죠.

첫 번째 FileHandle, 이것은 IOCP 랑 연결할 Device 에 대한 핸들을 가리킵니다. File IO 라면 File Obeject 이고 그리고 우리가 자주 써먹는 곳이라면 Socket 이겠죠? 그리고 다음 인자를 보죠

ExistingCompletionPort 는 기존에 존재한 CompletionPort 객체를 가리킵니다. 그러니까 기존의 CompletionPort 객체와 IOCP 와 연결하는 작업에서 이 기존의 Completion Port Object 를 가리키는 겁니다. 그리고 두 번째 CompletionKey 요거 참 중요합니다. Key 라는 거 보니까 중요한 것은 아시겠죠? 이것은 FileHandle 에 넣어준 이 핸들을 구분해주는 역할을 합니다. 이거 가지고 완료되었을 때 어떤 것이 완료된 것인지를 알 수 있습니다. 물론 이것을 확장해서 쓰는 것이 거의 일반적이라고 볼 수 있겠네요.

마지막 인자 NumberOfConcurrentThreads 입니다. 이것은 동시에 돌아갈 수 있는 쓰레드 숫자를 지칭합니다. 이게 아까 말했던 쓰레드 스케쥴링 어쩌고 에서 얘기했던 겁니다. 이것을 0 으로 주면 디폴트로 CPU 숫자를 지정합니다. 그러니까 보통 컴퓨터라면 1 개가 됩니다. 이게 쓰레드 컨텍스트 스위칭을 최소한으로 할 수 있는 숫자입니다. 즉 최대로 돌아가는 것이 1 개라면 동시에 5 개 돌아가는 이것보다 일어나는 컨텍스트 스위칭이 없기 때문에 오버헤드를 줄일 수 있는 거죠. 하지만 이것도 조절 가능합니다. 그래서 인자를 둔 거겠죠? 자신의 서버 프로그램 성격에 따라 틀립니다. 그러니까 조절해보고 테스트 함 해보고 숫자를 정하란 말이죠.

그리고 리턴값을 알아봐야 하겠죠? 이미 알고 계시겠지만 IOCP Kernel Object 에 대한 핸들 값입니다. 이렇게 끝나면 좀 섭하겠죠? 물론 더 있죠? 설명을 보시고 파악하셨겠지만 이 함수를 두 가지 작용을 합니다. 하나는 디바이스 오브젝트랑 IOCP 랑 연결하는 하는 작업이고 하나는 IOCP 를 생성하는 작업이죠. 어쨔 이름에 좀 걸맞지 않은 것 같지만 그렇게 만들어 놨으니 따라야겠지요?

그래서 일단 IOCP Kernel 객체를 만들려면 어떻게 해야 하는데? 요걸 알아보죠. 어떻게 하느냐면

```
CreateCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 쓰레드숫자);
```

요렇게 하는 겁니다. Concurrent Thread 숫자를 지정해 주고요. 딴 인자를 이렇게 주면 IOCP 객체를 생성하는 게 됩니다. 그럼 연결하는 것도 한번 봐야겠지요?

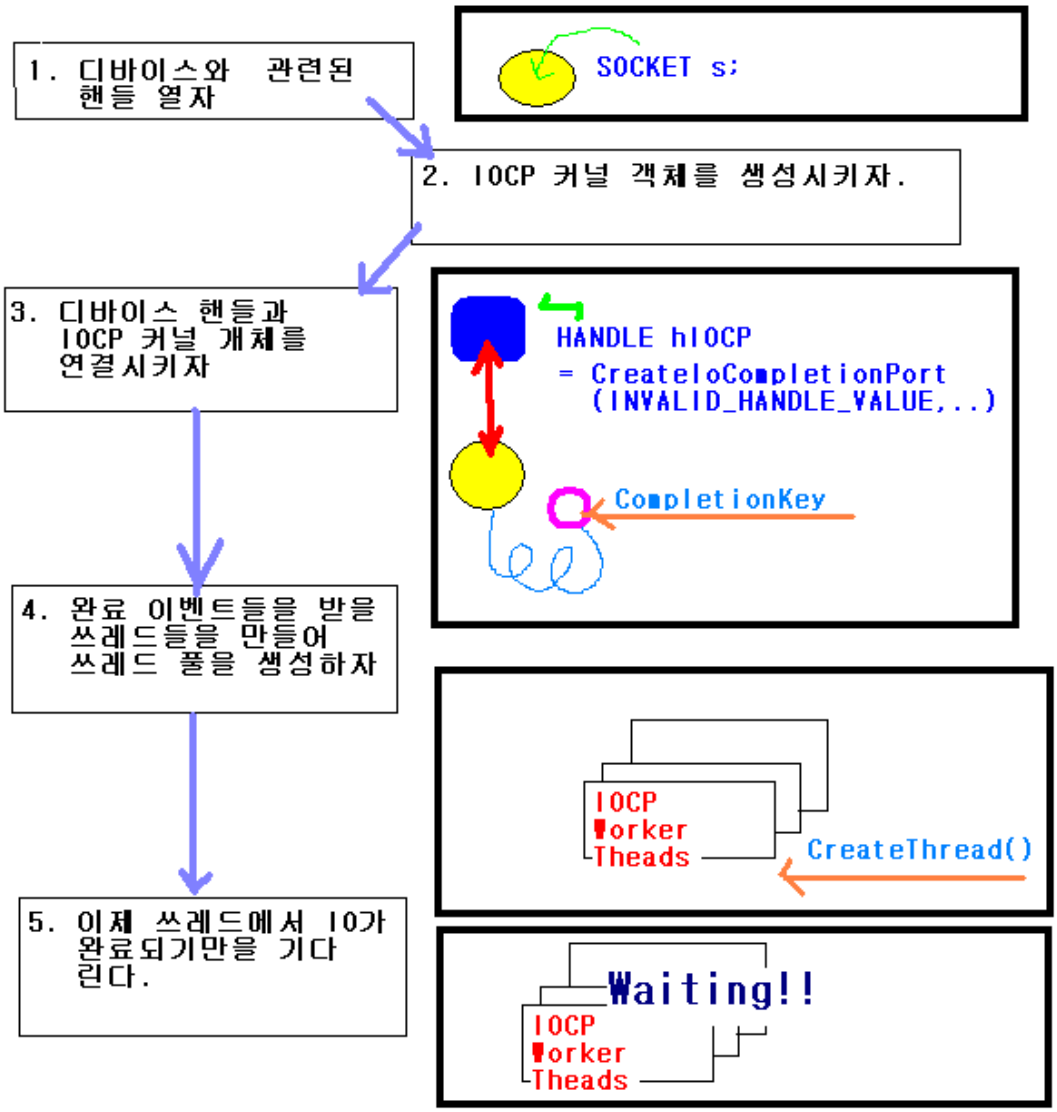
```
CreateCompletionPort(sock, hIocp, 컴플리션 키, 0 );
```

이런 식으로 합니다. 여기서 hIocp 는 위처럼 해서 생성된 IOCP 커널 객체에 대한 핸들입니다. 자 이렇게 하며 sock 과 hIocp 가 연결됩니다. 이렇게 연결되서 나서 리턴값이 hIocp 위치에 있는 값이 그대로 리턴됩니다. 만약 같지 않다면 뭔가 문제가 있는 겁니다.

◆ IOCP 프로그래밍의 기본?

IOCP 프로그래밍의 기본만 봅시다. 위에서 우리는 IOCP 커널 객체를 생성하는 방법과 API 모양새 봤습니다. 어때요? 죽이지 않습니까? (어째 이거는 다른 상황에서 쓰이는 말 같네용 ^^: 읽으시는 분들이 남자분들이라면 이미 눈치 채셨을 듯 ^^)
뭐 아니라고요? 아님 어쩔 수 없고, 그러려니 하고 넘어가야죠 ^^:::::

그래서 프로그래밍을 하는 기본적인 순서를 보죠.



자 좀 허접하지만 IOCP 가 제대로 동작하기 위한 순서들을 그림으로 간단히 그려봤습니다. 이해가 잘 되실는지 모르겠군요. 현재 위 그림에서 보면 3 번까지는 얘기했습니다. 이제 4 번, 5 번이 남았군요.

그럼 4 번을 얘기해보십니다. 쓰레드 풀을 만듭니다. 이전에 얘기했듯 그런 이유 때문에 만드는 것이라는 것을 기억하세요. 지금 쓰레드를 만드는 것은 제가 임의로 IOCP Worker Thread 라고 이름을 붙여봤습니다만 이것은 IO 가 완료되었다는 정보 패레코드를 받는 쓰레드 입니다. 이 쓰레드에서 정보를 받고 이 정보를 가지고 IO 가 더 필요하다. 그럼 더 요청하는 겁니다. 아님 이제 패킷 받았으니 DB Access 하자 그럼 DB Access 하는 거죠. 이런 작업들을 여기서 할 수 있습니다.

그리고 5 번에서는 이제 IO 완료 정보들이 오기만을 하염없이 기다리는 겁니다. IOCP 큐에 완료 정보 레코드가 들어와서 IOCP 가 자기(쓰레드)를 깨워주기를 기다리면서요

그러니까 Suspend 되어 있는 겁니다. IOCP 가 깨워 주기 까지는 CPU 스케쥴링을 받지 않습니다. 그래서 CPU 도 무의미한 IO 가 왔는지 안 왔는지 작업도 물론 없앨 수 있고 쓰레드가 계속 깸으로 인한 컨텍스트 스위칭 오버헤드 또한 줄일 수 있고요. 하지만 이건 IOCP 만의 장점이라 보기는 힘들겠군요. WSAEventSelect 나 어쩡정하긴 하지만 Alertable IO 도 비슷하긴 하니까요.

자 그런데 처음 듣는 IOCP 큐라는 용어가 나왔습니다. 이건 나중에 얘기할 겁니다. 하지만 뭐 생각난 김에 이거 가지고 할 수 있는 응용에 대해 말해보죠. - 응용부터 얘기하니까 좀 이상하긴 하군요 ^^: -

뭐 약간 얘기가 앞서가는 느낌이 듭니다만 모르겠다거나 왜 이런 이야기가 나오는데? 하시는 분들은 보다 넘어가셔도 좋습니다.

일단 서버에서 돌아가는 아키텍처(이렇게 말하는 게 맞는지 모르겠군요. ^^:)를 설계 시에 이러한 IOCP 큐와 쓰레드 풀로 Processing 단위로 일종의 Software Pipe Lining 을 만들 수 있습니다. 에 저번에 제가 얘기했죠? IOCP 는 Inter-Thread Communication 에도 쓰인다고요. 뭐 이건 제가 생각한 거긴 합니다만 이미 몇몇 분들이 사용하시기도 하는 방법이더군요. (항상 자기가 생각하면 최초 같은데 대부분 다른 사람이 이미 생각하고 있는 것이 세상 일이더군요 ^^)

어떻게 하느냐면 IOCP 를 이용함으로써 처음에 만드는 쓰레드들은 IO 관련 쓰레드 모임, 그리고 다음은 Processing 관련 쓰레드, 그리고 DB Access 쓰레드 모임 이렇게 만들고 이 사이를 IOCP 큐를 이용함으로써 연결 할 수 있습니다. 물론 쓰레드 모임들을 IOCP 가 큐에 뭔가가 들어오면 깨워 주므로 쓸데없이 큐에 뭐가 들어왔나 안 왔나를 체크하는 CPU cycle 낭비를 줄일 수 있습니다. 그래서 자기가 직접 큐를 구성하는 것보단 어째 보면 더 나을 수도 있다는 겁니다. 뭐 이렇게 하는 거에 동의하지 않는 분도 계실는지 모르겠군요. 뭐 여하튼 이렇게도 할 수 있더라 정도만 알아두시면 될 것 같습니다. 참 쓰레드 풀로 구성하실 부분은 되도록 블러킹이 많고 IO 가 있는 부분이 되면 좋을 것으로 보입니다. 그래서 예를 든 것과 같지 DB Access Thread 모임이라던가 이런 게 좋은 예가 될 것으로 보입니다.

자 두 번째 강좌는 여기까지 하죠. 역시 이번도 허접한 강좌입니다. 언제면 좋아질는지 뭔가를 쓰는 사람이란 영원히 만족 못하는 존재일는지도 모르겠네요.

여기까지 IOCP 에 대해 알아봤는데요. 다음에 중심 주제는 IOCP 가 어떻게 돌아가는가 그 원리를 설명하는 시간이 될 것 같습니다. 아무래도 안 되는 그림을 그려야 할 지도 모르는데 최소한 그림은 안 그리도록 노력해보죠 ^^: 그런데 강좌를 어떻게 끝내야 할런지가 감이 안 잡히네요. 끝내는 것도 잘 해야 하는데……^^ 어쨌든 담에 또 뵙죠/