Overlapped IO와 IOCP 이야기 - 3번째(추가)

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

실제로는 4번째네요. 그런데 이거 내용이 적어지는 바람에 추가분으로 하겠습니다. 여하튼 저번에 이어서 계속하도록 하겠습니다.

3) Waiting Thread Queue, Released Thread List, Paused Thread List

♤ Released Thread List

이제 Release Thread List와 Paused Thread List 얘기할 차례이군요. 두 개 같이 얘기하는 이유는 뭐 그렇게 얘기하면 더 편하고 이해하기 쉽다고나 할까요?

먼저 Released Thread List를 말해보죠. 이것은 현재 돌아가고 있는 IOCP Worker Thread에 대한 내용을 유지하고 있습니다. 그러니까 쓰레드 풀에서 꺼내온 쓰레드들이라고 할까요? Released 즉 뭐 자유롭게 풀어놓아놨다는 얘기죠. 뭐 이해하기도 어렵지 않죠?
간단히 언제 이게 추가되고 또 제거 되는지를 알면 될 것 같습니다.

♤ 정리

추가(생성): ① IOCP가 WTQ에서 쓰레드를 깨울 때
(쓰레드 깨우면 그건 곧 돌아가는 쓰레드가 되니까 여기에다 그 쓰레드의 아이디를 추가하는 겁니다.)
② Paused Thread가 깨어날 때
(이건 나중에 얘기하는 것이 좋겠군요.)

제거: ① 쓰레드가 다시 GetQueuedCompletionStatus 함수를 부를 때
(이건 저번 강좌에서도 얘기했으니 특별히 얘기 안 해도 될 것 같군요. 보시면 아시겠지만 3개가 연관되어서 돌아갑니다.)
② 쓰레드가 스스로 Suspend 될 때
(즉 쓰레드가 Sleep 을 불렀다던지 WaitForSingleObject를 불렀다던지 아님 그 외에 블록이 되는 함수를 불렀다던지 하는 상황을 말하는

겁니다.)

♤ Paused Thread List

이것 또한 이해에는 그리 어렵지 않습니다. Paused, 즉 정지된 쓰레드입니다. 동작이 정지되었다는 말은 위에서 말한 바와 같이 쓰레드가 어떤 이벤트로 인해서 Suspend되어 있다는 말이죠. 뭐 이것도 어렵진 않죠? 위와 같이 언제 추가, 제거 되는지 알면 이해하기 쉬울 거라고 생각합니다.

♤ 정리

추가(생성): Released Thread 즉 돌고 있는 쓰레드가 스스로 멈출 때
(즉 스스로 Suspend된다는 말입니다. 위와 연결된다는 것을 느끼실 수 있을 겁니다.)

제거: Suspend된 쓰레드가 깨어날 때
(어떤 이벤트가 완료되거나 하여서 쓰레드 스스로가 깨어나는 것을 말합니다. 그러면 dwThreadID는 RTL(Released Thread List로 옮겨갑니다.)

자 이렇게 RTL과 PTL이 정리가 되었군요. 정리는 되었지만 왠지 이렇게 끝내기는 허전하죠? 안 허전하다면야 ^^ 할 수 없죠 ^^:

이제 하려는 얘기는 Concurrent Thread 수와 이제 말했던 두 개 간의 관계입니다. 제가 저번 강좌에서 이 얘기를 하면서 Smart 하다더니 뭐 이런 얘기를 했습니다. 그 얘기를 여기서 하죠. (사실 저번 강좌에서 하려고 생각했었는데 이야기가 길어지면서 어쩔 수 없이 이번 강좌로 넘어왔습니다.)

♤ Concurrent Thread 수와 PTL, RTL

저번 강좌에서 제가 Concurrent Thread 수에 대해서 언급했습니다. 뭐라고 했었죠? 지정해준 숫자 이상은 더 쓰레드 안 돌린다고 했었죠? 뭐 그 뒤에 약간 여운이 남는 말도 했지만요. 그런데 조금 생각을 해보셨던 분들이라면 이런 의문이 생기실 겁니다. 그러면 쓰레드 풀이 왜 필요하지? 라는 의문이죠. 어차피 지정된 수만큼만 돌 것이라면 그것만 쓰레드 만들어 놓으면 안되나? 왜 더 만들어 놓을까? 어차피 돌지도 않을 텐데 이런 의문이 드셨을 겁니다. 하지만 이렇게 하는 이유가 다 있겠죠? 그걸 하나하나 생각해 봅시다.

위에서도 말했지만 IOCP는 IO Completion Queue에 내용이 있으면 쓰레드를 깨우고 그 쓰레드의 ID를 RTL에 가져다 놓는다고 했습니다. 그리고는 IOCP 는 무슨 일을 하느냐 하면 어떤 쓰레드가 깨어 있느냐라는 사실과(RTL이 있으니 잘 알겠죠?) 그 쓰레드 실행을 감시합니다. 그래서 이 쓰레드가 어떤 함수를 불러 스스로 Suspend 상태로 들어 간다면 IOCP는 자동으로 이것을 알아채고 PTL에다 가져다 놓습니다. 그럼 생각해보세요. RTL에 있는 쓰레드 숫자가 줄어들었죠? 그럼 여기까지 상황을 상상해보면서 Concurrent Thread 수에

대한 정의를 내려봅시다.

Concurrent Thread 수라는 것은 현재 CPU가 스케쥴링하는 쓰레드 수 입니다. 그러니까 지금 IOCP 큐에 있는 레코드를 처리하기 위하여 돌아가고 있는 쓰레드 수라는 거죠. 이것을 제한하는 것이 저번에 얘기했던 CreateIoCompletionPort에서 지정하는 수입니다. 이것을 RTL이라는 용어가 나왔으니까 연결시켜보면 즉 RTL에 있는 쓰레드 수라는 거죠. 그럼 정리해보면 CreateIoCompletionPort에 지정했던 수는 RTL에 들어갈 수 있는 쓰레드 숫자를 제한하는 겁니다. 그럼 정리되셨죠?

자 아까 잠시 정지시켰던 사고를 계속 해보지요. 위에서 말했던 바대로 지정된 Concurrent Thread 수에 따라 들어가는 수를 제한시켜 놓고 있습니다. 그런데 하나가 Suspend 상태에 들어가 RTL이 하나 줄었습니다. 그럼 생각해보세요. RTL에 있는 숫자를 최대한 Concurrent Thread 수와 같이 해놓는 것이 더 좋겠죠? 하나 거기 들어갔다고 그냥 아무 일도 안 하고 있으면 왠지 비효율적일 것 같지 않겠습니까? 그래서 IOCP 큐에 뭔가 더 있다면 쓰레드 풀에서 쓰레드를 하나 더 가져와서 처리를 계속합니다. 그러면 왜 쓰레드 풀에 더 많은 쓰레드를 만들어 놓는지를 아시겠죠?

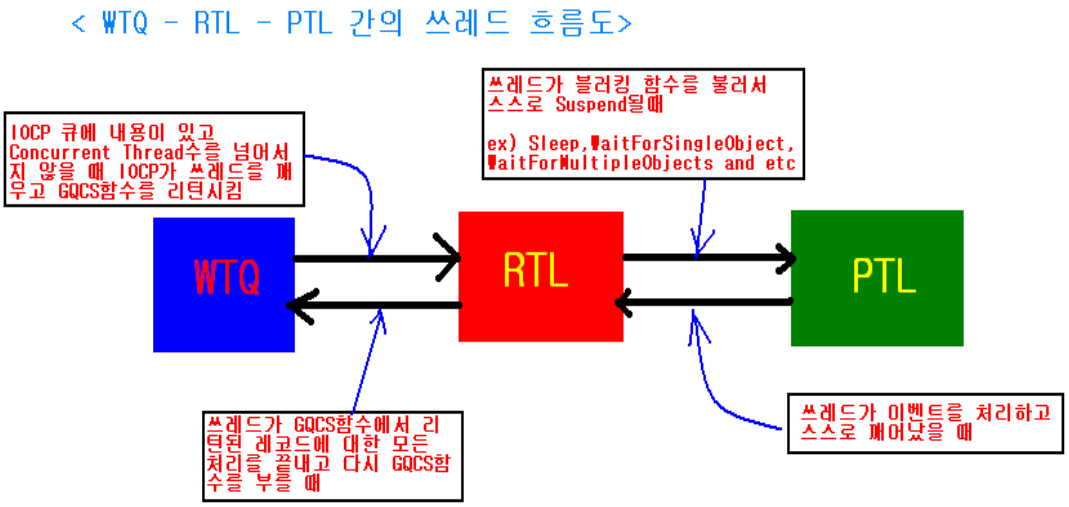
그런데 또 하나 생각할 상황이 생겼네요. PTL 에 있던 쓰레드, 즉 Suspend되어 있던 쓰레드가 깨어나면 어떻게 될까요? 위 정리에서 했던 말 대로 이것은 RTL로 갑니다. 그럼 RTL에 있는 쓰레드 수가 지정된 Concurrent Thread 수를 초과하는 상황이 발생합니다. (물론 상황에 따라서는 아닐 수도 있습니다.) 아 이거 숫자 넘었으니 위반 아니야 하시겠지만 그게 아닙니다. 이런 상황이 된다면 IOCP는 자동으로 이 상황을 알아채고 이 이상의 쓰레드는 깨지 않도록 합니다. 즉 지정된 Concurrent Thread 수를 넘어선다면 쓰레드를 안 깬다고 보면 되네요. 하지만 이미 도는 쓰레드 수는 넘어섰습니다. 그러면 조금 있다가 이 쓰레드는 일을 처리하고 쓰레드 풀로 돌아가겠죠? 그럼 다시 Concurrent Thread 수 이하로 내려갑니다. 그래서 IOCP는 이런 식으로 작용을 함으로써 최소한의 시간 동안만 Concurrent Thread 수 이상의 쓰레드를 돌게 합니다. 그럼으로써 효율적으로 Thread Context Switching Overhead를 조절합니다.

그런데 이렇게 생각하실 분도 있으실 겁니다. PTL에서 RTL에 넘어갈 때에 숫자가 넘어가면 안 넘어가게 하면 되는 것이 아닌가? 라고요. 그럼 생각해보세요. 이미 깨는 조건이 완료되어 있는 쓰레드를 계속 멈추게 하고 있는 것이 효율적이겠습니까? 아님 약간의 오버헤드를 감수하고서라도 빨리 끝내도록 하는 게 효율적이겠습니까? 두 개의 상황을 따져본다면 후자가 더 이익임을 알 수 있는 겁니다.

자 그럼 Concurrent Thread 수에 대한 얘기까지 해봤고요. 여기서 IOCP의 최종 목표가 뭔지 이제 눈에 보이실 겁니다. 즉 IOCP의 최종 목적은 오버헤드는 최소한으로 줄이면서 CPU는 돌릴 수 있다면 최대한으로 놀리지 않고 돌리게 하는 겁니다. 그럼 성능은 최상이 되겠죠?

♤ WTQ – RTL – PTL

그럼 여기까지 WTL에 대해 알아봤고, WTQ, RTL, PTL에 대해 알아봤습니다. 그럼 마지막으로 할 것은 약간은 피상적으로 이해되었던 이들 간의 관계를 알아봐야겠죠? 이건 그림으로 알아보는 게 좋을 것 같으니 그림으로 한번 봅시다. 정리하는 기분으로 보세요.



자 이런 식으로 정리를 할 수 있겠습니다. 이제 좀 정리가 어느 정도 되시나요?

그럼 여기까지 해서 IOCP가 어떻게 동작해서 효율적인지를 알아 봤습니다.

여기까지 하고 나니 이제 뭘 얘기해야 할 지 감이 안 잡히는 군요. 일단 여기까지 끊도록 하겠습니다. 혹시나 좋은 고견 있으시다면 반영하도록 하겠습니다.