작성자(학번 이름) : 2019180052 한창우

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **주차** | 10주차 | **기간** | 2024.8. 26~ 9. 1 | **지도교수** |  |
| 이번주 한일 요약 | Windows Via C/C++  챕터 10 “동기 및 비동기 장치 I/O” | | | | |

<상세 수행내용>

**챕터 10: 동기 및 비동기 장치 I/O**

**I/O 요청에 대한 완료 통지의 수신**

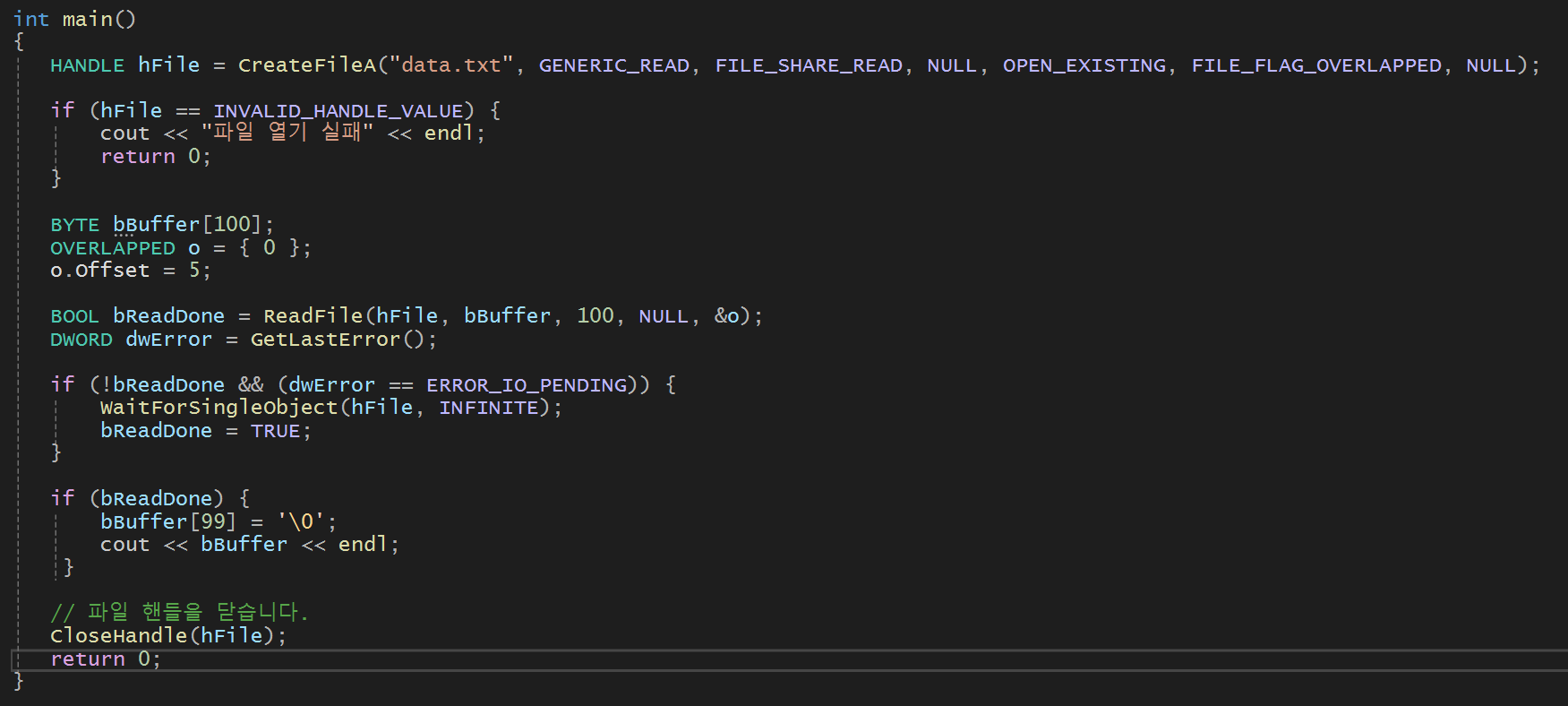
윈도우는 4가지의 방법으로 I/O 완료 통지를 받을 수 있다.

|  |  |
| --- | --- |
| **방법** | **요약** |
| 디바이스 커널 오브젝트의 시그널링 | 단일의 장치에 대해 다수의 I/O 요청을 수행하는 경우에는 적합하지 않다. 특정 스레드가 I/O 요청을 삽입하고 다른 스레드가 완료 통지를 수신할 수 있다. |
| 이벤트 커널 오브젝트의 시그널링 | 단일의 장치에 대해 다수의 I/O 요청을 수행할 수 있다. 특정 스레드가 I/O요청을 삽입하고 다른 스레드가 완료 통지를 수신할 수 있다. |
| 얼러터블 I/O 사용 | 단일의 장치에 대해 다수의 I/O 요청을 수행할 수 있다. 항상 I/O 요청을 삽입한 스레드가 완료 통지를 수신한다. |
| I/O 컴플리션 포트 사용 | 단일의 장치에 대해 다수의 I/O 요청을 수행할 수 있다. 특정 스레드가 I/O 요청을 삽입하고 다른 스레드가 완료 통지를 수신할 수 있다. 이 방법이 가장 확장성이 뛰어나고 유연성이 있다. |

위 4가지의 방법 중 **“I/O 컴플리션 포트”**를 사용하는 방법이 가장 뛰어나다.

**디바이스 커널 오브젝트의 시그널링**

디바이스 커널 오브젝트도 여타 다른 커널 오브젝트와 마찬가지로 시그널과 논시그널 상태를 가지기 때문에 스레드의 동기화에 사용될 수 있다. ReadFile과 WriteFile 함수를 통해 I/O 요청을 시도하면 요청이 삽입되기 전에 해당 디바이스 커널 오브젝트는 논시그널 상태가 된다. 그리고 디바이스 드라이버가 처리를 마치면 디바이스 커널 오브젝트는 시그널 상태로 변경된다.



OVERLAPPED 구조체의 멤버 변수인 Offset을 5로 설정하여, 해당 위치로부터 100바이트를 읽는 비동기 I/O 파일읽기 실습을 진행하였다.

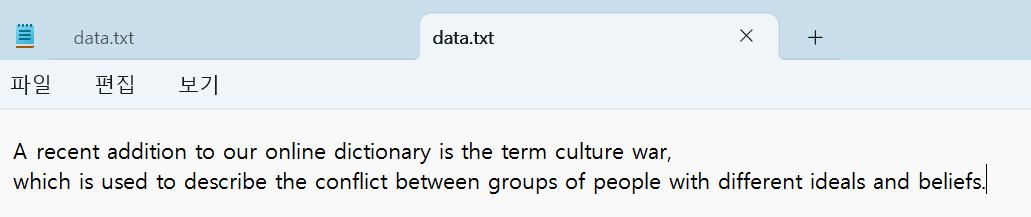
위 코드는 비동기 I/O 작업을 수행한 후 작업이 완료될 때까지 대기한다.

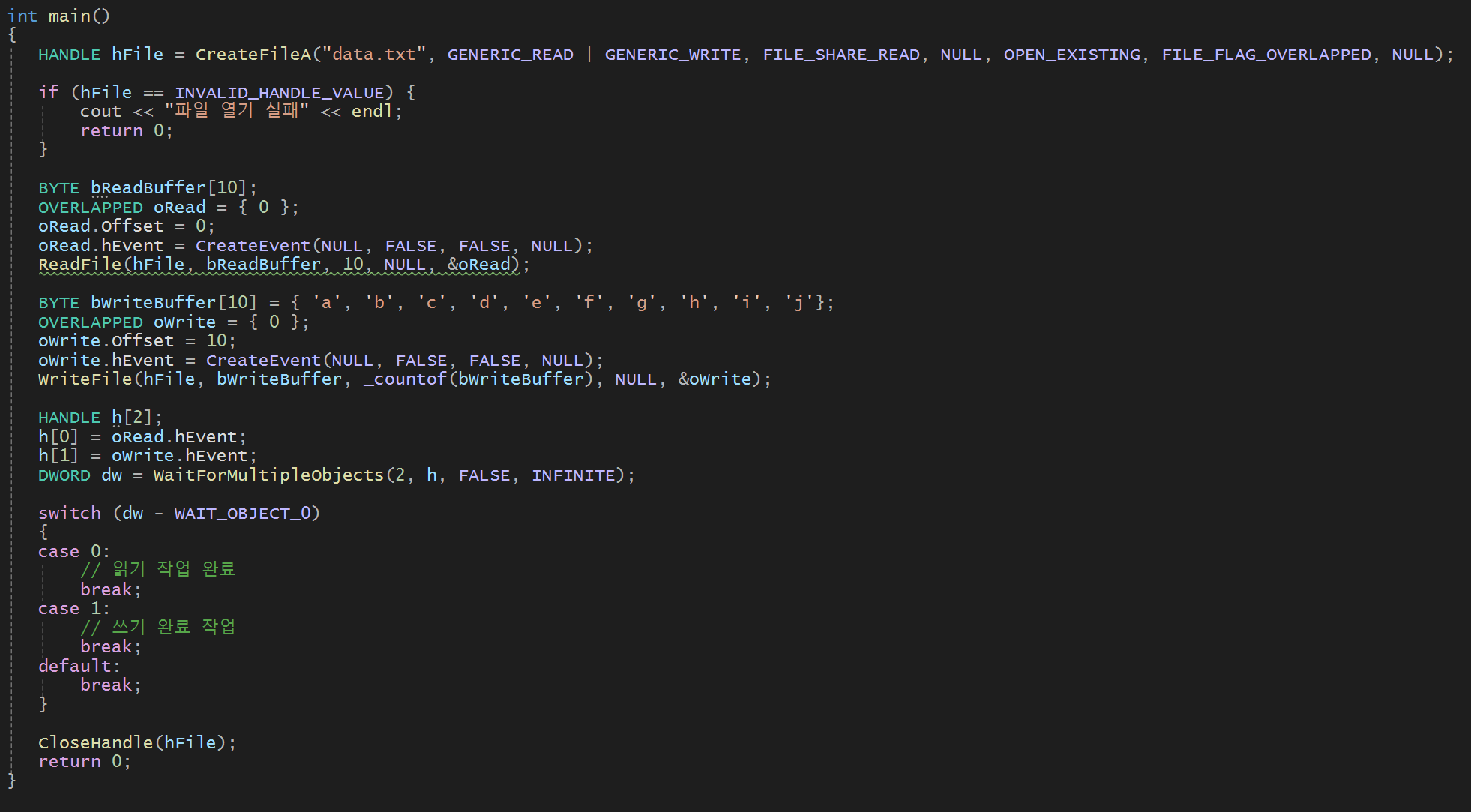
중요한 개념들을 다시 한번 정리해보기

* 비동기 I/O를 위해 디바이스를 열 때는 반드시 “FILE\_FLAG\_OVERLAPPED” 플래그를 사용한다.
* GetLastError로 얻은 값을 통해 작업이 비동기적으로 수행되었는지 알 수 있다. 반환 값이 “ERROR\_IO\_PENDING” 이라면 비동기적으로 수행된 것이다.
* 작업이 완료될 때까지 “WairForSingleObject”에서 스레드가 대기할 것이다.

**이벤트 커널 오브젝트의 시그널링**

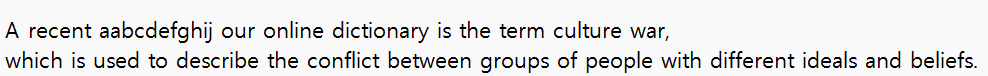
OVERLAPPED 구조체의 마지막 멤버인 hEvent는 이벤트 커널 오브젝트에 대한 핸들을 저장할 수 있다. 비동기 I/O 요청이 완료되면 디바이스 드라이버는 가장 먼저 OVERLAPPED 구조체의 hEvent 멤버가 NULL인지 여부를 확인하고, 아니라면 SetEvent를 호출하여 시그널 상태로 만들어준다. 만일 여러 번의 비동기 I/O 요청을 동시에 수행하기를 원한다면 각 요청마다 서로 다른 이벤트 커널 오브젝트를 생성해야하기 때문에 요청별로 OVERLAPPED 구조체를 생성해야한다.



위 텍스트 이미지는 수정 전 모습이다.

이벤트를 생성하여 기존 파일을 열어서 읽고, 쓰는 동작을 실행하는 코드이다.

파일을 열어서 읽고, 쓰는 동작은 모두 비동기 I/O 방식이다. 완료통지는 생성한 이벤트를 통하여 받고 있다.

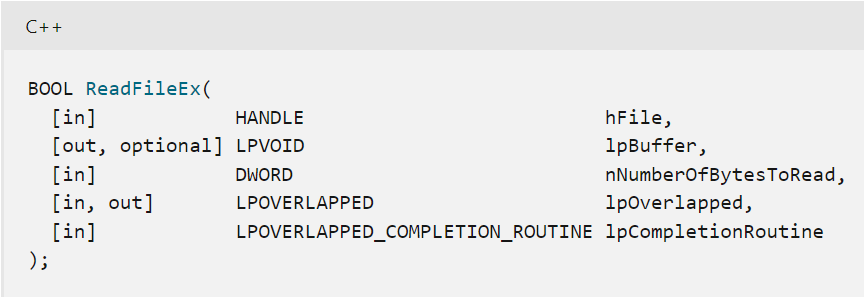


파일 포인터를 10으로 설정하여, 10바이트 위치부터 bWriteBuffer에 있는 내용을 bWriteBuffer 크기만큼 파일에 저장하였다.

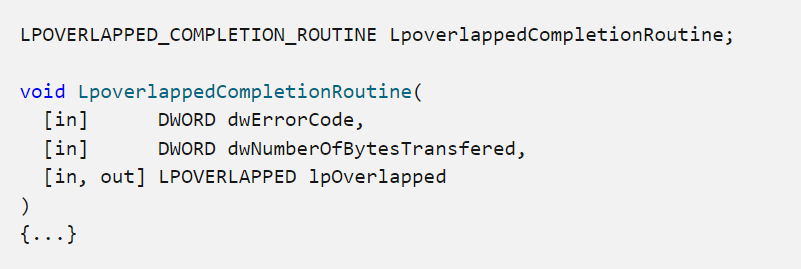
**얼러터블 I/O**

얼러터블 I/O를 고성능의 확장성 있는 애플리케이션을 개발할 사용할 수 있는 최상의 메커니즘이라고 소개했지만 실상은 그렇지 않았다. 그래서 마이크로소프트에서 이를 잘 동작할 수 있도록 운영체제에 상당히 유용한 몇몇 기능들을 추가하였다.

스레드가 생성되면 시스템은 각 스레드별로 비동기 프로시저 콜 (APC) 큐라고 불리는 큐를 하나씩 생성한다. 비동기 함수를 호출할 떄 **디바이스 드라이버에게 I/O 작업 완료 통지**를 스레드의 APC 큐에 삽입해 줄 것을 요철할 수 있다. 이를 위해서는 ReadFileEx 나 WriteFileEx 함수를 사용하면 된다.



ReadFileEx 함수의 매개변수를 살펴보면 ReadFile 함수와 다르게 **“컴플리션 루틴”** 이라고 불리는 콜백함수의 주소를 필요하다는 점을 볼 수 있다.



컴플리션 루틴은 반드시 위 그림과 같은 형태로 구현되어야 한다. 그리고 함수의 주소를 넘겨서 ReadFileEx 함수를 호출하게 되면 디바이스 드라이버에게 해당 콜백함수의 주소를 전달하게 된다.

디비아스 드라이버가 I/O 요청을 마치면 스레드의 APC 큐에 완료 통지를 나타내는 항목을 추가하는데, 이 항목에는 컴플리션 루틴의 주소와 최초 I/O 요청 시 사용되었던 OVERLAPPED 구조체의 주소가 포함되어 있다.

얼러터블 상태에 돌입한 스레드는 APC큐에 있는 모든 항목에 대해 컴플리션 루틴을 호출해 준다. 이때 I/O 에러 코드, 송수신된 바이트 수, OVERLAPPED 구조체의 주소가 전달된다.

윈도우는 스레드를 얼러터블 상태로 변경할 수 있는 6개의 함수를 제공한다.

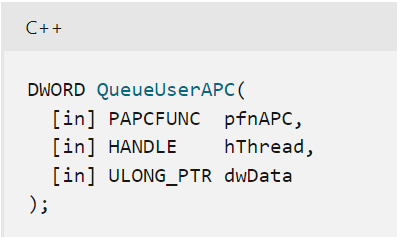
* SleepEx
* WaitForSingleObjectEx
* WaitForMultipleObjectEx
* SignalObjectAndWait
* GetQueuedCompletionStatusEx
* MsgWairForMultipleObjectsEx

**얼러터블 I/O의 장단점**

**콜백함수**: 얼러터블 I/O는 콜백함수를 필요로 하며, 이는 코드를 더욱 이해하기 어렵게 만든다.

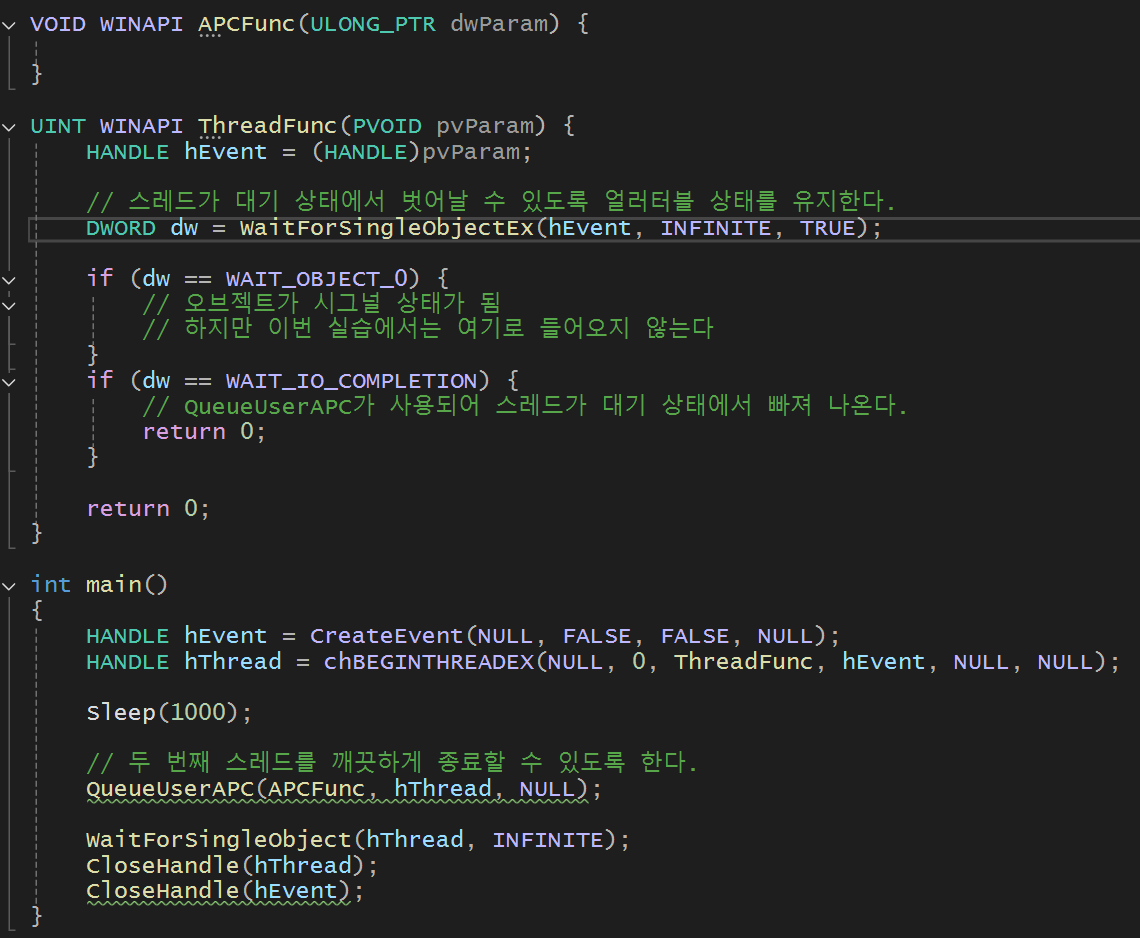
**스레딩 문제**: 얼러터블 I/O의 가장 큰 문제점은 I/O 작업을 요청한 스레드가 반드시 완료 통지도 함께 처리해야 한다는 것이다. 즉, 아무 작업을 하고 있지 않는 스레드가 존재함에도 불구하고 I/O 요청을 한 스레드가 반드시 다 처리해야 하는 것이다. 이렇듯 부하 분산을 제대로 수행하지 못하는 문제로 인해 얼러터블 I/O를 이용하여 확장성 있는 애플리케이션을 만드는 것은 매우 어렵다.

윈도우는 사용자가 임의로 APC 큐에 항목을 추가할 수 있는 함수를 제공하고 있다.



위 함수를 사용하면 스레드를 대기 상태에서 강제로 빠져나오게 할 때에도 유용하게 사용될 수 있다. 첫 번째 매개변수로는 APC 콜백함수를 가리키는 포인터를 전달하면 되는데, 이 함수의 원형은 다음과 같다.





QueueuUserAPC 함수를 사용하여 대기 중인 스레드를 강제로 깨우는 실습을 진행해봤다.

main 함수에서 두번째 스레드를 생성하여 해당 스레드가 얼러터블 상태에 돌입할 수 있도록 한다. 그리고 main에서 두번째 스레드 핸들을 이용하여 해당 스레드의 **APC 큐**에 항목을 추가한다.

그렇다면 두번째 스레드는 자신의 **APC 큐**에 추가된 항목을 처리하기 위해 대기상태에서 나올 것이고 콜백함수인 **APCFunc** 함수로 가서 작업을 처리할 것이다. 그리고 돌아와서 **WaitForSingleObjectEX** 함수가 **WAIT\_IO\_COMPLETION** 플래그를 반환해 줄 것이다. 그렇다면 두번째 스레드도 안전하게 종료될 수 있다.

**I/O 컴플리션 포트**

전통적으로 서비스 애플리케이션은 다음의 두 가지 아키텍처 중 하나의 형태로 설계되어 왔다.

**시리얼 모델:** 하나의 스레드가 사용자의 요청을 대기한다. 사용자의 요청이 들어오면 대기하던 스레드가 꺠어나 클라이언트의 요청을 처리한다.

**컨커런트 모델:** 하나의 스레드는 사용자의 요청을 대기하고, 사용자의 요청을 처리하기 위해 새로운 스레들르 생성한다. 새로운 스레드가 요청을 처리하는 동안에, 원래의 스레드는 다시 다른 사용자의 요청을 기다린다. 사용자의 요청을 처리하던 스레드는 작업이 완료되었을 때 종료된다.

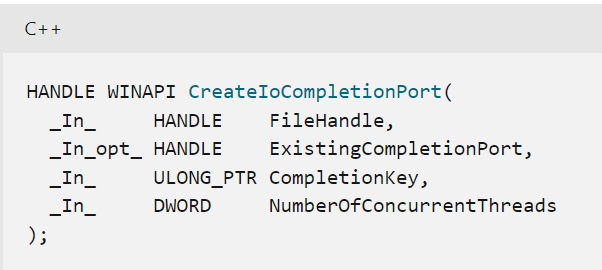
시리얼 모델은 하나의 스레드가 요청과 처리를 모두 맡고 있기 때문에 멀티프로세서를 가진 머신의 성능을 활용할 수 없다. 이러한 이유 때문에 컨커런트 모델이 일반적으로 좀 더 많이 사용된다.

**하지만 컨커런트도 처음에는 기대한 만큼 성능이 좋진 않았다…**

수많은 클라이언트 요청이 동시에 들어왔을 때 시스템이 수많은 스레드들을 동시에 수행해야 한다. 대부분의 스레드들이 스케줄 가능 상태였기 때문에 이러한 스레드들 간의 컨텍스트 전환을 수행하느라 윈도우 커널이 너무 많은 시간을 허비하고 있으며, 각각의 스레드들은 충분한 CPU 시간을 받지 못하는 점도 있었다. 그래서 마이크로소프트에서 이러한 부분을 개선하기 위해 만든것이 **“I/O 컴플리션 포트”** 이다.

**I/O 컴플리션 포트 생성**

IOCP를 구현한 이론적 배경은 **동시에 수행할 수 있는 스레드 개수의 상한을 설정할 수 있어야 한다**는 것이다. 그래서 애플리케이션을 초기화할 때 스레드에 대한 풀을 생성하여 종료 시까지 유지시키면 성능을 상당부분 개선할 수 있다. 실제로 IOCP는 스레드 풀을 이용하도록 설계되었다.



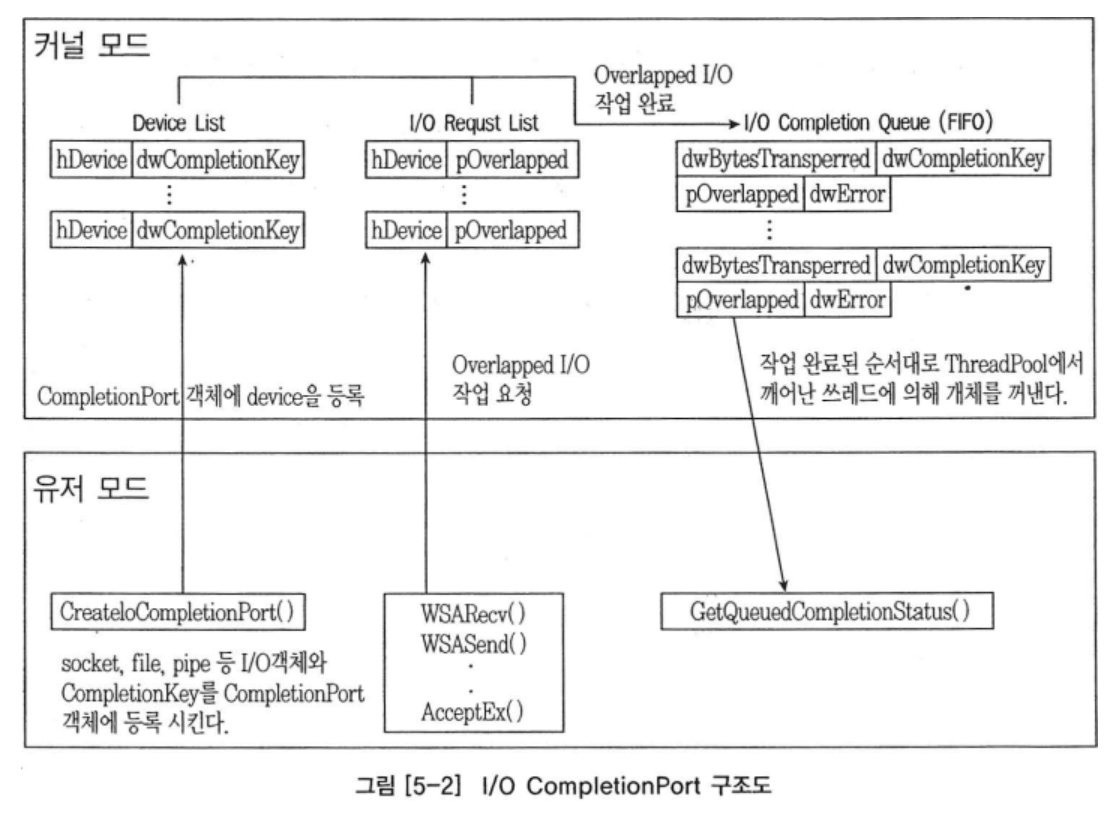
위 함수는 2가지 역할을 한다.

* IOCP를 생성
* 장치와 IOCP 연계 작업도 수행
* 매개변수 NumberOfConcurrentThreads는 IOCP에게 동일 시간에 동시에 수행할 수 있는 스레드의 최대 개수를 알려주는 역할을 한다.

**장치와 I/O 컴플리션 포트 연계**

IOCP를 생성하면 윈도우 커널은 내부적으로 5개의 서로 다른 데이터 구조를 생성한다.

* 장치 리스트
* I/O 컴플리션 큐(I/O Completion Queue, FIFO)
* 대기 스레드 큐(Waiting Thread Queue, LIFO)
* 릴리즈 스레드 리스트
* 일시 정지 스레드 리스트



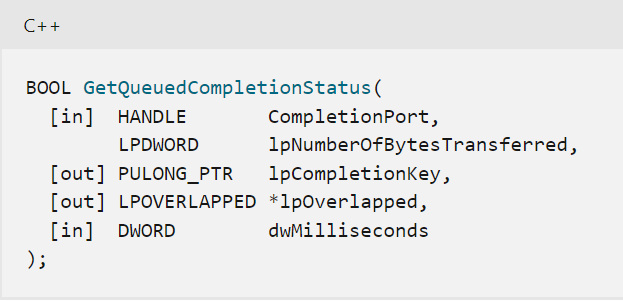
**(위 그림은 작업일지 6주차에 사용했던 자료이다.)**

장치에 대한 핸들(파일, 소켓 등등) 그리고 컴플리션 키값을 전달하여 CreateIoCompletionPort 함수를 호출하면 IOCP의 내부적인 데이터 구조인 **“장치 리스트”**에 새로운 항목을 추가한다.

IOCP를 구성하는 두 번째 데이터 구조는 “**I/O 컴플리션 큐**” 이다. 장치에 대한 비동기I/O 요청이 완료되면 시스템은 장치와 연계된 IOCP가 있는지 확인하고 있다면, 연계된 “I/O 컴플리션 큐”에 완료 통지를 나타내는 새로운 항목을 삽입한다. 각각의 항목에는 **송수신된 바이트 수**, 장치와 IOCP를 연계할 때 지정한 **컴플리션 키 값**, 비동기I/O 작업을 요청할 때 사용하였던 **OVERLAPPED 구조체를 가리키는 포인터** 그리고 **에러 코드**를 가지고 있다.

**I/O 컴플리션 포트를 이용한 아키텍처 설계**

비동기 장치 I/O 작업이 완료되어 I/O 컴플리션 포트를 통해 완료 통지가 전달될 때 이를 곧바로 처리할 수 있도록 스레드를 대기상태로 유지해야 하는데, 이를 위해 GetQueuedCompletionStatus 함수를 사용하면 된다.



IOCP를 구성하는 세 번째 데이터 구조는 **“대기 스레드 큐”** 이다. GetQueuedCompletionStatus 함수를 호출한 스레드는 대기 스레드 큐에 삽입되며, 이를 통해 IOCP 커널 오브젝트는 어떤 스레드들이 비동기 I/O 요청에 대한 완료 통지를 처리할 것인지를 알 수 있다.

대기 스레드 큐는 LIFO 방식이기 때문에 가장 마지막으로 GetQueuedCompletionStatus 함수를 호출한 스레드가, I/O 컴플리션 큐에 일감이 들어오면 가장 먼저 일어나서 이를 처리하게 된다.

그리고 적절하게 처리 후 다시 GetQueuedCompletionStatus 함수를 호출하여 대기 스레드 큐에 삽입된다. 이렇게 후입선출 알고리즘을 이용하면 스케줄되지 않는 스레드들이 사용하는 메모리를 디스크로 내보낼 수 있으며, 프로세서의 캐시를 비울 수도 있다.

**I/O 컴플리션 포트의 스레드 풀 관리 방법**

IOCP에서 관리되는 스레드의 수는 사용자가 IOCP를 생성할 때 입력한 스레드 수로 결정된다. 또는 0을 전달하여 IOCP를 생성했을 경우 머신의 CPU 개수와 동일한 수로 설정된다.

그렇게 생성된 IOCP의 컴플리션 큐에 완료 통지가 삽입되면 IOCP는 대기중인 스레드를 깨우는데, 설정된 개수를 초과할 수 없다. 예를 들어, IOCP를 생성할 때 값을 2로 설정하여 생성했고 GetQueuedCompletionStatus 함수를 호출하여 대기 중인 스레드는 4개라고 가정한다. 만약 컴플리션 큐에 4개의 일감이 도착해도 2개의 스레드만 일어나서 일을 처리하게 되는 것이다.

그렇다면 4개의 스레드가 있더라도 2개의 스레드는 무조건 일을 하지 않고 놀 것처럼 보인다.

하지만 IOCP는 상당히 지능적으로 동작한다. GetQueuedCompletionStatus 함수를 호출하여 대기 중이던 스레드가 컴플리션 큐에 일감이 들어와서 꺠어나면, 시스템은 IOCP 내부적으로 관리되는 네 번째 자료 구조인 **릴리즈 스레드 리스트**에 깨어난 스레드의 ID를 기록해 둔다. 이로써 IOCP는 어떤 스레드가 깨어났는지를 알 수 있으며, 지속적으로 확인할 수 있게 된다.

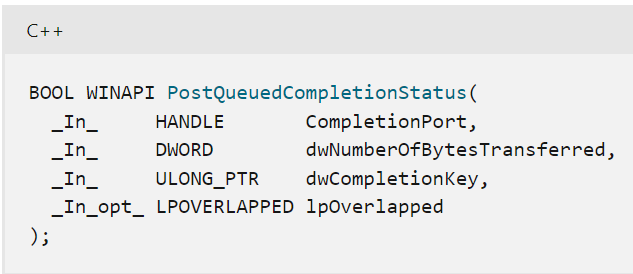
만약 릴리즈 스레드 리스트에 삽입된 스레드 중 하나가 어떤 함수를 호출하였더니 대기 상태로 진입하게 되었다고 하자. 그렇다면 IOCP에서 이를 감지하고 해당 스레드의 ID 값을 빼내어 IOCP의 내부적으로 관리되는 다섯 번째 자료 구조인 **“일시 정지 스레드 리스트”**로 옮긴다.

이렇게 되면 릴리즈 스레드 리스트의 항목 개수가 줄게 되므로 IOCP는 대기 상태에 있는 스레드 중 하나를 릴리즈 스레드 리스트로 옮겨온다. 만약 대기 상태에 있던 스레드가 다시 깨어난다면 일시 정지 스레드 리스트에 있던 스레드를 다시 릴리즈 스레드 리스트로 옮긴다. 그렇다면 릴리즈 스레드 리스트는 IOCP에서 설정한 동시에 수행 가능한 스레드의 개수를 일시적으로 초과하는 개수의 항목을 가지게 된다.

**“중요한 것은 IOCP는 동시에 수행 가능한 스레드의 개수를 초과하지 않는 범위 내에서만 스레드 풀로부터 스레드를 깨워서 작업을 수행한다는 것이다.”**

**I/O 완료 통지 흉내 내기**

앞서 얼러터블 I/O 모델을 공부할 때, QueueUserAPC 함수를 사용하여 스레드간의 통신하는 방법에 대해서 공부했었다. IOCP도 마찬가지로 이런 유사한 기능을 제공하는데 그 함수 이름은 PostQueuedCompletionStatus 함수이다.



이 함수를 호출하면 완료 통지를 I/O 컴플리션 큐에 삽입해 준다. 즉, 비동기 I/O 작업의 완료로 시스템이 큐에 일감을 넣었을 때 스레드가 대기 상태에서 벗어나는 방법 말고도 사용자가 직접 대기 중인 스레드를 깨울 수 있게 되는 것이다.

--------------------------------------------------------------------------------------------------------------

- 아래 링크는 책과 관련된 영상이 아닌, 인프런 강의를 수강하여 배운 내용에 대한 실습을 올린 영상 -

진행 상황 유튜브 링크: <https://youtu.be/N03b3-_yfFU>

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | | |
| **해결방안** |  | | |
| **다음주차** | 11주차 | **다음기간** | 9. 2 ~ 9. 8 |
| **다음주 할일** | 11주차에 계획되어 있는 내용 | | |
| **지도 교수**  **Comment** |  | | |