작성자(학번 이름) : 2019180052 한창우

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **주차** | 5주차 | **기간** | 2024.7. 22 ~ 28 | **지도교수** |  |
| 이번주 한일 요약 | Windows Via C/C++  챕터 8 “유저 모드 스레드 동기화” | | | | |

<상세 수행내용>

챕터 8. 유저 모드에서의 스레드 동기화

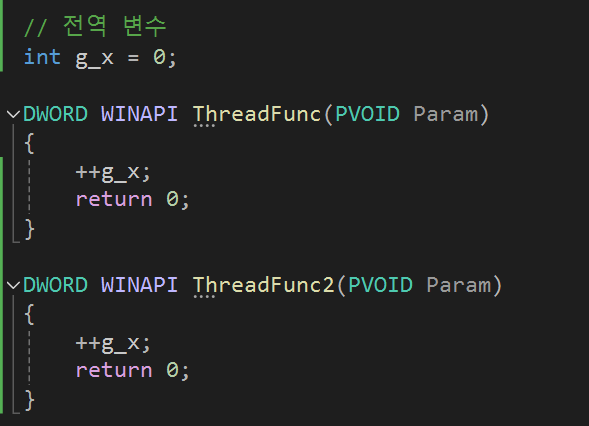
스레드는 동일 리소스를 사용해야하는 경우, 다른 스레드가 사용 중이라면 접근할 수 없게 된다. 한 스레드는 읽기를 수행하고 다른 스레드는 쓰기를 수행한다면, 이는 마치 다른 사람이 수정 중인 책을 읽고는 있는 것이다.

스레드는 상호 통신을 수행해야 한다.

* 다수의 스레드가 공유 리소스에 접근해야 하며, 리소스가 손상되지 않도록 해야하는 경우
* 어떤 스레드가 하나 혹은 다수의 다른 스레드에게 작업이 완료되었음을 알려야 하는 경우

**원자적 접근: Interlocked 함수들**

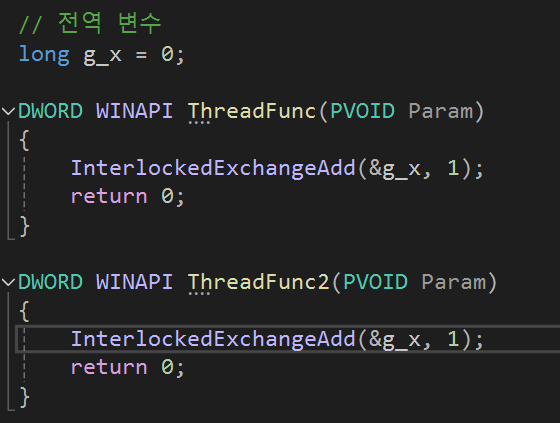
**원자적 접근**: 어떤 스레드가 특정 리소스에 접근할 때 다른 스레드는 동일 시간에 동일 리소스에 접근할 수 없는 것



위 전역변수 “g\_x” 를 1씩 증가시키는 두 개의 스레드가 있다. 그리고 동시에 스레드를 실행한다면 기대하던 결과값 2가 나올 수도 있고 안 나올 수도 있다.

원도우는 선점형 멀티스레드 운영체제이라서, 언제든지 수행 중에 다른 스레드로 제어권을 빼앗길 수 있다. 이러한 것은 컴파일러가 어떻게 컴파일을 수행하느냐에 따라 그리고 CPU가 생성된 코드를 어떻게 수행하느냐에 따라 각기 달라질 뿐더러 프로그램을 수행하는 컴퓨터에 CPU가 몇 개 설치되어 있느냐에 따라서도 그 결과가 달라진다는 것이다. 이것은 수행 환경과 관련된 것이기 때문에 우리가 제어할 방법은 없지만, 윈도우에서는 방법을 제공해 주고 있다.

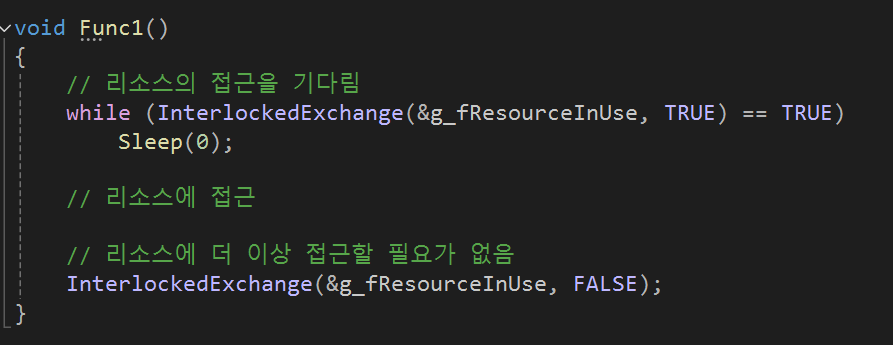
이러한 것 중 하나가 인터락 계열 함수들이다.



위 InterlockedExchangeAdd 함수를 통해 공유되는 변수 데이터의 값을 증가 시켜야 한다.

**(주의 : 함수에 전달하는 주소 값은 반드시 정렬되어 있어야 한다, “데이터 정렬” 은 13장에서 다룸)**

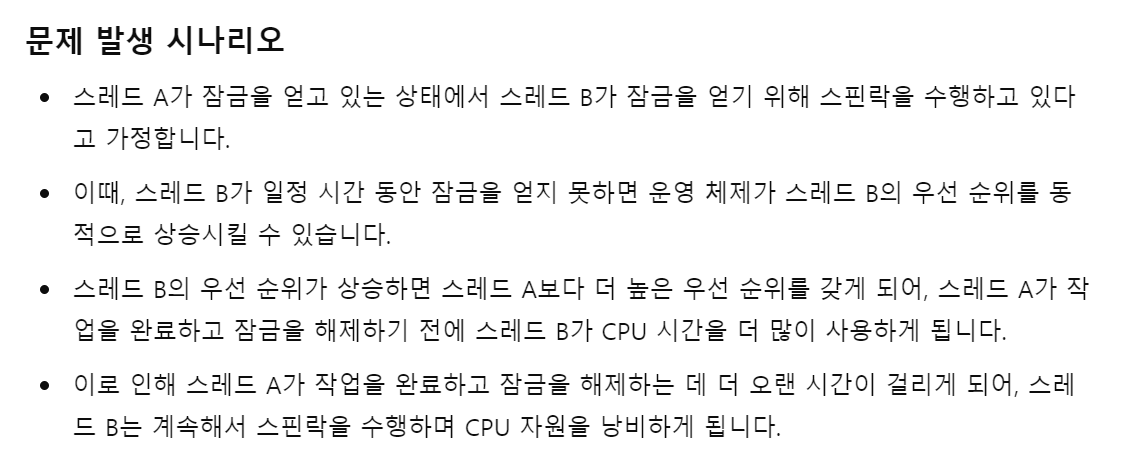
또 다른 함수.



위 함수는 InterlockedExchange이다. 이 함수는 첫번째 매개변수 값을 2번째 매개변수 값으로 변경하고, 이전의 값을 반환한다. 이처럼 스핀락을 구현해야하는 경우 유용하게 사용할 수 있는 함수이다. 스핀락 기법은 CPU의 사용량을 많이 차지하기에 적절한 상황에서 사용해야 한다.

또한 위와 같은 코드는 스핀락을 사용하는 모든 스레드가 동일한 우선순위 레벨에 있는 것으로 가정하고 있다. 또한 스핀락을 수행하는 스레드들은 모두 스레드 우선 순위 동적 상승 기능이 불가능하도록 설정해야 한다.

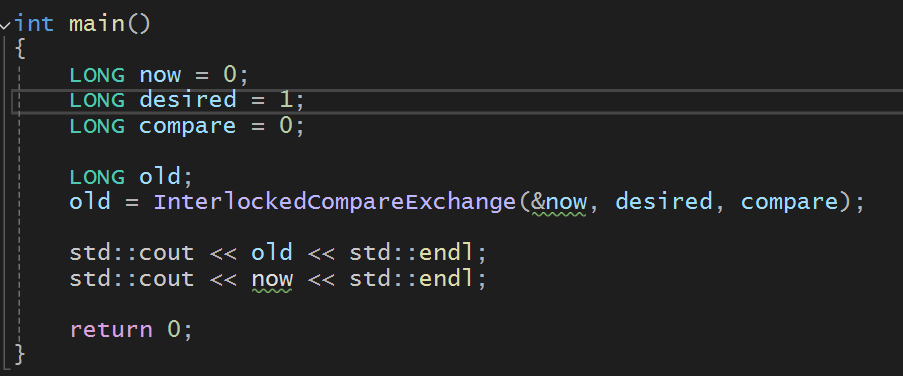
(우선 순위 동적 상승 기능을 불가능하도록 설정해야 하는 이유)



**(gpt 검색)**

스핀락을 통해 리소스 접근 시간이 너무 길어지면 커널 모드로 스레드를 전환(CPU 시간을 소비하지 않는)해서 대기하는 것이 좀 더 효과적이다. 이러한 방법이 “**크리티컬 섹션**” 의 구현 방식이기도 하다.

또 다른 부류의 인터락 함수

위 **InterlockedCompareExchange** 함수는 두 개의 값을 비교해서 일치하면 원하는 값으로 변경하고, 반환값으로는 변경되기 이전의 값을 반환한다. 이 과정을 원자적으로 수행한다.

지금까지 인터락 부류의 함수들을 알아보았다. 마이크로소프트에서는 더 많은 인터락 함수들을 제공하지만 지금까지 알아본 함수들만 잘 이용한다면 다른 함수들이 수행하는 모든 작업들을 수행할 수 있다.

**“인터락 싱글 링크드 리스트”**

정수 값이나 부울 값을 원자적으로 다룰 수 있는 방법과 더불어 인터락 싱글 링크드 리스트라고 불리는 스택을 제공하고 있다. 이 자료구조를 이용하면 푸시와 팝을 원자적으로 수행할 수 있다.

**캐시 라인**

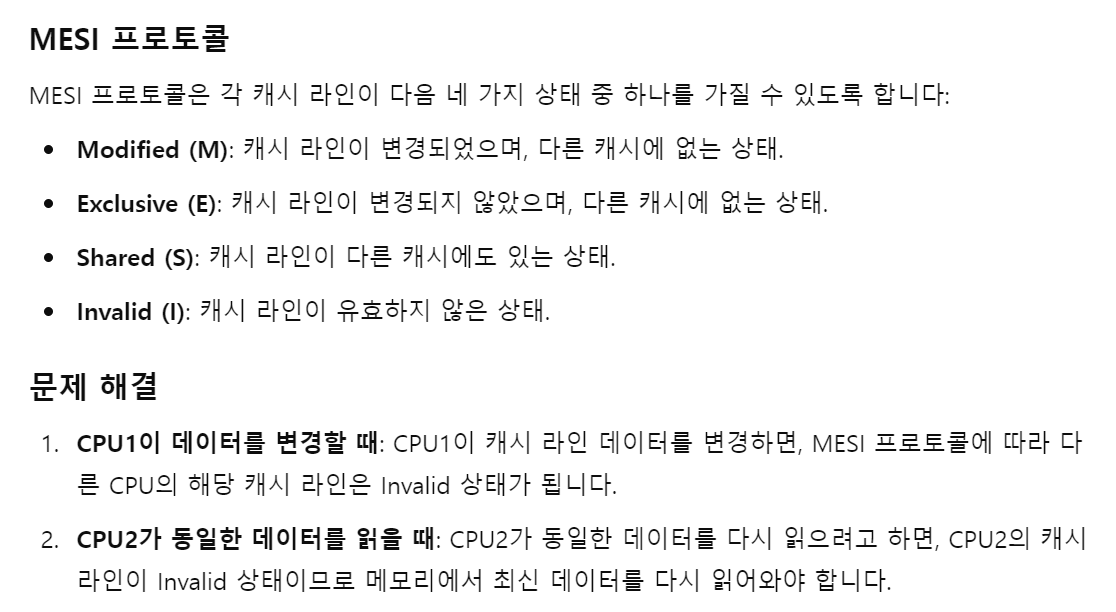
CPU가 메모리로부터 값을 가져올 때는 바이트 단위로 값을 가져오는 것이 아니라 캐시 라인을 가득 채울만큼 충분한 양을 한 번에 가져온다. CPU 마다 다르지만 캐시라인은 32, 64, 128 바이트 크기로 구성되어 있다. 그래서 각기 32, 64, 128 바이트 경계로 정렬되어 있다.

캐시는 성능 향상을 위해 존재한다. 보통의 애플리케이션의 경우 인접한 바이트들을 자주 사용하는 경향이 있는데, 만일 인접한 바이트들이 캐시 라인에 존재한다면 데이터를 찾는 CPU 입장에서는 메모리에 추가적으로 접근할 필요가 없어서 효율적이다. 하지만 멀티프로세스 환경에서 메모리의 갱신을 어렵게 만든다.

1. CPU1이 메모리의 특정 위치에서 읽고자 하는 바이트와 인접한 바이트도 같이 CPU1 캐시 라인으로 들어온다.
2. CPU2도 동일한 위치로부터 1바이트를 읽는다. CPU1 캐시 라인에 존재하는 바이트오 ㅏ동일한 내용이 CPU2 캐시 라인에도 들어온다.
3. CPU1이 내용을 변경한다. 이러한 밴경 내용은 CPU1 캐시 라인 내의 내용을 변경하게 될 것이다. 하지만 실제 램에는 아직 변경된 내용이 쓰여지지 않는다.
4. CPU2가 동일한 ㅜ이치로부터 1바이트를 다시 읽어오려고 시도한다. CPU2의 캐시 라인에 이미 읽고자 하는 바이트가 들어 있으므로 메모리에 추가적으로 접근할 필요가 없다. 하지만 CPU2는 CPU1이 변경한 내용을 알 수 없다.

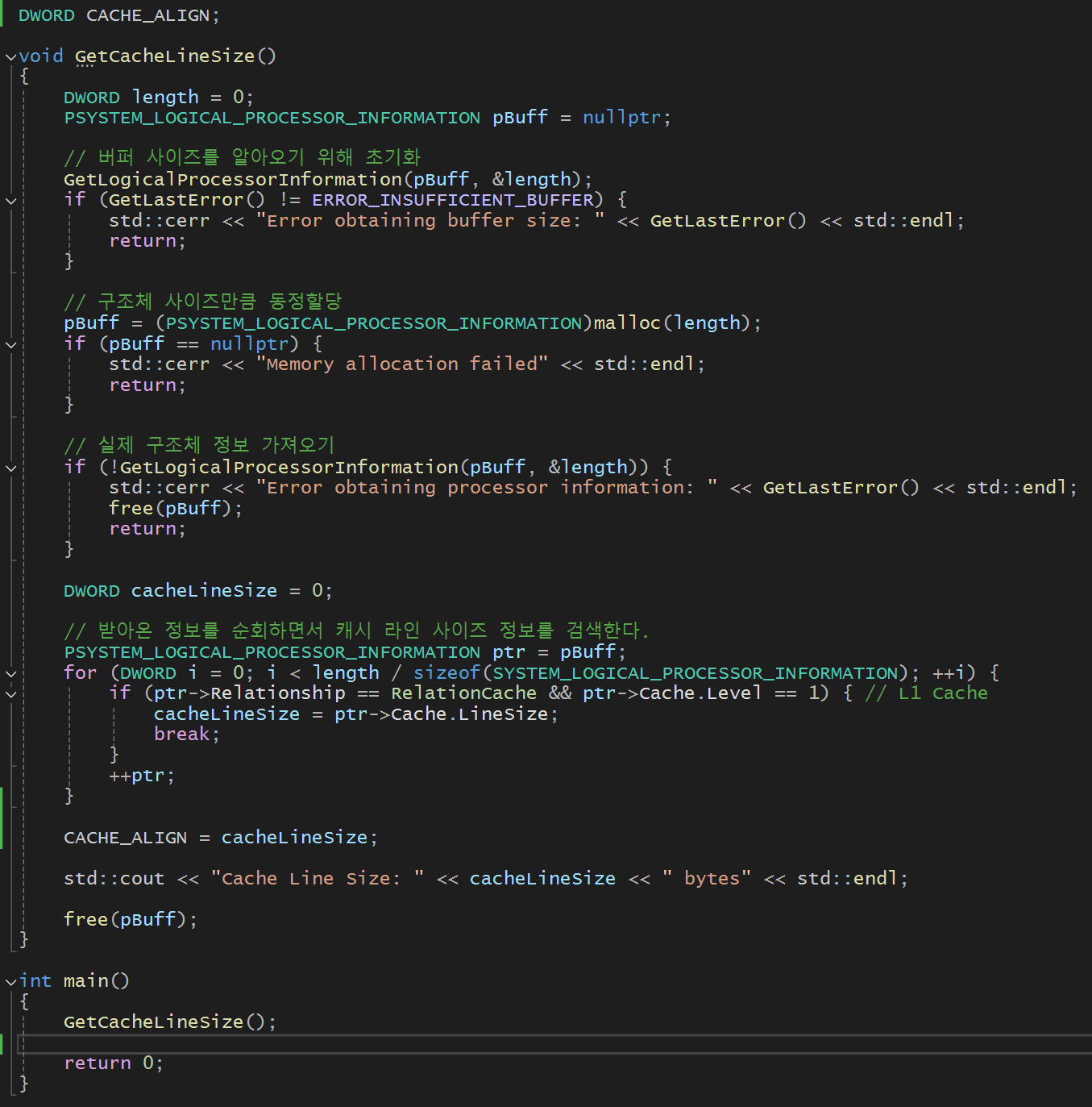
위오 같은 시나리오를 극복할 수 있도록 CPU는 설계되어있다.

아래는 챗GPT에 검색한 결과이다.

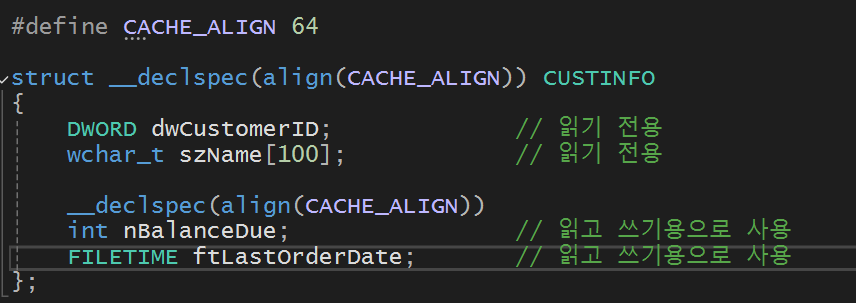


즉, 캐시 라인 변경 시 다른 CPU 캐시 라인을 적절히 Invalid 상태로 만들어 최신 데이터를 유지하도록 보장해 준다.

이러한 특성 때문에 애플리케이션이 사용하는 데이터는 캐시 라인의 크기오 ㅏ그 경계 단위로 묶어서 다루는 것이 좋다. 이렇게 함으로써 적어도 하나 이상의 캐시라인 경계로 분리된 서로 다른 메모리 블록에 각각의 CPU가 독립적으로 접근하는 것을 보장할 수 있게 된다.



위는 책에서 소개된 CPU의 캐시 라인 크기를 얻어오는 가장 쉬운 함수들이고, GPT의 도움을 받아 실습을 해봤습니다.



구조체의 멤버 데이터를 위 그림처럼 캐시라인에 정렬하게되면 캐시 라인 무효화 상황을 해결할 수 있다.

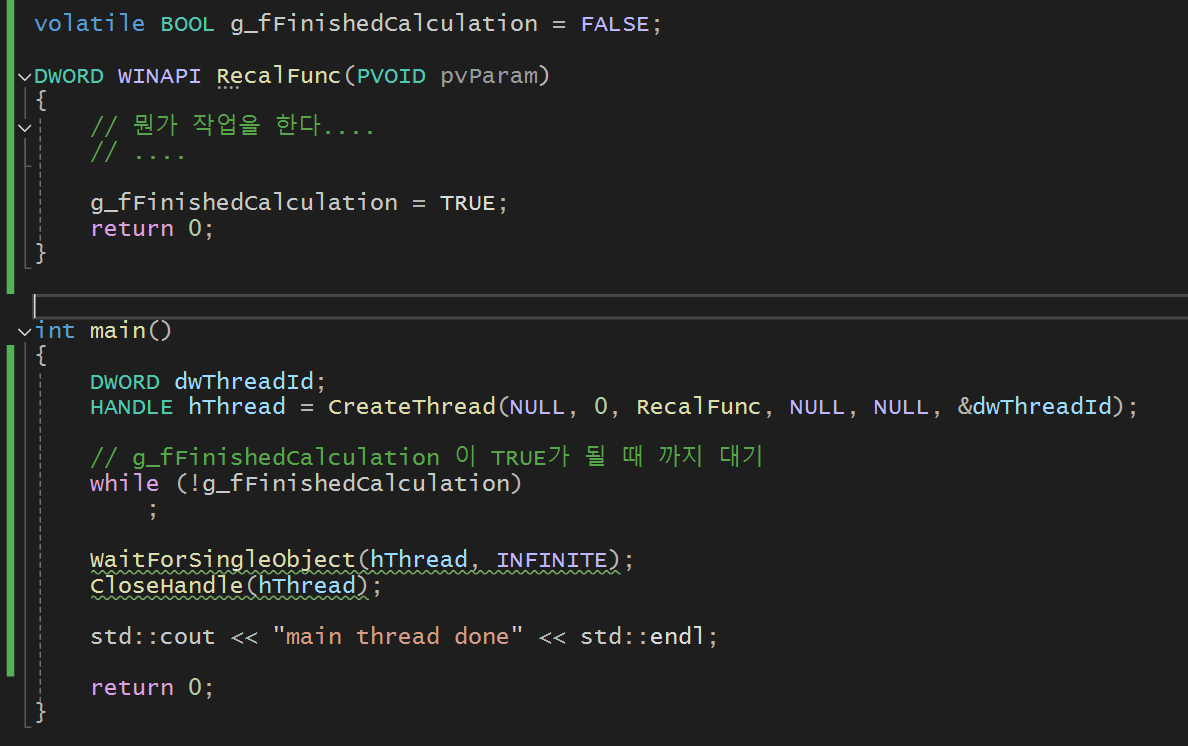
**고급 스레드 동기화 기법**

인터락 계열 함수들은 하나의 값에 원자적으로 접근하는 경우 훌륭하게 동작한다. 가장 먼저 인터락 계열함수로 해결할 수 있는지 검토해야하는데, 대부분의 애플리케이션은 32비트나 64비트 값 보다 복잡한 자료 구조를 다룬다. 이 때는 인터락 함수는 고려될 수 없으며 윈도우에서 제공하는 다른 기능을 사용해야한다.

그래서 복잡한 자료 구조나 64비트를 초과하는 데이터를 원자적으로 처리하려면, 다음과 같은 동기화 기법을 사용해야 합니다:

* **크리티컬 섹션(Critical Section):** 스레드가 특정 코드 블록에 진입할 때 다른 스레드가 동시에 진입하지 못하도록 보호합니다.
* **뮤텍스(Mutex):** 크리티컬 섹션과 유사하지만, 프로세스 간 동기화도 가능합니다.
* **스핀락(Spinlock):** 스레드가 잠금이 해제될 때까지 바쁘게 대기(spin)합니다. 잠금 시간이 짧을 때 유용합니다.

**회피 기술**



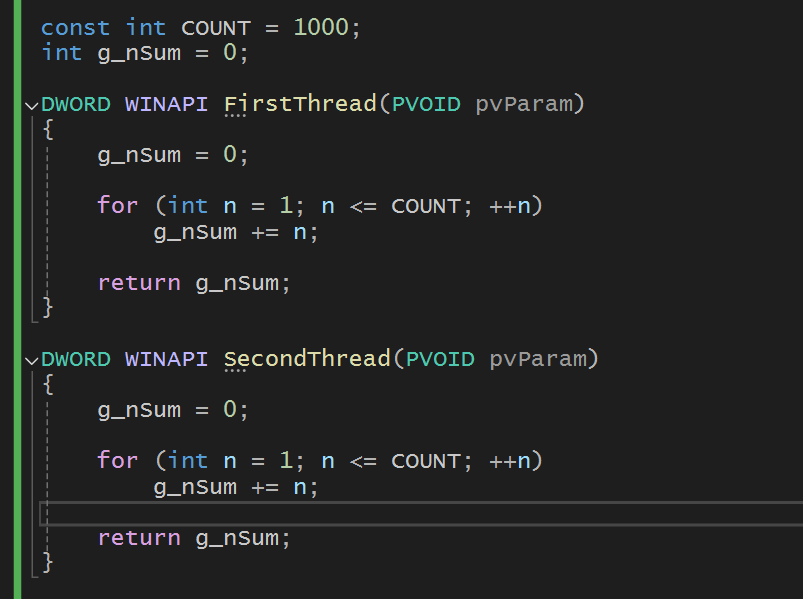
위 같은 방법으로 동기화를 수행할 수 있지만 절대 좋은 방법은 아니다.

위는 변수의 상태를 지속적으로 폴링 하여 다른 스레드가 작업을 완료했는지의 여부를 확인하는 동기화 기법이다. 이러한 방법이나 스핀락 보다 더 좋은 방법은 스레드가 필요로 하는 자원이 가용 상태가 될 때 까지 대기하는 것이다.

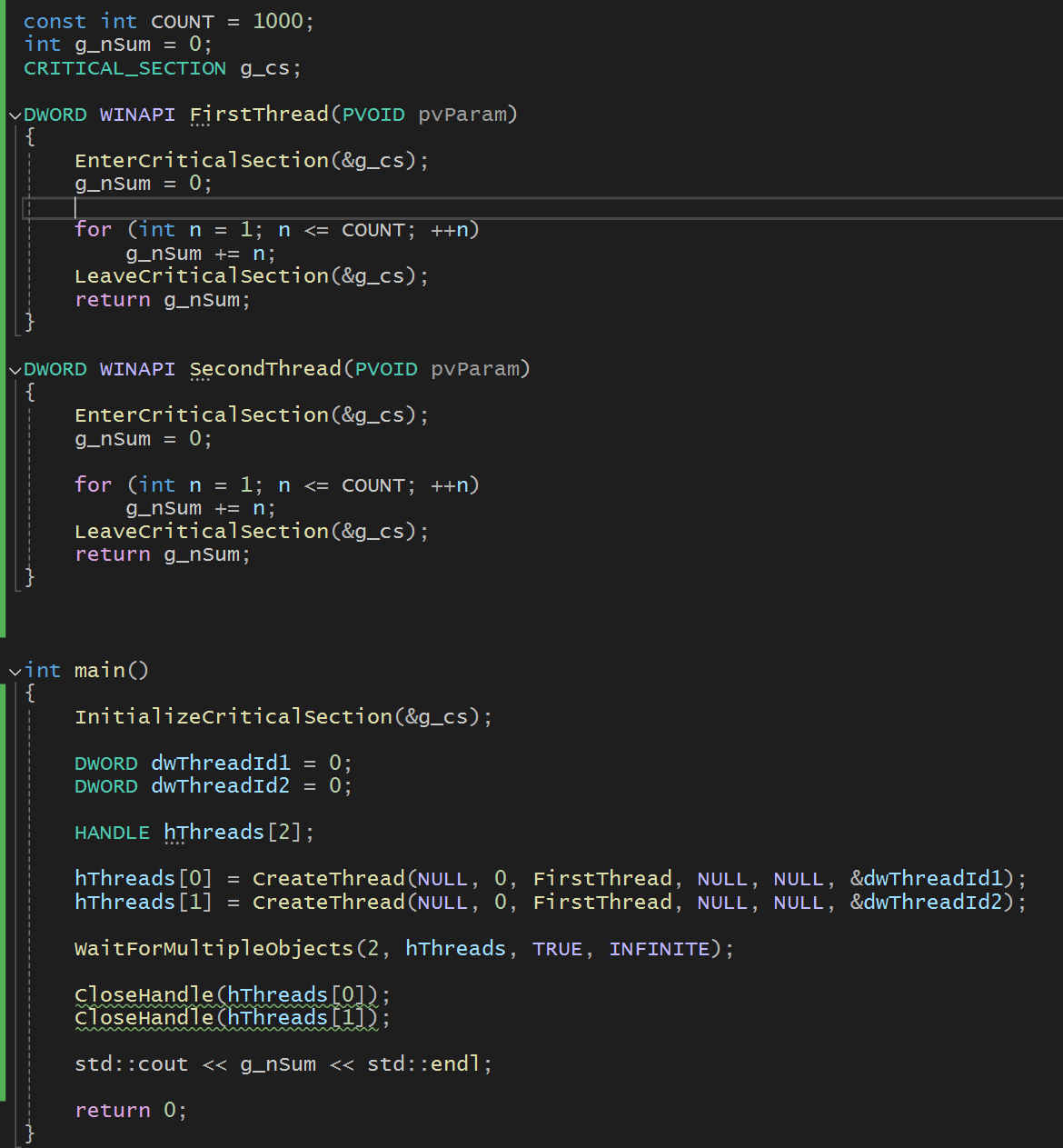
위에 volatile 이라는 키워드는 컴파일러에게 해당 데이터는 외부에서 그 내용이 변경될 수 있음을 알려주고, 그 어떠한 최적화도 수행하지 못하게 하는 것이다. Volatile이 지정되지 않으면 CPU는 레지스터에 한번 로드하고 그 값을 필요할 때 마다 참조하게 된다. 성능에 효율성은 좋지만 경우에 따라 반드시 메모리에서 그 값이 다시 참조해야하는 경우라면 volatile 키워드를 사용하면 된다.

**크리티컬 섹션**

크리티컬 섹션이란 공유 리소스에 대해 배타적으로 접근해야 하는 작은 코드의 집합을 의미한다.

즉, 공유 리소스를 다루는 여러 줄의 코드를 “원자적”으로 수행하기 위한 방법이다.

위는 공유변수를 접근해야하는 2개의 스레드가 실행하는 코드의 모습이다.



위 코드는 크리티컬 섹션 구조체를 사용하여 동기화를 진행하는 코드이다. 공유해야하는 리소스가 여러 개이라도 해당 크리티컬 섹션내에 포함시키면 된다. Enter를 했으면 사용중이니 다른 스레드는 접근 불가이다. 그리고 LeaveCriticalSection을 반드시 호출하여 다른 스레드에게 더 이상 공유 리소스에 볼 일이 없다는 것을 알려야한다.

인터락 함수로 사용할 수 없는 경우에는 위 크리티컬 섹션을 사용하면 된다. 하지만 서로 다른 프로세스에 존재하는 스레드 사이의 동기화에는 사용할 수 없다는 치명적인 단점이 있다.

**크리티컬 섹션과 스핀락**

다른 스레드가 이미 진입한 크리티컬 섹션을 진입을 시도하면, 스레드는 바로 대기상태에 들어간다. 이는 스레드가 유저 모드에서 커널 모드로 전환되는 컨텍스트 스위칭을 뜻하며 이 작업은 비용이 비싼 동작에 해당된다. 만약 스레드가 커널 모드로 완전히 전화하기도 전에, 공유 리소스를 차지하고 있는 스레드가 매우 짧은 시간 이내에 반환한다면, 상당한 CPU 시간이 낭비되는 것이다. **그래서 마이크로소프트는 성능 개선을 위해 EnterCriticalSection이 호출되면 일정 횟수 동안 스핀락을 사용하여리소스 획득을 시도하는 루프를 수행하도록 하였다.**

****

스레드가 대기 상태로 들어가기 전에 스핀락을 사용하여 리소스 획득 시도를 하고싶다면 위 함수를 호출하여 초기화를 진행해야한다. 매개변수에서 숫자는 몇 번을 시도할건지에 대한 정보이다.

최상의 성능을 얻기 위해서는 저 숫자를 변경해 가며 테스트 해야한다. **책에서는 프로세스의 힙을 보호하기 위해 사용하는 크리티컬 섹션의 스핀 카운트는 대략 4000이라고 한다.**

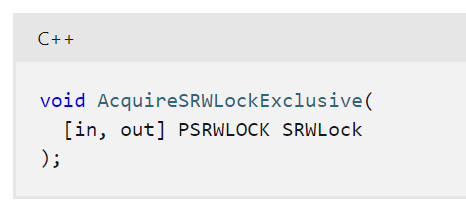
**슬림 리더-라이터 락 (slim Reader-Writer Lock)**

크리티컬 섹션과 유사하지만, SRWLock의 경우 리소스의 값을 읽기만 하는 스레드(리더)들과 그 값을 수정하려는 스레드(라이터)들이 완전히 구분되어 있을 경우에만 사용할 수 있다는 것이다.

리더들은 공유 리소스에 동시에 접근하여도 값을 바꾸는 것은 아니기 때문에 상관이 없다.

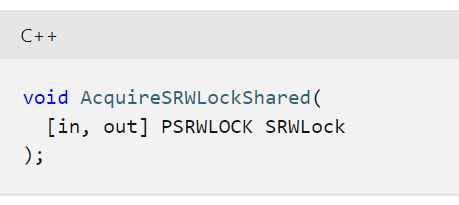
동기화는 라이터 스레드가 리소싕 내용을 수정하려고 시도하는 경우에만 필요하다.

라이터 스레드가 내용을 수정하는 동안에는 어떠한 리더, 라이터 스레드도 공유 리소스에 접근해서는 안된다.



라이터 스레드는 공유 리소스에 대한 배타적인 접근 권한을 획득하기 위하여 위 AcqureSRWLockExclusive 함수를 호출해야 한다.

그리고 볼일이 끝나면 ReleaseSRWLockExclusive 함수를 호출하면 된다.

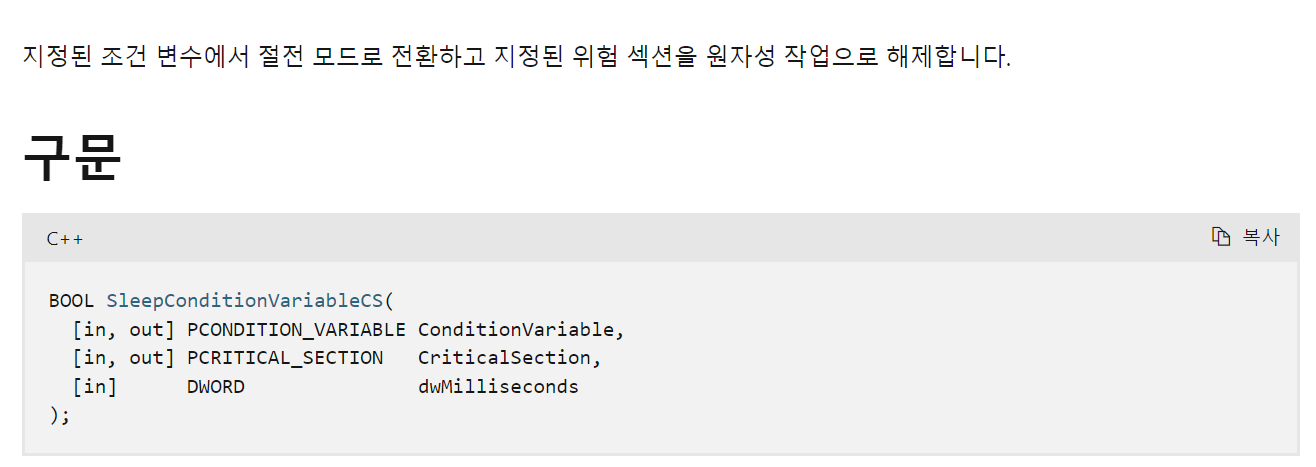


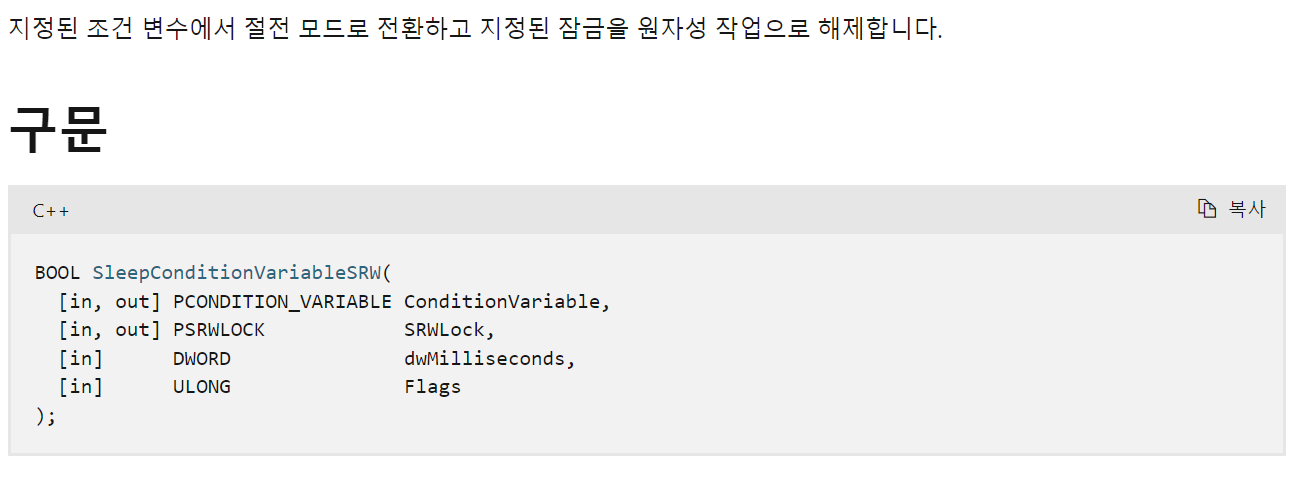
리더 스레드의 경우에도 위 함수를 호출하여 권한을 획득하면 하면 된다.

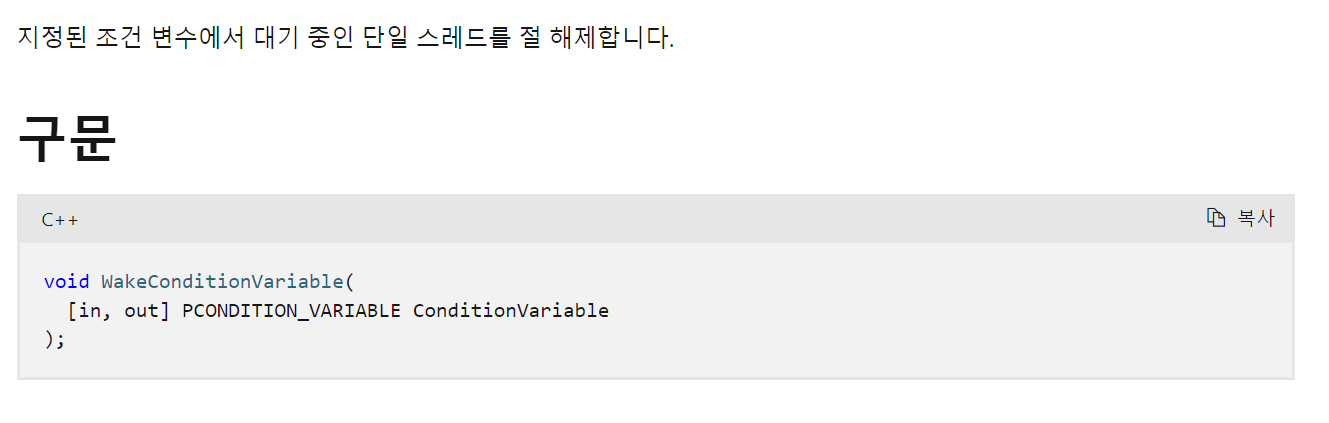
공유 리소스를 읽기만 한다면 크리티컬 섹션보다SRWLock 기법을 사용하는게 더 효율적이라고 볼 수 있다.

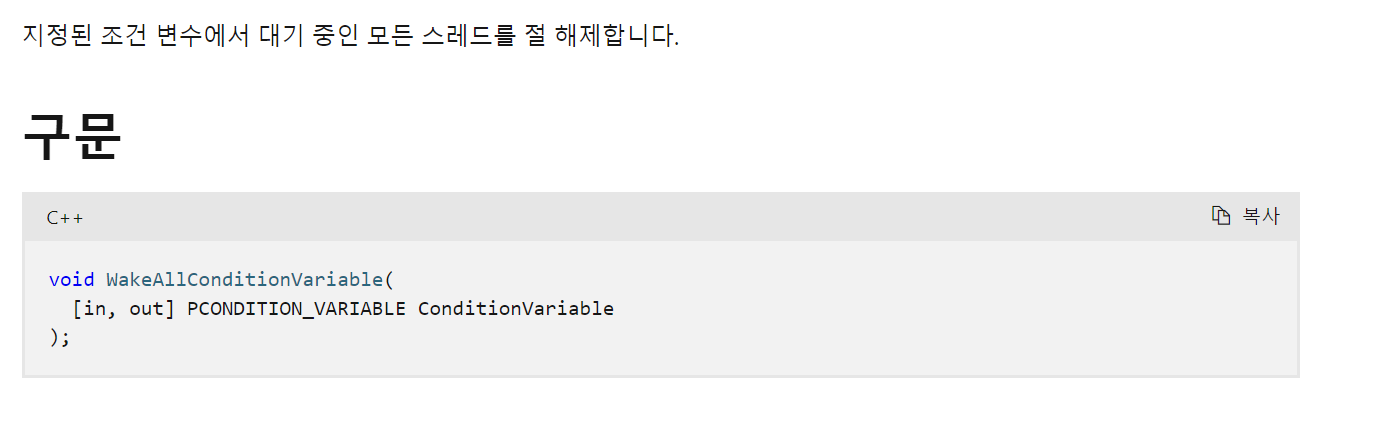
정리를 하자면, 최상의 성능으로 동작하는 애플리케이션을 작성하고 싶다면 가장 먼저 공유 리소스를 사용하지 않도록 작성할 수 있는지 검토한다. 그게 안된다면 인터락 계열 함수들, SRWLock 그리고 크리티컬 섹션 순으로 검토해야 한다. 이러한 방식들 모두 상황에 적합하지 않으면 커널 오브젝트를 사용하는 것이 좋다.

**조건변수**

조건변수(condition variable)를 사용하면 스레드가 리소스에 대한 락을 해제하고 SleepConditionVariableCS 함수나 SleepConditionVariableSRW 함수에서 지정한 상태가 될 때까지 스레드를 블로킹해 준다. 또한 이러한 동작이 원자적으로 수행되도록 설계되어 있다.



매개변수 Flags는 조건변수가 시그널 상태가 되었을 때 어떻게 락을 수행할지를 알려주는 값을 전달하면 된다. Flags로 0을 전달하면 라이터 스레드를 위한 배타적인 락을 설정한다. 만약 CONDITION\_VARIABLE\_LOCKMODE\_SHARED를 전달하면 리더 스레드를 위한 공유 가능 락을 설정한다. 위 함수는 성공하면 TRUE를 반환하지만 FALSE를 반환하면 락을 수행하지 않으며, 크리티컬 섹션을 획득하지 못한다.



블로킹 되어있다가 적절한 상황이 되어 블로킹된 스레드를 깨워야 할 필요가 있다면 위 두 함수를 호출하면 된다.

WakeAllConditionVariable 함수를 호출하면 동일한 조건변수를 인자로 SleepConditionVariable 함수를 호출한 모든 스레드가 수행을 재개하게 된다.

진행 상황 유튜브 링크:

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | | |
| **해결방안** |  | | |
| **다음주차** | 6주차 | **다음기간** | 7. 29~ 8. 4 |
| **다음주 할일** | 6주차에 계획되어있는 내용들 | | |
| **지도 교수**  **Comment** |  | | |