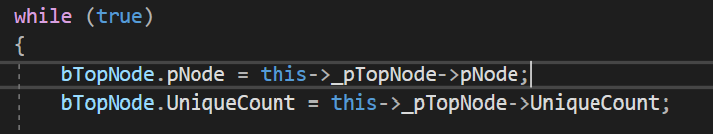
**LockFere - FreeList(MemoryPool)**

**1. ABA문제 해결 - UniqueCount적용**

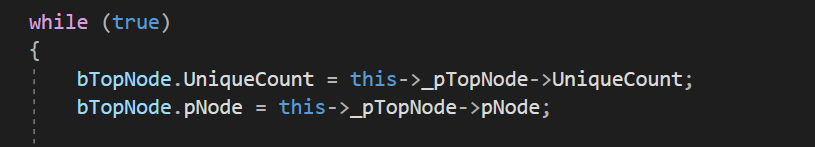
자료구조의 변화를 UC카운트로 DCAS하여 판별한다.

**2. ABA 문제를 체크하지못하는 경우**

****

위 경우 노드를 백업하고나서 자료구조자체에 변화가 생겼다면(UC가 바뀌었다면),  
변화된 UC가 비교노드로 들어가므로 ABA문제를 판별하지 못한다.

따라서 다음과같이 로직을 바꾼다.

****

이 경우 도중에 바뀌었다 하더라도 UC가 맞지않으므로 반복문이 재시도된다. (ABA판별가능)

**3. 메모리풀 AllocSize / UseSize 비교**

**메모리풀은 AllocSize와 UseSize를 비교하여 노드의 Alloc여부 결정한다.**

선/후 기준은 AllocSize와 UseSize를 비교하는 구문을 말한다.

**메모리풀 Alloc -> UseSize++ (선증가) malloc이후에 AllocSize++**

**메모리풀 Free -> 반납후에 UseSize--;**

노드를 할당해야한다고 판단하고 malloc중인데, 만약 그사이 누군가 Free되어 노드가 생겼다면?

**메모리를 약간 더 사용하겠지만 로직상 문제없음.**

하지만 노드가 있다고 판단하고 노드를 빼내러갔는데 노드가없다면?

**노드가 없는데 freeList에 pop을 요청한 꼴이됨.**

**(로직상에서는 TopNode->next접근으로 인한 nullptr 메모리참조 오류가 난다)**

**4. FreeList 에서 UseSize Inc/Dec에 대한 문제**

선/후 기준은 CAS를 말한다.

스택 push -> push이후에 UseCount++; 후증가

스택 pop -> pop이전에 UseCount--; 선감소

**자료구조 입장에서도 push가 완전히 끝나야 자료가 들어왔다고 볼수있고,**

**pop을 요청한다면 이미 나간것으로 보는 것이 맞다.**

**만약 (PUSH를 후증가가 아닌)선증가 한다면 아직 PUSH중인데 UseSize가 1이므로, POP 로직이 진행 가능해지고 에러.**

**POP을 선감소가 아닌 후감소 한다면 UseSize가 1인데 Pop이 여러 번 호출될수있으므로 에러.**

**6. Empty()문제**

**6-1. Stack - 내부 Empty()**

FreeList(또는 메모리풀) 는 애초에 해제하지않기 때문에 Empty()가 필요없음.

필요하면(없다면) 만들고, 있다면 가지고있는 노드를 뱉어주면 된다.

스택역시 프리리스트와 같이 후증가, 선차감 방식을 사용한다. 이유도 같다.

Empty()의 경우는

**1. UseSize == 0,**

**2. pTopNode->pNode == nullptr;**

**3. 1,2둘다 확인하기**

**와 같은 방법으로 체크할수있겠다.**

**하지만 어떠한 방법으로 체크하든, pop내부에서 IsEmpty에 의존하여 pop여부를 결정하는 것은 불가능하다.**

(멀티스레드 환경이므로 단한줄만 내려가도 보장이 되지않기 때문)

**1. Empty()구문을 지나고 다른쪽에서 PUSH가 된다면, pop이 가능한데 Empty == true.**

바깥에서 재시도하면 되므로 큰 문제는 아님.

**2. 데이터가 한개이고 pop이 두번시도되었는데 동시에 Empty()구문을 지났다면?**

**한쪽은 nullptr메모리 참조오류 (pTopNode가 nullptr을 가리키므로 pNext에 접근할 때 에러.)**

**따라서 값이 보장되는 InterlockedDecrement함수의 리턴값을 사용한다.**

아래구문은 Stack의 pop내부이다.

int lUseSize = InterlockedDecrement64((LONG64\*)&\_UseSize);

if (lUseSize < 0)

{

int lCurSize = InterlockedIncrement64((LONG64\*)&\_UseSize);

pOutData = nullptr;

return false;

}

(반환값이 0이상인경우 패스. 0으로 만든경우도 포함.)

**하지만 0일 때 마이너스로 만들고, 다른쪽에서 PUSH가 완료된다면 UseSize는 0이되어버린다.**

(데이터가 들어있는데 없다고 나오는 상황)

**따라서 마이너스로 바꾼 스레드가 본인이 직접 Increment시킨다.**

**이 경우 UseSize가 마이너스인 상황이라면 다른쪽에서 PUSH가 완료되었더라도 POP을 처리하지못하게된다.(return false) 이는 자료구조가 준비되지않은 것으로 보고, 넘어간다.**

이때 UseSize가 0일 때 pop이 여러 번 요청되어 UseSIze가 마이너스인 상황에서 PUSH가 완료된다면 최종적으로는 UseSize가 1이 되긴하겠지만 UseSize가 복구되기전까지는 pop이 실패하게된다.

따라서 마지막으로 올린 pop로직이 1이상인 경우에는 pop로직을 돌게끔 한다.

**(이부분 역시 자료구조자체로는 push가 완료되지않았지만 UseSize복구가 되지않았으므로 실패시키고 다음번 pop때 빼주어도 큰상관은 없지만 정확도를 올리기 위함.)**

**6-2. Stack - 외부에서 호출하는 Empty()**

**1. UseSize == 0,**

**2. pTopNode->pNode == nullptr;**

**3. 1,2둘다 확인하기**

**EmptyFlag를 두고 내부에서 UseSize를 Inc/Dec할 때 변화시켜주면 어떤가?**

**PUSH - CAS가 끝나고 UseSize를 1로 만들었다. 그리고 Empty를 false체크하려고하는데, pop이 완료되어버리고 이후 Empty false 체크된다면?**

데이터가 없는데 Empty는 false가 되어버린다.

반대의 경우도, POP - CAS가 끝나고 UseSize를 0으로 만들고 Empty를 true라고 하려고했는데 다른쪽에서 push가 완료되었다. 그리고 다시 Empty true를 만들면 자료가 있는데 true라고나옴.

Empty()체크는 사실 pop의 가능유무를 묻는것과 같다.

하지만 멀티스레드 환경에서 완벽하게 이를 동기화할 방법은 없다.

따라서 1,2를 확인하여 최대한 정확도를 높이는 형태로 간다.

어떨 때 동기화가 맞지않을까?

UseSize가 마이너스인데 복구중이라 push는 가능한경우.

추가!

int lUseSize = InterlockedDecrement64((LONG64\*)&\_UseSize);

if (lUseSize < 0)

{

int lCurSize = InterlockedIncrement64((LONG64\*)&\_UseSize);

if (lCurSize <= 0)

{

pOutData = nullptr;

return false;

}

}

**Interlocked로 반환되는값도 최적화가 될때가있다!!!!!!!!!!!!!**

FreeList는 애초에 사이즈를 --하지않기때문에 문제 X

Stack에서 만약 사이즈가 없는데 pop하러 들어왔다?

바로 nullptr접근에러.

**7. Stack - PUSH에서는 ABA문제가 없는가?**

락프리 스택에서 pop의 경우 ABA문제가 문제가 되었음. (자료구조 깨짐)

하지만 push의 경우 단순히 나를 추가하는것이므로 TopNode가 뭘로 바뀌었던 문제되지않는다.

A(1) -> B(2) -> C(3)

상태에서, A를 백업했는데

A(4) -> Z(5)

위와같이 바뀌었다고 치자.

내부 자료는 바뀌었지만 문제되지않음. 새로 추가된 노드는 A를 가리키게 되기떄문.

**락프리큐 Enqueue(ABA문제)**

Enqueue는 Stack에서 push와 입장이 비슷하다.

[스레드1] [A->B] 에서 B를 백업. (Alloc::C 추가예정)

[스레드2] 자료구조 변경 [B]

[스레드3] Enqueue를 진행하기위해 A가 다시alloc되고, pNextNode = nullptr셋팅

[스레드1] 백업한 B뒤에 새노드 C를 CAS. B->C

[스레드3] ->next가 nullptr이 아니므로 실패

**어디서 CAS가 됐든 ptail->pNext은 nullptr로 세팅되므로 다른쪽 Enqueue는 실패한다.**

**따라서 UC를 넣지않더라도 상관없음**

**tail을 미는게 실패하는 경우**

Enqueue시 무조건 next가 nullptr인것을 확인하고 연결.

따라서 tail을 밀지 못하면 절대로 enqueue되지못함.

그런데도 실패하는 경우가 생긴다.

[스레드1] A->(B) A백업, B할당하려고함

**<다른곳에서 자료구조 변화>**

[스레드2] C->D A를 Alloc.

A를 Enqueue하기위해 A->pNext == nullptr.

[스레드1] 백업한 A의 pNext가 nullptr이므로 B - Enqueue성공

**[스레드1] C->D / A->B** <--------------(ㄱ)

**[스레드1] tail을 밀려고했는데 실패!**

해결방법

모든 Enqueue/Dequeue 시 행위에는 백업한 tail노드 뒤가 nullptr이 아니라면 무조건 밀고간다.

**기타**

push할 때 T자체를 던짐.

포인터 형으로 받아서 참조 -> 복사하면 카피를 한번 줄이는거아님?

사용자가 달라는대로 줘야함.

매우 긴시간의 삽질로 얻은 멀티스레딩 디버깅 팁…

최대한 단서를 모아 문제범위를 좁혀가는 식.

문제가 여러 개인경우 오히려 꼬일수있음

VS나 언어자체에 결함을 의심할 정도로 완벽하게 코드를 짰다고 생각해도,

막상 한줄씩 돌리다보면 내코드에 허점이 있을수있음

서버-클라 구조간 프로토콜을 짜듯이,메모리에 최대한 간단하게 숫자로 정보를 남기고

문제가 생겼을 때 프로세스를 다운하기직전에 파일로 메모리로그를 남기는방식으로 디버깅

이 역시 로그를 정확하게 남기려면 행위자체와 로그를 묶어서 넣어야한다.

로직은 실행됐는데 로그가 안남을 수도있음.

100%로그도 믿기 어려운상황이다.