//////////////////////////////////////////////////////////////////////////////////////////////////////

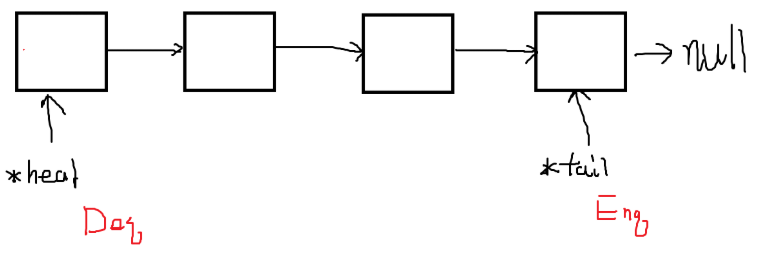
// Lock-Free Queue 정리

//////////////////////////////////////////////////////////////////////////////////////////////////////

1. LockFreeQueue 구현
2. 원리

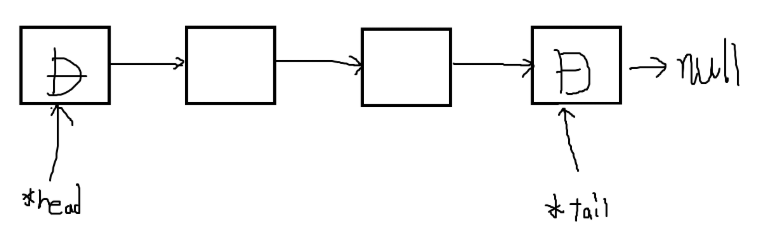
* Dequeue의 경우 head 노드를 큐에서 분리하고 head의 next를 새로운 head로 설정하는 작업인데 이건 스택의 Pop과 똑같음.
* Enqueue의 경우 2가지를 해야 함. 기존 tail 멤버가 가리키는 노드의 nextNode 변수를 내가 생성한 newNode로 바꾸고 tail 멤버 변수를 newNode로 바꾸어야 함.

1. Head, Tail 실제 노드 가리키는 경우



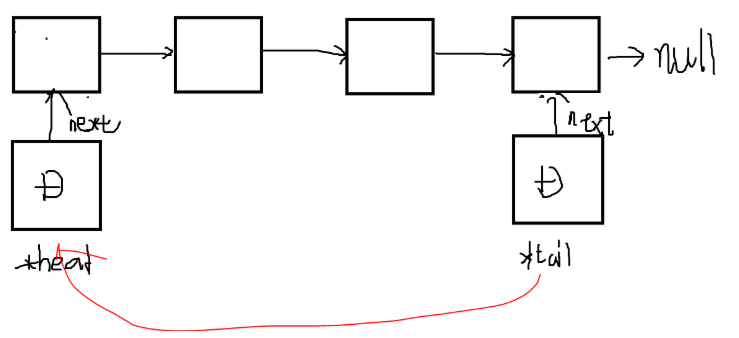
* 락프리 큐의 head, tail이라는 Node\* 타입의 멤버 변수가 있음. 그런데 이런 멤버 변수가 가리키는 노드가 실제 노드인 경우를 보면 문제가 있음.
* 큐에 아무런 노드가 없을 때 Enqueue 작업을 한다고 해보자. 이때 head, tail 멤버 변수를 내가 생성한 newNode로 원자적으로 바꿔야 함. 그런데 2개의 멤버 변수를 원자적으로 바꿀 방법이 없음.
* 그래서 실제 노드를 사용 못함.

1. Head, Tail 각각이 더미 노드를 가리키는 경우



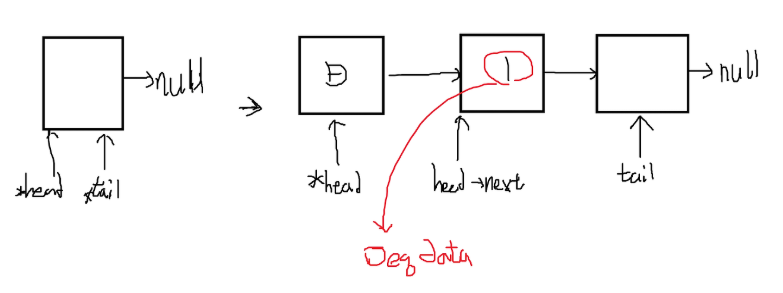
* 그러면 각각 더미 노드를 설정한다고 해보자.
* 이 구조에서 Enqueue 작업을 어떻게 할까? 이건 단방향 리스트임. 지금 Enqueue 작업을 하려면 tail의 prev 노드를 찾아야 하는데 못함.
* 그러면 더블 링크드 리스트 구조로 만들면 가능할까? 문제는 노드의 prev, next를 원자적으로 바꿔줘야 하는데 2개 변수를 원자적으로 바꿀 수 없음.
* 다른 방법으로 head의 next를 계속 참조해 나가면서 tail을 찾아서 그 이전 노드를 찾는 방식은? 지금 이 구조에서 tail이라는 멤버 변수는 어디 쓸모가 없음. enqueue를 하려면 tail이 가리키는 prev를 알아야 하는데 prev를 모르니 tail멤버 변수는 쓸모가 없음.
* 그러면 head만 가지고 enqueue, dequeue 작업을 하는데 이런것을 큐라고 부를 수 있을 까? 그리고 가능은 할까? 에 문제가 남아 있음.

1. Head, Tail은 더미 노드를 가리키고 next가 실제 노드를 가리키는 경우



* 아까는 head, tail이 더미 노드를 가리키고 head의 next는 실제 노드 tail의 next는 nullptr인 경우임.
* 이런 구조는 head, tail이 실제 노드를 가리키는 경우와 똑같은 상황임. 비어있는 상태에서 enqueue를 할 때 head->next, tail->next를 원자적으로 바꿔야 하는데 이건 안됨

1. 더미 노드 1개를 쓰는 경우

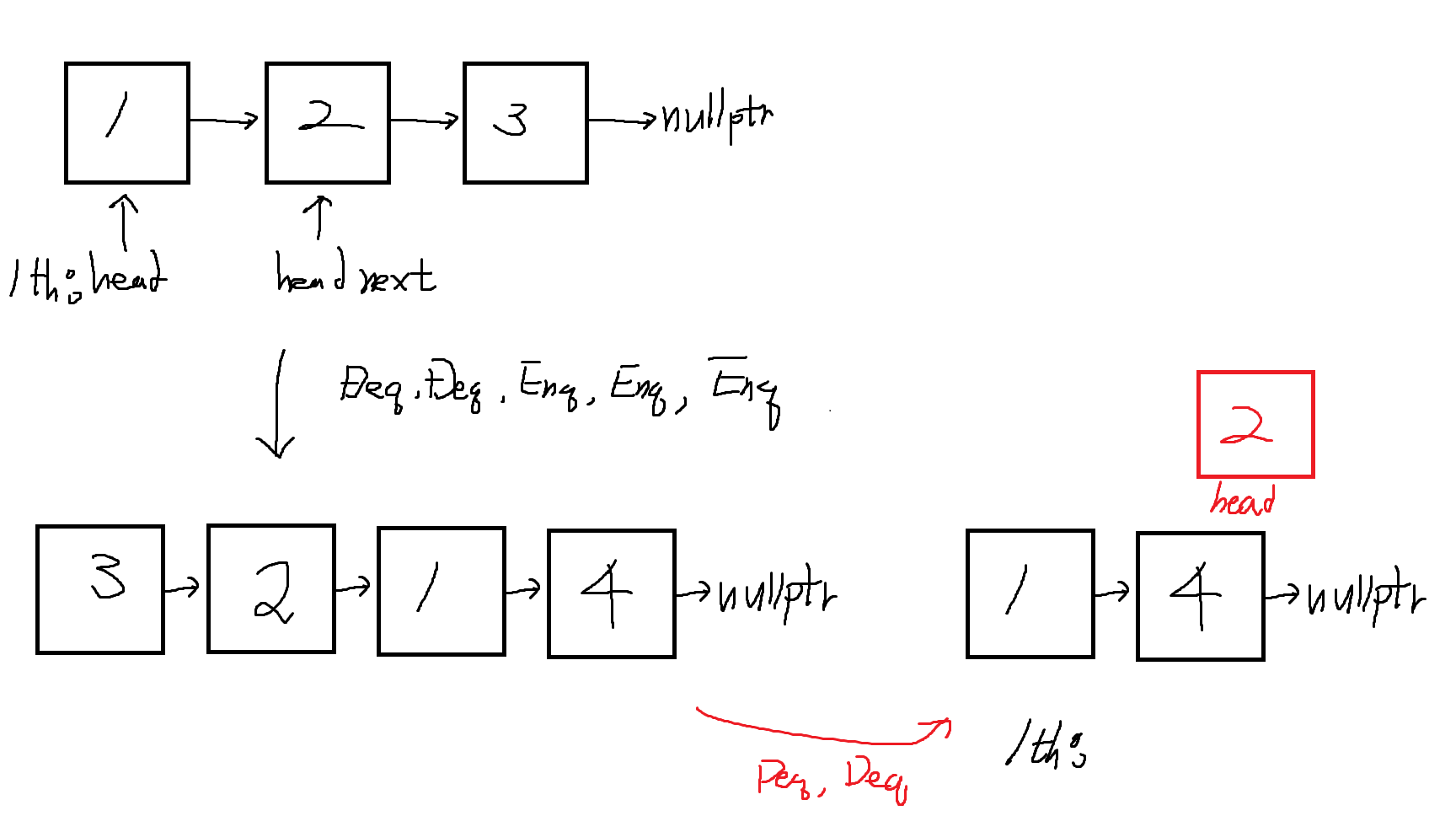


* 그럼 남은 방법은 더미를 1개 쓰는 방법임.
* 아무것도 없는 상태에서 노드를 넣을 때 2개의 변수를 원자적으로 바꿀 수 없으니 애초에 더미 노드를 1개 두고 이게 아무것도 없는 상태인 것임. 그러면 노드를 넣을 때 tail만 바꿔주면 됨.
* 마찬가지로 큐가 비어있는 경우 원래 1개에서 0으로 만들때, head, tail을 nullptr로 바꿔야 했었는데 애초에 nullptr말고 더미 노드를 가리키게 만드는 것임. 그래서 tail은 그냥 그 노드 계속 가리키게 하고 head만 바꾸는 것임.
* Dequeue 작업할 때 실제 노드는 head의 next임. Dequeue를 해서 Head의 노드는 삭제하고 head의 next가 head가 되어 더미 노드가 되면 기존 head의 next(새로운 head)가 저장한 데이터를 외부로 전달하면 됨.

1. ABA 문제에 대한 고찰
2. Enqueue에서의 문제

* Stack의 Push 똑같음.
* Tail에 Cnt를 달아서 tail이 변경된 적이 있는지 체크하는 것임. 그런데 tail이 내가 바라봤던 그대로던 나갔다 다시 들어와서 tail된 것이던 어차피 내가 enqueue하는데 문제가 있는지 생각해보면 stack의 push 처럼 enqueue하는데 문제가 없을 것임. Tail이 나갔다 들어온 놈이던 내가 바라봤던 그대로건 어차피 큐의 끝인데 내가 enqueue를 하는데 문제는 없을 것임.
* 그래서 Enqueue할 때 최상위 17bit에 count를 붙일 필요는 없음.
* 상식적으로 stack에서도 내가 newNode를 스택에 붙이는데 top의 next가 뒤 바뀌던 말던 뭔 상관임? 그냥 top에 대해서 newNode로 바꿀 수 있으면 끝임. 마찬가지로 락프리 큐에서도 실제 tail이 4번 노드라고 했을 때 ABA 문제 발생해서 앞에 노드가 원래 1->2->4인데 5->6->4 가 되었다고 해도 내가 enq 작업하는데 뭔 상관이 있냐는 것임. 아무런 의미가 없음.

1. Dequeue에서의 문제

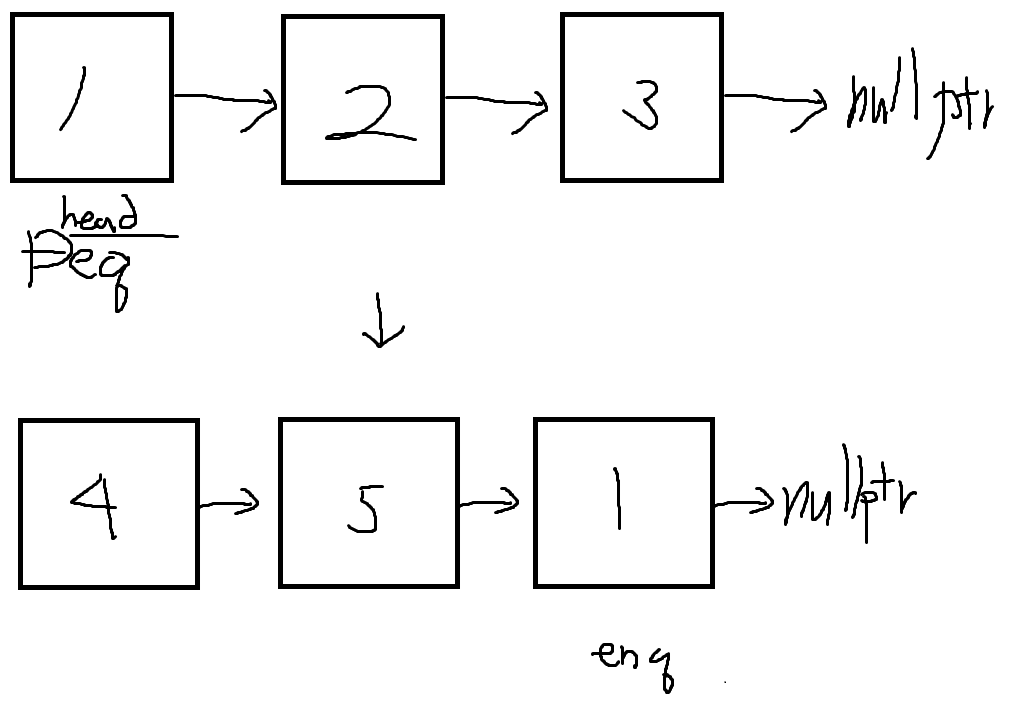


* 다만 dequeue는 pop처럼 문제가 있음.
* 1번 스레드가 1번 노드를 head로 보고 2번 노드를 head next로 보고 이걸 지역에 저장한 순간 다른 스레드가 Deq, Deq, Enq, Enq, Enq해서 큐를 위 처럼 바꿈.
* 다시 Deq, Deq를 해서 큐의 head가 1번 노드가 됨. 이때 1번 스레드가 Dequeue를 하는데 새로운 head가 될 노드는 2번인데 이건 이미 메모리 풀에 반납된 상태임. 그러니 문제가 발생함.

1. Head->next가 nullptr 문제
2. 문제 상황

* 모든 스레드가 Enqueue를 3번 하고 나서 Dequeue를 3번하는데 Dequeue 내부에서 head->next가 nullptr인 경우 중단점을 건다고 했을 때 문제가 터짐.

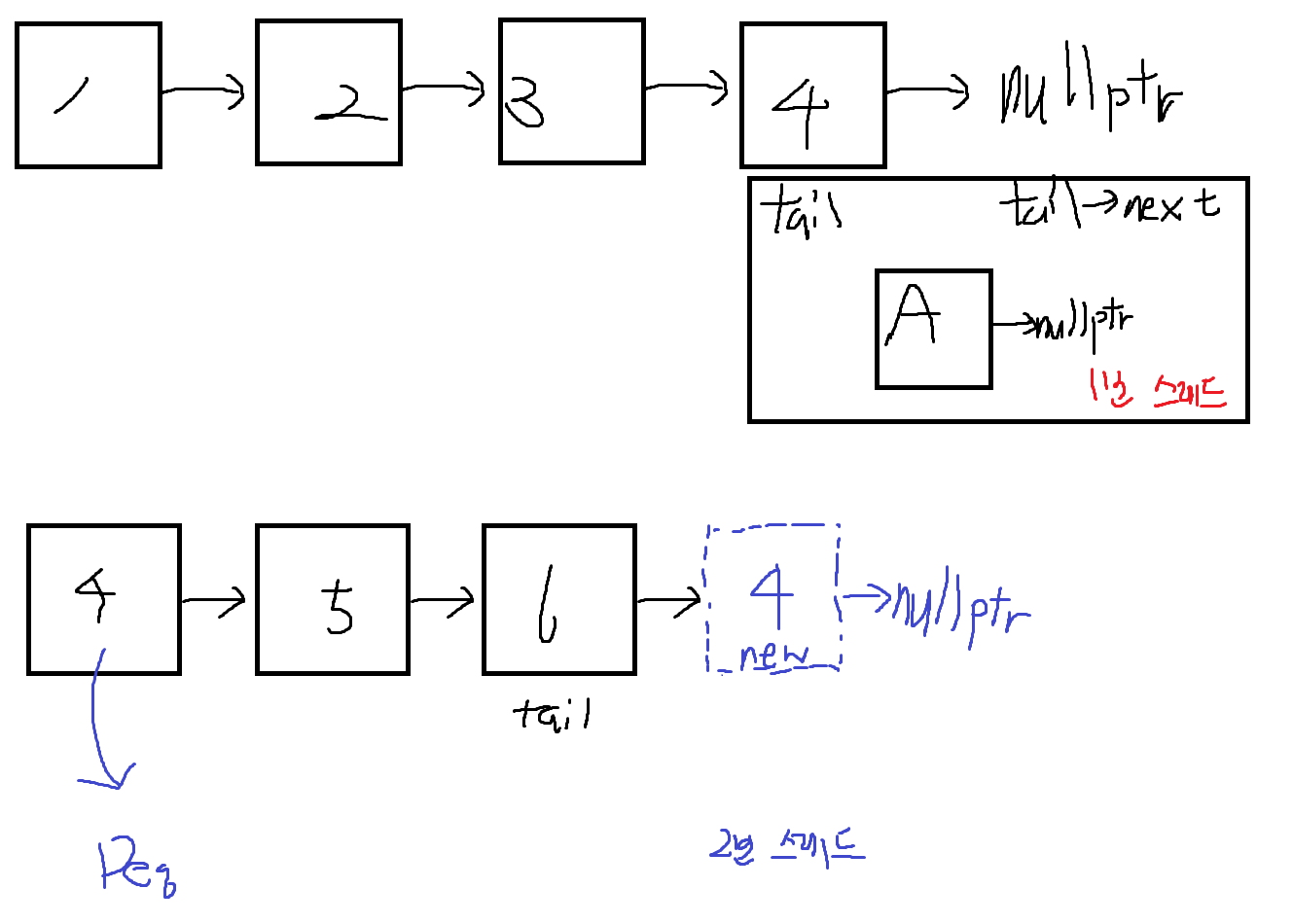
1. 문제 원인



* Dequeue할때 지역에 저장했던 head가 진짜 head인지 알 수가 없음.
* 1번 스레드가 Deq를 하려고 하는데 1번 노드를 head로 관측 했음. 그런데 다른 스레드가 deg, enq를 하면서 큐 구조를 4,5,1 노드 순으로 바꿈. 그런데 1번 노드를 큐에 enq할 때 1번 노드의 next가 nullptr이 되니 1번 스레드가 head->next == nullptr인지 체크할 때 체크가 될 수 밖에 없음. 실제로 큐에 노드가 있어도 head->next가 nullptr로 인지해버림. 이때는 그냥 continue를 해서 다시 사전작업을 하면 됨.

1. 2nd CAS 실패 문제
2. 문제 발생 원인

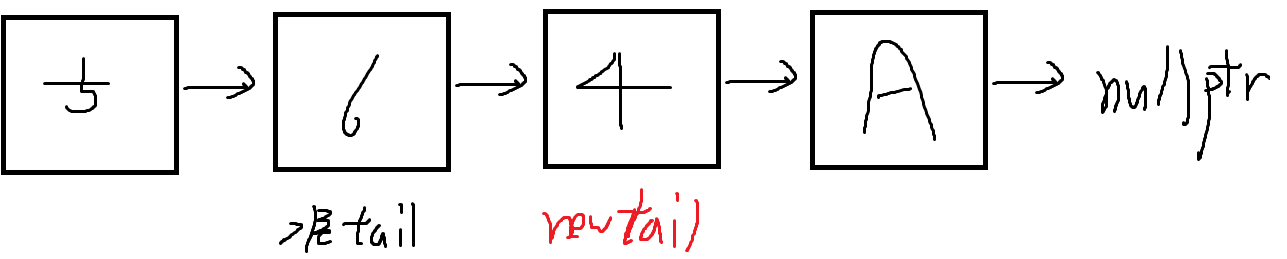
* Enqueue의 경우 2번의 과정이 있다고 했음. NewNode를 tail->next로 설정하는 것과 tail을 newNode로 변경하는 것임. 그런데 newNode를 tail의 next로 설정한다는 것 자체가 큐에 노드가 삽입이 된 상태임. 그래서 이 과정도 CAS를 통해 원자적으로 할 필요가 있음.
* 어쨌든 문제의 원인을 생각해보자. 2nd CAS를 실패했다는 것은 1st CAS를 성공한 스레드가 동시에 여러개 나왔고 2nd CAS를 할때 경합이 발생했다는 것임.
* 생각해보자. 1st CAS를 할때 CAS의 대상이 무엇일까? Tail->next임. 그런데 우리는 tail->next가 nullptr일때 newNode로 바꾸는 것인데 이것가지고 tail이 큐에서 나갔다가 다시 큐에 들어왔을 때 이를 감지할 수 있을까? 못함. 정리하면 1st CAS만으로 tail의 next가 nullptr인지 체크하는게 실제 tail의 next가 nullptr인지 큐에서 빠져나갔다가 Enqueue를 해서 1st CAS로 인해 큐에 맨 뒤에 다시 붙은 노드의 next인지 알 수가 없음.



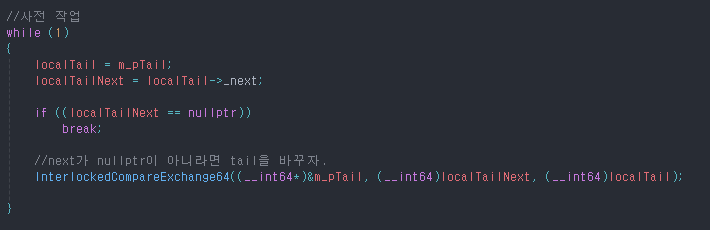
* 이게 문제임. 예를 들어 1번 스레드가 tail을 4번 노드 tail의 next를 nullptr로 봤는데 이 순간 3번 스레드가 Enqueue 작업을 해서 5, 6번 노드를 넣고 2번 스레드가 Dequeue를 해서 4번 노드를 꺼낸 것임. 그런데 Enqueue 작업을 할 때 큐에 넣으려고 메모리 풀에서 꺼낸 노드가 4번 노드가 나온 것임. 이때 2번 스레드가 그냥 1st CAS를 하면 당연히 그냥 성공할 것임.
* 이 순간 2번 스레드가 2nd CAS를 하기 직전에 이 순간 1번 스레드가 1st CAS를 하려고 하는데 1번 스레드는 tail을 4번 노드로 인지하고 있고 그것의 next가 nullptr이니 tail의 next와 nullptr을 비교하는 1st CAS를 그냥 통과해버림. 그래서 1번 스레드의 A 노드가 4번 노드 뒤로 오게 됨. 1번 스레드가 2nd CAS를 하려는데 1번 스레드가 바라봤던 tail은 4번인데 실제 큐의 tail은 6번 노드니 2nd CAS를 실패하는 것임.

1. 해결

* 1번 CAS를 해서 tail의 next를 newNode로 바꿨는데 2번 CAS인 newNode를 tail로 변경하는 작업을 실패했을 때 되돌리는건 말이 안됨. 이미 큐에 노드가 삽입된 것이라 누가 Dequeue할 수 있음. 그래서 2nd CAS를 실패해도 그냥그대로 감.
* 해결을 한다고 했을 때 방법으로 1st CAS를 2nd CAS할때 까지 못하게 하는 것은 어떻게 보면 락프리의 목적에 맞지 않음. 락없이 1st CAS를 해서 큐에 노드를 삽입할 수 있는데 못하게 막아 버리는 방식은 옳지 않음. 스핀락이랑 다를게 없음.
* 즉, 2nd CAS를 실패해도 이를 보정하는 방식으로 가야 함.



* 우선 2개의 스레드가 1st CAS를 통과해 2nd CAS에서 경쟁하는 상황을 보자. 위 상황을 그대로 사용하면 2번 스레드는 실제 tail인 6번 노드 뒤에 4번 노드를 붙이는 1st CAS 작업을 하고 1번 스레드는 본인의 newNode인 A 노드를 tail로 착각한 4번 노드 뒤에 붙임. 이때 1번 스레드가 생각한 localTail은 4번 노드고 2번 스레드가 저장한 localtail은 6번 노드이고 실제 tail은 6번 노드임.
* 이때 2nd CAS를 하면 실제로 큐 맨 뒤에 노드를 붙인 1번 스레드는 실패해버리고 2번 스레드는 성공함. 그러면 tail이 A 노드가 아니라 4번 노드가 됨.
* Tail이 실제 큐보다 밀려있음. 그래서 이를 보정하기 위해 enq, deq할때 사전 작업의 일환으로 tail을 뒤로 밀어주는 작업을 하는 것임.

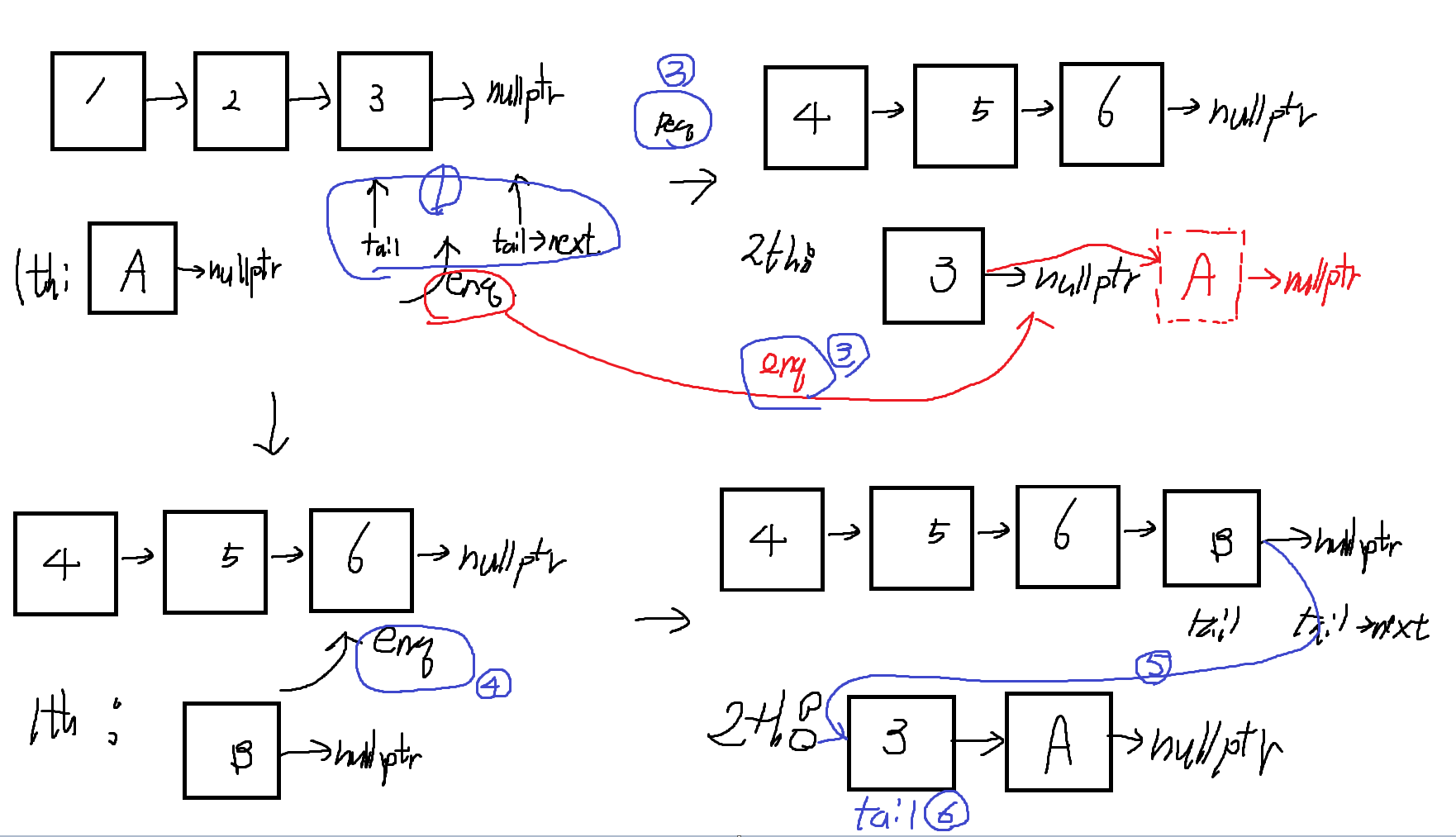


* localTail을 구하고 그것의 next를 참조해서 그게 nullptr이면 사전 작업 끝났다고 보고 탈출하는 것임.
* Nullptr이 아니면 지금 tail이 실제 tail이 아니니 tail 멤버 변수를 tail의 next로 교체해주는 작업을 하는 것임. 정확히는 실제 멤버 변수에 저장된 값이 내가 지역에 저장했던 값과 똑같아서 내가 바라봤던 tail이 같은 tail일 때 내가 바라봐서 저장했던 localTailNext로 tail을 바꾸겠다는 것임. 내가 바라봤던 값으로 교체하겠다는 의미임.
* 이때 교체 작업도 단순 interlockexchange가 아니라 CAS를 사용해서 해야 함. 만약 CAS를 안쓰고 그냥 하면 문제가 있음. 위 그림으로 나온 큐를 다시 보자. 현재 tail은 4번 노드고 1번 스레드가 enq 작업을 하려고 들어와서 사전작업을 하는데 그 때 tail이 4번 노드고 tail의 next가 A 노드임. 그런데 그 순간 다른 스레드가 Deq를 하면서 사전 작업을 하고 있는데 사전 작업을 하면서 tail을 A 노드로 변경하고 나서 Deq 작업을 해서 5, 6, 4 노드를 메모리 풀에 반납하고 Enq 작업을 해서 A 노드 뒤에 B, C, D 노드를 붙임. 그리고 A 노드를 Deq함. 그런데 이때 1번 스레드가 tail 멤버를 자기가 바라봤던 tail의 next인 A 노드를 다시 tail로 설정해버릴 수 있음. 그래서 CAS는 필수임.
* 항상 tail을 tail의 next로 설정하려고 할때 그때의 tail이 내가 바라봤던 tail인지 체크를 하고 해야 함. 만약 다른 쪽에서 tail을 밀어줬으면 다음 루프에서 tailnext가 nullptr인지 보고 탈출할 것임.
* 물론 2nd CAS를 다른 스레드들이 계속 실패를 하면서 한 스레드에서 사전작업 하는데 계속 tail의 next에 노드가 붙어 있어서 CAS 시도하고 다시 다음 루프 왔는데 다른 스레드들이 계속 Enq하면서 tail의 next에 노드를 붙여버리면 내가 사전 작업 탈출 못할 수도 있는데 이런건 무시하자. 이론적으로 무한 뺑뺑이 돌 수 있는데 극히 희박함.

1. 고찰

* Enq할때 tailCnt가 필요없다고 했음. 그러면 1st CAS할때 localTailNext == nullptr인지 하는것 대신 m\_pTail->next == nullptr이면 단순하게 tail을 제대로 볼 것이니 그러면 2nd CAS할 때도 실패할 수 없다고 생각했음.
* 예를 들어 1->2->3->nullptr 인데 1번 스레드가 3번 노드를 tail로 보고 1st CAS 직전에 멈추고 다른 스레드가 Enq, Deq를 해서 큐를 4->5->nullptr로 만든 상태에서 그 다른 스레드가 3번 노드를 enq를 1st CAS 성공해서 4->5->3->nullptr로 만들어도 1번 스레드에서 이때 1st CAS를 할때 tail->next를 비교할 텐데 이때는 tail이 3번 노드가 아니라 5번 노드이니 실패를 할 것이고 그러면 2nd CAS에서 경쟁을 안하게 되니 2nd CAS 실패 상황은 안나오겠구나 했음.
* 단순하게 1st CAS하는 순간 tail에 저장된 노드 주소값을 가져와서 nextNode가 저장된 주소값을 가지고 nullptr이랑 비교하는 것으로 생각함.
* 실제 atomic하게 연산되는 부분은 nextNode의 주소값을 가지고 해당 메모리에 저장된 값이 nullptr인제 체크해서 바꾸는 작업임. Tail이 내가 바라보던 tail인지와는 관계가 없음.
* 어셈블리상에서도 락프리 큐 주소값을 가져와서 tail 멤버 변수에 저장된 값을 가져오고 거기서 +8연산을 해서 nextNode가 저장된 메모리의 주소값을 rax에 저장한 다음 그걸 rsp+ N으로 지역에 저장하고 나서 이걸 가지고 lock cmp chg를 하고 있음.
* 그래서 1st CAS가 localTail이든 tail이든 2nd CAS 보정 작업해야 함.

1. LockFreeQueue 순서 뒤바뀜 문제
2. 문제 발생 원인



* 이런 문제가 어떻게 발생하는 지 부터 보자.
* 1번 스레드가 enq 작업을 하려는데 tail을 3번 노드 tail의 next를 nullptr로 본 상태임. 이때 3번 스레드가 enq 작업을 해서 4, 5, 6노드를 넣고 deq작업을 해서 1, 2, 3 번 노드를 메모리 풀에 반납함.
* 이 때 2번 스레드가 노드를 락프리 스택 메모리 풀에서 꺼내면 3번 노드가 꺼내질 것임.
* 2번 스레드가 메모리 풀에서 노드를 꺼내서 노드의 data를 매개인자로 받은 데이터로 설정을 하고 newNode의 next를 nullptr로 설정한 순간 1번 스레드가 1st CAS를 수행할때 본인이 tail로 봤던 3번 노드의 next가 nullptr인지 체크할텐데 당연히 성공해버림. 그러면 1번 스레드는 2번 스레드가 메모리 풀에서 뽑은 3번 노드 뒤에 본인 노드 A를 연결함.
* 그 다음 1번 스레드가 다시 enq 작업을 할때 B 노드를 들고 오고 현재 큐에 그냥 enq 작업을 아무런 방해받지 않고 해버림.
* 그 다음 2번 스레드가 enq를 이제 시작하려고 사전작업하고 1st CAS를 하면 tail의 next가 nullptr이니 그냥 통과하고 2nd CAS도 성공하면 tail이 3번 노드로 바뀜.
* 추후 deq or enq하는 함수에서 tail을 밀어줘서 tail은 정상 노드를 가리키게 될 것임.
* 그러면 1번 스레드의 입장에서 메세지를 노드에 담았다고 해보자. A 메세지를 큐에 넣고 B 메세지를 큐에 넣었다고 생각했는데 막상 Deq를 해서 보니 B 메세지가 먼저 뽑이고 A 메세지 뽑히는 메세지 순서의 뒤바뀜이 발생함.
* 이 문제의 원인이 무엇일까? 우선 이 문제가 발생한 시점을 생각해보면 Enqueue작업을 할 때 메모리 풀에서 노드를 뽑고 next를 nullptr로 설정한 순간 이 문제가 발생할 수 있어짐. 그러면 next를 nullptr 말고 다른 값으로 밀어 버리면 해결이 될까? 가능함.

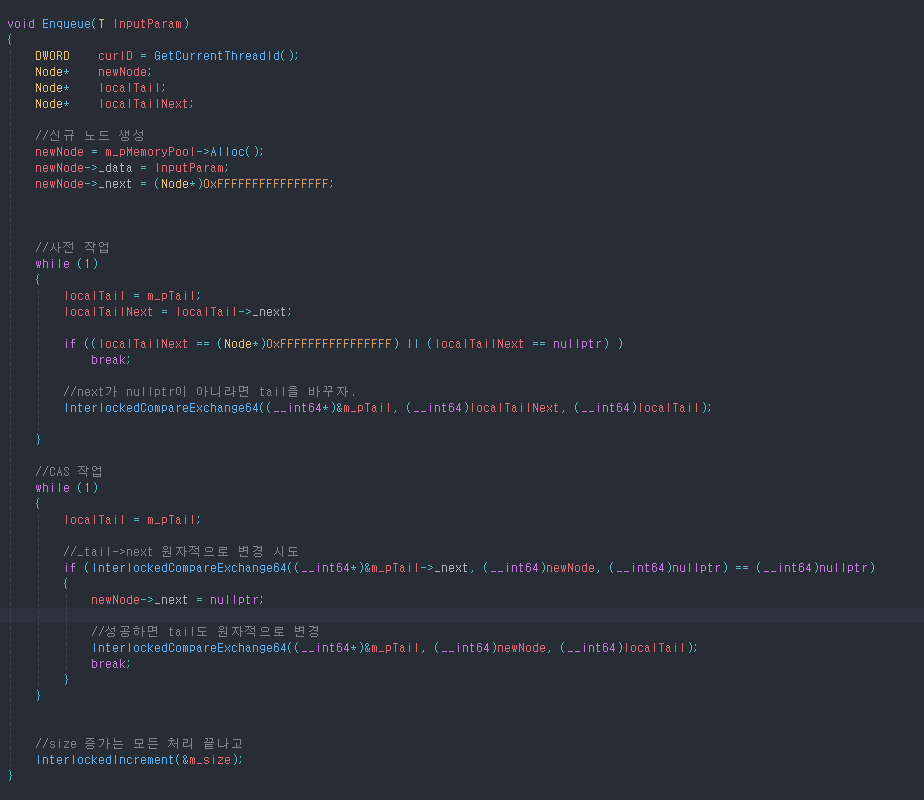
1. 문제 재현

* 재현 방법은 위에서 문제 원인 설명했던 과정 그대로 스레드를 제어해보면 됨.
* 메세지 순서가 뒤바뀌는 문제의 경우는 한 스레드에서 1, 2, 3의 값을 Enq 함수 호출해서 큐에 저장하고 Deq할 때 값을 출력하는데 1, 2, 3 순서가 뒤바뀌면 순서가 뒤바뀐 것으로 판단할 수 있음. 단, Deq하는 스레드는 1개임.
* 문제 재현 순서를 보자.
* Main 스레드에서 기본적으로 큐에 더미 노드 1개와 1, 2 노드를 Enqueue를 함.
* 1번 스레드가 Enqueue 작업을 하려고 사전 작업 통과해서 1st CAS 직전에 tail의 next만 지역에 저장하고 중지함. 이때 1번 스레드 Enqueue할 때 넣으려는 값은 1000임.
* 3번 스레드가 단독으로 Enqueue 작업을 해서 100, 101, 102 노드를 넣고 그대로 Dequeue 작업을 해서 1, 2, 3 노드를 큐에서 제거함.
* 그 다음 2번 스레드가 메모리 풀에서 Enqueue작업을 위해 3번 노드(200)를 꺼냄. 이 newNode의 next를 nullptr로 만들고 잠깐 정지함.
* 1번 스레드를 재개해서 1st CAS를 성공시켜서 3번 노드 뒤에 1000 노드를 붙임. 2nd CAS는 현재 tail이 102 노드니 실패하고 함수 끝냄.
* 1번 스레드가 다시 Enqueue작업을 위해 들어옴. 사전 작업을 통해 tail을 갱신해주고 이때 newNode는 1001임. 그냥 enqueue 작업해서 큐에 노드를 붙임. 그리고 1번 스레드는 멈춤.
* 2번 스레드를 재개해서 3번 노드를 1st CAS 통해 큐에 붙이고 2nd CAS를 통해 tail이 3번 노드로 바뀜.
* 마지막으로 3번 스레드가 전체 큐에 있는 노드를 뽑아서 출력할 것임.
* Deq 시작할 때 tail을 밀어줄 것이고 3번 스레드는 큐에서 뽑아서 값을 출력할 텐데 분명히 1번 스레드는 1000, 1001 순서대로 Enqueue를 했는데 Dequeue를 해보니 1001이 출력되고 그 다음 1000이 출력되는 현상이 보일 것임.

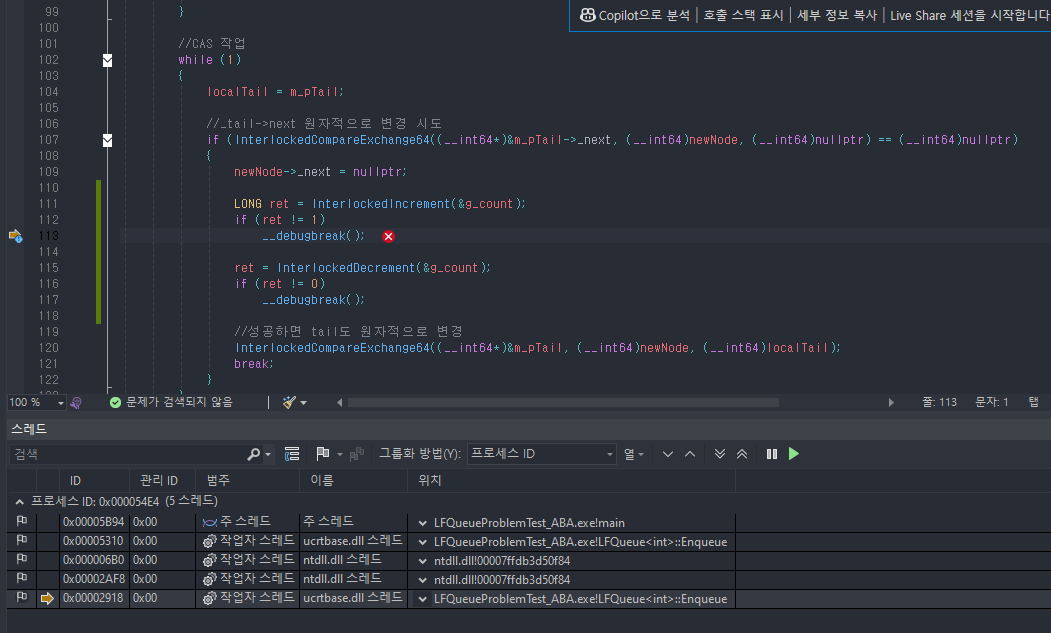
1. 고찰

* 2nd CAS 실패는 어차피 발생함. 1st CAS 작업의 atomic 성은 tail이 가리키는 노드의 nextNode값이 저장된 주소값을 가지고 atomic하게 처리하는 것임. Tail이 원자적인지는 살피지 않음.
* 1st CAS 테스트할 때 어셈블리 단위로 안하고 그냥 코드 단위로 하다 보니 tail을 가져오는 것조차 원자적으로 처리되버린 형태가 되니 메세지 순서가 안바뀌게 되었고 그걸보고 localTail이 아니라 tail이 되었으니 문제가 안생기겠구나 판단해버림.
* CAS할때 내부적으로 스택 지역에 nextNode가 저장된 주소값을 저장하는 순간 멈추고 나서 똑같이 재현하면 똑같이 순서 뒤바뀜이 발생함.

1. 문제 해결

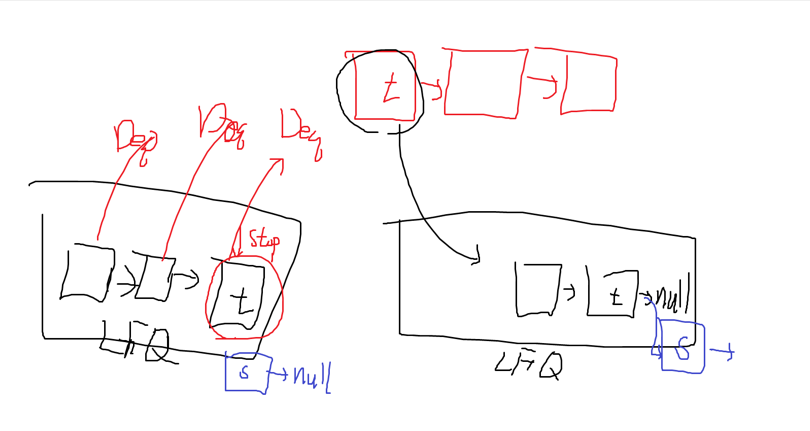


* 1번 스레드가 1st CAS를 했을 때 노드의 nextNode값이 저장된 주소값을 가지고 특정 값과 비교해서 nextNode가 저장된 메모리의 값을 내가 원하는 값으로 교체하는 작업을 하는 것인데 이게 노드가 실제 tail인지는 모르겠고 어쨌든 그때 바라봤던 노드의 nextNode값이 저장된 값을 원자적으로 nullptr이랑 비교해서 바꾸겠다는 것임.
* 그러니까 다른 스레드에서 그때 바라봤던 노드를 재활용해서 메모리 풀에서 꺼내고 나서 next를 nullptr로 바꿔버리면 1st CAS의 조건이 충족되어 버리는게 문제임.
* 그래서 이를 해결하기 위해 1st CAS할때 비교값은 nullptr로 하되 우리가 메모리 풀에서 newNode를 꺼내서 next를 nullptr로 밀어버렸기 때문에 1st CAS 조건이 충족되어서 이 문제가 터지게 되는게 내 생각임. 그래서 메모리 풀에서 꺼내서 초기화 할 때 nullptr이 아니라 0xFFFF FFFF 로 밀어버리고 1st CAS 조건은 그대로 nullptr로 하는 것임.
* 대신에 이대로 가면 1st CAS 자체를 못하니 우선 사전 작업 할때 0xFFFF FFFF도 탈출 조건으로 걸어주고 1st CAS 성공해주면 큐에 붙인 newNode의 next를 nullptr로 바꿔주는 것임. 그 순간 다른 스레드는 1st CAS를 성공할 수 있는 조건을 달성하게 됨.



* 이 방식은 2nd CAS를 할 때까지 다른 스레드가 1st CAS를 못하게 막는 방식이 아님. 실제로 1st CAS 후에 전역 변수를 Inc해서 1인지 체크하고 아니면 중단점 걸고 바로 다시 dec해서 0인지 체크해서 아니면 중단점 거는 로직을 했을 때 2개의 스레드가 1st CAS 통과해서 중단점 걸리는 것을 볼 수 있음.
* 이런 방식에서 문제가 터질 수가 있다고 하면 newNode를 메모리 풀에서 뽑았을 때 애초에 next가 nullptr이라면 당연히 순서 뒤바뀔 수 있는 조건일 것임. 그런데 생각해보면 애초에 newNode를 뽑았는데 next가 nullptr이라는 말이 뭘까? 이 노드는 내부 메모리 풀에서 다른 사람이 써서 반납한 노드가 아니라 새롭게 페이지 할당해서 밀어버린 페이지에서 뽑은 싱싱한 노드인 것임. 즉, newNode는 절대 다른 스레드가 썼었던 노드가 아니라는 말임. 그러니까 newNode를 뽑아서 nextNode를 FFFF FFFF로 밀기 전에 이미 nullptr이어도 상관이 없음.

1. LockFreeQueue 공용 노드 풀 문제
2. 문제 발생 원인



* 1번 스레드가 t라는 노드를 tail로 바라보고 enqueue 작업을 위해 s라는 노드를 메모리 풀에서 꺼냈음.
* 이때 다른 스레드가 deq를 해서 t 노드를 공용 노드 풀에 넣고 다른 스레드가 다른 락프리 큐에 enq해서 넣어버렸을 때 1번 스레드다 1st CAS를 할 때 t의 next가 nullptr이니 착각해서 s 노드를 다른 락프리 큐에 넣는 다는 것임.

1. 고찰

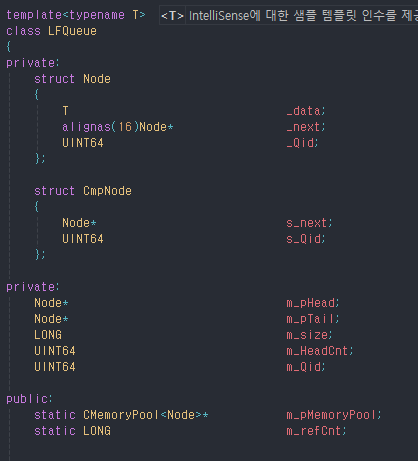
* 그러나 순서 뒤바뀜 문제에 언급했다시피 이 문제도 가능함. 1st CAS는 nextNode에한 원자적 비교와 교환을 해주는거지 이게 1번 큐에 있던 노드이건 2번 큐에 있던 노드이건 상관이 없다는 것임.

1. 문제 재현

* 락프리큐를 2개 만들어서 1번 스레드는 1번 락프리 큐에 enq를 시도하는데 이때 1st CAS 직전에 멈춤.
* 2번 스레드가 1번 락프리 큐에 대해서 enq를 3번 하고 deq를 3번 해서 1번 스레드가 tail로 봤던 노드를 공용 노드 풀에 반납함.
* 같은 2번 스레드가 이제 2번 락프리 큐에 대해서 enq 작업을 할때 반납했던 tail 노드를 뽑음. 그걸 2번 락프리 큐에 enq를 해버림.
* 그 순간 1번 스레드가 1st CAS를 시도해서 통과해버리고 enq 작업이 끝남.
* 3번 스레드가 1번 락프리 큐에 있는 데이터 출력하고 2번 락프리 큐에 있는 데이터 뽑아서 출력을 했을 때 1번 스레드는 분면 1번 락프리 큐에 10이라는 값을 enq했는데 3번 스레드가 데이터를 출력해보니 10이 1번 락프리 큐에 대해서 출력할 때 나오는게 아니라 2번 락프리 큐에 대해서 출력할 때 나오는 것을 보여주면 됨.
* 1번 큐에 저장되는 데이터 값은 0 ~ 999 대역이고 2번 큐는 1000 이상임. 그런데 2번 큐에서 데이터를 뽑는데 1번 큐 대역이 나오면 이는 버그가 맞음.

1. 문제 해결

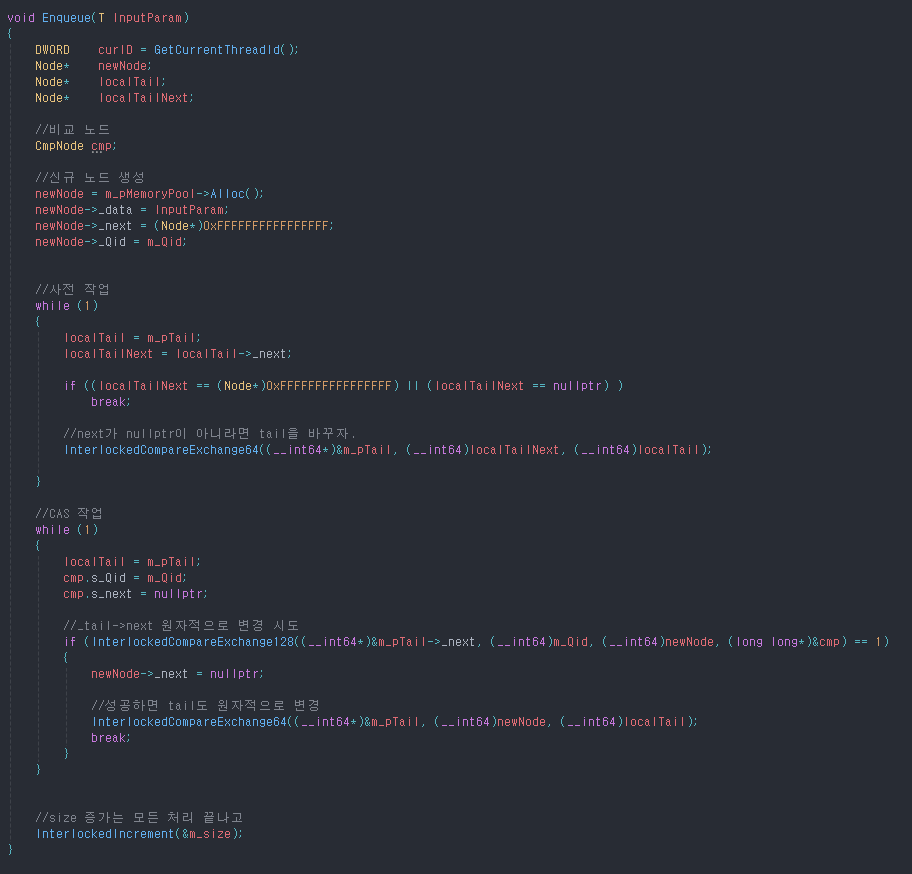
* 그러면 1st CAS를 할때 단순히 nextNode가 nullptr이란건만 비교하기 보다는 이게 추가적인 tag를 달아서 nextNode랑 같이 비교하는 것임. TailNext가 nullptr인지만 보는게 아니라 내 큐것인지 다른 큐 것인지 노드 구조체에 Qid 멤버 변수를 추가로 달아서 1st CAS를 interlockedCompareExchange128로 처리하는 것임.



* 우선 노드 구조체가 바뀌어야 함. next변수 뒤에 큐 마다 id를 들고 있고 enq할 때 노드에 id를 삽입해서 1stCAS를 할 때 단순히 next가 nullptr인지 보다 거기에 추가해서 enq 호출한 락프리 큐의 멤버로 설정된 qid랑 1stCAS하려고 바라본 노드의 id까지 확인해서 같음이 확인되어 그 노드가 내 락프리큐의 노드임이 확인되면 1st CAS 성공시켜서 enq 작업하도록 하는 것임.
* 그래서 노드마다 큐의 id를 삽입할 변수가 필요하니 그게 Qid이고 CAS128를 해야 하니 next를 16바이트 경계에 맞춤.
* 그리고 CAS128할때 매개인자로 넣을 비교인자에 대한 구조체가 CmpNode임.
* M\_Qid는 말했듯이 큐 마다 부여되는 id임.
* RefCnt는 락프리 큐가 생성될 때마다 count를 증가시키고 락프리 큐가 소멸할 때 마다 감소 시켜서 0이 되면 메모리 풀까지 삭제하기 위한 참조 카운트임.



* 생성자에서 참조 카운트를 올려서 0에서 1로 올린 단 하나의 스레드가 메모리 풀을 생성하도록 하고 다른 스레드도 동시에 락프리 큐를 생성할 때 메모리 풀이 생성이 안되었는데 아래 로직을 실행하면 안되서 메모리 풀 변수에 객체 주소가 들어선게 확인되면 루프 탈출해서 로직 처리하게 함.
* 큐의 id는 해당 큐의 멤버 변수의 주소값을 사용함. 주소값을 고유하기 때문에 이를 사용했음.



* Enqueue 로직임. 노드를 메모리 풀에서 가져왔으면 Qid를 본인 멤버 변수인 qid로 설정해주고 CAS 작업할때 비교작업할 구조체인 cmp의 멤버를 초기화 해주고 CAS 128을 통해 내가 바라본 노드의 nextNode가 저장된 메모리 부터 16바이트를 cmp랑 비교해서 같으면 바라봤던 노드의 멤버를 m\_qid, newNode로 변경함.