



6. 병렬 알고리즘 - QUEUE

멀티쓰레드 프로그래밍
정내훈

내용

- 풀
- 큐
- ABA

풀(Pool)

- 리스트는 Set객체
- Queue와 Stack은 Pool 객체
- Pool객체
 - 같은 아이템의 복수 존재를 허용
 - Contains()메소드를 항상 제공하지 않는다.
 - Get()과 Set()메소드를 제공한다.
 - 보통 생산자-소비자 문제의 버퍼로 사용된다.

풀(PPOOL)

- 풀의 종류

- 길이제한

- 있다 : 제한 큐
 - 구현하기 쉽다.
 - 생산자와 소비자의 간격을 제한한다.

- 없다 : 무제한 큐

- 메소드의 성질

- 완전 (total) : 특정 조건을 기다릴 필요가 없을 때
 - 비어있는 풀에서 get() 할 때 실패 코드를 반환
 - 부분적(partial) : 특정 조건의 만족을 기다릴 때
 - 비어있는 풀에서 get() 할 때 다른 누군가가 Set() 할 때 까지 기다림
 - 동기적(synchronous)
 - 다른 스레드의 메소드 호출의 종첩을 필요로 할 때
 - 랑데부(rendezvous) 라고도 한다..

큐 (QUEUE)

- 정의 Queue<T>
 - 타입 T인 아이템의 순서가 있는 수열
 - Enq(x) 메소드
 - 아이템 x를 큐의 끝 (tail)에 추가한다.
 - Deq() 메소드
 - 큐의 다른쪽 끝 (head)에서 아이템을 제거해서 반환한다.

큐 (QUEUE)

- 제한 큐 및 부분 큐

- Lock-Free의 구현성격과는 맞지 않으므로 생략

- ConditionVariable이라는 메소드가 필요

- Lock을 가진 채로 Block()이 필요

- Block()시 Lock을 해제, 다시 스케줄 될 때 Lock() 재 획득

- 운영체제 호출 필요.

- 스레드 스케줄링과 연동이 필수

- 무제한 완전 큐를 구현해보자.

무제한 완전 큐

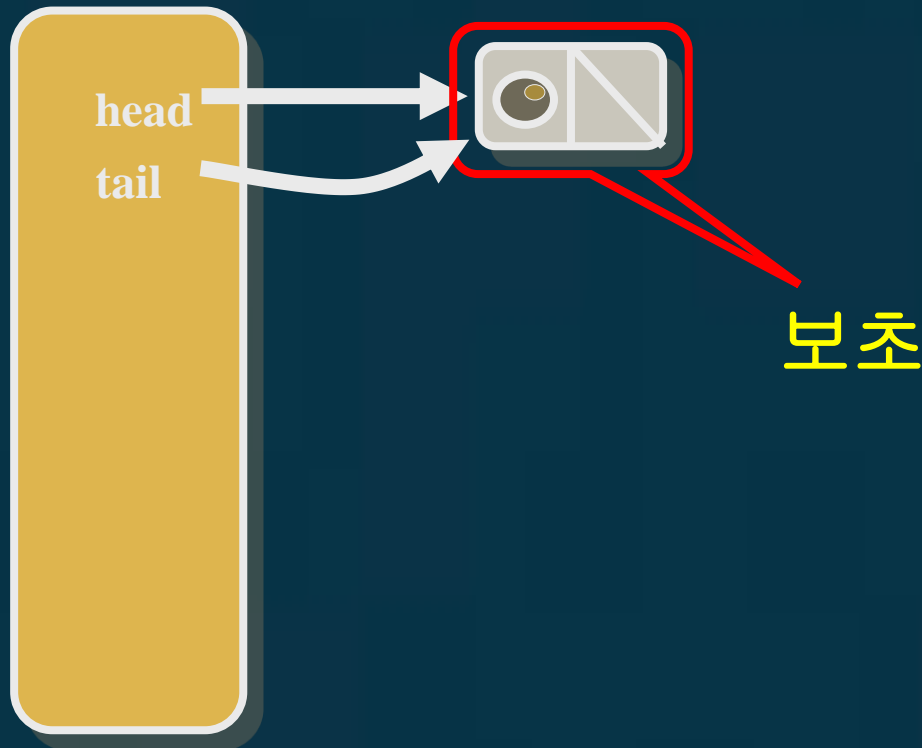
- Coarse Grain

- 모든 메소드를 Locking
 - enq용 Lock과 deq용 Lock을 따로 갖는다.
- 가장 간단한 구현
- 성능 비교의 시작
- enq(X)와 deq()메소드를 갖는다.
- Head pointer에서 deq를 하고
- Tail pointer에 enq를 한다.

무제한 완전 큐

- Coarse Grain

- 구조 : 초기 형태, 비어 있는 큐

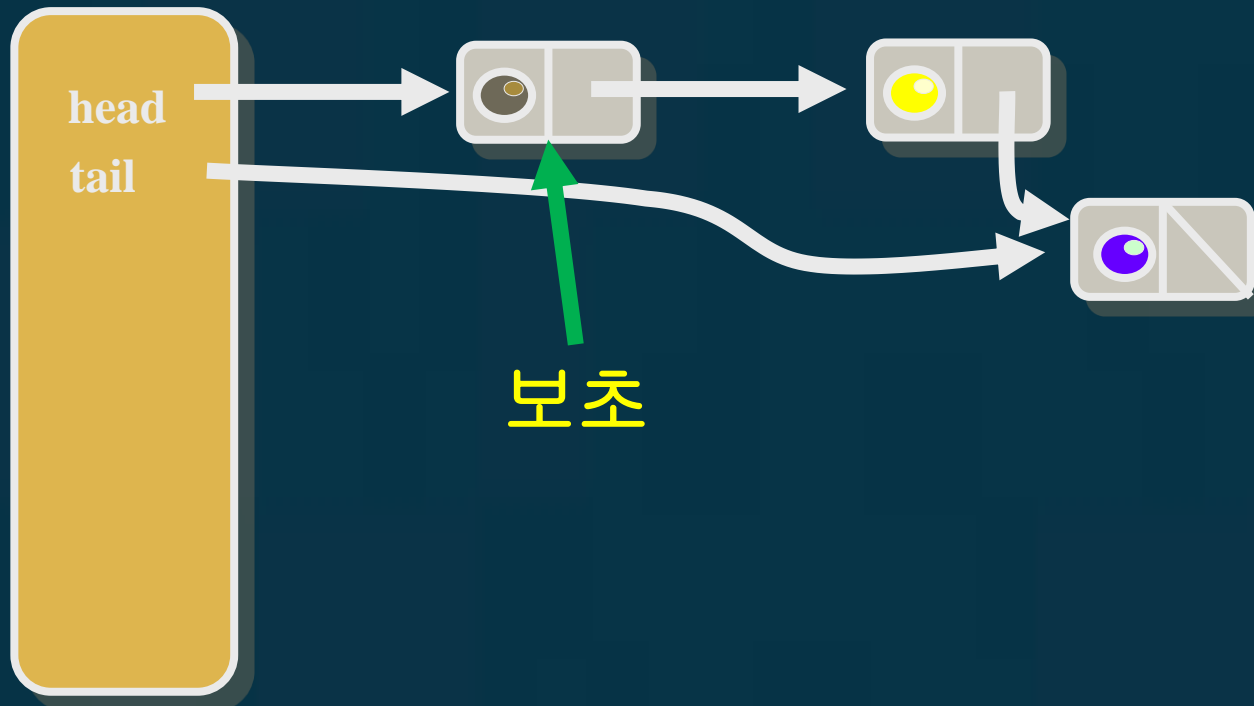


무제한 완전 큐

- Coarse Grain

- 구조 : 일반적 형태

- 노드가 2개 추가된 경우



무제한 완전 큐

● Coarse Grain

— 구현

- C++ : dequeue시 delete 필요

```
1  public void enq(T x) {  
2      enqLock.lock();  
3      try {  
4          Node e = new Node(x);  
5          tail.next = e;  
6          tail = e;  
7      } finally {  
8          enqLock.unlock();  
9      }  
10 }
```

```
11 public T deq() throws EmptyException {  
12     T result;  
13     deqLock.lock();  
14     try {  
15         if (head.next == null) {  
16             throw new EmptyException();  
17         }  
18         result = head.next.value;  
19         head = head.next;  
20     } finally {  
21         deqLock.unlock();  
22     }  
23     return result;  
24 }
```

무제한 완전 큐

- 실습 #24 : Coarse Grain 무제한 완전 큐를 구현하시오.
 - 아래 벤치마크 프로그램을 사용하시오.
 - Empty Queue에 Deq를 할 경우 -1을 리턴
 - thread 개수 1, 2, 4, 8에서의 성능을 비교하시오.
 - 각각 실행 전 Queue 를 클리어 하시오
 - 실행 후 queue에 있는 원소 20개를 출력하시오

```
void ThreadFunc(int threadNum)
{
    for (int i=0; i<100000000 / threadNum; i++) {
        if ((rand() % 2) || i < 32 / threadNum) {
            UnboundedQ.Enqueue(i);
        } else {
            UnboundedQ.Dequeue();
        }
    }
    return 0;
}
```

비멈춤 동기화

- 속제 9 :
 - 성긴 동기화 큐의 구현
 - 제출물
 - .cpp 파일
 - 쓰레드 개수 별 실행속도 비교표
 - CPU의 종류 (모델명, 코어 개수, 클럭)
 - 제출 : eclass

비멈춤 동기화

- 성능 (목금)
 - Nomutex : 814
 - 1 : 1062
 - 2 : 1260
 - 4 : 1525
 - 8 : 2080

비멈춤 동기화

- 성능 (화목)
 - Nomutex : 768
 - 1 : 1092
 - 2 : 1468
 - 4 : 1685
 - 8 : 2054

무제한 무잠금 큐

● 무잠금 (Lock Free)

- CAS를 사용
- 다른 스레드가 임의의 위치에 멈추어 있어도 진행 보장

← $while(\text{false} = \text{cas}(\text{tail} \rightarrow \text{next}, \text{null}, e, \&\text{tail}, \text{tail}))$
 |
 $\rightarrow \text{tail} \rightarrow \text{next} = e$
 $\text{tail} = e$

개
안
행

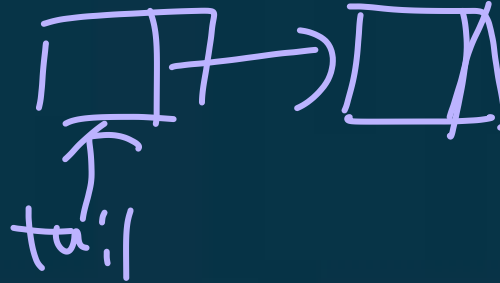
무제한 무잠금 큐

- ENQUEUE의 기본 동작
 - Tail이 가리키는 Node에 CAS로 새 노드를 추가.
 - 실패하면 재시도
 - 성공하면 Tail을 이동
- 이 아이디어를 기반으로 무잠금 구현 시작

무제한 무잠금 큐

- ENQUEUE

— 직관적인 구현



```
void enq(int x) {  
    Node *e = new Node(x);  
    while (true) {  
        if (CAS(&tail->next, NULL, &e)) {  
            tail = e;  
            return;  
        }  
    }  
}
```

무제한 무잠금 큐

- ENQUEUE

- Non-blocking이 아니다.
- CAS을 성공하고 tail을 업데이트 하지 않을 경우 모든 다른 스레드가 기다리게 된다.

- 해결책

↳ enqueue 실패 ⇒ blocking

- Tail의 전진이 모든 스레드에서 가능하게 한다.

무제한 무잠금 큐

● ENQUEUE

— 1차 수정

```
void enq(int x) {  
    Node *e = new Node(x);  
    while (true) {  
        if (CAS(&(tail->next), NULL, &e)) {  
            tail = e;  
            return;  
        }  
        if (nullptr != tail->next) tail = tail->next;  
    }  
}
```

실패하면 다시
시작

무제한 무잠금 큐

● ENQUEUE

– 1차 수정 : 문제

```
void enq(int x) {
    Node *e = new Node(x);
    while (true) {
        if (CAS(&(tail->next), NULL, &e)) {
            tail = e; // 이제는 안전하지 않다.
            return;
        }
        if (nullptr != tail->next)
            tail = tail->next; // 다른 스레드의 변경을 덮어 쓸 수 있다.
    }
}
```

이게 나쁜게 아니라

무제한 무잠금 큐

● ENQUEUE

— 해결 : CAS사용

- tail값을 last에 저장해서 비교
- next값도 저장 필요

```
void enq(int x) {
    Node *e = new Node(x);
    while (true) {
        Node *last = tail;
        Node *next = last->next;
        if (last != tail) continue;
        if (nullptr == next) {
            if (CAS(&(last->next), nullptr, e)) {
                CAS(&tail, last, e);
                return;
            }
        }
        else CAS(&tail, last, next);
    }
}
```

다중스레드가 있을 때
 → 노드 값이 변경되는가?
 변경되지 않으면 안전

무제한 무잠금 큐

● DEQUEUE : 1차 구현

- 비어 있는지 검사
- Head를 전진 시키면 deque 끝

```
int deq(int x) {
    while (true) {
        Node *first = head;
        if (first->next == nullptr) EMPTY_ERROR();
        if (!CAS(&head, first, first->next))
            continue;
        int value = first->next->item;
        delete first;
        return value;
    }
}
```

계대로된값 X

니카르스데스가다다

→ delete된것을 리턴

무제한 무잠금 큐

● DEQUEUE

```
-int value = first->next->item;
```

- 다른 스레드에서 first->next가 가리키는 노드를 꺼내서 delete시키고 어떤 일이 벌어질지 알 수 없음!!!
- value값이 queue원래 있었던 값이라는 보장이 없다.



무제한 무잠금 큐

● DEQUEUE

—Next의 사용

```

int deq(int x) {
    while (true) {
        Node *first = head;
        Node *next = first->next;
        if (first != head) continue;
        if (next == nullptr) EMPTY_ERROR();
        int value = next->item;
        if (false == CAS(&head, first, next))
            continue;
        delete first;
        return value;
    }
}

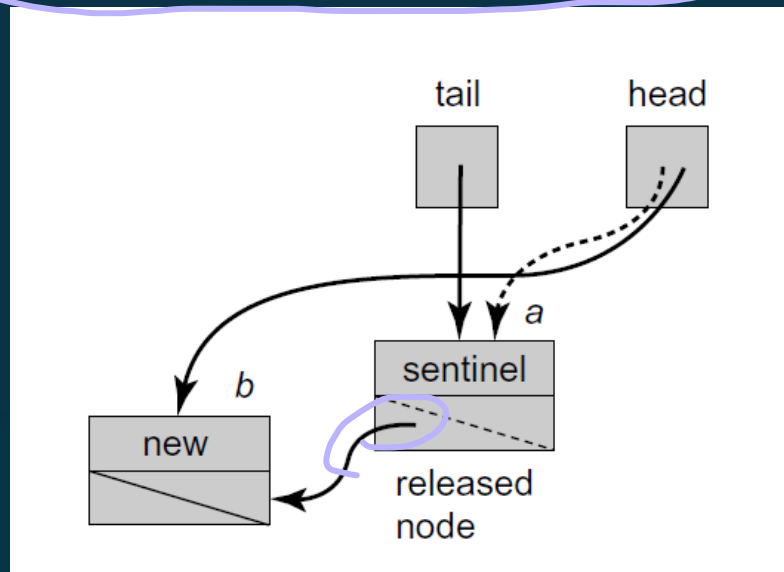
```

가운데 인덱스를
→ 동시에 다른 주원 값이아-

무제한 무잠금 큐

● DEQUEUE

- Tail과 Head가 만나는 경우?
- Tail이 enq하는 도중에 deq가 일어나서 tail이 삭제된 보초노드를 가리키는 순간이 생긴다.



tail / head가 인큐
→ 탈락된 노드에
대입, 정상화
포인터를

tail, head가 dequeue를 할 때

무제한 무잠금 큐

● DEQUEUE

– Enq에서의 tail의 전진을 보조해 준다.

```
int deq(int x) {
    while (true) {
        Node *first = head;
        Node *last = tail;
        Node *next = first->next;
        if (first != head) continue;
        if (nullptr == next) EMPTY_ERROR();
        if (first == last) {
            CAS(&tail, last, next);
            continue;
        }
        int value = next->item;
        if (false == CAS(&head, first, next)) continue;
        delete first;
        return value;
    }
}
```

→ 꼬인 경우
↳ tail 전진

무제한 무잠금 큐

- 주의

- 컴파일러 최적화 문제

```
Node * volatile tail;  
Node * volatile head;
```

무제한 무잠금 큐

- 실습 #25 : 무제한 무잠금 Lock-Free Queue를 구현하라.
 - 실습 #24의 벤치마크프로그램을 사용하여 실습 #24와 속도비교를 실시하라.

속제 10

– Lock-Free 무제한 Queue의 구현

– 제출물

- .cpp 파일
- 실행속도 비교표 (성긴동기화)
- CPU의 종류 (모델명, 코어 개수, 클럭)
- 여러 번 실행 시 thread 개수가 많을 경우 드물게 크래시나 무한루프를 경험할 수 있다.
 - 가능하면 debugging해보고 완벽히 수정되지 않더라도 제출할 것

– 제출 : eclass

비mutex 동기화

- 성능 (2021-화목)

- Nomutex : 768
- 1 : 1092
- 2 : 1468
- 4 : 1685
- 8 : 2054

- LockFree

- 1 : 925
- 2 : 1037
- 4 : 841
- 8 : 1248

비멈춤 동기화

- 성능 (목금)
 - Nomutex : 814
 - 1 : 1062
 - 2 : 1260
 - 4 : 1525
 - 8 : 2080
- LockFree
 - 1 : 936
 - 2 : 967
 - 4 : 874
 - 8 : 1240

- cache line - out of order 마지막 난관!
- data race ABA
- Compiler - ABA

● 드물게 Crash!

- Debug모드에서 실행해보자
- 벤치마크에서 초기 큐 길이를 단축해보자
- 스레드의 개수를 늘려보자.

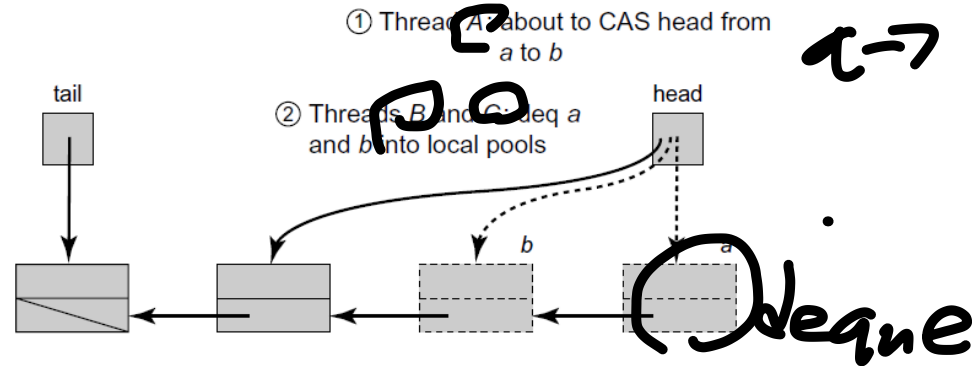
● 노드의 재사용시 생기는 문제

- new(), free()는 메모리를 재사용한다.
- CAS사용시 다른 스레드에서 그 주소를 재사용했을 가능성이 있다.

ABA

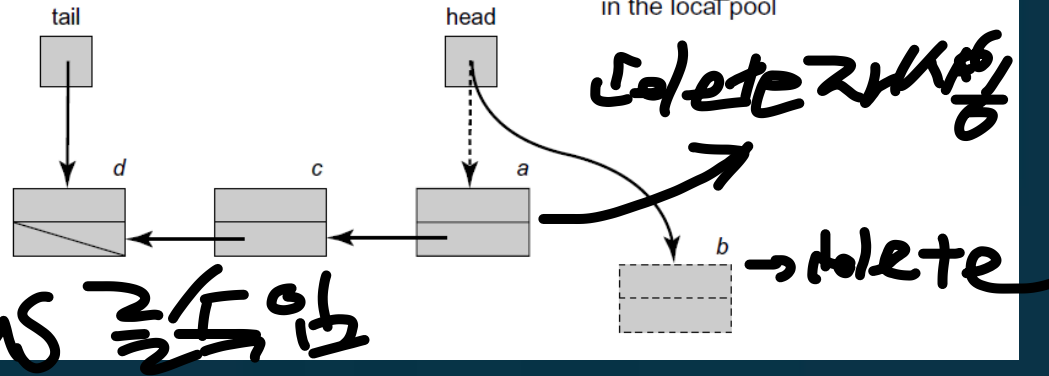
A가 읽고 B가 읽고 A가 읽

(a)



(b)

③ Threads B and C: enq a , b , and d ④ Thread A: CAS succeeds, incorrectly pointing to b which is still in the local pool



ABA 4 B 4
 v7 v8 v9 (version
 number)

● 해결책 (1/3)

– 포인터를 포인터 + 스탬프로 확장하고 포인터 값을 변경할 때마다 스탬프 값을 변경시킨다.

• 구현

- 64비트인 경우 몇 개의 비트를 스탬프에 할당.
- 32비트인 경우 복잡...

– LL, SC 명령의 사용 (ARM, Alpha, PowerPC)

- 값을 검사하는 것이 아니라 변경 여부를 검사
- LL, SC 구조는 CAS보다 우월하다.
 - 그러나. LL, SC는 Wait-free가 불가능 하다.

Load Linked, Store Conditional

$GS(m, 0, n)$

ABA

$if(m == 0) \&\& (m \text{ is not dirty}) \text{ else return } false$
 $m = n \text{ return true}$

● 해결책 (2/3) : 스마트 메모리 관리자

— Reference Counter를 사용한다.

- shared_ptr
- 'first'가 첫번째 노드를 가리키고 있는 동안에는 그 노드는 free-list에 들어갈 수 없다.
- shared_ptr에 대한 atomic_compare_exchange가 가능하다.

— Java는 garbage collection을 사용하므로 이러한 문제가 없다.

$atomic_shared_ptr$ // C++20

다행히도 성능 문제가 아니다

ABA

- 해결책 (3/3)
 - 별도의 메모리 관리 기법을 사용한다.
 - EBR (Epoch Based Memory Reclamation)
 - Hazard Pointer

ABA

0 A

1 B

2 A

• Time Stamp Version

```

1  public T deq() throws EmptyException {
2      int[] lastStamp = new int[1];
3      int[] firstStamp = new int[1];
4      int[] nextStamp = new int[1];
5      int[] stamp = new int[1];
6      while (true) {
7          Node first = head.get(firstStamp);
8          Node last = tail.get(lastStamp);
9          Node next = first.next.get(nextStamp);
10         if (first == last) {
11             if (next == null) {
12                 throw new EmptyException();
13             }
14             tail.compareAndSet(last, next,
15                 lastStamp[0], lastStamp[0]+1);
16         } else {
17             T value = next.value;
18             if (head.compareAndSet(first, next, firstStamp[0],
19                 firstStamp[0]+1)) {
20                 free(first);
21                 return value;
22             }
23         }
24     }

```

28 bits
CAS

무제한 무잠금 큐

- 실습 #26 : 실습 #25의 Lock-Free Queue에서 ABA문제를 64bit CAS를 사용하여 해결하라.
 - visual studio에서 64비트 data type은 LONGLONG, 혹은 “long long”이다.

```
bool CAS(STPTR *next, NODE* old_p, NODE* new_p, int old_st, int new_st)
{
    STPTR old_v{ old_p, old_st };
    STPTR new_v{ new_p, new_st };
    return atomic_compare_exchange_strong(
        reinterpret_cast<volatile atomic_llong*>(next->ptr),
        reinterpret_cast<long long*>(&old_v.ptr),
        reinterpret_cast<long long*>(new_v.ptr));
}
```

LF Stamped Queue

- 속제 11 :

- Lock free stamped QUEUE의 구현

- 제출물

- .cpp 파일

- 실행속도 비교표 (Lock버전, Lock free, Lock free stamped)

- CPU의 종류 (모델명, 코어 개수, 클럭)

- 제출 : eclass

성능 (수목반)

– Nomutex : 851

– 1 : 1234

– 2 : 1378

– 4 : 1555

– 8 : 1669

– LockFree

• 1 : 1015

• 2 : 979

• 4 : 829

• 8 : 1073

● LFSTAMPEDQUEUE

– 507

– 650

– 856

– 928

– 946

ABA

● Lock-Free Stamp Queue 문제해결

– 중간 값 문제

- STAMPED NODE는 64비트 자료구조이고 캐시경계선에 놓일 수 있다.

- 해결

- atomic_llong으로 선언

```
struct stamped_pointer {  
    long long ptr;  
};
```

- alignas사용

```
struct alignas(8) stamped_pointer {  
    NODE *ptr;  
    int stamp;  
};
```

ABA

- Lock-Free Stamp Queue 문제 해결
 - 128bit CAS
 - LINUX는 `_int128` 사용
 - 또는 `__asm__`
 - Windows는 `InterlockedCompareExchange128` 사용

```
BOOLEAN InterlockedCompareExchange128(  
    LONG64 volatile *Destination,  
    LONG64           ExchangeHigh,  
    LONG64           ExchangeLow,  
    LONG64           *ComparandResult  
);
```

ABA

● Lock-Free Stamp Queue 문제 해결

– 메모리 재사용 시 안정성 문제

- head->next->value 접근 시 head가 가리키는 NODE가 재사용 되면서 head->next가 오염될 수 있다.
- 해결
 - free_list를 사용하여 오염을 막는다.
 - 이 free_list는 lock_free 이거나, thread 별로 따로 존재해도 된다.
 - thread_local 사용.
 - head->next의 nullptr검사 필요.

ABA

- EBR 샘플
 - eClass 실습자료 : ebr_if_queue.cpp
 - Oversubscription Pattern 문제

ABA

- 성능

- Intel® Core™ i7-7700@ 3.60Hz, 4Core, Hyperthread, 64GB Memory
- Single Thread : 756ms

Thread	Coarse QUEUE	Lock-Free QUEUE	STAMPED LF QUEUE	EBR LF QUEUE
1	1151	725	1135	473
2	1197	632	1027	534
4	1352	604	1158	652
8	1526	724	1269	774
16	1517	730	1280	790

정리

- Lock-Free Queue의 구현
- ABA문제
- ABA문제의 해결법