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(POOL)

- 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 중첩을 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)

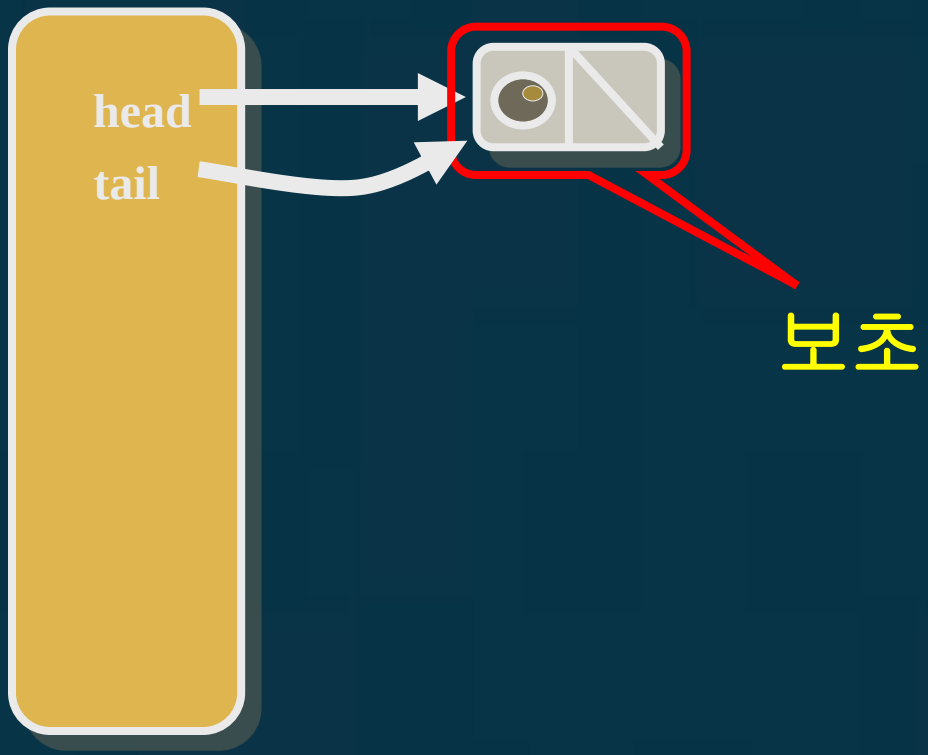
- 제한 큐 및 부분 큐
  - Lock-Free의 구현성격과는 맞지 않으므로 생략
  - ConditionVariable이라는 메소드가 필요
    - Lock을 가진 채로 Block()이 필요
    - Block()시 Lock을 해제, 다시 스케줄 될 때 Lock() 재 획득
    - 운영체제 호출 필요.
      - 쓰레드 스케줄링과 연동이 필수
- 무제한 완전 큐를 구현해보자.

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.

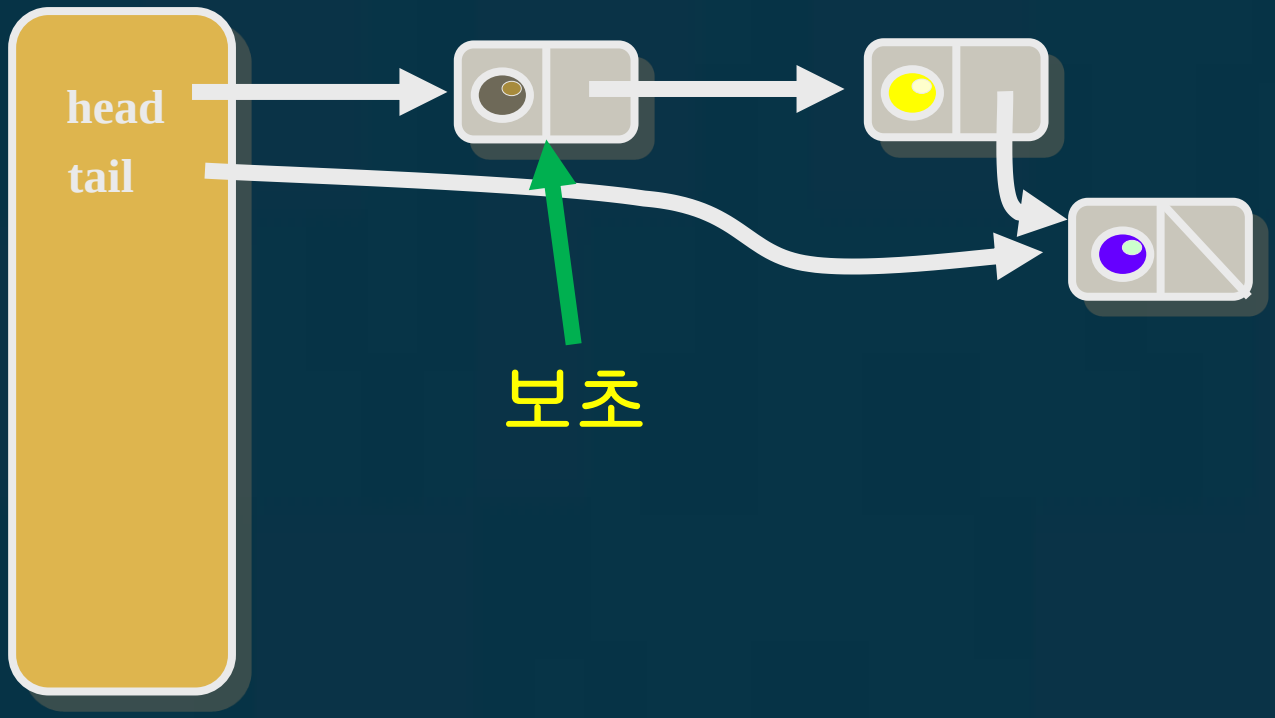
무제한 완전 큐

- 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, 8, 16에서의 성능을 비교하시오.
    - 각각 실행 전 Queue 를 클리어 하시오
    - 실행 후 queue에 있는 원소 20개를 출력하시오

```
int key = 0;

for (int i = 0; i < NUM_TEST / num_thread; i++) {
        if ((i < 32) || (rand() % 2 == 0))
                my_queue.Enqueue(key++);
        else
                my_queue.Dequeue();
}
```

비멈춤 동기화

- 과제 7 :
  - 성긴 동기화 큐의 구현
  - 제출물
    - .cpp 파일
    - 쓰레드 개수 별 실행속도 비교표
    - CPU의 종류 (모델명, 코어 개수, 클럭)
  - 제출 : eclass

성긴 동기화

- 결과

	성긴동기화			
1	1259			
2	1296			
4	1554			
8	1749			
16	1744			

싱글 쓰레드 : 550

비멈춤 동기화

- 성능 2022년
  - Nomutex : 995
  - 1 : 1384
  - 2 : 1233
  - 4 : 1558
  - 8 : 1719

무제한 무잠금 큐

- 무잠금 (Lock Free)
  - CAS를 사용
  - 다른 스레드가 임의의 위치에 멈추어 있어도 진행 보장

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
  - 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;
  }
}
```

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 쓰레드의 변경을 덮어 쓸 수 있다.

   }
}
```

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);
}  }
```

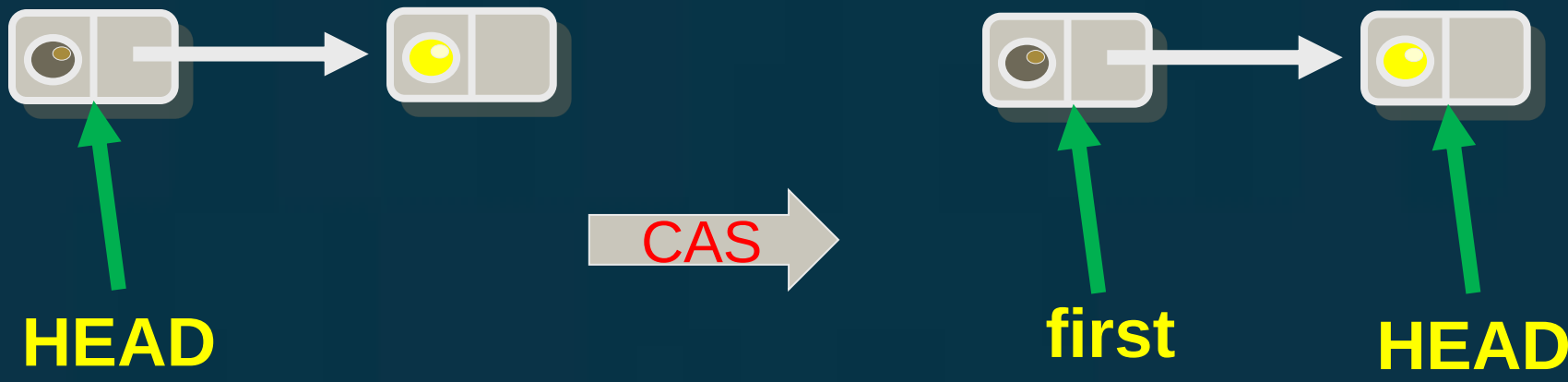
무제한 무잠금 큐 (2023 월금)

- 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;
   }
}
```

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.



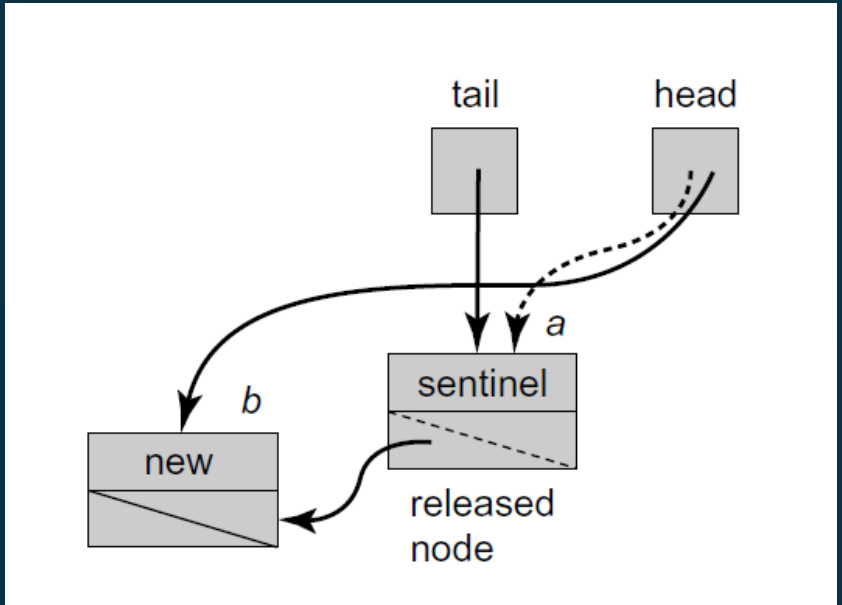
무제한 무잠금 큐

- 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;
   }
}
```

무제한 무잠금 큐

- DEQUEUE
  - ENQUEUE를 구현할 때 TAIL이 전진하지 못하는 경우가 발생할 수 있도록 했다.
    - Queue의 상태가 완전하지 못한 경우가 허용된다.
  - DEQUEUE도 그 경우에도 오류없이 Non-blocking으로 동작해야 한다.



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) return -1;  // -1 means ERROR
      if (first == last) {
         CAS(&tail, last, next);
         continue;
      }
      int value = next->item;
      if (false == CAS(&head, first, next)) continue;
      delete first;
      return value;
   }
}
```

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.
  - 주의
    - Single Thread에서는 문제없이 동작해야 한다.
    - Multi Thread에서 오동작 하는 경우 delete first를 제거해 보고 오류가 없어지면 제대로 구현한 것임.

숙제 10

- Lock-Free 무제한 Queue의 구현
  - 샘플 프로그램을 수정해서 제출
- 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.
    - Dequeue에서 delete를 생략했을 때 오류가 발생하지 않으면 그대로 제출하시오.
- 제출 : eclass

성능

- 결과 (2024, 2학기

	성긴동기화	LF 동기화		
1	1259	1117		
2	1296	1081		
4	1554	932		
8	1749	1201		
16	1744	1214		

싱글 쓰레드 : 550

비멈춤 동기화

- 성능 2024년 1학기
  - Nomutex : 957

	성긴동기화	Lock Free		
1	1232	1318		
2	1059	1012		
4	1441	1012		
8	1738	1199		

비멈춤 동기화

- 성능 2023년 화금
  - Nomutex : 869

	성긴동기화	Lock Free		
1	1293	1077		
2	1244	1053		
4	1525	847		
8	1670	1130		

비멈춤 동기화

- 성능 (2021-화목)
  - Nomutex : 768
  - 1 : 1092
  - 2 : 1468
  - 4 : 1685
  - 8 : 2054
- LockFree
  - 1 : 925
  - 2 : 1037
  - 4 : 841
  - 8 : 1248

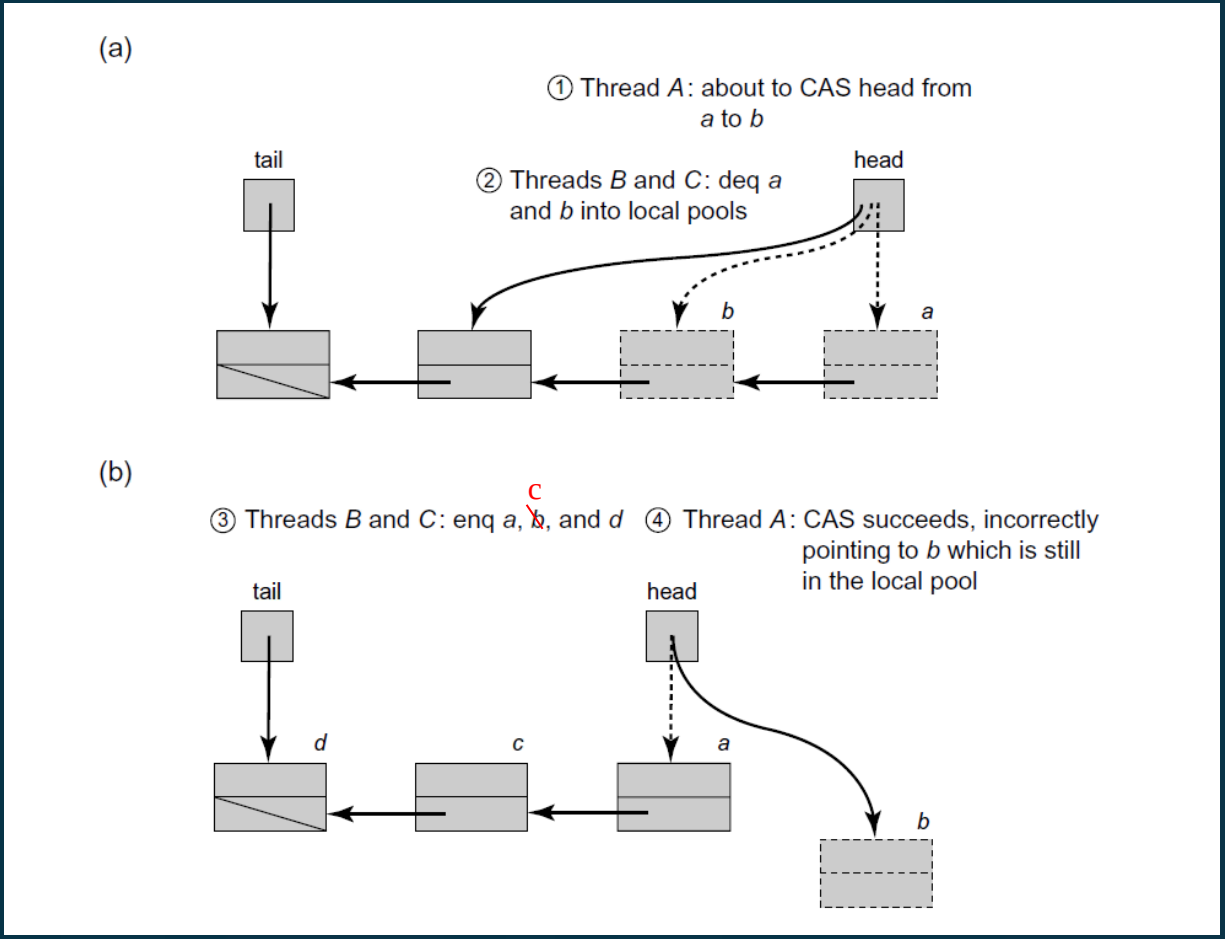
비멈춤 동기화

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

ABA

- 드물게 Crash!
  - Debug모드에서 실행해보자
  - 벤치마크에서 초기 큐 길이를 단축해보자
  - 쓰레드의 개수를 늘려보자.
- 노드의 재사용시 생기는 문제
  - new(), free()는 메모리를 재사용한다.
  - CAS사용시 다른 스레드에서 그 주소를 재사용했을 가능성이 있다.

ABA


(a)
① Thread A: about to CAS head from a to b
tail
② Threads B and C: deq a and b into local pools
head
b
a
(b)
③ Threads B and C: enq a, b, and d
C
④ Thread A: CAS succeeds, incorrectly pointing to b which is still in the local pool
tail
head
d
c
a
b

ABA

- 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, SC명령의 사용 (ARM, Alpha, PowerPC)
    - 값을 검사하는 것이 아니라 변경 여부를 검사
    - LL, SC 구조는 CAS보다 우월하다.
      - 그러나. LL,SC는 Wait-free가 불가능 하다.

ABA

- 해결책 (2/3) : 스마트 메모리 관리자
  - Reference Counter를 사용한다.
    - 다행히 합성포인터가 아니다.
    - `atomic<shared_ptr<>>` `// C++20`
    - ABA 해결 : 'first'가 첫번째 노드를 가리키고 있는 동안에는 그 노드는 재사용 될 수 없다.
    - 의미없다…. `atomic<shared_ptr<>>`가 lock free가 아니다.
  - Java는 garbage collection을 사용하므로 이러한 문제가 없다.

ABA

- atomic shared_ptr

```
class SH_NODE {
public:
   int v;
   atomic<shared_ptr<SH_NODE>> next;
   SH_NODE() : v(-1) {}
   SH_NODE(int x) : v(x) {}
};
```

```
class LF_SH_QUEUE {
   atomic<shared_ptr<SH_NODE>> head;
   atomic<shared_ptr<SH_NODE>> tail;
public:
   LF_SH_QUEUE() { head = make_shared<SH_NODE>(-1); tail.store(head); }
   bool CAS(atomic<shared_ptr<SH_NODE>>& next, shared_ptr<SH_NODE> old_p, const shared_ptr<SH_NODE>& new_p)
   { return next.compare_exchange_strong(old_p, new_p); }
   void ENQ(int x) {
      shared_ptr<SH_NODE> e = make_shared<SH_NODE>(x);
      while (true) {
         shared_ptr<SH_NODE> last = tail;
         shared_ptr<SH_NODE> next = last->next;
         shared_ptr<SH_NODE> comp = tail;
if (last != comp) continue;
if (nullptr == next) {
if (true == CAS(last->next, nullptr, e)) {
CAS(tail, last, e);
return;
}
}
else
CAS(tail, last, next);
}
}

int DEQ()
{
while (true) {
shared_ptr<SH_NODE>  first = head;
shared_ptr<SH_NODE>  last = tail;
shared_ptr<SH_NODE>  next = first->next;
shared_ptr<SH_NODE> comp = head;
if (first != comp) continue;
if (nullptr == next) return -1;
if (first == last) {
CAS(tail, last, next);
continue;
}
int value = next->v;
if (false == CAS(head, first, next))
continue;
return value;
}
}

void print20()
{
shared_ptr<SH_NODE> p = head;
p = p->next;
for (int i = 0; i < 20; ++i) {
if (p == nullptr) break;
cout << p->v << ", ";
p = p->next;
}
cout << endl;
}

void clear()
{
head.store(tail);
}
};
```

비멈춤 동기화

- 성능 2024년 1학기
  - Nomutex : 957

	성긴동기화	Lock Free	Shared_ptr	
1	1232	1318	257 0	
2	1059	1012	483 0	
4	1441	1012	1008 0	
8	1738	1199	1442 0	

비멈춤 동기화

- 성능 2022년
  - Nomutex : 764

	성긴동기화	Lock Free	Shred_ptr LF	
1	1283	968	2514	
2	1284	1024	4245	
4	1730	868	10992	
8	2163	1210	16036	

ABA

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

ABA

- 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],
                firstStamp[0]+1)) {
19            free(first);
20            return value;
21          }
22        }
23    }
24  }
```

무제한 무잠금 큐

- 실습 #26 : 실습 #25의 Lock-Free Queue에서 ABA문제를 64bit CAS를 사용하여 해결하라. x86 모드로 컴파일 하시오
  - visual studio에서 64비트 data type은 LONGLONG, 혹은 "long long"이다.

```
class STPTR {
std::atomic_llong m_stpr;
bool CAS(STNODE* old_ptr, STNODE* new_ptr, int old_st, int new_st)
{
   long long old_v = reinterpret_cast<int>(old_ptr);
   old_v = old_v << 32 + old_st;
   long long new_v = reinterpret_cast<int>(new_ptr);
   new_v = new_v << 32 + new_st;
   return std::atomic_compare_exchange_strong(&m_stpr, &old_v, new_v);
}
```

ABA

- Lock-Free Stamp Queue 문제해결 (32bit)
  - 중간 값 문제
    - STAMPED NODE는 64비트 자료구조이고 캐시경계선에 놓일 수 있다.
    - 해결
      - atomic_llong으로 선언

```
struct stamped_pointer {
    atomic_llong ptr;
};
```

ABA

- Lock-Free Stamp Queue 문제해결 (64bit)
  - 128bit CAS 필요, CPU에 명령어 존재
  - LINUX는 _int128사용
    - 또는 __asm__
  - Windows는 InterlockedCompareExchange128사용

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

```
bool CAS(ST_PTR *next1, ST_NODE *old_ptr, ST_NODE *new_ptr, long long old_stamp, long long
new_stamp)
{
   ST_PTR old_st { old_ptr, old_stamp };
   return InterlockedCompareExchange128( reinterpret_cast<LONG64 volatile*>(next1),
                                         new_stamp, reinterpret_cast<LONG64>(new_ptr),
                                         reinterpret_cast<LONG64 *>(&old_st));
}
```

  - 자료구조를 128bit 단위로 정렬해야 한다. (캐시라인 문제)

```
class alignas(16) ST_PTR {
public:
   ST_NODE*  volatile ptr;
   long long volatile stamp;
```

  - 하지만 모든 Stamped Pointer load/store가 atomic 이어야 한다.
    - atomic_128int는 C++11에 없다.
    - 모든 load/store에 InterlockedCompareExchange128을 사용해야 한다.
      - 성능이 떨어진다!!!!

LF Stamped Queue

- 숙제 12 :
  - Lock free stamped QUEUE의 구현
    - 첨부 파일의 SLF_QUEUE 완성 시키시오.
    - 32bit 모드에서 동작시키시오.
  - 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

성능

- 결과 (2024, 2학기

	성긴동기화	LF 동기화	32bit Stamp	
1	1259	1117	2021	
2	1296	1081	1750	
4	1554	932	2160	
8	1749	1201	2816	
16	1744	1214		

싱글 쓰레드 : 550

ABA

- Lock-Free Stamp Queue 문제해결
  - 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_lf_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