|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **작성자** | **2014132002 구태균** | **팀명** | - |
| **주차** | **23 주차** | **기간** | **2020.06.18 ~ 2020.06.24** | **지도교수** | **정 내 훈** (서명) |
| **이번주 한일** | * Atomic shaerd\_ptr 구현 (1)  1. 벤치마크 프로그램 2. 게으른 동기화 | | | | |

**[ Atomic 구현이 필요한 method ]**

[control\_block]

1. ctr\_block(Tp\* other) // x : make\_shared()에서만 이용
2. virtual ~ctr\_block() // x : 구현 내용 없음
3. void destroy(std::true\_type) // x : delete ptr 외 내용 없음
4. void destroy(std::false\_type) // ? : CAS-method 사이에서 읽기는? (0 확인) // 변경
5. void add\_ref\_copy() // o : CAS(ref\_count + 1), return(\*this) // 변경
6. void release() // ? : CAS(ref\_count - 1)를 --로 교체 가능? // 변경
7. int get\_use\_count() // ? : CAS-method 사이에서 읽기는? (return ref\_count)
8. void weak\_add\_ref() // o : CAS(ref\_count + 1), return(\*this) // 변경
9. void weak\_release() // ? : CAS(ref\_count - 1)를 --로 교체 가능? // 변경

[shaerd\_ptr]

1. Tp\* get\_ptr() // x : 사라질 함수 (Atomic 구현) // 제거
2. ctr\_block<Tp>\* get\_ctr() // x : 사라질 함수 (Atomic 구현) // 제거
3. ctr\_block<Tp>\* add\_ref\_copy() // o : // 추가
4. shared\_ptr() // x : 하나의 thread에서만 생성
5. shared\_ptr(nullptr\_t) // x : shared\_ptr() 와 동일 // 변경
6. shared\_ptr(Tp\* other) // x : 하나의 thread에서만 생성 or 사라질 함수 (Atomic 구현) ??
7. shared\_ptr(const shared\_ptr& other) // o : other.ctr.add\_ref() / ctr = other.ctr
8. shared\_ptr& operator=(nullptr\_t) // o : ctr.release() / ctr = nullptr // 변경
9. shared\_ptr& operator=(const shared\_ptr& other) // 변경

// o : ctr.release() / other.ctr.add\_ref() / ctr = other.ctr

1. ~shared\_ptr() // x : ctr.release()만 구현 // release()자체가 atomic
2. Tp\* get() // ? : CAS-method 사이에서 읽기는? (ctr->ptr)
3. Tp& operator\*() // x : get()자체가 atomic
4. Tp\* operator->() // x : get()자체가 atomic
5. operator bool() // ? : CAS-method 사이에서 읽기는? (ctr->ptr)
6. int use\_count() // x : ctr->get\_use\_count()자체가 atomic
7. void reset() // o : ctr->release() // CAS(ctr = nullptr)
8. void swap(shared\_ptr& other) // o : ctr = other.ctr & other.ctr = ctr을 동시에
9. operator== & operator!= // ? : CAS-method 사이에서 읽기는? (ctr->ptr)
10. shared\_ptr<Tp> make\_shared(Args&&... \_Args) // x : 하나의 thread에서만 shared 생성

[weak\_ptr]

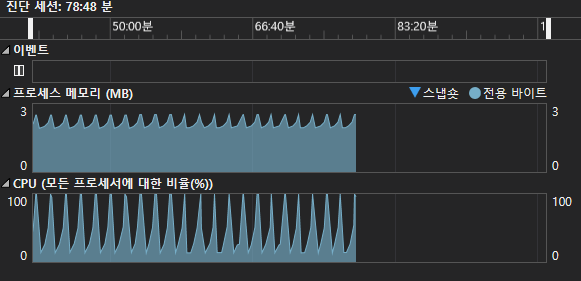
1. weak\_ptr() // x : 하나의 thread에서만 생성
2. weak\_ptr(nullptr\_t) // x : weak\_ptr() 와 동일
3. weak\_ptr(const shared\_ptr<Tp>& other) // o : other.ctr.weak\_add\_ref() / ctr=other.ctr
4. weak\_ptr(const weak\_ptr<Tp>& other) // o : other.ctr.weak\_add\_ref() / ctr=other.ctr
5. ~weak\_ptr() // x : ctr.weak\_release()만 구현 // weak\_release()자체가 atomic
6. weak\_ptr& operator=(nullptr\_t) // o : ctr.weak\_release() / CAS(ctr = nullptr)
7. weak\_ptr& operator=(const shared\_ptr<Tp>& other)

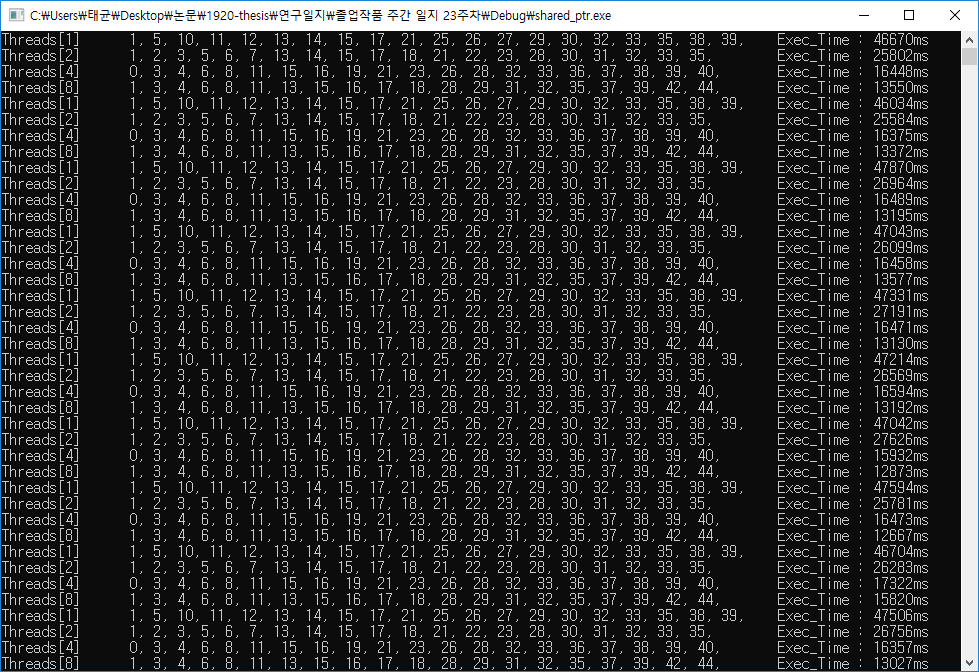
// o : ctr.weak\_release() / other.ctr.weak\_add\_ref() / CAS(ctr = other.ctr)

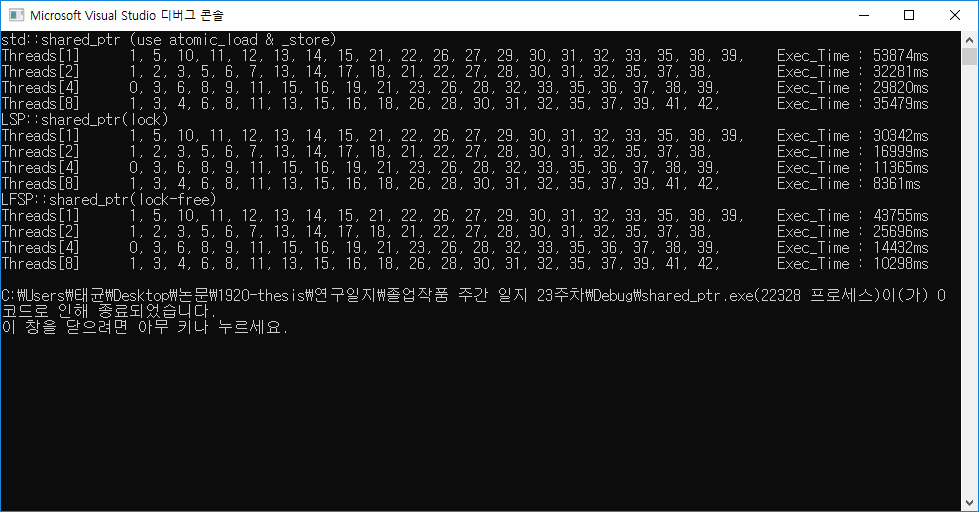
1. weak\_ptr& operator=(const weak\_ptr& other)

// o : ctr.weak\_release() / other.ctr.weak\_add\_ref() / CAS(ctr = other.ctr)

1. shared\_ptr<Tp> lock() // ? : CAS-method 사이에서 읽기는? (ctr->ptr)
2. void reset() // o : ctr->weak\_release() // CAS(ctr = nullptr)
3. void swap(weak\_ptr& other) // o : ctr = other.ctr & other.ctr = ctr을 동시에







|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | **해결 방안** |  |
| **다음 주차** | **24 주차** | **다음 기간** | **2020.06.25 ~ 2020.07.01** |
| **다음주 할 일** |  | | |
| **지도교수**  **Comment** |  | | |