|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **작성자** | **2014132002 구태균** | **팀명** | - |
| **주차** | **12, 13 주차** | **기간** | **2020.03.30 ~ 2020.04.05**  **2020.04.06 ~ 2020.04.12** | **지도교수** | **정 내 훈** (서명) |
| **이번주 한일** | * Weak\_ptr lock() 구현 * Blocking atomic shared\_ptr 구현 * Enable\_shared\_from\_this 미완성 * Atomic\_load Atomic\_store 미완성 | | | | |

**[Blocking shared\_ptr]**

template <typename Tp>

class shared\_ptr

shared\_ptr(const shared\_ptr& other)

{ **other.ctr->lock();** … **other.ctr->unlock();** }

shared\_ptr(const weak\_ptr<Tp>& other)

{ **other.ctr->lock();** … **other.ctr->unlock();** }

shared\_ptr& operator=(const shared\_ptr& other)

{ **other.ctr->lock();** … **other.ctr->unlock();** }

shared\_ptr& operator=(const weak\_ptr<Tp>& other)

{ **other.ctr->lock();** … **other.ctr->unlock();** }

template<typename Tp>

class weak\_ptr

weak\_ptr(const shared\_ptr<Tp>& other)

{ **other.ctr->lock();** … **other.ctr->unlock();** }

weak\_ptr(const weak\_ptr<Tp>& other)

{ **other.ctr->lock();** … **other.ctr->unlock();** }

weak\_ptr& operator=(const shared\_ptr<Tp>& other)

{ **other.ctr->lock();** … **other.ctr->unlock();** }

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

{ **other.ctr->lock();** … **other.ctr->unlock();** }

**Blocking shared\_ptr 구현 완료**

* **테스트 미확인 ( atomic\_load(), atomic\_store() 구현 부족 )**

**[Enable\_shared\_from\_this]**

template<typename Tp>

class enable\_shared\_from\_this

{

public:

shared\_ptr<Tp> shared\_from\_this()

{

return (shared\_ptr<Tp>(Wptr));

}

weak\_ptr<Tp> weak\_from\_this()

{

return Wptr;

}

protected:

enable\_shared\_from\_this()

: Wptr()

{

//Wptr.ptr = reinterpret\_cast<Tp\*>(this);

//Wptr.ctr = ?

}

enable\_shared\_from\_this(const enable\_shared\_from\_this& other)

: Wptr()

{}

enable\_shared\_from\_this& operator=(const enable\_shared\_from\_this&)

{

return (\*this);

}

~enable\_shared\_from\_this() = default;

private:

weak\_ptr<Tp> Wptr;

};

**현재상황**

* **class 구조 구현 완료**
* **정상작동x**

**문제점**

shared\_ptr<TEST> get() { return shared\_from\_this(); }

shared\_ptr<Tp> shared\_from\_this() { return (shared\_ptr<Tp>(Wptr)); }

**-------------------------------------------------------------------------------------------------**

TEST\* num = new TEST(0, 100); // 객체 생성

shared\_ptr<TEST> n1(num); // [ptr] : num / [ctr\_block] shared\_ref : 1, weak\_ref : 0

shared\_ptr<TEST> n2(n1->get()); // [ptr] : num / [ctr\_block] nullptr

* 문제 : n2의 경우, control\_block이 비정상.
* Control\_block을 shared\_ptr(Tp\* other) : ptr(other), ctr(new ctr\_block<Tp>(other)) {}에서만 생성해, Wptr가 초기화되지 않음.

**결론**

* Enable\_shared\_from\_this에서 Wptr의 초기화 방법

**[atomic\_load()]**

**<Visual Studio>**

template< class T >  
T atomic\_load( const [std::atomic](http://en.cppreference.com/w/cpp/atomic/atomic)<T>\* obj ) noexcept;

**<구현 함수>**

template<typename Tp>

Tp& atomic\_load(const shared\_ptr<Tp>& sptr);

template<typename Tp>

Tp& atomic\_load(const weak\_ptr<Tp>& wptr);

template<typename Tp>

shared\_ptr<Tp>& atomic\_load(const shared\_ptr<Tp>& sptr)

template<typename Tp>

weak\_ptr<Tp>& atomic\_load(const weak\_ptr<Tp>& wptr);

**[atomic\_store()]**

**<Visual Studio>**

template< class T >

void atomic\_store( [std::atomic](http://en.cppreference.com/w/cpp/atomic/atomic)<T>\* obj,typename [std::atomic](http://en.cppreference.com/w/cpp/atomic/atomic)<T>::value\_type desr ) noexcept;

**<구현 함수>**

template<typename Tp>

Tp atomic\_store(const shared\_ptr<Tp>\* sptr, Tp ptr);

template<typename Tp>

Tp atomic\_store(const weak\_ptr<Tp>\* wptr, Tp ptr);

**<미구현 함수>**

template<typename Tp>

Tp atomic\_store(const shared\_ptr<Tp>\* sptr, ctr\_block cb);

template<typename Tp>

Tp atomic\_store(const weak\_ptr<Tp>\* wptr, ctr\_block cb);

template<typename Tp>

Tp atomic\_store(const shared\_ptr<Tp>\* in, shared\_ptr<Tp>\* data);

template<typename Tp>

Tp atomic\_store(const weak\_ptr<Tp>\* in, weak\_ptr<Tp>\* data);

**문제점**

* Visual에서 제공하는 atomic\_load(), atomic\_store()와 달리, shaerd\_ptr은 ptr과 control\_block을 모두 필요로 함. 하지만 control\_block을 직접 접근하면 위험함.
* Atomic\_load(), atomic\_store()가 shared\_ptr,weak\_ptr의 operator=과 기능이 동일함.

**결론**

* 객체 pointer만 이용하기에 control\_block을 이용할 수 없고, shared\_ptr을 이용하기엔 operator=와 기능이 동일함.

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | **해결 방안** |  |
| **다음 주차** | **14 주차** | **다음 기간** | **2020.04.13 ~ 2020.04.19** |
| **다음주 할 일** |  | | |
| **지도교수**  **Comment** |  | | |