2024 01 08

Std::thread

```
#include <thread>
void foo(int &x)
{

}

using namespace std;

int main()
{
    int x{ 35 };

    jthread t1{ foo , ref(x)};
    jthread t2{ foo , ref(x)};
    jthread t3{ foo , ref(x)};

}
```

```
/*
    Birden fazla thread'in ortaklaşa olarak kullandığı paylaşımlı bir değiken
varsa
    tanımsız davranış olmaması için:

    a) paylaşımlı değişken const bir değişken olacak
    b) thread'ler okuma amaçlı paylaşımlı değişkeni kullanacak

    sequenced-before relationship
        eğer tek bir thread söz konuysa:
            x = 5;
            y = x;

            kodunda, y = 5  olmasının garantisi var

    happens-before relationship
        birden fazla thread varsa:
        Mesela,
        A thread'in oluşturduğu sonuç B Thread'i tarafınan görülebilir durumdadır.
*/
```

MUTEX SINIFLARI

```
/*
    MUTEX SINIFLARI
        std::mutex
        std::timed_mutex
        std::recursive_mutex
        std::recursive_timed_mutex
        std::shared_mutex
        std::shared_time_mutex
*/

/*
    std::mutex
    lock() - mutex'i edinmek için
    unlock() - mutex'i serbest bırakmak için
    try_lock() - mutex'i edinmeye çalışmak için
*/
```

```
#include <mutex>
std::mutex mtx;
void func()
{
    mtx.lock();
    // critical section
    mtx.unlock();

    if (mtx.try_lock()) // bool döndürür
    {
        // edinirse burayı yapacak
    }

    mtx.native_handle(); //başka apilerde kullanmak için
}

//////

std::mutex mtx;
int main()
{
    // kopyalama ve taşıma yok
    auto y = move(mtx);
    auto x = mtx;
}
```

```
class Myclass
{
    public:
        void foo() const
        {
            mtx.lock();

            mtx.unlock();
        }
    private:
        mutable std::mutex mtx;
}
```

```cpp
// SORU
using namespace std;

std::mutex mtx;
int cnt = 0;

void foo()
{
    for (int i = 0; i< 10'000; ++i)
        mtx.lock();
        ++cnt;
        mtx.unlock();
}

void bar()
{
    for (int i = 0; i < 10'000; ++i)
        mtx.lock();
        --cnt;
        mtx.unlock();
}

int main()
{
    std::thread t1{foo};
    std::thread t2{bar};

    // senkronizasyon gerekir
    t1.join();
    t2.join();
}
```

```cpp
std::mutex mtx;
void foo()
{
    throw std::runtime_error{ "error from foo "};
}

void func()
{
    mtx.lock();
    //lock_guard(mtx) // RAII
    try
    {
        foo();
        mtx.unlock(); // unlock olmıcak buna dead lock denir
    }
    catch (const std::exception& ex)
    {

    }
}
```

MUTEX SARMALAYAN RAII SINIFLARI

```
MUTEX SARMALAYAN RAII SINIFLARI
    lock_guard
    unique_lock
    scoped_lock
    shared_lock
*/
```

```
// std::lock_guard
std::mutex mtx;

void foo()
{
    // mutex'i sarmalar ve lock eder
    lock_guard<mutex> lock{ mtx }; // scope sonunda unlock eder
    // lock_guard lock{mtx} // CTAD

    lock_guard lg{mtx}
    auto x = lg; // no copy
    auto y = move(lg); // no move
}
```

```
std::mutex mtx;
void func()
{
    // adopt_lock
    lock_guard lg{ mtx, adopt_lock};
}
```

```
// std::timed_mutex
/*
    try_lock_for
    try_lock_until
*/
void func()
{
    std:.timed_mutex mtx;

    if (mtx.try_lock_for(50ms))
    {
        // 50ms boyunca denicek ama kilidi elde edemezse false döncek
    }
}
```

```cpp
// ÖRNEK
using namespace std;

int cnt{};
std::timed_mutex mtx;
void try_increment()
{
    for (int i = 0; i < 100'000; ++i)
    {
        if (mtx.try_lock_for(1ms))
        {
            ++cnt;
            mtx.unlock();
        }
    }
}

int main()
{
    vector<thread> tvec;

    for (int i = 0; i < 10; ++i)
        tvec.emplace_back(try_increment);

    for (auto& t : tvec)
        t.join();

    cout << "cnt = " << cnt << "\n";
}
```

```cpp
// ÖRNEK
std::mutex mtx;

void foo()
{
    std::cout << "foo is trying to lock the mutex\n";
    mtx.lock();
    std::cout << "foo has locked the mutex\n";
    std::this_thread::sleep_for(600ms);
    std::cout << "foo is unlocking the mutex\n";
    mtx.unlock();

}

void bar()
{
    std::this_thread::sleep_for(100ms);

    std::cout << " bar is trying to lock the mutex\n";

    while (!mtx.try_lock())
    {
        std::cout << " bar could not lock the mutex\n";
        std::this_thread::sleep_for(100ms);
    }

    std::cout << "bar has locked the mutex\n";
    mtx.unlock();
}

int main()
{
    std::jthread t1{ foo };
    std::jthread t2{ bar };

}
```

```cpp
class List{

    public:
        void push_back(int x)
        {
            mtx.lock();
            mlist.push_back(x);
            mtx.unlock();
        }

        void print()const
        {
            std::lock_guard lg{ mtx};
            for (const auto val : mlist)
            {
                std::cout << val << ' ';
            }
            std::cout << "\n";
        }


    private:
        std::mutex mtx;
        std::list<int> mlist;
};

void func(List& list, int x)
{
    for (int i = 0; i < 10; ++i)
        list.push_back(x + i);
}
```

```cpp
//dead lock: bir threadin ileryememesi

 // std::lock --birden fazla mutex veriyoruz dead lock'tan korur

// std::scoped_lock
// lock_guard maliyet olarak farkı yok ama dead lock'tan korur
```

```cpp
std::mutex m;
timed_mutex tm;
recursive_mutex rm;

void foo()
{
    scoped_lock<std::mutex, std::timed_mutex, std::recursive_mutex> slock{ m, tm, 
rm};
}

// std::recursive_mutex
class Myclass
{
    public:
        void foo()
        {
            mtx.lock();
            bar();
            mtx.unlock();
        }

        void bar()
        {
            mtx.lock();

            mtx.unlock();
        }

    private:
        //mutable std::mutex mtx; // mutex'i birden fazla kitlemek tanımsız 
davranış
}       mutable std::recursive_mutex mtx; // birden fazla kitlemek legal

int main()
{
    using namespace std;

    Myclass m;

    thread t{ &Myclass::foo, ref(m)};
    t.join();
}
```