2024 01 05

Std::thread

```
/*
    thread t{foo , x}
        x bir dizi ise pointera dönüşecek
        x bir fonksiyon ise fonksiyon adresine dönüştürülecek.
        x const ise non-const'a dönüşecek
*/
```

```
class Myclass
{
    public:
        Myclass() = default;
        Myclass(const Myclass&)
        {
            std::cout << "copy ctor\n";
        }
        Myclass(Myclass&&)
        {
            std::cout << "move ctor\n";
        }
};

void foo(Myclass)
{
    /*
        -- Myclass& olsaydı  syntax hatası olur çünkü fonksiyonun
        parametresine l ref değil r expr  gönderiyoruz.
        R value expr L ref ile bağlanamaz

        -- Myclass&& olsaydı syntax hatası olmaz
        sadece copy ctor çağrılır

        -- const Myclass&  olsaydı syntax hatası olmaz
        const L value ref R value expr bağlanabilir
        sadece copy ctor çağrılır

    */
}

int main()
{
    using namespace std;

    Myclass m;

    thread t{ foo, m };
    /*
        decay copy yapılacak --copy ctor çağrılacak
        fonksiyon parametresine r value veriyoruz --move ctor çağrılacak

        copy ctor
        move ctor
    */
```

```
    t.join();

    thread t1{foo, Myclass{}};
    /*
        gönderdiğimiz r value olduğu için move ctor ile oluşturulur
        fonksiyona geçerken r value veriyoruz move ctor çağrılır.

        move ctor
        move ctor
    */
    t1.join();

    const Myclass m1;
    /*
        const'luk düşer
        copy ctor
        move ctor
    */
    thread t2{foo, m1};
    t2.join();

    Myclass m2;
    thread t3{foo, ref(m)};
    /*
        copy veya move ctor çağrılmaz.
    */
    t3.join();
}
```

```
// get_id
void func()
{

}

int main()
{
    using namespace std;

    thread t1{ func };
    auto x = t1.get_id(); // auto --> class std::thread_id
    cout <<  x  << "\n";
    t1.join()
}
```

```
/*
    tüm threadlerin id'leri farklı
    eğer thread joinable durumda değilse get_id 0 döner
*/

// this_thread
#include <syncstream>
void func()
{
    osyncstream{cout} << std::this_thread::get_id() << "\n";
```

```
}
int main()
{
    using namespace std;

    thread t1{func};
    cout << t1.get_id() << "\n";
    t1.join();
}
```

```
std::thread::id main_thread_id;
 void func()
 {
     if (std::this_thread::get_id() == main_thread_id)
         std::cout << "main thread\n";
     else
         std::cout << "ikincil thread\n";
 }

 int main()
 {
     using namespace std;

     main_thread_id = std::this_thread::get_id();
     func(); // main thread

     thread t{ func };
     t.join(); // ikincil thread
 }
```

```
/*

 thread kütüphanesinde sonu for ve until ile biten fonksiyonlar bulunur
 xxxfor     duration ister
 xxxuntil   timepoint ister
 */

 // sleep_for
 void func()
 {
     std::this_thread::sleep_for(2300ms);
     std::cout << "emre bahtiyar\n";
 }

 int main()
 {
     using namespace std;

     thread t{func};
     t.join();
 }
```

```
// native_handle()
 int main()
```

```
{
    using namespace std;

    thread t{ [] {}};
    // işletim sistemin apilerine geçicek handle verir
    auto handle = t.native_handle();
}
```

```
// hardware_concurrency()
int main()
{
    using namespace std;
    // kaç tane thread açabilirim
    auto b = thread::hardware_concurrency()
}
```

```
// thread exceptions
void func(int x)
{
    // exception burda yakalanırsa catch bloğuna girer
    if (x % 2 == 0)
    {
        throw std::runtime_error{ "even number error\n" };
    }
}

int main()
{
    using namespace std;
    /*
       exception vermez doğrudan terminate fonksiyonu çağrılır.
    */
    try
    {
        thread t{ func, 12};
        t.join();
    }
    catch (const std::exception &ex)
    {

        std::cout << "exception caught: " << ex.what() << "\n";
    }
}
```

```
/*
   std::exception_ptr
       exception sarmalar
   current_exception()
       bir exception yakalandığı zaman exception ptr nesnesi döndürür döndürür
   rethrow_exception()
       exception ptr'yi rethrow eder.
*/
```

```cpp
void handle_saved_exception(std::exception_ptr eptr)
{
    try
    {
        if (eptr)
        {
            std::rethrow_exception(eptr);
        }
    }
    catch (const std::out_of_range& ex)
    {
        std::cout << "hata yakalandi : " << ex.what() << "\n";
    }

}

int main()
{
    std::exception_ptr exptr;
    try{
        std::string name{"emre"};
        auto c = name.at(45);
    }
    catch (...) {
        eptr = std::current_exception();
    }
}
```

```cpp
std::exception_ptr eptr{ nullptr };
void func(int x)
{
    std::cout << "func cagrildi x = " << x << "\n";
    try {
        if (x % 2  != 0)
        {
            throw std::runtime_error{"hatali arguman " + std::to_string(x)}
        }
    }
    catch(...)
    {
        eptr = std::current_exception();
    }
    std::cout << "func islevi sona eriyor\n";
}
```

```cpp
int main()
{
    thread tx{ func, 12};
    tx.join();
    try {
        if (eptr)
          rethrow_exception(eptr);
    }
    catch (const std::exception& ex)
    {
        std::cout << "exception caught: " << ex.what() << "\n";
    }
}
std::vector<std::exception_ptr> g_epvec;
std::mutex mtx;
void f1() {throw std::runtime_error{ "hata" };}
void f2() {throw std::invalid_argument{ "gecersiz arguman" };}
void f3() {throw std::out_of_range{ "aralik disi değer" };}

void tfunc1()
{
    try {
        f1();
    }
    catch (...) {
        std::lock_guard guard{mtx};
        g_epvec.push_back(std::current_exception();
    }
}
void tfunc2()
{
    try {
        f2();
    }
    catch (...) {
        g_epvec.push_back(std::current_exception();
    }
}
void tfunc3()
{
    try {
        f3();
    }
    catch (...) {
        g_epvec.push_back(std::current_exception();
    }
}

int main()
{
    using namespace sd;

    thread t1 {tfunc1};
    thread t2 {tfunc2};
    thread t3 {tfunc3};

    t1.join();
    t2.join();
```

```
    t3.join();

    for (const auto &ex : g_epvec)
    {
         try {
             rethrow_exception(ex);
         }
         catch (const std::exception& ex) {
             std::cout << "exception caught : " << ex.what() << "\n";
         }
    }
}
```

```
// make_exception_ptr()
int main()
{
    using namespace std;

    exception_ptr eptr;

    try
    {
        throw std::runtime_error{ "hata" };
    }
    catch(...)
    {
        eptr = current_exception();
    }

     // yukarıdaki yerine
     auto eptr = make_exception_ptr(std::runtime_error{ "hata" };
}
```

thread_local storage

```
// thread_local bir keyword

thread_local int x{0};

void foo(const std::string& thread_name)
{
    // her thread'in thread_local değişkeni o thread için tek
    ++x;

    osyncstream{cout} < < "thread" << thread_name << " x = " <<  << "\n";
}

int main()
{
    jthread tx{ foo, "tx" };    // thread tx x = 1
    jthread ty{ foo, "ty" };    // thread tx x = 2
    jthread tm{ foo, "tm" };    // thread tx x = 3
    jthread tz{ foo, "tz" };    // thread tx x = 4
}
```

```cpp
thread_local int ival = 0;

void thread_func(int *p)
{
     *p  = 42;
     std::cout << "*p = " << *p << "\n";
     std::cout << "ival = " << ival << "\n";
}

int main()
{
    std::cout << "ival = " <<  ival << "\n"; // ival = 0

    ival = 9;
    std::cout << "ival = " << ival << "\n"; // ival = 9

    std::thread t{thread_func, &ival};
    t.join() // *p = 42, ival = 0

    std::cout << "ival = " << "\n"; // ival = 42
    /*
        -thread_func'ta ival hala 0 thread_local olduğu için
        -main thread ival'nin adresini gönderdik o 9 değerini 42 yaptı
        bundan dolayı ival son olarak 42 kaldı
    */
}
```

```cpp
// ÖRNEK
 std::mutex mtx;
 void foo(int id)
 {
     int x = 0;              // automatic storage
     static int y= 0;        // static storage
     thread_local int z = 0; // thread_local storage
     ++x;
     ++z;
     mtx.lock();
     ++y;

    std::cout << "thread id: " << id << " x(automatic) = " << x << "\n";
    std::cout << "thread id: " << id << " y(static) = " << y << "\n";
    std::cout << "thread id: " << id << " z(thread_local) = " << z << "\n";
    mtx.unlock();
```

```
}
int main()
{
      using namespace std;
      jthread t1{ foo, 1};
      jthread t2{ foo, 2};
      jthread t3{ foo, 3};
      jthread t4{ foo, 4};
      /*
           static değişken 4 e kadar çıkar
           automatic ve thread_local 1 çıkar hep
      */

}
// ÖRNEK
thread_local std::string name{ "cemal" };
void func(const std::string& surname)
{
    name += surname;
    // name'lerin adresleri farklı olur.
    std::osyncstream{ std::cout } << name << "&name = " << &name << "\n";
}

int main()
{
    const char* const pa[] = { "akkan", "toprak", "bahtiyar", "ersoy",
"canberk"};
    vector<thread> tvec;

    for (auto p : pa)
    {
        tvec.emplace_back(func, p);
    }

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

```
class Myclass {
public:
   Myclass(int x)
   {
       std::osyncstream{ std::cout } << "Myclass ctor  x = " << x << " this = "
<< this << "\n";
   }
   ~Myclass()
   {
       std::osyncstream{ std::cout } << "Myclass dtor this = " << this << "\n";
   }
};
void func(int x)
{
thread_local Myclass m{ x };
}
void foo(int x)
{
  std::osyncstream{ std::cout } <<  "foo(int) cagrildi x = " << x << "\n";
  func(x);
```

```
   std::osyncstream{ std::cout } <<  "foo(int) sona eriyor x = " << x << "\n";
}
int main()
{
 using namespace std;
 {
     jthread t1{ foo, 1};
     jthread t2{ foo, 1};
     jthread t3{ foo, 1};
     jthread t4{ foo, 1};
 }

 std::cout << "main devam ediyor\n";
}
// ÖRNEK

 /*
    thread_local değişken yerel değişken de olabilir. Böyle bir değişkenin hayatı
    kapsamı sonununda bitmez. Threadin yürütülmesi sonunda hayatı sona erer.
 */

thread_local std::mt19937 eng{ 4542345u};

void foo()
{
 std::uniform_int_distribution dist{ 10, 99};

 for (int i = 0; i< 10; ++i)
 {
     std::cout << dist(eng) << ' ';
 }
}

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

 t1.join();

 std::cout << "\n";
 std::thread t2{ foo };
 t2.join();
}
```

```cpp
//ÖRNEK
// Myclass nesnesi thread_local olduğu için thread'in sonuna kadar yaşar
class Myclass
{
    public:
        Myclass()
        {
            std::osyncstream{ std::cout } << "Myclass ctor this : " << this  <<
"\n";
        }

        ~Myclass()
        {
            std::osyncstream{ std::cout } << "Myclass dtor this : " << this  <<
"\n";
        }
}

void foo()
{
    std::ostringstream{ std::cout } << "foo called\n";
    thread_local Myclass m;
    std::ostringstream{ std::cout } << "foo ends\n";
}

void bar()
{
    using namespace std::chrono_literals;

    std::ostringstream{ std::cout } << "bar called\n";
    foo();
    std::this_thread::sleep_for(3s);
    std::ostringstream{std::cout} << "bar ends\n";
}

int main()
{
    std::thread t{ bar };
    t.join();
}
```

```cpp
/*
    data race: birden fazla thread paylaşımlı değişkeni kullancak ve
    en az biri yazma amaçlı kullancak. data race varsa tanımsız davranıştır

    race condition: paylaşımlı değişkenin kullanılması ama
    herhangi bir tanımsız davranış olmak zorunda değil

*/

// data race
int gcount = 0;
void foo()
{
    for (int i = 0; i < 100'000; ++i)
        ++gcount;

}

void bar()
{
    for (int i = 0; i < 100'000 << ++i)
            --gcount;
}

int main()
{
    /*
        thread kullanmasak önce gcount  100'000 olacak sonra  0 olacak
        data race olmıcak bu durumda.

        thread kullanınca gcount son değeri belirsiz olacak bu data race
        tanımsız davranış
    */
    using namespace std;
    {
        jthread t1{ foo };
        jthread t2{ bar };
    }
    cout << "gcount = "<< gcount << "\n";
}
```

```cpp
// mutual exclusion
/*
    acquire
    critical section
    release
*/
```

```cpp
// std::mutex
int gcount = 0;

std::mutex mtx;

void func()
{
    mtx.lock(); // kilit boştaysa buraya gelen thread kilidi edenir. Diğer thread'ler giremez
    ++gcount;
    mtx.unlock(); // kilit serbest bırakılr
}

int main()
{
    thread t1{ func };
    thread t2{ func };
    thread t3{ func };
    thread t4{ func };
}
```