Лекция по C++

threads

Thread example

```
#include <thread>
#include <iostream>

void Foo() {
    std::cout << "foo";
}

int main() {
    std::thread foo_thread(Foo);
    foo_thread.join();
}
```

Threads basics

1. thread принимает в конструктор функцию, которую будет выполнять и ее аргументы

```
std::thread(func, args...)
```

2. Метод join ждет завершения выполнения потока

Threads basics

```cpp
#include <thread>
#include <iostream>

void Print(const std::string& text) {
    std::cout << text;
}

int main() {
    std::thread print_thread(Print, "asasa");
    print_thread.join();
}
```

Threads basics

```
void Loop() {
    while (1) {};
}

int main() {
    std::thread infinite_thread(Loop);

//  infinite_thread.join();
// wait for infinite_thread, would never stop

    infinite_thread.detach();
// infinite_thread is detached
}
```

Thread methods

```
class thread {
    bool joinable();
    void join();
    void detach();
    id get_id();
    static unsigned hardware_concurrency();
};
```

Threads basics

Обязательно вызывать join или detach!

```
void Print(const std::string& text) {
    std::cout << text;
}

int main() {
    std::thread print_thread(Print, "asasa");
} // std::terminate in thread destructor
```

Threads

```
void GetId() {
    std::cout << std::this_thread::get_id();
}

int main() {
    std::thread thread1(GetId);
    std::thread thread2(GetId);

    if (thread1.joinable()) thread1.join();
    else assert(0);

    if (thread1.joinable()) {
        assert(0);
        thread1.join();
    }
    thread2.join();
}
```

Race condition

```
int cnt = 0;
void Inc() {
    for (int i = 0; i < 10000; ++i) {
        ++cnt;
    }
}

int main() {
    std::thread thread1(Inc);
    std::thread thread2(Inc);

    thread1.join();
    thread2.join();

    std::cout << cnt;
}
```

Race condition

Состояние гонки – ошибка, при которой результат выполнения многопоточной программы зависит, от того, в каком порядке будут переключаться потоки.

Increment

1. mov cnt, eax
2. inc
3. mov eax, cnt

Mutex

```
#include <mutex>

int cnt = 0;
std::mutex m;
void Inc() {
    for (int i = 0; i < 100000; ++i) {
        m.lock();
        ++cnt; //thread safe code here
        m.unlock();
    }
}
int main() {
    std::thread thread1(Inc);
    std::thread thread2(Inc);
    thread1.join();
    thread2.join();
    std::cout << cnt; // ok, 200000
}
```

Mutex

```
class mutex {
    mutex(const mutex&) = delete;
    mutex& operator=(const mutex&) = delete;
    void lock();
    bool try_lock();
    void unlock();
};
```

Problems with mutex

```
std::mutex m;
void SomeFunc() {
    ...
}
void Test() {
    m.lock();
    SomeFunc();
    m.unlock();
}
```

Problems with mutex

```
std::mutex m;
void SomeFunc() {
    throw std::runtime_error("fail");
}
void Test() {
    try {
        m.lock();
        SomeFunc();
        m.unlock();
    }
    catch (std::exception&) {
        ...
    }
}
```

Lock guards

```
std::mutex m;
void SomeFunc() {
    throw std::runtime_error("fail");
}
void Test() {
    try {
        std::unique_lock<std::mutex> guard(m);
        SomeFunc();
    }
    catch (std::exception&) {
        ...
    }
}
```

Atomics

```
#include <atomic>

std::atomic_int cnt = 0;
void Inc() {
    for (int i = 0; i < 100000; ++i) {
        ++cnt;
    }
}
int main() {
    std::thread thread1(Inc);
    std::thread thread2(Inc);
    thread1.join();
    thread2.join();
    std::cout << cnt; // ok, 200000
}
```

Atomics

`std::atomic<T>`

- Операции, изменяющие состояние atomic выполняются атомарно, что позволяет избегать рейсов

Atomics

```
std::atomic<T>

store
load
compare_exchange_weak
compare_exchange_strong
fetch_add
...
```

Atomics

```
std::atomic_int cnt1 = 0;
void Inc1() {
    for (int i = 0; i < 1e7; ++i) {
        ++cnt1;
    }
}

int cnt2 = 0;
std::mutex m;
void Inc2() {
    for (int i = 0; i < 1e7; ++i) {
        m.lock();
        ++cnt2;
        m.unlock();
    }
}
```

Atomics

```
std::vector<std::function<void()>> funcs({Inc1, Inc2});
for (auto func : funcs) {
    auto start = std::chrono::steady_clock::now();
    std::thread thread1(func);
    std::thread thread2(func);
    thread1.join();
    thread2.join();
    auto end = std::chrono::steady_clock::now();
    auto ns = std::chrono::duration_cast
        <std::chrono::nanoseconds>(end - start).count();
    std::cout << ns << std::endl;
}
```

» 521887880
» 1893677893

Atomics

```
std::atomic_int value = 0;
int expected = value.load() + 1;
int desired = 2;
while (!value.compare_exchange_weak(expected, desired)) {
    // CAS cycle
}
std::cout << value;
```

если value == expected, положит desired в value, вернет true
если value != expected, положит value в expected, вернет false

Conditional variables

`std::condition_variable`

`notify_one`
`notify_all`
`wait`

Conditional variables

```
std::condition_variable cv_;

unique_lock<mutex> lck(m);
while (!something)
  cv.wait(lck);
```

Conditional variables

```
class Application {
    std::mutex mutex_;
    std::condition_variable condition_variable_;
    bool data_loaded_;
public:
    void LoadData() {
        std::lock_guard<std::mutex> guard(mutex_);
        data_loaded_ = true;
        condition_variable_.notify_one();
    }
    void MainTask() {
        std::unique_lock<std::mutex> mlock(mutex_);
        while (!data_loaded_)
            condition_variable_.wait(mlock);
    }
};
```