Семестр 2. Лекция 9. C++11. Threads. Множественное наследование.

Евгений Линский

12 Мая 2017

Неформальные воспоминания про потоки 1

- Процессы (программы)
  - “несколько программ запущены одновременно”
  - независимые адресные пространства
- Потоки (функции в программе)
  - ”несколько функций внутри одной программы запущены одновременно в разных потоках”
  - общее адресное пространство (могут иметь доступ к общим переменным)

Параллельное сложение векторов

```
typedef vector<int> ivec;
void sum_vec(const ivec& v1, size_t start, size_t end,
    const ivec& v2, ivec& res){
    for(int i = start; i < end; i++) {
        res[i] = v1[i] + v2[i];
    }
}
void parallel_sum_vec(const ivec& v1, const ivec& v2,
    ivec& res) {
    thread t1(sum_vec, (size_t)0, v1.size() / 2,
      cref(v1), cref(v2), ref(res));

    thread t2(sum_vec, v1.size() / 2, v1.size(),
      cref(v1), cref(v2), ref(res));
    t1.join(); t2.join();
}
parallel_sum_vec(v1, v2, res);
```

- std::cref — const reference
- g++ –std=c++11 vec_sum.cpp -o vs -lpthread

Неформальные воспоминания про потоки 2

- Переключением между процессами/потоками занимается планировщик ОС
- При переключении с потока 1 на поток 2 необходимо все регистры, используемые потоком 1 сохранить в память, а все регистры, используемые потоком 2 восстановить из памяти (переключени е контекста)
- NB:С точки зрения процесса/потока переключение происходит в произвольный момент времени!
- А еще бывает несколько процессоров. =)

Пример с прошлой лекции.

```
void hello(){
    std::cout << "Hello from thread " <<
      std::this_thread::get_id() << std::endl;
}

int main(){
    std::vector<std::thread> threads;

    for(int i = 0; i < 5; ++i){
        threads.push_back(std::thread(hello));
    }

    for(auto& thread : threads){
        thread.join();
    }

    return 0;
}
```

Пример с прошлой лекции. Гонки (Race conditions).

Может так:

```
Hello from thread 140276650997504
Hello from thread 140276667782912
Hello from thread 140276659390208
Hello from thread 140276642604800
Hello from thread 140276676175616
```

А может и так:

```
Hello from thread Hello from thread Hello from
thread 139810974787328Hello
from thread 139810983180032Hello from thread
139810966394624
139810991572736
139810958001920
```

Гонки (Race conditions). Еще примеры.

- Примитивные типы (int, ...) неатомарны (атомарный — операция не может быть прервана)
- Связный список: поток 1 вставляет элементы в середину, поток 2 выводит элементы на экран. “Контекст переключился” с 1 на 2, когда не все указатели next были проставлены.
- Еще классический пример

```
1 int x = 0;
2 // Thread 1:
3 while (!stop) {
4   x++;
5 }
6 // Thread 2:
7 while (!stop) {
8   if (x%2 == 0) cout<< x;
9 }
```

На экране может быть нечетное число. Почему?

Гонки (Race conditions). Проблема.

```
struct Counter {
    int value;
    Counter() : value(0){}
    void increment(){
        ++value;
    }
};
```

```
Counter counter;
vector<thread> threads;
for(int i = 0; i < 5; ++i){
    threads.push_back(thread([&counter](){
        for(int i = 0; i < 100; ++i){
            counter.increment();
        }
    }));
}
for(auto& thread : threads){ thread.join();}
cout << counter.value << endl;
```

Гонки (Race conditions). Вывод.

```
442
500
477
400
422
487
```

Причина:
Thread 1 : read the value, get 0, add 1, so value = 1
Thread 2 : read the value, get 0, add 1, so value = 1
Thread 1 : write 1 to the field value and return 1
Thread 2 : write 1 to the field value and return 1

Мутексы.

```
struct Counter {
    mutex mutex;
    int value;

    Counter() : value(0) {}

    void increment(){
        mutex.lock();
        ++value;
        mutex.unlock();
    }
};
```

- Внутри блока std::mutex.lock(); ... std::mutex.unlock(); может находиться только один поток, остальные будут ожидать.
- NB: для синхронизации одной переменной примитивного типа хватило бы std::atomic, а вот для линейного списка mutex бы подошел.
- std::lock_guard — RAII обертка для mutex (типа scoped_ptr)

Множественное наследование

Множественное наследование (multiple inheritance) — возможность наследовать сразу несколько классов.

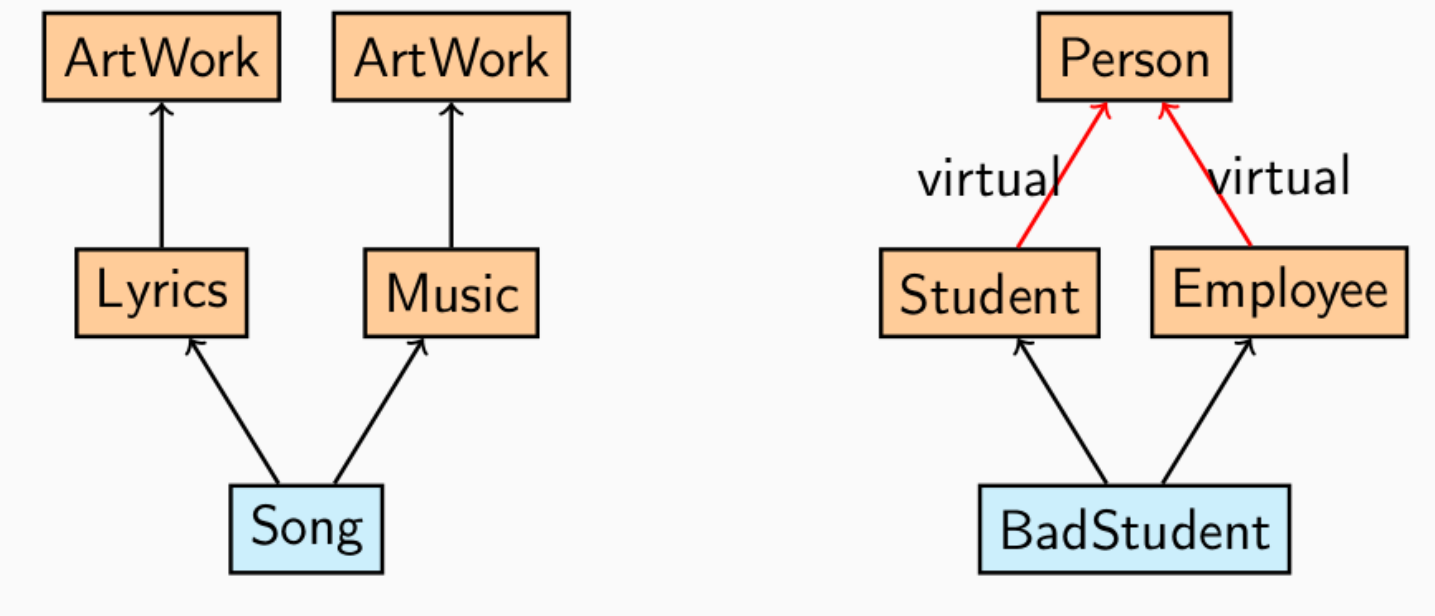
```
struct Student {
    string name()       const { return name_; }
    string university() const { return university_; }
private:
    string name_, university_;
};
struct FullTimeEmployee {
    string name()    const { return name_; }
    string company() const { return company_; }
private:
    string name_, company_;
};
struct BadStudent: Student, FullTimeEmployee {
    string name() const { return Student::name(); }
};
```

Два экземпляра name_!

Интерфейсы

```
struct Person {
    string name() const { return name_; }
    string name_;
};
struct IStudent {
    virtual string name()       const = 0;
    virtual string university() const = 0;
    virtual ~IStudent() {}
};
struct IFullTimeEmployee {
    virtual string name()    const = 0;
    virtual string company() const = 0;
    virtual ~IFullTimeEmployee() {}
};
struct BadStudent: Person, IStudent, IFullTimeEmployee {
    string name()       const { return Person::name(); }
    string university() const { return university_; }
    string company()    const { return company_; }
    string university_, company_;
};
```

Виртуальное наследование




```
struct Person {};
struct Student   : virtual Person {};
struct Employee  : virtual Person {};
struct BadStudent : Student, Employee {};
```

Виртуальное наследование: конструктор

Кто вызывает конструктор базового класса?

```
struct Person {
    explicit Person(string const& name)
        : name_(name) {}
    string name_;
};
struct Student  : virtual Person {
    explicit Student(string const& name) : Person(name) {}
};
struct Employee : virtual Person {
    explicit Employee(string const& name) : Person(name) {}
};
struct BadStudent : Student, Employee {
    explicit BadStudent(string const& name)
        : Person(name), Student(name), Employee(name)
    {}
};
```

Конструктор "дедушки" вызовется из BadStudent.