



ОНЛАЙН-ОБРАЗОВАНИЕ

Онлайн-образование



Меня хорошо видно && слышно?

Ставьте +, если все хорошо
Напишите в чат, если есть проблемы

Проверить, идет ли запись!





Проблемы многопоточности

Павел Филонов

Маршрут вебинара

Состояние гонки



Мьютексы



Взаимные блокировки



Выводы

The background of the slide is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this image is a semi-transparent blue band across the center, which contains a white network or molecular structure pattern. The pattern consists of numerous small dots connected by thin white lines, creating a complex web of triangles and polygons.

Состояние гонки

Задание. Продажа билетов

- 1 Скачайте исходники примеров из материалов вебинара
- 2 Соберите все примеры
`cmake -B build`
`cmake --build build`
- 3 Запустите пример `ex1_tickets.cpp` с параметром 10
`./build/ex1_tickets 10`
и скопируйте результаты в чат

Состояние гонки

Состояние гонки (race condition) - ошибка проектирование многопоточной системы, при которой работа системы зависит от того в каком порядке выполняются части кода.

Поток 1

```
while (tickets_left > 0) {  
  
    usefull_work();  
    --tickets_left;  
    usefull_work();  
}
```

Поток 2

```
while (tickets_left > 0) {  
  
    usefull_work();  
    --tickets_left;  
    usefull_work();  
}
```


Реальный пример состояния гонок

Therac-25 - аппарат лучевой терапии запущенный в серию в 1982 году

Программная ошибка связанная с состоянием гонок привела к летальным случаям



Ищем гонки с помощью ThreadSanitizer

- Включаем в gcc или clang `-fsanitize=thread`
- Работает только под Linux и MacOS x86_64 (ждем Windows)
- Замедляет выполнение в 5-15 раз
- Увеличивает потребление памяти в 5-10 раз
- Предназначен только для поиска data races

The background of the entire image is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. A semi-transparent blue overlay covers the entire image. In the center, there is a network pattern of white lines connecting various points, resembling a data or communication network. The word "Мьютексы" is written in a large, white, sans-serif font across the middle of the image.

Мьютексы

std::mutex

Мьютекс (mutual exclusion - взаимное исключение) - примитив синхронизации, обеспечивающий взаимное исключение выполнения критических участков кода (критических секций)

```
class mutex {  
public:  
    mutex (const mutex&) = delete;  
    mutex& operator=(const mutex&) = delete;  
  
    void lock();    // захватить мьютекс  
    void unlock(); // освободить мьютекс  
  
    bool try_lock();  
};
```

Задание. Продажа билетов

- 1 Возьмите за основу пример `ex1_ticket.mutex.cpp`
- 2 Добавьте в правильное место захват и освобождение мьютекса
- 3 Запустите пример `ex1_tickets.mutex.cpp` с параметром 10
`./build/ex1_tickets.mutex 10`
и скопируйте результаты в чат

Основные правила работы с мьютексами

- только один поток выполняет код в критической секции
- важно не выполнять в критической секции долгих вычислений
- при попытке захватить мьютекс дважды в одном потоке произойдет блокировка
 - когда это необходимо , можно использовать `std::recursive_mutex`
- важно отпустить мьютекс в любом случае, в том числе при генерации исключения
 - можно использовать `std::lock_guard` для мьютекса, который реализует RAII идиому

The background of the image is a high-angle, blue-tinted aerial photograph of a dense urban skyline, likely New York City. Overlaid on this is a semi-transparent blue band across the middle, which contains a white network diagram of interconnected nodes and lines. Centered within this band is the title text.

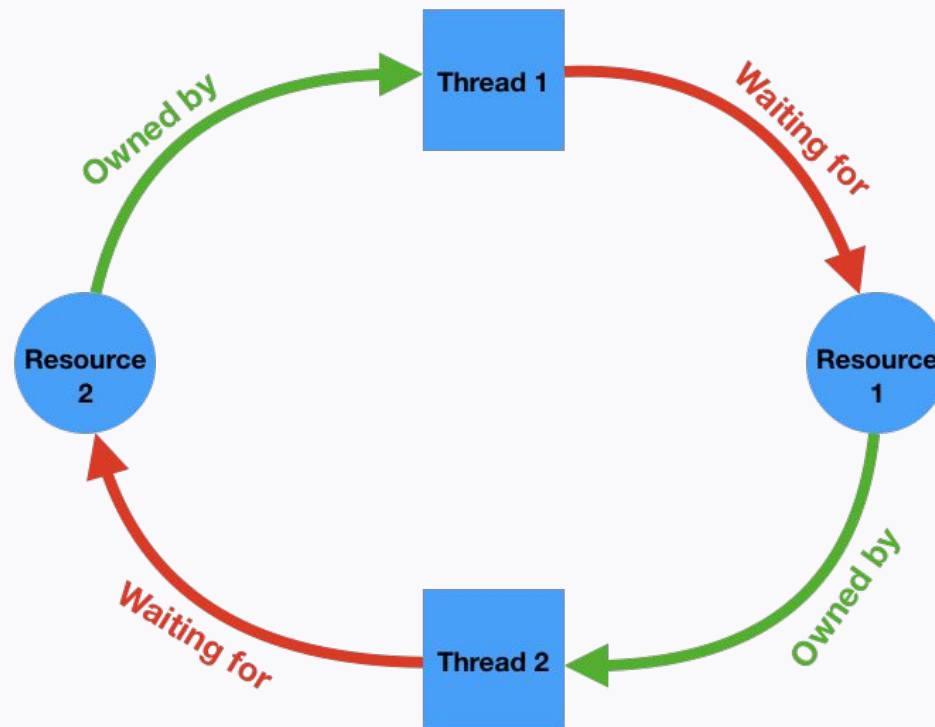
Взаимная блокировка

Задание. Банковские транзакции

- 1 Запустите пример `ex2_bank` с параметром 10
`./build/ex2_bank 10`
- 2 Запускайте несколько раз пока он не зависнем
- 3 Напишите в чат с какой попытки произошло зависание

Взаимная блокировка

Взаимная блокировка (deadlock) - ситуация в многопоточном программировании, когда несколько потоков ждут ресурсов, занятых друг другом и ни один не может продолжить выполнение.



Ищем deadlock с помощью отладчика

- LLDB, GDB или другие
- Изучаем стек вызовов различных потоков
- Ищем блокировки на ожидании мьютексов
- Выделяем взаимные блокировки

Гарантированный порядок блокировки

```
friend bool transaction(sync_account &from, sync_account &to, int amount) {  
    std::lock(from.mutex_, to.mutex_);  
    std::lock_guard<std::mutex> lock1(from.mutex_, std::adopt_lock);  
    std::lock_guard<std::mutex> lock2(to.mutex_, std::adopt_lock);  
    if (amount > from.balance_) {  
        return false;  
    }  
    from.balance_ -= amount;  
    to.balance_ += amount;  
  
    return true;  
}
```


Гарантированный порядок блокировки (C++17)

```
friend bool transaction(sync_account &from, sync_account &to, int amount) {  
    std::scoped_lock lock(from.mutex_, to.mutex_);  
    if (amount > from.balance_) {  
        return false;  
    }  
    from.balance_ -= amount;  
    to.balance_ += amount;  
  
    return true;  
}
```




Выводы

Итоги - тезисы

- 1 Познакомились с состоянием гонки
- 2 Узнали как избежать состояние гонки с помощью мьютексов
- 3 Научились использовать `std::lock_guard`
- 4 Увидели пример deadlock
- 5 Научились избегать простых deadlock с помощью `std::lock` или `std::scoped_lock`




Рефлексия



С какими основными мыслями и инсайтами уходите с вебинара



Каких целей вебинара не удалось достичь

The background of the image is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a white geometric network pattern of dots and lines. The text is centered within this blue layer.

Заполните, пожалуйста,
опрос о занятии по ссылке в чате



Спасибо за внимание!
Приходите на следующие вебинары

Павел Филонов

Бонус. Проблема обедающих философов

Алгоритм работы философа:

- Размышлять, пока не освободится левая вилка.
Когда вилка освободится — взять её.
- Размышлять, пока не освободится правая вилка.
Когда вилка освободится — взять её.
- Есть
- Положить левую вилку
- Положить правую вилку
- Повторить алгоритм сначала

