

!concurrent, worldHello

или

многопоточность глазами VM-инженера

Александр Филатов
filatovaur@gmail.com

<https://github.com/Svazars/snowone-2024-vm-engineer-concurrency>

Bio: Александр Филатов

Уже 8 лет как VM-инженер¹

- 2015 - 2019, Excelsior JVM with AOT compilation²
- 2019 - now, Huawei, Languages and Compilers lab³

¹ Иван Углынский, Один день из жизни JVM-инженера, <https://habr.com/ru/company/jugru/blog/719614/>

² <https://habr.com/ru/company/jugru/blog/437180/>

³ <http://rnew.tilda.ws/excelsiorathuawei>

Bio: Александр Филатов

Уже 8 лет как VM-инженер¹

- 2015 - 2019, Excelsior JVM with AOT compilation²
- 2019 - now, Huawei, Languages and Compilers lab³

Специализация – рантаймы виртуальных машин

Узкая специализация – сборщики мусора

¹ Иван Углынский, Один день из жизни JVM-инженера, <https://habr.com/ru/company/jugru/blog/719614/>

² <https://habr.com/ru/company/jugru/blog/437180/>

³ <http://rnew.tilda.ws/excelsiorathuawei>

Bio: Александр Филатов

Область интересов:

- автоматическое управление памятью

Bio: Александр Филатов

Область интересов:

- автоматическое управление памятью
- многопоточность

Bio: Александр Филатов

Область интересов:

- автоматическое управление памятью
- многопоточность
- слабые модели памяти

Bio: Александр Филатов

Область интересов:

- автоматическое управление памятью
- многопоточность
- слабые модели памяти
- корректность многопоточных структур данных

Bio: Александр Филатов

Область интересов:

- автоматическое управление памятью
- многопоточность
- слабые модели памяти
- корректность многопоточных структур данных

Персональное когнитивное искажение: много страдал, отлаживая баги

- своего параллельного кода

Bio: Александр Филатов

Область интересов:

- автоматическое управление памятью
- многопоточность
- слабые модели памяти
- корректность многопоточных структур данных

Персональное когнитивное искажение: много страдал, отлаживая баги

- своего параллельного кода
- чужих реализаций многопоточных структур данных

Bio: Александр Филатов

Область интересов:

- автоматическое управление памятью
- многопоточность
- слабые модели памяти
- корректность многопоточных структур данных

Персональное когнитивное искажение: много страдал, отлаживая баги

- своего параллельного кода
- чужих реализаций многопоточных структур данных
- компилятора

Bio: Александр Филатов

Область интересов:

- автоматическое управление памятью
- многопоточность
- слабые модели памяти
- корректность многопоточных структур данных

Персональное когнитивное искажение: много страдал, отлаживая баги

- своего параллельного кода
- чужих реализаций многопоточных структур данных
- компилятора

Опасаюсь нового. Не люблю поддерживать старое.

Как было получено название доклада

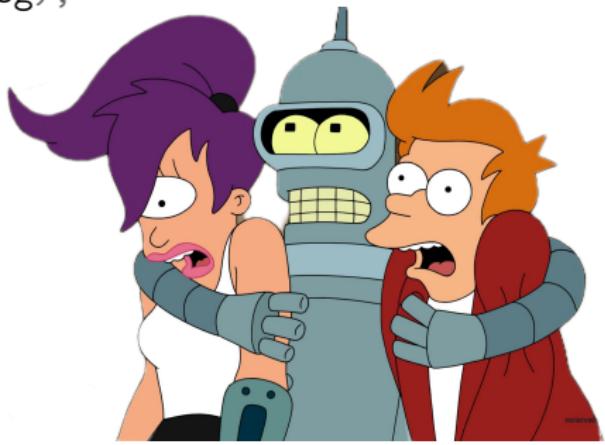
!concurrent,worldHello

```
for (String msg : new String[] {  
    "Hello", ", ", "concurrent", "world", "!"  
}) {  
    new Thread() {  
        public void run() {  
            System.out.println(msg);  
        }  
    }.start();  
}
```

Как было получено название доклада

!concurrent,worldHello

```
for (String msg : new String[] {  
    "Hello", ", ", "concurrent", "world", "!"  
}) {  
    new Thread() {  
        public void run() {  
            System.out.println(msg);  
        }  
    }.start();  
}
```



План выступления

- 1 Знакомство
- 2 Как я вижу программистов
- 3 Как я вижу системных программистов
- 4 Как я вижу компиляторы
- 5 Как я вижу процессоры
- 6 Как я вижу языки программирования
- 7 Подведение итогов

Вы находитесь здесь

1 Знакомство

2 Как я вижу программистов

3 Как я вижу системных программистов

4 Как я вижу компиляторы

5 Как я вижу процессоры

6 Как я вижу языки программирования

7 Подведение итогов

Идея №1: давай распараллелим!

Обычно бывает так

Коллега предлагает переписать часть проекта с использованием нескольких потоков

Идея №1: давай распараллелим!

Обычно бывает так

Коллега предлагает переписать часть проекта с использованием нескольких потоков



Идея №1: давай распараллелим!

Обычно бывает так

Коллега предлагает переписать часть проекта с использованием нескольких потоков

Проходит время ...



Идея №1: давай распараллелим!

Обычно бывает так

Коллега предлагает переписать часть проекта с использованием нескольких потоков

Проходит время ...

- Стало медленнее



Идея №1: давай распараллелим!

Обычно бывает так

Коллега предлагает переписать часть проекта с использованием нескольких потоков

Проходит время ...

- Стало медленнее
- Зависает



Идея №1: давай распараллелим!

Обычно бывает так

Коллега предлагает переписать часть проекта с использованием нескольких потоков

Проходит время ...

- Стало медленнее
- Зависает
- Код непонятный



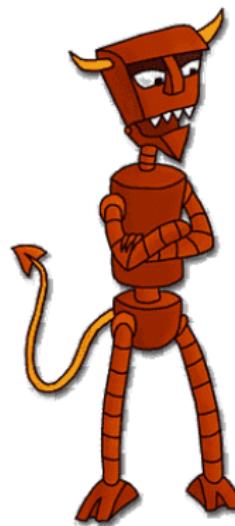
Идея №1: давай распараллелим!

Обычно бывает так

Коллега предлагает переписать часть проекта с использованием нескольких потоков

Проходит время ...

- Стало медленнее
- Зависает
- Код непонятный



Идея №1: давай распараллелим!

Почему так происходит?

Идея №1: давай распараллелим!

Почему так происходит?

```
synchronized(objectA) {  
    synchronized(objectB) {  
        computeStuff();  
    }  
}
```

```
synchronized(objectB) {  
    synchronized(objectA) {  
        computeStuff();  
    }  
}
```

Идея №1: давай распараллелим!

Почему так происходит?

```
synchronized(objectA) {  
    synchronized(objectB) {  
        computeStuff();  
    }  
}
```

```
synchronized(objectB) {  
    synchronized(objectA) {  
        computeStuff();  
    }  
}
```

Легко сломать многопоточную систему неаккуратным изменением.

Идея №1: давай распараллелим!

Почему так происходит?

```
synchronized(objectA) {  
    synchronized(objectB) {  
        computeStuff();  
    }  
}
```

```
synchronized(objectB) {  
    synchronized(objectA) {  
        computeStuff();  
    }  
}
```

Легко сломать многопоточную систему неаккуратным изменением.
Это ВСЕГДА случается после написания оригинального кода.

Идея №1: давай распараллелим!

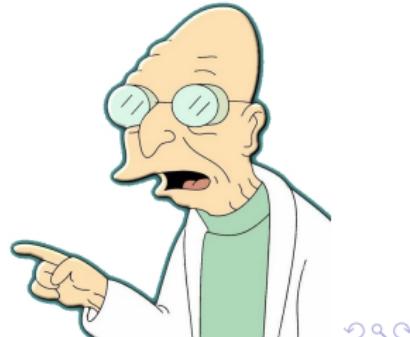
Почему так происходит?

```
synchronized(objectA) {  
    synchronized(objectB) {  
        computeStuff();  
    }  
}
```

```
synchronized(objectB) {  
    synchronized(objectA) {  
        computeStuff();  
    }  
}
```

Легко сломать многопоточную систему неаккуратным изменением.
Это ВСЕГДА случается после написания оригинального кода.

KEEP IT SIMPLE STUPID



Идея №1: давай распараллелим!

Выводы

Параллелизм полезен, но многопоточность приносит с собой:

- Риски
- Издержки на поддержку

Идея №1: давай распараллелим!

Выводы

Параллелизм полезен, но многопоточность приносит с собой:

- Риски
- Издержки на поддержку

Реакция VM-инженера: изыди! Не хочу отлаживать этот код через полгода, после серии несвязанных правок, когда ты уже уволишься.

Идея №1: давай распараллелим!

Выводы

Параллелизм полезен, но многопоточность приносит с собой:

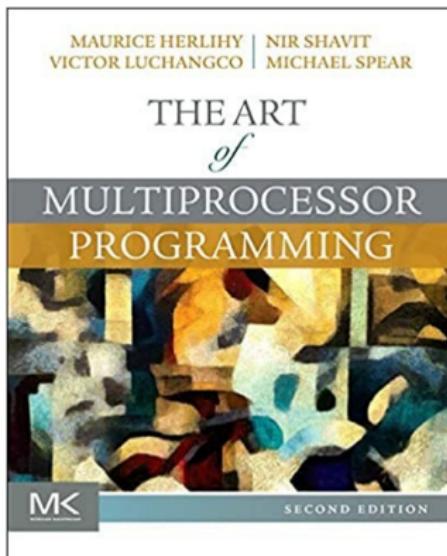
- Риски
- Издержки на поддержку

Реакция VM-инженера: изыди! Не хочу отлаживать этот код через полгода, после серии несвязанных правок, когда ты уже уволишься.
Реакция здорового человека: давай подумаем, стоит ли оно того.

Идея №2: давай по книжке сделаем!

Взгляд VM-инженера

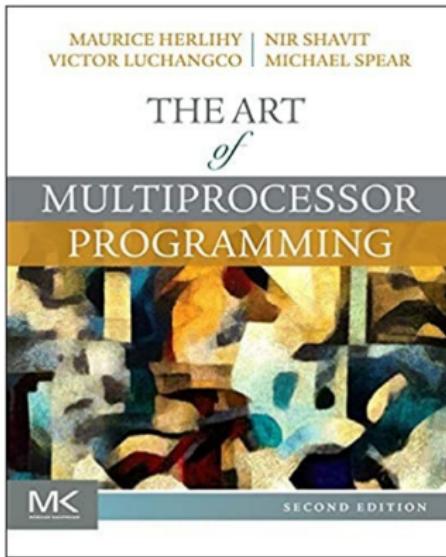
Реальность



Идея №2: давай по книжке сделаем!

Взгляд VM-инженера

Реальность



Мои ощущения



Идея №2: давай по книжке сделаем!

KSUH lock

“A Fair Fast Scalable Reader-Writer Lock”, 1993

- Orran Krieger, Michael Stumm, Ron Unrau, Jonathan Hanna
- In Proc. of the International Conference on Parallel Processing

Идея №2: давай по книжке сделаем!

KSUH lock

“A Fair Fast Scalable Reader-Writer Lock”, 1993

- Orran Krieger, Michael Stumm, Ron Unrau, Jonathan Hanna
- In Proc. of the International Conference on Parallel Processing
- реализовано на языке Си

Идея №2: давай по книжке сделаем!

KSUH lock

“A Fair Fast Scalable Reader-Writer Lock”, 1993

- Orran Krieger, Michael Stumm, Ron Unrau, Jonathan Hanna
- In Proc. of the International Conference on Parallel Processing
- реализовано на языке Си
- алгоритм формализован на языке Promela и верифицирован инструментом SPIN

Идея №2: давай по книжке сделаем!

KSUH lock

“A Fair Fast Scalable Reader-Writer Lock”, 1993

- Orran Krieger, Michael Stumm, Ron Unrau, Jonathan Hanna
- In Proc. of the International Conference on Parallel Processing
- реализовано на языке Си
- алгоритм формализован на языке Promela и верифицирован инструментом SPIN

“Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm”, 2013

Идея №2: давай по книжке сделаем!

KSUH lock

“A Fair Fast Scalable Reader-Writer Lock”, 1993

- Orran Krieger, Michael Stumm, Ron Unrau, Jonathan Hanna
- In Proc. of the International Conference on Parallel Processing
- реализовано на языке Си
- алгоритм формализован на языке Promela и верифицирован инструментом SPIN

“Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm”, 2013

- 20 лет спустя!

Идея №2: давай по книжке сделаем!

KSUH lock

“A Fair Fast Scalable Reader-Writer Lock”, 1993

- Orran Krieger, Michael Stumm, Ron Unrau, Jonathan Hanna
- In Proc. of the International Conference on Parallel Processing
- реализовано на языке Си
- алгоритм формализован на языке Promela и верифицирован инструментом SPIN

“Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm”, 2013

- 20 лет спустя!
- Обнаружена ошибка в алгоритме, связанная с управлением памятью. Проявлялась на реальной системе.

Идея №2: давай по книжке сделаем!

KSUH lock

“A Fair Fast Scalable Reader-Writer Lock”, 1993

- Orran Krieger, Michael Stumm, Ron Unrau, Jonathan Hanna
- In Proc. of the International Conference on Parallel Processing
- реализовано на языке Си
- алгоритм формализован на языке Promela и верифицирован⁴ инструментом SPIN

“Using Hardware Transactional Memory to Correct and Simplify and Readers-writer Lock Algorithm”, 2013

- 20 лет спустя!
- Обнаружена ошибка в алгоритме, связанная с управлением памятью. Проявлялась на реальной системе.

⁴частично

Идея №2: давай по книжке сделаем!

Выводы

Сложные алгоритмы нужны, но приносят с собой:

- Риски
- Издержки на поддержку

Идея №2: давай по книжке сделаем!

Выводы

Сложные алгоритмы нужны, но приносят с собой:

- Риски
- Издержки на поддержку

Реакция VM-инженера: ты точно понимаешь, что написано в книжке и почему оно будет работать?

Идея №2: давай по книжке сделаем!

Выводы

Сложные алгоритмы нужны, но приносят с собой:

- Риски
- Издержки на поддержку

Реакция VM-инженера: ты точно понимаешь, что написано в книжке и почему оно будет работать?

Реакция здорового человека: неудобно, если для редактирования кода требуется кандидатская степень по распределенным системам.

Идея №2: давай по книжке сделаем!

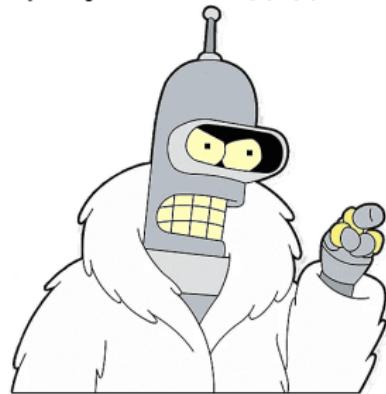
Выводы

Сложные алгоритмы нужны, но приносят с собой:

- Риски
- Издержки на поддержку

Реакция VM-инженера: ты точно понимаешь, что написано в книжке и почему оно будет работать?

Реакция здорового человека: неудобно, если для редактирования кода требуется кандидатская степень по распределенным системам.



Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.

Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.



Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.



Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.

Пример из Java-мира: biased locking.

Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.

Пример из Java-мира: biased locking.

Хитрая штука, чтобы ваши synchronized работали очень быстро⁵.

```
while (0 != count--) {  
    synchronized (jvmLock) {  
        ++counter;  
    }  
}
```

⁵ <https://mechanical-sympathy.blogspot.com/2011/11/biased-locking-osr-and-benchmarking-fun.html>

Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.

Пример из Java-мира: biased locking.

Хитрая штука, чтобы ваши synchronized работали очень быстро⁵.

```
while (0 != count--) {
    synchronized (jvmLock) {
        ++counter;
    }
}
```

| Sandy Bridge 2.0GHz - Ops/Sec | | | |
|-------------------------------|-------------------|--------------------|---------------|
| Threads | -UseBiasedLocking | +UseBiasedLocking | ReentrantLock |
| 1 | 34,500,407 | 396,511,324 | 43,148,808 |

⁵ <https://mechanical-sympathy.blogspot.com/2011/11/biased-locking-osr-and-benchmarking-fun.html>

Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.

Пример из Java-мира: biased locking.

- Опубликован в 2006 году⁶

⁶ <https://dl.acm.org/doi/10.1145/1167473.1167496>

Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.

Пример из Java-мира: biased locking.

- Опубликован в 2006 году⁶
- Служил верой и правдой

⁶ <https://dl.acm.org/doi/10.1145/1167473.1167496>

Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.

Пример из Java-мира: biased locking.

- Опубликован в 2006 году⁶
- Служил верой и правдой
- "Costly to maintain"

⁶ <https://dl.acm.org/doi/10.1145/1167473.1167496>

Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.

Пример из Java-мира: biased locking.

- Опубликован в 2006 году⁶
- Служил верой и правдой
- "Costly to maintain"
- Deprecated в Java 15 (JEP 374, 2019)

⁶ <https://dl.acm.org/doi/10.1145/1167473.1167496>

Идея №2: давай по книжке сделаем!

Но если ты эксперт, то тогда можно?

Промышленные системы быстры благодаря хитрым решениям.

Пример из Java-мира: biased locking.

- Опубликован в 2006 году⁶
- Служил верой и правдой
- "Costly to maintain"
- Deprecated в Java 15 (JEP 374, 2019)



⁶ <https://dl.acm.org/doi/10.1145/1167473.1167496>

Идея №3: давай возьмем готовое!

Хоть кто-то в состоянии написать сложную систему безошибочно?

Идея №3: давай возьмем готовое!

Хоть кто-то в состоянии написать сложную систему безошибочно?

Linux futex_wait() bug⁷

⁷ <https://groups.google.com/g/mechanical-sympathy/c/QbampZxp6C64/m/BonaHiVbEmsJ>

Идея №3: давай возьмем готовое!

Хоть кто-то в состоянии написать сложную систему безошибочно?

Linux futex_wait() bug⁷

Gil Tene (CEO of Azul, co-author of C4 GC):

We had this one bite us hard and scare the %\$^ out of us, so I figured I'd share the fear.

⁷ <https://groups.google.com/g/mechanical-sympathy/c/QbampZxp6C64/m/BonaHiVbEmsJ>

Идея №3: давай возьмем готовое!

Хочь кто-то в состоянии написать сложную систему безошибочно?

Linux `futex_wait()` bug⁷

Gil Tene (CEO of Azul, co-author of C4 GC):

We had this one bite us hard and scare the %\$^ out of us, so I figured I'd share the fear.

The linux `futex_wait` call has been broken for about a year.

⁷ <https://groups.google.com/g/mechanical-sympathy/c/QbampZxp6C64/m/BonaHiVbEmsJ>

Идея №3: давай возьмем готовое!

Хоть кто-то в состоянии написать сложную систему безошибочно?

Linux `futex_wait()` bug⁷

Gil Tene (CEO of Azul, co-author of C4 GC):

We had this one bite us hard and scare the %\$^ out of us, so I figured I'd share the fear.

The linux `futex_wait` call has been broken for about a year.

The impact of this kernel bug is very simple: user processes can deadlock and hang in seemingly impossible situations. `Thread.park()` in Java may stay parked.

⁷ <https://groups.google.com/g/mechanical-sympathy/c/QbmpZxp6C64/m/BonaHiVbEmsJ>

Идея №3: давай возьмем готовое!

Хочь кто-то в состоянии написать сложную систему безошибочно?

Linux `futex_wait()` bug⁷

Gil Tene (CEO of Azul, co-author of C4 GC):

We had this one bite us hard and scare the %\$^ out of us, so I figured I'd share the fear.

The linux `futex_wait` call has been broken for about a year.

The impact of this kernel bug is very simple: user processes can deadlock and hang in seemingly impossible situations. `Thread.park()` in Java may stay parked.

You'll spend a couple of months of someone's time trying to find the fault in your code, when there is nothing there to find.

⁷ <https://groups.google.com/g/mechanical-sympathy/c/QbmpZxp6C64/m/BonaHiVbEmsJ>

Идея №3: давай возьмем готовое!

Хоть кто-то в состоянии написать сложную систему безошибочно?



imgflip.com

Идея №3: давай возьмем готовое!

Выводы

- Выбирайте проверенные технологии

Идея №3: давай возьмем готовое!

Выводы

- Выбирайте проверенные технологии
- Регулярно обновляйтесь

Идея №3: давай возьмем готовое!

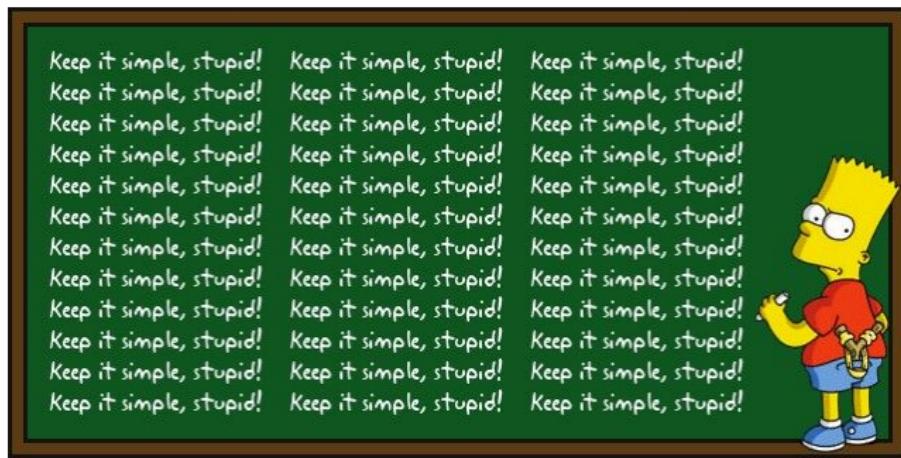
Выводы

- Выбирайте проверенные технологии
- Регулярно обновляйтесь
- Можете не использовать хитрые алгоритмы – не используйте

Идея №3: давай возьмем готовое!

Выводы

- Выбирайте проверенные технологии
- Регулярно обновляйтесь
- Можете не использовать хитрые алгоритмы – не используйте



Как страшно жить

Жизнерадостный коллега



Почему так?

Почему так?

Личный опыт

Человеческий мозг не предназначен для многопоточности.

Почему так?

Личный опыт

Человеческий мозг не предназначен для многопоточности.
Сложно представлять одновременное исполнение потоков и все возможные пересечения.

Почему так?

Личный опыт

Человеческий мозг не предназначен для многопоточности.
Сложно представлять одновременное исполнение потоков и все возможные пересечения.



Неконкретные советы

Используйте

- Простые решения

Неконкретные советы

Используйте

- Простые решения
- Готовые библиотеки для сложных стандартных задач

Неконкретные советы

Используйте

- Простые решения
- Готовые библиотеки для сложных стандартных задач
- Шаблоны проектирования многопоточных систем

Неконкретные советы

Используйте

- Простые решения
- Готовые библиотеки для сложных стандартных задач
- Шаблоны проектирования многопоточных систем
- Предметно-ориентированные языки

Неконкретные советы

Используйте

- Простые решения
- Готовые библиотеки для сложных стандартных задач
- Шаблоны проектирования многопоточных систем
- Предметно-ориентированные языки

А если меня беспокоит производительность?



Вы находитесь здесь

- 1 Знакомство
- 2 Как я вижу программистов
- 3 Как я вижу системных программистов
- 4 Как я вижу компиляторы
- 5 Как я вижу процессоры
- 6 Как я вижу языки программирования
- 7 Подведение итогов

Ленивая инициализация

Простой случай

```
public class LazyInit {  
  
    private Object heavyToInitialize = null;  
  
    /**  
     * Initializes resource lazily, on first `getResource`.  
     * `init` should be invoked only once.  
     */  
    Object getResource() {  
        if (heavyToInitialize == null) {  
            heavyToInitialize = init();  
        }  
        return heavyToInitialize;  
    }  
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {
        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {
        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {
        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {
        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Ленивая инициализация

Многопоточность

```
private Object heavyToInitialize;

// thread 1
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

```
// thread 2
Object getResource() {
    if (heavyToInitialize == null) {

        heavyToInitialize = init();
    }
    return heavyToInitialize;
}
```

Многопоточная ленивая инициализация

Double checked locking

```
private volatile Object heavyToInitialize = null;  
Object getResource() {  
    if (heavyToInitialize == null) {  
        synchronized(this) {  
            if (heavyToInitialize == null) {  
                heavyToInitialize = init();  
            }  
        }  
    }  
    return heavyToInitialize;  
}
```

Многопоточная ленивая инициализация

Double checked locking

```
private volatile Object heavyToInitialize = null;  
Object getResource() {  
    if (heavyToInitialize == null) {  
        synchronized(this) {  
            if (heavyToInitialize == null) {  
                heavyToInitialize = init();  
            }  
        }  
    }  
    return heavyToInitialize;  
}
```



Многопоточная ленивая инициализация

Double checked locking

```
private volatile Object heavyToInitialize = null;  
Object getResource() {  
    if (heavyToInitialize == null) {  
        synchronized(this) {  
            if (heavyToInitialize == null) {  
                heavyToInitialize = init();  
            }  
        }  
    }  
    initialize;  
    return heavyToInitialize;  
}
```



initialize;



Многопоточная ленивая инициализация

А почему так медленно?

- Приходит системный программист.

Многопоточная ленивая инициализация

А почему так медленно?

- Приходит системный программист.
- Смотрит на реализацию стандартного шаблона программирования.

Многопоточная ленивая инициализация

А почему так медленно?

- Приходит системный программист.
- Смотрит на реализацию стандартного шаблона программирования.
- Считает, что можно лучше.

Многопоточная ленивая инициализация

Trampolines: idea

```
void* getResource() {
    if (heavyToInitialize == NULL) {
        heavyToInitialize = create();
    }
    return heavyToInitialize;
}
```

Многопоточная ленивая инициализация

Trampolines: idea

```
void* getResource() {
    if (heavyToInitialize == NULL) {
        heavyToInitialize = create();
    }
    return heavyToInitialize;
}

getResource: mov      rax, qword ptr [heavyToInitialize]
            test     rax, rax
            je       .LBB0_1
            ret
.LBB0_1:   ...
            call     create
            mov      qword ptr [heavyToInitialize], rax
            ...
```

Многопоточная ленивая инициализация

Trampolines: idea

```
void* getResource() {
    if (heavyToInitialize == NULL) {
        heavyToInitialize = create();
    }
    return heavyToInitialize;
}

getResource: mov      rax, qword ptr [heavyToInitialize]
            test     rax, rax
            je       .LBB0_1
            ret
.LBB0_1:
            ...
            call     create
            mov      qword ptr [heavyToInitialize], rax
            ...
```

Когда инициализация завершилась, не нужно проверять rax на NULL.

Многопоточная ленивая инициализация

Trampolines

До инициализации:

```
getResource: mov      rax, qword ptr [heavyToInitialize]
            test     rax, rax
            je       .LBB0_1
            ret
.LBB0_1:   ...
            call     create
            mov      qword ptr [heavyToInitialize], rax
            ...
```

Многопоточная ленивая инициализация

Trampolines

Когда проинициализировали, требуется замена выделенного участка:

```
getResource: mov      rax, qword ptr [heavyToInitialize]
            test     rax, rax ; <<
            je       .LBB0_1 ; <<
            ret      ; <<
.LBB0_1:   ...
            call     create
            mov      qword ptr [heavyToInitialize], rax
            ...
```

Многопоточная ленивая инициализация

Trampolines

После инициализации:

```
getResource: mov      rax, qword ptr [heavyToInitialize]
            ret      ; test      rax, rax
            nop      ; je       .LBB0_1
            nop      ; ret
.LBB0_1:   ...
            call     create
            mov      qword ptr [heavyToInitialize], rax
            ...
```

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.



Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Используется в ядре⁸:

⁸ <https://lwn.net/Articles/484687/>

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Используется в ядре⁸:

- boot-time kernel configuration

⁸ <https://lwn.net/Articles/484687/>

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Используется в ядре⁸:

- boot-time kernel configuration
- run-time profiling/monitoring

⁸ <https://lwn.net/Articles/484687/>

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Используется в ядре⁸:

- boot-time kernel configuration
- run-time profiling/monitoring

Недостатки:

⁸ <https://lwn.net/Articles/484687/>

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Используется в ядре⁸:

- boot-time kernel configuration
- run-time profiling/monitoring

Недостатки:

- Security

⁸ <https://lwn.net/Articles/484687/>

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Используется в ядре⁸:

- boot-time kernel configuration
- run-time profiling/monitoring

Недостатки:

- Security
- Portability

⁸ <https://lwn.net/Articles/484687/>

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Используется в ядре⁸:

- boot-time kernel configuration
- run-time profiling/monitoring

Недостатки:

- Security
- Portability
- Хакеризм a.k.a. концептуальная сложность

⁸ <https://lwn.net/Articles/484687/>

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Используется в ядре⁸:

- boot-time kernel configuration
- run-time profiling/monitoring

Недостатки:

- Security
- Portability
- Хакеризм a.k.a. концептуальная сложность
- Data race еще фатальнее

⁸ <https://lwn.net/Articles/484687/>

Многопоточная ленивая инициализация

Trampolines: use cases

Очень элегантное и производительное решение.

Используется в ядре⁸:

- boot-time kernel configuration
- run-time profiling/monitoring

Недостатки:

- Security
- Portability
- Хакеризм a.k.a. концептуальная сложность
- Data race еще фатальнее
- Вы так не можете!

⁸ <https://lwn.net/Articles/484687/>

Многопоточная ленивая инициализация

Just-In-Time constants: idea

Фундамент ОС малоподвижен:

- Ядро уже скомпилировано

Многопоточная ленивая инициализация

Just-In-Time constants: idea

Фундамент ОС малоподвижен:

- Ядро уже скомпилировано
- Ему сложно изменять свой код

Многопоточная ленивая инициализация

Just-In-Time constants: idea

Фундамент ОС малоподвижен:

- Ядро уже скомпилировано
- Ему сложно изменять свой код

Прикладные программы могут быть хитре:

Многопоточная ленивая инициализация

Just-In-Time constants: idea

Фундамент ОС малоподвижен:

- Ядро уже скомпилировано
- Ему сложно изменять свой код

Прикладные программы могут быть хитре:

- В первый раз функция интерпретируется

Многопоточная ленивая инициализация

Just-In-Time constants: idea

Фундамент ОС малоподвижен:

- Ядро уже скомпилировано
- Ему сложно изменять свой код

Прикладные программы могут быть хитре:

- В первый раз функция интерпретируется
- Во второй – компилируется

Многопоточная ленивая инициализация

Just-In-Time constants: idea

Фундамент ОС малоподвижен:

- Ядро уже скомпилировано
- Ему сложно изменять свой код

Прикладные программы могут быть хитре:

- В первый раз функция интерпретируется
- Во второй – компилируется
- Just-in-time compilation, JIT

Многопоточная ленивая инициализация

Just-In-Time constants: example

```
Object resource = new Object();
Object[] array = new Object[] { resource };
Object getResource(int index) {
    if (array == null) throw new NullPointerException();
    if (index < 0 || index >= array.length) throw new OutOfBounds();
    return array[index];
}
```

Многопоточная ленивая инициализация

Just-In-Time constants: example

```
Object resource = new Object();
Object[] array = new Object[] { resource };
Object getResource(int index) {
    if (array == null) throw new NullPointerException();
    if (index < 0 || index >= array.length) throw new OutOfBounds();
    return array[index];
}
```

Допустим, виртуальная машина видит, что

- в поле array больше ничего не присваивают
- содержимое array не изменяется

Многопоточная ленивая инициализация

Just-In-Time constants: example

```
Object resource = new Object();
Object[] array = new Object[] { resource };
Object getResource(int index) {
    if (array == null) throw new NullPointerException();
    if (index < 0 || index >= array.length) throw new OutOfBounds();
    return array[index];
}
```

Допустим, виртуальная машина видит, что

- в поле array больше ничего не присваивают
- содержимое array не изменяется

Тогда можно:

- переместить массив в память по адресу 0x7ffdcaa1c908
- сгенерировать специализированный код

Многопоточная ленивая инициализация

Just-In-Time constants: example

```
Object resource = new Object();
Object[] array = new Object[] { resource };
Object getResource(int index) {
    if (array == null) throw new NullPointerException();
    if (index < 0 || index >= array.length) throw new OutOfBounds();
    return array[index];
}
Object getResource_optimized(int index) {
    if (address(array) == 0x7ffdcaa1c908) {
        if (index == 0) {
            return someResource;
        }
    }
    uncommon_trap();
}
```

uncommon_trap – магический метод, который «вернет всё, как было».

Многопоточная ленивая инициализация

Just-In-Time constants: example

```

Object resource = new Object();
Object[] array = new Object[] { resource };
Object getResource(int index) {
    if (array == null) throw new NullPointerException();
    if (index < 0 || index >= array.length) throw new OutOfBoundsException();
    return array[index];
}
Object getResource_optimized(int index) {
    if (address(array) == 0x7ffdcaa1c908) {
        if (index == 0) {
            return someResource;
        }
    }
    uncommon_trap();
}

```



`uncommon_trap` – магический метод, который «вернет всё, как было».

Многопоточная ленивая инициализация

Выводы

- Используйте готовые реализации шаблонов программирования

Многопоточная ленивая инициализация

Выводы

- Используйте готовые реализации шаблонов программирования
- Много труда вложено, чтобы они работали верно и быстро

Многопоточная ленивая инициализация

Выводы

- Используйте готовые реализации шаблонов программирования
- Много труда вложено, чтобы они работали верно и быстро
- Узнавайте, какие трюки умеет ваш язык программирования⁹

⁹ <https://shipilev.net/jvm/anatomy-quarks/>

Многопоточная ленивая инициализация

Выводы

- Используйте готовые реализации шаблонов программирования
- Много труда вложено, чтобы они работали верно и быстро
- Узнавайте, какие трюки умеет ваш язык программирования⁹



⁹ <https://shipilev.net/jvm/anatomy-quarks/>

Многопоточная ленивая инициализация

Выводы

- Используйте готовые реализации шаблонов программирования
- Много труда вложено, чтобы они работали верно и быстро
- Узнавайте, какие трюки умеет ваш язык программирования⁹



Кажется, что системные программисты не соблюдают принцип KISS.

⁹ <https://shipilev.net/jvm/anatomy-quarks/>

Вы находитесь здесь

- 1 Знакомство
- 2 Как я вижу программистов
- 3 Как я вижу системных программистов
- 4 **Как я вижу компиляторы**
- 5 Как я вижу процессоры
- 6 Как я вижу языки программирования
- 7 Подведение итогов

Классические оптимизации однопоточного кода

Removing reads

```
static int a;
void foo_1() {
    while (true) {
        int tmp = a;
        if (tmp == 0) break;
        do_something_with(tmp);
    }
}
```

Классические оптимизации однопоточного кода

Removing reads

```
static int a;                                // "optimized" version
void foo_1() {                               void foo_2() {
    while (true) {                           int tmp = a;
        int tmp = a;                         if (tmp != 0)
        if (tmp == 0) break;                  while (true) {
            do_something_with(tmp);          do_something_with(tmp);
        }                                     }
    }                                         }
```

Имеет ли право компилятор уменьшить количество загрузок из памяти и переписать функцию?

Классические оптимизации однопоточного кода

Godbolt

```
static int a;
void foo_1() {
    while (true) {
        int tmp = a;
        if (tmp == 0) break;
        do_something_with(tmp);
    }
}
```

Классические оптимизации однопоточного кода

Godbolt

```

static int a;                      foo_1:
void foo_1() {                     push    rbx
    while (true) {                mov     ebx, [a]
        int tmp = a;              test   ebx, ebx
        if (tmp == 0) break;       je      .LBB1_2
        do_something_with(tmp);   .LBB1_1:          #<- /
    }                                mov     edi, ebx      # /
}                                call    do_something_with # /
x86-64 clang 16.0.0 -O210           jmp    .LBB1_1          #--/
x86-64 gcc 13.1 -O211             .LBB1_2:          #
                                                pop    rbx
                                                ret

```

¹⁰ <https://godbolt.org/z/99j3erzaE>

¹¹ <https://godbolt.org/z/fxzGEo1qf>

Классические оптимизации однопоточного кода

Godbolt

```
static int a;
void foo_1() {
    while (true) {
        int tmp = a;
        if (tmp == 0) break;
        do_something_with(tmp);
    }
}
```

x86-64 clang 16.0.0 -O2¹⁰
x86-64 gcc 13.1 -O2¹¹

```
foo_1:
    push    rbx
    mov     ebx, [a]
    test   ebx, ebx
    je      .LBB1_2
.LBB1_1:
    mov     edi, ebx
    call   do_some
    jmp   .LBB1_1
.LBB1_2:
    pop    rbx
    ret
```

¹⁰ <https://godbolt.org/z/99j3erzaE>

¹¹ <https://godbolt.org/z/fxzGEo1qf>

Кто виноват и что делать?

Очевидные преобразования однопоточных программ искажают поведение многопоточного кода.

Кто виноват и что делать?

Очевидные преобразования однопоточных программ искажают поведение многопоточного кода.

Приходится изучать разные решения:

Кто виноват и что делать?

Очевидные преобразования однопоточных программ искажают поведение многопоточного кода.

Приходится изучать разные решения:

- компиляторные барьеры

Кто виноват и что делать?

Очевидные преобразования однопоточных программ искажают поведение многопоточного кода.

Приходится изучать разные решения:

- компиляторные барьеры
- memory orderings (volatile, seq-cst, acquire-release...)¹²

¹² <https://snowone.ru/2024/speakers/alantsov>

Кто виноват и что делать?

Очевидные преобразования однопоточных программ искажают поведение многопоточного кода.

Приходится изучать разные решения:

- компиляторные барьеры
- memory orderings (volatile, seq-cst, acquire-release...)¹²
- примитивы синхронизации и их интеграцию с языком программирования

¹² <https://snowone.ru/2024/speakers/alantsov>

Кто виноват и что делать?

Очевидные преобразования однопоточных программ искажают поведение многопоточного кода.

Приходится изучать разные решения:

- компиляторные барьеры
- memory orderings (volatile, seq-cst, acquire-release...)¹²
- примитивы синхронизации и их интеграцию с языком программирования
- нативные отладчики и time-travel debugging

¹² <https://snowone.ru/2024/speakers/alantsov>

Кто виноват и что делать?

Очевидные преобразования однопоточных программ искажают поведение многопоточного кода.

Приходится изучать разные решения:

- компиляторные барьеры
- memory orderings (volatile, seq-cst, acquire-release...)¹²
- примитивы синхронизации и их интеграцию с языком программирования
- нативные отладчики и time-travel debugging
- узкоспециализированные инструменты (model checking, deterministic scheduling...)

¹² <https://snowone.ru/2024/speakers/alantsov>

Вы находитесь здесь

- 1 Знакомство
- 2 Как я вижу программистов
- 3 Как я вижу системных программистов
- 4 Как я вижу компиляторы
- 5 Как я вижу процессоры
- 6 Как я вижу языки программирования
- 7 Подведение итогов

Кругом враги

Не только компиляторы (software) пытаются сломать ваше представление об исполнении программы в многопоточном контексте.

Кругом враги

Не только компиляторы (*software*) пытаются сломать ваше представление об исполнении программы в многопоточном контексте. Есть еще процессор и подсистема памяти (*hardware*).

Кругом враги

Не только компиляторы (*software*) пытаются сломать ваше представление об исполнении программы в многопоточном контексте. Есть еще процессор и подсистема памяти (*hardware*). Которые умеют:

Кругом враги

Не только компиляторы (software) пытаются сломать ваше представление об исполнении программы в многопоточном контексте. Есть еще процессор и подсистема памяти (hardware). Которые умеют:

- Исполнять независимые инструкции одновременно (out-of-order execution)

Кругом враги

Не только компиляторы (*software*) пытаются сломать ваше представление об исполнении программы в многопоточном контексте. Есть еще процессор и подсистема памяти (*hardware*). Которые умеют:

- Исполнять независимые инструкции одновременно (*out-of-order execution*)
- Задействовать одни и те же ресурсы для исполнения логически независимых потоков (*hyper-threading*)

Кругом враги

Не только компиляторы (software) пытаются сломать ваше представление об исполнении программы в многопоточном контексте. Есть еще процессор и подсистема памяти (hardware). Которые умеют:

- Исполнять независимые инструкции одновременно (out-of-order execution)
- Задействовать одни и те же ресурсы для исполнения логически независимых потоков (hyper-threading)
- Спекулировать^{13,14}
 - О предстоящих переходах (branch prediction)
 - О требуемой памяти (cache prefetching)
 - О результате вычислений (speculative execution)
 - И многом другом

¹³ [https://en.wikipedia.org/wiki/Spectre_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Spectre_(security_vulnerability))

¹⁴ [https://en.wikipedia.org/wiki/Meltdown_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability))

x86: Store buffering

```
int x, y;  
  
void threadA() {  
    x = 1;  
    int a = y;  
}  
  
void threadB() {  
    y = 1;  
    int b = x;  
}
```

x86: Store buffering

```
int x, y;

void threadA() {
    x = 1;
    int a = y;
}

# thread A
mov [x] , 1 # (A.1)
mov EAX , [y] # (A.2)

void threadB() {
    y = 1;
    int b = x;
}

# thread B
mov [y] , 1 # (B.1)
mov EBX, [x] # (B.2)
```

x86: Store buffering

thread A

```
mov [x] , 1 # (A.1)  
mov EAX , [y] # (A.2)
```

thread B

```
mov [y] , 1 # (B.1)  
mov EBX, [x] # (B.2)
```

x86: Store buffering

```
# thread A                                # thread B
mov [x] , 1 # (A.1)                      mov [y] , 1 # (B.1)
mov EAX , [y] # (A.2)                      mov EBX, [x] # (B.2)
```

Какие значения для (EAX EBX) допустимы?

(1 1) , (0 1) , (1 0) , (0 0)

x86: Store buffering

```
# thread A
mov [x] , 1 # (A.1)
mov EAX , [y] # (A.2)
```

```
# thread B
mov [y] , 1 # (B.1)
mov EBX, [x] # (B.2)
```

Какие значения для (EAX EBX) допустимы?

(1 1) , (0 1) , (1 0) , (0 0)

Варианты исполнения:

- A.1 -> A.2 -> B.1 -> B.2
- B.1 -> A.2 -> B.2
- B.2 -> A.2
- B.1 -> A.1 -> A.2 -> B.2
- B.2 -> A.2
- B.2 -> A.1 -> A.2

x86: Store buffering

```
# thread A                                # thread B
mov [x] , 1 # (A.1)                      mov [y] , 1 # (B.1)
mov EAX , [y] # (A.2)                      mov EBX, [x] # (B.2)
```

Какие значения для (EAX EBX) допустимы?

(1 1) , (0 1) , (1 0) , (0 0)

Варианты исполнения:

- A.1 -> A.2 -> B.1 -> B.2 : (0, 1)
- B.1 -> A.2 -> B.2 : (1, 1)
- B.2 -> A.2 : (1, 1)
- B.1 -> A.1 -> A.2 -> B.2 : (1, 1)
- B.2 -> A.2 : (1, 1)
- B.2 -> A.1 -> A.2 : (1, 0)

x86: Store buffering

```
# thread A                                # thread B
mov [x] , 1 # (A.1)                      mov [y] , 1 # (B.1)
mov EAX , [y] # (A.2)                      mov EBX, [x] # (B.2)
```

Какие значения для (EAX EBX) допустимы?

Ответ: (1 1), (0 1), (1 0)

Варианты исполнения:

- A.1 -> A.2 -> B.1 -> B.2 : (0, 1)
- B.1 -> A.2 -> B.2 : (1, 1)
- B.2 -> A.2 : (1, 1)
- B.1 -> A.1 -> A.2 -> B.2 : (1, 1)
- B.2 -> A.2 : (1, 1)
- B.2 -> A.1 -> A.2 : (1, 0)

x86: Store buffering

```
# thread A                                # thread B
mov [x] , 1 # (A.1)                      mov [y] , 1 # (B.1)
mov EAX , [y] # (A.2)                      mov EBX, [x] # (B.2)
```

Какие значения для (EAX EBX) допустимы?

Ответ: (1 1) , (0 1) , (1 0)

x86: Store buffering

```
# thread A                                # thread B
mov [x] , 1 # (A.1)                      mov [y] , 1 # (B.1)
mov EAX , [y] # (A.2)                      mov EBX, [x] # (B.2)
```

Какие значения для (EAX EBX) допустимы?

Правильный ответ: (1 1) , (0 1) , (1 0) , (0 0)

x86: Store buffering

```
# thread A                                # thread B
mov [x] , 1 # (A.1)                      mov [y] , 1 # (B.1)
mov EAX , [y] # (A.2)                      mov EBX, [x] # (B.2)
```

Какие значения для (EAX EBX) допустимы?

Правильный ответ: (1 1) , (0 1) , (1 0) , (0 0)

Процессор иногда переупорядочивает записи и чтения.

x86: Store buffering

```
# thread A                                # thread B
mov [x] , 1 # (A.1)                      mov [y] , 1 # (B.1)
mov EAX , [y] # (A.2)                      mov EBX, [x] # (B.2)
```

Какие значения для (EAX EBX) допустимы?

Правильный ответ: (1 1) , (0 1) , (1 0) , (0 0)

Процессор иногда переупорядочивает записи и чтения.

Вывод: порядок инструкций в машинном коде \neq порядок наблюдаемых эффектов этих инструкций.

x86: Store buffering

```
# thread A
mov [x] , 1 # (A.1)
mov EAX , [y] # (A.2)
```

```
# thread B
mov [y] , 1 # (B.1)
mov EBX, [x] # (B.2)
```

Какие значения для (EAX EBX) допустимы?

Правильный ответ: (1 1) , (0 1) , (1 0) , (0 0)

Процессор иногда переупорядочивает записи и ч

Вывод: порядок инструкций в машинном коде \neq
наблюдаемых эффектов этих инструкций.



arm64: Independent Reads of Independent Writes

thread1

$x = 1$

thread2

$y = 1$

thread3

$r1 = x$
 $r2 = y$

thread4

$r3 = y$
 $r4 = x$

arm64: Independent Reads of Independent Writes

thread1

$x = 1$

thread2

$y = 1$

thread3

$r1 = x$
 $r2 = y$

thread4

$r3 = y$
 $r4 = x$

Может ли быть так, что ($r1 = 1$, $r2 = 0$, $r3 = 1$, $r4 = 0$)?

arm64: Independent Reads of Independent Writes

thread1

$x = 1$

thread2

$y = 1$

thread3

$r1 = x$
 $r2 = y$

thread4

$r3 = y$
 $r4 = x$

Может ли быть так, что ($r1 = 1$, $r2 = 0$, $r3 = 1$, $r4 = 0$)?
При условии, что переупорядочивание чтений не происходит.

arm64: Independent Reads of Independent Writes

thread1

 $x = 1$

thread2

 $y = 1$

thread3

 $r1 = x$
 $r2 = y$

thread4

 $r3 = y$
 $r4 = x$

Может ли быть так, что ($r1 = 1$, $r2 = 0$, $r3 = 1$, $r4 = 0$)?

При условии, что переупорядочивание чтений не происходит.

- На x86 или x86_64 (TSO): нет

arm64: Independent Reads of Independent Writes

thread1

 $x = 1$

thread2

 $y = 1$

thread3

 $r1 = x$ $r2 = y$

thread4

 $r3 = y$ $r4 = x$

Может ли быть так, что ($r1 = 1$, $r2 = 0$, $r3 = 1$, $r4 = 0$)?

При условии, что переупорядочивание чтений не происходит.

- На x86 или x86_64 (TSO): нет
- На ARM или POWER: да¹⁵

¹⁵ A Tutorial Introduction to the ARM and POWER Relaxed Memory Models, section 6.1

arm64: Independent Reads of Independent Writes

thread1

 $x = 1$

thread2

 $y = 1$

thread3

 $r1 = x$ $r2 = y$

thread4

 $r3 = y$ $r4 = x$

Может ли быть так, что ($r1 = 1$, $r2 = 0$, $r3 = 1$, $r4 = 0$)?

При условии, что переупорядочивание чтений не происходит.

- На x86 или x86_64 (TSO): нет
- На ARM или POWER: да¹⁵

Записи могут "доехать" до других процессоров в разном порядке.

¹⁵ A Tutorial Introduction to the ARM and POWER Relaxed Memory Models, section 6.1

arm64: Independent Reads of Independent Writes

thread1

 $x = 1$

thread2

 $y = 1$

thread3

 $r1 = x$ $r2 = y$

thread4

 $r3 = y$ $r4 = x$

Может ли быть так, что ($r1 = 1$, $r2 = 0$, $r3 = 1$, $r4 = 0$)?

При условии, что переупорядочивание чтений не происходит.

- На x86 или x86_64 (TSO): нет
- На ARM или POWER: да¹⁵

Записи могут "доехать" до других процессоров в разном порядке.

У каждого процессора своя временная шкала и некоторое видение окружающего мира. Возможно, отличающееся от других процессоров.

¹⁵ A Tutorial Introduction to the ARM and POWER Relaxed Memory Models, section 6.1

arm64: Independent Reads of Independent Writes

| thread1 | thread2 | thread3 | thread4 |
|---------|---------|----------------------|----------------------|
| $x = 1$ | $y = 1$ | $r1 = x$ $r2 = y$ | $r3 = y$ $r4 = x$ |

Может ли быть так, что ($r1 = 1$, $r2 = 0$, $r3 = 1$, $r4 = 0$)?

При условии, что переупорядочивание чтений не происходит.

- На x86 или x86_64 (TSO): нет
- На ARM или POWER: да¹⁵

Записи могут "доехать" до других процессоров в разном порядке. У каждого процессора своя временная шкала и некоторое видение окружающего мира. Возможно, отличающееся от других процессоров. Вывод: нельзя рассматривать "запись в ячейку памяти" как точку на единой временной шкале¹⁶.

¹⁵ A Tutorial Introduction to the ARM and POWER Relaxed Memory Models, section 6.1

¹⁶ The Art of Multiprocessor Programming by Maurice Herlihy & Nir Shavit, Chapter 3 "Concurrent Objects" ↗ ↘ ↙

arm64: Independent Reads of Independent Writes

| thread1 | thread2 | thread3 | thread4 |
|---------|---------|----------------------|----------------------|
| $x = 1$ | $y = ?$ | $r1 = x$ $r2 = y$ | $r3 = y$ $r4 = x$ |

Может ли быть так, что $r1 = 1$, $r2 = 0$, $r3 = 1$, $r4 = 0$)?
При условии, что переупорядочивание не происходит.

- На x86 или x86_64: да¹⁵
- На ARM или POWER: да¹⁶

Записи могут "доехать" до другого процессора в разном порядке.
У каждого процессора своя временная шкала и некоторое видение окружающего мира. Возможно, отличающееся от других процессоров.
Вывод: нельзя рассматривать "запись в ячейку памяти" как точку на единой временной шкале¹⁶.

¹⁵ A Tutorial Introduction to the ARM and POWER Relaxed Memory Models, section 6.1

¹⁶ The Art of Multiprocessor Programming by Maurice Herlihy & Nir Shavit, Chapter 3 "Concurrent Objects"

Почему так сложно?

- порядок инструкций в машинном коде \neq порядок наблюдаемых эффектов этих инструкций

Почему так сложно?

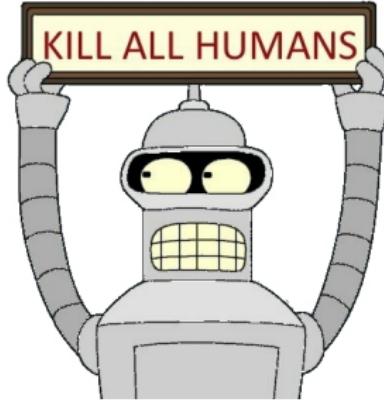
- порядок инструкций в машинном коде \neq порядок наблюдаемых эффектов этих инструкций
- нельзя рассматривать "чтение/запись в ячейку памяти" как точку на единой временной шкале

Почему так сложно?

- порядок инструкций в машинном коде \neq порядок наблюдаемых эффектов этих инструкций
- нельзя рассматривать "чтение/запись в ячейку памяти" как точку на единой временной шкале
- у каждого процессора свои правила

Почему так сложно?

- порядок инструкций в машинном коде \neq порядок наблюдаемых эффектов этих инструкций
- нельзя рассматривать "чтение/запись в ячейку памяти" как точку на единой временной шкале
- у каждого процессора свои правила



Почему так сложно?

- порядок инструкций в машинном коде \neq порядок наблюдаемых эффектов этих инструкций
- нельзя рассматривать "чтение/запись в ячейку памяти" как точку на единой временной шкале
- у каждого процессора свои правила

Почему вообще хоть кто-то пользуется ARM/POWER/RISC-V и другими процессорами со слабой моделью памяти?

Почему так сложно?

- порядок инструкций в машинном коде \neq порядок наблюдаемых эффектов этих инструкций
- нельзя рассматривать "чтение/запись в ячейку памяти" как точку на единой временной шкале
- у каждого процессора свои правила

Почему вообще хоть кто-то пользуется ARM/POWER/RISC-V и другими процессорами со слабой моделью памяти?

- производительность

Почему так сложно?

- порядок инструкций в машинном коде \neq порядок наблюдаемых эффектов этих инструкций
- нельзя рассматривать "чтение/запись в ячейку памяти" как точку на единой временной шкале
- у каждого процессора свои правила

Почему вообще хоть кто-то пользуется ARM/POWER/RISC-V и другими процессорами со слабой моделью памяти?

- производительность
- Производительность!

Почему так сложно?

- порядок инструкций в машинном коде \neq порядок наблюдаемых эффектов этих инструкций
- нельзя рассматривать "чтение/запись в ячейку памяти" как точку на единой временной шкале
- у каждого процессора свои правила

Почему вообще хоть кто-то пользуется ARM/POWER/RISC-V и другими процессорами со слабой моделью памяти?

- производительность
- Производительность!
- энергосбережение :)

Вы находитесь здесь

- 1 Знакомство
- 2 Как я вижу программистов
- 3 Как я вижу системных программистов
- 4 Как я вижу компиляторы
- 5 Как я вижу процессоры
- 6 **Как я вижу языки программирования**
- 7 Подведение итогов

Holy war warning

- Все языки хороши и полезны!

Holy war warning

- Все языки хороши и полезны!
- Просто некоторые вещи в некоторых языках могут мешать при выполнении некоторых задач.

Holy war warning

- Все языки хороши и полезны!
- Просто некоторые вещи в некоторых языках могут мешать при выполнении некоторых задач.
- Не относитесь серьезно к моим словам.

Swift: race to the crash

Swift¹⁷

Concurrent write/write or read/write access to the same location in memory generally remains undefined/illegal behavior, unless all such access is done through a special set of primitive atomic operations.

¹⁷ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Swift: race to the crash

Swift¹⁷

Concurrent write/write or read/write access to the same location in memory generally remains undefined/illegal behavior, unless all such access is done through a special set of primitive atomic operations.

¹⁷ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Swift: race to the crash

Swift¹⁷

Concurrent write/write access to the same location remains illegal behavior, unless is done through atomic operations.

¹⁷ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Swift: race to the crash

Swift¹⁷

Concurrent write/write access to the same location remains illegal behavior, unless is done through atomic operations.

```
import Foundation
class Bird {}
var S = Bird()
let q = DispatchQueue.global(qos: .default)
q.async { while(true) { S = Bird() } }
while(true) { S = Bird() }
```

¹⁷ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Swift: race to the crash

Swift¹⁷

Concurrent write/write access to the same location remains illegal behavior, unless is done through atomic operations.

```
import Foundation
class Bird {}
var S = Bird()
let q = DispatchQueue.global(qos: .default)
q.async { while(true) { S = Bird() } }
while(true) { S = Bird() }
```

При запуске происходит ошибка double free or corruption.

¹⁷ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Swift: race to the crash

Swift¹⁷

Concurrent write/write access to the same location remains illegal behavior, unless is done through atomic operations.

```
import Foundation
class Bird {}
var S = Bird()
let q = DispatchQueue.global(qos: .default)
q.async { while(true) { S = Bird() } }
while(true) { S = Bird() }
```

При запуске происходит ошибка double free or corruption.

Почему? Попробуйте догадаться сами¹⁸ или подсмотрите в решебник¹⁹.

¹⁷ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

¹⁸ <https://tonygold.github.io/arceMPIRE/>

¹⁹ <https://github.com/apple/swift/blob/main/docs/proposals/Concurrency.rst>

Python: VIP mutex

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени²⁰.

²⁰

https://en.wikipedia.org/wiki/Consistency_model#Strict_consistency



Python: VIP mutex

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

```
void thread1() {           | void thread2() {  
    |   foo()          |   baz()  
    |   bar()          |   foo()  
    | }               | }
```

Python: VIP mutex

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

```
void thread1() {           |   void thread2() {  
    lock()                |       lock()  
    foo()                 |       baz()  
    unlock()              |       unlock()  
    lock()                |       lock()  
    bar()                 |       foo()  
    unlock()              |       unlock()  
}  
                           | }
```

Python: VIP mutex

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени. Защищать глобальным мьютексом каждую операцию.

```
static GlobalInterpreterLock GIL = ...;
void thread1() {           |   void thread2() {
    GIL.lock()             |       GIL.lock()
    foo()                  |       baz()
    GIL.unlock()            |       GIL.unlock()
    GIL.lock()              |       GIL.lock()
    bar()                  |       foo()
    GIL.unlock()            |       GIL.unlock()
}
}                           | }
```

Python: VIP mutex

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени. Защищать глобальным мьютексом каждую операцию.

Python: VIP mutex

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени. Защищать глобальным мьютексом каждую операцию.

- Потоки в языке есть²¹, а их неэффективность является "особенностью" интерпретатора CPython.

²¹ <https://docs.python.org/3/library/threading.html>

Python: VIP mutex

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени. Защищать глобальным мьютексом каждую операцию.

- Потоки в языке есть²¹, а их неэффективность является "особенностью" интерпретатора CPython.
- PyPy тоже не собирается отказываться от GIL²².

²¹ <https://docs.python.org/3/library/threading.html>

²² <https://doc.pypy.org/en/latest/faq.html#does-pypy-have-a-gil-why>

Python: VIP mutex

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени. Защищать глобальным мьютексом каждую операцию.

- Потоки в языке есть²¹, а их неэффективность является "особенностью" интерпретатора CPython.
- PyPy тоже не собирается отказываться от GIL²².
- Попытка переделать модель языка пока не увенчалась успехом²³.

²¹ <https://docs.python.org/3/library/threading.html>

²² <https://doc.pypy.org/en/latest/faq.html#does-pypy-have-a-gil-why>

²³ <https://peps.python.org/pep-0583/>

Python: VIP mutex

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени. Защищать глобальным мьютексом каждую операцию.

- Потоки в языке есть²¹, а их неэффективность является "особенностью" интерпретатора CPython.
- PyPy тоже не собирается отказываться от GIL²².
- Попытка переделать модель языка пока не увенчалась успехом²³.
- Выбрасыванию GIL мешает нежелание замедлять скриптовый язык еще на 10-30%, ломая интероп с нативными библиотеками²⁴.

²¹ <https://docs.python.org/3/library/threading.html>

²² <https://doc.pypy.org/en/latest/faq.html#does-pypy-have-a-gil-why>

²³ <https://peps.python.org/pep-0583/>

²⁴ <https://peps.python.org/pep-0703/>

JavaScript: say no to threading

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

JavaScript: say no to threading

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

Пусть в языке вообще не будет потоков.

JavaScript: say no to threading

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

Пусть в языке вообще не будет потоков.

Единственный поток обрабатывает все события. Событие может породить другие, возможно отложенные, события.

JavaScript: say no to threading

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

Пусть в языке вообще не будет потоков.

Единственный поток обрабатывает все события. Событие может породить другие, возможно отложенные, события. Event loop.

JavaScript: say no to threading

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

Пусть в языке вообще не будет потоков.

Единственный поток обрабатывает все события. Событие может породить другие, возможно отложенные, события. Event loop.

- Пользователи любят использовать все ядра своих систем.

JavaScript: say no to threading

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

Пусть в языке вообще не будет потоков.

Единственный поток обрабатывает все события. Событие может породить другие, возможно отложенные, события. Event loop.

- Пользователи любят использовать все ядра своих систем.
- Можно запускать дополнительных независимых агентов (web workers) и общаться сообщениями²⁵.

²⁵ https://www.w3schools.com/html/html5_webworkers.asp

JavaScript: say no to threading

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

Пусть в языке вообще не будет потоков.

Единственный поток обрабатывает все события. Событие может породить другие, возможно отложенные, события. Event loop.

- Пользователи любят использовать все ядра своих систем.
- Можно запускать дополнительных независимых агентов (web workers) и общаться сообщениями²⁵.
- Можно разделять между агентами массивы байтов²⁶.

²⁵ https://www.w3schools.com/html/html5_webworkers.asp

²⁶ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer

JavaScript: say no to threading

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени.

Пусть в языке вообще не будет потоков.

Единственный поток обрабатывает все события. Событие может породить другие, возможно отложенные, события. Event loop.

- Пользователи любят использовать все ядра своих систем.
- Можно запускать дополнительных независимых агентов (web workers) и общаться сообщениями²⁵.
- Можно разделять между агентами массивы байтов²⁶.
- Можно получить data race и думать, что это значит²⁷.

²⁵ https://www.w3schools.com/html/html5_webworkers.asp

²⁶ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer

²⁷ "Repairing and Mechanising the JavaScript Relaxed Memory Model" <https://arxiv.org/abs/2005.10554> ↗ ↘ ↙ ↘

Java: математично!

Java: математично!

Используется тяжелая артиллериya:

Java: математично!

Используется тяжелая артиллериya:

- An action a is described by a tuple $\langle t, k, v, u \rangle$ comprising ...

Java: математично!

Используется тяжелая артиллерия:

- An action a is described by a tuple $\langle t, k, v, u \rangle$ comprising ...
- Частичные, линейные порядки; транзитивное замыкание бинарных отношений; happens-before

Java: математично!

Используется тяжелая артиллерия:

- An action a is described by a tuple $\langle t, k, v, u \rangle$ comprising ...
- Частичные, линейные порядки; транзитивное замыкание бинарных отношений; happens-before
- Causality requirements, circular hp, out-of-thin-air problem

Java: математично!

Используется тяжелая артиллериya:

- An action a is described by a tuple $\langle t, k, v, u \rangle$ comprising ...
- Частичные, линейные порядки; транзитивное замыкание бинарных отношений; happens-before
- Causality requirements, circular hp, out-of-thin-air problem
- Adaptation to h/w models²⁸

²⁸"JSR-133 Cookbook for Compiler Writers" <https://gee.cs.oswego.edu/dl/jmm/cookbook.html>

Java: математично!

Используется тяжелая артиллериия:

- An action a is described by a tuple $\langle t, k, v, u \rangle$ comprising ...
- Частичные, линейные порядки; транзитивное замыкание бинарных отношений; happens-before
- Causality requirements, circular hp, out-of-thin-air problem
- Adaptation to h/w models²⁸
- Каждый data race имеет разрешенные и запрещенные последствия

²⁸"JSR-133 Cookbook for Compiler Writers" <https://gee.cs.oswego.edu/dl/jmm/cookbook.html>

Java: математично!

Используется тяжелая артиллериия:

- An action a is described by a tuple $\langle t, k, v, u \rangle$ comprising ...
- Частичные, линейные порядки; транзитивное замыкание бинарных отношений; happens-before
- Causality requirements, circular hp, out-of-thin-air problem
- Adaptation to h/w models²⁸
- Каждый data race имеет разрешенные и запрещенные последствия

Подход обладает рядом недостатков:

- Очень сложно, долго и дорого.
- Будут недочеты²⁹.
- Мало кто в мире будет в состоянии полностью понять написанное.
Еще меньше людей смогут применить на практике.

²⁸ "JSR-133 Cookbook for Compiler Writers" <https://gee.cs.oswego.edu/dl/jmm/cookbook.html>

²⁹ "Java Memory Model Examples: Good, Bad and Ugly" <https://groups.inf.ed.ac.uk/request/jmmexamples.pdf>

Use patterns, Luke

Doug Lea, private communication with Aleksey Shipilev, 2013³⁰

The best way to build up a small repertoire of constructions that you know the answers for and then never think about the JMM rules again unless you are forced to do so! Literally nobody likes figuring things out from the JMM rules as stated, or can even routinely do so correctly. This is one of the many reasons we need to overhaul JMM someday.

³⁰ Citation from <https://shipilev.net/blog/2014/jmm-pragmatics>, slide 109

Многопоточность + Язык программирования = ?

Существуют языки программирования на любой вкус:

- Простые и не очень

Многопоточность + Язык программирования = ?

Существуют языки программирования на любой вкус:

- Простые и не очень
- Быстрые и не очень

Многопоточность + Язык программирования = ?

Существуют языки программирования на любой вкус:

- Простые и не очень
- Быстрые и не очень
- Многопоточные и не очень

Многопоточность + Язык программирования = ?

Существуют языки программирования на любой вкус:

- Простые и не очень
- Быстрые и не очень
- Многопоточные и не очень

Выбирайте подходящий язык для конкретной задачи.

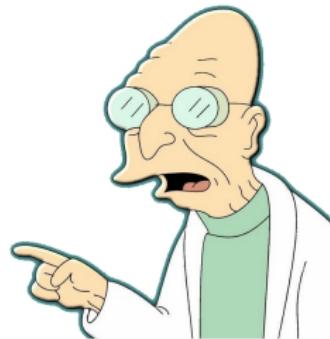
Многопоточность + Язык программирования = ?

Существуют языки программирования на любой вкус:

- Простые и не очень
- Быстрые и не очень
- Многопоточные и не очень

Выбирайте подходящий язык для конкретной задачи.

**KEEP IT
SIMPLE
STUPID**



Вы находитесь здесь

- 1 Знакомство
- 2 Как я вижу программистов
- 3 Как я вижу системных программистов
- 4 Как я вижу компиляторы
- 5 Как я вижу процессоры
- 6 Как я вижу языки программирования
- 7 Подведение итогов

Где же научиться писать правильные многопоточные программы?

Где же научиться писать правильные многопоточные программы?

Простых способов я не знаю.

Где же научиться писать правильные многопоточные программы?

Простых способов я не знаю.

- Java Concurrency in Practice – 403 страницы
- The Art of Multiprocessor Programming – 508 страниц
- Is Parallel Programming Hard, And, If So, What Can You Do About It? – 634 страницы

Где же научиться писать правильные многопоточные программы?

Простых способов я не знаю.

- Java Concurrency in Practice – 403 страницы
- The Art of Multiprocessor Programming – 508 страниц
- Is Parallel Programming Hard, And, If So, What Can You Do About It? – 634 страницы

Russ Cox, Go programming language tech lead at Google, author of Go memory model:³¹

Twenty-five years after the first Java memory model, and after many person-centuries of research effort, we may be starting to be able to formalize entire memory models. Perhaps, one day, we will also fully understand them.

³¹ <https://research.swtch.com/plmm>

Заключение

Я рассказал вам о своих страхах, привычках и предубеждениях о многопоточном программировании.

Заключение

Я рассказал вам о своих страхах, привычках и предубеждениях о многопоточном программировании.

Гораздо больше осталось за кадром:

- Сколько интересного скрыто в планировщике задач ОС и его альтернативах

Заключение

Я рассказал вам о своих страхах, привычках и предубеждениях о многопоточном программировании.

Гораздо больше осталось за кадром:

- Сколько интересного скрыто в планировщике задач ОС и его альтернативах
- Об огромных богатствах слабых моделей памяти

Заключение

Я рассказал вам о своих страхах, привычках и предубеждениях о многопоточном программировании.

Гораздо больше осталось за кадром:

- Сколько интересного скрыто в планировщике задач ОС и его альтернативах
- Об огромных богатствах слабых моделей памяти
- Про фундаментальные алгоритмы и нерешенные проблемы в многопоточности

Заключение

Я рассказал вам о своих страхах, привычках и предубеждениях о многопоточном программировании.

Гораздо больше осталось за кадром:

- Сколько интересного скрыто в планировщике задач ОС и его альтернативах
- Об огромных богатствах слабых моделей памяти
- Про фундаментальные алгоритмы и нерешенные проблемы в многопоточности

Если вам стало интересно – ищите правильных людей. Ходите в университет и на конференции, читайте сложные книги, перенимайте опыт у практикующих разработчиков.

Заключение

Я рассказал вам о своих страхах, привычках и предубеждениях о многопоточном программировании.

Гораздо больше осталось за кадром:

- Сколько интересного скрыто в планировщике задач ОС и его альтернативах
- Об огромных богатствах слабых моделей памяти
- Про фундаментальные алгоритмы и нерешенные проблемы в многопоточности

Если вам стало интересно – ищите правильных людей. Ходите в университет и на конференции, читайте сложные книги, перенимайте опыт у практикующих разработчиков.

Помните, с вами говорил не нормальный человек, а VM-инженер.

Где можно найти приключения

- Для студентов: стажировка у системных программистов.
<http://rnew.tilda.ws/excelsiorathuawei>



- Для абитуриентов: профиль "Системное программирование"
ММФ НГУ. <https://education.nsu.ru/syspro/>

*** $((N^*)\text{nullptr})$**

*The real system programming

MMF NSU

Спасибо за внимание!



Почитать

Книги

- "The Art of Multiprocessor Programming" by M. Herlihy & N. Shavit
- "Is Parallel Programming Hard, And, If So, What Can You Do About It?" by Paul E. McKenney
- "Java Concurrency in Practice" by Brian Goetz et al.

Статьи

- "Memory Models" series by Russ Cox³²
- "Threads Cannot be Implemented as a Library" by Hans-J. Boehm
- "A Tutorial Introduction to the ARM and POWER Relaxed Memory Models" by L. Maranget et al.
- "Memory Barriers: a Hardware View for Software Hackers" by Paul E. McKenney

³² <https://research.swtch.com/mm>

Посмотреть

- Роман Елизаров, "Многопоточное программирование — теория и практика" <https://youtu.be/mf4lC6TpclM>
- Алексей Шипилев, JMM series <https://shipilev.net>
- Herb Sutter, C++ and Beyond 2012, "Atomic Weapons" series <https://youtu.be/A8eCG0qgvH4>