

Пара слов про модели памяти языков программирования

Александр Филатов
filatovaur@gmail.com

<https://github.com/Svazars/lang-mem-models-intro>

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

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

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

- 2015 - 2019, Excelsior JVM with AOT compilation²

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

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

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

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

¹ Иван Угрянский, 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>

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

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

Специализация – рантайм JVM

¹ Иван Угрянский, 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>

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

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

Специализация – рантайм JVM

Узкая специализация – сборщики мусора

¹ Иван Угрянский, 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>

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

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

Специализация – рантайм JVM

Узкая специализация – сборщики мусора

Область интересов:

¹ Иван Угрянский, 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>

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

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

Специализация – рантайм JVM

Узкая специализация – сборщики мусора

Область интересов:

- МНОГОПОТОЧНОСТЬ

¹ Иван Угрянский, 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>

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

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

Специализация – рантайм JVM

Узкая специализация – сборщики мусора

Область интересов:

- многопоточность
- слабые модели памяти

¹ Иван Угрянский, 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>

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

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

Специализация – рантайм JVM

Узкая специализация – сборщики мусора

Область интересов:

- многопоточность
- слабые модели памяти
- корректность многопоточных структур данных

¹ Иван Угрянский, 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>

Био: Александр Филатов

Уже 8 лет как JVM-инженер¹

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

Специализация – рантайм JVM

Узкая специализация – сборщики мусора

Область интересов:

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

¹ Иван Угрянский, 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>

Мое персональное когнитивное искажение

Я много страдал, отлаживая:

Мое персональное когнитивное искажение

Я много страдал, отлаживая:

- баги компилятора

Мое персональное когнитивное искажение

Я много страдал, отлаживая:

- баги компилятора
- своего параллельного кода

Мое персональное когнитивное искажение

Я много страдал, отлаживая:

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

Мое персональное когнитивное искажение

Я много страдал, отлаживая:

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

Очень хочу поручить это дело бездушной машине. Потому питаю слабость к формализмам и математической нотации.

Disclaimer

Disclaimer

Лекция называется "пара слов про ..." и будет идти 1.5 часа.

Disclaimer

Лекция называется "пара слов про ..." и будет идти 1.5 часа.
В последующих слайдах будет много упрощений и далеко не вся важная информация будет рассказана.

Disclaimer

Лекция называется "пара слов про ..." и будет идти 1.5 часа.
В последующих слайдах будет много упрощений и далеко не вся важная информация будет рассказана.
Лекция носит ознакомительно-развлекательный характер.

Disclaimer

Лекция называется "пара слов про ..." и будет идти 1.5 часа. В последующих слайдах будет много упрощений и далеко не вся важная информация будет рассказана.

Лекция носит ознакомительно-развлекательный характер.

По ходу дела мы больше углубимся в теорию и разные философские вопросы. Если вам хочется знать, почему это небесполезно на практике, то рекомендую посмотреть интересный доклад про биржи, котировки и их обработку на Java⁴.

⁴ Роман Елизаров "Миллионы котировок в секунду на чистой Java" <https://youtu.be/j3wF0mRmSeg>

Disclaimer

Лекция называется "пара слов про ..." и будет идти 1.5 часа. В последующих слайдах будет много упрощений и далеко не вся важная информация будет рассказана.

Лекция носит ознакомительно-развлекательный характер.

По ходу дела мы больше углубимся в теорию и разные философские вопросы. Если вам хочется знать, почему это небесполезно на практике, то рекомендую посмотреть интересный доклад про биржи, котировки и их обработку на Java⁴.

Смотрите ссылки, читайте книги, задавайте вопросы, обязательно ходите на продвинутые курсы по параллелизму, многопоточности, распределенным системам, верификации программ в нашем университете и лучших ВУЗах мира⁵.

⁴ Роман Елизаров "Миллионы котировок в секунду на чистой Java" <https://youtu.be/j3wF0mRmSeg>

⁵ Спасибо youtube, coursera и аналогам за такую возможность

Блиц-опрос

Пожалуйста, поднимите руку, если вы:

Блиц-опрос

Пожалуйста, поднимите руку, если вы:

- Запускали `http://deadlockempire.github.io/`

Блиц-опрос

Пожалуйста, поднимите руку, если вы:

- Запускали `http://deadlockempire.github.io/`
- Прошли обучалку

Блиц-опрос

Пожалуйста, поднимите руку, если вы:

- Запускали `http://deadlockempire.github.io/`
- Прошли обучалку
- Прошли все уровни

Блиц-опрос

Пожалуйста, поднимите руку, если вы:

- Запускали `http://deadlockempire.github.io/`
- Прошли обучалку
- Прошли все уровни
- Вам показалось слишком просто

Блиц-опрос

Пожалуйста, поднимите руку, если вы:

- Запускали `http://deadlockempire.github.io/`
- Прошли обучалку
- Прошли все уровни
- Вам показалось слишком просто

Спасибо!

План выступления


- 1 Знакомство
- 2 Зачем писать многопоточный код?
- 3 Компилятор хотел как лучше, а получилось...
- 4 Процессор хотел как лучше, а получилось...
- 5 Программист хочет как лучше...
- 6 Language memory models. Примеры из жизни.
- 7 Подведение итогов

Вы находитесь здесь

- 1 Знакомство
- 2 Зачем писать многопоточный код?**
- 3 Компилятор хотел как лучше, а получилось...
- 4 Процессор хотел как лучше, а получилось...
- 5 Программист хочет как лучше...
- 6 Language memory models. Примеры из жизни.
- 7 Подведение итогов

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

⁶Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk> 

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

Возможные варианты ответа:

⁶Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk>

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

Возможные варианты ответа:

- Так принято еще с 2000-х годов

⁶Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk>

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

Возможные варианты ответа:

- Так принято еще с 2000-х годов
- Современное оборудование фантастически многоядерное


⁶Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk>

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

Возможные варианты ответа:

- Так принято еще с 2000-х годов
- Современное оборудование фантастически многоядерное
- Необходимо строить сложные системы, ориентированные на огромное число пользователей

⁶Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk> 

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

Возможные варианты ответа:

- Так принято еще с 2000-х годов
- Современное оборудование фантастически многоядерное
- Необходимо строить сложные системы, ориентированные на огромное число пользователей
- А как иначе написать

⁶Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk>

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

Возможные варианты ответа:

- Так принято еще с 2000-х годов
- Современное оборудование фантастически многоядерное
- Необходимо строить сложные системы, ориентированные на огромное число пользователей
- А как иначе написать
 - сервер обработки входящий соединений?

⁶ Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk>

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

Возможные варианты ответа:

- Так принято еще с 2000-х годов
- Современное оборудование фантастически многоядерное
- Необходимо строить сложные системы, ориентированные на огромное число пользователей
- А как иначе написать
 - сервер обработки входящих соединений?
 - многоагентную симуляцию?

⁶ Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk>

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

Возможные варианты ответа:

- Так принято еще с 2000-х годов
- Современное оборудование фантастически многоядерное
- Необходимо строить сложные системы, ориентированные на огромное число пользователей
- А как иначе написать
 - сервер обработки входящих соединений?
 - многоагентную симуляцию?
 - игру-песочницу типа Minecraft?

⁶ Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk>

Зачем писать параллельные программы?

Зачем в языке программирования давать средства для написания параллельных(parallel)/конкурентных(concurrent) программ?⁶

Возможные варианты ответа:

- Так принято еще с 2000-х годов
- Современное оборудование фантастически многоядерное
- Необходимо строить сложные системы, ориентированные на огромное число пользователей
- А как иначе написать
 - сервер обработки входящих соединений?
 - многоагентную симуляцию?
 - игру-песочницу типа Minecraft?











В подавляющем большинстве случаев, написание параллельного/распределенного/многопоточного кода – это оптимизация.

⁶Rob Pike, Concurrency Is Not Parallelism <https://go.dev/blog/waza-talk>

Альтернативы

Bash

Задача: подсчитать число слов в текстовом файле⁷.

⁷<https://github.com/Svazars/lang-mem-models-intro/tree/main/samples/bash>          

Альтернативы











Bash

Задача: подсчитать число слов в текстовом файле⁷.

Последовательное исполнение

```
wc 50_000_000.txt
```

```
real    0m 4,900s
user    0m 4,212s
sys     0m 0,240s
```

⁷<https://github.com/Svazars/lang-mem-models-intro/tree/main/samples/bash>          

Альтернативы

Bash

Задача: подсчитать число слов в текстовом файле⁷.

Последовательное исполнение

```
wc 50_000_000.txt
```

```
real    0m 4,900s
user    0m 4,212s
sys     0m 0,240s
```

Параллельное исполнение (2 процесса)

```
{ ( head -n 25000000 50_000_000.txt | wc ) & }
{ ( tail -n 25000000 50_000_000.txt | wc ) & }
wait
```

```
real    0m 2,323s
user    0m 4,576s
sys     0m 1,084s
```

⁷<https://github.com/Svazars/lang-mem-models-intro/tree/main/samples/bash>

Альтернативы

make

Задача: скомпилировать большой проект на языке C.

Альтернативы

make

Задача: скомпилировать большой проект на языке C.

```
make -j8
```

Альтернативы

make

Задача: скомпилировать большой проект на языке C.

```
make -j8
```

Аналогичные инструменты:

- ant/maven (Java)
- groovy DSL (Jenkins)
- sbt (Scala)
- ...

Параллелизм уровня процессов

Обсуждение

Параллелизм уровня процессов

Обсуждение

- ОС создана для того, чтобы быстро, эффективно и надежно управлять процессами

Параллелизм уровня процессов

Обсуждение

- ОС создана для того, чтобы быстро, эффективно и надежно управлять процессами
- Независимые шаги вычислений можно выполнять в разных процессах

Параллелизм уровня процессов

Обсуждение

- ОС создана для того, чтобы быстро, эффективно и надежно управлять процессами
- Независимые шаги вычислений можно выполнять в разных процессах
- Write programs that do one thing and do it well⁸

⁸ https://en.wikipedia.org/wiki/Unix_philosophy

Параллелизм уровня процессов

Обсуждение

- ОС создана для того, чтобы быстро, эффективно и надежно управлять процессами
- Независимые шаги вычислений можно выполнять в разных процессах
- Write programs that do one thing and do it well⁸

Исполнение нескольких потоков вычислений внутри одного процесса не требуется.

⁸ https://en.wikipedia.org/wiki/Unix_philosophy

Параллелизм уровня процессов

Обсуждение

- ОС создана для того, чтобы быстро, эффективно и надежно управлять процессами
- Независимые шаги вычислений можно выполнять в разных процессах
- Write programs that do one thing and do it well⁸

Исполнение нескольких потоков вычислений внутри одного процесса не требуется.

Параллелизм уровня потоков не нужен.

⁸ https://en.wikipedia.org/wiki/Unix_philosophy

Параллелизм уровня процессов

Обсуждение

- ОС создана для того, чтобы быстро, эффективно и надежно управлять процессами
- Независимые шаги вычислений можно выполнять в разных процессах
- Write programs that do one thing and do it well⁸

Исполнение нескольких потоков вычислений внутри одного процесса не требуется.

Параллелизм уровня потоков не нужен.

Многопоточность не нужна.

⁸ https://en.wikipedia.org/wiki/Unix_philosophy

Параллелизм уровня процессов

Обсуждение

- ОС создана для того, чтобы быстро, эффективно и надежно управлять процессами
- Независимые шаги вычислений можно выполнять в разных процессах
- Write programs that do one thing and do it well⁸

Исполнение нескольких потоков вычислений внутри одного процесса не требуется.

Параллелизм уровня потоков не нужен.

Многопоточность не нужна.

В чем недостатки подхода с использованием только процессов?

⁸ https://en.wikipedia.org/wiki/Unix_philosophy

Параллелизм уровня процессов

Недостатки

Процесс – это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и независимого виртуального адресного пространства⁹.

⁹[https://ru.wikipedia.org/wiki/Процесс_\(информатика\)](https://ru.wikipedia.org/wiki/Процесс_(информатика))

Параллелизм уровня процессов

Недостатки

Процесс – это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и независимого виртуального адресного пространства⁹.

Межпроцессное взаимодействие бывает:

⁹[https://ru.wikipedia.org/wiki/Процесс_\(информатика\)](https://ru.wikipedia.org/wiki/Процесс_(информатика))

Параллелизм уровня процессов

Недостатки

Процесс – это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и независимого виртуального адресного пространства⁹.

Межпроцессное взаимодействие бывает:

- Долгое (latency)

⁹[https://ru.wikipedia.org/wiki/Процесс_\(информатика\)](https://ru.wikipedia.org/wiki/Процесс_(информатика))

Параллелизм уровня процессов

Недостатки

Процесс – это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и независимого виртуального адресного пространства⁹.

Межпроцессное взаимодействие бывает:

- Долгое (latency)
- Дорогое (throughput)

⁹[https://ru.wikipedia.org/wiki/Процесс_\(информатика\)](https://ru.wikipedia.org/wiki/Процесс_(информатика))

Параллелизм уровня процессов

Недостатки

Процесс – это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и независимого виртуального адресного пространства⁹.

Межпроцессное взаимодействие бывает:

- Долгое (latency)
- Дорогое (throughput)
- Не всегда кросс-платформенное (Windows/POSIX API vs. protobuf)¹⁰

⁹ [https://ru.wikipedia.org/wiki/Процесс_\(информатика\)](https://ru.wikipedia.org/wiki/Процесс_(информатика))

¹⁰ <https://stackoverflow.com/questions/60649/cross-platform-ipc>

Параллелизм уровня процессов

Недостатки

Процесс – это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и независимого виртуального адресного пространства⁹.

Межпроцессное взаимодействие бывает:

- Долгое (latency)
- Дорогое (throughput)
- Не всегда кросс-платформенное (Windows/POSIX API vs. protobuf)¹⁰
- Не всегда надежное (разрыв соединения, смерть процесса)

⁹[https://ru.wikipedia.org/wiki/Процесс_\(информатика\)](https://ru.wikipedia.org/wiki/Процесс_(информатика))

¹⁰<https://stackoverflow.com/questions/60649/cross-platform-ipc>

Параллелизм уровня процессов

Недостатки

Процесс – это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и независимого виртуального адресного пространства⁹.

Межпроцессное взаимодействие бывает:

- Долгое (latency)
- Дорогое (throughput)
- Не всегда кросс-платформенное (Windows/POSIX API vs. protobuf)¹⁰
- Не всегда надежное (разрыв соединения, смерть процесса)
- Не всегда безопасное

⁹[https://ru.wikipedia.org/wiki/Процесс_\(информатика\)](https://ru.wikipedia.org/wiki/Процесс_(информатика))

¹⁰<https://stackoverflow.com/questions/60649/cross-platform-ipc>

Параллелизм уровня процессов

Недостатки

Процесс – это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и **независимого виртуального адресного пространства**⁹.

Межпроцессное взаимодействие бывает:

- Долгое (latency)
- Дорогое (throughput)
- Не всегда кросс-платформенное (Windows/POSIX API vs. protobuf)¹⁰
- Не всегда надежное (разрыв соединения, смерть процесса)
- Не всегда безопасное

⁹[https://ru.wikipedia.org/wiki/Процесс_\(информатика\)](https://ru.wikipedia.org/wiki/Процесс_(информатика))

¹⁰<https://stackoverflow.com/questions/60649/cross-platform-ipc>

Параллелизм уровня потоков

Взаимодействие потоков исполнения (threads) внутри общего адресного пространства.

Параллелизм уровня потоков

Взаимодействие потоков исполнения (threads) внутри общего адресного пространства.

У каждого потока есть

- машинный стек
- идентификатор
- обработчик сигналов
- приоритет в планировщике задач
- ...

Параллелизм уровня потоков

Взаимодействие потоков исполнения (threads) внутри общего адресного пространства.

У каждого потока есть

- машинный стек
- идентификатор
- обработчик сигналов
- приоритет в планировщике задач
- ...
- быстрый доступ к общей разделяемой памяти

Параллелизм уровня потоков

Применения

Параллелизм уровня потоков

Применения

- Web-сервер. Клиент шлет запросы, получает ответы, но одна система допускает одновременную обработку нескольких соединений.

Параллелизм уровня потоков

Применения

- Web-сервер. Клиент шлет запросы, получает ответы, но одна система допускает одновременную обработку нескольких соединений.
- Многопользовательские игры. Сервер Minecraft позволяет различным игрокам одновременно взаимодействовать с окружением.

Параллелизм уровня потоков

Применения

- Web-сервер. Клиент шлет запросы, получает ответы, но одна система допускает одновременную обработку нескольких соединений.
- Многопользовательские игры. Сервер Minecraft позволяет различным игрокам одновременно взаимодействовать с окружением.
- Сборщик мусора. Вы создали объект, попользовались им и забыли. Не задумываясь о том, что где-то там есть сборщик мусора, которые отследит ненужность объекта и переиспользует память под следующий объект.

Параллелизм уровня потоков

Применения

- Web-сервер. Клиент шлет запросы, получает ответы, но одна система допускает одновременную обработку нескольких соединений.
- Многопользовательские игры. Сервер Minecraft позволяет различным игрокам одновременно взаимодействовать с окружением.
- Сборщик мусора. Вы создали объект, попользовались им и забыли. Не задумываясь о том, что где-то там есть сборщик мусора, которые отследит ненужность объекта и переиспользует память под следующий объект.
 - Независимые задачи (parallel marking, parallel sweeping)

Параллелизм уровня потоков

Применения

- Web-сервер. Клиент шлет запросы, получает ответы, но одна система допускает одновременную обработку нескольких соединений.
- Многопользовательские игры. Сервер Minecraft позволяет различным игрокам одновременно взаимодействовать с окружением.
- Сборщик мусора. Вы создали объект, попользовались им и забыли. Не задумываясь о том, что где-то там есть сборщик мусора, которые отследит ненужность объекта и переиспользует память под следующий объект.
 - Независимые задачи (parallel marking, parallel sweeping)
 - Одновременные задачи (concurrent marking, concurrent copying)

Параллелизм уровня потоков

Применения

- Web-сервер. Клиент шлет запросы, получает ответы, но одна система допускает одновременную обработку нескольких соединений.
- Многопользовательские игры. Сервер Minecraft позволяет различным игрокам одновременно взаимодействовать с окружением.
- Сборщик мусора. Вы создали объект, попользовались им и забыли. Не задумываясь о том, что где-то там есть сборщик мусора, которые отследит ненужность объекта и переиспользует память под следующий объект.
 - Независимые задачи (parallel marking, parallel sweeping)
 - Одновременные задачи (concurrent marking, concurrent copying)
 - Работоспособность при наличии блокирующих вызовов (поток завис в `epoll_wait`, а мусор всё равно был собран)

Параллелизм уровня потоков

Применения

- Web-сервер. Клиент шлет запросы, получает ответы, но одна система допускает одновременную обработку нескольких соединений.
- Многопользовательские игры. Сервер Minecraft позволяет различным игрокам одновременно взаимодействовать с окружением.
- Сборщик мусора. Вы создали объект, попользовались им и забыли. Не задумываясь о том, что где-то там есть сборщик мусора, которые отследит ненужность объекта и переиспользует память под следующий объект.
 - Независимые задачи (parallel marking, parallel sweeping)
 - Одновременные задачи (concurrent marking, concurrent copying)
 - Работоспособность при наличии блокирующих вызовов (поток завис в `epoll_wait`, а мусор всё равно был собран)

Параллелизм уровня потоков

Выводы

Весьма хорошая и актуальная оптимизация.

Параллелизм уровня потоков

Выводы

Весьма хорошая и актуальная оптимизация.
Способ добиться одновременного исполнения в рамках одного процесса.

Параллелизм уровня потоков

Выводы

Весьма хорошая и актуальная оптимизация.
Способ добиться одновременного исполнения в рамках одного процесса.

- Всё виртуальное адресное пространство – общий ресурс

Параллелизм уровня потоков

Выводы

Весьма хорошая и актуальная оптимизация.
Способ добиться одновременного исполнения в рамках одного процесса.

- Всё виртуальное адресное пространство – общий ресурс
- Все CPU – общий ресурс

Параллелизм уровня потоков

Выводы

Весьма хорошая и актуальная оптимизация.
Способ добиться одновременного исполнения в рамках одного процесса.

- Всё виртуальное адресное пространство – общий ресурс
- Все CPU – общий ресурс
- Быстрая скорость обмена информацией

Параллелизм уровня потоков

Выводы

Весьма хорошая и актуальная оптимизация.
Способ добиться одновременного исполнения в рамках одного процесса.

- Всё виртуальное адресное пространство – общий ресурс
- Все CPU – общий ресурс
- Быстрая скорость обмена информацией
- Возможность совместно использовать данные

Параллелизм уровня потоков

Выводы

Весьма хорошая и актуальная оптимизация.

Способ добиться одновременного исполнения в рамках одного процесса.

- Всё виртуальное адресное пространство – общий ресурс
- Все CPU – общий ресурс
- Быстрая скорость обмена информацией
- Возможность совместно использовать данные

Есть общий ресурс – есть проблемы.

Deadlock, livelock, starvation, priority inversion, lock convoy, thundering herd problem, ABA problem, use-after-free ...

Параллелизм уровня потоков

Выводы

Весьма хорошая и актуальная оптимизация.

Способ добиться одновременного исполнения в рамках одного процесса.

- Всё виртуальное адресное пространство – общий ресурс
- Все CPU – общий ресурс
- Быстрая скорость обмена информацией
- Возможность совместно использовать данные

Есть общий ресурс – есть проблемы.

Deadlock, livelock, starvation, priority inversion, lock convoy, thundering herd problem, ABA problem, use-after-free ...

Сегодня не об этом.

Параллелизм уровня потоков

Выводы

Весьма хорошая и актуальная оптимизация.

Способ добиться одновременного исполнения в рамках одного процесса.

- Всё виртуальное адресное пространство – общий ресурс
- Все CPU – общий ресурс
- Быстрая скорость обмена информацией
- Возможность совместно **изменять** данные

Есть общий ресурс – есть проблемы.

Deadlock, livelock, starvation, priority inversion, lock convoy, thundering herd problem, ABA problem, use-after-free ...

Сегодня не об этом.

Вы находитесь здесь

- 1 Знакомство
- 2 Зачем писать многопоточный код?
- 3 Компилятор хотел как лучше, а получилось...
- 4 Процессор хотел как лучше, а получилось...
- 5 Программист хочет как лучше...
- 6 Language memory models. Примеры из жизни.
- 7 Подведение итогов

Классические оптимизации однопоточного кода

Inventing reads

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

Классические оптимизации однопоточного кода

Inventing reads

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

Имеет ли право компилятор сэкономить регистры и переписать функцию следующим образом?

```
void foo_2() {
    while(true) {
        if (a == 0) break;
        do_something_with(a);
    }
}
```

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

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

```
void foo_3() {
    int tmp = a;
    if (tmp != 0)
        while(true) { do_something_with(tmp); }
}
```

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

x86-64 clang 16.0.0 -O2¹¹

x86-64 gcc 13.1 -O2¹²

```
foo_1:
    push    rbx
    mov     ebx, dword ptr [rip + a]
    test    ebx, ebx
    je      .LBB1_2
.LBB1_1:
    mov     edi, ebx
    call    do_something_with
    jmp     .LBB1_1
.LBB1_2:
    pop     rbx
    ret
```

¹¹<https://godbolt.org/z/99j3erzaE>

¹²<https://godbolt.org/z/fxzGEo1qf>

Классические оптимизации однопоточного кода

Конфликт интересов

- Хотим мощный оптимизирующий компилятор, чтобы наш однопоточный код работал как можно быстрее
- Не хотим, чтобы преобразования программ ломали наш многопоточный код, который улучшает производительность

Классические оптимизации однопоточного кода

Конфликт интересов

- Хотим мощный оптимизирующий компилятор, чтобы наш однопоточный код работал как можно быстрее
- Не хотим, чтобы преобразования программ ломали наш многопоточный код, который улучшает производительность

Классический пример конфликтующих оптимизаций.

Классические оптимизации однопоточного кода

Конфликт интересов

- Хотим мощный оптимизирующий компилятор, чтобы наш однопоточный код работал как можно быстрее
- Не хотим, чтобы преобразования программ ломали наш многопоточный код, который улучшает производительность

Классический пример конфликтующих оптимизаций.

Как бы вы решали данную проблему?

Классические оптимизации однопоточного кода

Конфликт интересов

- Хотим мощный оптимизирующий компилятор, чтобы наш однопоточный код работал как можно быстрее
- Не хотим, чтобы преобразования программ ломали наш многопоточный код, который улучшает производительность

Классический пример конфликтующих оптимизаций.

Как бы вы решали данную проблему?

- Запретить какие-то преобразования (blacklist)
- Разрешить только "правильные" преобразования (whitelist)

Вызовы для авторов языков

Необходимо понять, какие преобразования "подозрительные" и включить нужные ремарки в спецификацию языка.

Вызовы для авторов языков

Необходимо понять, какие преобразования "подозрительные" и включить нужные ремарки в спецификацию языка.

- Как понять, что запреты исчерпывающие?
- Как проверить, что указания непротиворечивы?

Вызовы для авторов языков

Необходимо понять, какие преобразования "подозрительные" и включить нужные ремарки в спецификацию языка.

- Как понять, что запреты исчерпывающие?
- Как проверить, что указания непротиворечивы?

В примере "removing reads" используются довольно простые классические преобразования. Не хотелось бы запрещать слишком много оптимизаций.

Вызовы для авторов языков

Необходимо понять, какие преобразования "подозрительные" и включить нужные ремарки в спецификацию языка.

- Как понять, что запреты исчерпывающие?
- Как проверить, что указания непротиворечивы?

В примере "removing reads" используются довольно простые классические преобразования. Не хотелось бы запрещать слишком много оптимизаций.

Надо написать некоторую спецификацию того, как различные потоки видят изменения разделяемой памяти (переменных, объектов, полей).

Вызовы для авторов языков

Необходимо понять, какие преобразования "подозрительные" и включить нужные ремарки в спецификацию языка.

- Как понять, что запреты исчерпывающие?
- Как проверить, что указания непротиворечивы?

В примере "removing reads" используются довольно простые классические преобразования. Не хотелось бы запрещать слишком много оптимизаций.

Надо написать некоторую спецификацию того, как различные потоки видят изменения разделяемой памяти (переменных, объектов, полей). Соблюсти баланс между *понятностью*, *производительностью* и *устойчивостью*.

Выводы

- Очевидные и верные преобразования однопоточных программ очень часто искажают поведение многопоточного кода, использующего общую память

Выводы




- Очевидные и верные преобразования однопоточных программ очень часто искажают поведение многопоточного кода, использующего общую память
- Если язык программирования не готов радикально отказаться от общего изменяемого состояния^{13,14} или попытался, но не смог¹⁵, то необходимо определить "границы дозволенного" для оптимизаций и не очень сильно удивлять пользователей языка

Выводы

- Очевидные и верные преобразования однопоточных программ очень часто искажают поведение многопоточного кода, использующего общую память
- Если язык программирования не готов радикально отказаться от общего изменяемого состояния^{13,14} или попытался, но не смог¹⁵, то необходимо определить "границы дозволенного" для оптимизаций и не очень сильно удивлять пользователей языка
- Language memory model – описание того, какие есть гарантии у различных потоков приложения при обращении к разделяемым ячейкам памяти (переменным, объектам, полям, элементам массивов ...)

¹³ https://en.wikipedia.org/wiki/Actor_model

¹⁴ https://en.wikipedia.org/wiki/Communicating_sequential_processes

¹⁵ <https://kotlinlang.org/docs/multiplatform-mobile-concurrency-overview.html#global-state>   

Вы находитесь здесь

- 1 Знакомство
- 2 Зачем писать многопоточный код?
- 3 Компилятор хотел как лучше, а получилось...
- 4 Процессор хотел как лучше, а получилось...**
- 5 Программист хочет как лучше...
- 6 Language memory models. Примеры из жизни.
- 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^{16,17}

¹⁶ [https://en.wikipedia.org/wiki/Spectre_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Spectre_(security_vulnerability))

¹⁷ [https://en.wikipedia.org/wiki/Meltdown_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability))

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^{16,17}
 - О предстоящих переходах (branch prediction)

¹⁶ [https://en.wikipedia.org/wiki/Spectre_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Spectre_(security_vulnerability))

¹⁷ [https://en.wikipedia.org/wiki/Meltdown_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability))

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^{16,17}
 - О предстоящих переходах (branch prediction)
 - О требуемой памяти (cache prefetching)

¹⁶ [https://en.wikipedia.org/wiki/Spectre_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Spectre_(security_vulnerability))

¹⁷ [https://en.wikipedia.org/wiki/Meltdown_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability))

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^{16,17}
 - О предстоящих переходах (branch prediction)
 - О требуемой памяти (cache prefetching)
 - О результате вычислений (speculative execution)

¹⁶ [https://en.wikipedia.org/wiki/Spectre_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Spectre_(security_vulnerability))

¹⁷ [https://en.wikipedia.org/wiki/Meltdown_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability))

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^{16,17}
 - О предстоящих переходах (branch prediction)
 - О требуемой памяти (cache prefetching)
 - О результате вычислений (speculative execution)
 - И многом другом

¹⁶ [https://en.wikipedia.org/wiki/Spectre_\(security_vulnerability\)](https://en.wikipedia.org/wiki/Spectre_(security_vulnerability))

¹⁷ [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;
}
```

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

```
# 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

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

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)

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
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 : (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

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?

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

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?

Ответ: (1 1) , (0 1) , (1 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?

Правильный ответ: (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?

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

Процессор может переупорядочить записи и чтения, если это не нарушает intra-thread order.

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 может переупорядочить записи и чтения, если это не нарушает intra-thread order. В данном случае изменился наблюдаемый другими процессорами порядок store и load операций^{18,19,20}.

¹⁸<https://habr.com/ru/company/JetBrains-education/blog/523298/>

¹⁹<https://diy.inria.fr/doc/SB.litmus>

²⁰<https://www.cl.cam.ac.uk/~pes20/weakmemory/cacm.pdf>

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 может переупорядочить записи и чтения, если это не нарушает intra-thread order. В данном случае изменился наблюдаемый другими процессорами порядок store и load операций^{18,19,20}.

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

¹⁸<https://habr.com/ru/company/JetBrains-education/blog/523298/>

¹⁹<https://diy.inria.fr/doc/SB.litmus>

²⁰<https://www.cl.cam.ac.uk/~pes20/weakmemory/cacm.pdf>

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: да²¹

²¹ 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: да²¹

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

²¹ 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: да²¹

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.

²¹ 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: да²¹

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 шкале²².

²¹ A Tutorial Introduction to the ARM and POWER Relaxed Memory Models, section 6.1

²² 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 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 Зачем писать многопоточный код?
- 3 Компилятор хотел как лучше, а получилось...
- 4 Процессор хотел как лучше, а получилось...
- 5 Программист хочет как лучше...**
- 6 Language memory models. Примеры из жизни.
- 7 Подведение итогов

Проблема №1

Компилятор, в погоне за производительностью, начинает "фантазировать":

Проблема №1

Компилятор, в погоне за производительностью, начинает "фантазировать":

- Добавлять операции, которых не было в исходной программе

Проблема №1

Компилятор, в погоне за производительностью, начинает "фантазировать":

- Добавлять операции, которых не было в исходной программе
- Удалять написанные в исходном тексте операции

Проблема №1

Компилятор, в погоне за производительностью, начинает "фантазировать":

- Добавлять операции, которых не было в исходной программе
- Удалять написанные в исходном тексте операции
- Изменять порядок операций

Проблема №1

Компилятор, в погоне за производительностью, начинает "фантазировать":

- Добавлять операции, которых не было в исходной программе
- Удалять написанные в исходном тексте операции
- Изменять порядок операций

Программисту хотелось бы явно сказать "делай в точности как написано".

Проблема №1

Компилятор, в погоне за производительностью, начинает "фантазировать":

- Добавлять операции, которых не было в исходной программе
- Удалять написанные в исходном тексте операции
- Изменять порядок операций

Программисту хотелось бы иметь языковые средства, позволяющие контролировать происходящее.

Compiler barriers

```
int x, y;  
void foo() {  
    x = 1;  
    y = 2;  
    x = 3;  
}
```

Compiler barriers

```
int x, y;  
void foo() {  
    x = 1;  
    y = 2;  
    x = 3;  
}
```

```
foo1:  
    mov [y], 2  
    mov [x], 3  
    ret
```

```
foo2:  
    mov [x], 1  
    mov [y], 2  
    mov [x], 3  
    ret
```

Compiler barriers

```
int x, y;  
void foo() {  
    x = 1;  
    y = 2;  
    x = 3;  
}
```

```
foo1:  
    mov [y], 2  
    mov [x], 3  
    ret
```

```
foo2:  
    mov [x], 1  
    mov [y], 2  
    mov [x], 3  
    ret
```

Современный оптимизирующий компилятор выберет вариант слева²³.

²³ <https://godbolt.org/z/r6sMzrj6K>

Compiler barriers

```
int x, y;
void foo() {
    x = 1;
    y = 2;
    x = 3;
}
```

```
foo1:
    mov [y], 2
    mov [x], 3
    ret
```

```
foo2:
    mov [x], 1
    mov [y], 2
    mov [x], 3
    ret
```

Современный оптимизирующий компилятор выберет вариант слева²³. Но можно ему сказать – "вот эта точка в программе – барьер, она запрещает двигать операции через него".

²³ <https://godbolt.org/z/r6sMzrj6K>

Compiler barriers

```
int x, y;  
void foo1() {  
    x = 1;  
    y = 2;  
    x = 3;  
}
```

```
int x, y;  
void foo2() {  
    x = 1;  
    barrier();  
    y = 2;  
    x = 3;  
}
```

Compiler barriers

```
int x, y;  
void foo1() {  
    x = 1;  
    y = 2;  
    x = 3;  
}
```

```
foo1:  
    mov [y], 2  
    mov [x], 3  
    ret
```

```
int x, y;  
void foo2() {  
    x = 1;  
    barrier();  
    y = 2;  
    x = 3;  
}
```

```
foo2:  
    mov [x], 1  
    mov [y], 2  
    mov [x], 3  
    ret
```


Compiler barriers

```
int x, y;
void foo1() {
    x = 1;
    y = 2;
    x = 3;
}
```

```
foo1:
    mov [y], 2
    mov [x], 3
    ret
```

```
int x, y;
void foo2() {
    x = 1;
    barrier();
    y = 2;
    x = 3;
}
```

```
foo2:
    mov [x], 1
    mov [y], 2
    mov [x], 3
    ret
```

Обратите внимание, что это **ОЧЕНЬ** низкоуровневый механизм и при написании современного C/C++ кода его использовать не следует²⁴.

²⁴<https://preshing.com/20120625/memory-ordering-at-compile-time/>

Проблема №2

Процессор, в погоне за производительностью, начинает "чудить":

Проблема №2

Процессор, в погоне за производительностью, начинает "чудить":

- С разной скоростью передавать информацию другим процессорам

Проблема №2

Процессор, в погоне за производительностью, начинает "чудить":

- С разной скоростью передавать информацию другим процессорам
- Изменять порядок операций

Проблема №2

Процессор, в погоне за производительностью, начинает "чудить":

- С разной скоростью передавать информацию другим процессорам
- Изменять порядок операций

Программисту хотелось бы иметь языковые средства, позволяющие контролировать происходящее.

Проблема №2

Процессор, в погоне за производительностью, начинает "чудить":

- С разной скоростью передавать информацию другим процессорам
- Изменять порядок операций

Программисту хотелось бы иметь языковые средства, позволяющие контролировать происходящее.

В многоядерном процессоре предусмотрены специальные инструкции, которые позволяют установить порядок "видимости" среди операций чтения и записи²⁵.

²⁵ https://en.wikipedia.org/wiki/Memory_barrier

Memory barriers

Пример

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

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

Допустимые результаты:

Memory barriers

Пример

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

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

Допустимые результаты: (0, 0) (1, 0) (0, 1) (1, 1)

Memory barriers

Пример

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

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

Допустимые результаты: (0, 0) (1, 0) (0, 1) (1, 1)

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

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

Memory barriers

Пример

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

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

Допустимые результаты: (0, 0) (1, 0) (0, 1) (1, 1)

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

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

Инструкция `mfence` отменяет переупорядочивание операций с памятью на уровне процессора²⁶.

²⁶ <https://www.felixcloutier.com/x86/mfence.html>

Memory barriers

Пример

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

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

Допустимые результаты: (0, 0) (1, 0) (0, 1) (1, 1)

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

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

Инструкция `mfence` отменяет переупорядочивание операций с памятью на уровне процессора²⁶.

Допустимые результаты:

²⁶ <https://www.felixcloutier.com/x86/mfence.html>

Memory barriers

Пример

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

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

Допустимые результаты: (0, 0) (1, 0) (0, 1) (1, 1)

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

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

Инструкция `mfence` отменяет переупорядочивание операций с памятью на уровне процессора²⁶.

Допустимые результаты: (1, 0) (0, 1) (1, 1)

²⁶ <https://www.felixcloutier.com/x86/mfence.html>

Memory barriers

Предостережение

Не забывайте

Memory barriers

Предостережение

Не забывайте

- Барьеры дорогие

Memory barriers

Предостережение

Не забывайте

- Барьеры дорогие
- Инструкции обладают процессорно-специфичной и иногда весьма запутанной семантикой

Memory barriers

Предостережение

Не забывайте

- Барьеры дорогие
- Инструкции обладают процессорно-специфичной и иногда весьма запутанной семантикой
- Расстановка барьеров в ряде многопоточных алгоритмов – не единственная и, по сути, является смесью искусства и инженерного мастерства

Memory barriers

Предостережение

Не забывайте

- Барьеры дорогие
- Инструкции обладают процессорно-специфичной и иногда весьма запутанной семантикой
- Расстановка барьеров в ряде многопоточных алгоритмов – не единственная и, по сути, является смесью искусства и инженерного мастерства
- Практически все, кто так делает, похожи на глотателей огня. Даже если живы, то со шрамами от ожогов²⁷.

²⁷ https://www.researchgate.net/publication/228824849_Memory_Barriers_a_Hardware_View_for_Software_Hackers

Языковые средства

Вопросы взаимного влияния

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Языковые средства

Вопросы взаимного влияния

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Например, с помощью барьеров разного вида.

Языковые средства

Вопросы взаимного влияния

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Например, с помощью барьеров разного вида.

Кажется, что компиляторные и процессорные барьеры – это разные сущности и, наверное, независимые друг от друга.

Языковые средства

Вопросы взаимного влияния

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Например, с помощью барьеров разного вида.

Кажется, что компиляторные и процессорные барьеры – это разные сущности и, наверное, независимые друг от друга.

Однако, кажется логичным, чтобы компилятор уважал процессорные барьеры.

Языковые средства

Вопросы взаимного влияния

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Например, с помощью барьеров разного вида.

Кажется, что компиляторные и процессорные барьеры – это разные сущности и, наверное, независимые друг от друга.

Однако, кажется логичным, чтобы компилятор уважал процессорные барьеры.

А наоборот?

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Существуют разные подходы:

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Существуют разные подходы:

- Языки, с маниакальной страстью пытающиеся дать разработчикам способы написать очень быструю, но некорректную программу.

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Существуют разные подходы:

- Языки, с маниакальной страстью пытающиеся дать разработчикам способы написать очень быструю, но некорректную программу. C/C++

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Существуют разные подходы:

- Языки, с маниакальной страстью пытающиеся дать разработчикам способы написать очень быструю, но некорректную программу. C/C++
- Языки, сфокусированные на безопасности и с помощью управляемой среды создающие "песочницу".

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Существуют разные подходы:

- Языки, с маниакальной страстью пытающиеся дать разработчикам способы написать очень быструю, но некорректную программу. C/C++
- Языки, сфокусированные на безопасности и с помощью управляемой среды создающие "песочницу". Java

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Существуют разные подходы:

- Языки, с маниакальной страстью пытающиеся дать разработчикам способы написать очень быструю, но некорректную программу. C/C++
- Языки, сфокусированные на безопасности и с помощью управляемой среды создающие "песочницу". Java

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Существуют разные подходы:

- Языки, с маниакальной страстью пытающиеся дать разработчикам способы написать очень быструю, но некорректную программу. C/C++
- Языки, сфокусированные на безопасности и с помощью управляемой среды **пытающиеся создать** "песочницу". Java

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Существуют разные подходы:

- Языки, с маниакальной страстью пытающиеся дать разработчикам способы написать очень быструю, но некорректную программу. C/C++
- Языки, сфокусированные на безопасности и с помощью управляемой среды **пытающиеся создать** "песочницу". Java

Языковые средства

Два мира

Краткая выжимка из предыдущих слайдов. Для "починки" многопоточных алгоритмов бывает необходимо:

- запрещать компилятору "фантазировать"
- запрещать процессору "чудить"

Нужен ли программисту независимый контроль над обеими проблемами, с учетом того, что допустить ошибку невероятно легко, а негативные последствия практически невозможно отладить?

Существуют разные подходы:

- Языки, предоставляющие разработчикам документированное и низкоуровневое API для написания очень быстрых программ. C/C++
- Языки, сфокусированные на безопасности и с помощью управляемой среды пытающиеся создать "песочницу". Java

Разные миры

Пример

```
static volatile int x, y;
```

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

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

Разные миры

Пример

```
static volatile int x, y;
```

```
void threadA() {
    x = 1;
    int a = y;
}
C28:
```

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

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

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

²⁸ <https://godbolt.org/z/q4raxqrTe>

Разные миры

Пример

```
static volatile int x, y;
```

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

C²⁸:

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

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

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

Допустимые результаты: (0, 0) (1, 0) (0, 1) (1, 1)

²⁸<https://godbolt.org/z/q4raxqrTe>

Разные миры

Пример

```
static volatile int x, y;
```

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

Java²⁹:

```
MOV [x] , 1           # (A.1)
lock add [rsp], 0x0    # (A.2)
MOV EAX , [y]         # (A.3)
```

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

```
MOV [y] , 1           # (B.1)
lock add [rsp], 0x0    # (B.2)
MOV EBX , [x]         # (B.3)
```

²⁹<https://github.com/Svazars/lang-mem-models-intro/tree/main/samples/java>

Разные миры

Пример

```
static volatile int x, y;
```

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

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

Java²⁹:

```
MOV [x] , 1           # (A.1)
lock add [rsp], 0x0    # (A.2)
MOV EAX , [y]         # (A.3)
```

```
MOV [y] , 1           # (B.1)
lock add [rsp], 0x0    # (B.2)
MOV EBX , [x]         # (B.3)
```

`lock add [rsp], 0x0` \approx `mfence` \approx full memory barrier³⁰

²⁹<https://github.com/Svazars/lang-mem-models-intro/tree/main/samples/java>

³⁰<https://shipilev.net/blog/2014/on-the-fence-with-dependencies/>

Разные миры

Пример

```
static volatile int x, y;
```

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

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

Java²⁹:

```
MOV [x] , 1           # (A.1)
lock add [rsp], 0x0    # (A.2)
MOV EAX , [y]         # (A.3)
```

```
MOV [y] , 1           # (B.1)
lock add [rsp], 0x0    # (B.2)
MOV EBX , [x]         # (B.3)
```

`lock add [rsp], 0x0` \approx `mfence` \approx full memory barrier³⁰

Допустимые результаты: (1, 0) (0, 1) (1, 1)

²⁹<https://github.com/Svazars/lang-mem-models-intro/tree/main/samples/java>

³⁰<https://shipilev.net/blog/2014/on-the-fence-with-dependencies/>

Разные миры

Пример

```
static volatile int x, y;
```

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

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

Допустимые результаты в C: (0, 0) (1, 0) (0, 1) (1, 1)

Допустимые результаты в Java: (1, 0) (0, 1) (1, 1)

Разные миры

Пример

```
static volatile int x, y;
```

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

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

Допустимые результаты в C: (0, 0) (1, 0) (0, 1) (1, 1)

Допустимые результаты в Java: (1, 0) (0, 1) (1, 1)

Обычно пишут, что `volatile` в языке Си ограничивает только преобразования на уровне компилятора, а в языке Java – ограничивает как компилятор, так процессор.

Разные миры

Пример

```
static volatile int x, y;
```

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

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

Допустимые результаты в C: (0, 0) (1, 0) (0, 1) (1, 1)

Допустимые результаты в Java: (1, 0) (0, 1) (1, 1)

Обычно пишут, что `volatile` в языке Си ограничивает только преобразования на уровне компилятора, а в языке Java – ограничивает как компилятор, так процессор.

Но давать слишком простую модель реального мира – плохо с педагогической точки зрения.

Разные миры

Пример

```
static volatile int x, y;
```

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

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

Допустимые результаты в C: (0, 0) (1, 0) (0, 1) (1, 1)

Допустимые результаты в Java: (1, 0) (0, 1) (1, 1)

Обычно пишут, что `volatile` в языке Си ограничивает только преобразования на уровне компилятора, а в языке Java – ограничивает как компилятор, так процессор.

Но давать слишком простую модель реального мира – плохо с педагогической точки зрения.

Поэтому защищу себя от гнева ваших будущих работодателей.

Совет от мудрой совы

В языке C вместо `volatile` **всегда** используйте типы из `stdatomic.h`³¹ для разделяемых переменных.

³¹<https://en.cppreference.com/w/c/language/atomic>

Совет от мудрой совы

В языке C вместо `volatile` **всегда** используйте типы из `stdatomic.h`³¹ для разделяемых переменных.

Если стандарт C11 не поддерживается используемым компилятором или в коде обычные переменные изменяются из разных потоков одновременно – бегите³².

³¹<https://en.cppreference.com/w/c/language/atomic>

³²<https://www.kernel.org/doc/Documentation/process/volatile-considered-harmful.rst>

Совет от мудрой совы

В языке C вместо `volatile` **всегда** используйте типы из `stdatomic.h`³¹ для разделяемых переменных.

Если стандарт C11 не поддерживается используемым компилятором или в коде обычные переменные изменяются из разных потоков одновременно – бегите³².

Для любителей почитать как ругается Линус Торвалдс, советую обратить внимание на цепочку обсуждений "volatile considered evil"³³:

"volatile" really _is_ misdesigned. The semantics of it are so unclear as to be totally useless. The only thing "volatile" can ever do is generate worse code, WITH NO UPSIDES.

³¹<https://en.cppreference.com/w/c/language/atomic>

³²<https://www.kernel.org/doc/Documentation/process/volatile-considered-harmful.rst>

³³<https://lkml.org/lkml/2006/7/6/159>

Языковые средства написания работающих многопоточных программ

Выводы

Языковые средства написания работающих многопоточных программ

Выводы

- Пожалуйста, никогда и никому не говорите что `volatile` в C и в Java имеют один и тот же смысл

Языковые средства написания работающих многопоточных программ

Выводы

- Пожалуйста, никогда и никому не говорите что `volatile` в C и в Java имеют один и тот же смысл
- Не думайте, что бездумное добавление `volatile` в вашу программу сделает её корректной (даже на Java это не так)

Языковые средства написания работающих многопоточных программ

Выводы

- Пожалуйста, никогда и никому не говорите что `volatile` в C и в Java имеют один и тот же смысл
- Не думайте, что бездумное добавление `volatile` в вашу программу сделает её корректной (даже на Java это не так)
- Всегда помните, что в разных языках "одинаковая" конструкция может иметь весьма разный смысл
 - смена языкового стека
 - адаптация алгоритма из библиотеки/публикации
 - кросс-языковая трансляция

Языковые средства написания работающих многопоточных программ

Выводы

- Пожалуйста, никогда и никому не говорите что `volatile` в C и в Java имеют один и тот же смысл
- Не думайте, что бездумное добавление `volatile` в вашу программу сделает её корректной (даже на Java это не так)
- Всегда помните, что в разных языках "одинаковая" конструкция может иметь весьма разный смысл
 - смена языкового стека
 - адаптация алгоритма из библиотеки/публикации
 - кросс-языковая трансляция
- Компиляторные и процессорные барьеры – прямолинейный способ добиться желаемого при написании многопоточных программ

Языковые средства написания работающих многопоточных программ

Выводы

- Пожалуйста, никогда и никому не говорите что `volatile` в C и в Java имеют один и тот же смысл
- Не думайте, что бездумное добавление `volatile` в вашу программу сделает её корректной (даже на Java это не так)
- Всегда помните, что в разных языках "одинаковая" конструкция может иметь весьма разный смысл
 - смена языкового стека
 - адаптация алгоритма из библиотеки/публикации
 - кросс-языковая трансляция
- Компиляторные и процессорные барьеры – прямолинейный способ добиться желаемого при написании многопоточных программ
- Барьерный подход несет с собой много неявной сложности и специфики (языка, компилятора, процессора)

Вы находитесь здесь

- 1 Знакомство
- 2 Зачем писать многопоточный код?
- 3 Компилятор хотел как лучше, а получилось...
- 4 Процессор хотел как лучше, а получилось...
- 5 Программист хочет как лучше...
- 6 Language memory models. Примеры из жизни.**
- 7 Подведение итогов

Есть ли более простые решения?

Я простой Java-программист, я не хочу даже думать о тысячах оптимизаций, которыми компилятор может сломать мою программу.

Есть ли более простые решения?

Я простой Java-программист, я не хочу даже думать о тысячах оптимизаций, которыми компилятор может сломать мою программу. Также я не хочу читать тысячи страниц спецификации процессорной архитектуры, чтобы расставлять какие-то барьеры.

Есть ли более простые решения?

Я простой Java-программист, я не хочу даже думать о тысячах оптимизаций, которыми компилятор может сломать мою программу. Также я не хочу читать тысячи страниц спецификации процессорной архитектуры, чтобы расставлять какие-то барьеры.

Intel® 64 and IA-32 Architectures Software Developer's Manual:

- Volume 1: Basic Architecture (458 страниц)
- Volume 2: Instruction Set Reference, A-Z (1513 страниц)
- Volume 3: System Programming Guide (1638 страниц)

Есть ли более простые решения?

Я простой Java-программист, я не хочу даже думать о тысячах оптимизаций, которыми компилятор может сломать мою программу. Также я не хочу читать тысячи страниц спецификации процессорной архитектуры, чтобы расставлять какие-то барьеры.

Intel® 64 and IA-32 Architectures Software Developer's Manual:

- Volume 1: Basic Architecture (458 страниц)
- Volume 2: Instruction Set Reference, A-Z (1513 страниц)
- Volume 3: System Programming Guide (1638 страниц)

Размеры мануала по ARM64 (v8, для конкретики) сами найдите.

Есть ли более простые решения?

Я простой Java-программист, я не хочу даже думать о тысячах оптимизаций, которыми компилятор может сломать мою программу. Также я не хочу читать тысячи страниц спецификации процессорной архитектуры, чтобы расставлять какие-то барьеры. Хочу кросс-платформенный код писать и чтобы он был понятный, простой и поддерживаемый.

Есть ли более простые решения?

Я простой Java-программист, я не хочу даже думать о тысячах оптимизаций, которыми компилятор может сломать мою программу. Также я не хочу читать тысячи страниц спецификации процессорной архитектуры, чтобы расставлять какие-то барьеры.

Хочу кросс-платформенный код писать и чтобы он был понятный, простой и поддерживаемый.

Требуется описание операций с памятью и их свойств в используемом языке программирования

Есть ли более простые решения?

Я простой Java-программист, я не хочу даже думать о тысячах оптимизаций, которыми компилятор может сломать мою программу. Также я не хочу читать тысячи страниц спецификации процессорной архитектуры, чтобы расставлять какие-то барьеры.

Хочу кросс-платформенный код писать и чтобы он был понятный, простой и поддерживаемый.

Требуется описание операций с памятью и их свойств в используемом языке программирования

- Независимое от платформы (ОС, процессорная архитектура)
- Независимое от среды исполнения (компилятор, сборщик мусора)
- Независимое от версии языка³⁴

³⁴Недостижимый идеал или суровая действительность обратной совместимости? 

Есть ли более простые решения?

Я простой Java-программист, я не хочу даже думать о тысячах оптимизаций, которыми компилятор может сломать мою программу. Также я не хочу читать тысячи страниц спецификации процессорной архитектуры, чтобы расставлять какие-то барьеры.

Хочу кросс-платформенный код писать и чтобы он был понятный, простой и поддерживаемый.

Требуется описание операций с памятью и их свойств в используемом языке программирования

- Независимое от платформы (ОС, процессорная архитектура)
- Независимое от среды исполнения (компилятор, сборщик мусора)
- Независимое от версии языка³⁴

Требуется language memory model.

³⁴Недостижимый идеал или суровая действительность обратной совместимости? 

Language Memory Model

- Как писать такой документ? Литературным английским? В виде алгоритма? В виде набора разрешающих правил? В виде набора запрещающих правил?

Language Memory Model

- Как писать такой документ? Литературным английским? В виде алгоритма? В виде набора разрешающих правил? В виде набора запрещающих правил?
- Должна ли спецификация меняться от версии к версии?

Language Memory Model

- Как писать такой документ? Литературным английским? В виде алгоритма? В виде набора разрешающих правил? В виде набора запрещающих правил?
- Должна ли спецификация меняться от версии к версии?
- Лучше чтобы она была более строгой или более слабой?

Language Memory Model

- Как писать такой документ? Литературным английским? В виде алгоритма? В виде набора разрешающих правил? В виде набора запрещающих правил?
- Должна ли спецификация меняться от версии к версии?
- Лучше чтобы она была более строгой или более слабой?
- Можно ли проверить что придуманные правила согласованы между собой?

Language Memory Model

- Как писать такой документ? Литературным английским? В виде алгоритма? В виде набора разрешающих правил? В виде набора запрещающих правил?
- Должна ли спецификация меняться от версии к версии?
- Лучше чтобы она была более строгой или более слабой?
- Можно ли проверить что придуманные правила согласованы между собой?
- Можно ли гарантировать, что любая программа будет адекватно и однозначно описываться придуманной моделью?

Language Memory Model

- Как писать такой документ? Литературным английским? В виде алгоритма? В виде набора разрешающих правил? В виде набора запрещающих правил?
- Должна ли спецификация меняться от версии к версии?
- Лучше чтобы она была более строгой или более слабой?
- Можно ли проверить что придуманные правила согласованы между собой?
- Можно ли гарантировать, что любая программа будет адекватно и однозначно описываться придуманной моделью?
- Существует ли какой-то специальный математический аппарат, облегчающий написание спецификации?

Language Memory Model

- Как писать такой документ? Литературным английским? В виде алгоритма? В виде набора разрешающих правил? В виде набора запрещающих правил?
- Должна ли спецификация меняться от версии к версии?
- Лучше чтобы она была более строгой или более слабой?
- Можно ли проверить что придуманные правила согласованы между собой?
- Можно ли гарантировать, что любая программа будет адекватно и однозначно описываться придуманной моделью?
- Существует ли какой-то специальный математический аппарат, облегчающий написание спецификации?
- А пользователи языка должны смочь это прочесть? Понять? Применить на практике?

Language Memory Model

- Как писать такой документ? Литературным английским? В виде алгоритма? В виде набора разрешающих правил? В виде набора запрещающих правил?
- Должна ли спецификация меняться от версии к версии?
- Лучше чтобы она была более строгой или более слабой?
- Можно ли проверить что придуманные правила согласованы между собой?
- Можно ли гарантировать, что любая программа будет адекватно и однозначно описываться придуманной моделью?
- Существует ли какой-то специальный математический аппарат, облегчающий написание спецификации?
- А пользователи языка должны смочь это прочесть? Понять? Применить на практике?

Соблюсти баланс между *понятностью*, *производительностью* и *устойчивостью*

О практической пользе данной лекции

На слайде 39 самое время задаться именно таким вопросом.

О практической пользе данной лекции

С точки зрения большинства прикладных программистов, модель памяти не нужна.

О практической пользе данной лекции

С точки зрения большинства прикладных программистов, модель памяти не нужна.

С другой стороны, большинство программистов, согласно опросам из интернета, пишет на таких языках как Python, JavaScript, VisualBasic, PHP.

О практической пользе данной лекции

~~С точки зрения большинства прикладных программистов, модель памяти не нужна.~~

О практической пользе данной лекции

~~С точки зрения большинства прикладных программистов, модель памяти не нужна.~~

С точки зрения большинства прикладных программистов, модель памяти не должна мешать «делать дело».

О практической пользе данной лекции

~~С точки зрения большинства прикладных программистов, модель памяти не нужна.~~

С точки зрения большинства прикладных программистов, модель памяти не должна мешать «делать дело».

А если случается необъяснимая бесовщина, то Senior Software Engineer «придёт и молча поправит всё».

О практической пользе данной лекции

~~С точки зрения большинства прикладных программистов, модель памяти не нужна.~~

С точки зрения большинства прикладных программистов, модель памяти не должна мешать «делать дело».

А если случается необъяснимая бесовщина, то Senior Software Engineer «придёт и молча поправит всё».

Глупо скрывать от вас тот факт, что материал про модели памяти – узкоспециализированный.

О практической пользе данной лекции

~~С точки зрения большинства прикладных программистов, модель памяти не нужна.~~

С точки зрения большинства прикладных программистов, модель памяти не должна мешать «делать дело».

А если случается необъяснимая бесовщина, то Senior Software Engineer «придёт и молча поправит всё».

Глупо скрывать от вас тот факт, что материал про модели памяти – узкоспециализированный.

С другой стороны, вы уже не доверяете компилятору и процессору.

О практической пользе данной лекции

~~С точки зрения большинства прикладных программистов, модель памяти не нужна.~~

С точки зрения большинства прикладных программистов, модель памяти не должна мешать «делать дело».

А если случается необъяснимая бесовщина, то Senior Software Engineer «придёт и молча поправит всё».

Глупо скрывать от вас тот факт, что материал про модели памяти – узкоспециализированный.

С другой стороны, вы уже не доверяете компилятору и процессору. Осталось совсем немного.

О практической пользе данной лекции

~~С точки зрения большинства прикладных программистов, модель памяти не нужна.~~

С точки зрения большинства прикладных программистов, модель памяти не должна мешать «делать дело».

А если случается необъяснимая бесовщина, то Senior Software Engineer «придёт и молча поправит всё».

Глупо скрывать от вас тот факт, что материал про модели памяти – узкоспециализированный.

С другой стороны, вы уже не доверяете компилятору и процессору. Осталось совсем немного.

Потеряйте веру в людей ;)

О практической пользе данной лекции

~~С точки зрения большинства прикладных программистов, модель памяти не нужна.~~

С точки зрения большинства прикладных программистов, модель памяти не должна мешать «делать дело».

А если случается необъяснимая бесовщина, то Senior Software Engineer «придёт и молча поправит всё».

Глупо скрывать от вас тот факт, что материал про модели памяти – узкоспециализированный.

С другой стороны, вы уже не доверяете компилятору и процессору. Осталось совсем немного.

Потеряйте веру в людей дизайнеров языков программирования.

Holy war warning

Следующие несколько слайдов могут бросить тень на ваш любимый язык программирования.

Holy war warning

Следующие несколько слайдов могут бросить тень на ваш любимый язык программирования.

Каждый из упомянутых промышленных языков программирования имеет спецификацию и, в том числе, описание модели памяти. Я педантично приведу соответствующие ссылки. Но продолжу говорить, что у некоторых языков нет модели памяти.

Holy war warning

Следующие несколько слайдов могут бросить тень на ваш любимый язык программирования.

Каждый из упомянутых промышленных языков программирования имеет спецификацию и, в том числе, описание модели памяти. Я педантично приведу соответствующие ссылки. Но продолжу говорить, что у некоторых языков нет **вменяемой** модели памяти.

Holy war warning

Следующие несколько слайдов могут бросить тень на ваш любимый язык программирования.

Каждый из упомянутых промышленных языков программирования имеет спецификацию и, в том числе, описание модели памяти. Я педантично приведу соответствующие ссылки. Но продолжу говорить, что у некоторых языков нет **вменяемой** модели памяти.

Порядок упоминания языков не соответствует «качеству» языка, просто так мне проще выстроить повествование.

Holy war warning

Следующие несколько слайдов могут бросить тень на ваш любимый язык программирования.

Каждый из упомянутых промышленных языков программирования имеет спецификацию и, в том числе, описание модели памяти. Я педантично приведу соответствующие ссылки. Но продолжу говорить, что у некоторых языков нет **вменяемой** модели памяти.

Порядок упоминания языков не соответствует «качеству» языка, просто так мне проще выстроить повествование.

Поехали.

Существующие подходы к описанию моделей памяти

Нет человека – нет проблемы

Если в программе нет data race – то нет необходимости говорить о модели памяти для многопоточных сред.

Существующие подходы к описанию моделей памяти

Нет человека – нет проблемы

Если в программе нет data race – то нет необходимости говорить о модели памяти для многопоточных сред.

Поэтому можно использовать *правильные* подходы к программированию:

Существующие подходы к описанию моделей памяти

Нет человека – нет проблемы

Если в программе нет data race – то нет необходимости говорить о модели памяти для многопоточных сред.

Поэтому можно использовать *правильные* подходы к программированию:

- Неизменяемые структуры данных

Существующие подходы к описанию моделей памяти

Нет человека – нет проблемы

Если в программе нет data race – то нет необходимости говорить о модели памяти для многопоточных сред.

Поэтому можно использовать *правильные* подходы к программированию:

- Неизменяемые структуры данных
- Декларативное описание вычислений

Существующие подходы к описанию моделей памяти

Нет человека – нет проблемы

Если в программе нет data race – то нет необходимости говорить о модели памяти для многопоточных сред.

Поэтому можно использовать *правильные* подходы к программированию:

- Неизменяемые структуры данных
- Декларативное описание вычислений

Clojure

Существующие подходы к описанию моделей памяти

Нет человека – нет проблемы

Если в программе нет data race – то нет необходимости говорить о модели памяти для многопоточных сред.

Поэтому можно использовать *правильные* подходы к программированию:

- Неизменяемые структуры данных
- Декларативное описание вычислений

Clojure

Haskell

Существующие подходы к описанию моделей памяти

Нет человека – нет проблемы

Если в программе нет data race – то нет необходимости говорить о модели памяти для многопоточных сред.

Поэтому можно использовать *правильные* подходы к программированию:

- Неизменяемые структуры данных
- Декларативное описание вычислений

Clojure

Haskell

All told, a monad in X is just a monoid in the category of endofunctors of X , with product \times replaced by composition of endofunctors and unit set by the identity endofunctor.

Существующие подходы к описанию моделей памяти

Нет человека – нет проблемы

Если в программе нет data race – то нет необходимости говорить о модели памяти для многопоточных сред.

Поэтому можно использовать *правильные* подходы к программированию:

- Неизменяемые структуры данных
- Декларативное описание вычислений

Clojure

Haskell

All told, a monad in X is just a monoid in the category of endofunctors of X , with product \times replaced by composition of endofunctors and unit set by the identity endofunctor.

Дело в языке или в технике программирования?

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

- разные потоки могут одновременно наблюдать её (читать из разделяемых ячеек памяти)

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

- разные потоки могут одновременно наблюдать её (читать из разделяемых ячеек памяти)
- невозможны конфликтующие операции

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

- разные потоки могут одновременно наблюдать её (читать из разделяемых ячеек памяти)
- невозможны конфликтующие операции

Невероятно удобно!

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

- разные потоки могут одновременно наблюдать её (читать из разделяемых ячеек памяти)
- невозможны конфликтующие операции

Невероятно удобно!

Но есть нюанс...

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

- разные потоки могут одновременно наблюдать её (читать из разделяемых ячеек памяти)
- невозможны конфликтующие операции

Невероятно удобно!

Но есть нюанс...

А как обновлять такую структуру данных в связи с изменением внешних условий?

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

- разные потоки могут одновременно наблюдать её (читать из разделяемых ячеек памяти)
- невозможны конфликтующие операции

Невероятно удобно!

Но есть нюанс...

А как обновлять такую структуру данных в связи с изменением внешних условий?

Создать новый неизменяемый экземпляр, который содержит самую актуальную информацию.

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

- разные потоки могут одновременно наблюдать её (читать из разделяемых ячеек памяти)
- невозможны конфликтующие операции

Невероятно удобно!

Но есть нюанс...

А как обновлять такую структуру данных в связи с изменением внешних условий?

Создать новый неизменяемый экземпляр, который содержит самую актуальную информацию.

Возникают определенные трудности:

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

- разные потоки могут одновременно наблюдать её (читать из разделяемых ячеек памяти)
- невозможны конфликтующие операции

Невероятно удобно!

Но есть нюанс...

А как обновлять такую структуру данных в связи с изменением внешних условий?

Создать новый неизменяемый экземпляр, который содержит самую актуальную информацию.

Возникают определенные трудности:

- Издержки на пересоздание крупных структур

Существующие подходы к описанию моделей памяти

Immutability

Если структура данных неизменяема – то

- разные потоки могут одновременно наблюдать её (читать из разделяемых ячеек памяти)
- невозможны конфликтующие операции

Невероятно удобно!

Но есть нюанс...

А как обновлять такую структуру данных в связи с изменением внешних условий?

Создать новый неизменяемый экземпляр, который содержит самую актуальную информацию.

Возникают определенные трудности:

- Издержки на пересоздание крупных структур
- Операция публикации – это же, все-таки, операция записи, не так ли?

Существующие подходы к описанию моделей памяти

Declarative DSL

Описать требуемый результат, а система сама разберется, как воспользоваться параллелизмом при организации вычислений:

Существующие подходы к описанию моделей памяти

Declarative DSL

Описать требуемый результат, а система сама разберется, как воспользоваться параллелизмом при организации вычислений:

- OpenMP <https://www.openmp.org/>
- Intel TBB <https://github.com/oneapi-src/oneTBB>
- MPI <https://www.open-mpi.org/>
- Java parallel streams
<https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html>
- MapReduce <https://research.google/pubs/pub62/>
- Resilient Distributed Datasets <https://dl.acm.org/doi/10.5555/2228298.2228301>

Существующие подходы к описанию моделей памяти

Declarative DSL

Описать требуемый результат, а система сама разберется, как воспользоваться параллелизмом при организации вычислений:

- OpenMP <https://www.openmp.org/>
- Intel TBB <https://github.com/oneapi-src/oneTBB>
- MPI <https://www.open-mpi.org/>
- Java parallel streams
<https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html>
- MapReduce <https://research.google/pubs/pub62/>
- Resilient Distributed Datasets <https://dl.acm.org/doi/10.5555/2228298.2228301>

Абстракции текут³⁵ и в большинстве случаев вносят издержки.

³⁵<https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/>

Существующие подходы к описанию моделей памяти

Declarative DSL

Описать требуемый результат, а система сама разберется, как воспользоваться параллелизмом при организации вычислений:

- OpenMP <https://www.openmp.org/>
- Intel TBB <https://github.com/oneapi-src/oneTBB>
- MPI <https://www.open-mpi.org/>
- Java parallel streams
<https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html>
- MapReduce <https://research.google/pubs/pub62/>
- Resilient Distributed Datasets <https://dl.acm.org/doi/10.5555/2228298.2228301>

Абстракции текут³⁵ и в большинстве случаев вносят издержки. При реализации таких DSL неизбежно нужно записывать значения в разделяемые ячейки памяти, не так ли?

³⁵<https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/>

Существующие подходы к описанию моделей памяти

Альтернативные подходы

Модель программирования без (явного) разделяемого состояния.

Существующие подходы к описанию моделей памяти

Альтернативные подходы

Модель программирования без (явного) разделяемого состояния.

- Communicating sequential processes
- Actor model

Существующие подходы к описанию моделей памяти

Альтернативные подходы

Модель программирования без (явного) разделяемого состояния.

- Communicating sequential processes
- Actor model

Все вычислительные агенты (легковесные процессы) независимы друг от друга и обмениваются неизменяемыми сообщениями.

Существующие подходы к описанию моделей памяти

Альтернативные подходы

Модель программирования без (явного) разделяемого состояния.

- Communicating sequential processes
- Actor model

Все вычислительные агенты (легковесные процессы) независимы друг от друга и обмениваются неизменяемыми сообщениями.

- Erlang³⁶
- Akka actors³⁷

³⁶ <https://www.erlang.org/>

³⁷ <https://doc.akka.io/docs/akka/current/typed/actors.html#akka-actors>

Существующие подходы к описанию моделей памяти

Альтернативные подходы

Модель программирования без (явного) разделяемого состояния.

- Communicating sequential processes
- Actor model

Все вычислительные агенты (легковесные процессы) независимы друг от друга и обмениваются неизменяемыми сообщениями.

- Erlang³⁶
- Akka actors³⁷

Код реализации таких систем (Erlang VM, Akka internals) использует разделяемые ячейки памяти, чтобы передавать информацию от одного агента к другому, не так ли?

³⁶ <https://www.erlang.org/>

³⁷ <https://doc.akka.io/docs/akka/current/typed/actors.html#akka-actors>

Существующие подходы к описанию моделей памяти

Наивные подходы: запретить и не пущать

Swift³⁸

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.

³⁸ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Существующие подходы к описанию моделей памяти

Наивные подходы: запретить и не пущать

Swift³⁸

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.

³⁸ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Существующие подходы к описанию моделей памяти

Наивные подходы: запретить и не пущать

Swift³⁸

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

³⁸ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Существующие подходы к описанию моделей памяти

Наивные подходы: запретить и не пущать

Swift³⁸

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() }
```

³⁸ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Существующие подходы к описанию моделей памяти

Наивные подходы: запретить и не пущать

Swift³⁸

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.

³⁸ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

Существующие подходы к описанию моделей памяти

Наивные подходы: запретить и не пущать

Swift³⁸


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 сами³⁹ или подсмотрите в решебник⁴⁰.

³⁸ <https://github.com/apple/swift-evolution/blob/main/proposals/0282-atomics.md>

³⁹ <https://tonygoold.github.io/arcempire/>

⁴⁰ <https://github.com/apple/swift/blob/main/docs/proposals/Concurrency.rst> 

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification⁴¹: 14.5.4 Volatile fields (2)

ECMA-335, CLI⁴²: I.12.6 Memory model and optimizations (4)

⁴¹<https://www.ecma-international.org/publications-and-standards/standards/ecma-334>

⁴²<https://www.ecma-international.org/publications-and-standards/standards/ecma-335/>

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET⁴³.

⁴³<https://github.com/dotnet/runtime/blob/main/docs/design/specs/Memory-model.md>

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET⁴³. Официально несовместим с предыдущими документами.

⁴³ <https://github.com/dotnet/runtime/blob/main/docs/design/specs/Memory-model.md>

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET⁴³. Официально несовместим с предыдущими документами.

.NET runtime assumes that the side-effects of memory reads and writes include only observing and changing values at specified memory locations. This applies to all reads and writes - volatile or not. This is different from ECMA model.

⁴³ <https://github.com/dotnet/runtime/blob/main/docs/design/specs/Memory-model.md>

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

⁴⁴ <https://www.mono-project.com/docs/advanced/runtime/docs/atomics-memory-model/>

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

Модель памяти также описана для Mono⁴⁴, который старается сохранять конформность с .NET.

⁴⁴ <https://www.mono-project.com/docs/advanced/runtime/docs/atomics-memory-model/>

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

Модель памяти также описана для Mono⁴⁴, который старается сохранять конформность с .NET.

... here is a quirk in the .NET implementation where these methods actually use the MemoryBarrier method to insert a barrier. This is stronger than a simple acquire or release barrier. We do the same for compatibility.

⁴⁴ <https://www.mono-project.com/docs/advanced/runtime/docs/atomics-memory-model/>

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

Модель памяти также описана для Mono, который старается сохранять конформность с .NET.

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

Модель памяти также описана для Mono, который старается сохранять конформность с .NET.

Общий подход спецификации:

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ЕСМА-334, C# language specification: 14.5.4 Volatile fields (2)

ЕСМА-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

Модель памяти также описана для Mono, который старается сохранять конформность с .NET.

Общий подход спецификации:

- Сказать, что соответствующие операции имеют
 - release semantics
 - acquire semantics
 - full-fence semantics

что бы это ни значило :)

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

Модель памяти также описана для Mono, который старается сохранять конформность с .NET.

Общий подход спецификации:

- Использовать термины, не давая строгие определения.

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

Модель памяти также описана для Mono, который старается сохранять конформность с .NET.

Общий подход спецификации:

- Использовать термины, не давая строгие определения.
- Явно запретить некоторые перестановки операций с памятью.

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ЕСМА-334, C# language specification: 14.5.4 Volatile fields (2)

ЕСМА-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

Модель памяти также описана для Mono, который старается сохранять конформность с .NET.

Общий подход спецификации:

- Использовать термины, не давая строгие определения.
- Явно запретить некоторые перестановки операций с памятью.

Насколько полон список "запретных" оптимизаций, как будет система эволюционировать в будущем, будут ли еще отступления от стандарта не раскрыто.

Существующие подходы к описанию моделей памяти

Наивные подходы: reference implementation is a specification

ECMA-334, C# language specification: 14.5.4 Volatile fields (2)

ECMA-335, CLI: I.12.6 Memory model and optimizations (4)

Модель памяти также описана для .NET, официально несовместима с предыдущими документами.

Модель памяти также описана для Mono, который старается сохранять конформность с .NET.

Общий подход спецификации:

- Использовать термины, не давая строгие определения.
- Явно запретить некоторые перестановки операций с памятью.


Насколько полон список "запретных" оптимизаций, как будет система эволюционировать в будущем, будут ли еще отступления от стандарта не раскрыто.

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

Наиболее интуитивное определение для операций с памятью – они все происходят атомарно и их все можно расположить на единой шкале времени⁴⁵.

⁴⁵

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

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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

<code>void thread1() {</code>		<code>void thread2() {</code>
<code>foo()</code>		<code>baz()</code>
<code>bar()</code>		<code>foo()</code>
<code>}</code>		<code>}</code>

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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

<pre>void thread1() { lock() foo() unlock() lock() bar() unlock() }</pre>		<pre>void thread2() { lock() baz() unlock() lock() foo() unlock() }</pre>
---	--	---

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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()
}                          |      }
```

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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. Прямо как в Python!

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Прям как в Python!

Потоки в языке есть⁴⁶, просто их неэффективность является "особенностью" интерпретатора CPython.

⁴⁶ <https://docs.python.org/3/library/threading.html>

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Прям как в Python!

Потоки в языке есть⁴⁶, просто их неэффективность является "особенностью" интерпретатора CPython.

Но PyPy тоже не собирается отказываться от GIL⁴⁷.

⁴⁶ <https://docs.python.org/3/library/threading.html>

⁴⁷ <https://doc.pypy.org/en/latest/faq.html#does-pypy-have-a-gil-why>

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Прям как в Python!

Потоки в языке есть⁴⁶, просто их неэффективность является "особенностью" интерпретатора CPython.

Но PyPy тоже не собирается отказываться от GIL⁴⁷.

Попытка переделать модель языка пока не увенчалась успехом⁴⁸.

⁴⁶ <https://docs.python.org/3/library/threading.html>

⁴⁷ <https://doc.pypy.org/en/latest/faq.html#does-pypy-have-a-gil-why>

⁴⁸ <https://peps.python.org/pep-0583/>

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Прям как в Python!

Потоки в языке есть⁴⁶, просто их неэффективность является "особенностью" интерпретатора CPython.

Но PyPy тоже не собирается отказываться от GIL⁴⁷.

Попытка переделать модель языка пока не увенчалась успехом⁴⁸.

Просто так выбросить GIL мешает нежелание замедлять скриптовый язык еще на 10-20-30%, ломая интероп с нативными библиотеками⁴⁹.

⁴⁶ <https://docs.python.org/3/library/threading.html>

⁴⁷ <https://doc.pypy.org/en/latest/faq.html#does-pypy-have-a-gil-why>

⁴⁸ <https://peps.python.org/pep-0583/>

⁴⁹ <https://peps.python.org/pep-0703/>

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Один поток обрабатывает события, каждое из которых может породить другие, возможно отложенные, события.

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Один поток обрабатывает события, каждое из которых может породить другие, возможно отложенные, события. Event loop.

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Один поток обрабатывает события, каждое из которых может породить другие, возможно отложенные, события. Event loop.

Прям как в JavaScript!

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Один поток обрабатывает события, каждое из которых может породить другие, возможно отложенные, события. Event loop.

Прям как в JavaScript!

Оказалось, что пользователи любят использовать все ядра своих систем.

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Один поток обрабатывает события, каждое из которых может породить другие, возможно отложенные, события. Event loop.

Прям как в JavaScript!

Оказалось, что пользователи любят использовать все ядра своих систем. Можно запускать дополнительных независимых агентов (web workers) и общаться сообщениями⁵⁰.

⁵⁰ https://www.w3schools.com/html/html5_webworkers.asp

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Один поток обрабатывает события, каждое из которых может породить другие, возможно отложенные, события. Event loop.

Прям как в JavaScript!

Оказалось, что пользователи любят использовать все ядра своих систем. Можно запускать дополнительных независимых агентов (web workers) и общаться сообщениями⁵⁰. Можно разделять между агентами массивы байтов⁵¹.

⁵⁰ https://www.w3schools.com/html/html5_webworkers.asp

⁵¹ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer

Существующие подходы к описанию моделей памяти

Наивные подходы: strict consistency

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.

Один поток обрабатывает события, каждое из которых может породить другие, возможно отложенные, события. Event loop.

Прям как в JavaScript!

Оказалось, что пользователи любят использовать все ядра своих систем. Можно запускать дополнительных независимых агентов (web workers) и общаться сообщениями⁵⁰. Можно разделять между агентами массивы байтов⁵¹. Получить data race и думать, что это значит⁵².

⁵⁰ https://www.w3schools.com/html/html5_webworkers.asp

⁵¹ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer

⁵² "Repairing and Mechanising the JavaScript Relaxed Memory Model" <https://arxiv.org/abs/2005.10554>

Существующие подходы к описанию моделей памяти

Наивные подходы: метод страуса

C/C++ до стандартов C++11⁵³

⁵³

https://en.wikipedia.org/wiki/C%2B%2B11#Multithreading_memory_model

Существующие подходы к описанию моделей памяти

Наивные подходы: метод страуса

C/C++ до стандартов C++11⁵³

- С точки зрения стандарта, потоков не существовало. Были библиотеки их реализующие.
- Каждый проект изобретал свои костыли. Специфичные компилятору, процессору, среде запуска.
- Любой data race – undefined behaviour. Не подходит для безопасных языков.

⁵³

https://en.wikipedia.org/wiki/C%2B%2B11#Multithreading_memory_model

Существующие подходы к описанию моделей памяти

Наивные подходы: метод страуса

C/C++ до стандартов C++11⁵³

- С точки зрения стандарта, потоков не существовало. Были библиотеки их реализующие.
- Каждый проект изобретал свои костыли. Специфичные компилятору, процессору, среде запуска.
- Любой data race – undefined behaviour. Не подходит для безопасных языков.

Экономия времени дизайнеров языка ценой увеличенных издержек прикладных программистов.

⁵³

https://en.wikipedia.org/wiki/C%2B%2B11#Multithreading_memory_model

Существующие подходы к описанию моделей памяти

Прагматичные подходы: дай человеку удочку

С/C++ до соответствующих стандартов + POSIX threads⁵⁴

⁵⁴ <https://en.wikipedia.org/wiki/Pthreads>

Существующие подходы к описанию моделей памяти

Прагматичные подходы: дай человеку удочку

С/С++ до соответствующих стандартов + POSIX threads⁵⁴

- С точки зрения стандарта, потоков не существовало. Появилась универсальная библиотека их реализующая.
- Костыли собраны в одном месте, исходнике библиотеки.
- Любой data race – всё еще undefined behaviour. Use mutexes, Luke!

⁵⁴ <https://en.wikipedia.org/wiki/Pthreads>

Существующие подходы к описанию моделей памяти

Прагматичные подходы: дай человеку удочку

С/C++ до соответствующих стандартов + POSIX threads⁵⁴

- С точки зрения стандарта, потоков не существовало. Появилась универсальная библиотека их реализующая.
- Костыли собраны в одном месте, исходнике библиотеки.
- Любой data race – всё еще undefined behaviour. Use mutexes, Luke!

Во-первых, написание такой библиотеки представляет большой труд с кучей платформенно-специфичных трудностей.

⁵⁴ <https://en.wikipedia.org/wiki/Pthreads>

Существующие подходы к описанию моделей памяти

Прагматичные подходы: дай человеку удочку

С/C++ до соответствующих стандартов + POSIX threads⁵⁴

- С точки зрения стандарта, потоков не существовало. Появилась универсальная библиотека их реализующая.
- Костыли собраны в одном месте, исходнике библиотеки.
- Любой data race – всё еще undefined behaviour. Use mutexes, Luke!

Во-первых, написание такой библиотеки представляет большой труд с кучей платформенно-специфичных трудностей.

Во-вторых, "Threads Cannot Be Implemented As a Library"⁵⁵.

⁵⁴ <https://en.wikipedia.org/wiki/Pthreads>

⁵⁵ <https://www.hpl.hp.com/techreports/2004/HPL-2004-209.pdf>

Существующие подходы к описанию моделей памяти

Прагматичные подходы: дай человеку удочку

C/C++ до соответствующих стандартов + POSIX threads⁵⁴

- С точки зрения стандарта, потоков не существовало. Появилась универсальная библиотека их реализующая.
- Костыли собраны в одном месте, исходнике библиотеки.
- Любой data race – всё еще undefined behaviour. Use mutexes, Luke!

Во-первых, написание такой библиотеки представляет большой труд с кучей платформенно-специфичных трудностей.

Во-вторых, "Threads Cannot Be Implemented As a Library"⁵⁵.

The Pthreads specification prohibits races, i.e. accesses to a shared variable while another thread is modifying it. ... the problem here is that whether or not a race exists depends on the semantics of the programming language, which in turn requires that we have a properly defined memory model. Thus this definition is circular.

⁵⁴ <https://en.wikipedia.org/wiki/Pthreads>

⁵⁵ <https://www.hpl.hp.com/techreports/2004/HPL-2004-209.pdf>

Существующие подходы к описанию моделей памяти

Прагматичные подходы: дай человеку спиннинг

C/C++ до 11 версии + POSIX threads + санитайзеры

Существующие подходы к описанию моделей памяти

Прагматичные подходы: дай человеку спиннинг

С/C++ до 11 версии + POSIX threads + санитайзеры

Динамические проверки свойств программы:

- Поиск гонок
- Поиск неверной работы с памятью (use-after-free, leak, uninitialized memory access etc)
- Поиск undefined behaviour

Существующие подходы к описанию моделей памяти

Прагматичные подходы: дай человеку спиннинг

С/C++ до 11 версии + POSIX threads + санитайзеры

Динамические проверки свойств программы:

- Поиск гонок
- Поиск неверной работы с памятью (use-after-free, leak, uninitialized memory access etc)
- Поиск undefined behaviour

Valgrind⁵⁶

LLVM sanitizers⁵⁷

⁵⁶ <https://valgrind.org/>

⁵⁷ <https://github.com/google/sanitizers>

Существующие подходы к описанию моделей памяти

Прагматичные подходы: разрешенное подмножество операций

Специфицировать подмножество операций языка, предназначенных для многопоточных программ.

Существующие подходы к описанию моделей памяти

Прагматичные подходы: разрешенное подмножество операций

Специфицировать подмножество операций языка, предназначенных для многопоточных программ.

- C/C++ atomics

Существующие подходы к описанию моделей памяти

Прагматичные подходы: разрешенное подмножество операций

Специфицировать подмножество операций **языка**, предназначенных для многопоточных программ.

- C/C++ atomics
- Swift/ObjC NSLocking

Существующие подходы к описанию моделей памяти

Прагматичные подходы: разрешенное подмножество операций

Специфицировать подмножество операций языка, предназначенных для многопоточных программ.

- C/C++ atomics
- Swift/ObjC NSLocking
- Java-1996 volatile

Существующие подходы к описанию моделей памяти

Прагматичные подходы: разрешенное подмножество операций

Специфицировать подмножество операций **языка**, предназначенных для многопоточных программ.

- C/C++ atomics
- Swift/ObjC NSLocking
- Java-1996 volatile

Не путайте с реализацией на уровне библиотеки pthreads!

Существующие подходы к описанию моделей памяти

Прагматичные подходы: разрешенное подмножество операций

Специфицировать подмножество операций языка, предназначенных для многопоточных программ.

- C/C++ atomics
- Swift/ObjC NSLocking
- Java-1996 volatile

Не путайте с реализацией на уровне библиотеки pthreads!

Не так просто сделать: "The Java Memory Model is Fatally Flawed" , William Pugh, 2000⁵⁸

⁵⁸<http://www.cs.umd.edu/~pugh/java/broken.pdf>

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти

С использованием всего доступного арсенала:

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти

С использованием всего доступного арсенала:

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

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти

С использованием всего доступного арсенала:

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

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти

С использованием всего доступного арсенала:

- 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

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти

С использованием всего доступного арсенала:

- 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 models⁵⁹

⁵⁹"JSR-133 Cookbook for Compiler Writers" <https://gee.cs.oswego.edu/dl/jmm/cookbook.html>

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти

С использованием всего доступного арсенала:

- 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 models⁵⁹
- Каждый data race имеет разрешенные и запрещенные последствия

⁵⁹ "JSR-133 Cookbook for Compiler Writers" <https://gee.cs.oswego.edu/dl/jmm/cookbook.html>

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти

С использованием всего доступного арсенала:

- 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 models⁵⁹
- Каждый data race имеет разрешенные и запрещенные последствия

Подход обладает рядом недостатков:

- Очень сложно, долго и дорого.
 - Будут недочеты⁶⁰.
 - Мало кто в мире будет в состоянии полностью понять написанное.
- Еще меньше людей смогут применить на практике.

⁵⁹ "JSR-133 Cookbook for Compiler Writers" <https://gee.cs.oswego.edu/dl/jmm/cookbook.html>

⁶⁰ "Java Memory Model Examples: Good, Bad and Ugly" <https://groups.inf.ed.ac.uk/request/jmmexamples.pdf>

Java Memory Model

Java language specification: <https://docs.oracle.com/javase/specs/>

Java Memory Model

Java language specification: <https://docs.oracle.com/javase/specs/>
Глава 17 "Threads and Locks"

Java Memory Model

Java language specification: <https://docs.oracle.com/javase/specs/>

Глава 17 "Threads and Locks"

Раздел 17.4 Memory Model (15 страниц)

Java Memory Model

Java language specification: <https://docs.oracle.com/javase/specs/>
Глава 17 "Threads and Locks"

Раздел 17.4 Memory Model (15 страниц)

Основная идея – давайте попытаемся ввести частичный порядок среди различных событий: операции с полями, создание потоков, исполнение synchronized. Это поможет нам рассуждать о том, что было "раньше" либо "позже".

Java Memory Model

Java language specification: <https://docs.oracle.com/javase/specs/>
Глава 17 "Threads and Locks"

Раздел 17.4 Memory Model (15 страниц)

Основная идея – давайте попытаемся ввести частичный порядок среди различных событий: операции с полями, создание потоков, исполнение synchronized. Это поможет нам рассуждать о том, что было "раньше" либо "позже".

Попытки рассказать просто о сложном от Алексея Шипилева⁶¹:
<https://shipilev.net/talks/geecon-May2018-jmm.pdf>. Обратите внимание на слайд "Further Reading".

⁶¹ Для любителей выбирать между слайдами, конспектом и видео: <https://shipilev.net> 

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Doug Lea, private communication with Aleksey Shipilev, 2013⁶²

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.

⁶²Citation from <https://shipilev.net/blog/2014/jmm-pragmatics>, slide 109

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Нам понадобятся:

- Полная и исчерпывающая модель памяти языка
- Набор заведомо корректных и легко запоминаемых шаблонов многопоточного программирования

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Нам понадобятся:

- Полная и исчерпывающая модель памяти языка
- Набор заведомо корректных и легко запоминаемых шаблонов многопоточного программирования

В Java неувядающей классикой считается шаблон "Double checked locking"⁶³.

⁶³ "Double-Checked Locking is Broken"

<http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html>

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Нам понадобятся:

- Полная и исчерпывающая модель памяти языка
- Набор заведомо корректных и легко запоминаемых шаблонов многопоточного программирования

В Java неувядающей классикой считается шаблон "Double checked locking"⁶³. Который, если аккуратно его реализовать, в современной Java корректен. В некоторых командах вполне заслуженно считается антипаттерном/опасным кодом, т.к. существуют альтернативы⁶⁴.

⁶³ "Double-Checked Locking is Broken"

<http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html>

⁶⁴ https://en.wikipedia.org/wiki/Initialization-on-demand_holder_idiom

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Нам понадобятся:

- Полная и исчерпывающая модель памяти языка
- Набор заведомо корректных и легко запоминаемых шаблонов многопоточного программирования

В Java неувядающей классикой считается шаблон "Double checked locking"⁶³. Который, если аккуратно его реализовать, в современной Java корректен. В некоторых командах вполне заслуженно считается антипаттерном/опасным кодом, т.к. существуют альтернативы⁶⁴.

Рекомендую замечательные книги "Java Concurrency in Practice" , "Effective Java" и прекрасную документацию к пакету `java.util.concurrent`.

⁶³ "Double-Checked Locking is Broken"

<http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html>

⁶⁴ https://en.wikipedia.org/wiki/Initialization-on-demand_holder_idiom

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Недостатки подхода:

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Недостатки подхода:

- Недостатки? Какие недостатки?

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Недостатки подхода:

- Недостатки? Какие недостатки?
- Вы же на курсе по Java, а это самый лучший язык программирования :)

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Многопоточность сама по себе всё еще остается весьма сложной сущностью.

- 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

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Многопоточность сама по себе всё еще остается весьма сложной сущностью.

- 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

*... во многих мудрости много печали; и кто умножает познания,
умножает скорбь.*

Существующие подходы к описанию моделей памяти

Продвинутые подходы: исчерпывающая модель памяти + паттерны

Многопоточность сама по себе всё еще остается **интересной и непонятной штукой**.

- 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

Существующие подходы к описанию моделей памяти

А что же плюсы???

Существующие подходы к описанию моделей памяти

А что же плюсы???

Модель памяти для C++11 разрабатывалась с учетом опыта и ошибок Java Memory Model. Во многом – теми же людьми.

Существующие подходы к описанию моделей памяти

А что же плюсы???

Модель памяти для C++11 разрабатывалась с учетом опыта и ошибок Java Memory Model. Во многом – теми же людьми. Но у C++ есть свой "багаж" – undefined behaviour.

Существующие подходы к описанию моделей памяти

А что же плюсы???

Модель памяти для C++11 разрабатывалась с учетом опыта и ошибок Java Memory Model. Во многом – теми же людьми.

Но у C++ есть свой "багаж" – undefined behaviour.

С одной стороны, это развязывает руки писателям спецификации – не нужно описывать все-все краевые случаи (жизненно важно для управляемого безопасного языка).

Существующие подходы к описанию моделей памяти

А что же плюсы???

Модель памяти для C++11 разрабатывалась с учетом опыта и ошибок Java Memory Model. Во многом – теми же людьми.

Но у C++ есть свой "багаж" – undefined behaviour.

С одной стороны, это развязывает руки писателям спецификации – не нужно описывать все-все краевые случаи (жизненно важно для управляемого безопасного языка).

С другой стороны, инструментов для контроля всего и вся в языке гораздо больше (volatile, atomics, inline assembly, std::thread ...).

Существующие подходы к описанию моделей памяти

А что же плюсы???

Модель памяти для C++11 разрабатывалась с учетом опыта и ошибок Java Memory Model. Во многом – теми же людьми.

Но у C++ есть свой "багаж" – undefined behaviour.

С одной стороны, это развязывает руки писателям спецификации – не нужно описывать все-все краевые случаи (жизненно важно для управляемого безопасного языка).

С другой стороны, инструментов для контроля всего и вся в языке гораздо больше (volatile, atomics, inline assembly, std::thread ...).

Мне сложно кратко и по существу рассказать про такой монументальный артефакт человеческого труда как "C++11", поэтому просто посоветую почитать заметку от Russ Cox⁶⁵.

⁶⁵ <https://research.swtch.com/plmm>

Существующие подходы к описанию моделей памяти

А что же плюсы???

Модель памяти для C++11 разрабатывалась с учетом опыта и ошибок Java Memory Model. Во многом – теми же людьми.

Но у C++ есть свой "багаж" – undefined behaviour.

С одной стороны, это развязывает руки писателям спецификации – не нужно описывать все-все краевые случаи (жизненно важно для управляемого безопасного языка).

С другой стороны, инструментов для контроля всего и вся в языке гораздо больше (volatile, atomics, inline assembly, std::thread ...).

Мне сложно кратко и по существу рассказать про такой монументальный артефакт человеческого труда как "C++11", поэтому просто посоветую почитать заметку от Russ Cox⁶⁵.

Кратко: кое-какие дизайн решения некоторые уважаемые люди считают как минимум спорными.

⁶⁵ <https://research.swtch.com/plmm>

Существующие подходы к описанию моделей памяти

Продвинутые подходы: ahead-of-time checks

Существующие подходы к описанию моделей памяти

Продвинутые подходы: ahead-of-time checks

- Сделать **некоторые** некорректные многопоточные программы некомпilierуемыми (Rust)

Существующие подходы к описанию моделей памяти

Продвинутые подходы: ahead-of-time checks

- Сделать **некоторые** некорректные многопоточные программы некомпилируемыми (Rust)
- Выразить свойства программы в типах, вывести эти свойства из исходного текста (сепарационные логики, языки с зависимыми типами, инструменты дедуктивной верификации программ)

Существующие подходы к описанию моделей памяти

Продвинутые подходы: ahead-of-time checks

- Сделать **некоторые** некорректные многопоточные программы некомпilierуемыми (Rust)
- Выразить свойства программы в типах, вывести эти свойства из исходного текста (сепарационные логики, языки с зависимыми типами, инструменты дедуктивной верификации программ)
- Проверка моделей (model checking)

Существующие подходы к описанию моделей памяти

Продвинутые подходы: ahead-of-time checks

- Сделать **некоторые** некорректные многопоточные программы некомпilierуемыми (Rust)
- Выразить свойства программы в типах, вывести эти свойства из исходного текста (сепарационные логики, языки с зависимыми типами, инструменты дедуктивной верификации программ)
- Проверка моделей (model checking)

Особенности:

- Традиционно считается, что требуется бо́льшая квалификация.
- Иногда подход героически решает проблемы, которых и так нет в управляемых языках.
- Понимаете достоинства и недостатки таких инструментов – сумеете написать корректные программы почти на любом языке программирования.

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

- Игнорировать – некорректно

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

- Игнорировать – некорректно
- Запретить – undefined behaviour hell (C++)

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

- Игнорировать – некорректно
- Запретить – undefined behaviour hell (C++)
- Детектировать – проблема "spurious sanitizer fail"

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

- Игнорировать – некорректно
- Запретить – undefined behaviour hell (C++)
- Детектировать – проблема "spurious sanitizer fail"
- Объяснить – может быть весьма контринтуитивно (Java)

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

- Игнорировать – некорректно
- Запретить – undefined behaviour hell (C++)
- Детектировать – проблема "spurious sanitizer fail"
- Объяснить – может быть весьма контринтуитивно (Java)
- Не допускать – страдать от deadlock-ов и т.п.

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

- Игнорировать – некорректно
- Запретить – undefined behaviour hell (C++)
- Детектировать – проблема "spurious sanitizer fail"
- Объяснить – может быть весьма контринтуитивно (Java)
- ~~Не допускать – страдать от deadlock-ов и т.п.~~

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

- Игнорировать – некорректно
- Запретить – undefined behaviour hell (C++)
- Детектировать – проблема "spurious sanitizer fail"
- Объяснить – может быть весьма контринтуитивно (Java)
- Не допускать – страдать от deadlock-ов и т.п.
- Не допускать – "ляг, поспи, и всё пройдет"

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

- Игнорировать – некорректно
- Запретить – undefined behaviour hell (C++)
- Детектировать – проблема "spurious sanitizer fail"
- Объяснить – может быть весьма контринтуитивно (Java)
- Не допускать – страдать от deadlock-ов и т.п.
- Не допускать – "ляг, поспи, и всё пройдет"

Hardware and Software transactional memory⁶⁶

⁶⁶ https://en.wikipedia.org/wiki/Transactional_memory

Существующие подходы к описанию моделей памяти

Current research: hardware-level transactions

"Как перестать бояться data-race и начать жить?"

- Игнорировать – некорректно
- Запретить – undefined behaviour hell (C++)
- Детектировать – проблема "spurious sanitizer fail"
- Объяснить – может быть весьма контринтуитивно (Java)
- Не допускать – страдать от deadlock-ов и т.п.
- Не допускать – "ляг, поспи, и всё пройдет"

Hardware and Software transactional memory⁶⁶

Пока остается экспериментальной технологией.

⁶⁶ https://en.wikipedia.org/wiki/Transactional_memory

Существующие подходы к описанию моделей памяти

Current research: program synthesis

Генерировать программы по описанию алгоритма, корректные по построению.

Существующие подходы к описанию моделей памяти

Current research: program synthesis

Генерировать программы по описанию алгоритма, корректные по построению.

Нет, я не предлагаю использовать ChatGPT.

Существующие подходы к описанию моделей памяти

Current research: program synthesis

Генерировать программы по описанию алгоритма, корректные по построению.

Нет, я не предлагаю использовать ChatGPT.

Синтез программ – это интересно!

- Constraint solvers
- Fuzzers
- Evolutionary algorithms
- Conformance testing
- Program verification

Существующие подходы к описанию моделей памяти

Current research: program synthesis

Генерировать программы по описанию алгоритма, корректные по построению.

Нет, я не предлагаю использовать ChatGPT.

Синтез программ – это интересно!

- Constraint solvers
- Fuzzers
- Evolutionary algorithms
- Conformance testing
- Program verification

Пока – не очень доступно:

- Концептуальная сложность написания спецификаций
- Вычислительная сложность поиска подходящей программы
- Экономическая сложность внедрения и поддержки
- Фундаментальная сложность проверки свойств произвольной программы

Существующие подходы к описанию моделей памяти

Подходы следующего поколения

Существующие подходы к описанию моделей памяти

Подходы следующего поколения

Может быть именно вы станете их автором

Луч надежды

Вместо чтения фундаментальных трудов по многопоточному программированию, слабым моделям памяти и спецификациям процессоров, достаточно

Луч надежды

Вместо чтения фундаментальных трудов по многопоточному программированию, слабым моделям памяти и спецификациям процессоров, достаточно

- выбрать *правильный* язык программирования

Луч надежды

Вместо чтения фундаментальных трудов по многопоточному программированию, слабым моделям памяти и спецификациям процессоров, достаточно

- выбрать *правильный* язык программирования
- прочитать короткий сборник советов о том, как пользоваться `java.util.concurrent`

Луч надежды

Вместо чтения фундаментальных трудов по многопоточному программированию, слабым моделям памяти и спецификациям процессоров, достаточно

- выбрать *правильный* язык программирования
- прочитать короткий сборник советов о том, как пользоваться `java.util.concurrent`
- освоить пару распространенных Java-специфичных шаблонов написания потокобезопасного кода

Луч надежды

Вместо чтения фундаментальных трудов по многопоточному программированию, слабым моделям памяти и спецификациям процессоров, достаточно

- выбрать *правильный* язык программирования
- прочитать короткий сборник советов о том, как пользоваться `java.util.concurrent`
- освоить пару распространенных Java-специфичных шаблонов написания потокобезопасного кода

И можно приступать к написанию production кода.

Луч надежды

Вместо чтения фундаментальных трудов по многопоточному программированию, слабым моделям памяти и спецификациям процессоров, достаточно

- выбрать *правильный* язык программирования
- прочитать короткий сборник советов о том, как пользоваться `java.util.concurrent`
- освоить пару распространенных Java-специфичных шаблонов написания потокобезопасного кода

И можно приступать к написанию production кода.

Экономия времени прикладных программистов ценой увеличенных издержек дизайнеров языка.

Предостережение

Помните!

Предостережение

Помните!

Все алгоритмические проблемы многопоточности (deadlock, livelock, starvation, lock convoy, priority inversion и т.п.) всё еще представляют угрозу.

Предостережение

Помните!

Все алгоритмические проблемы многопоточности (deadlock, livelock, starvation, lock convoy, priority inversion и т.п.) всё еще представляют угрозу.

Берегитесь!

Предостережение

Помните!

Все алгоритмические проблемы многопоточности (deadlock, livelock, starvation, lock convoy, priority inversion и т.п.) всё еще представляют угрозу.

Берегитесь!

Очень мало людей **действительно** хорошо понимают современные модели памяти. Если вам приходится часто рассуждать про свой код в стиле "тут случается happens-before между двумя доступами к памяти, а потом через intra-thread order мы протягиваем зависимость до той volatile операции, чтобы в итоге ..." – это **ОЧЕНЬ** тревожный звоночек.

Предостережение

Помните!

Все алгоритмические проблемы многопоточности (deadlock, livelock, starvation, lock convoy, priority inversion и т.п.) всё еще представляют угрозу.

Берегитесь!

Очень мало людей **действительно** хорошо понимают современные модели памяти. Если вам приходится часто рассуждать про свой код в стиле "тут случается happens-before между двумя доступами к памяти, а потом через intra-thread order мы протягиваем зависимость до той volatile операции, чтобы в итоге ..." – это **ОЧЕНЬ** тревожный звоночек. С вероятностью 99% вы себя вводите в заблуждение.

Предостережение

Помните!

Все алгоритмические проблемы многопоточности (deadlock, livelock, starvation, lock convoy, priority inversion и т.п.) всё еще представляют угрозу.

Берегитесь!

Очень мало людей **действительно** хорошо понимают современные модели памяти. Если вам приходится часто рассуждать про свой код в стиле "тут случается happens-before между двумя доступами к памяти, а потом через intra-thread order мы протягиваем зависимость до той volatile операции, чтобы в итоге ..." – это **ОЧЕНЬ** тревожный звоночек. С вероятностью 99% вы себя вводите в заблуждение.

Уверены в корректности? Задумайтесь о поддерживаемости кода. Гораздо лучше, чтобы программа была собрана из понятных, широко известных и проверенных шаблонов-кирпичиков

Заключение

- Современные компиляторы – это сложно
- Современные процессоры – это сложно

Заключение

- Современные компиляторы – это сложно
- Современные процессоры – это сложно

Следствие – спецификация современного многопоточного высокопроизводительного надежного языка довольно сложна.

Заключение

- Современные компиляторы – это сложно
- Современные процессоры – это сложно

Следствие – спецификация современного многопоточного высокопроизводительного надежного языка довольно сложна.

К счастью, обычно авторы желают облегчить жизнь разработчикам и предоставляют рецепты/шаблоны/паттерны, которые надежно работают. Вы можете их заучить и применять.

Заключение

- Современные компиляторы – это сложно
- Современные процессоры – это сложно

Следствие – спецификация современного многопоточного высокопроизводительного надежного языка довольно сложна.

К счастью, обычно авторы желают облегчить жизнь разработчикам и предоставляют рецепты/шаблоны/паттерны, которые надежно работают. Вы можете их заучить и применять.

Иногда этого недостаточно, например, если вы

Заключение

- Современные компиляторы – это сложно
- Современные процессоры – это сложно

Следствие – спецификация современного многопоточного высокопроизводительного надежного языка довольно сложна.

К счастью, обычно авторы желают облегчить жизнь разработчикам и предоставляют рецепты/шаблоны/паттерны, которые надежно работают. Вы можете их заучить и применять.

Иногда этого недостаточно, например, если вы

- разработчик продвинутых многопоточных алгоритмов

Заключение

- Современные компиляторы – это сложно
- Современные процессоры – это сложно

Следствие – спецификация современного многопоточного высокопроизводительного надежного языка довольно сложна.

К счастью, обычно авторы желают облегчить жизнь разработчикам и предоставляют рецепты/шаблоны/паттерны, которые надежно работают. Вы можете их заучить и применять.

Иногда этого недостаточно, например, если вы

- разработчик продвинутых многопоточных алгоритмов
- автор оптимизирующего компилятора или рантайма

Заключение

- Современные компиляторы – это сложно
- Современные процессоры – это сложно

Следствие – спецификация современного многопоточного высокопроизводительного надежного языка довольно сложна.

К счастью, обычно авторы желают облегчить жизнь разработчикам и предоставляют рецепты/шаблоны/паттерны, которые надежно работают. Вы можете их заучить и применять.

Иногда этого недостаточно, например, если вы

- разработчик продвинутых многопоточных алгоритмов
- автор оптимизирующего компилятора или рантайма
- инженер по производительности, выжимающий максимальную скорость ценой хрупкого кода⁶⁷

⁶⁷ Красная зона в терминах <https://youtu.be/p2b4JHESE0c>

Заключение

- Современные компиляторы – это сложно
- Современные процессоры – это сложно

Следствие – спецификация современного многопоточного высокопроизводительного надежного языка довольно сложна.

К счастью, обычно авторы желают облегчить жизнь разработчикам и предоставляют рецепты/шаблоны/паттерны, которые надежно работают. Вы можете их заучить и применять.

Иногда этого недостаточно, например, если вы

- разработчик продвинутых многопоточных алгоритмов
- автор оптимизирующего компилятора или рантайма
- инженер по производительности, выжимающий максимальную скорость ценой хрупкого кода⁶⁷
- исследователь корректности и надежности concurrent software

⁶⁷ Красная зона в терминах <https://youtu.be/p2b4JHESE0c>

Заключение

- Современные компиляторы – это сложно
- Современные процессоры – это сложно

Следствие – спецификация современного многопоточного высокопроизводительного надежного языка довольно сложна.

К счастью, обычно авторы желают облегчить жизнь разработчикам и предоставляют рецепты/шаблоны/паттерны, которые надежно работают. Вы можете их заучить и применять.

Иногда этого недостаточно, например, если вы

- разработчик продвинутых многопоточных алгоритмов
- автор оптимизирующего компилятора или рантайма
- инженер по производительности, выжимающий максимальную скорость ценой хрупкого кода⁶⁷
- исследователь корректности и надежности concurrent software
- любите понимать происходящее на глубоком уровне

⁶⁷ Красная зона в терминах <https://youtu.be/p2b4JHESE0c>

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 Cox⁶⁸
- "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

⁶⁸<https://research.swtch.com/mm>

Посмотреть

- Роман Елизаров, "Многопоточное программирование — теория и практика" <https://youtu.be/mf4lC6Tpc1M>
- Алексей Шипилев, JMM series <https://shipilev.net>
- Herb Sutter, C++ and Beyond 2012, "Atomic Weapons" series <https://youtu.be/A8eCG0qgvH4>

Q & A