XII международная конференция
CEE-SECR / РАЗРАБОТКА ПО

28 - 29 октября, Москва

Помоги ближнему, или Как потоки помогают друг другу

Алексей Федоров

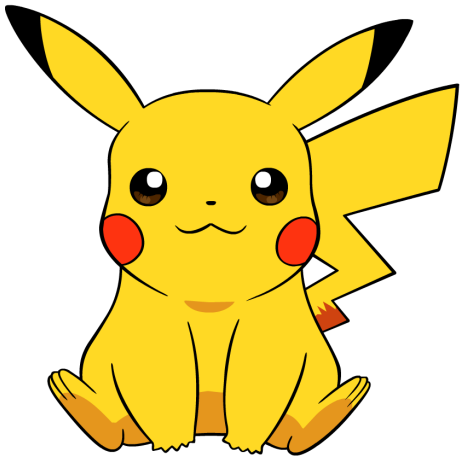
Одноклассники

Много интересных докладов в других залах

12:35 30 мин.	12:35 30 мин.	12:35 30 мин.	Технологии и средства программирования
Помоги ближнему, или Как потоки помогают друг другу Алексей Федоров, Одноклассники	Почему Agile больше не работает Борис Вольфсон, HeadHunter	Проектирование с учетом пользовательских требований: как от продукта для 20 человек мы пришли к массовому продукту Маргарита Титова, Onlinetours.ru	12:35 15 мин. О создании компилятора с высокоуровневого языка на компьютер с программируемой архитектурой Юрий Михайлуц, Южный федеральный университет

Pokemon или Big Data?

https://pixelastic.github.io/pokemonorbigdata/



VS.

- [ ] Big data, Smart data

Horsea

Big Data

Pokemon

Horsea is Pokemon!

Horsea is a small, blue, seahorse-like Pokémon with a single dorsal fin and a tightly curled tai

Impala

Big Data

Pokemon

Impala is Big Data!

Querying big data is too slow? Impala has a solution for it. But reconsider taming that one, it is built on C++.

Vulpix

Big Data

Pokemon

Vulpix is Pokemon!

Munchlax is a great counter to Vulpix, being able to take four Fire Blasts from the Life Orb set and hit back hard with Earthquake or STAB Return.

Это доклад не про Big Data!

Про что этот доклад

- Программирование
- Алгоритмы
- Многопоточность

Много интересных докладов в других залах

12:35 30 мин.	12:35 30 мин.	12:35 30 мин.	Технологии и средства программирования
Помоги ближнему, или Как потоки помогают друг другу Алексей Федоров, Одноклассники	Почему Agile больше не работает Борис Вольфсон, HeadHunter	Проектирование с учетом пользовательских требований: как от продукта для 20 человек мы пришли к массовому продукту Маргарита Титова, Onlinetours.ru	12:35 15 мин. О создании компилятора с высокоуровневого языка на компьютер с программируемой архитектурой Юрий Михайлуц, Южный федеральный университет

Про что этот доклад

Новичкам в области многопоточности

Sorry

Перейдите в другой зал

Про что этот доклад

Новичкам в области многопоточности

Sorry

Перейдите в другой зал

Новичкам в неблокирующей синхронизации

Короткое введение

Про что этот доклад

Новичкам в области многопоточности

Sorry

Перейдите в другой зал

Новичкам в неблокирующей синхронизации

Короткое введение

Продвинутым многопоточным программистам

Поговорим о **фишечках** неблокирующих алгоритмов

Модели

Модели

Модель **с разделяемой памятью**

- Операции на **атомарных регистрах**: **read**, **write**
- Удобно программировать, все привыкли

Модель **с передачей сообщений**

- Операции: **send**, **onReceive**
- Похожа на то, как реально работает железо

Преимущества параллелизма

- Использование **нескольких** ядер/процессоров
  - Да и на 1 ядре тоже! (async I/O)
- **Простота** моделирования: фреймворк забирает сложность
- Упрощенная обработка **асинхронных** событий
- Более **отзывчивые** интерфейсы пользователя
  - Event Dispatch Thread (EDT), async calls

Проблемы блокировок

- **Взаимоблокировки (Deadlocks)**

Проблемы блокировок

- **Взаимоблокировки (Deadlocks)**
- **Инверсия приоритетов**

Проблемы блокировок

- **Взаимоблокировки (Deadlocks)**
- **Инверсия приоритетов**
- **Надежность** — вдруг владелец блокировки помрет?

Проблемы блокировок

- **Взаимоблокировки (Deadlocks)**
- **Инверсия приоритетов**
- **Надежность** — вдруг владелец блокировки помрет?
- **Performance**
  - Параллелизма в критической секции нет!
  - Владелец блокировки может быть вытеснен планировщиком

Закон Амдала

α	часть общего объема вычислений, которую нельзя распараллелить
$1-\alpha$	часть, которую можно распараллелить
p	количество потоков

Закон Амдала

α — часть общего объема вычислений, которую нельзя распараллелить

$1-\alpha$ — часть, которую можно распараллелить

p — количество потоков

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

Фундаментальные законы природы

- 19 век — законы Ньютона

Фундаментальные законы природы

- 19 век — законы Ньютона
- 20 век — закон Мура
  - тактовые частоты
  - число транзисторов

Фундаментальные законы природы

- 19 век — законы Ньютона
- 20 век — закон Мура
  - тактовые частоты
  - число транзисторов
- 21 век — закон Амдала

Фундаментальные законы природы

- 19 век — законы Ньютона
- 20 век — закон Мура
  - тактовые частоты
  - число транзисторов
- 21 век — закон Амдала

Нарушение закона влечет за собой дисциплинарную, гражданско-правовую, административную или уголовную ответственность

If-Modify-Write

```
volatile int value = 0;
```

```
if (value == 0) {
    value = 42;
}
```

Нет атомарности

Compare and Swap

```
int value = 0;


LOCK
    if (value == 0) {
        value = 42;
    }
UNLOCK
```

Введем волшебную операцию

```
int value = 0;


i.compareAndSet(0, 42);
```

Compare and Swap — Hardware Support

compare-and-swap

CAS

x86

x64

cmpxchg

load-link / store-conditional

LL/SC

ARM

ldrex/strex

PowerPC

lwarx/stwcx

CAS Semantics

```
public class PseudoCAS {

    private long value;

    public synchronized long get() { return value; }

    public synchronized long compareAndSwap(long expected, long newV) {
        long oldValue = value;
        if (oldValue == expected) {
            value = newV;
        }
        return oldValue;
    }

    public synchronized boolean compareAndSet(long expected, long newV){
        return expected == compareAndSwap(expected, newV);
    }
}
```

Пример 1. Многопоточный счетчик

```
public class CasLoopCounter implements Counter {

    private AtomicLong value = new AtomicLong();

    public long get() {
        return value.get();
    }

    public void increment() {
        long v;
        do {
            v = value.get();
        } while (value.compareAndSet(v, v + 1));
    }

}
```

Недостатки CAS

Contended CAS —> tons of **useless** CPU cycles

```
do {
     v = value.get();
} while (value.compareAndSet(v, v + 1));
```

Написание быстрых и корректных алгоритмов на CAS
требует экспертизы

```
public void push(E item) {
    Node<E> newHead = new Node<>(item);
    Node<E> oldHead;
    do {
        oldHead = top.get();
        newHead.next = oldHead;
    } while (!top.compareAndSet(oldHead, newHead));
}
```

```
public E pop() {
    Node<E> newHead;
    Node<E> oldHead;
    do {
        oldHead = top.get();
        if (oldHead == null) {
            return null;
        }
        newHead = oldHead.next;
    } while (!top.compareAndSet(oldHead, newHead));

    return oldHead.item;
}
```

Неблокирующая очередь

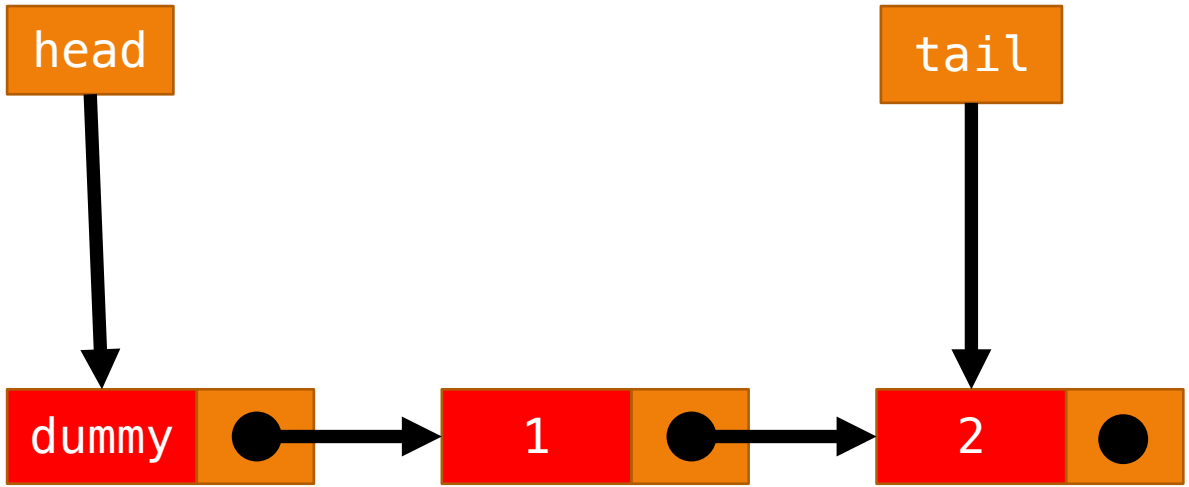
Michael and Scott, 1996

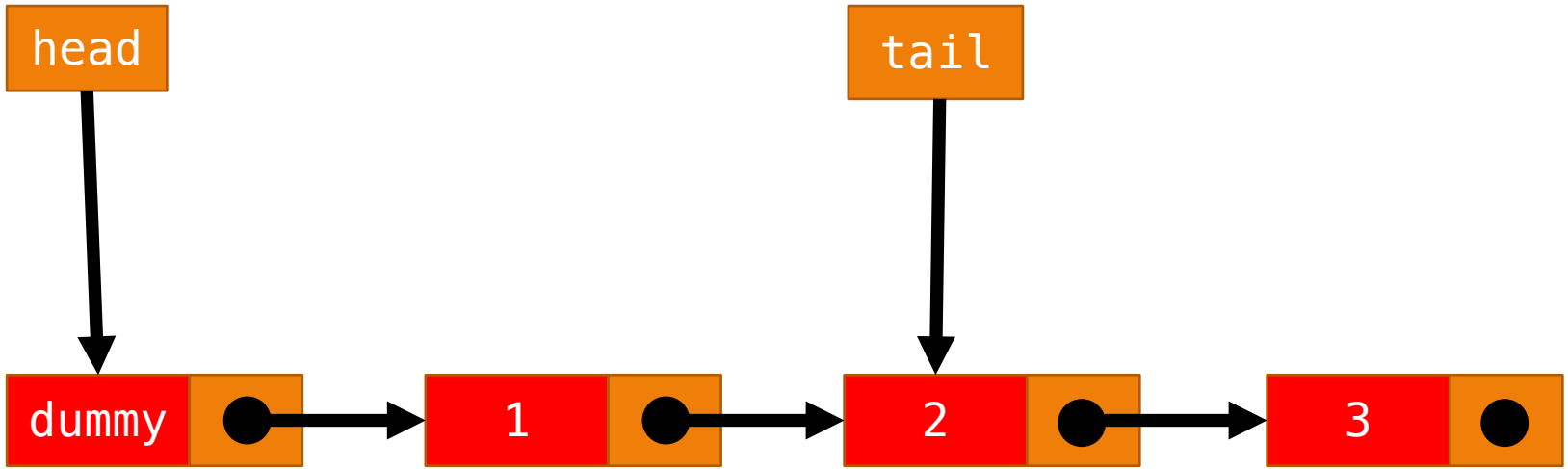
https://www.research.ibm.com/people/m/michael/podc-1996.pdf

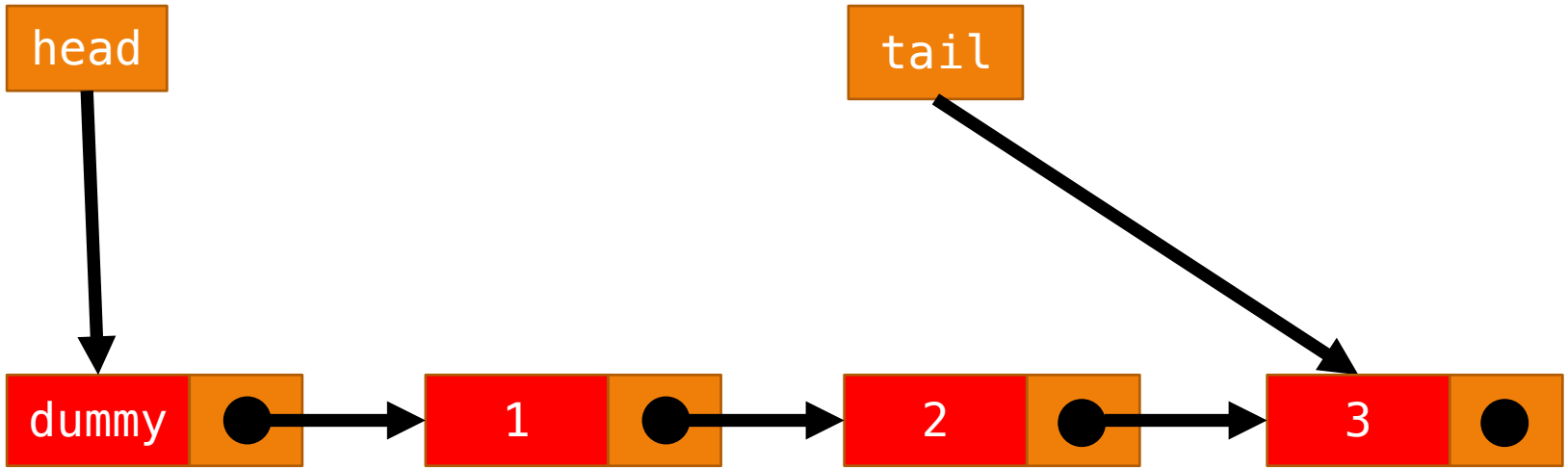
Потоки **помогают** друг другу

```java
public class LinkedQueue<E> {

    private static class Node<E> {
        final E item;
        final AtomicReference<Node<E>> next;

        public Node(E item, AtomicReference<Node<E>> next) {
            this.item = item;
            this.next = next;
        }
    }

    private final Node<E> dummy = new Node<>(null, null);
    private final AtomicReference<Node<E>> head = new AtomicReference<>(dummy);
    private final AtomicReference<Node<E>> tail = new AtomicReference<>(dummy);

}
```

```java
public class LinkedQueue<E> {

    private static class Node<E> {
        final E item;
        final AtomicReference<Node<E>> next;

        public Node(E item, AtomicReference<Node<E>> next) {
            this.item = item;
            this.next = next;
        }
    }

    private final Node<E> dummy = new Node<>(null, null);
    private final AtomicReference<Node<E>> head = new AtomicReference<>(dummy);
    private final AtomicReference<Node<E>> tail = new AtomicReference<>(dummy);

}
```


head
tail
dummy
1
2


head
tail
dummy
1
2
3


head
tail
dummy
1
2
3

```
public boolean put(E item) {
    Node<E> newNode = new Node<>(item, null);
    while (true) {
        Node<E> currentTail = tail.get();
        Node<E> tailNext = currentTail.next.get();
        if (currentTail == tail.get()) {
            if (tailNext != null) { // промежуточное состояние
               обновляем tail
            } else {// Консистентное состояние!
               Пытаемся добавить новую ноду
                 if (вставили_успешно) {
                    пытаемся обновить tail
                    return true;
                 }
            }
        }
    }
}
```

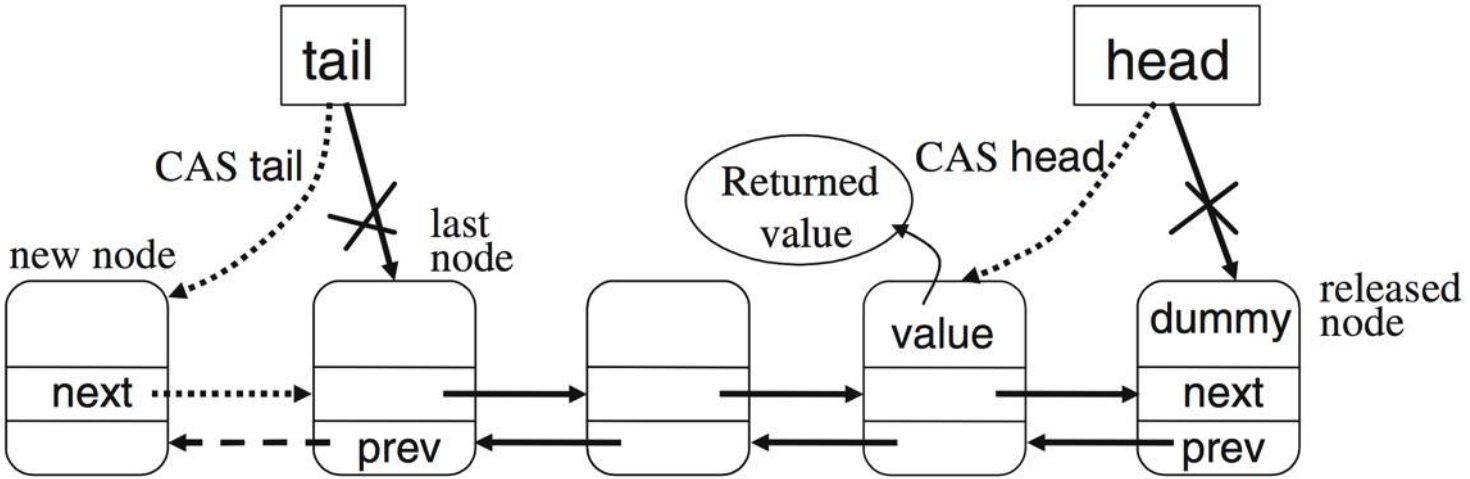
```
public boolean put(E item) {
    Node<E> newNode = new Node<>(item, null);
    while (true) {
        Node<E> currentTail = tail.get();
        Node<E> tailNext = currentTail.next.get();
        if (currentTail == tail.get()) {
            if (tailNext != null) {
                tail.compareAndSet(currentTail, tailNext);
            } else {
                if (currentTail.next.compareAndSet(null, newNode)) {
                    tail.compareAndSet(currentTail, newNode);
                    return true;
                }
            }
        }
    }
}
```

Модификации

Optimistic Approach

Ladan-Mozes, Shavit, 2004, 2008

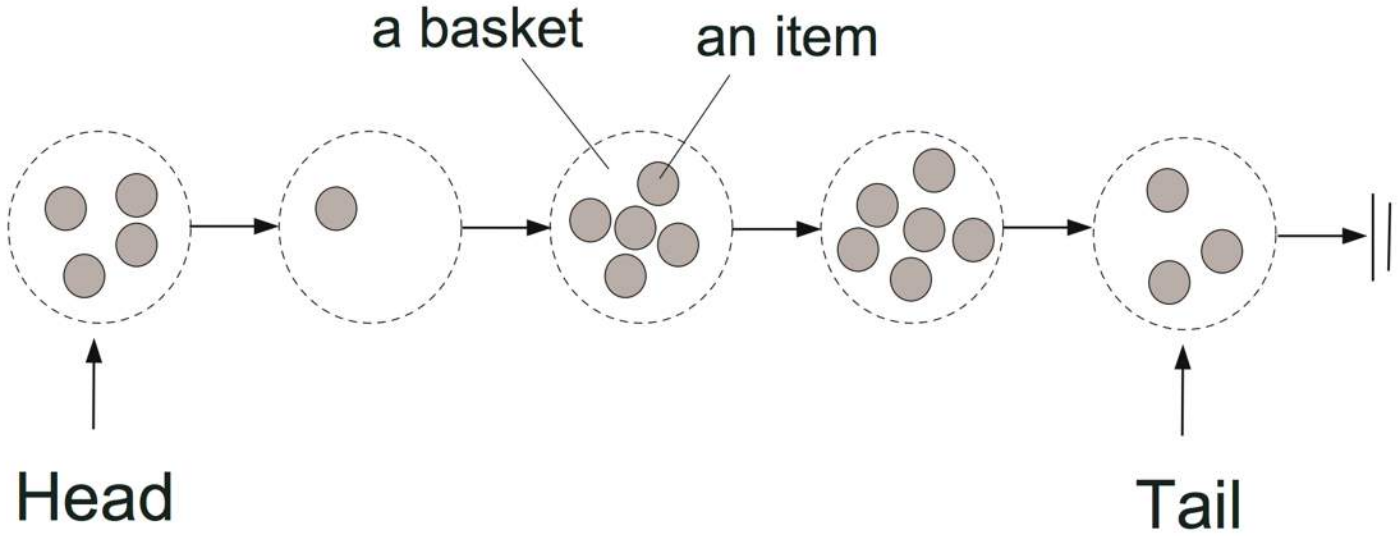
Идея: избавиться от второго CAS



http://people.csail.mit.edu/edya/publications/OptimisticFIFOQueue-journal.pdf

Baskets Queue

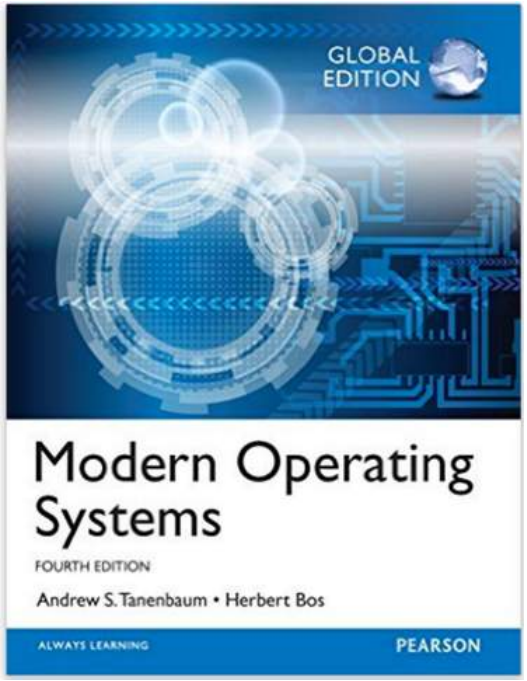
Hoffman, Shalev, Shavit, 2007

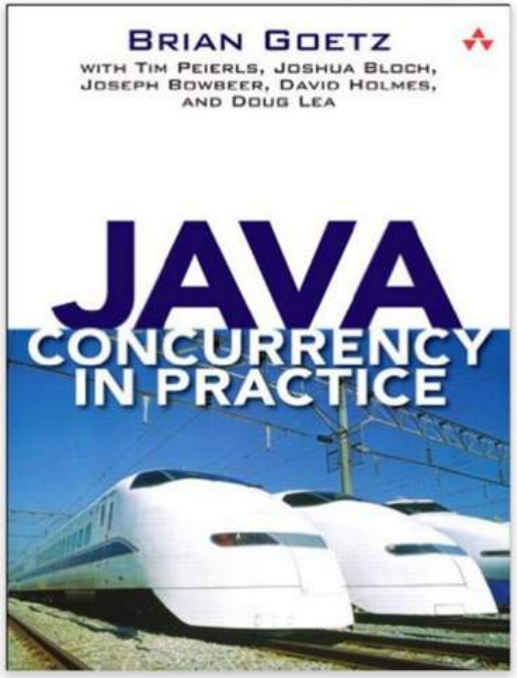


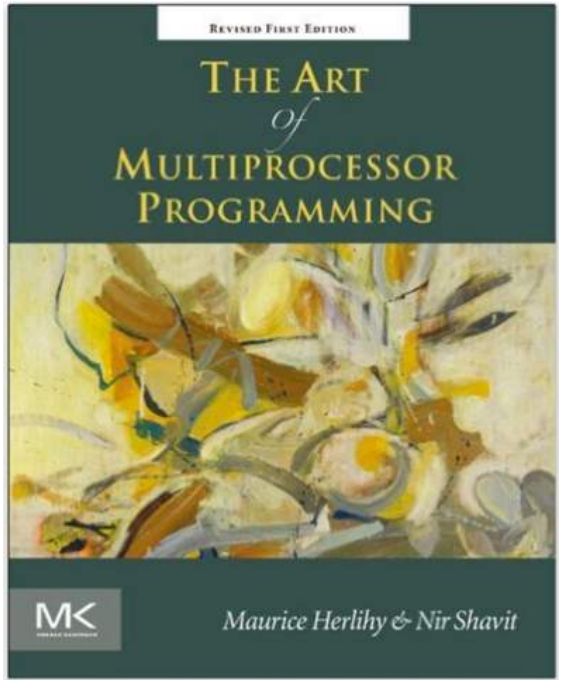
http://people.csail.mit.edu/shanir/publications/Baskets%20Queue.pdf

Материалы

Литература


GLOBAL EDITION
Modern Operating Systems
FOURTH EDITION
Andrew S. Tanenbaum • Herbert Bos
ALWAYS LEARNING
PEARSON


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA CONCURRENCY IN PRACTICE


REVISED FIRST EDITION
THE ART of MULTIPROCESSOR PROGRAMMING
MK
Maurice Herlihy & Nir Shavit

Материалы

- Nitsan Wakart — http://psy-lob-saw.blogspot.com/
- Алексей Шипилёв — https://shipilev.net/
- Максим Хижинский — https://habrahabr.ru/post/219201/

Вопросы и ответы