

программирован ие

ПРОБЛЕМЫ МНОГОПОТОЧНЫХ
ПРИЛОЖЕНИИ.

ПРИМИТИВЫ СИНХРОНИЗАЦИИ
ПОТОКОВ.



План лекции

Проблемы многопоточных приложений

- Введение
 - Задачи читателей и писателей
 - Задачи спящего парикмахера
- Не DeadLock
 - Доступ к разделяемым данным
 - Atomicity-Violation Bugs
 - Order-Violation Bugs
- DeadLock
 - Примеры MSSQL
- Примитивы синхронизации
 - Критические секции
 - Мьютексы
 - Семафоры
 - События
 - Обработка таймаутов

Простое различие

Последовательная программа - программа, которая выполняет логическую операцию и когда она заканчивается, выполняет следующую логическую операцию.

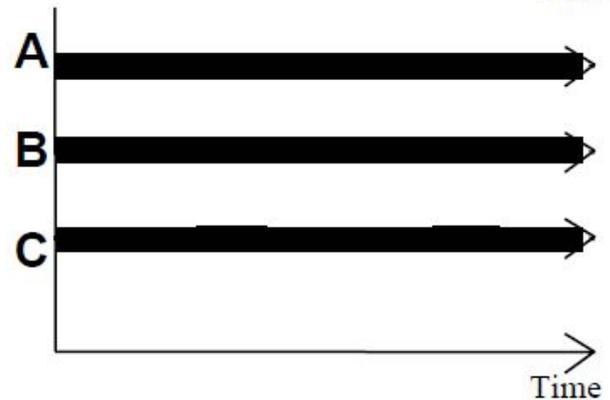
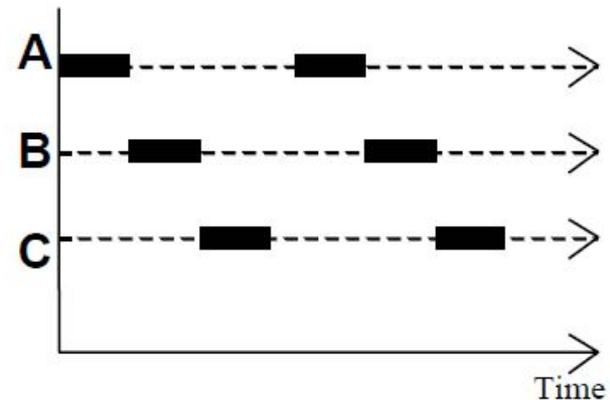
Параллельная программа - это **набор** независимых последовательных операций, **выполняющихся одновременно**.

Реализация параллельных вычислений

- Многопоточная
- Многопроцессная
- Распределенная

Parallelism vs Concurrency

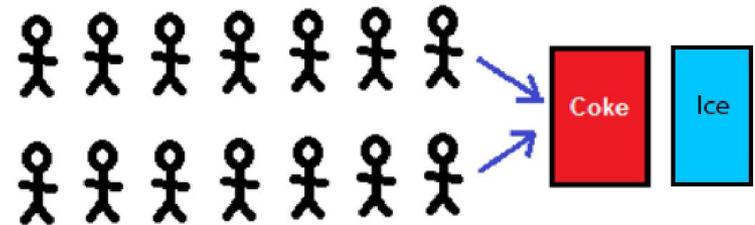
- Concurrency is not (only) parallelism
- Interleaved Concurrency
 - Logically simultaneous processing
 - Interleaved execution on a single processor
- Parallelism
 - Physically simultaneous processing
 - Requires a multiprocessors or a multicore system



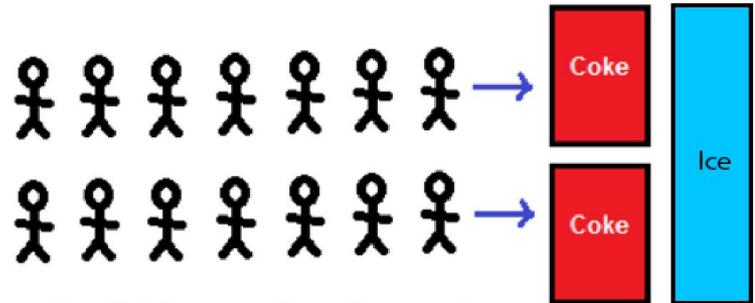
Parallelism vs Concurrency

Сходства

- Более быстрое выполнение по сравнению с одной очередью
- Последовательное выполнение в рамках одной очереди
- Борьба за ресурсы (теряется скорость параллелизма)



Concurrent: 2 queues, 1 vending machine

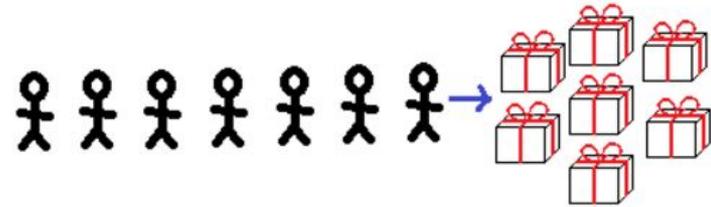


Parallel: 2 queues, 2 vending machines

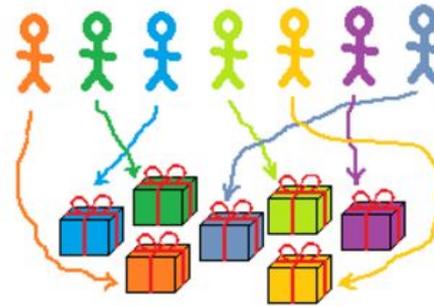
Parallelism vs Concurrency

Отличия

- В случае Concurrency
 - необходима очередь
 - первый получает любой подарок
- В случае Parallelism
 - очередь не нужна (выигрыш по скорости)
 - заранее известно кто какой подарок возьмет



Concurrency: 7 kids queueing for presents from 1 heap



Parallelism: 7 kids getting 7 labeled presents, no queue

Parallelism vs Concurrency

Лабораторная работа №1

- Concurrency ?
- Parallelism?

Проблемы

- Shared resources
- DeadLocks
- ABA problem

Проблемы и примеры

- **SharedResources**

- Банковский счет (*неверная запись переменной*)
- Слишком много молока (*одновременное выполнение действия*)
- Проблема спящего парикмахера (*неверное состояние системы*)

- **DeadLocks**

- Проблема обедающих философов (*взаимная блокировка, бездействие, простаивание*)

- **ABA problem**

Проблемы и примеры

SharedResources

ГОНКИ ПОТОКОВ. БАНКОВСКИЙ СЧЕТ.

```
import java.util.*;

public class Account {
    String id;
    String password;
    int balance;

    Account(String id, String password, String balance) {
        this.id = id;
        this.password = password;
        this.balance = balance;
    }

    boolean is_password(String password) {
        return password == this.password;
    }

    int getbal() {
        return balance;
    }

    void post(int v) {
        balance = balance + v;
    }
}
```

```
import java.util.*;

public class Bank {
    HashMap<String, Account> accounts;
    static Bank theBank = null;

    private Bank() {
        accounts = new HashMap<String, Account>();
    }

    public static Bank getbank() {
        if (theBank == null)
            theBank = new Bank();
        return theBank;
    }

    public Account get(String ID) {
        return accounts.get(ID);
    }
    ...
}
```

Гонки потоков. Терминал оператора.

```
public void run() {
    while(true) {
        try {
            out.print("Account ID > ");
            String id = in.readLine();
            String acc = bnk.get(id);
            if (acc == null) throw new Exception();
            out.print("Password > ");
            String pass = in.readLine();
            if (!acc.is_password(pass))
                throw new Exception();
            out.print("your balance is " + acc.getbal());
            out.print("Deposit or withdraw amount > ");
            int val = in.read();
            if (acc.getbal() + val > 0)
                acc.post(val);
            else
                throw new Exception();
            out.print("your balance is " + acc.getbal());
        } catch(Exception e) {
            out.println("Invalid input, restart");
        }
    }
}
```

Гонки потоков. Одновременное снятие.

balance

100

ATM 1

```
out.print("your balance is " + acc.getbal());  
Your account balance is 100
```

```
out.print("Deposit or withdraw amount > ");  
Deposit or Withdraw amount >
```

-90

```
int val = in.read();
```

100

```
if (acc.getbal() + val > 0)
```

100

```
acc.post(val);
```

10

10

```
out.print("your balance is " + acc.getbal());  
Your account balance is 10
```

ATM 2

```
out.print("your balance is " + acc.getbal());  
Your account balance is 100
```

```
out.print("Deposit or withdraw amount > ");  
Deposit or Withdraw amount >
```

-90

```
int val = in.read();
```

```
if (acc.getbal() + val > 0)
```

```
acc.post(val);
```

```
out.print("your balance is " + acc.getbal());  
Your account balance is 10
```

Гонки потоков. Длительные операции.

<i>time</i>	You	Your Roommate
3:00	Arrive home	
3:05	Look in fridge, no milk	
3:10	Leave for grocery	
3:15		Arrive home
3:20	Arrive at grocery	Look in fridge, no milk
3:25	Buy milk	Leave for grocery
3:35	Arrive home, put milk in fridge	
3:45		Buy Milk
3:50		Arrive home, put up milk
3:50		Oh no!



Гонки потоков. Длительные операции.

Нужна блокировка двойного выполнения.

Варианты?

Гонки потоков. Длительные операции.

thread A

```
if (no milk && no note)
```

```
  leave note
```

```
  buy milk
```

```
  remove note
```

thread B

```
if (no milk && no note)
```

```
  leave note
```

```
  buy milk
```

```
  remove note
```

Критическая секция

Участок исполняемого кода программы, в котором производится доступ к общему ресурсу (данным или устройству), который не должен быть одновременно использован более чем одним потоком исполнения.

При нахождении в критической секции двух (или более) процессов возникает состояние «гонки» («состязания»).

Гонки потоков. Одновременное снятие.

```
synchronized int getbal() {  
    return balance;  
}  
  
synchronized void post(int v) {  
    balance = balance + v;  
}
```

Эксклюзивный
доступ

```
if (acc.getbal() + val > 0)  
    acc.post(val);  
else  
    throw new Exception();
```

Код снятия со
счета

Решена ли проблема?

ГОНКИ ПОТОКОВ. ДВОЙНОЕ СПИСАНИЕ.

balance	ATM 1	ATM 2
100		<code>int val = in.read();</code>
	<code>int val = in.read();</code>	
100		<code>if (acc.getbal() + val > 0)</code>
100	<code>if (acc.getbal() + val > 0)</code>	
100		<code>acc.post(val);</code>
10	<code>acc.post(val);</code>	
-80		<code>out.print("your balance is " + acc.getbal());</code> <code>Your account balance is -80</code>
	<code>out.print("your balance is " + acc.getbal());</code> <code>Your account balance is -80</code>	

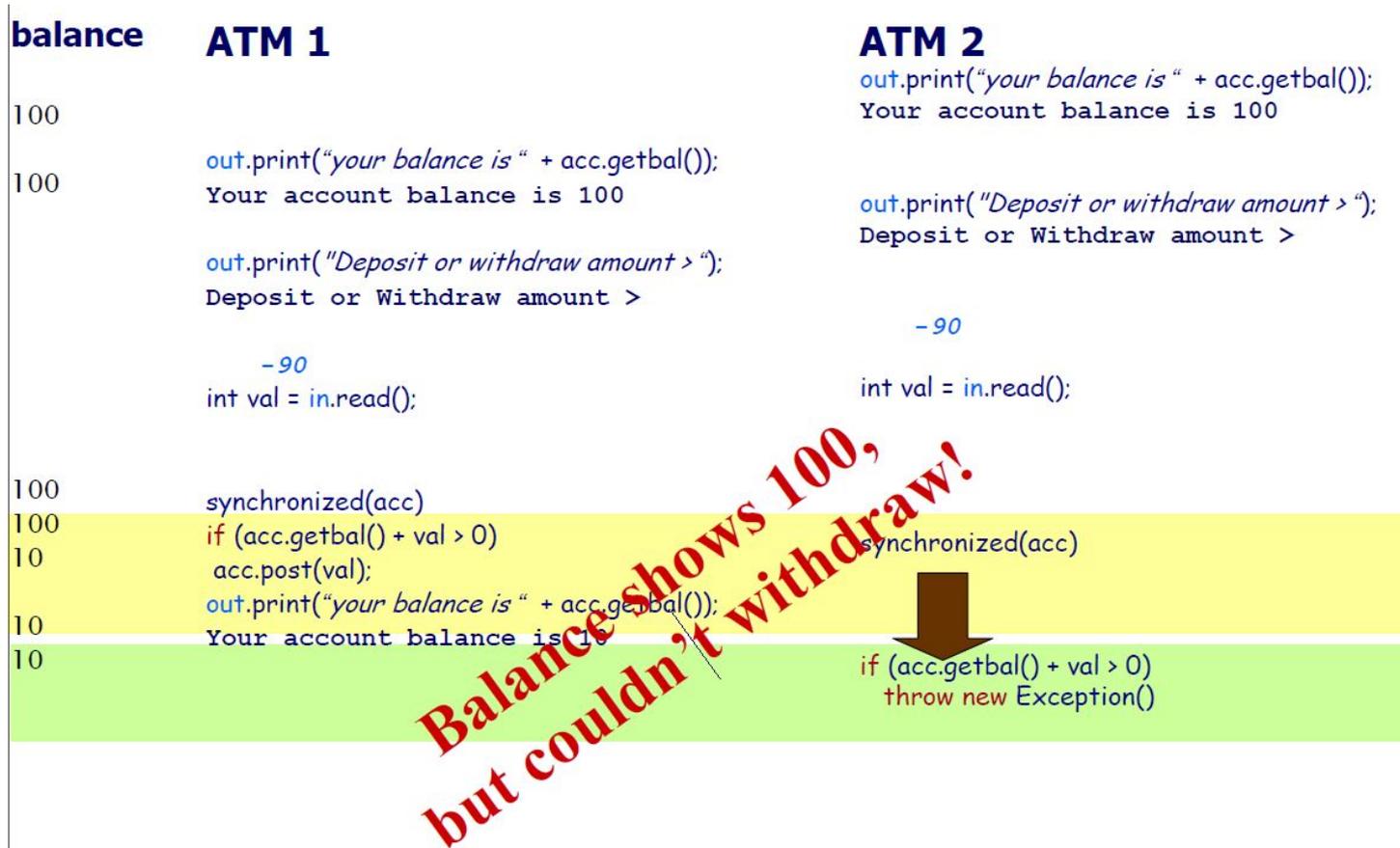
Negative Bank Balance!

Гонки потоков. Невозможность снятия.

```
public void run() {
    while(true) {
        try {
            out.print("Account ID > ");
            String id = in.readLine();
            String acc = bnk.get(id);
            if (acc == null) throw new Exception();
            out.print("Password > ");
            String pass = in.readLine();
            if (!acc.is_password(pass))
                throw new Exception();
            out.print("your balance is " + acc.getbal());
            out.print("Deposit or withdraw amount > ");
            int val = in.read();

            synchronized (acc) {
                if (acc.getbal() + val > 0)
                    acc.post(val);
                else
                    throw new Exception();
                out.print("your balance is " + acc.getbal());
            }
        } catch (Exception e) {
            out.println("Invalid input, restart");
        }
    }
}
```

ГОНКИ ПОТОКОВ. НЕВОЗМОЖНОСТЬ СНЯТИЯ.



Гонки потоков. Подвисание интерфейса.

```
synchronized (acc) {  
    out.print("your balance is " + acc.getbal());  
    out.print("Deposit or withdraw amount > ");  
    int val = in.read();  
    if (acc.getbal() + val > 0)  
        acc.post(val);  
    else  
        throw new Exception();  
    out.print("your balance is " + acc.getbal());  
}
```

Гонки потоков. Подвисание интерфейса.

ATM 1

Account ID >

ben

Password >

6189cell

`synchronized(acc)`

```
out.print("your balance is " + acc.getbal());
```

```
Your account balance is 100
```

```
out.print("Deposit or withdraw amount > ");
```

```
Deposit or Withdraw amount >
```



-90

ATM 2

Account ID >

ben

Password >

6189cell

`synchronized(acc)`



Гонки потоков. Трансфер денег.

```
public boolean transfer(Account from, Account to, int val) {  
    synchronized(from) {  
        if (from.getbal() > val)  
            from.post(-val);  
        else  
            throw new Exception();  
        synchronized(to) {  
            to.post(val);  
        }  
    }  
}
```

SharedResources. Безопасная запись.

- Доступ к ресурсу не изменяет ресурс – например операция чтения;
- Изменение данных является идемпотентным – повторные запросы на изменение приводят к одинаковому результату;
- Изменение данных выполняется только одним объектом – персональный доступ, критическая секция.

Проблемы и примеры

DeadLocks

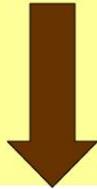
Гонки потоков. Взаимная блокировка.

Allyssa wants to transfer \$10 to Ben's account
While Ben wants to also transfer \$20 to Allyssa's account

Allyssa → Ben

```
synchronized(from)
if (from.getbal() > val)
from.post(-val);
```

```
synchronized(to)
Waiting for Ben's account
to be released to perform
```



Ben → Allysa

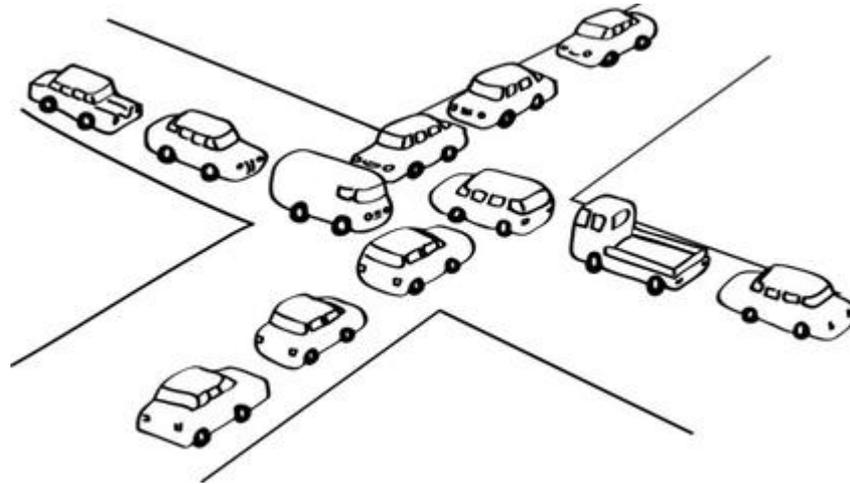
```
synchronized(from)
if (from.getbal() > val)
from.post(-val);
```

```
synchronized(to)
Waiting for Allyssa's account
to be released to perform
```



DEADLOCKED!

DeadLock



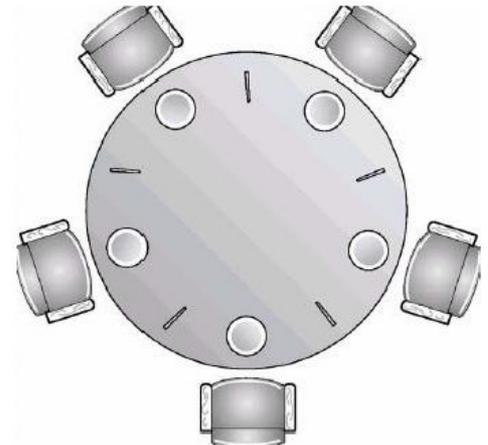
DeadLock

```
public boolean transfer(Account from,
                        Account to,
                        int val) {

    Account first = (from.rank > to.rank)?from:to;
    Account second = (from.rank > to.rank)?to:from;
    synchronized(first) {
        synchronized(second) {
            if (from.getbal() > val)
                from.post(-val);
            else
                throw new Exception();
            to.post(val);
        }
    }
}
```

Проблема обедающих философов.

- Пять философов
- Перед каждым тарелка спагетти.
- Между парой философов вилка.
- Либо есть двумя вилками
- Либо размышлять
- Может посмотреть и взять ближайшую вилку, если она доступна
- Может положить вилку



Проблема обедающих философов.

Нужен алгоритм действия философов.

Варианты?

Еще проблемы

- **Livelock** – потоки работают, но ничего не делают, потому как не могут захватить все необходимые ресурсы.
- **Starvation** (голодание) – потоку может совсем не доставаться ресурсов.
- **Lack of fairness** – кому-то достается больше ресурсов, кому то меньше.
- **Shared Resources** –
- **DeadLocks** –
- **ABA - ??**

DeadLocks

- Соблюдать последовательность входа и выхода в критические секции
- Использовать библиотечные классы (std, boost)
- Обрабатывать исключения
- Использовать правильную стратегию синхронизации

Примитивы синхронизации потоков.

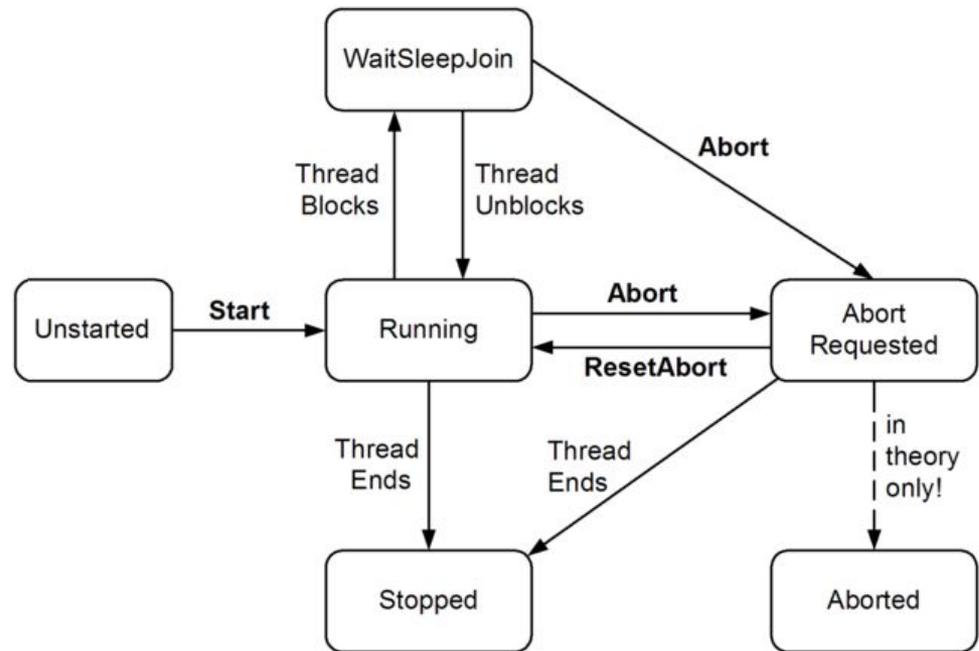
- Простые блокирующие методы
- Блокирующие конструкции
- Сигналы
- Неблокирующие конструкции

Примитивы синхронизации потоков.

Простые блокирующие методы

Простые блокирующие методы

- Wait
- Sleep
- Join



Простые блокирующие методы

Ожидание может завершиться по следующим причинам:

- Выполнилось условие ожидания
- Закончилось время ожидания
- Поток прерван функцией `TerminateThread`

Простые блокирующие методы. Условие ожидания

Минусы:

- Нагрузка на процессор
- Простои

```
while (!proceed);
```

or:

```
while (DateTime.Now < nextStartTime);
```

Простые блокирующие методы. Таймаут

`WaitForSingleObject(.. INFINITE)`

`WaitForMultipleObjects(.. INFINITE)`

Примитивы синхронизации потоков.

Блокирующие конструкции

Блокирующие конструкции

- Семафор — объект, ограничивающий количество потоков, которые могут войти в заданный участок кода.
- Мьютекс – семафор, разрешающий вход только одному потоку.

C++ Критическая секция

1. Назначение: предоставление доступа ОДНОМУ потоку
2. Скорость: высокая
3. Область видимости: процесс
4. Пример:
 1. `CRITICAL_SECTION cs;`
 2. `InitializeCriticalSection(&cs);`
 3. `EnterCriticalSection(&cs);`
 4. `// ?? Только один поток`
 5. `LeaveCriticalSection(&cs);`
 6. `DeleteCriticalSection(&cs);`

C++ Мьютекс

Именованная критическая секция, доступная для использования в рамках операционной системы.

1. Назначение: предоставление доступа ОДНОМУ потоку
2. Скорость: медленнее
3. Область видимости: ОС
4. Пример:
 1. HANDLE hMutex;
 2. hMutex = CreateMutex(NULL, false, NULL);
 3. WaitForSingleObject(hMutex, INFINITE);
 4. // ?? Только один поток
 5. ReleaseMutex(hMutex);
 6. CloseHandle(&cs);

C++ Семафор

1. Назначение: предоставление доступа **НЕСКОЛЬКИМ** потокам
2. Скорость: медленнее
3. Область видимости: ОС
4. Пример:
 1. HANDLE hSemaphore;
 2. hSemaphore = CreateSemaphore(NULL, [**CURRENT**],[**MAX**], NULL);
 3. WaitForSingleObject(hSemaphore, INFINITE);
 4. // ?? Не более **MAX** потоков
 5. ReleaseSemaphore(hSemaphore, 1, NULL);

Примитивы синхронизации потоков.

Сигналы

C++ Событие

1. HANDLE hEvent;
2. hEvent = CreateEvent(NULL, false (**autoreset event**), false, NULL);
3. WaitForSingleObject(hEvent, INFINITE);
4. // ?? один поток за счет autoreset
5. CloseHandle(hEvent);

Список литературы

Википедия

- **Введение**
 - <https://habrahabr.ru/company/piter/blog/274569>
- **Проблемы параллельного программирования**
 - https://ru.wikipedia.org/wiki/Проблема_спящего_парикмахера
 - https://ru.wikipedia.org/wiki/Проблема_обедающих_философов
- **Threading in C#**
 - <http://www.albahari.com/threading>
- **Best practices**
 - <https://msdn.microsoft.com/en-us/library/ff601929.aspx>
 - [https://msdn.microsoft.com/en-us/library/1c9txz50\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/1c9txz50(v=vs.110).aspx)

Введение в параллельное программирование

“To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.”

-- E. Dijkstra, 1972 Turing Award Lecture

Знаменитый закон Мура

Закон Мура (1965): каждые 2 года количество транзисторов в интегральной микросхеме удваивается.

Следствие Хауса: производительность центрального процессора компьютера удваивается каждые 18 месяцев.

Мур, 2007: закономерности перестанут работать вследствие атомарной природы вещества и ограничения скорости света.

Вопросы
