Уральский федеральный университет

Параллельные вычисления
Многопоточное программирование

Созыкин Андрей Владимирович

к.т.н.

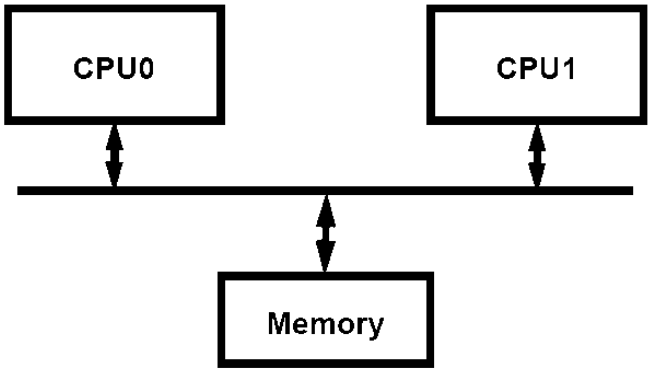
Заведующий кафедрой высокопроизводительных компьютерных технологий

Институт математики и компьютерных наук

Многопоточное программирование

Вычислительные системы с общей памятью:

- Hyper-Threading
- Многоядерные процессоры
- Многопроцессорные компьютеры



Многозадачная операционная система:

- Одновременно выполняется много программ (приложений и системных)

Процесс:

- абстракция операционной системе, позволяющая программе работать так, будто она выполняется на компьютере одна

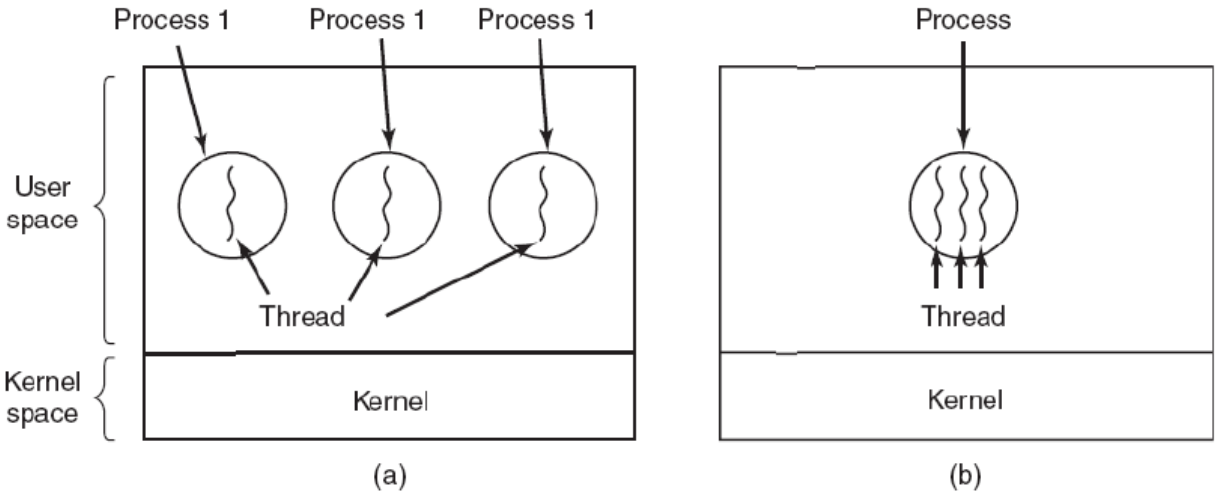
Процессы и потоки

Процесс – механизм выделения ресурсов

- Память, файлы и т.п.
- Как минимум один поток

Поток:

- Набор выполняемых команд
- Ресурсы у потоков общие



Таненбаум. Операционные системы

Зачем нужны несколько потоков?

Одновременная работа независимых задач (многозадачность)

Увеличение производительности программы

Увеличение пропускной способности

Отзывчивость графического интерфейса

Инструменты

Операционные системы:

- Pthreads (Linux, OS X), Windows threads

Библиотеки:

- Boost.threads, Intel Parallel Building Blocks

Языки программирования:

- Java, C#, C++11, Python

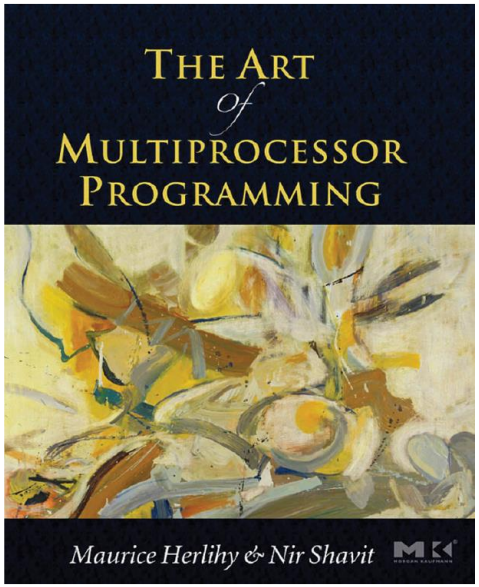
Многопоточное программирование на Java

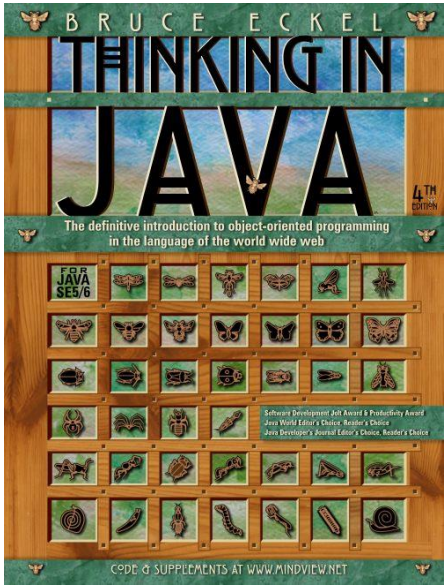
Maurice Herlihy. Nir Shavit. The Art of Multiprocessor Programming, Revised Reprint

Bruce Eckel. Thinking in Java (4th Edition)

The Java™ Tutorials. Concurrency

- http://docs.oracle.com/javase/tutorial/essential/concurrency/





Потоки в Java

Потоки в Java – класс java.lang.Thread

Потоки в Java

Потоки в Java – класс java.lang.Thread

```
public class MyThread extends Thread {

    public void run() {
      System.out.println(" Hello World ");
      // do something ...
    }

    public static void main ( String [] args ) {
      MyThread t = new MyThread ();
      t.start();
    }
}
```

java.lang.Thread

Методы класса java.lang.Thread:

- run() – Определяет, что будет делать поток
- start() – запускает НОВЫЙ поток, который выполняет метод run()

Если вместо start() вызвать run():

- новый поток не будет создан
- метод run() выполнится в текущем потоке

Интерфейс Runnable

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        HelloRunnable hr = new HelloRunnable();
        Thread t = new Thread(hr);
        t.start();
    }
}
```

Интерфейс Runnable

```
public class Main implements Runnable {
    private int id;
    public Main(int id){
        this.id = id;
    }
    public void run() {
        for (int i = 0; i < 10; i++)
            System.out.println("Hello from thread # " + id);
    }
    public static void main(String[] args) {
        Main m1 = new Main(1);
        Main m2 = new Main(2);
        Thread t1 = new Thread(m1);
        Thread t2 = new Thread(m2);
        t2.start();
        t1.start();
    }
}
```

Результаты запусков

```
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
```

```
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 2
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
```

```
Hello from thread # 2
Hello from thread # 2
Hello from thread # 1
Hello from thread # 2
Hello from thread # 1
Hello from thread # 2
Hello from thread # 1
Hello from thread # 2
Hello from thread # 1
Hello from thread # 2
Hello from thread # 1
Hello from thread # 2
Hello from thread # 1
Hello from thread # 2
Hello from thread # 1
Hello from thread # 2
Hello from thread # 1
Hello from thread # 1
Hello from thread # 1
Hello from thread # 2
```

Недетерминированность

Одно из неприятных свойств многопоточных и параллельных программ

Возможно большое количество вариантов выполнения программы

- Трасса – один из вариантов выполнения потоков в программе

Недетерминированность

Одно из неприятных свойств многопоточных и параллельных программ

Возможно большое количество вариантов выполнения программы

- Трасса – один из вариантов выполнения потоков в программе

Почему так происходит?

Недетерминированность

Одно из неприятных свойств многопоточных и параллельных программ

Возможно большое количество вариантов выполнения программы

- Трасса – один из вариантов выполнения потоков в программе

Почему так происходит?

Может ли программист задавать, какая именно трасса будет выполняться?

Недетерминированность

Одно из неприятных свойств многопоточных и параллельных программ

Возможно большое количество вариантов выполнения программы

- Трасса – один из вариантов выполнения потоков в программе

Многопоточные программы сложно отлаживать

Невозможно тестировать традиционными средствами

Банк

```
public class Bank {

    private Map <String , Account > accounts ;

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

    public void addAccount ( Account account ) {
        accounts.put(account.getId(), account );
    }

    public Account getAccount( String id) {
        return accounts.get(id );
    }

}
```

Банковский счет

```
public class Account {
    private String id;
    private int balance ;

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

   public String getId() {
       return id;
   }

   public int getBalance() {
       return balance ;
   }

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

Клиент банка

```
public class Client extends Thread {
    private int id;
    private Bank bank ;
    private BufferedReader in;
    private PrintStream out ;

    public Client (int id, Bank bank, BufferedReader in, PrintStream out) {
       // Initialization
    }

    public void run () {
       // Next slide
    }

    public static void main ( String [] args ) {
        Bank bank = new Bank();
        bank.addAccount( new Account("Acc001", 100));
        BufferedReader in = new BufferedReader(
            new InputStreamReader( System.in ));
        Client client = new Client (0, bank , in , System.out);
        client.start();
}
```

Клиент банка, метод run

```
public void run () {
  while ( true ) {
    try {
      out.print(" Account Id: ");
      String accountId = in.readLine();
      Account account = bank.getAccount( accountId );
        if ( account == null )
          throw new Exception (" Account not found !");
        else
          out.println (" Balance : " + account.getBalance());
      out.print (" Enter amount : ");
      int value = Integer.parseInt (in.readLine());
      if (account.getBalance () + value >= 0) {
        account.post(value);
        out.println(" Balance : " + account.getBalance());
      } else
        throw new Exception (" Not enough money !");
    } catch ( Exception e) {
      out.println(" Error ! " + e. getMessage ());
    }
  }
}
```

Запуск последовательной версии

```
Account Id: Acc000
Error ! Account not found !
Account Id: Acc001
Balance : 100
Enter amount : -90
Balance : 10
Account Id: Acc001
Balance : 10
Enter amount : -90
Error ! Not enough money !
```

Параллельный запуск клиентов

```
public static void main(String[] args ) throws IOException {
  int numClients = 2;
  Bank bank = new Bank();
  bank.addAccount(new Account("Acc001", 100));
  Client[] clients = new Client[numClients];
  for (int i =0; i< numClients ; i ++) {
    File inFile = new File ("input" + (i +1));
    BufferedReader in = new BufferedReader(
      new InputStreamReader(new FileInputStream(inFile)));
   clients[i] = new Client (i+1, bank, in, System.out );
   clients[i].start();
}
```

Входные данные

```
cat input1
Acc001
-90

cat input2
Acc01
-90
```

Результаты запусков

```
[1] Account Id: Acc001
[2] Account Id: Acc001
[2] Balance: 100
[2] Enter amount: -90
[2] Balance: 10
[1] Balance: 10
[1] Enter amount: -90
[1] Error! Not enough money!
```

```
[1] Account Id: Acc001
[1] Balance: 100
[2] Account Id: Acc001
[1] Enter amount: -90
[2] Balance: 100
[2] Enter amount: -90
[1] Balance: 10
[2] Error! Not enough money!
```

```
[1] Account Id: Acc001
[2] Account Id: Acc001
[1] Balance: 100
[1] Enter amount: -90
[2] Balance: 100
[2] Enter amount: -90
[1] Balance: 10
[2] Balance: -80
```

```
[1] Account Id: Acc001
[2] Account Id: Acc001
[2] Balance: 100
[2] Enter amount: -90
[1] Balance: 100
[1] Enter amount: -90
[1] Balance: -80
[2] Balance: -80
```

```
[1] Account Id: Acc001
[1] Balance: 100
[1] Enter amount: -90
[2] Account Id: Acc001
[2] Balance: 100
[2] Enter amount: -90
[2] Balance: 10
[1] Balance: 10
```

Условия гонок (race conditions)

Некорректное поведение программы при одновременной работе нескольких потоков с общими данными

Условия возникновения гонок

- Несколько потоков модифицируют общие данные

Почему возникают проблемы

Метод:

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

Как это выполняется:

- balance загружается в регистр
- value загружается в регистр
- выполняется сложение
- результат записывается в balance

Почему возникают проблемы

Метод:

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

Как это выполняется:

- balance загружается в регистр
- value загружается в регистр
- выполняется сложение

Что будет, если здесь поток прервётся?

- результат записывается в balance

Синхронизация

Безопасный доступ к данным:

- Устранение гонок

Координация действий между потоками:

- Условная синхронизация (следующая лекция)

Взаимное исключение (Mutual Exclusion)

Критическая секция:

- Часть кода, где выполняются потенциально опасные действия с общими данными

Необходимо, чтобы критическая секция выполнялась только одним потоком:

- В каждый момент времени в критической секции только один поток
- Остальные потоки ждут завершения выполнения критической секции

Мониторы в Java

Самый простой способ реализации взаимного исключения

Монитор включает:

- Объект
- Механизмы блокировки для взаимного исключения
- Механизмы условной синхронизации

Ключевое слово:

- synchronized
- добавляется к методам
- используется с объектами

Мониторы в Java

```
synchronized public int getBalance () {
    return balance ;
}

synchronized public void post (int value ) {
    balance += value ;
}
```

Метод synchronized

```
synchronized public int getBalance () {
    return balance ;
}

synchronized public void post (int value ) {
    balance += value ;
}
```

Все synchronized методы класса – ОДНА критическая секция

Результаты запусков

```
[1] Account Id: Acc001
[2] Account Id: Acc001
[2] Balance: 100
[2] Enter amount: -90
[2] Balance: 10
[1] Balance: 10
[1] Enter amount: -90
[1] Error! Not enough money!
```

```
[1] Account Id: Acc001
[1] Balance: 100
[2] Account Id: Acc001
[1] Enter amount: -90
[2] Balance: 100
[2] Enter amount: -90
[1] Balance: 10
[2] Error! Not enough money!
```

```
[1] Account Id: Acc001
[2] Account Id: Acc001
[1] Balance: 100
[1] Enter amount: -90
[2] Balance: 100
[2] Enter amount: -90
[1] Balance: 10
[2] Balance: -80
```

```
[1] Account Id: Acc001
[2] Account Id: Acc001
[2] Balance: 100
[2] Enter amount: -90
[1] Balance: 100
[1] Enter amount: -90
[1] Balance: -80
[2] Balance: -80
```

Блок synchronized

```
out.println ("[" + id + "] Balance : " + account.getBalance());
out.println ("[" + id + "] Enter amount : ");

int value = Integer.parseInt(in.readLine());

synchronized ( account ) {
    if ( account.getBalance () + value >= 0) {
        account.post( value );
        out.println ("[" + id + "] Balance : "
          + account . getBalance ());
    } else {
        throw new Exception ("Not enough money !");
    }
}
```

Результаты запусков

```
[1] Account Id: Acc001
[2] Account Id: Acc001
[2] Balance: 100
[2] Enter amount: -90
[2] Balance: 10
[1] Balance: 10
[1] Enter amount: -90
[1] Error! Not enough money!
```

```
[1] Account Id: Acc001
[1] Balance: 100
[2] Account Id: Acc001
[1] Enter amount: -90
[2] Balance: 100
[2] Enter amount: -90
[1] Balance: 10
[2] Error! Not enough money!
```

Блок synchronized

```
synchronized ( account ) {
    out.println ("[" + id + "] Balance : "
        + account.getBalance());
    out.println ("[" + id + "] Enter amount : ");

    int value = Integer.parseInt(in.readLine());

    if ( account.getBalance () + value >= 0) {
        account.post( value );
        out.println ("[" + id + "] Balance : "
          + account . getBalance ());
    } else {
        throw new Exception ("Not enough money !");
    }
}
```

Результаты запусков

```
[1] Account Id: Acc001
[2] Account Id: Acc001
[2] Balance: 100
[2] Enter amount: -90
[2] Balance: 10
[1] Balance: 10
[1] Enter amount: -90
[1] Error! Not enough money!
```

Мониторы в Java

Объявим все методы synchronized?

Мониторы в Java

Объявим все методы synchronized?

Синхронизация требует высоких накладных расходов

Блоки и секции synchronized должны быть как можно меньше

- Иначе многопоточное выполнение превращается в последовательное

Банк. Перевод денег

```
public void transfer(Account from, Account to, int value)
  throws Exception {
    if ( value <= 0) {
      throw new Exception (" Amount must be positive !");
    }
    if (from.getBalance () < value ) {
      throw new Exception ("Not enough money !");
    } else {
      from.post(-value);
      to.post(value);
    }
}
```

Банк. Перевод денег с синхронизацией

```
public void transfer(Account from, Account to, int value)
  throws Exception {
    if ( value <= 0) {
      throw new Exception ("Amount must be positive!");
    }
        synchronized ( from ) {
      if (from.getBalance () < value ) {
        throw new Exception ("Not enough money!");
      } else {
        from.post(-value);
      }
      synchronized (to) {
        to.post(value);
      }
    }
}
```

Входные данные

```
cat input1
Acc001
Acc002
10

cat input2
Acc002
Acc001
20
```

Результаты запуска

```
[2] From Account Id: Acc002
[2] From Account Id: Acc001
[2] Enter transfer amount: 20
[2] Balance: 80
[1] From Account Id: Acc001
[1] From Account Id: Acc002
[1] Enter transfer amount: 10
[1] Balance: 110
```

```
[1] From Account Id: Acc001
[1] From Account Id: Acc002
[1] Enter transfer amount: 10
[1] Balance: 90
[2] From Account Id: Acc002
[2] From Account Id: Acc001
[2] Enter transfer amount: 20
[2] Balance: 90
```

Результаты запуска

```
[2] From Account Id: Acc002
[2] From Account Id: Acc001
[2] Enter transfer amount: 20
[2] Balance: 80
[1] From Account Id: Acc001
[1] From Account Id: Acc002
[1] Enter transfer amount: 10
[1] Balance: 110
```

```
[1] From Account Id: Acc001
[1] From Account Id: Acc002
[1] Enter transfer amount: 10
[1] Balance: 90
[2] From Account Id: Acc002
[2] From Account Id: Acc001
[2] Enter transfer amount: 20
[2] Balance: 90
```

```
[1] From Account Id: Acc001
[1] From Account Id: Acc002
[1] Enter transfer amount: 10
[2] From Account Id: Acc002
[2] From Account Id: Acc001
[2] Enter transfer amount: 20
```

Взаимоблокировка (deadlock)

Как избежать взаимоблокировки

Захватывать только одну блокировку

Захватывать блокировки в одном порядке

Добровольно освобождать захваченную блокировку (после таймаута)

Упорядочивание блокировок

```
int fromHash = System.identityHashCode(from);
int toHash = System.identityHashCode(to);

if ( fromHash < toHash ) {
  synchronized (from) {
    synchronized (to) {
      doTransfer (from, to, value);
    }
  }
} else if (fromHash > toHash ) {
  synchronized (to) {
    synchronized (from ) {
      doTransfer (from, to, value);
}
```

Свойства параллельной программы

Безопасность (Safety)

- Программа никогда не попадает в «плохое» состояние

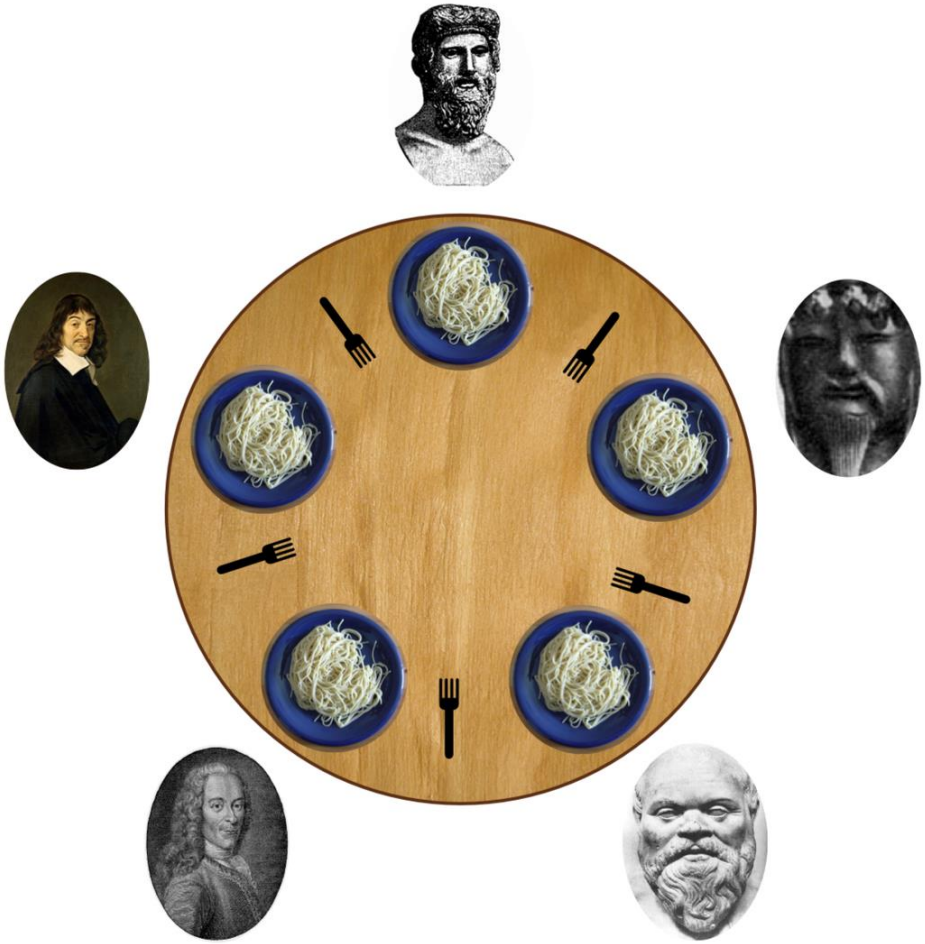
Живучесть (Liveness)

- Программа обязательно попадает в «хорошее» состояние

Справедливость

- Все потоки одинаково обеспечены ресурсами

Обедающие философы



Домашнее задание №1

Подробное описание задания и последовательные заготовки на Java на сайте:

- https://clck.ru/8rEVG

Сдавать задачи через систему anytask

Писать можно на Java или C++

Дедлайн:

- Пока не определен
- 2 недели после второй части первого домашнего задания

Вопросы?