Многопоточность 1

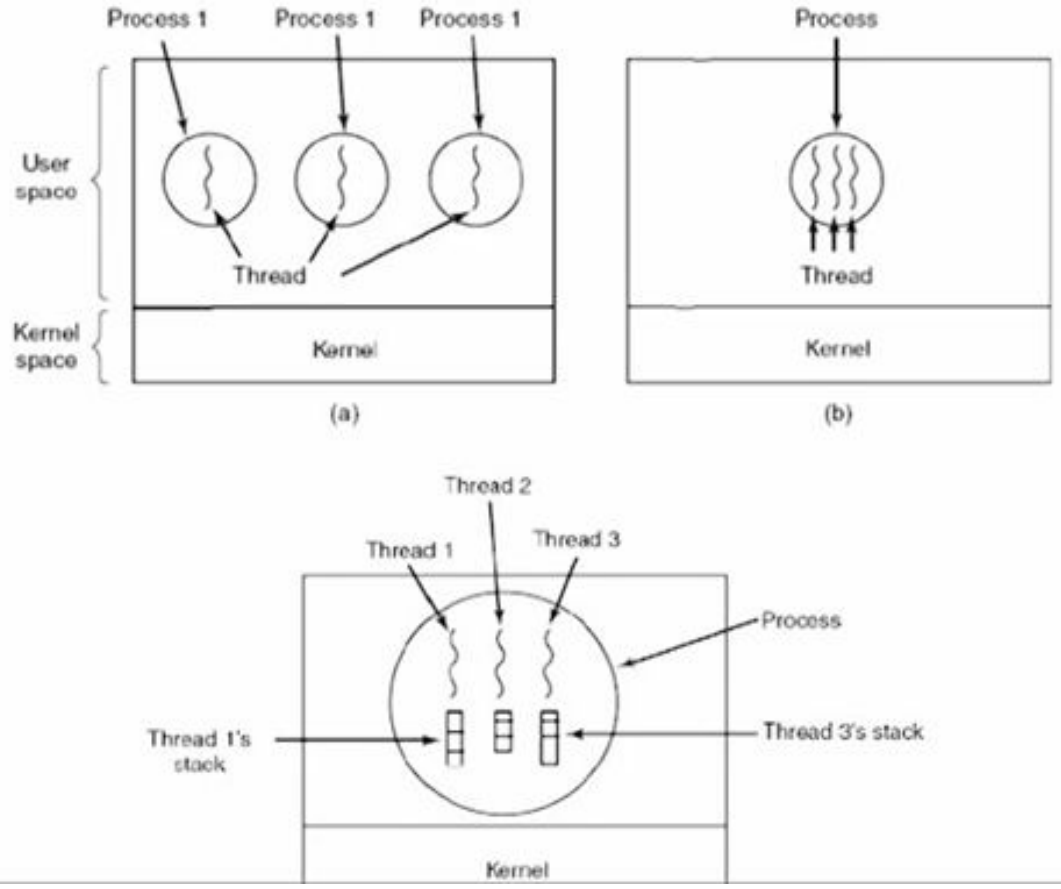
Concurrency

- Последовательная программа
  - Один процесс, последовательно выполняющий инструкции
- "Многопроцессная"(concurrent) программа
  - Несколько процессов, выполняющихся "одновременно"
- Варианты размещения процессов
  - Один процессор(ядро)
  - Многопроцессорная(-ядерная) система
  - Несколько машин

Зачем нужно несколько процессов?

- Естественность и удобство реализации
- Взаимодействие с пользователем, GUI
- Увеличение пропускной способности
- Увеличение скорости решения задачи

Процессы и потоки



Реализации потоков

- ОС: Pthreads (POSIX), Win32 threads, Solaris threads
- Библиотеки: Boost.Thread, OpenMP, Intel Threading Building Blocks
- Языки: Java, C#, Python, Erlang

Класс java.lang.Thread (Поток)

lecture1.helloworld.MyThread

```
1  public class MyThread extends Thread {
2
3      public void run() {
4          System.out.println("Hello World");
5          // do something...
6      }
7
8      public static void main(String[] args) {
9          MyThread t = new MyThread();
10         t.start();
11     }
12
13 }
```

Класс Person

lecture1.helloworld.Person

```
1  public class Person {
2
3      private String name;
4
5      public Person(String name) {
6          this.name = name;
7      }
8
9      public String getName() {
10         return name;
11     }
12
13 }
```

Интерфейс java.lang.Runnable (Задание)

lecture1.helloworld.RunnablePerson

```
1  public class RunnablePerson extends Person implements Runnable {
2
3      public RunnablePerson(String name) {
4          super(name);
5      }
6
7      public void run() {
8          for (int i=0; i<10; i++) {
9              System.out.println(getName() + ": Hello World");
10         }
11     }
12
13     public static void main(String[] args) {
14         RunnablePerson p1 = new RunnablePerson("Alice");
15         Thread t1 = new Thread(p1);
16         t1.start();
17         RunnablePerson p2 = new RunnablePerson("Bob");
18         Thread t2 = new Thread(p2);
19         t2.start();
20     }
21
22 }
```

Результаты запуска RunnablePerson

Bob: Hello World	Alice: Hello World	Alice: Hello World
Alice: Hello World	Alice: Hello World	Alice: Hello World
Bob: Hello World	Alice: Hello World	Alice: Hello World
Bob: Hello World	Bob: Hello World	Bob: Hello World
Alice: Hello World	Bob: Hello World	Bob: Hello World
Alice: Hello World	Bob: Hello World	Bob: Hello World
Alice: Hello World	Bob: Hello World	Bob: Hello World
Alice: Hello World	Bob: Hello World	Bob: Hello World
Alice: Hello World	Bob: Hello World	Bob: Hello World
Alice: Hello World	Bob: Hello World	Alice: Hello World
Alice: Hello World	Bob: Hello World	Alice: Hello World
Alice: Hello World	Bob: Hello World	Alice: Hello World
Alice: Hello World	Bob: Hello World	Alice: Hello World
Bob: Hello World	Alice: Hello World	Alice: Hello World
Bob: Hello World	Alice: Hello World	Alice: Hello World
Bob: Hello World	Alice: Hello World	Alice: Hello World
Bob: Hello World	Alice: Hello World	Bob: Hello World
Bob: Hello World	Alice: Hello World	Bob: Hello World
Bob: Hello World	Alice: Hello World	Bob: Hello World

Банк

lecture1.bank.v1_seq.Bank

```
1  public class Bank {
2
3      private Map<String,Account> accounts;
4
5      public Bank() {
6          accounts = new HashMap<String,Account>();
7      }
8
9      public void addAccount(Account account) {
10         accounts.put(account.getId(), account);
11     }
12
13     public Account getAccount(String id) {
14         return accounts.get(id);
15     }
16
17 }
```

Банковский счет

lecture1.bank.v1_seq.Account

```
1  public class Account {
2
3      private String id;
4      private int balance;
5
6      public Account(String id, int balance) {
7          this.id = id;
8          this.balance = balance;
9      }
10
11     public String getId() {
12         return id;
13     }
14
15     public int getBalance() {
16         return balance;
17     }
18
19     public void post(int value) {
20         balance += value;
21     }
```

Клиент v1

lecture1.bank.v1_seq.Client

```
1  public class Client {
2
3      private Bank bank;
4      private BufferedReader in;
5      private PrintStream out;
6
7      public Client(Bank bank, BufferedReader in, PrintStream out) {
8          ...
9      }
10
11     public void run() {
12         ...
13     }
14
15     public static void main(String[] args) {
16         Bank bank = new Bank();
17         bank.addAccount(new Account("Acc001", 100));
18         BufferedReader in = new BufferedReader(
19                 new InputStreamReader(System.in));
20         Client client = new Client(bank, in, System.out);
21         client.run();
22     }
```

Клиент v1 - run()

```
1  public void run() {
2      while(true) {
3          try {
4              out.print("Account Id: ");
5              String accountId = in.readLine();
6              Account account = bank.getAccount(accountId);
7              if (account == null) {
8                  throw new Exception("Account not found!");
9              } else {
10                 out.println("Balance: " + account.getBalance());
11             }
12             out.print("Enter amount: ");
13             int value = Integer.parseInt(in.readLine());
14             if (account.getBalance() + value >= 0) {
15                 account.post(value);
16                 out.println("Balance: " + account.getBalance());
17             } else {
18                 throw new Exception("Not enough money!");
19             }
20         } catch (Exception e) {
21             out.println("Error! " + e.getMessage());
22         }
23     }
```

Запуск

```
1  Account Id: Acc000
2  Error! Account not found!
3  Account Id: Acc001
4  Balance: 100
5  Enter amount: -90
6  Balance: 10
7  Account Id: Acc001
8  Balance: 10
9  Enter amount: -90
10 Error! Not enough money!
```

Клиент v2

lecture1.bank.v2_concurr.Client

```
1  public class Client extends Thread {
2
3      static final int numClients = 2;
4
5      private int id;
6      private Bank bank;
7      private BufferedReader in;
8      private PrintStream out;
9
10     public Client(int id, Bank bank,
11             BufferedReader in, PrintStream out) {
12         ...
13     }
14
15     public void run() {
16         ...
17     }
18
19     public static void main(String[] args) {
20         ...
21     }
```

Клиент v2 - main()

lecture1.bank.v2_concurr.Client

```
1  public static void main(String[] args) throws IOException {
2      Bank bank = new Bank();
3      bank.addAccount(new Account("Acc001", 100));
4      Client[] clients = new Client[numClients];
5      for (int i=0; i<numClients; i++) {
6          File inFile = new File(
7              "src/lecture1/bank/v2_concurr/input" + (i+1));
8          BufferedReader in = new BufferedReader(
9              new InputStreamReader(new FileInputStream(inFile)));
10         clients[i] = new Client(i+1, bank, in, System.out);
11         clients[i].start();
12     }
13 }
```

Входные данные

src/lecture1/bank/v2_concurr/input1

```
1 Acc001
2 -90
```

src/lecture1/bank/v2_concurr/input2

```
1 Acc001
2 -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**

Отрицательный баланс

lecture1.bank.v2_concurr.Client

```
1  public void run() {
2      //while(true) { // run one transaction for testing
3            ...
4
5      out.println("[" + id + "]Enter amount: ");
6      int value = Integer.parseInt(in.readLine());
7      if (account.getBalance() + value >= 0) {
8          Thread.sleep(100);
9          account.post(value);
10         Thread.sleep(100);
11         out.println("[" + id + "]Balance: "
12                 + account.getBalance());
13     } else {
14         throw new Exception("Not enough money!");
15     }
16
17     ...
18     //}
19 }
```

Возможная трасса run()

Client1

```
int value = Integer.parseInt(in.readLine());
if (account.getBalance() + value >= 0) {
```

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

Client2

```
int value = Integer.parseInt(in.readLine());
if (account.getBalance() + value >= 0) {
account.post(value);
```

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

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**

Возможная трасса post()

```
1 public void post(int value) {
2     balance += value;
3 }
```

Client1	Client2
value (-90)	
	value (-90)
balance (100)	
+ (10)	
	balance (100)
	+ (10)
	balance = (10)
balance = (10)	

Состояние гонки (Race Condition)

- Не все варианты чередования потоков (трассы), реализующиеся при выполнении программы, приводят к корректному результату
- Часто возникает при доступе к общим данным
- Трудно обнаруживать, воспроизводить и устранять

Безопасность (Safety)

Программа никогда не попадает в "плохое"состояние, в котором некоторые переменные имеют нежелательные значения

Синхронизация

- Безопасный доступ к общим данным
  - Устранение состояния гонок и нежелательных трасс программы
- Координация действий между потоками
  - Условная синхронизация (см. следующую лекцию)

Взаимное исключение (Mutual Exclusion)

- Требуется запретить выполнение критической секции более чем одним потоком (атомарность)
- В каждый момент времени внутри критической секции может находиться только один поток
- Остальные потоки ждут окончания выполнения секции первым потоком

Модификатор synchronized

lecture1.bank.v3_sync.Account

```
1  synchronized public int getBalance() {
2      return balance;
3  }
4
5  synchronized public void post(int value) {
6      balance += value;
7  }
```

Возможная трасса post()

```
1  public void post(int value) {
2      balance += value;
3  }
```

Client1	Client2
value (-90)	
	value (-90)
balance (100)	
+ (10)	
	balance (100)
	+ (10)
	balance = (10)
balance = (10)	

Модификатор synchronized

lecture1.bank.v3_sync.Account

```
1  synchronized public int getBalance() {
2      return balance;
3  }
4
5  synchronized public void post(int value) {
6      balance += value;
7  }
```

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

lecture1.bank.v3_sync.Client

```
1  out.println("[" + id + "]Balance: " + account.getBalance())
2  out.println("[" + id + "]Enter amount: ");
3  int value = Integer.parseInt(in.readLine());
4
5  synchronized (account) {
6      if (account.getBalance() + value >= 0) {
7          account.post(value);
8          out.println("[" + id + "]Balance: "
9                  + account.getBalance());
10     } else {
11         throw new Exception("Not enough money!");
12     }
13 }
```

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
```

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!

Результат

[1] Account Id: Acc001
[2] Account Id: Acc001
[1] Balance: 100
[1] Enter amount: (1 ДУМАЕТ... 2 ЖДЕТ...) -90
[1] Balance: 10
[2] Balance: 10

Bank

```
1  public void transfer(Account from, Account to, int value)
2          throws Exception {
3      if (value <= 0) {
4          throw new Exception("Amount must be positive!");
5      }
6      if (from.getBalance() < value) {
7          throw new Exception("Not enough money!");
8      } else {
9          from.post(-value);
10     }
11     to.post(value);
12 }
```

Bank

lecture1.bank.v4_deadlock.Bank

```
1  public void transfer(Account from, Account to, int value)
2          throws Exception {
3      if (value <= 0) {
4          throw new Exception("Amount must be positive!");
5      }
6      synchronized (from) {
7          if (from.getBalance() < value) {
8              throw new Exception("Not enough money!");
9          } else {
10             from.post(-value);
11         }
12         synchronized (to) {
13             to.post(value);
14         }
15     }
16 }
```

Client

lecture1.bank.v4_deadlock.Client

```
1  out.println("[" + id + "]" + "From Account Id: ");
2  String accountId = in.readLine();
3  Account account = bank.getAccount(accountId);
4  if (account == null) {
5      throw new Exception("Account not found!");
6  }
7
8  out.println("[" + id + "]" + "To Account Id: ");
9  String toAccountId = in.readLine();
10 Account toAccount = bank.getAccount(toAccountId);
11 if (toAccount == null) {
12     throw new Exception("Account not found!");
13 }
14
15 out.println("[" + id + "]Enter transfer amount: ");
16 int value = Integer.parseInt(in.readLine());
17 bank.transfer(account, toAccount, value);
18 out.println("[" + id + "]Balance: " + account.getBalance());
```

Входные данные

src/lecture1/bank/v4_deadlock/input1

```
1 Acc001
2 Acc002
3 10
```

src/lecture1/bank/v4_deadlock/input2

```
1 Acc002
2 Acc001
3 20
```

Результаты

```
[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
...
```

Bank

lecture1.bank.v4_deadlock.Bank

```
1  public void transfer(Account from, Account to, int value)
2          throws Exception {
3      if (value <= 0) {
4          throw new Exception("Amount must be positive!");
5      }
6      synchronized (from) {
7          if (from.getBalance() < value) {
8              throw new Exception("Not enough money!");
9          } else {
10             from.post(-value);
11         }
12         synchronized (to) {
13             to.post(value);
14         }
15     }
16 }
```

Взаимная блокировка (Deadlock)

Живучесть (Liveness)

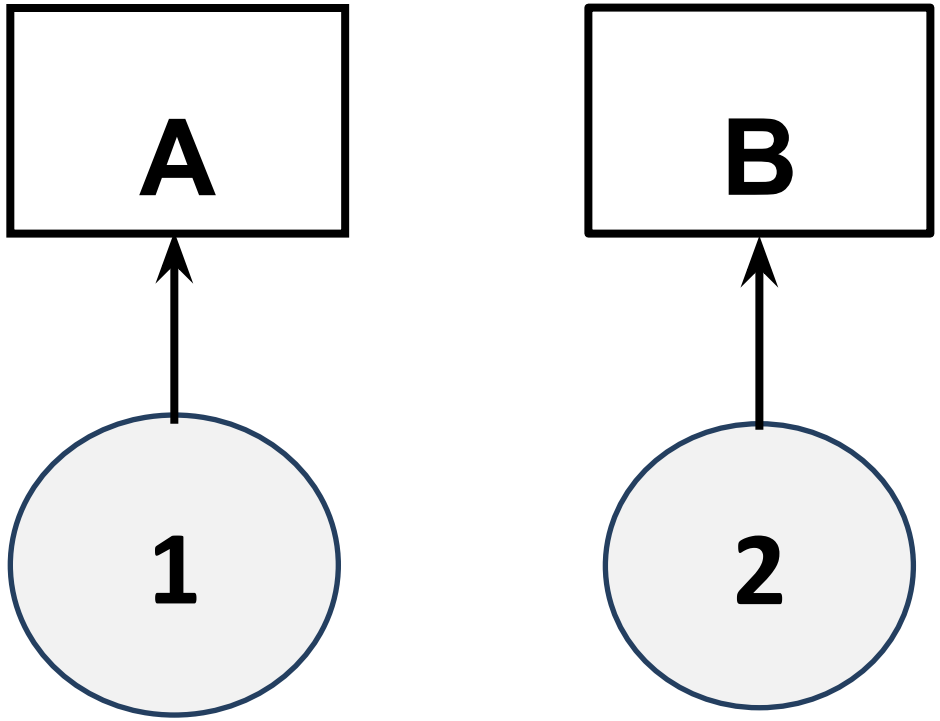
Программа в конце концов попадает в "хорошее"состояние, в котором все переменные имеют желаемые значения

Как избежать взаимной блокировки?

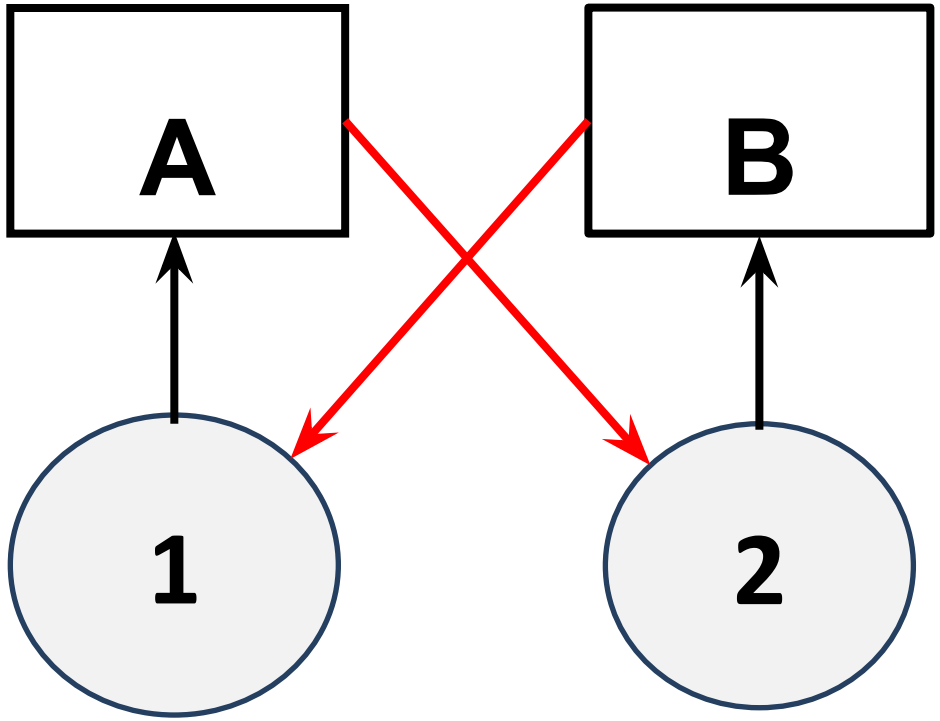
Условия возникновения взаимной блокировки

- Цикл в графе блокировок
- Каждый поток захватил как минимум один объект (lock) и ждет освобождения объекта, который захвачен следующим потоком в цикле
- Ни один из потоков не хочет уступать

Ни один из потоков не хочет уступать



Ни один из потоков не хочет уступать



Как избежать взаимной блокировки

Устранить любое из условий возникновения:

- Не захватывать более одной блокировки одновременно
- Всегда захватывать блокировки в одном порядке (упорядочить блокируемые объекты)
- Добровольно освобождать захваченную блокировку

Модифицированный Bank

lecture1.bank.v5_lockorder.Bank

```
1  int fromHash = System.identityHashCode(from);
2  int toHash = System.identityHashCode(to);
3  if (fromHash < toHash) {
4      synchronized (from) {
5          synchronized (to) {
6              doTransfer(from, to, value);
7          }
8      }
9  } else if (fromHash > toHash) {
10     synchronized (to) {
11         synchronized (from) {
12             doTransfer(from, to, value);
13         }
14     }
15 } else { // hash collision!
16     synchronized (tieLock) {
17         synchronized (from) {
18             synchronized (to) {
19                 doTransfer(from, to, value);
20             }
21         }
22     }
```

Пакет java.util.concurrent.locks

java.util.concurrent.locks.Lock

```
1  public interface Lock {
2      void lock();
3      void lockInterruptibly() throws InterruptedException;
4      boolean tryLock();
5      boolean tryLock(long timeout, TimeUnit unit)
6          throws InterruptedException;
7      void unlock();
8      ...
9  }
```

Схема использования

```
1  Lock lock = new ReentrantLock();
2  ...
3  lock.lock();
4  try {
5      // update object state
6      // catch exceptions and restore invariants if necessary
7  } finally {
8      lock.unlock();
9  }
```

Еще одна реализация перевода

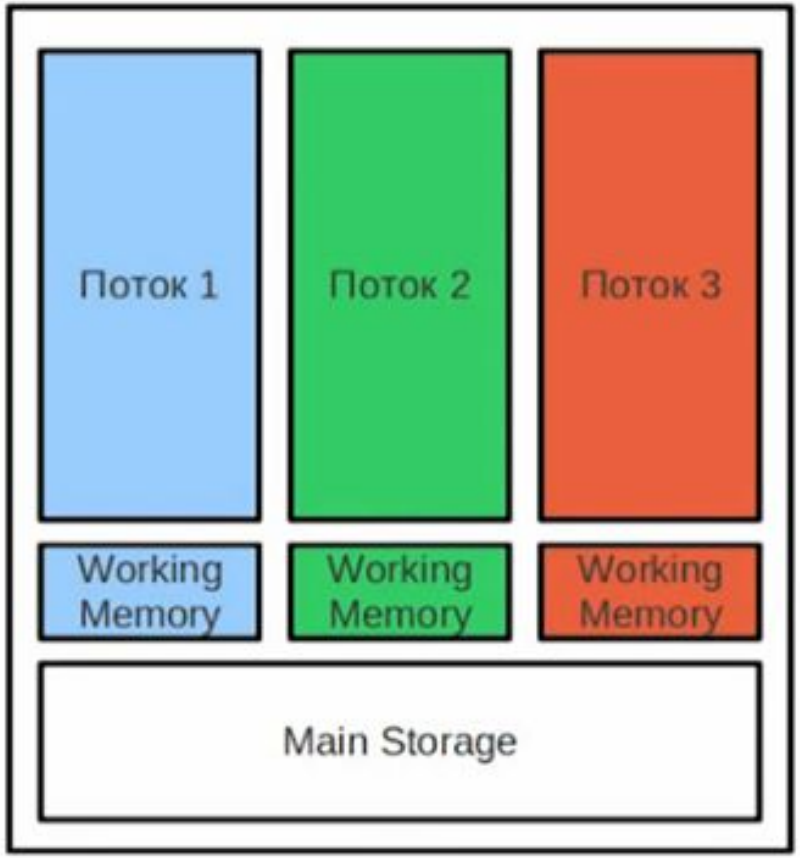
```
1  while (true) {
2      if (fromAcct.lock.tryLock()) {
3          try {
4              if (toAcct.lock.tryLock()) {
5                  try {
6                      if (fromAcct.getBalance().compareTo(amount) < 0)
7                          throw new InsufficientFundsException();
8                      else {
9                          fromAcct.debit(amount);
10                         toAcct.credit(amount);
11                         return true;
12                     }
13                 } finally {
14                     toAcct.lock.unlock();
15                 }
16             }
17         } finally {
18             fromAcct.lock.unlock();
19         }
20     }
21     if (System.nanoTime() < stopTime)
22         return false;
23     TimeUnit.NANOSECONDS.sleep(fixedDelay + rnd.nextLong() % randMod);
24 }
```

Остановка потока

lecture1.stop.StopThread

```
1  public class StopThread {
2
3      private static boolean stopRequested;
4
5      public static void main(String[] args)
6              throws InterruptedException {
7          Thread backgroundThread = new Thread(new Runnable() {
8              public void run() {
9                  int i = 0;
10                 while (!stopRequested)
11                     i++;
12             }
13         });
14         backgroundThread.start();
15         TimeUnit.SECONDS.sleep(1);
16         stopRequested = true;
17     }
18
19 }
```

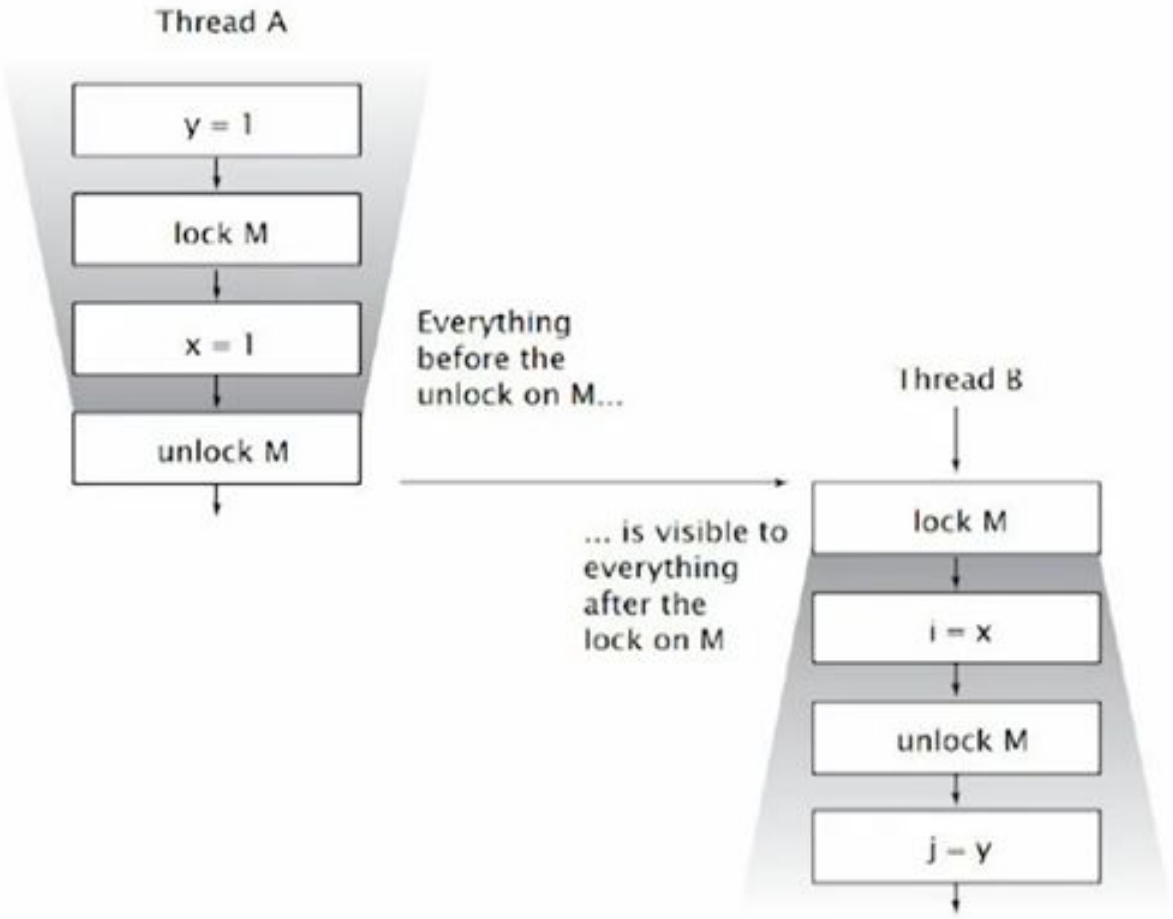
Java Memory Model



Пример с двумя переменными

```
1  public class NoVisibility {
2      private static boolean ready;
3      private static int number;
4
5      private static class ReaderThread extends Thread {
6          public void run() {
7              while (!ready)
8                  Thread.yield();
9              System.out.println(number);
10         }
11     }
12
13     public static void main(String[] args) {
14         new ReaderThread().start();
15         number = 42;
16         ready = true;
17     }
18 }
```

Обеспечение видимости изменений между потоками



Остановка потока

lecture1.stop.StopThread

```
1  public class StopThread {
2
3      private static boolean stopRequested;
4
5      public static void main(String[] args)
6              throws InterruptedException {
7          Thread backgroundThread = new Thread(new Runnable() {
8              public void run() {
9                  int i = 0;
10                 while (!stopRequested)
11                     i++;
12             }
13         });
14         backgroundThread.start();
15         TimeUnit.SECONDS.sleep(1);
16         stopRequested = true;
17     }
18
19 }
```

StopThread2

lecture1.stop.StopThread2

```
1  public class StopThread2 {
2
3      private static boolean stopRequested;
4
5      private static synchronized void requestStop() {
6          stopRequested = true;
7      }
8
9      private static synchronized boolean stopRequested() {
10         return stopRequested;
11     }
12
13     public static void main(String[] args) throws InterruptedException {
14         Thread backgroundThread = new Thread(new Runnable() {
15             public void run() {
16                 int i = 0;
17                 while (!stopRequested())
18                     i++;
19             }
20         });
21         backgroundThread.start();
22         TimeUnit.SECONDS.sleep(1);
23         requestStop();
24     }
25
26 }
```

Остановка потока

- Проверка флага (volatile!) внутри run()
- Thread.interrupt()
  - для заблокированных потоков выбрасывает InterruptedException
    - Работает для sleep(), join(), wait()
    - Не работает для I/O, synchronized
  - для не заблокированных потоков проверка Thread.interrupted()

Общая схема реализации

```
1  volatile boolean doneFlag;
2
3  public void run() {
4      try {
5          while(!doneFlag && !Thread.interrupted()) {
6              // do something...
7              // sleep...
8          }
9          // Exiting via while test
10     } catch (InterruptedException e) {
11         // Exiting via InterruptedException
12     }
13 }
```

Методы yield() и sleep()

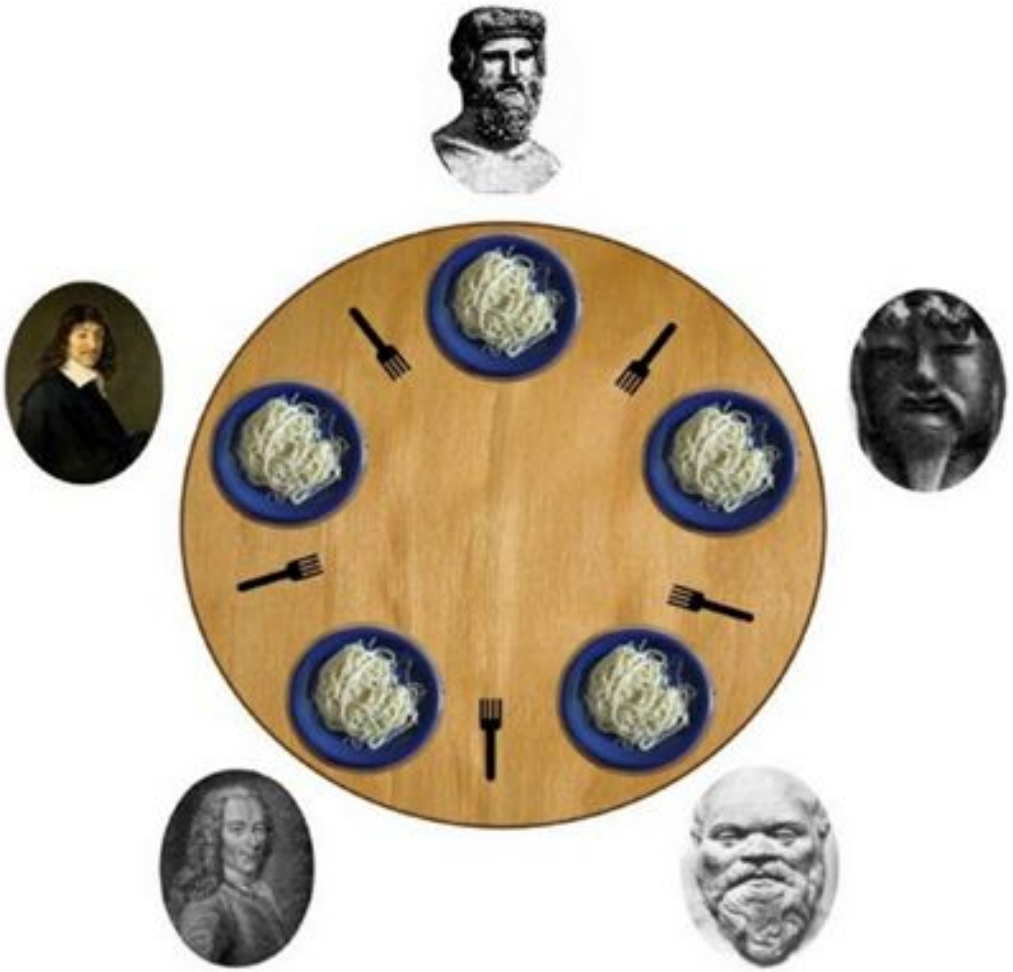
```
1  public void run() {
2      while(...) {
3          // do something
4
5          Thread.yield();
6      }
7  }
```

```
1  public void run() {
2      try {
3          while(...) {
4              // do something
5
6              // Thread.sleep(100);
7              TimeUnit.MILLISECONDS.sleep(100);
8          }
9      } catch(InterruptedException e) {
10         System.err.println("Interrupted");
11     }
12 }
```

Приоритеты потоков

- 10 уровней приоритетов для Java-потоков
- JVM отображает эти приоритеты в приоритеты планировщика конкретной ОС $\Rightarrow$ зависимость от платформы
- В подавляющем большинстве случаев достаточно приоритета по-умолчанию Thread.NORM_PRIORITY
- Попытки манипулировать приоритетами потоков приводят к зависимости от ОС и могут вызвать "голодание"(starvation)

Задача об обедающих философах



Философ

lecture1.philosophers.Philosopher

```
1  public class Philosopher {
2
3      int position;
4      Fork left;
5      Fork right;
6      int eatCount = 0;
7      long waitTime = 0;
8      long startWait;
9      Random rnd = new Random();
10
11     public Philosopher(int position, Fork left, Fork right) {
12         this.position = position;
13         this.left = left;
14         this.right = right;
15     }
```

Наивное решение

lecture1.philosophers.MyPhilosopher

```
1  public class MyPhilosopher extends Philosopher implements Runnable {
2
3      volatile boolean stopFlag = false;
4
5      public MyPhilosopher(int position, Fork left, Fork right) {
6          super(position, left, right);
7      }
8
9      public void run() {
10         while (!stopFlag) {
11             think();
12             synchronized (left) {
13                 System.out.println("[Philosopher " + position
14                         + "] took left fork");
15                 synchronized (right) {
16                     System.out.println("[Philosopher " + position
17                             + "] took right fork");
18                     eat();
19                 }
20             }
21         }
22         System.out.println("[Philosopher " + position + "] stopped");
23     }
24 }
```

Запуск

lecture1.philosophers.Run

```
1  public static void main(String[] args) throws Exception {
2      MyPhilosopher[] phils = new MyPhilosopher[count];
3
4      Fork last = new Fork();
5      Fork left = last;
6      for (int i=0; i<count; i++) {
7          Fork right = (i==count-1)?last:new Fork();
8          phils[i] = new MyPhilosopher(i, left, right);
9          left = right;
10     }
11
12     Thread[] threads = new Thread[count];
13     for (int i = 0; i < count; i++) {
14         threads[i] = new Thread(phils[i]);
15         threads[i].start();
16     }
17
18     Thread.sleep(60000);
19
20     for (MyPhilosopher phil : phils) {
21         phil.stopFlag = true;
22     }
23     for (Thread thread : threads) {
24         thread.join();
25     }
26     for (MyPhilosopher phil : phils) {
27         System.out.println("[Philosopher " + phil.position + "] ate " +
28                 phil.eatCount + " times and waited " + phil.waitTime + " ms");
29     }
30 }
```

Результат 1

```
1   ................
2   [Philosopher 2] stopped
3   [Philosopher 3] took left fork
4   [Philosopher 1] took right fork
5   [Philosopher 1] is eating
6   [Philosopher 1] finished eating
7   [Philosopher 1] stopped
8   [Philosopher 0] took right fork
9   [Philosopher 0] is eating
10  [Philosopher 0] finished eating
11  [Philosopher 0] stopped
12  [Philosopher 4] took right fork
13  [Philosopher 4] is eating
14  [Philosopher 4] finished eating
15  [Philosopher 4] stopped
16  [Philosopher 3] took right fork
17  [Philosopher 3] is eating
18  [Philosopher 3] finished eating
19  [Philosopher 3] stopped
20  [Philosopher 0] ate 319 times and waited 28621 ms
21  [Philosopher 1] ate 319 times and waited 28350 ms
22  [Philosopher 2] ate 318 times and waited 28171 ms
23  [Philosopher 3] ate 326 times and waited 28253 ms
24  [Philosopher 4] ate 327 times and waited 28235 ms
```

Результат 2

```
1  ................
2  [Philosopher 3] is hungry
3  [Philosopher 3] took left fork
4  [Philosopher 1] took right fork
5  [Philosopher 1] is eating
6  [Philosopher 2] is thinking
7  [Philosopher 1] finished eating
8  [Philosopher 1] is thinking
9  [Philosopher 0] took right fork
10 [Philosopher 0] is eating
11 [Philosopher 2] is hungry
12 [Philosopher 2] took left fork
13 [Philosopher 0] finished eating
14 [Philosopher 0] is thinking
15 [Philosopher 4] took right fork
16 [Philosopher 4] is eating
17 [Philosopher 1] is hungry
18 [Philosopher 1] took left fork
19 [Philosopher 4] finished eating
20 [Philosopher 4] is thinking
21 [Philosopher 0] is hungry
22 [Philosopher 0] took left fork
23 [Philosopher 4] is hungry
24 [Philosopher 4] took left fork
```

Домашнее задание

- Предложите решение задачи об обедающих философах, исключающее
  - взаимную блокировку (deadlock)
  - голодание (starvation, livelock)
  - неравномерное распределение ресурсов (lack of fairness)
- Дополнительно требуется
  - минимизировать время ожидания философом еды
  - обеспечить масштабируемость решения при увеличении кол-ва философов