

Vláknové programování

část VII

Lukáš Hejmánek, Petr Holub

`{xhejtman, hopet}@ics.muni.cz`



Laboratoř pokročilých síťových technologií

PV192

2013-04-16

Přehled přednášky

ThreadPoolExecutors Revisited

Java NIO

Uvážnutí

Optimalizace výkonu

Domácí úloha

Měření

(příklady povětšinou převzaty z JCiP, Goetz)

Typy úloh pro TPE

- Nezávislé úlohy – ideální
- Problémy
 - závislost/komunikace úloh zaslanych do jednoho TPE
 - ◆ ohraničená velikost TPE
 - jednovláknový executor → TPE
 - úlohy citlivé na latenci odpovědi
 - ◆ ohraničená velikost TPE
 - ◆ dlouho běžící úlohy
 - problém s úlohami využívajícími `ThreadLocal`
 - ◆ recyklace vláken
 - nestejně velké úlohy v jednom TPE

Typy úloh pro TPE

Je tohle správně?

```
1  static ExecutorService exec = Executors.newSingleThreadExecutor();
2
3
4  public static class RenderPageTask implements Callable<String> {
5      public String call() throws Exception {
6          Future<String> header, footer;
7          header = exec.submit(new LoadFileTask("header.html"));
8          footer = exec.submit(new LoadFileTask("footer.html"));
9          String page = renderBody();
10         return header.get() + page + footer.get();
11     }
12
13     private String renderBody() {
14         return " body ";
15     }
16 }
```

Typy úloh pro TPE

ANO

```
1  ExecutorService mainExec = Executors.newSingleThreadExecutor();
2  Future<String> task = mainExec.submit(new RenderPageTask());
3
4  try {
5      System.out.println("Vysledek: " + task.get());
6  } catch (InterruptedException e) {
7      e.printStackTrace();
8  } catch (ExecutionException e) {
9      e.printStackTrace();
10 }
11 exec.shutdown();
12 mainExec.shutdown();
```

Typy úloh pro TPE

NE

```
1 Future<String> task = exec.submit(new RenderPageTask());
2 try {
3     System.out.println("Vysledek: " + task.get());
4 } catch (InterruptedException e) {
5     e.printStackTrace();
6 } catch (ExecutionException e) {
7     e.printStackTrace();
8 }
9 exec.shutdown();
```

Typy úloh pro TPE

- Nezávislé úlohy – ideální
- Problémy
 - závislost/komunikace úloh zaslanych do jednoho TPE
 - ◆ ohraničená velikost TPE
 - jednovláknový executor → TPE
 - úlohy citlivé na latenci odpovědi
 - ◆ ohraničená velikost TPE
 - ◆ dlouho běžící úlohy
 - problém s úlohami využívajícími **ThreadLocal**
 - ◆ recyklace vláken
 - nestejně velké úlohy v jednom TPE

Velikost TPE

- Doporučení Javy: $N_{CPU} + 1$ pro výpočetní úlohy
- Obecněji

$$N_{vlaken} = N_{CPU} \cdot U_{CPU} \cdot \left(1 + \frac{W}{C}\right)$$

kde U_{CPU} je cílové využití CPU, W je čas čekání, C je výpočetní čas

- `Runtime.getRuntime().availableProcessors();`

Vytváření a ukončování vláken v TPE

- `corePoolSize` – cílová velikost zásobárny vláken
 - startují se, až jsou potřeba (default policy)
 - `prestartCoreThread()` – nastartuje jedno core vlákno a vrátí `boolean`, zda se povedlo
 - `prestartAllCoreThreads()` – nastartuje všechna core vlákna a vrátí jejich počet
- `maximumPoolSize` – maximální velikost zásobárny vláken
- `keepAliveTime` – doba lelkujícího života
 - od Javy 6: `allowCoreThreadTimeOut` – dovoluje timeout i core vláknům

Správa front v TPE

- Kdy se množí vlákna v TPE?
 - pokud je fronta **plná**
 - co se stane, pokud `corePoolSize = 0` a používáme neomezenou frontu?
- Použití synchronní fronty
 - `SynchronousQueue` není fronta v pravém slova smyslu!
 - synchronní předávání dat mezi úlohami
 - pokud žádné vlákno na předání úlohy nečeká, TPE natvoří nové
 - při dosažení limitu se postupuje podle saturační politiky
 - lze použít při neomezeném počtu vláken (`Executors.newCachedThreadPool`) nebo pokud je akceptovatelné použití saturační politiky
 - efektivní (čas i zdroje) – `Executors.newCachedThreadPool` je efektivnější než `Executors.newCachedThreadPool`, který využívá `LinkedBlockingQueue`
 - implementováno pomocí neblokujícího algoritmu v Java 6, 3× větší výkon než Java 5

Správa front v TPE

- Použití prioritní fronty
 - task musí implementovat `Comparable` (přirozené pořadí) nebo `Comparator`
- Saturační politiky
 - nastupuje v okamžiku zaplnění fronty
 - nastavuje se pomocí `setRejectedExecutionHandler` nebo konstruktoru TPE
 - `AbortPolicy` – default, úloha dostane `RejectedExecutionException`
 - `CallerRunsPolicy` – využití volajícího vlákna
 - ◆ řízení formou zpětné vazby
 - `DiscardPolicy` – vyhodí nově zaslanou úlohu
 - `DiscardOldestPolicy` – vyhodí „nejstarší“ úlohu
 - ◆ vyhazuje z hlavy front \implies nevhodné pro použití s prioritními frontami
 - ◆ pomáhá vytlačit problém do vnějších vrstev: např. pro web server – nemůže zavolat další `accept` – spojení čekají v TCP stacku

Správa front v TPE

```

1  ThreadPoolExecutor tpe =
2      new ThreadPoolExecutor(1, 10, 60, TimeUnit.SECONDS,
3      new LinkedBlockingQueue<Runnable>(100));
4  tpe.setRejectedExecutionHandler
      (new ThreadPoolExecutor.CallerRunsPolicy());

```

- Implementace omezení plnění fronty pomocí semaforu
 - semafor se nastaví na požadovanou velikost fronty + počet běžících úloh

```

1  @ThreadSafe
2  public class BoundedExecutor {
3      private final Executor exec;
4      private final Semaphore semaphore;
5
6      public BoundedExecutor(Executor exec, int bound) {
7          this.exec = exec;
8          this.semaphore = new Semaphore(bound);
9      }

```

Správa front v TPE

```
2 public void submitTask(final Runnable command)
3     throws InterruptedException {
4     semaphore.acquire();
5     try {
6         exec.execute(new Runnable() {
7             public void run() {
8                 try {
9                     command.run();
10                } finally {
11                    semaphore.release();
12                }
13            }
14        });
15    } catch (RejectedExecutionException e) {
16        semaphore.release();
17    }
18 }
```

ThreadFactory

- TPE vytváří vlákna pomocí ThreadFactory
 - metoda `newThread`
 - default ThreadFactory: nedémonická, bez speciálních nastavení
- Možnost předefinovat, jak se budou vytvářet vlákna
 - nastavení pojmenování vláken
 - vlastní třída vytvářených vláken (statistiky, ladění)
 - specifikace vlastního `UncaughtExceptionHandler`
 - nastavení priorit (raději nedělat)
 - nastavení démonického stavu (raději nedělat)
 - v případě použití bezpečnostních politik (security policies) lze použít `privilegedThreadFactory`
 - ◆ podědění oprávnění, `AccessControlContext` a `contextClassLoader` od vlákna vytvářejícího `privilegedThreadFactory`, nikoli od vlákna volajícího `execute/submit` (default)

ThreadFactory

```
2 public class MyThreadFactory implements ThreadFactory {  
3     private final String poolName;  
4     class MyAppThread extends Thread {  
5         public MyAppThread(Runnable runnable, String poolName) {  
6             super(runnable, poolName);  
7         }  
8     }  
9  
10    public MyThreadFactory(String poolName) {  
11        this.poolName = poolName;  
12    }  
13  
14    public Thread newThread(Runnable runnable) {  
15        return new MyAppThread(runnable, poolName);  
16    }  
17 }
```

Modifikace Executorů za běhu

- settery a gettery na různé vlastnosti
- možnost přetypování executorů vyrobených přes factory metody (kromě `newSingleThreadExecutor`) na `ThreadPoolExecutor`
- omezení modifikací
 - nechceme nechat vývojáře šťourat do svých TPE
 - factory metoda `Executor.unconfigurableExecutorService`
 - ◆ bere `ExecutorService`
 - ◆ vrací omezenou `ExecutorService` pomocí `DelegatedExecutorService`, která rozšiřuje `AbstractExecutorService`
 - využíváno metodou `newSingleThreadExecutor` (vrací omezený `Executor` – ačkoli implementace ve skutečnosti používá TPE s jediným vláknem)

Modifikace TPE

- Háčky pro modifikace
 - `beforeExecute`
 - `afterExecute`
 - `terminated`
- Např. sběr statistik

Modifikace TPE

```
public class TimingThreadPool extends ThreadPoolExecutor {
2
    public TimingThreadPool() {
4        super(1, 1, 0L, TimeUnit.SECONDS, null);
    }
6
    private final ThreadLocal<Long> startTime = new ThreadLocal<Long>();
8    private final Logger log = Logger.getLogger("TimingThreadPool");
    private final AtomicLong numTasks = new AtomicLong();
10    private final AtomicLong totalTime = new AtomicLong();
12
    protected void beforeExecute(Thread t, Runnable r) {
        super.beforeExecute(t, r);
14        log.fine(String.format("Thread %s: start %s", t, r));
        startTime.set(System.nanoTime());
16    }
}
```

Modifikace TPE

```
2   protected void afterExecute(Runnable r, Throwable t) {
3       try {
4           long endTime = System.nanoTime();
5           long taskTime = endTime - startTime.get();
6           numTasks.incrementAndGet();
7           totalTime.addAndGet(taskTime);
8           log.fine(String.format("Thread %s: end %s, time=%dns",
9                                   t, r, taskTime));
10          } finally {
11              super.afterExecute(r, t);
12          }
13      }
14
15      protected void terminated() {
16          try {
17              log.info(String.format("Terminated: avg time=%dns",
18                                      totalTime.get() / numTasks.get()));
19          } finally {
20              super.terminated();
21          }
22      }
23  }
```

Kompletně vlastní implementace TPE

- Zdrojové kódy:
 - `http://kickjava.com/src/java/util/concurrent/ThreadPoolExecutor.java`
 - `http://kickjava.com/src/java/util/concurrent/ScheduledThreadPoolExecutor.java`

Java NIO

- Zavedeno v Javě 1.4 (JSR 51)
- Abstraktní třída `Buffer`
 - umožňuje držet pouze primitivní typy

```
ByteBuffer  
CharBuffer  
DoubleBuffer  
FloatBuffer  
IntBuffer  
LongBuffer  
ShortBuffer
```

- direct vs. non-direct buffery
 - přímé buffery se snaží vyhýbat zbytečným kopiím mezi JVM a systémem
- vytváření pomocí metod
 - `allocate` – alokace požadované velikosti
 - `allocateDirect` – alokace požadované velikosti typu direct
 - `wrap` – zabalí existující pole bytů (`bytearray`)

Java NIO

- ByteBuffer

- **http:**

- `//download.oracle.com/javase/6/docs/api/java/nio/ByteBuffer.html`

- přístup k binárním datům, např.

```
float  getFloat()
float  getFloat(int index)
void   putFloat(float f)
void   putFloat(int index, float f)
```

- mapování souborů do paměti (`FileChannel`, metoda `map`)
 - čtení/vložení z/do bufferu bez parametru `index` (`get/put`) inkrementuje pozici
 - pokud není řečeno jinak, metody vrací odkaz na buffer – řetězení volání

```
buffer.putShort(10).putInt(0x00ABBCCD).putShort(11);
```

Java NIO

- Vlastnosti bufferů

capacity celková kapacita bufferu

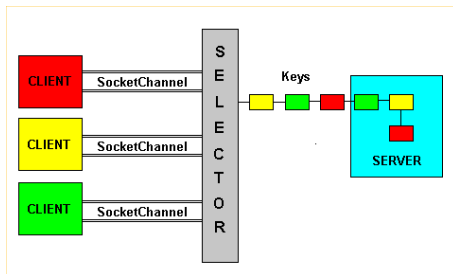
limit umělý limit uvnitř bufferu, využití s metodami `flip` (nastaví limit na současnou pozici a skočí na pozici 0) či `remaining`

mark pomocná značka, využití např. s metodou `reset` (skočí na označovanou pozici)

```
1 buffer.position(10);  
2 buffer.flip();  
3 while (buffer.hasRemaining()) {  
4     byte b = buffer.get();  
5     // něco  
6 }
```

Java NIO

- Selektor
 - serializace požadavků
 - výběr požadavků
- Klíč
 - identifikace konkrétního spojení



Zdroj: <http://onjava.com/lpt/a/2672>

Java NIO – Server

- Generický postup

```
create SocketChannel;
2 create Selector
associate the SocketChannel to the Selector
4 for(;;) {
    waiting events from the Selector;
6 event arrived; create keys;
    for each key created by Selector {
8     check the type of request;
        isAcceptable:
10     get the client SocketChannel;
        associate that SocketChannel to the Selector;
12     record it for read/write operations
        continue;
14     isReadable:
        get the client SocketChannel;
16     read from the socket;
        continue;
18     isWritable:
        get the client SocketChannel;
20     write on the socket;
        continue;
22 }
}
```

Zdroj: <http://onjava.com/lpt/a/2672>

Java NIO – Server

```
1 // Create the server socket channel
  ServerSocketChannel server = ServerSocketChannel.open();
3 // nonblocking I/O
  server.configureBlocking(false);
5 // host-port 8000
  server.socket().bind(new java.net.InetSocketAddress(host, 8000));
7 // Create the selector
  Selector selector = Selector.open();
9 // Recording server to selector (type OP_ACCEPT)
  server.register(selector, SelectionKey.OP_ACCEPT);
```

Zdroj: <http://onjava.com/lpt/a/2672>

Java NIO – Server

```

// Infinite server loop
2  for(;;) {
    // Waiting for events
4   selector.select();
    // Get keys
6   Set keys = selector.selectedKeys();
    Iterator i = keys.iterator();
8
    // For each keys...
10  while(i.hasNext()) {
        SelectionKey key = (SelectionKey) i.next();
12
        // Remove the current key
14  i.remove();

        // if isAcetatable = true
        // then a client required a connection
18  if (key.isAcceptable()) {
            // get client socket channel
20  SocketChannel client = server.accept();
            // Non Blocking I/O
22  client.configureBlocking(false);
            // recording to the selector (reading)
24  client.register(selector, SelectionKey.OP_READ);
            continue;
26  }

```

Java NIO – Server

```

2 // if isReadable = true
3 // then the server is ready to read
4 if (key.isReadable()) {
5
6     SocketChannel client = (SocketChannel) key.channel();
7
8     // Read byte coming from the client
9     int BUFFER_SIZE = 32;
10    ByteBuffer buffer = ByteBuffer.allocate(BUFFER_SIZE);
11    try {
12        client.read(buffer);
13    }
14    catch (Exception e) {
15        // client is no longer active
16        e.printStackTrace();
17        continue;
18    }
19
20    // Show bytes on the console
21    buffer.flip();
22    Charset charset=Charset.forName('' ISO-8859-1'' );
23    CharsetDecoder decoder = charset.newDecoder();
24    CharBuffer charBuffer = decoder.decode(buffer);
25    System.out.print(charBuffer.toString());
26    continue;
27 }
28 }

```

Java NIO

- Další čtení:

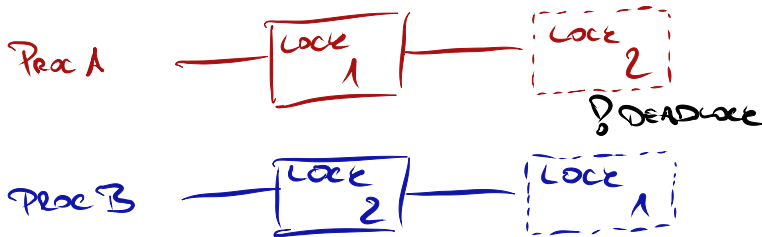
- `http://onjava.com/lpt/a/2672`
- `http://onjava.com/lpt/a/5127`
- `http://download.oracle.com/javase/6/docs/api/java/nio/channels/Selector.html`
- `http://download.oracle.com/javase/6/docs/api/java/nio/channels/SelectionKey.html`

Asynchronní programování versus vlákna

- Asynchronní programování
 - + umožňuje obsluhovat řádově větší množství klientů
 - za cenu zvýšení latence
 - složitější, náchylnější na chyby
- Vlákňové programování
 - + jednodušší
 - + poměrně efektivní do „rozumného“ počtu vláken
 - nativní vlákna nejsou stavěna na (deseti)tisíce vláken a více
- Potenciálně lze kombinovat

Deadlock

- Deadlock – uváznutí, smrtelné objetí ;-)
- Vzájemné nekončící čekání na zámky



- Potřeba globálního uspořádání zámek
 - zamykání podle globálního uspořádání
- Možnost využití `Lock.tryLock()`
 - náhodný rovnoměrný back-off
 - náhodný exponenciální back-off
 - nelze použít s `monitor`
- Řešení deadlocků runtime (ne v Javě)

Deadlock

```
2 public static void transferMoney(Account fromAccount,
3                                 Account toAccount,
4                                 DollarAmount amount)
5     throws InsufficientFundsException {
6     synchronized (fromAccount) {
7         synchronized (toAccount) {
8             if (fromAccount.getBalance().compareTo(amount) < 0)
9                 throw new InsufficientFundsException();
10            else {
11                fromAccount.debit(amount);
12                toAccount.credit(amount);
13            }
14        }
15    }
16 }
```


Deadlock

```
2 public static void transferMoney(Account fromAccount,
3                                 Account toAccount,
4                                 DollarAmount amount)
5     throws InsufficientFundsException {
6     synchronized (fromAccount) {
7         synchronized (toAccount) {
8             if (fromAccount.getBalance().compareTo(amount) < 0)
9                 throw new InsufficientFundsException();
10            else {
11                fromAccount.debit(amount);
12                toAccount.credit(amount);
13            }
14        }
15    }
```



Deadlock

```
2 public void transferMoney(final Account fromAcct,
3                             final Account toAcct,
4                             final DollarAmount amount)
5     throws InsufficientFundsException {
6     class Helper {
7     public void transfer() throws InsufficientFundsException {
8         if (fromAcct.getBalance().compareTo(amount) < 0)
9             throw new InsufficientFundsException();
10        else {
11            fromAcct.debit(amount);
12            toAcct.credit(amount);
13        }
14    }
15 }
16 int fromHash = System.identityHashCode(fromAcct);
17 int toHash = System.identityHashCode(toAcct);
```

- `System.identityHashCode(o)` může vrátit pro dva různé objekty identický hash
 - řídký problém

Deadlock

```
2   if (fromHash < toHash) {
3       synchronized (fromAcct) {
4           synchronized (toAcct) {
5               new Helper().transfer();
6           }
7       }
8   } else if (fromHash > toHash) {
9       synchronized (toAcct) {
10          synchronized (fromAcct) {
11              new Helper().transfer();
12          }
13      }
14  } else {
15      synchronized (tieLock) {
16          synchronized (fromAcct) {
17              synchronized (toAcct) {
18                  new Helper().transfer();
19              }
20          }
21      }
22  }
```

Deadlock

```
1 private static Random rnd = new Random();  
2  
3 public boolean transferMoney(Account fromAcct,  
4                             Account toAcct,  
5                             DollarAmount amount,  
6                             long timeout,  
7                             TimeUnit unit)  
8     throws InsufficientFundsException, InterruptedException {  
9     long fixedDelay = getFixedDelayComponentNanos(timeout, unit);  
10    long randMod = getRandomDelayModulusNanos(timeout, unit);  
11    long stopTime = System.nanoTime() + unit.toNanos(timeout);
```

Deadlock

```
2   while (true) {
3       if (fromAcct.lock.tryLock()) {
4           try {
5               if (toAcct.lock.tryLock()) {
6                   try {
7                       if (fromAcct.getBalance().compareTo(amount) < 0)
8                           throw new InsufficientFundsException();
9                       else {
10                          fromAcct.debit(amount);
11                          toAcct.credit(amount);
12                          return true;
13                      }
14                  } finally {
15                      toAcct.lock.unlock();
16                  }
17              }
18          } finally {
19              fromAcct.lock.unlock();
20          }
21      }
22      if (System.nanoTime() < stopTime)
23          return false;
24      NANOSECONDS.sleep(fixedDelay + rnd.nextLong() % randMod);
25  }
```

Otevřená volání

```
1  class Taxi {
2      @GuardedBy("this") private Point location, destination;
3      private final Dispatcher dispatcher;
4
5      public Taxi(Dispatcher dispatcher) {
6          this.dispatcher = dispatcher;
7      }
8
9      public synchronized Point getLocation() {
10         return location;
11     }
12
13     public synchronized void setLocation(Point location) {
14         this.location = location;
15         if (location.equals(destination))
16             dispatcher.notifyAvailable(this);
17     }
18
19     public synchronized Point getDestination() {
20         return destination;
21     }
22
23     public synchronized void setDestination(Point destination) {
24         this.destination = destination;
25     }
26 }
```

Otevřená volání

```
2  class Dispatcher {
3      @GuardedBy("this") private final Set<Taxi> taxis;
4      @GuardedBy("this") private final Set<Taxi> availableTaxis;
5
6      public Dispatcher() {
7          taxis = new HashSet<Taxi>();
8          availableTaxis = new HashSet<Taxi>();
9      }
10
11     public synchronized void notifyAvailable(Taxi taxi) {
12         availableTaxis.add(taxi);
13     }
14
15     public synchronized Image getImage() {
16         Image image = new Image();
17         for (Taxi t : taxis)
18             image.drawMarker(t.getLocation());
19         return image;
20     }
21 }
```

Otevřená volání

```

1  class Taxi {
2      @GuardedBy("this") private Point location, destination;
3      private final Dispatcher dispatcher;
4
5      public Taxi(Dispatcher dispatcher) {
6          this.dispatcher = dispatcher;
7      }
8
9      public synchronized Point getLocation() {
10         return location;
11     }
12
13     public synchronized void setLocation(Point location) {
14         this.location = location;
15         if (location.equals(destination))
16             dispatcher.notifyAvailable(this);
17     }
18
19     public synchronized Point getDestination() {
20         return destination;
21     }
22
23     public synchronized void setDestination(Point destination) {
24         this.destination = destination;
25     }
26 }

```

```

1  class Dispatcher {
2      @GuardedBy("this") private final Set<Taxi> taxis;
3      @GuardedBy("this") private final Set<Taxi> availableTaxis;
4
5      public Dispatcher() {
6          taxis = new HashSet<Taxi>();
7          availableTaxis = new HashSet<Taxi>();
8      }
9
10     public synchronized void notifyAvailable(Taxi taxi) {
11         availableTaxis.add(taxi);
12     }
13
14     public synchronized Image getImage() {
15         Image image = new Image();
16         for (Taxi t : taxis)
17             image.drawMarker(t.getLocation());
18         return image;
19     }
20 }

```



- setLocation → notifyAvailable
- getImage → getLocation

Otevřená volání

- Otevřené volání (open call)
 - volání metody, kdy volající nadržuje žádný zámek
 - preferovaný způsob
- Převod na otevřené volání
 - synchronizace by měla být omezena na lokální proměnné
 - problém se zachováním sémantiky
- Možnost globálního zámku

Otevřená volání

```

1  class Taxi {
2      @GuardedBy("this") private Point location, destination;
3      private final Dispatcher dispatcher;
4
5      public Taxi(Dispatcher dispatcher) { this.dispatcher = dispatcher; }
6
7      public synchronized Point getLocation() { return location; }
8
9      public void setLocation(Point location) {
10         boolean reachedDestination;
11         synchronized (this) {
12             this.location = location;
13             reachedDestination = location.equals(destination);
14         }
15         if (reachedDestination)
16             dispatcher.notifyAvailable(this);
17     }
18
19     public synchronized Point getDestination() { return destination; }
20
21     public synchronized void setDestination(Point destination) {
22         this.destination = destination;
23     }
24 }

```

Otevřená volání

```

1  class Dispatcher {
2      @GuardedBy("this") private final Set<Taxi> taxis;
3      @GuardedBy("this") private final Set<Taxi> availableTaxis;
4
5      public Dispatcher() {
6          taxis = new HashSet<Taxi>();
7          availableTaxis = new HashSet<Taxi>();
8      }
9
10     public synchronized void notifyAvailable(Taxi taxi) {
11         availableTaxis.add(taxi);
12     }
13
14     public Image getImage() {
15         Set<Taxi> copy;
16         synchronized (this) {
17             copy = new HashSet<Taxi>(taxis);
18         }
19         Image image = new Image();
20         for (Taxi t : copy)
21             image.drawMarker(t.getLocation());
22         return image;
23     }
24 }

```

Hladovění

- Hladovění (starvation) nastává, pokud je vláknu neustále odpírán zdroj, který je potřeba k dalšímu postupu
 - běžné použití zámků je férové
 - problém při nastavování priorit

```
2 t.setPriority(Thread.MIN_PRIORITY); // 1
  t.setPriority(Thread.NORM_PRIORITY); // 5
  t.setPriority(Thread.MAX_PRIORITY); // 10
```

- ◆ problém platformové závislosti priorit
 - ◆ možná pomoc pro zvýšení responsivity GUI
- typické pokusy o „řešení“ problémů

```
1 Thread.yield();
  Thread.sleep(100);
```

Další typy uváznutí

- Livelock
 - uváznutí, při němž se vlákno (aktivně) snaží o činnosti, která opakovaně selhává
 - náhodnostní exponenciální back-off
- Ztracené zprávy
 - `o.wait()` a `o.notify()` resp. `o.notifyAll` nemají mechanismus zdržení notifikace
 - pokud vlákno usne na `o.wait()` později, než mělo být notifikováno přes `o.notify`, nikdy se nevzbudí

Hledání problémů

- Výpis stavu JVM
 - `SIGQUIT` na unixech (ev. `Ctrl-\` pokud mapuje na `SIGQUIT`)
 - `Ctrl-Break` na Windows

Hledání problémů

```
2 public static void main(String[] args) {  
3     final Object a = new Object();  
4     final Object b = new Object();  
  
5     Thread t1 = new Thread(new Runnable() {  
6         public void run() {  
7             try {  
8                 synchronized (a) {  
9                     Thread.sleep(1000);  
10                    System.out.println("t1 - cekam na b");  
11                    synchronized (b) {  
12                        System.out.println("t1 - jsem zde");  
13                    }  
14                }  
15            } catch (InterruptedException e) {  
16            }  
17        }  
18    });
```

Hledání problémů

```
2 Thread t2 = new Thread(new Runnable() {
3     public void run() {
4         try {
5             synchronized (b) {
6                 Thread.sleep(1000);
7                 System.out.println("t2 - cekam na a");
8                 synchronized (a) {
9                     System.out.println("t2 - jsem zde");
10                }
11            }
12        } catch (InterruptedException e) {
13        }
14    }
15 });
16
17 t1.start();
18 t2.start();
```


Hledání problémů

```

$ java IntentionalDeadlock
2 t2 - cekam na a
  t1 - cekam na b
4 2010-04-22 11:46:25
  Full thread dump Java HotSpot(TM) Client VM (16.2-b04 mixed mode, sharing):
6
  "DestroyJavaVM" prio=6 tid=0x020b1000 nid=0x164c waiting on condition [0x00000000]
8     java.lang.Thread.State: RUNNABLE

10 "Thread-1" prio=6 tid=0x02149800 nid=0x1b4c waiting for monitor entry [0x0480f000]
    java.lang.Thread.State: BLOCKED (on object monitor)
12        at IntentionalDeadlock$2.run(IntentionalDeadlock.java:35)
    - waiting to lock <0x243e6928> (a java.lang.Object)
14        - locked <0x243e6930> (a java.lang.Object)
    at java.lang.Thread.run(Unknown Source)
16

18 "Thread-0" prio=6 tid=0x02146c00 nid=0x1a38 waiting for monitor entry [0x0477f000]
    java.lang.Thread.State: BLOCKED (on object monitor)
    at IntentionalDeadlock$1.run(IntentionalDeadlock.java:20)
20        - waiting to lock <0x243e6930> (a java.lang.Object)
    - locked <0x243e6928> (a java.lang.Object)
22        at java.lang.Thread.run(Unknown Source)

```

Hledání problémů

```

1 "Low Memory Detector" daemon prio=6 tid=0x02121400 nid=0xbd8 runnable [0x00000000]
   java.lang.Thread.State: RUNNABLE
3
5 "CompilerThread0" daemon prio=10 tid=0x02119800 nid=0x1708 waiting on condition [0x00000000]
   java.lang.Thread.State: RUNNABLE
7
9 "Attach Listener" daemon prio=10 tid=0x02118400 nid=0x13d0 runnable [0x00000000]
   java.lang.Thread.State: RUNNABLE
11 "Signal Dispatcher" daemon prio=10 tid=0x02115400 nid=0x5a0 waiting on condition [0x00000000]
   java.lang.Thread.State: RUNNABLE

```

```

Heap
2 def new generation total 4928K, used 466K [0x243b0000, 0x24900000, 0x29900000)
   eden space 4416K, 10% used [0x243b0000, 0x24424828, 0x24800000)
   from space 512K, 0% used [0x24800000, 0x24800000, 0x24880000)
   to space 512K, 0% used [0x24880000, 0x24880000, 0x24900000)
4
6 tenured generation total 10944K, used 0K [0x29900000, 0x2a3b0000, 0x343b0000)
   the space 10944K, 0% used [0x29900000, 0x29900000, 0x29900200, 0x2a3b0000)
8 compacting perm gen total 12288K, used 42K [0x343b0000, 0x34fb0000, 0x383b0000)
   the space 12288K, 0% used [0x343b0000, 0x343ba960, 0x343baa00, 0x34fb0000)
10    ro space 10240K, 51% used [0x383b0000, 0x388dae00, 0x388dae00, 0x38db0000)
   rw space 12288K, 54% used [0x38db0000, 0x394472d8, 0x39447400, 0x399b0000)

```

Hledání problémů

```

2 Found one Java-level deadlock:
3 =====
4 "Thread-1":
5   waiting to lock monitor 0x020d53ac (object 0x243e6928, a java.lang.Object),
6   which is held by "Thread-0"
7 "Thread-0":
8   waiting to lock monitor 0x020d6c74 (object 0x243e6930, a java.lang.Object),
9   which is held by "Thread-1"
10
11 Java stack information for the threads listed above:
12 =====
13 "Thread-1":
14   at IntentionalDeadlock$2.run(IntentionalDeadlock.java:35)
15   - waiting to lock <0x243e6928> (a java.lang.Object)
16   - locked <0x243e6930> (a java.lang.Object)
17   at java.lang.Thread.run(Unknown Source)
18 "Thread-0":
19   at IntentionalDeadlock$1.run(IntentionalDeadlock.java:20)
20   - waiting to lock <0x243e6930> (a java.lang.Object)
21   - locked <0x243e6928> (a java.lang.Object)
22   at java.lang.Thread.run(Unknown Source)
23
24 Found 1 deadlock.

```

Statická analýza kódu

- FindBugs

<http://findbugs.sourceforge.net/>

The screenshot shows the FindBugs application window titled "FindBugs: Příklad". The interface includes a menu bar (File, Edit, View, Navigation, Designation, Help), a "Class search strings:" field, and a "Category" dropdown menu. The "Category" menu is expanded, showing a tree structure of bug categories: "Multithreaded correctness (2)", "Sleep with lock held (2)", "Method calls Thread.sleep() with a lock held (2)", "Performance (1)", "Inner class could be made static (1)", and "Should be a static inner class (1)". The "Method calls Thread.sleep() with a lock held (2)" category is selected, and a specific bug is highlighted: "IntentionalDeadlock\$1.run() calls Thread.sleep() with a lock held".

The main editor displays the source code of the "IntentionalDeadlock.java" file. The code is as follows:

```
7 /*
8 public class IntentionalDeadlock {
9     public static void main(String[] args) {
10         final Object a = new Object();
11         final Object b = new Object();
12
13         Thread t1 = new Thread(new Runnable() {
14             public void run() {
15                 try {
16                     synchronized (a) {
17                         Thread.sleep(1000);
18                         System.out.println("t1 - oceká na b");
19                         synchronized (b) {
20                             System.out.println("t1 - jasná záda");
21                         }
22                     }
23                 } catch (InterruptedException e) {
24                 }
25             }
26         });
```

The bug description at the bottom of the window reads: "IntentionalDeadlock\$1.run() calls Thread.sleep() with a lock held. At IntentionalDeadlock.java [line 17]. In method IntentionalDeadlock\$1.run() [Lines 16 - 25]. Method calls Thread.sleep() with a lock held. This method calls Thread.sleep() with a lock held. This may result in very poor performance and scalability, or a deadlock, since other threads may be waiting to acquire the lock. It is a much better idea to call wait() on the lock, which releases the lock and allows other threads to run."

At the bottom left of the window, the URL <http://findbugs.sourceforge.net> is displayed. At the bottom right, the logo of the University of Maryland is visible.

Anotace

- Vícečlenný tým programátorů – předávání myšlenek
 - komentáře v kódy
 - anotace
 - ◆ anotace se dají použít i pro statickou analýzu kódu

```
2 import net.jcip.annotations.GuardedBy;  
   // http://www.javaconcurrencyinpractice.com/jcip-annotations.jar
```

Anotace

- Anotace tříd

- `@Immutable`
- `@ThreadSafe`
- `@NotThreadSafe`

- Anotace polí

- `@GuardedBy("this")`
 - ◆ monitor (intrinsic lock) na `this`
- `@GuardedBy("jmenoPole")`
 - ◆ explicitní zámek na `jmenoPole` pokud je potomkem `Lock`
 - ◆ jinak monitor na `jmenoPole`
- `@GuardedBy("JmenoTridy.jmenoPole")`
 - ◆ obdobné, odkazuje se statické pole jiné třídy
- `@GuardedBy("jmenoMetody()")`
 - ◆ metoda `jmenoMetody()` vrací zámek
- `@GuardedBy("JmenoTridy.class")`
 - ◆ literál třídy (objekt) pro pojmenovanou třídu

Omezování zámků

$$\text{zrychlení} \leq \frac{1}{s + \frac{1-s}{n}}$$

- JVM se snaží dělat
 - eliminaci synchronizací, které nemohou nastat (např. pomocí escape analysis – lokální objekt, který není nikdy publikován na haldu a je tudíž thread-local)
 - kombinace více zámků do jednoho (lock coarsening)
- Zbytečně nesynchronizovat
 - delegace bezpečnosti (thread safety delegation)
 - omezení rozsahu synchronizace (get in – get out principle, např. Taxi/Dispatcher)
 - dělení zámků (lock splitting) – pouze pro **nezávislé** proměnné/objekty
 - ořezávání zámků (lock stripping)
 - RW zámků
- Neprovádět object pooling na jednoduchých objektech
 - `new` je levnější jako `malloc`

```

synchronized (new Object()) {
    System.out.println("bleeee");
}

```

Omezování zámků

$$\text{zrychlení} \leq \frac{1}{s + \frac{1-s}{n}}$$

- JVM se snaží dělat
 - eliminaci synchronizací, které nemohou nastat (např. pomocí escape analysis – lokální objekt, který není nikdy publikován na haldu a je tudíž thread-local)
 - kombinace více zámků do jednoho (lock coarsening)
- Zbytečně nesynchronizovat
 - delegace bezpečnosti (thread safety delegation)
 - omezení rozsahu synchronizace (get in – get out principle, např. Taxi/Dispatcher)
 - dělení zámků (lock splitting) – pouze pro **nezávislé** proměnné/objekty
 - ořezávání zámků (lock stripping)
 - RW zámků
- Neprovádět object pooling na jednoduchých objektech
 - `new` je levnější jako `malloc`



```
2 synchronized (new Object()) {  
    System.out.println("bleeee");  
}
```


Omezování zámků

- Podobně jako dělení zámků, ale pro proměnný počet nezávislých proměnných/objektů
 - Příklad ořezávání zámků – `ConcurrentHashMap`
 - 16 zámků
 - každý z N hash buckets je chráněný zámkem $N \bmod 16$
 - předpokládáme rovnoměrné rozdělení položek mezi kbelíky
- ⇒ 16 paralelních přístupů
- ⇒ přístup k celé kolekci vyžaduje všech 16 zámků
- rozdělení kumulativních polí do jednotlivých kbelíků

Domácí úloha

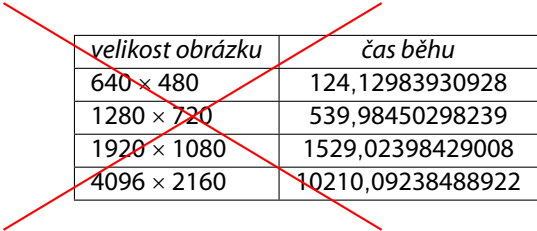
1. Naimplementuje webový harvester, který rekurzivně stahuje a ukládá webové stránky do stromu a hledá v nich zadaný text.
 - Soubory není třeba ukládat.
 - Všechny řádky staženého souboru spojte do jednoho, najděte všechny výskyty `` a uložte si relativní i absolutní URL do fronty k dalšímu stažení.
 - Pokud najdete zadaný text, vytiskněte ho do GUI okna spolu s URL, kde byl nalezen.
 - Program bude mít GUI zobrazující hlavičky aktuálně stahovaných dokumentů.
2. Pro implementaci stahování vytvořte vlastní thread pool (rozšířením třídy `ThreadPool`), který se bude dynamicky zvětšovat/zmenšovat podle počtu čekajících požadavků ve frontě. Tj. začne růst nad `corePoolSize` and již před zaplněním fronty – po dosažení např. 75% zaplnění fronty.

Interakce s JVM při měření

- Problém garbage collection
 - `-verbose:gc`
 - krátká měření: vybrat pouze běhy, v nichž nedošlo ke GC
 - dlouhé běhy: dostatečně dlouhé, aby se přítomnost GC projevila reprezentativně
- Problém HotSpot kompilace
 - `-XX:+PrintCompilation`
 - dostatečný warm-up (minuty!)
 - mohou se vyskytovat rekompilace (optimalizace, nahrání nové třídy která zruší dosavadní předpoklady)
 - housekeeping tasks: oddělení nesouvisejících měření pauzou nebo restartem JVM

Délka zpracování obrázku

<i>velikost obrázku</i>	<i>čas běhu</i>
640 × 480	124,12983930928
1280 × 720	539,98450298239
1920 × 1080	1529,02398429008
4096 × 2160	10210,09238488922



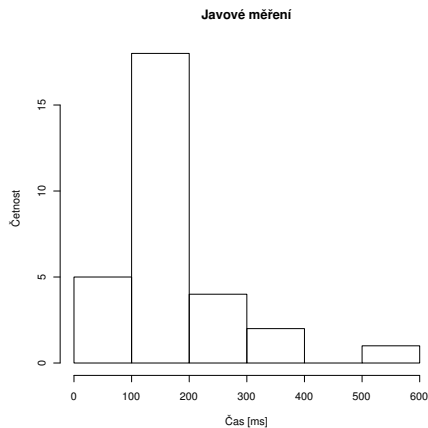
<i>velikost obrázku</i>	<i>čas běhu</i>
640 × 480	124,12983930928
1280 × 720	539,98450298239
1920 × 1080	1529,02398429008
4096 × 2160	10210,09238488922

Měříme délku výpočtu v Javě

```
> library(psych)
> runlength <- read.csv(file="java-example.table", head=FALSE, sep=",")
> summary(runlength$V1)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 92.08 104.70 108.80 166.80 187.20 594.70
> describe(runlength$V1)
  var  n  mean    sd median trimmed  mad  min    max  range skew kurtosis
1   1 30 166.82 113.67 108.78  142.1 20.88 92.08 594.71 502.63 2.14    4.55
  se
1 20.75
```


$$\begin{aligned}N &= 30 \\ \bar{x} &= 166,82 \\ s_x &= 113,67 \\ s_{\bar{x}} &= \frac{s_x}{\sqrt{N}} = 20,75 \\ t_{0,05;29} &= 2,045\end{aligned}$$

$$\bar{x} \pm t_{0,05;N-1} s_{\bar{x}} = 167 \pm 42 \text{ ms}$$



$$N = 30$$

$$\bar{x} = 166,82$$

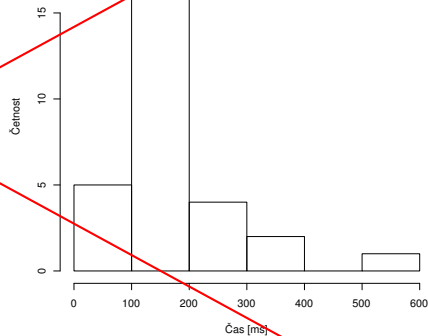
$$sx = 113,67$$

$$s_{\bar{x}} = \frac{sx}{\sqrt{N}} = 20,75$$

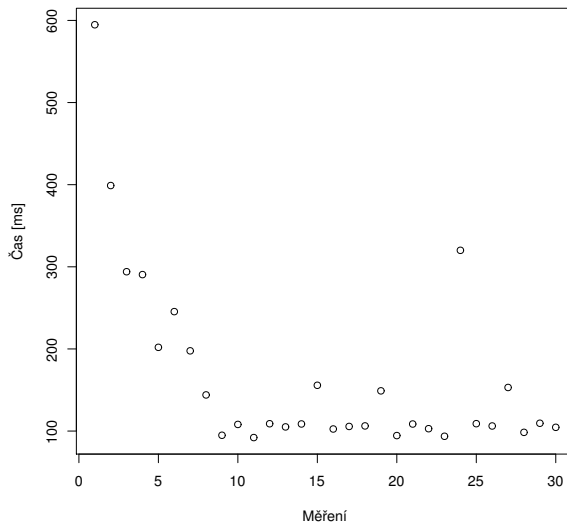
$$t_{0,05;29} = 2,045$$

$$\bar{x} \pm t_{0,05;N-1} s_{\bar{x}} = 167 \pm 42 \text{ ms}$$

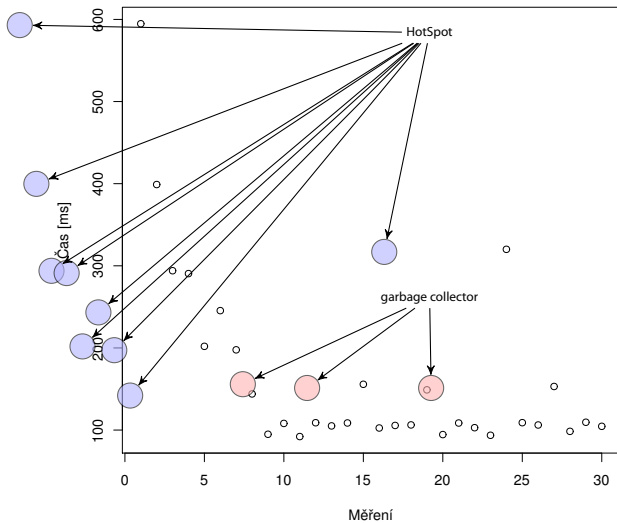
Javové měření



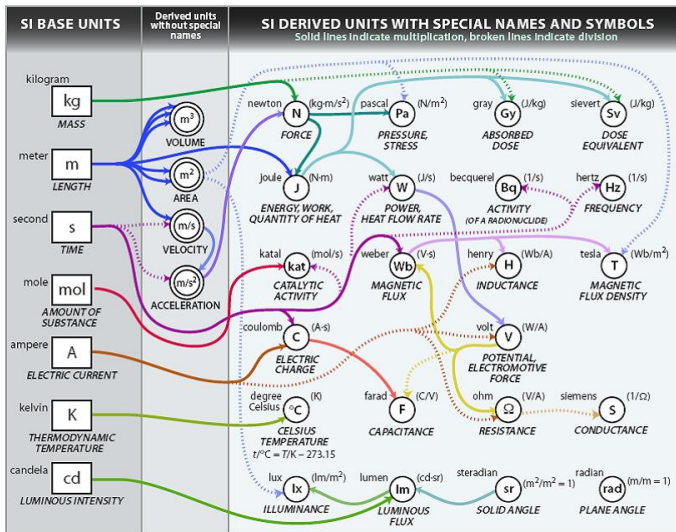
Javové měření



Javové měření



Soustava jednotek pro informatiky



Zdroj: [http://www.icrf.nl/Portals/106/SI_units_diagram\(1\).jpg](http://www.icrf.nl/Portals/106/SI_units_diagram(1).jpg)

Soustava jednotek pro informatiky

- Předpony nejen speciálně inforatické

yocto-	10^{-24}	y	-	-	-
zepto-	10^{-21}	z	-	-	-
atto-	10^{-18}	a	-	-	-
femto-	10^{-15}	f	-	-	-
pico-	10^{-12}	p	-	-	-
nano-	10^{-9}	n	-	-	-
micro-	10^{-6}	μ	-	-	-
milli-	10^{-3}	m	-	-	-
kilo-	10^3	k	kibi	2^{10}	Ki
mega-	10^6	M	mebi	2^{20}	Mi
giga-	10^9	G	gibi	2^{30}	Gi
tera-	10^{12}	T	tebi	2^{40}	Ti
peta-	10^{15}	P	pebi	2^{50}	Pi
exa-	10^{18}	E	exbi	2^{60}	Ei
zetta-	10^{21}	Z	zebi	2^{70}	Zi
yotta-	10^{24}	Y	yobi	2^{80}	Yi

Amendment 2 to "IEC 60027-2: Letter symbols to be used in electrical technology – Part 2: Telecommunications and electronics" (1999)

Výsledky měření

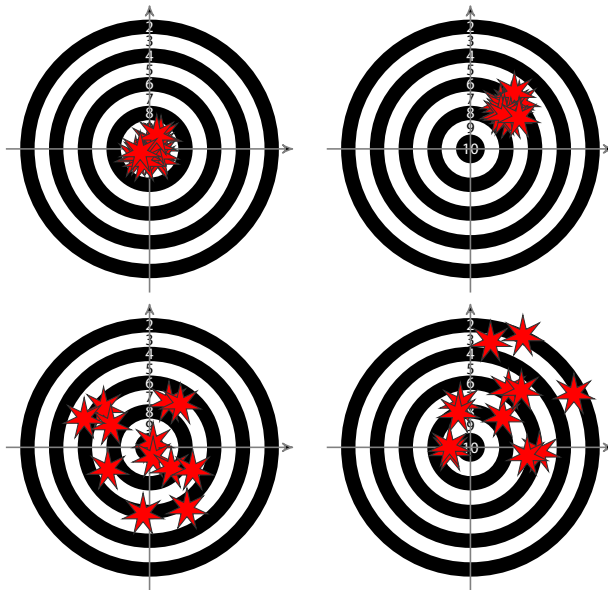
$$x = (\hat{\mu}_x \pm z_x) \text{ [jednotka]}$$

- $\hat{\mu}_x$... nejpravděpodobnější hodnota měřené veličiny
- z_x ... interval spolehlivosti / přesnost
- jak tyto věci spočítat / odhadnout?

Chyby měření

- Klasifikace chyb podle místa vzniku
 - instrumentální (přístrojové) chyby
 - metodické chyby
 - teoretické chyby (principy, model)
 - chyby zpracování
- Klasifikace chyb podle původu
 - hrubé (omyly)
 - systematické
 - náhodné

Chyby měření

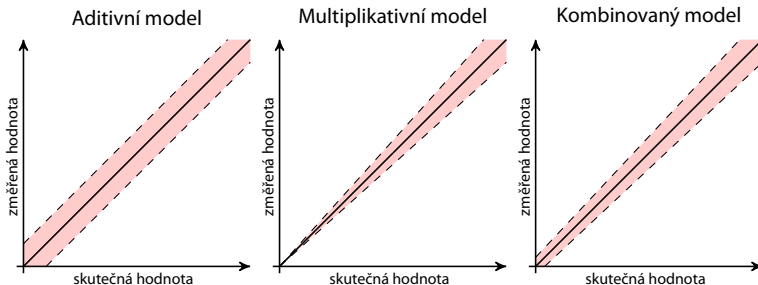


Přesnost měřících nástrojů

Přesnost přístroje ... náhodná chyba

Správnost přístroje ... systematická chyba

- Aditivní vs. multiplikativní chyby
- Mezní hodnota chyb
- Třída přesnosti přístroje



Náhodné chyby

aneb proč se běžně pracuje s normálním rozdělením chyb?

- Hypotéza elementárních chyb (Horák, 1958)
 - každá náhodná chyba v měření je složena z řady malých chyb
 - při velkém počtu měření se vyskytne zhruba stejný počet chyb kladných i záporných a malé chyby jsou početnější než velké
- 1. m elementárních náhodných vlivů
- 2. každý elementární vliv generuje chybu α (dále označováno jako případ a) nebo $-\alpha$ (dále případ b)
- 3. chyby a a b jsou stejně časté
- dostáváme binomické rozdělení kumulace vlivů elementárních chyb

$$\binom{m}{0}a^m, \binom{m}{1}a^{m-1}b, \dots, \binom{m}{l}a^{m-l}b^l, \dots, \binom{m}{m}b^m$$

$$P(0) = \frac{1}{2^m} \binom{m}{m/2} \quad P(\varepsilon_l) = \frac{1}{2^m} \binom{m}{l}, \quad \varepsilon_l = (l - (m - l))\alpha = (2l - m)\alpha = 2s\alpha$$

Náhodné chyby

aneb proč se běžně pracuje s normálním rozdělením chyb?

- Co se stane, pokud $m \rightarrow \infty$?
 - pro sudá $m = 2k \implies k \rightarrow \infty$ (sudá, abychom měli $P(0)$)

$$P(\varepsilon) = P(2s\alpha) = \frac{1}{2^{2k}} \binom{2k}{k+s}$$

$$\frac{P(2s\alpha)}{P(0)} = \frac{\binom{2k}{k+s}}{\binom{2k}{k}} = \frac{k(k-1)\cdots(k-s+1)}{(k+1)(k+2)\cdots(k+s)} = \frac{\left(1 - \frac{1}{k}\right)\left(1 - \frac{2}{k}\right)\cdots\left(1 - \frac{s-1}{k}\right)}{\left(1 + \frac{1}{k}\right)\left(1 + \frac{2}{k}\right)\cdots\left(1 + \frac{s}{k}\right)}$$

- pro $s \ll k$

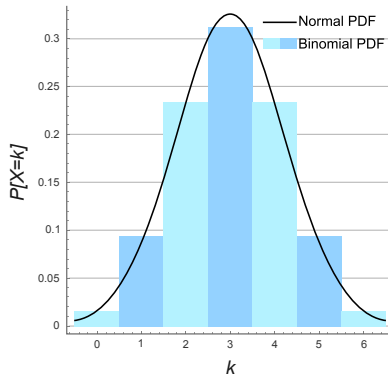
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots \approx x$$

$$\ln \frac{P(2s\alpha)}{P(0)} = -\frac{1}{k} - \frac{2}{k} - \cdots - \frac{s-1}{k} - \frac{1}{k} - \frac{2}{k} - \cdots - \frac{s}{k} = -\frac{2s(s-1)}{k} - \frac{s}{k} = -\frac{s^2}{k}$$

$$P(2s\alpha) = P(0)e^{-\frac{s^2}{k}} = P(0)e^{-\frac{\varepsilon^2}{4k\alpha^2}}$$

Binomické vs. normální rozdělení

Srovnání binomického a normálního rozdělení pro $p = 0,5$ a $n = 6$



Zdroj: http://en.wikipedia.org/wiki/File:Binomial_Distribution.svg

Studentovo rozdělení t

- Používá se pro normální rozdělení při malém vzorku

$$f(t) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{t^2}{\nu}\right)^{-(\nu+1)/2}$$

kde ν je počet stupňů volnosti.

- odhad průměrů a chyby
- t-test – odlišení průměrů

Odhad spolehlivosti

$$x = (\hat{\mu}_x \pm z_x) \text{ [jednotka]}$$

Statistická definice (Šťastný, 1997): Je-li výsledek měření $\hat{\mu}_x$ a z_x je chyba tohoto měření odpovídající míře jistoty p , pak skutečná hodnota měřené veličiny leží v intervalu $(\hat{\mu}_x \pm z_x)$ s pravděpodobností p .

- Intervaly
 - 0,68 – střední kvadratická chyba
 - 0,95
 - 0,99 – krajní chyba
- Zaokrouhlování
 - z_x nejvýše na 2 platná místa
 - $\hat{\mu}_x$ podle z_x

Odhad spolehlivosti

$$x = (\hat{\mu}_x \pm z_x) [\text{jednotka}]$$

Pro normální rozdělení chyby

- $\hat{\mu}_x = \bar{X} = \frac{\sum_{i=1}^N x_i}{n}$
- s směrodatná odchylka jednoho měření, D rozptyl

$$s = \sqrt{D} = \sqrt{\frac{\sum_{i=1}^N (\bar{X} - x_i)^2}{n - 1}}$$

- $s_{\bar{x}} = \sqrt{\sum_{i=1}^N (\frac{1}{n})^2 s_{x_i}}$ a protože měření byly prováděny za stejných podmínek

$$s_{\bar{x}} = \frac{s_x}{\sqrt{n}} = \sqrt{\frac{\sum_{i=1}^N (\bar{X} - x_i)^2}{n(n - 1)}}$$

Odhad spolehlivosti

$$x = (\hat{\mu}_x \pm z_x) [\text{jednotka}]$$

Pro normální rozdělení chyby

- $z_x = t_{(p;n-1)} S_{\bar{x}}$

n \ P	0,683	0,954	0,99	n \ P	0,683	0,954	0,99
1	1,8395	13,8155	63,6567	16	1,0329	2,1633	2,9208
2	1,3224	4,5001	9,9248	18	1,0292	2,1433	2,8784
3	1,1978	3,2923	5,8409	20	1,0263	2,1276	2,8453
4	1,1425	2,8585	4,6041	30	1,0176	2,0817	2,75
5	1,1113	2,6396	4,0321	40	1,0133	2,0595	2,7045
6	1,0913	2,5084	3,7074	50	1,0108	2,0463	2,6778
7	1,0775	2,4214	3,4995	60	1,0091	2,0377	2,6603
8	1,0673	2,3594	3,3554	70	1,0078	2,0315	2,6479
9	1,0594	2,3131	3,2498	80	1,0069	2,0269	2,6387
10	1,0533	2,2773	3,1693	90	1,0062	2,0234	2,6316
12	1,0441	2,2253	3,0545	100	1,0057	2,0206	2,6259
14	1,0377	2,1895	2,9768				

Odhad spolehlivosti

$$x = (\hat{\mu}_x \pm z_x) \text{ [jednotka]}$$

Příklad – měření výšky válečku (Šťastný, 1997):

výška v [mm]	4,6	4,5	4,7	4,4	4,5	4,6	4,4	4,4	4,3	4,5
----------------	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

- $n = 10$
- $\bar{v} = 4,49$ [mm]
- $s_{\bar{v}} = 0,038$ [mm]
- $t_{(0,68;9)} = 1,059$
- $t_{(0,99;9)} = 3,250$

$$v = (4,49 \pm 0,04) \text{ mm} \quad \text{pro } p = 0,68$$

$$v = (4,49 \pm 0,12) \text{ mm} \quad \text{pro } p = 0,99$$

Zákon přenosu chyb

- Na základě Taylorova rozvoje do druhého členu

$$s_z^2 = \sum_{i=1}^N \left(\frac{\partial z}{\partial x_i} \right)^2 s_{x_i}^2 + 2 \sum_{i=1}^{N-1} \sum_{j=i+1}^N \frac{\partial z}{\partial x_i} \frac{\partial z}{\partial x_j} s_{x_i} s_{x_j} \rho_{ij},$$

kde $s_{x_i}^2$ je rozptyl (variance) x_i a ρ_{ij} je kovariance x_i a x_j .

Pro jednoduché případy, kdy x a y jsou nezávislé ($\rho_{ij} = 0$):

- aditivní funkce $z = ax \pm by$

$$s_z = \sqrt{a^2 s_x^2 + b^2 s_y^2}, \quad (1)$$

- multiplikativní funkce $z = ax^b y^c$

$$s_z = \bar{z} \sqrt{\left(\frac{b s_x}{x} \right)^2 + \left(\frac{c s_y}{y} \right)^2}. \quad (2)$$

kde $\bar{z} = a \bar{x}^b \bar{y}^c$, protože

$$\sum_{i=1}^N \left(\frac{\partial z}{\partial x_i} \right)^2 s_i^2 = \left(\frac{a b x^{b-1} y^c s_x}{x} \right)^2 + \left(\frac{a x^b c y^{c-1} s_y}{y} \right)^2 = z^2 \left(\left(\frac{b s_x}{x} \right)^2 + \left(\frac{c s_y}{y} \right)^2 \right)$$

- Příklad použití: <http://www.phy.ohiou.edu/~murphy/courses/sample.pdf>

Skripta Fr. Šťastného (Šťastný, 1997)

<http://amper.ped.muni.cz/jenik/nejistoty/>