Jazykové konstrukce pro paralelní výpočty

Paralelismus se vyskytuje na:

1. **Úrovni strojových instrukcí**
2. **Úrovni příkazů programovacího jazyka**
3. **Úrovni podprogramů**
4. **Úrovni programů**

Vývoj multiprocesorových architektur

1. **konec let 50. Jeden základní procesor a jeden či více speciálních procesorů pro I/O**
2. **polovina 60. víceprocesorové systémy užívané pro paralelní zpracování na úrovni programů**
3. **konec 60. víceprocesorové systémy užívané pro paralelní zpracování na instrukční úrovni**
4. **SIMD architektury (stejná instrukce současně zpracovávaná na více procesorech, na každém s jinými daty) - vektorové procesory**
5. **MIMD architektury (nezávisle pracující procesory, které mohou být synchronizovány)¨**

Paralelismus na úrovni podprogramů

Sekvenční výpočetní proces je v čase uspořádaná posloupnost operací

-Program	**- Procesy**	**- Procesor**
-Scénář	**- Role (Představení)**	**- Herec**

Paralelní procesy jsou vykonávány paralelně či pseudoparalelně

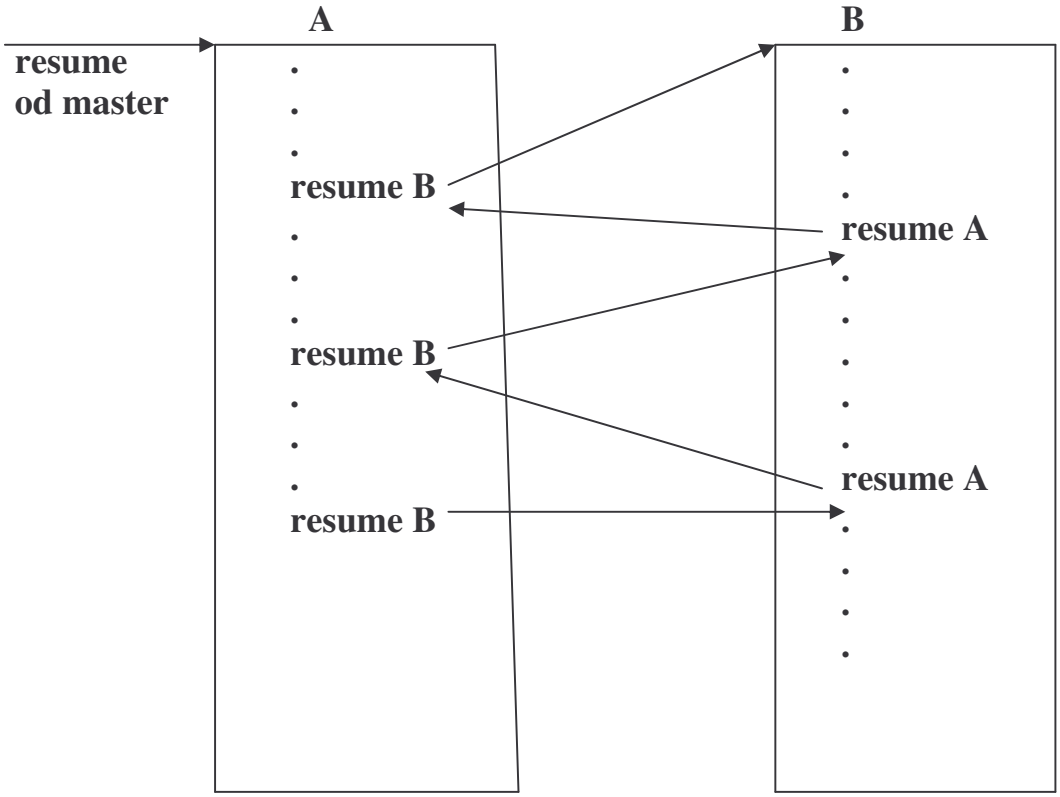
-nové problémy: **1. zablokování**
2. rychlostní závislost

Kategorie paralelismu:

1. **Fyzický paralelismus (více procesorů pro více procesů)**
2. **Logický paralelismus (time-sharing jednoho procesoru, v programu je více procesů)**
3. **Kvaziparalelismus (korutiny)**

Korutiny

Speciální druh podprogramů – volající a volaný nejsou v relaci „master slave“
Jsou si rovni (symetričtí)
Mají více vstupních bodů
Zachovávají svůj stav mezi aktivacemi
V daném okamžiku je prováděna jen jedna



Master (není korutinou) vytvoří deklarací korutiny, ty provedou inicializační kód a vrátí mastru řízení.
Master příkazem resume spustí jednu z korutin
Příkaz resume slouží pro start i pro restart
Pakliže jedna z korutin dojde na konec svého programu, předá řízení mastru

Paralelismus na úrovni podprogramů

Procesy mohou být

1. **nekomunikující (neví o ostatních, navzájem si nepřekáží)**
2. **komunikující (např. producent a konzument)**
3. **soutěžící (např. o sdílený prostředek)**

Vlákno (thread) výpočtu v programu je sekvence míst programu, kterými výpočet prochází.
Úkol (task) je programová jednotka (část programu), která může být prováděna paralelně s ostatními částmi programu. Každý úkol může představovat jedno vlákno.

Odlišnost vláken/úkolů od podprogramů:

- **mohou být implicitně spuštěny**
- **programová jednotka, která je spouští nemusí být pozastavena**
- **po jejich skončení se řízení nemusí vracet do místa odkud byly odstartovány**

Způsoby komunikace

- **sdílené nelokální proměnné**
- **parametry**
- **zasílání zpráv**

Při komunikaci musí A čekat na B

Při soutěžení sdílí A s B zdroj, který není použitelný simultánně (např. sdílený čítač) a vyžaduje přístup ve vzájemném vyloučení.

Části programu, které pracují ve vzájemném vyloučení se nazývají kritickými sekcemi.

Stavy úkolů:

1. **new**
2. **ready**
3. **running**
4. **blocked**
5. **completed**
6. **terminated**

Prostředky pro uskutečnění vzájemně výlučného přístupu ke sdíleným zdrojům

1. **Semafory**
2. **Monitory**
3. **Zasílání zpráv**

Semafor - datová struktura obsahující čítač a frontu pro ukládání deskriptorů úkolů. Má dvě operace - zaber a uvolni (P a V). Je použitelný jak pro soutěžící, tak pro spolupracující úkoly.

P a V jsou atomické operace

P(semafor) /* binární*/
if semafor = 1 then semafor := 0
else pozastav volající proces a dej ho do fronty na semafor

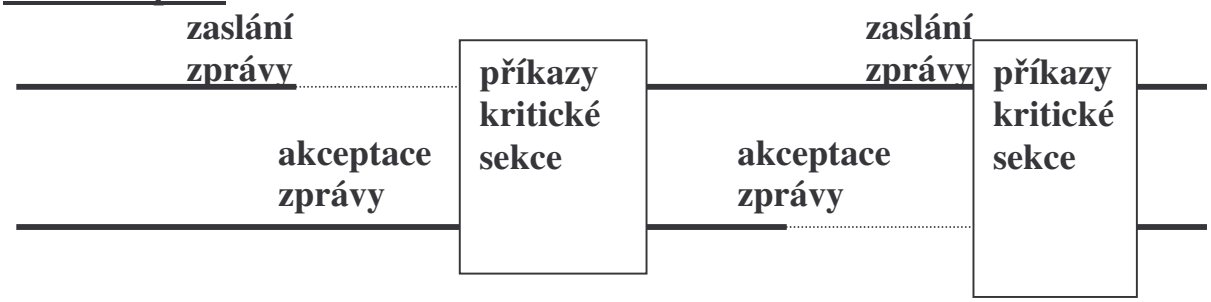
V(semafor) /* binární*/
if fronta na semafor je prázdná then semafor := 1
else vyber prvého z fronty a aktivuj ho

Nebezpečnost semaforů -nelze kontrolovat řádnost použití
deadlock

Obdobou je použití signálů
Send(signal) --je akcí procesu 1
Wait(signal) --je akcí procesu 2 (rozdíl proti P a V)

Monitor - programový modul zapouzdřující data spolu s procedurami, které s daty pracují. Procedury mají vlastnost, že vždy jen jeden úkol/vlákno může provádět monitorovou proceduru, ostatní čekají ve frontě. (Ada95, Java)

Zasílání zpráv -



Vede k principu schůzky (rendezvous Ada)

Paralelní konstrukce v jazyce ADA

Paralelně proveditelnou jednotkou je task

```
task  T is
        Deklarace tzv. vstupů (entry)      } specifikační část
end T;

task body T is
        Lokální deklarace a příkazy        } tělo
end T;
```

Lze zavést i typ úkol zápisem *task type T is* ve specifikační části a použít jej k deklaraci úkolů. Oproti proměnným,úkolům nelze přiřazovat a porovnávat.

Schůzka (rendezvous)

```
task ČIŠNÍK is
        entry PIVO (X: in INTEGER);
        entry PLATIT;
                …
end ČIŠNÍK ;

task body ČIŠNÍK is
 … --lokalni deklarace
  begin
    loop
        …--blouma u pultu
        accept PIVO (X: in INTEGER) do
        …--donese pivo
        end PIVO;
        …--sbira sklenice
        akcept PLATIT do
        …--inkasuje
        end PLATIT;
     end loop;
  end ČIŠNÍK ;
```

```
task HOST1 is
end HOST1;

task body HOST1 is
        …
        ČIŠNÍK.PIVO(2);
        …--pije pivo
        ČIŠNÍK.PLATIT;
        …
end HOST1;
```

Všechny úkoly se spustí současně, jakmile hlavní program dojde k begin své příkazové části.

Př.Schránka pro komunikaci producenta s konzumentem

```
task SCHRANKA is
        entry PUT(X: in INTEGER);
        entry GET(X: out INTEGER);
end SCHRANKA;

task body SCHRANKA is
        V: INTEGER;
begin
        loop
                accept PUT(X: in INTEGER) do
                        V := X;
                end PUT;
                accept GET(X: out INTEGER) do
                        X := V;
                end GET;
        end loop;
end SCHRANKA;
```

Producent: `SCHRANKA.PUT(456);`

Konzument: `SCHRANKA.GET( I );`

Př. Sdílená proměnná realizovaná úkolem

```
task SDILENA is
        entry PUT(X: in INTEGER);
        entry GET(X: out INTEGER);
end SDILENA;

task body SDILENA is
        V: INTEGER;
begin
        loop
          select                --dovoli alternativni provedeni
                accept PUT(X: in INTEGER) do
                        V := X;
                end PUT;
          or
                accept GET(X: out INTEGER) do
                        X := V;
                end GET;
          or
                terminate;  --umozni ukolu skoncit aniz projde koncovym end
          end select;
        end loop;
end SDILENA;
```

```
generic                    -- 1STACK5ADA
SIZE: POSITIVE;
type POLOZKA is private;
package STACK5 is
        task ZASOBNIK is
                entry VLOZ(X: in POLOZKA);
                entry UBER(X: out POLOZKA);
        end ZASOBNIK;
end STACK5;

package body STACK5 is
        task body ZASOBNIK is
        S: array (1..SIZE of POLOZKA;
        SP: INTEGER range 0..SIZE;
        begin
                SP := 0;
                loop
                        select
                                when SP < SIZE =>
                                        accept VLOZ(X : in POLOZKA) do
                                                SP := SP + 1;
                                                S(SP) := X;
                        or
                                when SP > 0 =>
                                        accept UBER(X : out POLOZKA) do
                                                X := S(SP);
                                                SP := SP - 1;
                                        end UBER;
                        or
                                terminate;
                        end select;
                wnd loop;
        end ZASOBNIK;
end STACK5;

with STACK5, TEXT_IO; use TEXT_IO;
procedure MAIN is
        package STACKI is new STACK5(20, INTEGER);
        package STACKC is new STACK5(100, CHARACTER);
        C: CHARACTER := 'A';
        I: INTEGER := 111;
begin ...
        STACKC.ZASOBNIK.VLOZ('B');                    . . .
        STACKI.ZASOBNIK.VLOZ(22);            . . .
        STACKC.ZASOBNIK.UBER(C);
        ...
end MAIN;
```

```
generic Size: INTEGER;
package IntMatrices is
  type IntMatrix is array (1..Size,1..Size) OF INTEGER;
  function ParMult(a,b: in IntMatrix) return IntMatrix;
end;
package body IntMatrices is
  function ParMult(a,b: in IntMatrix) return IntMatrix is
    c:IntMatrix;
    task type Mult is
      entry DoRow (i: in INTEGER); entry EndRow ;
    end;
    task body Mult is
      iloc: INTEGER;
    begin
      accept DoRow (i: in INTEGER) do  iloc := i; end;
      for j in 1..Size loop
        c(iloc, j) := 0;
        for k in 1..Size loop
          c(iloc, j) := c(iloc,j) + a(iloc,k) * b(k,j);
        end loop;
      end loop;
      accept EndRow;
    end Mult;
  begin -- ParMult
    declare m: array (1..Size) of Mult;
    begin
      for i in 1..Size loop
        m(i).DoRow(i);
      end loop;
      for i in 1..Size loop
        m(i).EndRow;
      end loop;
    end;
    return c;
  end ParMult;
end IntMatrices;
with IntMatrices;  with Text_IO; use Text_IO;
procedure Zkus is
    package Muj is new IntMatrices(2);
    package Muj_IO is new Integer_IO(integer);
    use Muj, Muj_IO;
    A: IntMatrix := ((5,5),(5,5)); B: IntMatrix := ((3,3),(3,3));
    C: IntMatrix;
begin
        C:=ParMult(A,B);
        put(c(1,1)); put(c(1,2));put(c(2,1));put(c(2,2));
end Zkus;
```

```
with ADA.Text_IO; use ADA.Text_IO;          -- pr. 2NasobMaticeADA
procedure NAS_MAT is
  type RADKOVA_MATICE is array (INTEGER range <>) of FLOAT;
  type POINTER is access RADKOVA_MATICE;
  RAD: INTEGER;
  package F_IO is new Float_IO(Float); use F_IO;
  package I_IO is new Integer_IO(Integer); use I_IO;
  task type PARCIALNI_SOUCIN is
    entry PRIJATA_HODNOTA(P : out FLOAT);
    entry VYSLANA_HODNOTA(V_1, V_2: in RADKOVA_MATICE);
  end PARCIALNI_SOUCIN;
  task body PARCIALNI_SOUCIN is
    SOUCIN: FLOAT;
    VEKTOR_1, VEKTOR_2: POINTER;
  begin
    accept VYSLANA_HODNOTA(V_1,V_2: in RADKOVA_MATICE) do
         VEKTOR_1 := new RADKOVA_MATICE'(V_1); --inicializuje VEKTOR_1
         VEKTOR_2 := new RADKOVA_MATICE'(V_2); --hodnotou V_1, pod.V_2
    end VYSLANA_HODNOTA;
    SOUCIN := 0.0;
    for I in VEKTOR_1.all'RANGE   --ukazuje na RADKOVA_MATICE
         loop SOUCIN := SOUCIN + (VEKTOR_1(I)*VEKTOR_2(I));
    end loop;
    accept PRIJATA_HODNOTA(P: out FLOAT) do
         P := SOUCIN;
    end PRIJATA_HODNOTA;
  end PARCIALNI_SOUCIN;
begin
  get(RAD);
  declare
    type MATICE is array(1..RAD) of RADKOVA_MATICE(1..RAD);
    X:MATICE;
    U,P:RADKOVA_MATICE(1..RAD);
    PARALELNI_SOUCIN: array(U'RANGE) of PARCIALNI_SOUCIN;
  begin
    for I in X(1)'RANGE loop
         for J in X(2)'RANGE loop
           GET(X(I)(J));
         end loop;
    end loop;
    for I in U'RANGE loop GET(U(I));
    end loop; --tento cyklus musi byt zvlast od toho nasledujiciho
    for I in U'RANGE loop
         PARALELNI_SOUCIN(I).VYSLANA_HODNOTA(X(I), U);
    end loop;
    for I in P'RANGE loop
         PARALELNI_SOUCIN(I).PRIJATA_HODNOTA(P(I));
         PUT(P(I)); NEW_LINE;
    end loop;
  end;
end NAS_MAT;
```

PROTECTED objekt (je to monitor)

Mechanismus rendezvous : -vede k vytváření dodatečných úkolů pro obsluhu sdílených dat
Typ protected má specifikační část (obsahuje přístupový protokol k objektu) a tělo (obsahuje implementační detaily)
Umožňuje synchronizovaný přístup k privátním datům objektů bez zavádění dodatečných úkolů pomocí operací:

1. **funkce – může je provádět lib. počet klientů, pokud není právě volána procedura nebo vstup**
2. **2. procedury – pouze 1 volající ji může provádět a nesmí současně být volána funkce či vstup**
3. **3. vstupy – jako procedury, ale jen při splněné tzv. bariéře**

```
př.     protected  Variable  is
            function  Read  return  Item ;
            procedure  Write ( New_Value : Item ) ;
        private
            Data : Item ;
        end  Variable;

        protected  body  Variable  is
            function  Read  return  Item  is
            begin
                return  Data ;
            end  Read ;
            procedure  Write ( New_Value : Item )  is
            begin
                Data := New_Value;
            end  Write ;
        end Variable;
```

Chráněný objekt Variable poskytuje řízený přístup k privátní proměnné Data. Způsob volání prostřednictvím „ .“ notace:

```
X  :=  Variable . Read ;
. . .
Variable . Write ( New_Value  => Y ) ;
```

Realizuje monitor = pasivnost
= programový modul zapouzdřující data a procedury s nimi pracující. Procedury mají vlastnost, že vždy jen jeden proces může provádět monitorovou proceduru, ostatní čekají ve frontě na monitor. Funkci může provádět více procesů najednou

Chráněné typy mohou mít kromě proc. a fcí i vstupy (entry) s bariérou ≅ accept úkolu s hlídkou (ale pozměněný efekt vyhodnocení bariéry)

př. Omezený buffer

```
protected type Bounded_Buffer is
        entry Put ( X : in Item ) ;
        entry Get ( X : out Item ) ;
private
        A : Item_Array ( 1 . . Max );
        I , J : Integer range 1 . . Max := 1;
        Count : Integer range 0 . . Max := 0 ;
end Bounded_Buffer;

protected body Bounded_Buffer is
        entry Put ( X : in Item ) when Count < Max is
        begin
                A ( I ) := X ;
                I := I mod Max + 1 ; Count := Count + 1;
        end Put;
        entry Get ( X : out Item ) when Count > 0 is
        begin
                X := A ( J ) ;
                J := J mod Max + 1 ; Count := Count - 1 ;
        end Get;
end Bounded_Buffer ;
```

```
př. použití:        Muj_Buffer : Bounded_buffer ;
                    Muj_Buffer . Put ( X ) ;
```

1. Při volání se vyhodnotí bariéra a je-li false je volání frontováno.
2. Na konci exekuce každého těla entry či procedury (ale ne funkce - ta nemění stav) se přepočítají všechny bariery a ta volání z front, která mají nyní bariéry true se všechna umožní provést.

Vstupy (stejně jako procedury) jsou prováděny v režimu vzájemného vyloučení

Nebylo potřeba vytvářet úkol

Chráněné proměnné jsou efektivním způsobem realizace synchronizačních prostředků

př. Obecný semafor

```
protected  type  Semafor ( Start : Integer := 1 ) is
        entry  Zaber ;                  -- P   (Passeren)
        entry  Uvolni ;                 -- V    (Vrijmaken)
        function Citac  return  Integer;
private
        CC : Integer := Start ;
end  Semafor ;

protected  body  Semafor  is
        entry Zaber  when  CC > 0  is
        begin
                CC := CC - 1 ;
        end  Zaber;
        entry  Uvolni  when  CC <  Start  is
        begin
                CC := CC + 1 ;
        end  Uvolni ;
        function  Citac  return  Integer  is
        begin
                return  CC ;
        end  Citac ;
end  Semafor ;
```

Semafor je chráněný typ s diskriminantem Start (t.j. počet chráněných zdrojů). Diskriminant může mít i úkol.

Zdroje informace: http://www.adahome.com/Tutorials/Lovelace/lovelace.htm

Java threads

Paralelně proveditelné jednotky jsou objekty s metodou run, jejíž kód může být prováděn souběžně s jinými takovými metodami a s metodou main. Metoda run se spustí nepřímo vyvoláním start( )
Jak definovat třídy, jejichž objekty mohou mít paralelně prováděné metody

1. **jako podtřídy třídy Thread (je součástí java.lang balíku)**
2. **implementací rozhrani Runnable**

ad1.

```
class MyThread extends Thread   //1.Z třídy Thread odvodíme potomka (s run metodou)
{public void run( )  { . . .}
  . . .
}
. . .
MyThread t = new MyThread( ); //2.Vytvoření instance této třídy potomka
. . .
```

---

ad2.

```
class MyR implements Runnable     //1.konstruujeme třídu implementující Runnable
{public void run( )  { . . .}
  . . .
}
. . .
MyR  m = new MyR( );  // 2.konstrukce objektu teto třídy (s metodou run)
Thread t = new Thread(m); //3.vytvoření vlákna na tomto objektu
//je zde pouzit konstruktor Thread(Runnable threadOb)
. . .
```

Vlákno t se spustí až provedením příkazu `t.start( );`

Třída Thread má metody:

```
final String getName( )
final int getPriority( )
final int setPriority( )
final boolean isAllive( )
final void join( )
void run( )
static void sleep(long milisekundy)
void start( )
. . .
```

Rozhraní Runnable má jen metodu run( )

```
class MyThread extends Thread { // pr. 3aVlakna
  int count;

  MyThread(String name) {
    super(name);
    count = 0;
  }

    public void run() { // vstupni bod vlakna
    System.out.println(getName() + " startuje.");
    try {
      do {
        Thread.sleep(500); //Kvalifikace není nutna
        System.out.println("Ve vlaknu " + getName() +
                           ", citac je " + count);
        count++;
      } while(count < 5);
    }
    catch(InterruptedException exc) {//nutno ošetřit přerušeni spani
      System.out.println(getName() + " prerusene.");
    }
    System.out.println(getName() + " ukoncene.");
  }
}

class Vlakno {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    // Nejdříve konstruujeme MyThread objekt.
    MyThread mt = new MyThread("potomek");

    // Az pak startujeme vypocet vlakna
    mt.start();

    do {
      System.out.print(".");
      try {
        Thread.sleep(100); //Kvalifikace je nutna
      }
      catch(InterruptedException exc) {//nutno ošetřit přerušeni spani
        System.out.println("Hlavni vlakno prerusene.");
      }
    } while (mt.count != 5);
    System.out.println("Konci hlavni vlakno");
  }
}
```

```
class MyThread implements Runnable { // pr. 3Vlakna
  int count;
  String thrdName;

  MyThread(String name) {
    count = 0;
    thrdName = name; //retezec slouzici jako jmeno vlakna
  }

    public void run() { // vstupni bod vlakna
    System.out.println(thrdName + " startuje.");
    try {
      do {
        Thread.sleep(500);
        System.out.println("Ve vlaknu " + thrdName +
                           ", citac je " + count);
        count++;
      } while(count < 5);
    }
    catch(InterruptedException exc) {//nutno ošetřit přerušeni spani
      System.out.println(thrdName + " preruseny.");
    }
    System.out.println(thrdName + " ukonceny.");
  }
}

class Vlakno {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    // Nejdříve konstruujeme MyThread objekt.
    MyThread mt = new MyThread("potomek");

    // Pak konstruujeme vlakno z tohoto objektu
    Thread newThrd = new Thread(mt);

    // Az pak startujeme vypocet vlakna
    newThrd.start();

    do {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {//nutno ošetřit přerušeni spani
        System.out.println("Hlavni vlakno prerusene.");
      }
    } while (mt.count != 5);

    System.out.println("Konci hlavni vlakno");
  }
}
```

```
class MyThread implements Runnable { // pr. 4Vlakna
  int count;
  Thread thrd;

  // Konstruuje nove vlakno
  MyThread(String name) {
    thrd = new Thread(this, name);
    count = 0;
    thrd.start(); // startuje vlakno rovnou v konstruktoru
  }

  // Začátek exekuce vlakna
  public void run() {
    System.out.println(thrd.getName() + " startuje ");
    try {
      do {
        Thread.sleep(500);
        System.out.println("V potomkovi " + thrd.getName() +
                           ", citac je " + count);
        count++;
      } while(count < 5);
    }
    catch(InterruptedException exc) {
      System.out.println(thrd.getName() + " preruseny.");
    }
    System.out.println(thrd.getName() + " ukonceny.");
  }
}

class VlaknoLepsi {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt = new MyThread("potomek");

    do {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {
        System.out.println("Hlavni vlakno prerusene.");
      }
    } while (mt.count != 5);

    System.out.println("Hlavni vlakno konci");
  }
}
```

```
class MyThread extends Thread {      // pr. 5Vlakna = totez jako 4Vlakna ale dedenim z Thread
  int count;

    MyThread(String name) {
    super(name); // jmeno vlakna
    count = 0;
    start(); // startuje v konstruktoru
  }

   public void run() {
    System.out.println(getName() + " startuje");
    try {
      do {
        Thread.sleep(500);  //zde kvalifikace neni nutna
        System.out.println("V " + getName() +
                           ", citac je " + count);
        count++;
      } while(count < 5);
    }
    catch(InterruptedException exc) {
      System.out.println(getName() + " prerusene");
    }
    System.out.println(getName() + " ukoncene");
  }
}

class DediThread {
  public static void main(String args[]) {
    System.out.println("Hlavni vl.startuje");

    MyThread mt = new MyThread("potomek");

    do {
      System.out.print(".");
      try {
        Thread.sleep(100); //zde je nutna kvalifikace
      }
      catch(InterruptedException exc) {
        System.out.println("Hlavni vl. prerusene");
      }
    } while (mt.count != 5);

    System.out.println("Hlavni vl. konci");
  }
}
```

```
class MyThread extends Thread { // pr 6a Spusteni vice vlaken
  int count;

  MyThread(String name) {
    super(name);
    count = 0;
    start(); // start
  }

  public void run() {
    System.out.println(getName() + " startuje");
    try {
      do {
        Thread.sleep(500);
        System.out.println("Ve " + getName() +
                           ", citac je " + count);
        count++;
      } while(count < 3);
    }
    catch(InterruptedException exc) {
      System.out.println(getName() + " preruseny");
    }
    System.out.println(getName() + " ukonceny");
  }
}

class ViceVlaken {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt1 = new MyThread("potomek1");
    MyThread mt2 = new MyThread("potomek2");
    MyThread mt3 = new MyThread("potomek3");

    do {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {
        System.out.println("Hlavni vlakno prerusene");
      }
    } while (mt1.count < 3 ||
             mt2.count < 3 ||
             mt3.count < 3);

    System.out.println("Hlavni vl. konci");
  }
}
```

```
class MyThread implements Runnable {  // pr. 6Vlakna spusteni vice vlaken
  int count;
  Thread thrd;

  MyThread(String name) {
    thrd = new Thread(this, name);
    count = 0;
    thrd.start(); // start
  }

  public void run() {
    System.out.println(thrd.getName() + " startuje");
    try {
      do {
        Thread.sleep(500);
        System.out.println("Ve " + thrd.getName() +
                           ", citac je " + count);
        count++;
      } while(count < 3);
    }
    catch(InterruptedException exc) {
      System.out.println(thrd.getName() + " prerusene");
    }
    System.out.println(thrd.getName() + " ukoncene");
  }
}

class ViceVlaken {
  public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt1 = new MyThread("potomek1");
    MyThread mt2 = new MyThread("potomek2");
    MyThread mt3 = new MyThread("potomek3");

    do {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {
        System.out.println("Hlavni vlakno prerusene");
      }
    } while (mt1.count < 3 ||
             mt2.count < 3 ||
             mt3.count < 3);

    System.out.println("Hlavni vl. konci");
  }
}
```

Identifikace ukončení činnosti vláken

-nejčastěji k zastavení dojde doběhnutím metody run( )

-stav lze testovat metodou isAllive( )
tvaru: final boolean isAllive( )
? jak využít v modifikaci předchozích programů?

-čekáním na skončení jiného vlákna vyvoláním metody join()
tvaru: final void join( ) throws InterruptedException

```
Thread t = new Thread(m);
t.start( ); // zahaji cinnost
//rodic neco dela
t.join( ); //rodic ceka na skonceni t
// rodic pokracuje po skonceni t
```

-existuje alternativa čekání na skončení vlákna, informující, že se čeká na jeho konec

```
Thread t = new Thread(m);
t.start( ); // zahaji cinnost
//rodic neco dela
t.interrupt( ) ; //oznamuje t, ze na nej cekame, predcasne ho probudi
t.join( ); //rodic ceka na skonceni t
// rodic pokracuje po skonceni t
```

-existuje alternative pro timeout t.join(milisekundy)

```
...
} while (mt1.thrd.isAlive() ||
         mt2.thrd.isAlive() ||
         mt3.thrd.isAlive());

    System.out.println("Main thread ending.");
  }
}
```

```
//Pr 7aVlakna   pouziti join
class MyThread extends Thread {
  int count;
  MyThread(String name) {
    super(name);
    count = 0;
    start(); // start
  }
   public void run() {
    System.out.println(getName() + " startuje");
    try {
      do {
        Thread.sleep(500);
        System.out.println("Ve " + getName() +
                           ", citac je " + count);
        count++;
      } while(count < 3);
    }
    catch(InterruptedException exc) {
      System.out.println(getName() + " preruseny");
    }
    System.out.println(getName() + " konci");
  }
}

class Join {
    public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt1 = new MyThread("potomek1");
    MyThread mt2 = new MyThread("potomek2");
    MyThread mt3 = new MyThread("potomek3");

    try {
      mt3.join();
      System.out.println("potomek3 joined.");
      mt2.join();
      System.out.println("potomek2 joined.");
      mt1.join();
      System.out.println("potomek1 joined.");
    }
    catch(InterruptedException exc) {
      System.out.println("Hlavni vlakno prerusene");
    }
     System.out.println("Hlavni vl. konci");
  }
}
```

Pr. 7Vlakna

class MyThread implements Runnable {

```
...
class Join {
    public static void main(String args[]) {
    System.out.println("Hlavni vlakno startuje");

    MyThread mt1 = new MyThread("potomek1");
    MyThread mt2 = new MyThread("potomek2");
    MyThread mt3 = new MyThread("potomek3");

    try {
      mt1.thrd.join();
      System.out.println("potomek1 joined.");
      mt2.thrd.join();
      System.out.println("potomek2 joined.");
      mt3.thrd.join();
      System.out.println("potomek3 joined.");
    }
    catch(InterruptedException exc) {
      System.out.println("Hlavni vlakno preruseno");
    }

    System.out.println("Hlavni vl. konci");
  }
}
```

Priorita vláken

- **Vysoká priorita = hodně času procesoru**
- **Nízká priorita = méně času procesoru**
- **Implicitně je přidělena priorita potomkovi jako má nadřízený process**
- **Změnit lze prioritu metodou setPriority()**

final void setPriority(int cislo)

Min_Priority ≤ cislo ≥ Max_Priority
1 . . 10
Norm_Priority = 5

final int getPriority( )

```
class Priority extends Thread { //Pr.8aVlakna  - projevi se v OS s Time-Slicing
  int count;
  static boolean stop = false;
  static String currentName;
  Priority(String name) {
    super(name);
    count = 0;
    currentName = name;
  }
 public void run() {
    System.out.println(getName() + " start ");
    do {
      count++;
      if(currentName.compareTo(getName()) != 0) {
        currentName = getName();
        System.out.println("Ve " + currentName);
      }
    } while(stop == false && count < 50);
    stop = true;
    System.out.println("\n" + getName() + " konci");
  }
}

class Priorita {
  public static void main(String args[]) {
    Priority mt1 = new Priority("Vysoka Priorita");
    Priority mt2 = new Priority("Nizka Priorita");

    // nastaveni priorit
    mt1.setPriority(Thread.NORM_PRIORITY+2);
    mt2.setPriority(Thread.NORM_PRIORITY-2);
    // start vlaken
    mt1.start();
    mt2.start();
    try {
      mt1.join();
      mt2.join();
    }
    catch(InterruptedException exc) {
      System.out.println("Hlavni vlakno konci");
    }
    System.out.println("Vlakno s velkou prioritou nacitalo " +
                       mt1.count);
    System.out.println("Vlakno s malou prioritou nacitalo " +
                       mt2.count);
  }
}
```

```
class Priority implements Runnable {  //Pr. 8Vlakna
  int count;
  Thread thrd;
  static boolean stop = false;
  static String currentName;
  Priority(String name) {
    thrd = new Thread(this, name);
    count = 0;
    currentName = name;
  }
  public void run() {
    System.out.println(thrd.getName() + " start ");
    do {
      count++;

      if(currentName.compareTo(thrd.getName()) != 0) {
        currentName = thrd.getName();
        System.out.println("V " + currentName);
      }
    } while(stop == false && count < 500);
    stop = true;
    System.out.println("\n" + thrd.getName() + " terminating.");
  }
}

class Priorita {
  public static void main(String args[]) {
    Priority mt1 = new Priority("Vysoka Priorita");
    Priority mt2 = new Priority("Nizka Priorita");
    // nastaveni priorit
    mt1.thrd.setPriority(Thread.NORM_PRIORITY+2);
    mt2.thrd.setPriority(Thread.NORM_PRIORITY-2);
    // start vlaken
    mt1.thrd.start();
    mt2.thrd.start();

    try {
      mt1.thrd.join();
      mt2.thrd.join();
    }
    catch(InterruptedException exc) {
      System.out.println("Hlavni vlakno preruseno");
    }

    System.out.println("Vlakno s velkou prioritou nacitalo " +
                       mt1.count);
    System.out.println("Vlakno s malou prioritou nacitalo" +
                       mt2.count);
  }
}
```

Kritické sekce

(řešení problému sdílení zdrojů formou vzájemného vyloučení současného přístupu)

- **metoda s označením synchronized uzamkne objekt pro který je volána**
- **jiná vlákna pokoušející se použít synchr. metodu uzamčeného objektu musí čekat ve frontě**
- **když proces opustí synchr. metodu, objekt se odemkne**
- **objekt může mít současně synch.i nesynch. metody, a ty nevyžadují zámek = vada**

(Každý objekt Javy je vybaven zámkem, který musí vlákno vlastnit, chce-li provést synchronized metodu na objektu.)

např.
```
class Queue {
        . . .
         public synchronized int vyber( ){ …}
        . . .
        public synchronized  void uloz(int co){ …}
        . . .
}
```

- **synchronizovaný příkaz tvaru**
  synchronized (výraz s hodnotou objekt) příkaz

zamkne přístup k objektu nad kterým pracuje. Objekt musí systém vybavit frontou pro metody, které chtějí s objektem pracovat.

Komunikace mezi vlákny

(řeší situaci, kdy metoda vlákna potřebuje přístup k dočasně nepřístupnému zdroji

- **může čekat v nějakém cyklu (neefektivní využití objektu nad nímž pracuje)**
- **může se zřeknout kontroly nad objektem a jiným vláknům umožnit ho používat, musí jim to ale dát na vědomí**

Kooperace procesů zajišťují metody:

- **wait() vlákno přejde do stavu blokovaný a uvolní zámek objektu**
  verze:
  final void wait( ) throws InterruptedException
  final void wait(long milisec) throws InterruptedException
  final void wait(long milisec, int nonosec) throws InterruptedException
- **final void notify() oživí vlákno z čela fronty na objekt**
- **final void notifyAll() oživí všechna vlákna nárokující si přístup k objektu**

mohou být volány jen ze synchronized metod, jsou děděny z pratřídy Object

Př. Semafor jako ADT v Javě

```
class Semafor {
        private int count;

        public Semafor(int initialCount) {
                count = initialCount;
        }

        public synchronized void cekej( ) {
                try {
                while (count <= 0 ) wait( );
                count--;
                }
                catch (InterruptedException e) { }
        }

        public synchronized void uvolni( ) {
                count++;
                notify( );
        }
}
```

Každé vlákno je instancí třídy java.lang.Thread nebo jejího potomka

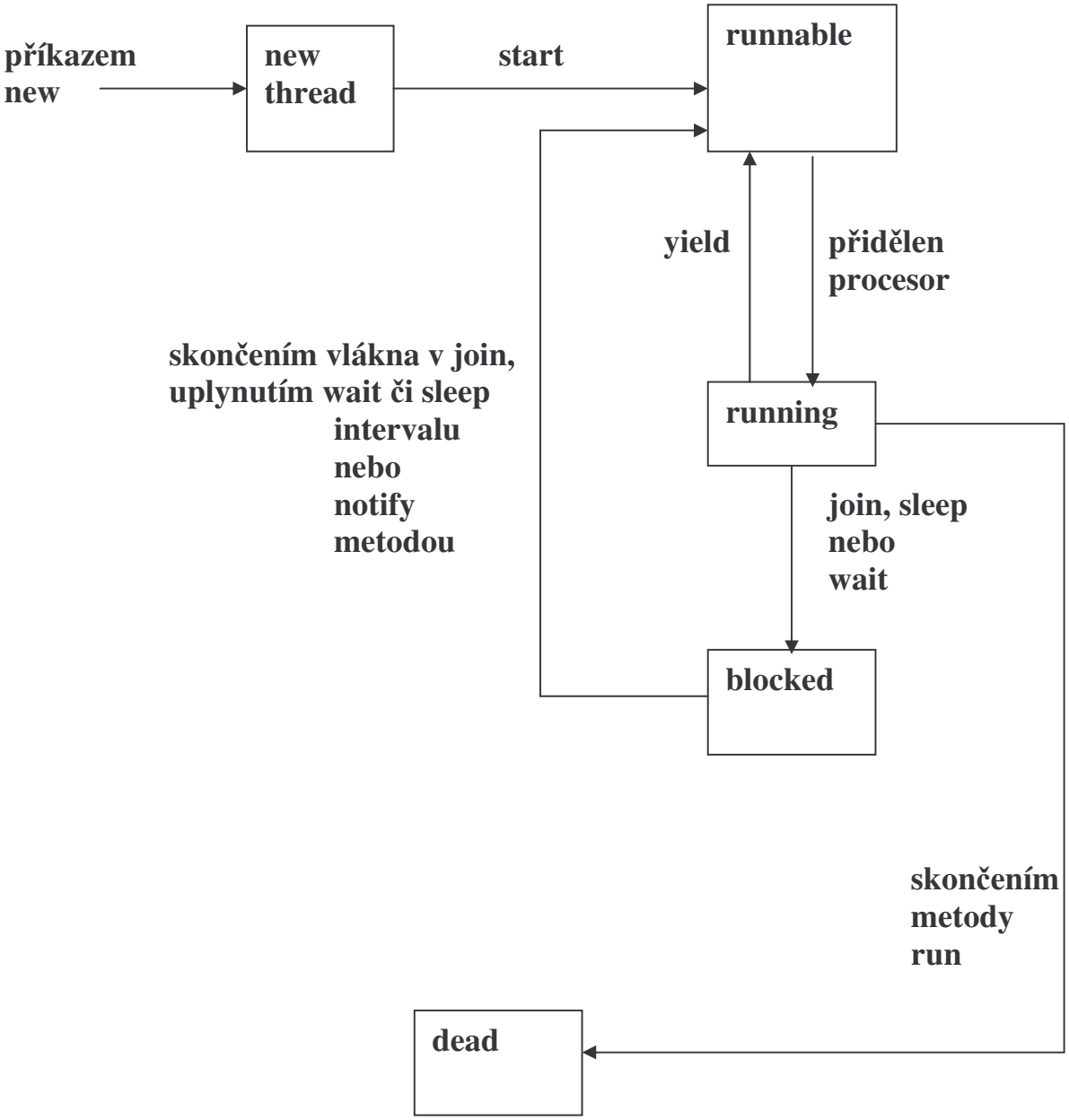
Thread má metody:

- **run()** **je vždy přepsána v potomku Thread, udává činnost vlákna**
- **start()** **spustí vlákno (tj. metodu run) a volající start pak pokračuje ve výpočtu. Metoda run není přímo spustitelná**
- **yield()** **odevzdání zbytku přiděleného času a zařazení do fronty na procesor**
- **sleep(milisec)** **zablokování vlákna na daný čas. Interval**
- **isAllive( )** **běží-li, vrací true, jinak false**
- **join( )**
- **getPriority**
- **setPriority**
- **final void notify()** **oživí vlákno z čela fronty na objekt**
- **final void notifyAll()** **oživí všechna vlákna nárokující si přístup k objektu**
-
- **. . . a další cca 20**

stavy vláken:

- **nové**
- **připravené**
- **běžící**
- **blokované**
- **mrtvé**

Plánovač vybere z fronty připravených vlákno s nejvyšší prioritou



Obr. Přechody mezi stavy vlákna Javy

```
import java.io.*;     // pr. 9Vlakna
class Queue { private int [] que;
        private int nextIn, nextOut, filled, queSize;
        public Queue(int size) {
                que = new int [size];
                filled = 0;
                nextIn = 1;
                nextOut = 1;
                queSize = size;
        } // konec konstruktoru

        public synchronized void deposit (int item) {
                try {
                        while (filled == queSize)
                                wait();
                        que [nextIn] = item;
                        nextIn = (nextIn % queSize) + 1;
                        filled++;
                        notify();
                } //konec try
                catch (InterruptedException e) {
                        System.out.println("int.depos");
                }
        } //konec deposit
        public synchronized int fetch() {
                int item = 0;
                try {
                        while (filled == 0)
                                wait();
                        item = que [nextOut];
                        nextOut = (nextOut % queSize) + 1;
                        filled--;
                        notify();
                } //konec try
                catch(InterruptedException e) {
                        System.out.println("int.fetch");
                }
                return item;
        } //konec fetch
} //konec tridy Queue
```

```
class Producer extends Thread {
        private Queue buffer;
        public Producer(Queue que) {
                buffer = que;
        }
        public void run() {
           int new_item = 0; // neprelozi bez inicializace
          opakuj: while (new_item > -1) { /*ukoncime-1 nebo
                                        zapornym cislem*/
                 try {//produkce
                byte[] vstupniBuffer = new byte[20];
                System.in.read(vstupniBuffer);
                String s = new String(vstupniBuffer).trim();
                new_item = Integer.valueOf(s).intValue();
                 }
                 catch (NumberFormatException e) {
                        System.out.println("nebylo to dobre");
                        continue opakuj;
                 }
                 catch (IOException e) {//zachytava nepripr.klavesnici
                        System.out.println("chyba cteni");
                 }
                 buffer.deposit(new_item);
                }
        }
}
class Consumer extends Thread {
        private Queue buffer;
        public Consumer(Queue que) {
                buffer = que;
        }
        public void run() {
                int stored_item = 0;  //chce inicializaci
                while (stored_item > -1) {/*ukoncime -1 nebo
                                        minus cislem*/
                 stored_item = buffer.fetch();
                 System.out.println(stored_item);       //konzumace
                }
        }
}
```

```
public class P_C {
	public static void main(String [] args) {
		Queue buff1 = new Queue(100);
		Producer producer1 = new Producer(buff1);
		Consumer consumer1 = new Consumer(buff1);
		producer1.start();
		consumer1.start();
	}
}
```

Paralelismus na úrovni příkazů jazyka

OCCAM

parbegin S1; S2; . . . Sn; parend;

High performance Fortran

Založen na modelu SIMD:

- **výpočet je popsán jednovláknovým programem**
- **proměnné (obvykle pole) lze distribuovat mezi více procesorů**
- **distribuce, přístup k proměnným a synchronizace procesorů je zabezpečena kompilátorem**

```
REAL DIMENSION (1000, 1000) :: A, B
INTEGER  I, J
…
DO I = 2, N
   DO J = 1, I – 1
         A(I, J) = A(I, J) / A(I, I)
   END DO
END DO
```

```
FORALL (I = 2 : N, J = 1 : N, J .LT. I )  A(I, J) = A(I, J) / A(I, I)
```

FORALL představuje zobecněný přiřazovací příkaz (a ne smyčku)
FORALL lze použít, pokud je zaručeno, že výsledek seriového i paralelního zpracování budou identické.

Paralelismus na úrovni programů

Pouze celý program může být paralelní aktivitou.
Fork příkazem Unixu vznikne potomek – přesná kopie volajícího procesu

```
#define SIZE 100
#define NUMPROCS 10
int a[SIZE] [SIZE], b[SIZE] [SIZE], c[SIZE] [SIZE];

void multiply(int myid)
{ int i, j, k;
  for (i = myid; i < SIZE; i+= NUMPROCS)
    for (j = 0; j < SIZE; ++j)
    { c[i][j] = 0;
      for (k = 0; k < SIZE; ++k)
        c[i][j] += a[i][k] * b[k][j];
    }
}

main()
{ int myid;
  /* prikazy pro vstup a, b */
  for (myid = 0; myid < NUMPROCS; ++myid)
    if (fork() == 0)
    { multiply(myid);
      exit(0);}
  for (myid = 0; myid < NUMPROCS; ++myid)
    wait(0);
  /* prikazy pro vystup c */
  return 0;
}
```