Paradigmata programování 4

Přednáška 1. Paralelní výpočty

verze z 12. února 2025

Jan Laštovička

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Obsah

Plán

1. Paralelní programování (usuzování o programu)
2. Logické programování (programování usuzováním)

Paralelní programování

Základní pojmy:

- procesor
- proces
- souběžné vykonávání
- paralelní program
- sdílená paměť

Paralelní programování

Základní pojmy:

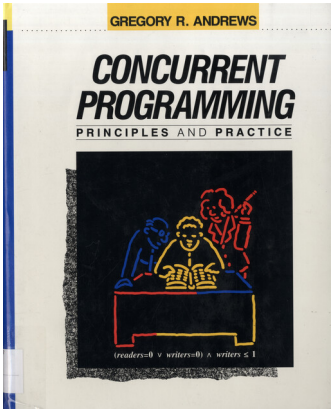
- procesor
- proces
- souběžné vykonávání
- paralelní program
- sdílená paměť

Použití:

- zvýšení výkonu
- rozdělení aplikace

Kniha

ANDREWS, Gregory R. Concurrent programming: principles and practice (1991)



Odkaz do knihovny

Model vykonávání paralelního programu

Model vykonávání paralelního programu

- jeden procesor
- více procesů
- střídavé vykonávání

Model vykonávání paralelního programu

- jeden procesor
- více procesů
- střídavé vykonávání
- Více procesorů nepřinese změnu.

Model vykonávání paralelního programu

- jeden procesor
- více procesů
- střídavé vykonávání
- Více procesorů nepřinese změnu.

jemná atomické instrukce

- Během vykonávání není možné přerušit vykonávání procesu.

Model vykonávání paralelního programu

- jeden procesor
- více procesů
- střídavé vykonávání
- Více procesorů nepřinese změnu.

jemná atomické instrukce

- Během vykonávání není možné přerušit vykonávání procesu.

Jazyk pro paralelní výpočty:

- založený na zásobníkovém jazyce
- známe z PP2

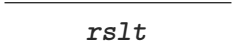
Obsah

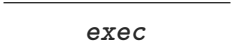
Zásobníkový stroj

- datový zásobník *rslt*
- programový zásobník *exec*

Zásobníkový stroj

- datový zásobník *rslt*
- programový zásobník *exec*



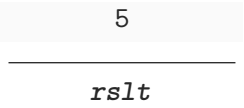


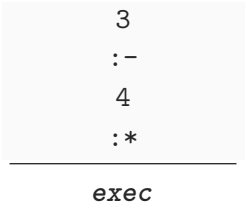
Zásobníkový stroj

- datový zásobník *rslt*
- programový zásobník *exec*

Zásobníkový stroj

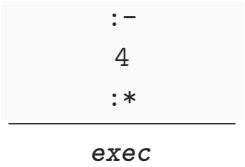
- datový zásobník *rslt*
- programový zásobník *exec*





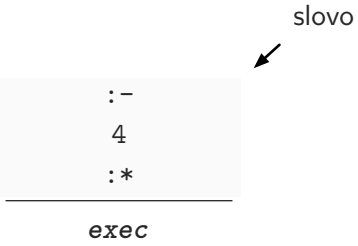
Zásobníkový stroj

- datový zásobník *rslt*
- programový zásobník *exec*



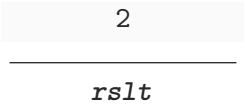
Zásobníkový stroj

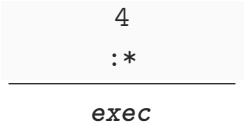
- datový zásobník *rslt*
- programový zásobník *exec*



Zásobníkový stroj

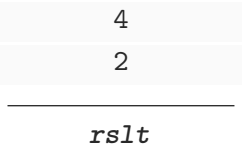
- datový zásobník *rslt*
- programový zásobník *exec*

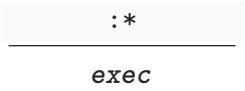




Zásobníkový stroj

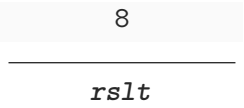
- datový zásobník *rslt*
- programový zásobník *exec*





Zásobníkový stroj

- datový zásobník *rslt*
- programový zásobník *exec*



Větvení

`:if ... :else ... :then`

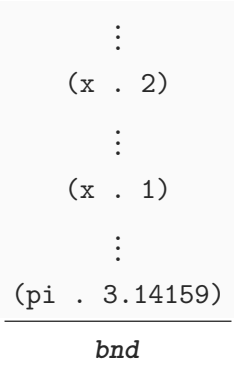
1. Odstraní vrchol zásobníku *rslt*.
2. Je-li odstraněná hodnota *Pravda*, způsobí vykonání kódu mezi `:else` a `then`.
3. Je-li *Nepravda*, způsobí vykonání kódu po `:else`.

Požívá pomocný zásobník: ***seek***

Příklad: `1 1 := :if 1 :else 2 :then`

Proměnné

- Ukládáme na zásobník *bnd*.
- Hodnota se hledá shora dolů, vazby se mohou zastiňovat (překrývat).
- Hodnoty vazeb lze měnit.
- Jsou dynamické.



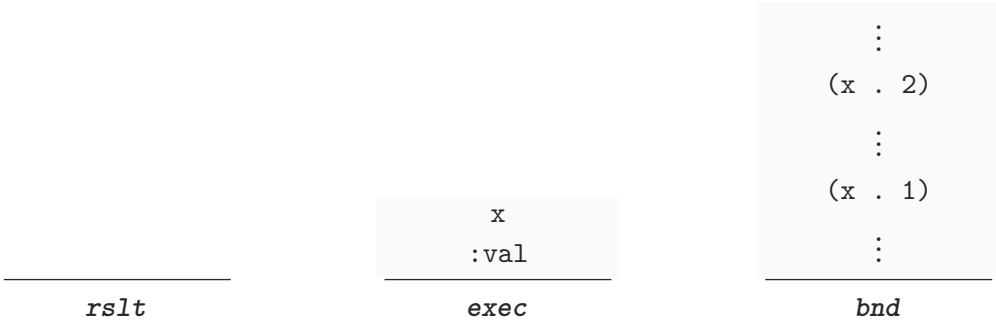
Získání hodnoty proměnné

Získání hodnoty proměnné

`:val` dá hodnotu proměnné na zásobník *rslt*

Získání hodnoty proměnné

`:val` dá hodnotu proměnné na zásobník *rslt*

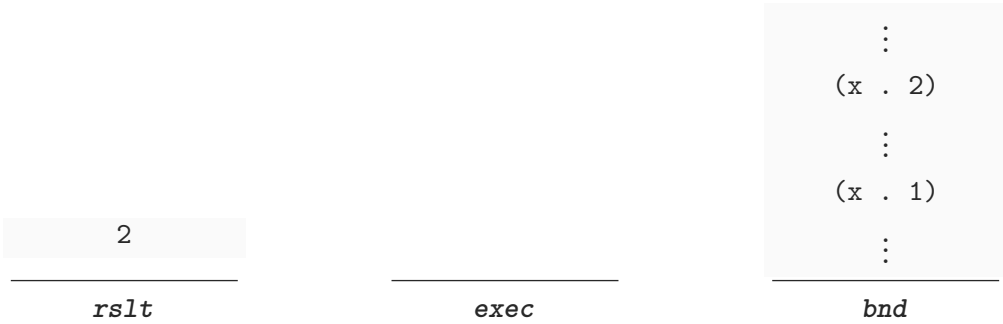


Získání hodnoty proměnné

`:val` dá hodnotu proměnné na zásobník *rslt*

Získání hodnoty proměnné

`:val` dá hodnotu proměnné na zásobník *rslt*



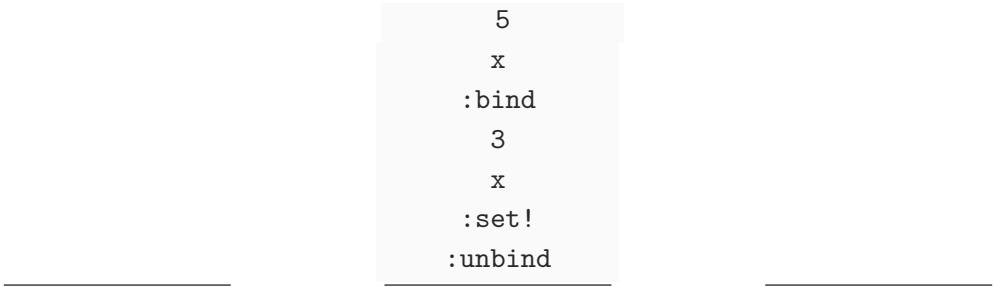
Další slova pro proměnné

`:bind`	vytvoří novou vazbu a dá ji na zásobník *bnd*
`:unbind`	odstraní vazbu z vrcholu zásobníku *bnd*
`:set!`	změní hodnotu vazby v zásobníku *bnd*

rslt	***exec***	***bnd***

Další slova pro proměnné

`:bind`	vytvoří novou vazbu a dá ji na zásobník *bnd*
`:unbind`	odstraní vazbu z vrcholu zásobníku *bnd*
`:set!`	změní hodnotu vazby v zásobníku *bnd*



Další slova pro proměnné

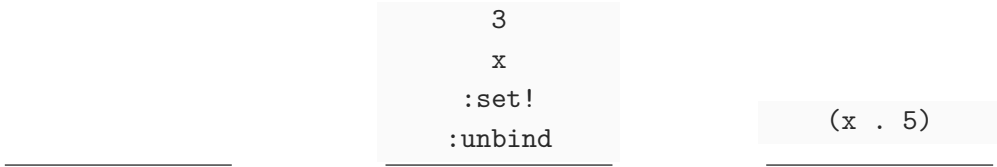
`:bind`	vytvoří novou vazbu a dá ji na zásobník *bnd*
`:unbind`	odstraní vazbu z vrcholu zásobníku *bnd*
`:set!`	změní hodnotu vazby v zásobníku *bnd*

Další slova pro proměnné

`:bind`	vytvoří novou vazbu a dá ji na zásobník *bnd*
`:unbind`	odstraní vazbu z vrcholu zásobníku *bnd*
`:set!`	změní hodnotu vazby v zásobníku *bnd*

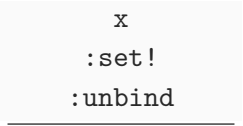
Další slova pro proměnné

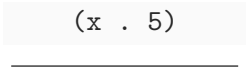
`:bind`	vytvoří novou vazbu a dá ji na zásobník *bnd*
`:unbind`	odstraní vazbu z vrcholu zásobníku *bnd*
`:set!`	změní hodnotu vazby v zásobníku *bnd*



Další slova pro proměnné

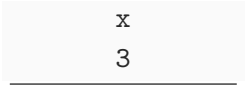
`:bind`	vytvoří novou vazbu a dá ji na zásobník *bnd*
`:unbind`	odstraní vazbu z vrcholu zásobníku *bnd*
`:set!`	změní hodnotu vazby v zásobníku *bnd*

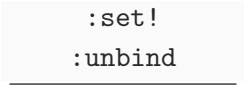


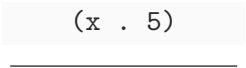


Další slova pro proměnné

`:bind`	vytvoří novou vazbu a dá ji na zásobník *bnd*
`:unbind`	odstraní vazbu z vrcholu zásobníku *bnd*
`:set!`	změní hodnotu vazby v zásobníku *bnd*

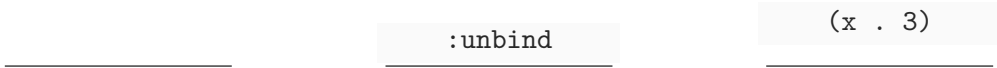






Další slova pro proměnné

`:bind`	vytvoří novou vazbu a dá ji na zásobník *bnd*
`:unbind`	odstraní vazbu z vrcholu zásobníku *bnd*
`:set!`	změní hodnotu vazby v zásobníku *bnd*



Další slova pro proměnné

`:bind`	vytvoří novou vazbu a dá ji na zásobník *bnd*
`:unbind`	odstraní vazbu z vrcholu zásobníku *bnd*
`:set!`	změní hodnotu vazby v zásobníku *bnd*

rslt	***exec***	***bnd***

Podprogramy

- reprezentujeme seznamem

Nový zásobník: *ret* (*návratový zásobník*)

Slova pro podprogramy:

`:clsub`	odstraní podprogram z vrcholu zásobníku *rslt* a nahradí jím programový zásobník jeho původní obsah uloží na návratový zásobník
`:ret`	vypustí z návratového zásobníku vrchol a nahradí jím programový zásobník

Příklad: `1 (2 :+ :ret) :clsub 3 :-`

Cykly

- Používají návratový zásobník.

Slova pro cykly:

Cykly

- Používají návratový zásobník.

Slova pro cykly:

`:lbl` uloží programový zásobník na vrchol návratového zásobníku

Cykly

- Používají návratový zásobník.

Slova pro cykly:

`:lbl`	uloží programový zásobník na vrchol návratového zásobníku
`:cjmp`	Vypustí vrchol datového zásobníku. Je-li vypuštěná hodnota *Pravda*, nastaví programový zásobník na vrchol návratového zásobníku. Jinak vypustí z návratového zásobníku vrchol.

Cykly

- Používají návratový zásobník.

Slova pro cykly:

`:lbl`	uloží programový zásobník na vrchol návratového zásobníku
`:cjmp`	Vypustí vrchol datového zásobníku. Je-li vypuštěná hodnota *Pravda*, nastaví programový zásobník na vrchol návratového zásobníku. Jinak vypustí z návratového zásobníku vrchol.

Příklad cyklu, který vytiskne čísla od pěti po jedničku:

```
5 :lbl :print 1 :- :dup 0 :> :cjmp
```

Obsah

Stav výpočtu

Stav výpočtu

Po každé instrukci je stav výpočtu procesu uložen na zásobnících:

1. *ret* (návratový zásobník)
2. *seek* (pomocný pro větvení)
3. *rslt* (datový zásobník)
4. *exec* (programový zásobník)
5. *bnd* (zásobník vazeb)

Stav výpočtu procesu reprezentujeme seznamem:

```
(ret seek rslt exec bnd)
```

Proces

Proces

Proces = stav výpočtu

Proces

Proces = stav výpočtu

Proces:

1. Buď je přímo vykonáván,
2. nebo je reprezentován seznamem.

Proces

Proces = stav výpočtu

Proces:

1. Buď je přímo vykonáván,
2. nebo je reprezentován seznamem.

- běžící proces – jeho kód je vykonáván
- připravený proces – jeho kód může být vykonáván

Proces

Proces = stav výpočtu

Proces:

1. Buď je přímo vykonáván,
2. nebo je reprezentován seznamem.

- běžící proces – jeho kód je vykonáván
- připravený proces – jeho kód může být vykonáván

Zavedeme frontu *rprs* (*ready processes*)

- obsahuje procesy připravené k vykonávání

Procesy

Při spuštění stroje vznikají dva procesy.

1. zahálčivý proces: `(() () () (:lbl t :cjmp) bnd)`
2. hlavní proces: `(() () () code bnd)`

Proces, který není hlavní, nazýváme vedlejší. Zahálčivý proces je tedy vedlejším procesem.

Procesy

Při spuštění stroje vznikají dva procesy.

1. zahálčivý proces: `(() () () (:lbl t :cjmp) bnd)`
2. hlavní proces: `(() () () code bnd)`

Proces, který není hlavní, nazýváme vedlejší. Zahálčivý proces je tedy vedlejším procesem.

Další procesy vznikají slovem `:prsub`

Procesy

Při spuštění stroje vznikají dva procesy.

1. zahálčivý proces: `(() () () (:lbl t :cjmp) bnd)`
2. hlavní proces: `(() () () code bnd)`

Proces, který není hlavní, nazýváme vedlejší. Zahálčivý proces je tedy vedlejším procesem.

Další procesy vznikají slovem `:prsub`

1. Odebere hodnotu *podprogram* ze zásobníku *rslt*.

Procesy

Při spuštění stroje vznikají dva procesy.

1. zahálčivý proces: `(() () () (:lbl t :cjmp) bnd)`
2. hlavní proces: `(() () () code bnd)`

Proces, který není hlavní, nazýváme vedlejší. Zahálčivý proces je tedy vedlejším procesem.

Další procesy vznikají slovem `:prsub`

1. Odebere hodnotu *podprogram* ze zásobníku *rslt*.
2. Odebere hodnotu *argument* ze zásobníku *rslt*.

Procesy

Při spuštění stroje vznikají dva procesy.

1. zahálčivý proces: `(() () () (:lbl t :cjmp) bnd)`
2. hlavní proces: `(() () () code bnd)`

Proces, který není hlavní, nazýváme vedlejší. Zahálčivý proces je tedy vedlejším procesem.

Další procesy vznikají slovem `:prsub`

1. Odebere hodnotu *podprogram* ze zásobníku *rslt*.
2. Odebere hodnotu *argument* ze zásobníku *rslt*.
3. Vytvoří proces `(() () (argument) podprogram bnd)`, kde *bnd* je aktuální zásobník vazeb.

Procesy

Při spuštění stroje vznikají dva procesy.

1. zahálčivý proces: `(() () () (:lbl t :cjmp) bnd)`
2. hlavní proces: `(() () () code bnd)`

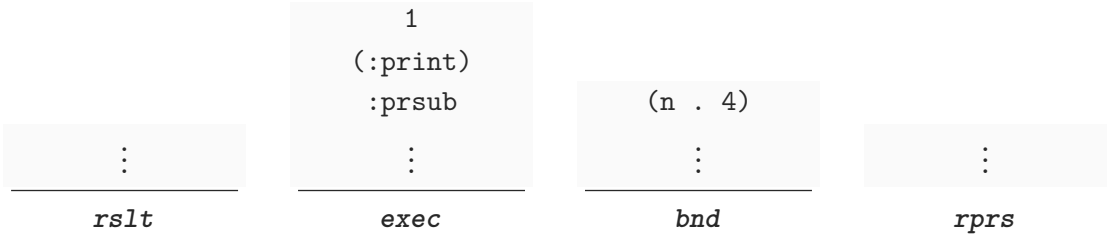
Proces, který není hlavní, nazýváme vedlejší. Zahálčivý proces je tedy vedlejším procesem.

Další procesy vznikají slovem `:prsub`

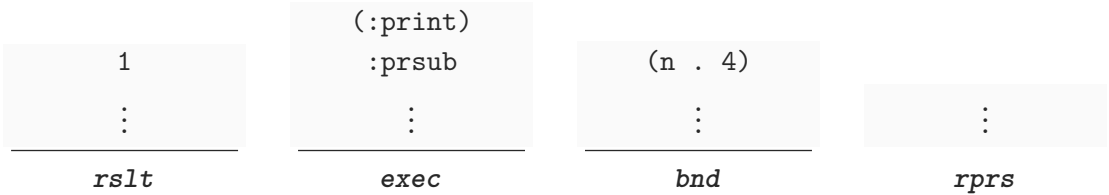
1. Odebere hodnotu *podprogram* ze zásobníku *rslt*.
2. Odebere hodnotu *argument* ze zásobníku *rslt*.
3. Vytvoří proces `(() () (argument) podprogram bnd)`, kde *bnd* je aktuální zásobník vazeb.
4. Přidá proces na konec fronty *rprs*.

Ukázka použití slova `:prsub`

Ukázka použití slova :prsub

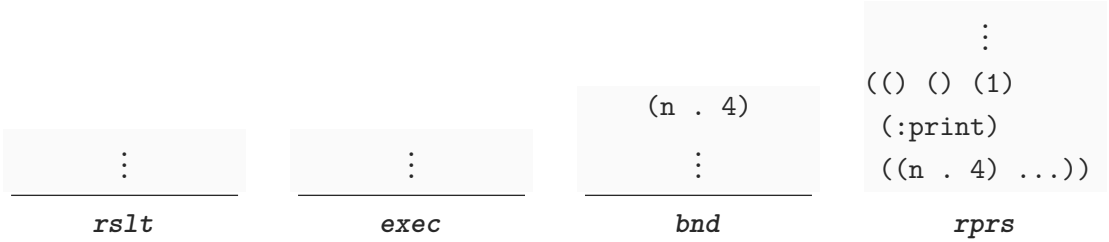


Ukázka použití slova :prsub



Ukázka použití slova `:prsub`

Ukázka použití slova :prsub



Sdílené vazby

Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.

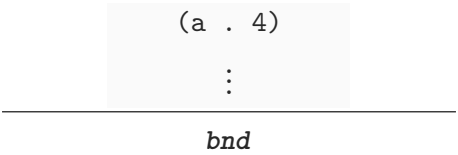
Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

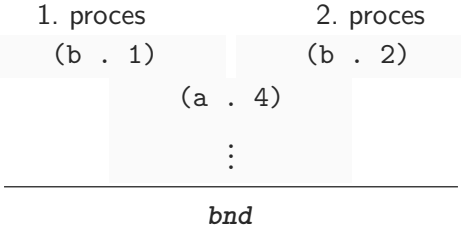
Například:



Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

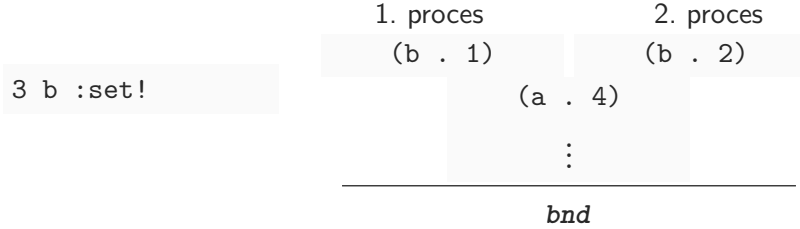
Například:



Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

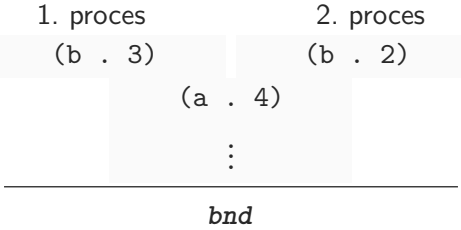
Například:



Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

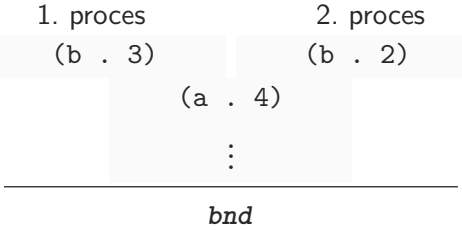
Například:



Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

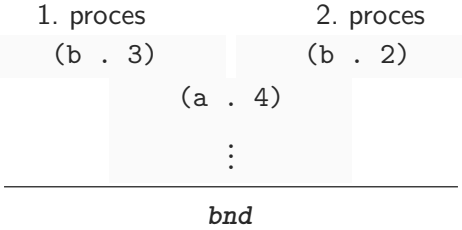
Například:



Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

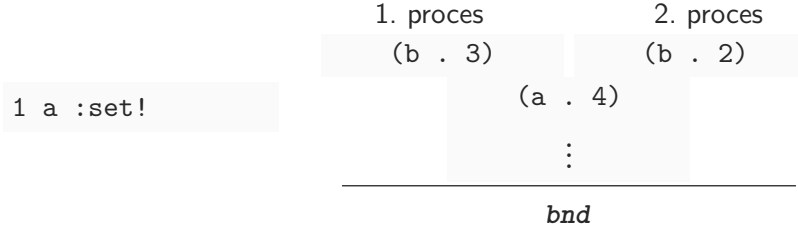
Například:



Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

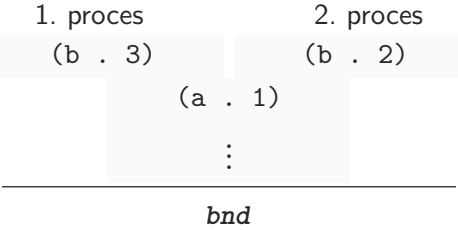
Například:



Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

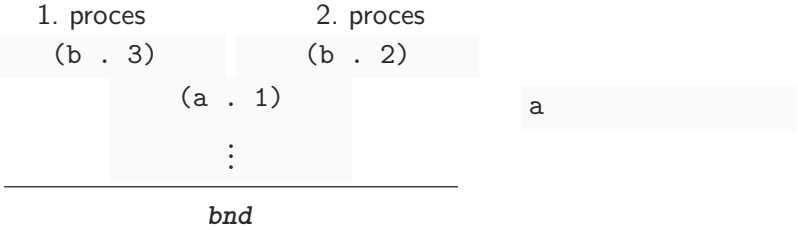
Například:



Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

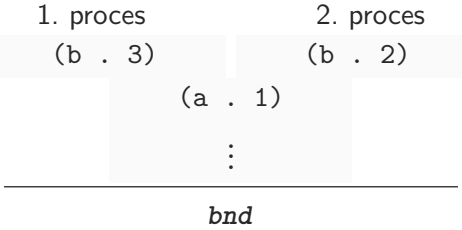
Například:



Sdílené vazby

- Nově vytvořený proces sdílí s procesem, který jej vytvořil, všechny vazby definované v době vzniku.
- Sdílené vazby umožňují komunikaci mezi procesy.

Například:



Obsah

Práce s procesy

Práce s procesy

Uložení běžícího procesu:

1. Vezme reprezentaci běžícího procesu
2. a vloží ji na konec fronty $rprs$.

Práce s procesy

Uložení běžícího procesu:

1. Vezme reprezentaci běžícího procesu
2. a vloží ji na konec fronty $rprs$.

Obnovení připraveného procesu:

1. Odebere proces ze začátku fronty $rprs$
2. a nastaví ho jako běžící.

Plánovač

Plánovač

Řídí přepínání mezi procesy.

Plánovač

Řídí přepínání mezi procesy.

Skládá se z:

- časovače a jeho obsluhy
- dispečera

Časovač

Časovač

Čas měříme v počtech kroků výpočtu zásobníkového stroje.

Časovač

Čas měříme v počtech kroků výpočtu zásobníkového stroje.

Časovač lze nastavit na určitou dobu.

Časovač

Čas měříme v počtech kroků výpočtu zásobníkového stroje.

Časovač lze nastavit na určitou dobu.

Po uplynulém čase se vyvolá obsluha časovače, která:

1. uloží běžící proces
2. vyvolá dispečera

Dispečer

Dispečer

Po spuštění:

1. Nastaví časovač.
2. Obnoví připravený proces.

Dispečer

Po spuštění:

1. Nastaví časovač.
2. Obnoví připravený proces.

Slovo `:quit` vyvolá dispečera.

Vedlejší proces by měl před ukončením zavolat dispečera.

Například:

```
1 (:print :quit) :prsub
```

Obsah

Běžící proces

Získání běžícího procesu:

```
(defun running-process ()
  (make-process *ret* *seek* *rslt* *exec* *bnd*))
```

Nastavení běžícího procesu:

```
(defun run-process (process)
  (setf *ret* (first process)
        *seek* (second process)
        *rslt* (third process)
        *exec* (fourth process)
        *bnd* (fifth process)))
```

Fronta připravených procesů

Fronta připravených procesů

Realizovaná zásobníky:

1. `*rprs-head*` (přední část fronty)
2. `*rprs-tail*` (zadní část fronty)

Přeskládání prvků:

```
(defun rearrange-ready-processes ()
  (when *rprs-tail*
    (push (pop *rprs-tail*) *rprs-head*)
    (rearrange-ready-processes)))
```

Připravené procesy

Přidání připraveného procesu:

```
(defun insert-ready-process (process)
  (push process *rprs-tail*))
```

Odebrání připraveného procesu:

```
(defun remove-ready-process ()
  (unless *rprs-head*
    (rearrange-ready-processes))
  (pop *rprs-head*))
```

Časovač

Doba do spuštění obsluhy: `*time*`

Obsluha časovače:

```
(defun timer-handler ()
  (insert-ready-process (running-process))
  (dispatcher))
```

Dispečer

Maximální doba časovače: `*max-time*`

Dispečer:

```
(defun dispatcher ()
  (setf *time* (1+ (random *max-time*)))
  (run-process (remove-ready-process)))
```

Slovo na zavolání dispečera:

```
(defprim :quit
  (dispatcher))
```

Původní execute

```
(defun execute (&rest code)
  (let ((*ret* '())
        (*seek* '())
        (*rslt* '())
        (*exec* code))
    (loop (when (null *exec*) (return))
          (exec-elem (pop *exec*)))
    (pop *rslt*)))
```

Nový execute

```
(defun execute (&rest code)
  (let ((*ret* '())
        (*seek* '())
        (*rslt* '())
        (*exec* code)
        (*time* 0)
        (*rprs-head* (list (make-idle-process)))
        (*rprs-tail* '()))
    (loop
       (cond
        ((null *exec*)
         (return))
        ((= *time* 0)
         (timer-handler))
        (t
         (exec-elem (pop *exec*))
         (decf *time*))))
    (pop *rslt*)))
```

Obsah

Problém

```
1 (:print :exit) :prsub
```

Problém

```
1 (:print :exit) :prsub
```

Hlavní proces nečeká na ukončení vedlejšího procesu.

Slovo `:await`

1. Vykoná podprogram na vrcholu datového zásobníku, který musí zanechat na vrcholu datového zásobníku jednu hodnotu.
2. Odstraní jedinou zanechanou hodnotu z datového zásobníku. Pokud je odstraněná hodnota *Nepravda*, pokračuje se prvním bodem.
3. Jinak se odstraní podprogram z datového zásobníku.

Slovo `:await`

1. Vykoná podprogram na vrcholu datového zásobníku, který musí zanechat na vrcholu datového zásobníku jednu hodnotu.
2. Odstraní jedinou zanechanou hodnotu z datového zásobníku. Pokud je odstraněná hodnota *Nepravda*, pokračuje se prvním bodem.
3. Jinak se odstraní podprogram z datového zásobníku.

Použití:

```
(end? :val :ret) :await
```

Slovo `:await`

1. Vykoná podprogram na vrcholu datového zásobníku, který musí zanechat na vrcholu datového zásobníku jednu hodnotu.
2. Odstraní jedinou zanechanou hodnotu z datového zásobníku. Pokud je odstraněná hodnota *Nepravda*, pokračuje se prvním bodem.
3. Jinak se odstraní podprogram z datového zásobníku.

Použití:

```
(end? :val :ret) :await
```

Implementace:

```
:def :await :lbl :dup :clsub :not :cjmp :drop
```

Řešení

```
nil end? :bind
1 (:print t end? :set! :quit) :prsub
(end? :val :ret) :await
:unbind
```