
KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Jan Laštovička
https://www.inf.upol.cz/lide/jan-lastovicka
jan.lastovicka@upol.cz
17. listopadu 12, 771 46 Olomouc

Paradigmata programování 4 ◇ poznámky k přednášce

1. Paralelní výpočty

verze z 12. února 2025

1 Úvod

Paralelní programování umožňuje souběžné vykonávání kódu. Tím můžeme docílit zvýšení výkonu aplikace nebo jejího lepšího rozdělení.

Procesor je část stroje umožňující vykonávat instrukce. **Proces** je část stroje uchovávající stav výpočtu včetně instrukcí, které mají být vykonány. Procesor může vykonávat instrukce nějakého procesu. Spuštění programu vede k vytvoření procesu uchovávajícího jeho instrukce. Program nazveme **paralelní**, pokud během jeho vykonávání mohou vzniknout další procesy. Procesy mohou být v systému vykonávány **souběžně** a to buď tak, že jsou vykonávány na více procesorech nebo je procesor střídavě vykonává. Procesy spolu komunikují pomocí **sdílené paměti**.

Stačí se omezit na stroje s jediným procesorem. Stroj s více procesory můžeme simulovat strojem s jedním procesorem tak, že každé souběžně vykonávané instrukce na různých procesorech libovolně uspořádáme a vykonáme na jediném procesoru.

Jemnou atomickou instrukcí rozumíme takovou instrukci, během jejíhož vykonávání není možné přerušit vykonávání procesu.

2 Zásobníkový stroj

Jako model stroje vykonávajícího paralelní program použijeme rozšíření zásobníkového stroje představeného v předmětu Paradigma programování 2 (PP2). Pro připomenutí zásobníkového stroje se podívejte do přiložených materiálů z předmětu PP2. Níže pouze shrneme zásobníkový stroj a přidáme několik novinek.

Zásobníkový stroj pracuje s prvky:

- slovo – lispový klíč
- hodnota – cokoliv jiného

Proměnné jsou symboly Lispu, které nejsou klíči. Tedy proměnné jsou hodnoty.

Přehled zásobníků:

1. `ret` (návratový zásobník) Zásobník zapamatovaných stavů programového zásobníku. Používaný při vykonávání podprogramů.

2. *seek* Pomocný zásobník pro větvení. Používají jej slova `:if`, `:else`, `:skip` a `:seek`.

3. *rslt* (datový zásobník) Obsahuje hodnoty: mezivýsledky i výsledek programu.

4. *exec* (programový zásobník) Obsahuje zdrojový kód. Na začátku program zadaný uživatelem.

5. *bnd* (zásobník vazeb) Obsahuje páry `(` *variable* `.` *value* `)`, kde *variable* je proměnná a *value* je její hodnota.

Jemnou atomickou instrukcí stroje je zpracování jednoho prvku na vrcholu programového zásobníku.

Následuje přehled základních slov.

Manipulace se zásobníkem

`:swap`	$(a\ b - b\ a)$
`:rot`	$(a\ b\ c - b\ c\ a)$
`:drop`	$(a\ -\ )$
`:dup`	$(a - a\ a)$
`:over`	$(a\ b - a\ b\ a)$

Aritmetické operace

`:+`	$(n_1\ n_2$ – součet $n_1 + n_2)$
`:-`	$(n_1\ n_2$ – rozdíl $n_1 - n_2)$
`:*`	$(n_1\ n_2$ – součin $n_1 \cdot n_2)$
`:/`	$(n_1\ n_2$ – podíl $n_1/n_2)$

Porovnávání

`:=`	$(n_1\ n_2$ – logická hodnota $n_1 = n_2)$
`:<`	$(n_1\ n_2$ – logická hodnota $n_1 < n_2)$
`:>`	$(n_1\ n_2$ – logická hodnota $n_1 > n_2)$
`:<=`	$(n_1\ n_2$ – logická hodnota $n_1 \leq n_2)$
`:>=`	$(n_1\ n_2$ – logická hodnota $n_1 \geq n_2)$

Logika

`:neg`	$(v$ – negace $v)$

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***

Slovo `:def` pro definici nového uživatelského slova:

1. Odebere vrchol programového zásobníku. Označme jej ***word***.
2. Definuje uživatelské slovo ***word***, jehož kód je obsah programového zásobníku.
3. Vymaže programový zásobník.
4. Na datový zásobník dá slovo ***word***.

Tisk

Slovo `:print` vytiskne vrchol datového zásobníku.

Proměnné

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

Slova pro proměnné:

`:val`	dá hodnotu proměnné na zásobník ***rslt***	(***var*** – ***val***)
`:bind`	vytvoří novou vazbu a dá ji na zásobník ***bnd***	(***val var*** –)
`:unbind`	odstraní vazbu z vrcholu zásobníku ***bnd***	(–)
`:set!`	změní hodnotu vazby v zásobníku ***bnd***	(–)

Podprogramy

Programy reprezentujeme seznamem. Vykonávání podprogramů používá *návratový zásobník* ***ret***.

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

Cykly

Vykonávání cyklu používá *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
```

REPL zásobníkového jazyka se spouští vyhodnocením výrazu `(stacks-repl)` v Lispu. Za výzvou `stacks >` REPL očekává kód zásobníkového jazyka, který se potvrdí dvojím odřádkováním. Poté se kód vykoná, vytiskne výsledek a znovu se čeká na vstup. Příklad použití:

```
CL-USER 1 > (stacks-repl)

stacks > 1 2 :+


3
stacks >
```

Původní zdrojové kódy k zásobníkovému stroji jsou v souborech `01_stacks.lisp` a `02_stacks.lisp`. Novinky se nalézají v souboru `03_stacks.lisp`.

3 Proces

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 ztotožníme se stavem výpočtu. Proces může být přímo vykonáván, nebo je reprezentován seznamem.

Pokud je proces vykonáván, říkáme, že **běží**. O procesu, který neběží, ale jehož kód může být vykonáván, říkáme, že je **připravený**. Zavedeme frontu *rprs* (*ready processes*) procesů připravených k vykonávání.

Při spuštění stroje vznikají dva procesy. Prvním je **hlavní proces** reprezentovaný seznamem `(() () () code bnd)`, kde *code* je program zadaný při spuštění stroje. Druhým je **zaháčivý proces** reprezentovaný seznamem `(() () () (:lbl t :cjmp) bnd)`. Při spuštění stroje bude hlavní proces běžící a zaháčivý proces bude připravený. Proces, který není hlavní, nazýváme **vedlejší**. Zaháčivý proces je tedy vedlejším procesem.

Další procesy vznikají slovem `:prsub`, které:

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*.

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

4 Plánovač

Plánovač je část stroje přidělující procesorům procesy, které mají vykonávat. Plánovač provádí dva následující úkony 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 se skládá z časovače a jeho obsluhy a dispečera.

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

Po spuštění:

1. Nastaví časovač.
2. Obnoví připravený proces.

Slovo `:quit` vyvolá dispečera. Přímé vyvolání dispečera způsobí ukončení procesu. Vedlejší proces by měl před ukončením zavolat dispečera. Například:

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

5 Implementace

Začneme prací s běžícím procesem. Funkce `make-process` vytvoří reprezentaci procesu. 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)))
```

Frontu připravených procesů *rprs* realizujeme pomocí dvou zásobníků:

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

Funkce, která přeskládá procesy ze zadní části fronty do přední:

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

Dále napíšeme přidání a odebrání procesu z fronty *rprs*. 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*))
```

Čas do spuštění obsluhy bude uložen v globální proměnné `*time*`. Obsluha časovače:

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

Doba, do které se jistě časovač spustí, bude uložena v proměnné `*max-time*`. Dispečer:

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

Slovo na zavolání dispečera:

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

Připomeňme si původní funkci `execute` zodpovědnou za vykonávání kódu zásobníkového stroje:

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

Nahradíme ji verzí s přidanou frontou připravených procesů a časovačem:

```
(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*)))
```

6 Aktivní čekání

Následující program jedničku nevytiskne, protože po vyprázdnění programového zásobníku se stroj vypne.

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

Obecně si přejeme zařídit, aby proces čekal na skončení procesů, které vytvořil. Přidáme slovo `:await`, které cyklicky bude kontrolovat splnění zadané podmínky a skončí v případě, že podmínka je splněna. Podmínkou myslíme podprogram, který po vykonání zanechá na datovém zásobníku logickou hodnotu.

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
```

Čekání na skončení vykonávání podprogramu zařídíme tak, že vytvoříme proměnnou `end?`, která bude uchovávat logickou hodnotu rozhodující, zda vytvořený proces skončil. Vytvořený proces před skončením nastaví hodnotu proměnné `end?` na *Pravdu*. Vytvářející proces bude čekat, než hodnota proměnné `end?` bude *Pravda*. Řešení ukázkového příkladu:

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

Otázky a úkoly na cvičení

1. Co způsobí vykonání následujícího programu v zásobníkovém jazyce?

```
:quit
```

2. Co mohou vytisknout následující programy?

   (a) `1 (:print) :prsub 2 :print`

   (b) `1 (:print :quit) :prsub 2 :print`

   (c) `1 (:print :quit) :prsub 2 :print 3`

3. Napište paralelní program, který vytvoří dva vedlejší procesy, kde první proces vytiskne jedničku a druhý dvojku. Hlavní proces musí čekat, než vedlejší procesy skončí.

4. Vytvořte dva vedlejší procesy. První bude tisknout jedničku a trojku a druhý dvojku a čtyřku. Zaručte, že trojka bude vždy vytištěna po dvojce.

5. Vytvořte dva vedlejší procesy. První bude tisknout jedničku a trojku a druhý dvojku a čtyřku. Zaručte, aby se nejprve v libovolném pořadí vytiskla čísla jedna a dva a až poté opět v libovolném pořadí čísla tři a čtyři.

6. Jaké možné hodnoty může vrátit následující program?

```
0 val1 :bind
0 val2 :bind
nil (1 val1 :set!
     :quit) :prsub
nil (2 val2 :set!
     :quit) :prsub
(val1 :val 0 :> :if val2 :val 0 :> :else t :then :ret) :await
val1 :val val2 :val :+
:unbind
:unbind
```

7. Jaké možné hodnoty může vrátit následující program?

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

```
val :val
:unbind
:unbind
```

8. Dokázali byste zařídit, aby předchozí program vždy vrátil hodnotu dva?

9. Vytvořte dva vedlejší procesy, které na vstup dostanou libovolné hodnoty. Procesy si hodnoty prohodí a každý proces vytiskne obdrženou hodnotu.

10. Vytvořte dva vedlejší procesy. První proces dostane na vstupu číslo, přičte k němu jedničku a předá jej druhému procesu, který obdržené číslo zdvojnásobí a předá jej zpět prvnímu procesu, který obdržené číslo vytiskne. Například pro číslo pět program vytiskne číslo dvanáct.

11. Napište program, který paralelně spočítá faktoriál zadaného čísla.