

# Paralelní systém, kritická sekce

## Paralelní a distribuované systémy, přednáška 2

Tomáš Urbanec

Katedra informatiky PŘF UPOL

2.10.2024

# Paralelní systém

# Paralelní systém

- Rozehřátí opakováním
- Co to je paralelní systém/program?
- Co jste si odnesli z cvičení?

## (De)motivační příklady

- Chyba souběhu: prokládání operací při převodu peněz.
- Uváznutí (deadlock): problém 4 aut.
- Uváznutí (livelock): řešení problému 4 aut?
- Vyhladovění: více aut.

# Základní pojmy

## Co s tím?

Viděli jsme různé problémy:

- Chyba souběhu (Race condition, data race)
- Deadlock, livelock, vyhladovění, . . .

Jak vypadá naše situace?

- Paralelní program - konečná množina sekvenčních procesů.
- Každý proces sekvenčně (!) vykonává atomické operace.
  - atomické = dále nedělitelné
- Tím přechází mezi stavy
  - hodnoty proměnných
  - aktuální instrukce
- Operace procesů se mohou libovolně proložit
  - scénář paralelního výpočtu.

# Synchronizace

## Základní pojmy

Co tedy můžeme dělat?

→ Synchronizovat, tj. omezovat možné scénáře.

Jak na to?

- Atomické operace
  - základní instrukce
  - složené atomické operace
  - záleží na jemnosti (dále)
- Základní problém: kritická sekce
- Synchronizační primitiva a nástroje:
  - semaforey a mutexy,
  - monitory,
  - zámky,
  - bariéry,
  - ...
- Klasické problémy: modelové situace se známým řešením.

# Vlastnosti programu

## Základní pojmy

- Jak nahradit krokování/debugování sekvenčních programů?
- Více scénářů paralelního programu → procházet všechny?
- Dost těžko... (ledaže [1])
- Zachránit nás může (modální, temporální) logika.
- Typy vlastností paralelního programu:
  - živost (liveness)
    - tvrzení platné pro alespoň jeden stav výpočtu.
    - „dobrá věc se někdy stane“
    - např. Databáze je v konzistentním stavu.
  - bezpečnost (safety)
    - tvrzení platné pro všechny stavy výpočtu.
    - „špatná věc se nikdy nestane“
    - např. Nenastane deadlock.
- Systém je korektní, pokud každý jeho scénář splňuje každou požadovanou vlastnost.
- Další klíčová slova: prekondice, postkondice, Hoareho logika

# Férovost

## Základní pojmy

- Vlastnost běhového prostředí (plánovače, architektury).
- Korektnost vyžaduje férovost (příklad)
- Slabá férovost - akce, která vždy může nastat, někdy nastane.
- Silná férovost - akce, která někdy může nastat, někdy nastane.



# Příklad

## Férovost

---

### Stop flag

---

#### Sdílená paměť

$i \leftarrow 1$   
pokracuj  $\leftarrow$  true

---

**A**

$A_1$ : while pokracuj:  
 $A_2$ :  $i \leftarrow i + 1$   
 $A_3$ : print('konec')

**B**

$B_1$ : pokracuj  $\leftarrow$  false

- 
- Kdy je splněna vlastnost „Proces A skončí.“?
  - Co kdyby B vykonávalo svou instrukci podmíněně?

## Více k atomicitě

- nedělitelnost
  - = nepřerušitelnost
  - = nelze proložit operacemi jiného procesu
- Co už je atomické? Definice záleží na hrubosti systému.
- Na definici atomicity závisí korektnost algoritmů (příklad)

### Atomická proměnná?

- lze atomicky upravit (přečíst původní a zapsat novou hodnotu)
- speciální typy a/nebo instrukce
- u obvyklých (neatomických) proměnných nutná synchronizace

S tím souvisí i koncept volatelní proměnné, která má vždy poslední přiřazenou hodnotu (tj. změna nejen v cache, registru, ...).

## Kritická reference

- Výskyt proměnné je kritická reference pokud:
  - do ní zapisuje jeden proces a čte ji jiný proces
- Příkaz může obsahovat více kritických referencí (příklady).
- LCF (Limited critical reference)
  - Podmínka, kterou proces splňuje, pokud každý jeho příkaz obsahuje nejvýše jednu kritickou referenci.
  - Takový program se chová stejně, jako kdybychom každý příkaz rozepsali na atomické akce.
  - Neřeší problémy, jen umožňuje uvažovat nad většími bloky.

## Příkaz await

- Forma: `await podmínka`
- Čekání na splnění podmínky.
- Reálně se používá pasivní čekání (přerušení).
- Při podmínce splňující LCR stačí `while (not podmínka)`:
  - aktivní čekání
    - jednoduché, ale bere výkon
    - pro pochopení stačí

## Problém kritické sekce

- Vyřešen E. W. Dijkstrou v roce 1965 [2] (znám déle)
- „ $n$  procesů ve smyčce vykonává posloupnost akcí rozdělenou na dvě části: kritickou a nekritickou sekci.“
- Požadavky na řešení (korektnost):
  1. Vzájemné vyloučení - kritické sekce dvou procesů se nesmí proložit.
  2. Absence uváznutí - pokud se více procesů snaží najednou vstoupit do KS, jeden z nich uspěje.
  3. Absence vyhladovění - pokud se proces snaží vstoupit do KS, někdy uspěje.
- Synchronizace
  - = zajištění korektnosti
  - uvažujeme vstupní a výstupní protokoly okolo KS
  - Předpokládáme, že:
    - Kritická a nekritická sekce nesdílejí data
    - KS vždy skončí
    - Nekritická sekce skončit nemusí

# První pokus

---

## Pokus o KS 1

---

Sdílená paměť

turn  $\leftarrow$  1

---

**A**

opakuj:

$A_1$ : nekritická sekce

$A_2$ : await turn = 1

$A_3$ : kritická sekce

$A_4$ : turn  $\leftarrow$  2

**B**

opakuj:

$B_1$ : nekritická sekce

$B_2$ : await turn = 2

$B_3$ : kritická sekce

$B_4$ : turn  $\leftarrow$  1

---

# Analýza

## První pokus

- turn = právo na vstup do KS
  - každý proces čeká na svůj turn
  - ✓ nemáme deadlock - vždy někdo může do KS
  - ✗ můžeme vyhladovět - jak?
- jedna proměnná na řízení nestačí

## Druhý pokus

---

### Pokus o KS 2

---

#### Sdílená paměť

isA, isB  $\leftarrow$  false

---

**A**

opakuj:

$A_1$ : nekritická sekce

$A_2$ : await not(isB)

$A_3$ : isA  $\leftarrow$  true

$A_4$ : kritická sekce

$A_5$ : isA  $\leftarrow$  false

---

**B**

opakuj:

$B_1$ : nekritická sekce

$B_2$ : await not(isA)

$B_3$ : isB  $\leftarrow$  true

$B_4$ : kritická sekce

$B_5$ : isB  $\leftarrow$  false

---



# Analýza

## Druhý pokus

- $isA, isB$  = oznámení, že jsem v KS.
  - proces čeká na dokončení KS druhého procesu
  - ✓ nemáme deadlock - vždy někdo může do KS
  - ✓ nemůžeme vyhladovět - proces mimo KS nedrží výhradní právo na vstup
  - ✗ přesto špatně - proč?
- nelze povolovat/zakazovat vstup ostatním až v KS

## Třetí pokus

---

### Pokus o KS 3

---

#### Sdílená paměť

wantA, wantB  $\leftarrow$  false

---

**A**

opakuj:

$A_1$ : nekritická sekce

$A_2$ : wantA  $\leftarrow$  true

$A_3$ : await not(wantB)

$A_4$ : kritická sekce

$A_5$ : wantA  $\leftarrow$  false

---

**B**

opakuj:

$B_1$ : nekritická sekce

$B_2$ : wantB  $\leftarrow$  true

$B_3$ : await not(wantA)

$B_4$ : kritická sekce

$B_5$ : wantB  $\leftarrow$  false

---

# Analýza

## Třetí pokus

- wantA, wantB = oznámení, že chci do KS
  - proces čeká, až druhý opustí KS
  - ✗ deadlock - jak?
- nelze si vynucovat vstup KS

## Čtvrtý pokus

---

### Pokus o KS 4

---

#### Sdílená paměť

wantA, wantB  $\leftarrow$  false

---

**A**

opakuj:

$A_1$ : nekritická sekce

$A_2$ : wantA  $\leftarrow$  true

$A_3$ : while wantB

$A_4$ :     wantA  $\leftarrow$  false

$A_5$ :     wantA  $\leftarrow$  true

$A_6$ : kritická sekce

$A_7$ : wantA  $\leftarrow$  false

**B**

opakuj:

$B_1$ : nekritická sekce

$B_2$ : wantB  $\leftarrow$  true

$B_3$ : while wantA

$B_4$ :     wantB  $\leftarrow$  false

$B_5$ :     wantB  $\leftarrow$  true

$B_6$ : kritická sekce

$B_7$ : wantB  $\leftarrow$  false

---

# Analýza

## Čtvrtý pokus

- wantA, wantB = oznámení, že chci do KS
  - proces se vzdá vstupu do KS, když chce i druhý
  - instrukce  $A_3, A_4, A_5, B_3, B_4, B_5$  se mohou prokládat
  - ✗ livelock - jak?
- reakce na možný deadlock nesmí být symetrická

# Dekkerův algoritmus

---

## Dekkerův algoritmus

---

### Sdílená paměť

wantA, wantB  $\leftarrow$  false  
turn  $\leftarrow$  1

---

**A**

opakuj:

A<sub>1</sub>: nekritická sekce

A<sub>2</sub>: wantA  $\leftarrow$  true

A<sub>3</sub>: while wantB

A<sub>4</sub>:     if turn = 2

A<sub>5</sub>:         wantA  $\leftarrow$  false

A<sub>6</sub>:         await turn = 1

A<sub>7</sub>:         wantA  $\leftarrow$  true

A<sub>8</sub>: kritická sekce

A<sub>9</sub>: turn  $\leftarrow$  2

A<sub>10</sub>: wantA  $\leftarrow$  false

**B**

opakuj:

B<sub>1</sub>: nekritická sekce

B<sub>2</sub>: wantB  $\leftarrow$  true

B<sub>3</sub>: while wantA

B<sub>4</sub>:     if turn = 1

B<sub>5</sub>:         wantB  $\leftarrow$  false

B<sub>6</sub>:         await turn = 2

B<sub>7</sub>:         wantB  $\leftarrow$  true

B<sub>8</sub>: kritická sekce

B<sub>9</sub>: turn  $\leftarrow$  1

B<sub>10</sub>: wantB  $\leftarrow$  false

---

# Analýza

## Dekkerův algoritmus

- kombinujeme myšlenky z pokusu 1 a 4
  - hlídáme právo na vstup (turn)
  - žádáme o vstup (wantA, wantB) s možností vzdání se
- turn = právo na trvání na žádosti o vstup

## Petersonův algoritmus [3]

---

### Petersonův algoritmus

---

#### Sdílená paměť

wantA, wantB  $\leftarrow$  false  
last  $\leftarrow$  1

---

#### **A**

opakuj:

$A_1$ : nekritická sekce

$A_2$ : wantA  $\leftarrow$  true

$A_3$ : last  $\leftarrow$  1

$A_4$ : await not(wantB) or last = 2

$A_5$ : kritická sekce

$A_6$ : wantA  $\leftarrow$  false

---

#### **B**

opakuj:

$B_1$ : nekritická sekce

$B_2$ : wantB  $\leftarrow$  true

$B_3$ : last  $\leftarrow$  2

$B_4$ : await not(wantA) or last = 1

$B_5$ : kritická sekce

$B_6$ : wantB  $\leftarrow$  false

---



# Analýza

## Petersonův algoritmus

- Zvaný též „Tie-breaker“.
- Vychází z Dekkerova algoritmu.
  - místo cyklu s await máme await se složenou podmínkou
  - podmínka nesplňuje LCR
  - ... ale korektnost je ok i při neatomickém vyhodnocení

# Bakery algoritmus

---

## Základní bakery algoritmus

---

### Sdílená paměť

$nA, nB \leftarrow 0$

---

**A**

opakuj:

$A_1$ : nekritická sekce

$A_2$ :  $nA \leftarrow nB + 1$

$A_3$ : await  $nB = 0$  or  $nA \leq nB$

$A_4$ : kritická sekce

$A_5$ :  $nA \leftarrow 0$

**B**

opakuj:

$B_1$ : nekritická sekce

$B_2$ :  $nB \leftarrow nA + 1$

$B_3$ : await  $nA = 0$  or  $nB \leq nA$

$B_4$ : kritická sekce

$B_5$ :  $nB \leftarrow 0$

---

# Bakery algoritmus

---

## Základní bakery algoritmus ( $n$ procesů)

---

Sdílená paměť

pole[1, ..., n]  $\leftarrow$  [0, ..., 0]

---

**Proces  $i$**

opakuje:

$A_1$ : nekritická sekce

$A_2$ : pole[i]  $\leftarrow$  max(pole) + 1

$A_3$ : pro všechna  $j$  různá od  $i$ :

await(pole[j]=0 or pole[i] < pole[j] or (pole[i] = pole[j] and  $i < j$ ))

$A_4$ : kritická sekce

$A_5$ : pole[i]  $\leftarrow$  0

---

# Analýza

## Základní bakery algoritmus

- $nA$ ,  $nB$  a  $\text{pole}[i]$  představují pořadové lístky
- u víceprocesové varianty je nutné zjistit největší lístek
  - pomalé
  - nutnost atomické operace  $\text{max}$

## Lamportův bakery algoritmus [4]

---

### Lamportův bakery algoritmus ( $n$ procesů)

---

#### Sdílená paměť

pole[1, ..., n]  $\leftarrow$  [0, ..., 0]

vstup[1, ..., n]  $\leftarrow$  [false, ..., false]

---

#### Proces $i$

opakuj:

$A_1$ : nekritická sekce

$A_2$ : vstup[ $i$ ]  $\leftarrow$  true

$A_3$ : pole[ $i$ ]  $\leftarrow$  max(pole) + 1

$A_4$ : vstup[ $i$ ]  $\leftarrow$  false

$A_5$ : pro všechna  $j$  různá od  $i$ :

    await(not(vstup[ $j$ ]))

    await(pole[ $j$ ]=0 or pole[ $i$ ] < pole[ $j$ ] or (pole[ $i$ ] = pole[ $j$ ] and  $i < j$ ))

$A_6$ : kritická sekce

$A_7$ : pole[ $i$ ]  $\leftarrow$  0

---

# Analýza

## Lamportův bakery algoritmus

- Vstup zajišťuje, že budeme vždy vědět o případných kolizích u spočítaných maxim
  - V tom případě se rozhodneme podle indexu procesu
- Lze skutečně použít
  - třeba pro implementaci vzájemného vyloučení tam, kde hw nepodporuje synchronizační primitiva (příště)

# Reálný svět

- Složené atomické operace
- Vyšší synchronizační primitiva
- Další algoritmy
- Příště



Facebook. *Hermit*. Dostupné z

<https://github.com/facebookexperimental/hermit>.

Citováno 1.10.2024.



E. W. Dijkstra. 1965. *Solution of a problem in concurrent programming control*. Commun. ACM 8, 9, 569.



G. L. Peterson. 1981 *Myths about the mutual exclusion problem*. Information Processing Letters, 12, 3, 115-116.



L. Lamport. 1974. *A new solution of Dijkstra's concurrent programming problem*. Commun. ACM 17, 8, 453–455.



# Changelog

- 11.10.2024
  - styl úvodního slidu,
  - formátování pseudokódů,
  - nastavení beameru,
  - changelog,
  - překlepy.