Paralelní programování

přednášky

Jan Outrata

únor–květen 2011

Další řešení kritické sekce

Pekařský (bakery) algoritmus

- Lamport, 1974

= proces, který chce vstoupit do krit. sekce, si musí vzít **číselný lístek (ticket)**, jehož číslo je větší než čísla lístků ostatních čekající procesů, a čeká, až číslo jeho lístku je nejmenší z čekajících

- analogie s „automatem na lístky“ pro obsluhu na principu „první přijde, první obsloužen“

Další řešení kritické sekce

Pekařský (bakery) algoritmus

int $nA \leftarrow 0$, $nB \leftarrow 0$

A		B	
	loop forever		loop forever
1:	nekritická sekce	1:	nekritická sekce
2:	$nA \leftarrow nB + 1$	2:	$nB \leftarrow nA + 1$
3:	await $nB = 0$ or $nA \leq nB$	3:	await $nA = 0$ or $nB < nA$
4:	kritická sekce	4:	kritická sekce
5:	$nA \leftarrow 0$	5:	$nB \leftarrow 0$

Obrázek: Pekařský (bakery) algoritmus řešení problému krit. sekce pro 2 procesy

- proměnné nA a nB obsahují čísla lístků procesů
- pokud jsou čísla lístků procesů, které chtějí vstoupit do krit. sekce, stejná (*scénář?*), vybere se jeden lib. z nich (A)
- předpokládá atomické akce A_2 a B_2!, jinak neplatí vzájemné vyloučení: *scénář?*, *řešení?*

Další řešení kritické sekce

Pekařský (bakery) algoritmus

int $number[N] \leftarrow [0, \ldots, 0]$

```
      loop forever
1:        nekritická sekce
2:        number[i] ← 1 + max(number)
3:        for all other processes j
4:            await (number[j] = 0 or (number[i] ≪ number[j])
5:        kritická sekce
6:        number[i] ← 0
```

Obrázek: Pekařský (bakery) algoritmus řešení problému kritické sekce pro N procesů

for all other processes j:
for j from 1 to N if $j \neq i$

$number[i] \ll number[j]$:
$(number[i] < number[j])$ or $((number[i] = number[j])$ and $(i < j))$

Další řešení kritické sekce

Pekařský (bakery) algoritmus

- nevýhody: čísla lístků mohou růst donekonečna (*scénář?*), proces musí testovat čísla lístků všech ostatních procesů, i když žádný z nich nechce vstoupit do krit. sekce – **neefektivní**, pokud procesy nesoupeří o vstup do krit. sekce a procesů je více
- existuje (originální) řešení korektní i při neatomickém čtení a zápisu do glob. proměnných

Další řešení kritické sekce

„Rychlé“ algoritmy

= při nesoupeření o vstup do krit. sekce proces vykoná vstupní i výstupní protokol sestávající z pevného (a malého) počtu **ne-await akcí**

– proces testuje stav jiných procesů a příp. čeká jen v případě soupeření o vstup do krit. sekce

– první Lamport

Další řešení kritické sekce

„Rychlý“ algoritmus (návrh)

int $gate1 \leftarrow 0$, $gate2 \leftarrow 0$

A		B	
	loop forever		loop forever
1:	nekritická sekce	1:	nekritická sekce
2:	$gate1 \leftarrow idA$	2:	$gate1 \leftarrow idB$
3:	if $gate2 \neq 0$ goto A_2	3:	if $gate2 \neq 0$ goto B_2
4:	$gate2 \leftarrow idA$	4:	$gate2 \leftarrow idB$
5:	if $gate1 \neq idA$	5:	if $gate1 \neq idB$
6:	if $gate2 \neq idA$ goto A_2	6:	if $gate2 \neq idB$ goto B_2
7:	kritická sekce	7:	kritická sekce
8:	$gate2 \leftarrow 0$	8:	$gate2 \leftarrow 0$

Obrázek: „Rychlý“ algoritmus řešení problému krit. sekce pro 2 procesy (návrh)

- proměnné idA a idB jsou číselné nenulové identifikátory procesů A a B
- při absenci soupeření v vstup do krit. sekce jen 3 přiřazení konstanty do glob. proměnné a 2 srovnání glob. proměnné s konstantou

Další řešení kritické sekce

„Rychlý“ algoritmus

Fig. 5.1 – 5.3

Obrázek: Ilustrace rychlého algoritmu

- nesplňuje vzájemné vyloučení: *scénář?*

Další řešení kritické sekce

„Rychlý“ algoritmus

Fig. 5.1 – 5.3

Obrázek: Ilustrace rychlého algoritmu

- nesplňuje vzájemné vyloučení: *scénář?*

Další řešení kritické sekce

„Rychlý“ algoritmus

int $gate1 \leftarrow 0$, $gate2 \leftarrow 0$
bool $wantA \leftarrow false$, $wantB \leftarrow false$

	A		B
	loop forever		loop forever
1:	nekritická sekce	1:	nekritická sekce
2:	$gate1 \leftarrow idA$	2:	$gate1 \leftarrow idB$
3:	$wantA \leftarrow true$	3:	$wantB \leftarrow true$
4:	if $gate2 \neq 0$	4:	if $gate2 \neq 0$
5:	$wantA \leftarrow false$	5:	$wantB \leftarrow false$
6:	goto A_2	6:	goto B_2
7:	$gate2 \leftarrow idA$	7:	$gate2 \leftarrow idB$
8:	if $gate1 \neq idA$	8:	if $gate1 \neq idB$
9:	$wantA \leftarrow false$	9:	$wantB \leftarrow false$
10:	await $wantB = false$	10:	await $wantA = false$
11:	if $gate2 \neq idA$ goto A_2	11:	if $gate2 \neq idB$ goto B_2
12:	else $wantA \leftarrow true$	12:	else $wantB \leftarrow true$
13:	kritická sekce	13:	kritická sekce
14:	$gate2 \leftarrow 0$	14:	$gate2 \leftarrow 0$
15:	$wantA \leftarrow false$	15:	$wantB \leftarrow false$

Další řešení kritické sekce

„Rychlý“ algoritmus pro lib. počet procesů

- pole proměnných $want[N]$ a místo await:

$$\boxed{\begin{array}{l}\text{for all other processes } j \\ \quad \text{await } want[j] = false\end{array}}$$

- nesplňuje absenci vyhladovění procesu, ale jen při velkém soupeření o vstup do krit. sekce

Simulace obecného semaforu

- Barz
- pomocí **dvou binárních semaforů (mutexů) a jedné celočíselné proměnné**

- proměnná *count* obsahuje hodnotu semaforu, mutex *gate* zajišťuje čekání procesů, mutex *M* zajišťuje vzájemné vyloučení při přístupu k proměnné *count*
- inicializace na hodnotu $k \geq 0$: $M \leftarrow 1, gate \leftarrow \min(1, k), count \leftarrow k$

wait

```
1:  wait(gate)
2:  wait(M)
3:  count ← count − 1
4:  if count > 0
5:      signal(gate)
6:  signal(M)
```

signal

```
1:  wait(M)
2:  count ← count + 1
3:  if count = 1
4:      signal(gate)
5:  signal(M)
```

Obrázek: Simulace operace *wait* a *signal*