Paralelní programování

přednášky

Jan Outrata

únor–duben 2011

Semafory

Await

- synchronizace používající await běží na **„železe“** = využívají jen (atomické) instrukce poskytované strojem
- instrukce jsou příliš nízkoúrovňové

Semafor

- Dijkstra, 1968

= klasické synchronizační primitivum implementované v OS

- na vyšší úrovni abstrakce než instrukce
- dvě **atomické operace**: čekání na semaforu, signalizace semaforu
- motivace – semafor na silnici (červená, zelená)

Aktivní implementace semaforu (busy-wait semaphore)

- čekání pomocí **čekací smyčky**

= nezáporné celé **číslo** V

- inicializace na hodnotu $k \geq 0$ – **obecný**, $0 \leq k \leq 1$ – **binární**
- **atomické operace**:

wait(S)	
1:	await $S.V > 0$
2:	$S.V \leftarrow S.V - 1$

Obrázek: Operace *wait*(S) na semaforu S

signal(S)	
1:	$S.V \leftarrow S.V + 1$

signal(S)	
1:	if $S.V = 0$
2:	$S.V \leftarrow 1$

Obrázek: Operace *signal*(S) na obecném a binárním semaforu S

Aktivní implementace semaforu (busy-wait semaphore)

- operace **wait**(**S**) původně (Dijkstra) označovaná **P**(**S**), operace **signal**(**S**) původně označovaná **V**(**S**)
- při operaci wait může proces (aktivně) **čekat na semaforu**
- při operaci signal může **libovolný jeden** proces čekající na semaforu pokračovat (je splněna podmínka await)
- binární semafor je také nazývaný **mutex** – použití pro vzájemné vyloučení (mutual exclusion)
- použití **vhodné při multiprocesoringu** (čekající proces na samostatném procesoru a další volné) **a při menším soupeření procesů o provedení operace wait** (např. při krátkém čekání, ušetření relativně náročného plánování procesů)

Pasivní implementace semaforu

- čekání pomocí **změny stavu procesu** pomocí plánovače procesů v OS na **blokovaný (pozastavený) stav** – **kdykoliv!**:

Obr. 108

Obrázek: Změny stavů procesu

  - při blokovaném stavu je následující akce procesu **zakázaná**, dokud jej jiný proces neodblokuje
  - jen **1** (vybraný) **proces** je v každém čase **běžící**, ostatní připravené – viz jen 1 (vybraná) akce z následujících akcí procesů

= dvojice (V, L), kde V je nezáporné celé **číslo a** L je **množina procesů**

- inicializace na hodnotu $(k, \emptyset)$
- **atomické operace** (A je proces, který operaci vykonává): DALŠÍ SLAJD
- při operaci wait může být proces **blokovaný na semaforu**
- při operaci signal může být **libovolný jeden** proces blokovaný na semaforu odblokován

Pasivní implementace semaforu

wait(S)	
	proces A
1:	if $S.V > 0$
2:	$S.V \leftarrow S.V - 1$
3:	else
4:	$S.L \leftarrow S.L \cup A$
5:	A.state $\rightarrow$ blocked

Obrázek: Operace *wait*(S) na semaforu S

signal(S)	
	proces B
1:	if $S.L = \emptyset$
2:	$S.V \leftarrow S.V + 1$
3:	else
4:	B = libovolný prvek S.L
5:	$B.L \leftarrow B.L \setminus \{B\}$
6:	B.state = ready

signal(S)	
	proces B
1:	if $S.V = 0$
2:	if $S.L = \emptyset$
3:	$S.V \leftarrow 1$
4:	else
5:	B = libovolný prvek S.L
6:	$B.L \leftarrow B.L \setminus \{B\}$
7:	B.state = ready

Obrázek: Operace *signal*(S) na obecném a binárním semaforu S

Problém kritické sekce

– vstupní protokol = čekání na semaforu, výstupní protokol = signalizace semaforu

Pro 2 procesy

Kritická sekce	
binary semaphore $S \leftarrow 1$	
A	B
loop forever	loop forever
1: nekritická sekce	1: nekritická sekce
2: wait(S)	2: wait(S)
3: kritická sekce	3: kritická sekce
4: signal(S)	4: signal(S)

Obrázek: Řešení problému kritické sekce pro 2 procesy

- podobné druhému pokusu o řešení s await, ale operace na semaforu jsou atomické (testování a nastavení hodnoty)

Problém kritické sekce

- důkaz korektnosti pomocí stavového diagramu:

Kritická sekce (zkrácení)			
binary semaphore $S \leftarrow 1$			
A'		B'	
	loop forever		loop forever
1:	wait(S)	1:	wait(S)
2:	signal(S)	2:	signal(S)

Obrázek: Řešení problému kritické sekce pro 2 procesy (zkrácení)

Obr. 6.1

Obrázek: Stavový diagram řešení

Problém kritické sekce

- vzájemné vyloučení: není nežádoucí stav $(A'2, B'2, S)$, pro lib. hodnotu semaforu S, kdy oba procesy jsou v krit. sekci
- absence uváznutí: není stav, ve kterém by oba procesy čekaly/byly blokované
- absence vyhladovění s aktivními semafory: když proces nemůže vstoupit do krit. sekce, je v ní druhý proces, který ale po ukončení krit. sekce do ní může opět vstoupit (je vybrán)! $\rightarrow$ potřeba **silně férové plánování** procesů
- absence vyhladovění s pasivními semafory: když proces nemůže vstoupit do krit. sekce, je v ní druhý proces, který ale při ukončení krit. sekce první proces odblokuje a ten někdy vstoupí do krit. sekce, protože první je při opětovném pokusu o vstup do krit. sekce blokován

Problém kritické sekce

Pro lib. počet procesů

Kritická sekce
binary semaphore $S \leftarrow 1$
loop forever
1: nekritická sekce
2: wait(S)
3: kritická sekce
4: signal(S)

Obrázek: Řešení problému kritické sekce pro lib. počet procesů

- vzájemné vyloučení je splněno a k uváznutí dojít nemůže
- může dojít k vyhladovění procesu: *scénář?*
- proces(y) nemusí být vybrán(y) pro odblokování → potřeba **deterministické férové plánování** procesů, např. cyklický (round robin), z fronty procesů

Problém kritické sekce

Pro lib. počet procesů

Kritická sekce	
binary semaphore $S \leftarrow 1$	
loop forever	
1:	nekritická sekce
2:	wait(S)
3:	kritická sekce
4:	signal(S)

Obrázek: Řešení problému kritické sekce pro lib. počet procesů

- vzájemné vyloučení je splněno a k uváznutí dojít nemůže
- může dojít k vyhladovění procesu: *scénář?*
- proces(y) nemusí být vybrán(y) pro odblokování $\rightarrow$ potřeba **deterministické férové plánování** procesů, např. cyklický (round robin), z fronty procesů

Problém kritické sekce

- **silný (silně férový) semafor**: L je **fronta procesů** a proces pro odblokování v operaci signal není vybrán libovolně, ale např. v pořadí (!) blokování, jinak **slabý (slabě férový) semafor**

Otázka: *Kolik max. procesů může být v krit. sekci při inicializaci hodnoty semaforu na k?*

Problém producenta a konzumenta

– čekání konzumenta na produkci dat = čekání na semaforu, produkce dat = signalizace semaforu

Producent-konzument

dataType buffer[] ← nil
semaphore notEmpty ← 0

	producent		*konzument*
	dataType item		dataType item
	int i ← 0		int i ← 0
	loop forever		loop forever
1:	item = produce()	1:	wait(notEmpty)
2:	buffer[i] ← item	2:	item = buffer[i]
3:	i ← i + 1	3:	i ← i + 1
4:	signal(notEmpty)	4:	consume(item)

Obrázek: Řešení problému producenta a konzumenta s neomezeným bufferem

Problém producenta a konzumenta

– čekání producenta na konzumaci dat (při omezeném bufferu) = čekání na (jiném) semaforu, konzumace data = signalizace (jiného) semaforu

Producent-konzument

```
dataType buffer[N] ← nil
semaphore notEmpty ← 0
semaphore notFull ← N
```

producent	*konzument*
dataType item	dataType item
int i ← 0	int i ← 0
loop forever	loop forever
1: item = produce()	1: wait(notEmpty)
2: wait(notFull)	2: item = buffer[i]
3: buffer[i] ← item	3: i ← i + 1 mod N
4: i ← i + 1 mod N	4: signal(notFull)
5: signal(notEmpty)	5: consume(item)

Obrázek: Řešení problému producenta a konzumenta s omezeným bufferem

Problém producenta a konzumenta

- **rozdělené semafory (split semaphores)** = synch. mechanizmus, kdy **součet hodnot** V dvou a více semaforů je neustále roven nejvýše **pevné hodnotě** N
- operace wait a signal na semaforu jsou v **různých procesech**
- použití pro řešení problémů **pořadí vykonávání procesů**
- např. semafory notEmpty a notFull a N rovno velikosti bufferu
- pro $N = 1 \ldots$ **rozdělené binární semafory**

Problém bariéry

– čekání na bariéře = čekání na semaforu, signalizace příchodu k bariéře = signalizace semaforu

Pro 2 procesy

Bariéra	
binary semaphore $BA \leftarrow 0$	
binary semaphore $BB \leftarrow 0$	
A	B
loop forever	loop forever
1: akce	1: akce
2: signal(SA)	2: signal(SB)
3: wait(SB)	3: wait(SA)

Obrázek: Řešení problému bariéry pro 2 procesy

- semafory SA, SB . . . rozdělené binární semafory

Semafory

- jednodušší na použití než (atomické) instrukce poskytované strojem
- umožňují **pasivní čekání** (změnou stavu procesu na blokovaný)
- **středněúrovňové** synch. primitivum nejčastěji používané v OS a prog. jazycích pro implementaci **zapouzdřených** synch. primitiv na ještě vyšší úrovni