Paralelní programování

přednášky

Jan Outrata

únor–duben 2011

Problém kritické sekce

- Dijkstra, 1965
- centrální problém, nekorektní řešení demonstrují typické chyby konkurentních programů

Definice problému

- N procesů vykonává (v nekonečné smyčce) posloupnost akcí rozdělenou na dvě sekce: **kritickou** a **nekritickou sekci**
- korektnost řešení:
  1. **vzájemné vyloučení** – akce krit. sekcí (dvou a více) procesů nesmějí být proloženy, tj. (svou) krit. sekci může v daném čase vykonávat nejvýše jeden proces
  2. **absence uváznutí (deadlock)** – jestliže se nějaké procesy snaží současně vstoupit do (svých) krit. sekcí, pak jeden z nich musí někdy uspět
     - **uváznutí (deadlock)** = procesy se snaží současně vstoupit do (svých) krit. sekcí, ale nikdy nemohou
  3. **absence vyhladovění (starvation, zaručení vstupu)** – jestliže se nějaký proces snaží vstoupit do (své) krit. sekce, pak musí někdy uspět

Problém kritické sekce

→ synchronizace = další akce před a po krit. sekci – **vstupní** a **výstupní protokol** (**preprotocol** a **postprotocol**)

Problém kritické sekce	
globální proměnné	
A	*B*
lokální proměnné	lokální proměnné
loop forever	loop forever
nekritická sekce	nekritická sekce
vstupní protokol	vstupní protokol
kritická sekce	kritická sekce
výstupní protokol	výstupní protokol

Obrázek: Struktura (řešení) problému kritické sekce pro 2 procesy

Problém kritické sekce

- podmínky řešení:
  - **proměnné** použité v protokolech jsou **různé** od proměnných v sekcích
  - **krit. sekce vždy skončí** – jsou provedeny všechny její akce, nutné pro umožnění krit. sekcí jiných procesů
  - **nekrit. sekce nemusí skončit** – proces se v ní může ukončit (korektně i nekorektně) nebo může nekonečně cyklit
- efektivnost řešení – protokoly co nejjednodušší časově i paměťově
- **použití**: modelování přístupu k datům nebo hardwaru sdílených mezi více procesy, kdy není dovolen vícenásobný přístup, typicky **operace čtení–zápis** a **zápis–zápis**

První pokus

První pokus	
int turn $\leftarrow$ 1	
A	*B*
loop forever	loop forever
1: nekritická sekce	1: nekritická sekce
2: await turn $=$ 1	2: await turn $=$ 2
3: kritická sekce	3: kritická sekce
4: turn $\leftarrow$ 2	4: turn $\leftarrow$ 1

Obrázek: První pokus o řešení problému krit. sekce pro 2 procesy

await

await podmínka

= na implementaci nezávislé **čekání na splnění podmínky** = **podmíněná synchronizace**

- pokud podmínka splňuje podmínku kritických referencí, může být implementováno prázdnou **čekací smyčkou** (**busy-wait loop**) dokud není podmínka pravdivá, tzv. **aktivní čekání**

	Busy-wait loop
1:	while not podmínka
2:	nic

Obrázek: Aktivní čekání

await

- uváznutí (deadlock) $\sim$ zastavení výpočtu, s aktivním čekáním $\sim$ nekonečný cyklus testování podmínky = **livelock**
- při splnění podmínky akce **povolená**, jinak **zakázaná**

Přerušení cyklu 2			
int n $\leftarrow$ 0			
bool flag $\leftarrow$ false			
A		*B*	
1:	while flag = false	1:	await n = 1
2:	n $\leftarrow$ 1 - n	2:	flag $\leftarrow$ true

Obrázek: Demonstrace silné férovosti

Zastaví algoritmus vždy (pro všechny scénáře)? Tj. je korektní vzhledem k podmínce (postkondici), že vždy zastaví?

První pokus

První pokus	
int turn $\leftarrow$ 1	
A	*B*
loop forever	loop forever
1: nekritická sekce	1: nekritická sekce
2: await turn $= 1$	2: await turn $= 2$
3: kritická sekce	3: kritická sekce
4: turn $\leftarrow$ 2	4: turn $\leftarrow$ 1

Obrázek: První pokus o řešení problému krit. sekce pro 2 procesy

Je toto řešení problému krit. sekce korektní?

Dokazování korektnosti pomocí stavových diagramů

= **„neformální“ důkaz**

– pro určení efektu další akce procesu není nutné znát předchozí historii programu, tj. nezáleží na tom, jak se během výpočtu k akci došlo

– u sekvenčních programů je známa jediná další akce jednoho procesu, u konkurentních ne, ale jsou známy všechny možné další akce všech procesů

= **inkrementální konstrukce stavového diagramu** programu z počátečního stavu **a ověření všech dosažitelných stavů** a přechodů mezi nimi

Kolik může být stavů?

– pro N procesů s n_i akcemi v procesu $i, 1 \leq i \leq N$ a M proměnnými s m_j možnými hodnotami proměnné $j, 1 \leq j \leq M$

= počet všech n-tic nezávisle vybraných akcí a proměnných:
$\mathbf{n_1} \times \ldots \times \mathbf{n_N} \times \mathbf{m_1} \times \ldots \times \mathbf{m_M}$

Dokazování korektnosti pomocí stavových diagramů

Př. Počet stavů prvního pokusu

2 procesy s 4 a 4 akcemi, 1 proměnná s 2 možnými hodnotami (1 a 2) $\rightarrow$ počet stavů: $n_1 \times n_2 \times m_1 = 4 \times 4 \times 2 = 32$

- **ne všechny stavy jsou dosažitelné** v jakémkoliv scénáři z počátečního stavu

Fig. 3.1

Obrázek: Stavový diagram prvního pokusu (část)

Př. Počet dosažitelných stavů prvního pokusu: 16

- pro synchronizaci (a důkaz korektnosti) jsou **relevantní jen akce a proměnné v protokolech**, ne v sekcích samotných – používají různé proměnné, neovlivní synchronizaci

$\rightarrow$ **odstranění** (zakomentování) **akcí sekcí** a ponechání jen akcí protokolů

Dokazování korektnosti pomocí stavových diagramů

První pokus (zkrácení)	
int turn $\leftarrow$ 1	
A'	B'
loop forever	loop forever
1: await turn $=$ 1	1: await turn $=$ 2
2: turn $\leftarrow$ 2	2: turn $\leftarrow$ 1

Obrázek: První pokus o řešení problému krit. sekce pro 2 procesy (zkrácení)

Př. Počet stavů prvního pokusu (zkráceného): $2 \times 2 \times 2 = 8$

Fig. 3.2

Obrázek: Stavový diagram prvního pokusu (zkráceného)

Dokazování korektnosti řešení problému krit. sekce

= ověření splnění 3 vlastností korektnosti

Vzájemné vyloučení

- v každém stavu lib. scénáře stavového diagramu je nejvýše 1 následující akce z krit. sekce nějakého procesu, jinak **nežádoucí stav**
- důkaz splnění = absence nežádoucího stavu ve všech scénářích $\Rightarrow$ nutné zkonstruovat celý stavový diagram
- důkaz nesplnění = prezence nežádoucího stavu v lib. scénáři $\Rightarrow$ není nutné zkonstruovat celý stavový diagram

Absence uváznutí a vyhladovění

- absence stavu s akcí vstupního protokolu, ze kterého se nelze dostat do stavu s akcí krit. sekce
- u uváznutí pro více procesů (všechny procesy) zároveň
- u vyhladovění pro každý jeden proces zvlášť

Korektnost prvního pokusu

1. vzájemné vyloučení: ve stavovém diagramu není nežádoucí stav $(A'2, B'2, turn)$, pro lib. hodnotu proměnné *turn* (1 nebo 2), kdy oba procesy jsou v krit. sekci
2. absence uváznutí: proces se snaží vstoupit do krit. sekce ve stavech s akcí await, ze stavu $(A'1, B'1, 1)$ může vstoupit proces A', ze stavu $(A'2, B'1, 1)$ může pokračovat proces A' do stavu $(A'1, B'1, 2)$, ze kterého může vstoupit proces B', zvývající 2 stavy analogicky
3. absence vyhladovění: ze stavu $(A1, B2, 1)$ nemůže proces B vstupit do krit. sekce, dokud proces A nevykoná akci $A4$, ale ten je v nekrit. sekci a ta nemusí skončit!

Korektnost prvního pokusu

- proměnná *turn* funguje jako **„právo“ procesu ke vstupu do krit. sekce**
- proces čeká na „právo“ vstupu do (své) krit. sekce
- vždy nějaký proces má „právo“ $\rightarrow$ není uváznutí
- proces nemusí „právo“ předat, např. zůstane v nekrit. sekci $\rightarrow$ vyhladovění

$\rightarrow$ proces v nekrit. sekci nemůže bránit vstupu jiného do krit. sekce

$\rightarrow$ **procesy nemohou testovat a nastavovat jedinou (glob.) proměnnou**

Korektnost prvního pokusu

- proměnná *turn* funguje jako **„právo“ procesu ke vstupu do krit. sekce**
- proces čeká na „právo“ vstupu do (své) krit. sekce
- vždy nějaký proces má „právo“ $\rightarrow$ není uváznutí
- proces nemusí „právo“ předat, např. zůstane v nekrit. sekci $\rightarrow$ vyhladovění

$\rightarrow$ proces v nekrit. sekci nemůže bránit vstupu jiného do krit. sekce

$\rightarrow$ **procesy nemohou testovat a nastavovat jedinou (glob.) proměnnou**

Druhý pokus

→ každý proces má svoji (glob.) proměnnou

Druhý pokus	
bool isA ← false, isB ← false	
A	*B*
loop forever	loop forever
1: nekritická sekce	1: nekritická sekce
2: await isB = false	2: await isA = false
3: isA ← true	3: isB ← true
4: kritická sekce	4: kritická sekce
5: isA ← false	5: isB ← false

Obrázek: Druhý pokus o řešení problému krit. sekce pro 2 procesy

- proměnné *isA* a *isB* fungují jako **„oznámení“ o vykonávání krit. sekce**, dokud tato neskončí
- proces čeká na zrušení „oznámení“ jiných procesů

Druhý pokus

- proces nebrání vstupu jiného do krit. sekce, např. když zůstane v nekrit. sekci, jeho „oznámení“ zůstává zrušené, a nemůže dojít k uváznutí
- vzájemné vyloučení není splněné: *scénář?*
- stav po čekání, ale před nastavením „oznámení“ o vykonávání krit. sekce, je efektivně součástí krit. sekce, ale „oznámení“ ještě není nastaveno a jiné procesy nemusí čekat

→ **procesy nemohou měnit podmínky vstupu jiných procesů do (jejich) krit. sekcí až v (své) krit. sekci**

Druhý pokus

- proces nebrání vstupu jiného do krit. sekce, např. když zůstane v nekrit. sekci, jeho „oznámení“ zůstává zrušené, a nemůže dojít k uváznutí
- vzájemné vyloučení není splněné: *scénář?*
- stav po čekání, ale před nastavením „oznámení“ o vykonávání krit. sekce, je efektivně součástí krit. sekce, ale „oznámení“ ještě není nastaveno a jiné procesy nemusí čekat

→ **procesy nemohou měnit podmínky vstupu jiných procesů do (jejich) krit. sekcí až v (své) krit. sekci**

Druhý pokus

- proces nebrání vstupu jiného do krit. sekce, např. když zůstane v nekrit. sekci, jeho „oznámení“ zůstává zrušené, a nemůže dojít k uváznutí
- vzájemné vyloučení není splněné: *scénář?*
- stav po čekání, ale před nastavením „oznámení“ o vykonávání krit. sekce, je efektivně součástí krit. sekce, ale „oznámení“ ještě není nastaveno a jiné procesy nemusí čekat

→ **procesy nemohou měnit podmínky vstupu jiných procesů do (jejich) krit. sekcí až v (své) krit. sekci**

Třetí pokus

→ „oznámení“ je nastaveno už před čekáním

Třetí pokus	
bool wantA ← false, wantB ← false	
A	*B*
loop forever	loop forever
1: nekritická sekce	1: nekritická sekce
2: wantA ← true	2: wantB ← true
3: await wantB = false	3: await wantA = false
4: kritická sekce	4: kritická sekce
5: wantA ← false	5: wantB ← false

Obrázek: Třetí pokus o řešení problému krit. sekce pro 2 procesy

- proměnné *wantA* a *wantB* fungují jako **„přání“ vstupu do krit. sekce**, dokud tato neskončí
- proces čeká na zrušení „přání“ jiných procesů

Třetí pokus

- vzájemné vyloučení je splněno a k vyhladovění procesu dojít nemůže
- může dojít k uváznutí: *scénář?*
- proměnné ve skutečnosti fungují jako **„trvání na“ vstupu do krit. sekce**

→ uváznutí = oba procesy současně „trvají na“ vstupu do krit. sekce

→ **procesy si nemohou vynucovat vstup do (své) krit. sekce**

Třetí pokus

- vzájemné vyloučení je splněno a k vyhladovění procesu dojít nemůže
- může dojít k uváznutí: *scénář?*
- proměnné ve skutečnosti fungují jako **„trvání na“ vstupu do krit. sekce**

→ uváznutí = oba procesy současně „trvají na“ vstupu do krit. sekce

→ **procesy si nemohou vynucovat vstup do (své) krit. sekce**

Třetí pokus

- vzájemné vyloučení je splněno a k vyhladovění procesu dojít nemůže
- může dojít k uváznutí: *scénář?*
- proměnné ve skutečnosti fungují jako **„trvání na“ vstupu do krit. sekce**

→ uváznutí = oba procesy současně „trvají na“ vstupu do krit. sekce

→ **procesy si nemohou vynucovat vstup do (své) krit. sekce**

Čtvrtý pokus

→ proces se **„dočasně vzdá trvání na“ vstupu do (své) krit. sekce**, pokud na vstupu do (své) krit. sekce „trvá“ jiný proces

Čtvrtý pokus	
bool wantA ← false, wantB ← false	
A	*B*
loop forever	loop forever
1: nekritická sekce	1: nekritická sekce
2: wantA ← true	2: wantB ← true
3: while wantB	3: while wantA
4: wantA ← false	4: wantB ← false
5: wantA ← true	5: wantB ← true
6: kritická sekce	6: kritická sekce
7: wantA ← false	7: wantB ← false

Obrázek: Čtvrtý pokus o řešení problému krit. sekce pro 2 procesy

Čtvrtý pokus

- akce 4 a 5 za sebou mají v konkuretním programu, vzhledem k lib. proložení akcí, smysl, mezi nimi může druhý proces vstoupit do (své) krit. sekce
- k uváznutí nemůže dojít (není čekání) a vzájemné vyloučení je splněno
- může dojít k vyhladovění procesu: *scénář?*
- scénář vyhladovění „nerealistický“? V našem modelu libovolných proložení je ale **možný**.
- procesy se vyhladoví „navzájem“ $\sim$ „livelock“ ve vstupním protokolu

Čtvrtý pokus

- akce 4 a 5 za sebou mají v konkuretním programu, vzhledem k lib. proložení akcí, smysl, mezi nimi může druhý proces vstoupit do (své) krit. sekce
- k uváznutí nemůže dojít (není čekání) a vzájemné vyloučení je splněno
- může dojít k vyhladovění procesu: *scénář?*
- scénář vyhladovění „nerealistický“? V našem modelu libovolných proložení je ale **možný**.
- procesy se vyhladoví „navzájem“ $\sim$ „livelock“ ve vstupním protokolu

Čtvrtý pokus

- akce 4 a 5 za sebou mají v konkuretním programu, vzhledem k lib. proložení akcí, smysl, mezi nimi může druhý proces vstoupit do (své) krit. sekce
- k uváznutí nemůže dojít (není čekání) a vzájemné vyloučení je splněno
- může dojít k vyhladovění procesu: *scénář?*
- scénář vyhladovění „nerealistický“? V našem modelu libovolných proložení je ale **možný**.
- procesy se vyhladoví „navzájem“ $\sim$ „livelock“ ve vstupním protokolu

Dekkerův algoritmus

Dekkerův algoritmus

bool wantA ← false, wantB ← false
int turn ← 1

	A		*B*
	loop forever		loop forever
1:	nekritická sekce	1:	nekritická sekce
2:	wantA ← true	2:	wantB ← true
3:	while wantB	3:	while wantA
4:	if turn = 2	4:	if turn = 1
5:	wantA ← false	5:	wantB ← false
6:	await turn = 1	6:	await turn = 2
7:	wantA ← true	7:	wantB ← true
8:	kritická sekce	8:	kritická sekce
9:	turn ← 2	9:	turn ← 1
10:	wantA ← false	10:	wantB ← false

Obrázek: Dekkerův algoritmus řešení problému krit. sekce pro 2 procesy

Dekkerův algoritmus

= kombinace prvního a čtvrtého pokusu

– v prvním pokusu se předává „právo“ procesů ke vstupu do krit. sekce – nekorektní v případě, že procesy nesoupeří o vstup

– čtvrtý pokus řeší problém při nesoupeření o vstup, ale v případě soupeření procesy „trvají na“ vstupu do krit. sekce

→ předává se **„právo na trvání na“ vstupu do krit. sekce**

- korektní – důkaz pomocí stavového diagramu?!
- počet stavů (bez akcí sekcí): $8 \times 8 \times 2 \times 2 \times 2 = 512$ → **formální důkaz** pomocí (induktivních a deduktivních) důkazů tzv. **invariantů** v **temporální logice**

Petersonův algoritmus (Tie-breaker)

- Peterson, 1981

Petersonův algoritmus			
bool wantA ← false, wantB ← false int last ← 1			
A		*B*	
	loop forever		loop forever
1:	nekritická sekce	1:	nekritická sekce
2:	wantA ← true	2:	wantB ← true
3:	last ← 1	3:	last ← 2
4:	await wantB = false or last = 2	4:	await wantA = false or last = 1
5:	kritická sekce	5:	kritická sekce
6:	wantA ← false	6:	wantB ← false

Obrázek: Petersonův algoritmus řešení problému krit. sekce pro 2 procesy

Petersonův algoritmus (Tie-breaker)

- založený na Dekkerově algoritmu, cyklus a await složeny do jedné akce **await se složenou podmínkou**
- proměnná *last* indikuje, který proces začal vykonávat vstupní protokol jako poslední a bude (dál) čekat
- podmínka v **await nesplňuje podmínku kritických referencí**

Otázka: *Zůstane algoritmus korektní, i když složená podmínka v await není vyhodnocována atomicky (a await je implementováno aktivním čekáním)?*

Složené atomické akce

- atomické akce při řešení problému kritické sekce byly pouze jednoduché: načtení z a uložení do (sdílené) paměti – těžké problém vyřešit
- lehké, když **atom. akce může číst z i ukládat do (sdílené) paměti**
- implementované v OS nebo hardware

Test-and-set

test-and-set(lock, local)	
1:	local $\leftarrow$ lock
2:	lock $\leftarrow$ true

Obrázek: Složená atomická akce test-and-set

Složené atomické akce

Test-and-set

Problém kritické sekce s test-and-set

bool lock ← false

	A		*B*
	bool localA		bool localB
	loop forever		loop forever
1:	nekritická sekce	1:	nekritická sekce
	repeat		repeat
2:	test-and-set(lock, localA)	2:	test-and-set(lock, localB)
3:	until localA = false	3:	until localB = false
4:	kritická sekce	4:	kritická sekce
5:	lock ← false	5:	lock ← false

Obrázek: Řešení problému krit. sekce pomocí test-and-set

Složené atomické akce

Test-and-set

- proměnná *lock* funguje jako **„zámek“ krit. sekce**, dokud $lock = 1$, není dovoleno vstoupit
- cyklus repeat = **spin lock**
- bývá také implementována tak, že vrací původní hodnotu proměnné *lock*:

test-and-set(lock)	
	bool local
1:	local ← lock
2:	lock ← true
3:	return local

Obrázek: Složená atomická akce test-and-set (vracející hodnotu)

- splnění absence vyhladovění jen v silně férových scénářích

Složené atomické akce

Exchange (atomická výměna hodnot dvou proměnných)

exchange(a, b)	
	bool tmp
1:	tmp $\leftarrow$ a
2:	a $\leftarrow$ b
3:	b $\leftarrow$ tmp

Obrázek: Složená atomická akce exchange

Složené atomické akce

Exchange (atomická výměna hodnot dvou proměnných)

Problém kritické sekce s exchange			
bool lock ← false			
	A		*B*
	bool localA ← true		bool localB ← true
	loop forever		loop forever
1:	nekritická sekce	1:	nekritická sekce
	repeat		repeat
2:	exchange(lock, localA)	2:	exchange(lock, localB)
3:	until localA = false	3:	until localB = false
4:	kritická sekce	4:	kritická sekce
5:	exchange(lock, localA)	5:	exchange(lock, localB)

Obrázek: Řešení problému krit. sekce pomocí exchange

- řešení fungují i pro lib. počet procesů

Složené atomické akce

Fetch-and-add

```
fetch-and-add(counter, local, x)
1:  local ← counter
2:  counter ← counter + 1
```

Obrázek: Složená atomická akce fetch-and-add

Compare-and-swap

```
compare-and-swap(lock, old, new)
    bool tmp
1:  tmp ← lock
2:  if lock = old
3:      lock ← new
4:  return tmp
```

Obrázek: Složená atomická akce compare-and-swap

Složené atomické akce

- každou z uvedených složených atom. akcí lze implementovat pomocí jiné
- místo uvedených složených atom. akcí **lze pro zajištění atomicity** jejich implementace, tj. neproložitelnosti ($\sim$ vzájemného vyloučení) a izolovanosti ($\sim$ použití různých proměnných) jejich akcí, **použít lib. řešení problému krit. sekce s jednoduchými atom. akcemi**!

Problém kritické sekce

- použití aktivního čekání v řešeních problému kritické sekce – **nevýhodné při multitaskingu** (1 procesor, v nepreemptivním dokonce nepoužitelné) **a při větším soupeření procesů o vstup do krit. sekcí** (např. při delších krit. sekcích)

→ tzv. **pasivní čekání** = implementace await pomocí **blokování (pozastavení) procesu plánovačem procesů** v OS

- uvedená řešení problému kritické sekce v praxi nepoužívaná (kromě tzv. **„rychlých“ řešení** dále) → **vyšší synchronizační primitiva**