Paralelní programování

přednáška 3

Michal Krupka

1. března 2011

Ještě k atomickým proměnným

Další neatomické proměnné

- Mohou to být proměnné, jejichž velikost je *menší* než slovo: Příkaz

  $A[i] = 1;$

  nemusí být atomický, pokud je A bitové pole. (Proč?)

Vždy je potřeba konzultovat dokumentaci

Ještě k atomickým proměnným

Další neatomické proměnné

- Mohou to být proměnné, jejichž velikost je *menší* než slovo: Příkaz

  $A[i] = 1;$

  nemusí být atomický, pokud je A bitové pole. (Proč?)

Vždy je potřeba konzultovat dokumentaci

Sdílené proměnné, aktivní čekání

Synchronizace procesů

- obvykle nutná kvůli dosažení korektnosti programu
- nejjednodušší způsob: pomocí **sdílených proměnných** a **aktivního čekání**

Sdílené proměnné

- přístupné více procesům ke čtení i zápisu
- nejjednodušší příklad: globální proměnné, dále místo v paměti o známé adrese (např. předané procesu parametrem), lexikální uzávěr
- musí být atomické a volatilní

Aktivní čekání

- periodická kontrola hodnoty sdílené proměnné, kterou nastavuje jiný proces
- operace **await**, např:

$$\text{await } n = 0 \quad \Leftrightarrow \quad \text{loop while } n \neq 0$$

- v praxi se nepoužívá (pasivní čekání)

Sdílené proměnné, aktivní čekání

Synchronizace procesů

- obvykle nutná kvůli dosažení korektnosti programu
- nejjednodušší způsob: pomocí **sdílených proměnných** a **aktivního čekání**

Sdílené proměnné

- přístupné více procesům ke čtení i zápisu
- nejjednodušší příklad: globální proměnné, dále místo v paměti o známé adrese (např. předané procesu parametrem), lexikální uzávěr
- musí být atomické a volatilní

Aktivní čekání

- periodická kontrola hodnoty sdílené proměnné, kterou nastavuje jiný proces
- operace **await**, např:

$$\text{await } n = 0 \quad \Leftrightarrow \quad \text{loop while } n \neq 0$$

- v praxi se nepoužívá (pasivní čekání)

Sdílené proměnné, aktivní čekání

Synchronizace procesů

- obvykle nutná kvůli dosažení korektnosti programu
- nejjednodušší způsob: pomocí **sdílených proměnných** a **aktivního čekání**

Sdílené proměnné

- přístupné více procesům ke čtení i zápisu
- nejjednodušší příklad: globální proměnné, dále místo v paměti o známé adrese (např. předané procesu parametrem), lexikální uzávěr
- musí být atomické a volatilní

Aktivní čekání

- periodická kontrola hodnoty sdílené proměnné, kterou nastavuje jiný proces
- operace **await**, např:

$$\text{await } n = 0 \quad \Leftrightarrow \quad \text{loop while } n \neq 0$$

- v praxi se nepoužívá (pasivní čekání)

Dodavatel-odběratel

Dodavatel-odběratel (Producer-Consumer)

- jeden ze základních synchronizačních problémů
- dva procesy: **dodavatel** (**Producer**) periodicky získává data a ukládá je do sdíleného úložiště (**bufferu**), **odběratel** (**Consumer**) vyzvedává data z bufferu a zpracovává je
- dodavatel nesmí zapisovat do plného bufferu (příp. musí počkat)
- odběratel nesmí číst z prázdného bufferu (příp. musí počkat)

Jednoduchý případ: Pouze dva procesy, buffer pojme jednu položku.
Zobecnění: Buffer pojme více položek, více dodavatelů a odběratelů.

Pozn.: větší buffer má smysl kvůli vyrovnávání, ne pokud je rychlost dodavatele větší než odběratele.

Příklady

- dodavatel: komunikační linka, odběratel: webový prohlížeč
- dodavatel: klávesnice, odběratel: operační systém
- dodavatel: výpočetní proces, odběratel: GUI

Dodavatel-odběratel

Dodavatel-odběratel (Producer-Consumer)

- jeden ze základních synchronizačních problémů
- dva procesy: **dodavatel** (**Producer**) periodicky získává data a ukládá je do sdíleného úložiště (**bufferu**), **odběratel** (**Consumer**) vyzvedává data z bufferu a zpracovává je
- dodavatel nesmí zapisovat do plného bufferu (příp. musí počkat)
- odběratel nesmí číst z prázdného bufferu (příp. musí počkat)

Jednoduchý případ: Pouze dva procesy, buffer pojme jednu položku.
Zobecnění: Buffer pojme více položek, více dodavatelů a odběratelů.

Pozn.: větší buffer má smysl kvůli vyrovnávání, ne pokud je rychlost dodavatele větší než odběratele.

Příklady

- dodavatel: komunikační linka, odběratel: webový prohlížeč
- dodavatel: klávesnice, odběratel: operační systém
- dodavatel: výpočetní proces, odběratel: GUI

Dodavatel-odběratel

Dodavatel-odběratel (Producer-Consumer)

- jeden ze základních synchronizačních problémů
- dva procesy: **dodavatel** (**Producer**) periodicky získává data a ukládá je do sdíleného úložiště (**bufferu**), **odběratel** (**Consumer**) vyzvedává data z bufferu a zpracovává je
- dodavatel nesmí zapisovat do plného bufferu (příp. musí počkat)
- odběratel nesmí číst z prázdného bufferu (příp. musí počkat)

Jednoduchý případ: Pouze dva procesy, buffer pojme jednu položku.
Zobecnění: Buffer pojme více položek, více dodavatelů a odběratelů.

Pozn.: větší buffer má smysl kvůli vyrovnávání, ne pokud je rychlost dodavatele větší než odběratele.

Příklady

- dodavatel: komunikační linka, odběratel: webový prohlížeč
- dodavatel: klávesnice, odběratel: operační systém
- dodavatel: výpočetní proces, odběratel: GUI

Dodavatel-odběratel

Dodavatel-odběratel (Producer-Consumer)

- jeden ze základních synchronizačních problémů
- dva procesy: **dodavatel** (**Producer**) periodicky získává data a ukládá je do sdíleného úložiště (**bufferu**), **odběratel** (**Consumer**) vyzvedává data z bufferu a zpracovává je
- dodavatel nesmí zapisovat do plného bufferu (příp. musí počkat)
- odběratel nesmí číst z prázdného bufferu (příp. musí počkat)

Jednoduchý případ: Pouze dva procesy, buffer pojme jednu položku.
Zobecnění: Buffer pojme více položek, více dodavatelů a odběratelů.

Pozn.: větší buffer má smysl kvůli vyrovnávání, ne pokud je rychlost dodavatele větší než odběratele.

Příklady

- dodavatel: komunikační linka, odběratel: webový prohlížeč
- dodavatel: klávesnice, odběratel: operační systém
- dodavatel: výpočetní proces, odběratel: GUI

Dodavatel-odběratel

Zadání jednoduché verze

- Dodavatel v nekonečné smyčce volá funkci produce(), která vrací dodávaná data,
- data ukládá do sdílené proměnné buffer.
- Odběratel data vyzvedává z proměnné buffer,
- periodicky volá funkci consume() s těmito daty.
- Hodnota nil v proměnné buffer znamená žádná data.

Dodavatel-odběratel

Řešení jednoduché verze

Dodavatel-odběratel 1	
dataType buffer ← nil	
Producer	Consumer
dataType item	dataType item
p1: loop forever	c1: loop forever
p2: item ← produce()	c2: await buffer
p3: await not buffer	c3: item ← buffer
p4: buffer ← item	c4: buffer ← nil
	c5: consume(item)

Obrázek: Dodavatel-odběratel pro buffer délky 1

Dodavatel-odběratel

Korektnost

Dodavatel-odběratel 1

dataType buffer ← nil

Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await buffer
p3:	await not buffer	c3:	item ← buffer
p4:	buffer ← item	c4:	buffer ← nil
		c5:	consume(item)

Dodavatel-odběratel

Korektnost

Dodavatel-odběratel 1

dataType buffer ← nil

Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await buffer
p3:	await not buffer	c3:	item ← buffer
p4:	buffer ← item	c4:	buffer ← nil
		c5:	consume(item)

Podmínky **bezpečnosti** (neformálně):

Dodavatel-odběratel

Korektnost

Dodavatel-odběratel 1			
dataType buffer ← nil			
Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await buffer
p3:	await not buffer	c3:	item ← buffer
p4:	buffer ← item	c4:	buffer ← nil
		c5:	consume(item)

Podmínky **bezpečnosti** (neformálně):

- každá hodnota vrácená funkcí produce() se dostane do proměnné buffer,

Dodavatel-odběratel

Korektnost

Dodavatel-odběratel 1

dataType buffer ← nil

Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await buffer
p3:	await not buffer	c3:	item ← buffer
p4:	buffer ← item	c4:	buffer ← nil
		c5:	consume(item)

Podmínky **bezpečnosti** (neformálně):

- každá hodnota vrácená funkcí produce() se dostane do proměnné buffer,
- funkce consume() bude zavolána s každou hodnotou v proměnné buffer.

Dodavatel-odběratel

Korektnost

Dodavatel-odběratel 1

dataType buffer ← nil

Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await buffer
p3:	await not buffer	c3:	item ← buffer
p4:	buffer ← item	c4:	buffer ← nil
		c5:	consume(item)

Podmínky **bezpečnosti** (neformálně):

- každá hodnota vrácená funkcí produce() se dostane do buffer:
  - Producer.item je lokální proměnná,
  - podmínka živosti (a férovosti),

Dodavatel-odběratel

Korektnost

Dodavatel-odběratel 1

dataType buffer ← nil

Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await buffer
p3:	await not buffer	c3:	item ← buffer
p4:	buffer ← item	c4:	buffer ← nil
		c5:	consume(item)

Podmínky **bezpečnosti** (neformálně):

- každá hodnota vrácená funkcí produce() se dostane do buffer:
  - Producer.item je lokální proměnná,
  - podmínka živosti (a férovosti),
- funkce consume() bude zavolána s každou hodnotou v buffer:
  - p4 má splněnou prekondici buffer = nil,
  - je-li buffer ≠ nil, nevykonává se p4 a nutně dojde k c3 a c5 (živost).

Dodavatel-odběratel

Korektnost

Dodavatel-odběratel 1

dataType buffer ← nil

Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await buffer
p3:	await not buffer	c3:	item ← buffer
p4:	buffer ← item	c4:	buffer ← nil
		c5:	consume(item)

Podmínky **živosti**:

- p3: je-li buffer $\neq$ nil, v c2 se nečeká a dojde k c4 (férovost),
- c2: je-li buffer $=$ nil, v p3 se nečeká a dojde k p2 (rovněž férovost).

Dodavatel-odběratel

Korektnost

Dodavatel-odběratel 1

dataType buffer ← nil

Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await buffer
p3:	await not buffer	c3:	item ← buffer
p4:	buffer ← item	c4:	buffer ← nil
		c5:	consume(item)

Podmínky **živosti**:

- p3: je-li buffer $\neq$ nil, v c2 se nečeká a dojde k c4 (férovost),
- c2: je-li buffer $=$ nil, v p3 se nečeká a dojde k p2 (rovněž férovost).

Cvičení

1. Pozměňte řešení pro situaci, kdy nelze použít nil pro signalizaci prázdného bufferu (zavést novou proměnnou).
2. Přidejte do řešení možnost ukončení obou cyklů v případě, že funkce produce() vrátí nil.
3. Okomentujte zjednodušenou verzi bez proměnné item:

dataType buffer ← nil			
Producer		Consumer	
p1:	loop forever	c1:	loop forever
p2:	await not buffer	c2:	await buffer
p3:	buffer ← produce()	c3:	consume(buffer)
		c4:	buffer ← nil

Dodavatel-odběratel

Varianta: odběratel v hlavní smyčce GUI aplikace

- dodavatel provádí dlouhý výpočet, který nemá blokovat GUI,
- hlavní proces aplikace je odběratel, testuje data ve smyčce událostí (musí tedy dostávat pravidelné *Idle Events*; v praxi se moc nepoužívá).

Dodavatel-odběratel 2

dataType buffer ← nil

Producer	Consumer
dataType item	dataType item
p1: loop forever	c1: loop forever
p2: item ← produce()	// . . . zpracování události . . .
p3: await not buffer	c2: if buffer then
p4: buffer ← item	c3: item ← buffer
	c4: buffer ← nil
	c5: consume(item)

Obrázek: Dodavatel-odběratel, buffer délky 1, varianta pro smyčku událostí

Dodavatel-odběratel

Varianta: odběratel v hlavní smyčce GUI aplikace

- dodavatel provádí dlouhý výpočet, který nemá blokovat GUI,
- hlavní proces aplikace je odběratel, testuje data ve smyčce událostí (musí tedy dostávat pravidelné *Idle Events*; v praxi se moc nepoužívá).

Dodavatel-odběratel 2

dataType buffer ← nil

Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()		// . . . zpracování události . . .
p3:	await not buffer	c2:	if buffer then
p4:	buffer ← item	c3:	item ← buffer
		c4:	buffer ← nil
		c5:	consume(item)

Obrázek: Dodavatel-odběratel, buffer délky 1, varianta pro smyčku událostí

Dodavatel-odběratel

Zadání verze s omezeným bufferem

- buffer je jednorozměrné pole dané délky
- nil opět znamená prázdnou hodnotu
- varianta: předchozí bod neplatí (prázdná hodnota neexistuje)

Dodavatel-odběratel

Řešení problému omezeného bufferu s nil

Dodavatel-odběratel 3

dataType buffer[N] ← nil

Producer		Consumer	
	dataType item		dataType item
	integer i ← 0		integer i ← 0
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await buffer[i]
p3:	await not buffer[i]	c3:	item ← buffer[i]
p4:	buffer[i] ← item	c4:	buffer[i] ← nil
p5:	i ← i + 1 mod N	c5:	consume(item)
		c6:	i ← i + 1 mod N

Obrázek: Dodavatel-odběratel pro omezený buffer s nil

Dodavatel-odběratel

Řešení problému omezeného bufferu bez nil

- předpokládáme, že množina přirozených čísel je neomezená

Dodavatel-odběratel 4

dataType buffer[N]
integer bottom,top ← 0

Producer		Consumer	
	dataType item		dataType item
p1:	loop forever	c1:	loop forever
p2:	item ← produce()	c2:	await bottom < top
p3:	await top < bottom + N	c3:	item ← buffer[bottom mod N]
p4:	buffer[top mod N] ← item	c4:	bottom ← bottom + 1
p5:	top ← top + 1	c5:	consume(item)

Obrázek: Dodavatel-odběratel pro omezený buffer

Cvičení

4. Ověřte korektnost obou programů.
5. Přidejte do obou programů možnost ukončení (viz Cv. 2).
6. Upravte program Dodavatel-odběratel 4 tak, aby se proměnné bottom a top neinkrementovaly donekonečna.