Strategický rozvoj Univerzity Hradec Králové
CZ.02.2.69/0.0/0.0/16_015/0002427

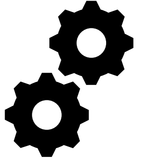
Synchronizace

Mgr. Josef Horálek, Ph.D.

- Kooperující proces je proces, který může ovlivnit nebo být ovlivněn jiným procesem právě spuštěným v systému
- Spolupracující procesy mohou sdílet:
  - Logický adresový prostor (jak kód, tak i data);
  - Data pouze prostřednictvím sdílených souborů;

Modelový příklad

- Server – obsluhuje dotazy klientů ve dvou vláknech
  - Na začátku obě čekají na příchod požadavku
- Klient zašle zprávu
  - Jedno vlákno se probudí
    - zprávu vyzvedne z fronty příchozích
    - zpracuje ji
    - zašle odpověď
    - vrací se do stavu čekající
  - Druhé vlákno čeká na požadavek od dalšího klienta



Jak to funguje v kódu

```
VAR
Completed: Integer = 0; //globální proměnná
…
Procedure Server;
VAR
   P: Trequest;
begin
While WaitForRequest do //uspí vlákno, dokud je vstupní fronta prázdná
   begin
   P := GetRequest;         //odstranění příchozí zprávy ze vstupní fronty
   ProcessRequest (P);      //realizuje zpracování zprávy
   SendResponse (P);        //odeslání výsledku klientovi
   Completed := Completed +1; //počet vyřízených požadavků serverem
   end;
end.
```

- Proměnná `Completed` neošetřena proti paralelnímu přístupu
  - ➜ zkreslená statistika

Implementace pomocí instrukcí

```
LOAD R1, Completed      // Do registru R1 načte hodnotu hodnotu Completed
ADD R1, 1               // Inkrementace registru o 1
STORE R1, Completed     // Uložení registru na adresu proměnné
```

- Neošetření paralelního přístupu
  - Lze kdykoli přeplánovat na jiné vlákno
- Server zpracovává mnoho požadavků
  - obě vlákna běží paralelně
    - vysoká pravděpodobnost inkrementace sdílené proměnné ve stejném čase

Chybový stav – chyba souběhu

Vlákno 1

Vlákno 2

LOAD R1, Completed
ADD R1, 1

Přeplánování

LOAD R1, Completed
ADD R1, 1
STORE R1, Completed

Přeplánování

STORE R1, Completed

- **Vlákno 2** načte hodnotu proměnné `Completed` dříve, než ji **Vlákno 1** inkrementuje a uloží ☹

Kritická sekce

- Problém způsobují POUZE sdílená data mezi více entitami
- Činnost vlákna dělíme:
  - Vlákno pracuje pouze s lokálními proměnnými
    - nezávisle na sekvenci plánovače procesoru – vždy správný výsledek
  - Vlákno pracuje s globálními či jinak sdíleými daty
    - neřešení synchronice - výsledek výpočtu může záviset na konkrétním plánovanání jednotlivých vláken
    - tato vlákna pracují v tzv. KRITICKÉ SEKCI

Podmínky řešení - Kritické sekce

- Vzájemné vyloučení
  - V daném okamžiku se v kritické sekci nachází VŽDY jen jedno vlákno
  - Vlákno NEnacházející se v kritické sekci, nesmí žádným způsobem blokovat běh vlákna v kritické sekci
    - obsazená kritická sekce blokuje případné další čekatele na vstup
  - Vlákno nesmí v kritické sekci strávit neomezenou dobu
- Synchronizační primitiva
  - Obecná charakteristika řešení Kritické sekce

Obecná řešení Kritické sekce
ZAKÁZANÍ PŘERUŠENÍ

- Vznik synchronizačních problémů
  - Nečekaná sekvence plánování vláken na neočekávaných místech
    - řešením je vypnutí plánovače v těchto obtížných místech
- Zákaz HW přerušení
  - Spouštění plánovače jen:
    - reaguje-li na volání některých rutin jádra
      - v kritické sekci nepoužíváme procedury a funkce
    - během obsluhy přerušení od časovače
      - zakázání hardwarových přerušení

- Instrukce CLI a STI
  - Instrukce na procesorech kompatibilních s architekturou x86 a AMD64
    - zákaz přerušení na začátku Kritické sekce
    - povolení přerušení na konci Kritické sekce
  - Zajištění podmínky vzájemného vyloučení
- Možné problémy
  - CLI zakazuje přerušení jen na aktuálním procesoru
    - nevhodné pro víceprocesorové výpočetní systémy
  - Zakázání HW přerušení jen kódu běžícím v privilegovaném módu
    - nelze použít pro kód běžící v uživatelském režimu

- Atomické operace
  - Operace u nichž procesory garantují NEpřerušení při jejich provádění
    - zápis a čtení bloku paměti o velikosti jeden, dva a čtyři bajty
  - Tvořeny jednou instrukcí

- Instrukce test and set (TSL)
  - Instrukce podporované přímo v instrukční sadě (x86 a AMD64)
  - Akceptují:
    - dva operandy
    - registr
    - adresu v paměti

```
TSL R1, LOCK
```

```
Do registru R1 přečte hodnotu z umístění proměnné LOCK
a následně do ni zapíše původní obsah R1
```

  - Instrukce tyto dva úkony provede atomicky
- TSL dostatečně silný prostředek pro implementaci složitějších primitiv
  - zajistí podmínku vzájemného vyloučení nezávisle na konfiguraci stroje

- Instrukce test and set (TSL)
  - Instrukce podporované přímo v instrukční sadě (x86 a AMD64)
  - Akceptují:
    - dva operandy
    - registr
    - adresu v paměti

```
TSL R1, LOCK
```

```
Do registru R1 přečte hodnotu z umístění proměnné LOCK
a následně do ni zapíše původní obsah R1
```

  - Instrukce tyto dva úkony provede atomicky
- TSL dostatečně silný prostředek pro implementaci složitějších primitiv
  - zajistí podmínku vzájemného vyloučení nezávisle na konfiguraci stroje

TSL a vytvoření zámku

```
LOCK:
MOV R1, 1        //Vlákno chce obsadit kritickou sekci
TSL R1, L        //Atomická výměna obsahu registru a proměnné L
CMP R1, 0        //Porovnání hodnoty registru R1
JNE LOCK         //Pokud je R1 nenulový, skoč na návěstí LOCK
RET              //Návrat z procedury

UNLOCK:
MOV R1, 0        //Informuj čekající, že je kritická sekce volná
MOV L, R1        //Atomická výměna obsahu registru a proměnné L
RET              //Návrat z procedury
```

- Proměnná L – obsazená/volná kritická sekce
- Při vstupu do kritické sekce vlákno volá rutinu LOCK
  - Hodnota L = 0 – volná kritická sekce – RET
  - Hodnota L < > 0 – skok na začátek – vlákno uvízne v proceduře LOCK dokud L nebude 0
  - Práce ve smyčce - SPINLOCK
- Při opuštění kritické sekce volá vlákno rutinu UNLOCK

Čekání na vstup do kritické sekce

- Aktivní
  - Při obsazení kritické sekce je vykonáván kód v krátké smyčce
    - čekání vytěžuje procesor
    - přechod do stavu čekání a zpět s téměř nulovou režií
    - efektivní pro krátký kód v kritické sekci
  - Spinlock
- Pasivní
  - Otestuje, zda je kritická sekce volná – pokud ne USNE
    - nevytěžuje procesor
    - při opuštění kritické sekce nutno ověřit, zda nějaké vlákno nečeká na vstup
      - pokud ANO musí jej explicitně probudit
    - usínání/probouzení realizováno interními rutinami plánovače
      - Výkonově náročnější než aktivní čekání
    - vhodné pro komplexnější kódy – řádově stovky mikrosekund a více
  - Binární semafor, Mutex

Problém výrobce a spotřebitele

- Úloha producer-customer problem
  - Uvažujeme seznam výrobků a dvě vlákna
    - vlákno `Producer` – vytváří nové výrobky
    - vlákno `Consumer` – odebírá zpracované výrobky
    - nejsou kladeny žádné požadavky na rychlost produkce a zpracování
    - do seznamu se vejde konečný počet položek `N`
- Krajní případy
  - Výrobce pracuje rychleji než spotřebitel
    - seznam dosáhne N položek
    - výrobce nemůže ukládat nové položky
    - výrobce musí pozastavit činnost
  - Spotřebitel pracuje rychleji než výrobce
    - seznam je zcela vyprázdněn
    - spotřebitel musí pozastavit činnost

Návrh řešení P-C problému

```
VAR
Count: Integer = 0; …
Procedure Producer;
VAR V: TItem;
begin
While true do
   begin
   V := ProduceItem;
   If (Count = N) Then
        Sleep;
   Insert(V);
   Inc(Count);
   If (Count = 1) Then
        Wakeup (Consumer);
   end;
end.
```

```
Procedure Consumer;
VAR V: TItem;
begin
While true do
   begin
   If (Count = 0) Then
        Sleep;
   V:=RemoveFirst;
   Dec(Count);
   If (Count = N - 1) Then
        Wakeup (Producer);
   end;
end.
```

Návrh řešení P-C problému

- Popis řešení
  - Výrobce i spotřebitel pracují v nekonečné smyčce
  - `Producer`
    - v každém cyklu vytvoří předmět
    - otestuje, zda jej může přidat do seznamu
    - pokud ne (seznam obsahuje N položek) – čeká na signál od `Consumer`
  - `Consumer`
    - otestuje zda seznam obsahuje nezpracovaný výrobek
    - pokud ne čeká na signál od `Producer`
    - pokud ano – odebere jeden ze seznamu – podívá se zda seznam měl velikost N.
    - pokud ano – posílá signál `Producer (ten dříve zjistil, že nemůže přidat nové výrobky a usnul)`
    - `Consumer` zpracuje předmět

- Nekorektní běh - možné problémy
  - `Consumer` otestuje, zda je seznam výrobků prázdný – dostane kladnou odpoveď, protože na počátku celého procesu `Producer` ještě žádný předmět nevytvořil.
  - Plánovač rozhodne, že vlákno `Consumer` již vyčerpalo svůj procesorový čas a přeplánuje jej vláknem `Producer`
  - `Producer` vytvoří nový předmět a přidá jej do seznamu. Porotže byl předtím seznam prázdný, `Producer` se též pokusí probudit spotřebitele. Vyslaný signál vyjde nazmar, jelikož spotřebitel ještě neusnul.
  - Na procesor je opět naplánován `Consumer`
  - Spotřebitel si ze svého minulého běhu pamatuje, že seznam je prázdný (což nyní již není pravda) a proto usne.
  - `Consumer` produkuje nové a nové výrobky, které přidává do seznamu
  - Po přidání N-tého výrobku usne i výrobce
  - Obě vlákna tak budou nekonečně dlouho spát

Návrh řešení P-C problému

- Řešení nekorektního běhu od E. W. Dijkstera
  - Vláknu přidán *probouzecí bit*
    - pokud procedura `Wakeup` probouzí aktivní vlákno, nastaví mu probouzecí bit na 1
    - rutina `Sleep` kontroluje hodnotu probouzecího bitu
    - Vhodné jen pro jednoho `Consume` a jednoho `Producer`.
  - Vlákno si informaci, že jej někdo probouzel dříve než usnulo pamatuje v celočíselné proměnné
    - kapacita 2^{32}-1
  - Vznik nového synchronizačního primitiva - SEMAFOR

- Semafor
  - Čítač s operacemi `Down` (rutina `Sleep`) a `Up` (rutina `Wakeup`)
  - `Down`
    - kontroluje hodnotu čítače
    - pokud < > 0 snižuje o 1
    - pokud = 0 uspí aktuální vlákno dokud někdo čítač nenavýší
    - vše probíhá jako atomická operace – nikdo jiný nemůže se strukturou semafor pracovat
  - `Up`
    - probíhá atomicky
    - kontroluje, zda nějaká vlákna čekají zablokována
    - pokud ano vybere jedno z nich a probudí jej
    - pokud žádné nečeká – operace Up pouze zvýší hodnotu čítače o 1

Semafor

```
Procedure Down (Semaphore s); Atomic;
begin
If (s.counter = 0) then
   Sleep(s);
else Dec(s.counter);
end;

Procedure Up (Semaphore s); Atomic;
begin
If (s.counter > 0) then
   Wakeup(s);
else Inc(s.counter);
end;
```

- Procedury Sleep a Wakeup za využití parametru rutiny umožňuí specifikovat, na kterém semaforu chce vlákno čekat, nebo odkud se má vybírat kandidát na probuzení.
- Zároveň spravují aktuální počet spících vláken udávaných položkou **`speeping`**.

Řešení problému výrobce a spotřebitele

```
Var
Empty: Semaphore = N;
Full: Semaphore = 0;
Procedure Producer;
VAR V: TItem;
begin
While true do
   begin
   V := ProduceItem;
   Down(Empty);
   Insert(V);
   Up(Full);
   end;
end.
```

```
Procedure Consumer;
VAR V: TItem;
begin
While true do
   Down(Full);
   V := RemoveFirst;
   Up(Empty);
   Cosumer(V);
    end;
end.
```

- Řešení využívá dva semafory `Full` a `Empty`
  - Součet hodnot jejich čítačů se rovná `N`
  - Čítač `Full` udává počet výrobků v seznamu
  - Čítač `Empty` udává počet volných slotů
  - Semafory pracují proti sobě
- Díky vlastnostem semaforu je `Producer` zablokován, chce-li přidat položku do zaplněného seznamu, dokud nevznikne prázdné místo
- Při synchronizaci často najdeme využití semaforu, jehož čítač může nabývat pouze hodnot 0 a 1. Tyto synchronizační primitiva se označují jako *binární semafor* nebo *mutex*.

- Úloha readers-writers problem
  - V kritické sekci se v jednom okamžiku může nacházet libovolný počet čtenářů, nebo nejvýše jeden zapisovatel
  - Čtenář označuje vlákna, která chtějí sdílená data pouze číst
  - Zapisovatel označuje vlákna, která plánují data číst a měnit

Řešení R-W problému

```
Var
   Mutex: Semaphore = 1;
   Open: Semaphore = 1;
ReadersCount: Integer = 0;
Procedure Reader;
Begin
Down(Mutex);
If (ReadersCount = 0) then
   Down(open);
Inc(ReadersCount);
Up(Mutex);
ReadData;
Down(Mutex);
Dec(ReadersCount);
If (ReadersCount = 0) then
   Up(Open);
Up(Mutex);
end;
```

```
Procedure Writer;
Begin
Down(Open);
WriteData;
Up(Open);
end;
```

- `Writer` – zajišťuje podmínku jedinečnosti zapisovatele za využití binárního semaforu (`open`) s počáteční hodnotou 1 (volná kritická sekce)
- `ReadersCount` – aktuální počet čtenářů

Nevýhody řešení R-W problému

- Nespravedlivé vůči `Writer`
  - Pokud v každém okamžiku je v kritické sekci vlákno `Read`, `Writer` se nikdy nedostane přes operaci `Down`.
- Řešení spočívá v implementaci reader-write lock
  - datovou strukturu lze zamykat pro čtení i pro zápis
  - zamykání probíhá na vstupu do kritické sekce
- Implementace
  - `TRWLock` – zámek reader-writer
    - binární semafor `Lock` a `Open`
    - celočíselná proměnná `ReadersCount`
      - `Open` – identifikace sekce prázdná/plná
      - `Lock` – zajištění nedělitelnosti operací
      - `RWLockInit` – inicializace struktury TRWLock
        - reprezentace odemčeného zámku – nastaveni semaforů Open a Lock na 1 a počet čtenářů na 0

Implementace reader-write lock

```
Type
TRWLock = Record;
    Lock: Semaphore;
    Open: Semaphore;
    ReadersCount: Integer;

Procedure RWLockInit(Var Z:TRWLock);
Begin
Z.Lock:=1;
Z.Open:=1;
Z.ReadersCount :=0;
end;

Procedure RWLockLockReader(Var
    Z:TRWLock);
Begin
Down(Z.Lock);
If (Z.ReadersCount = 0) then
    Down(Open);
Inc(Z.ReadersCount);
Up(Z.Lock);
end;

Procedure RWLockUnLockReader(Var
    Z:TRWLock);
Begin
Down(Z.Lock);
Dec(Z.ReadersCount);
If (Z.ReadersCount = 0) then
    Up(Z.Open);
Up(Z.Lock);
end;

Procedure RWLockLockWrite(Var Z:TRWLock);
Begin
Down(Z.Lock);
end;

Procedure RWLockUnLockWrite(Var
    Z:TRWLock);
Begin
Up(Z.Lock);
end;
```

Univerzita Hradec Králové
Fakulta informatiky a managementu

Děkuji za pozornost…