# Synchronizace procesů

V multiprogramových OS je potřeba zajistit sdílení jednoho prostředku (soubor, paměť) několika procesy. Pokud by totiž dva nebo více procesů například přistupovalo k jednomu souboru s tím, že by více procesů do něj zapisovalo, došlo by ke zhroucení celého systému. Proto se využívá synchronizace procesů, která se těmto kolizím snaží zabránit. Využívá k tomu přerušení v této posloupnosti:

1. Odebrání procesoru procesu - přerušení
2. Procesor se předá druhému procesu
3. Odebrání procesu procesoru – přerušení
4. Procesor se předá prvnímu procesu

**Postupový prostor**

proces P

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| proces Q |  |  |  |  |  |  |  |
| **D** |  |  |  |  |  |  |  |
| **KSQ** |  |  |  |  |  |  |  |
| **C** |  |  |  |  |  |  |  |
| **D** |  |  |  |  |  |  |  |
| **KSQ** |  | Postup procesů  Zakázaná oblast |  |  |  |  |  |
| **C** |  |  |  |  |  |  |  |
|  | **A** | **KSP** | **B** | **A** | **KSP** | **B** |  |
|  | Zakázaná oblast |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

obrázek postupová cesta, postupový prostor, zakázaná oblast

P, Q – paralelní procesy

A, B, C, D – příkazy

KSP, KSQ – kritické sekce – použití společného prostředku (disk, tiskárna, proměnná)

Př. Programu

cobegin

--P:

loop A;

KSP;

B;

end loop;

--Q:

loop C;

KSQ;

D;

end loop;

coend;

Vstup do zakázané oblasti znamená **časově závislou chybu,** v důsledku **nedeterministického** chování procesů (posloupnost akcí procesoru závisí na relativní rychlosti postupu procesu).

## Synchronizační metody

### Vzájemné vyloučení kritických sekcí

**Schéma pro dva procesy**

Postupový prostor

Z – zahájení

* zákaz přerušení
* nastavení společné proměnné

K – ukončení

* povolení přerušení
* nastavení společné proměnné

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Q** |  |  |  |  |  |  |
| **D** |  |  |  |  |  |  |
| **K** |  |  |  |  |  |  |
| **KSQ** |  |  |  |  |  |  |
| **Z** |  |  |  |  |  |  |
| **C** |  |  |  |  |  |  |
|  | **A** | **Z** | **KSP** | **K** | **B** | **P** |

přechod

**Schéma pro více procesů – Petriho síť**

stav

C

A

P

KSQ

KSP

B

D

**obrázek 22 vzájemné vyloučení procesů KSP a KSQ**

Nejvýhodnějším modelem paralelismu, ve kterém se snadno znázorňuje synchronizace více pro více procesů, jsou **Petriho sítě**. Petriho sítě lze zadat orientovaným ohodnoceným multigrafem, v němž se užívají dva typy uzlů: místa (znázorňují se kolečky) a přechody (znázorňují se čárkou kolmou na hrany. Místa a přechody se propojí orientovanými hranami tak, aby žádní sousední uzly nebyly téhož typu. Přechod mezi místy **p** a **q** je připraven, jestliže každé místo **p** obsahuje alespoň tolik teček, kolik vede z místa **p** do přechodu hran.

**Připravený přechod se může provést:**

* Z každého místa p se odebere tolik teček, kolik vede hran z místa p do přechodu.
* Do každého místa q se přidá tolik teček, kolik vede hran z přechodu do místa q.

### Producent – konzument

Asynchronní komunikace procesů pomocí společné paměti (asynchronní - procesy na sebe nečekají). Využívá se zásobníku, kam jeden proces posílá data a druhý je z něj čte. V případě většího počtu procesů se zvětší i počet teček v níže uvedeném schématu.

**Producent** – proces, který produkuje, nemůže produkovat, pokud je zásobník **plný**.

**Konzument** – proces, který odebírá. Nemůže odebírat z **prázdného** zásobníku.

buffer

1 (4, 7, …)

3 (6, 9, …)

2 (5, 8, …)

producent

konzument

|  |  |  |
| --- | --- | --- |
| 2 | 5 |  |

obrázek producent a konzument

15

Produkuj

22

Čti

4

volné

Spotřebuj

Zapiš

obsazené

obrázek producent a konzument v Petriho síti pro 3 čísla

Časově závislá chyba – konzument načte prázdnou hodnotu

– producent nezapíše do bufferu

### 

## Čtenáři a písař

* Používá se v databázových systémech.
* **Písař** je proces, který může zapisovat do společné datové struktury.
  + Operace tohoto zápisu je jedinečná.
  + Je pouze jeden
* **Čtenář** je proces, který může číst.
  + Jeho operace je vícenásobná.
  + Je jich více
* V případě více čtenářů se musí zvětšit také počet teček.

Písař zapisuje – nelze číst

Alespoň jeden čtenář čte – nelze zapisovat

Číst může více čtenářů současně

Písař

Čtenář 2

Čtenář 1

Čtení 2

Zápis

Čtení 1

**obrázek 25 čtenáři a písaři**

### Souběh

* Používá se v distribuovaném prostředí.
* Jeden proces vyčká na druhý a společně provedou určitou část programu.
* Zabudován v jazyce ADA.

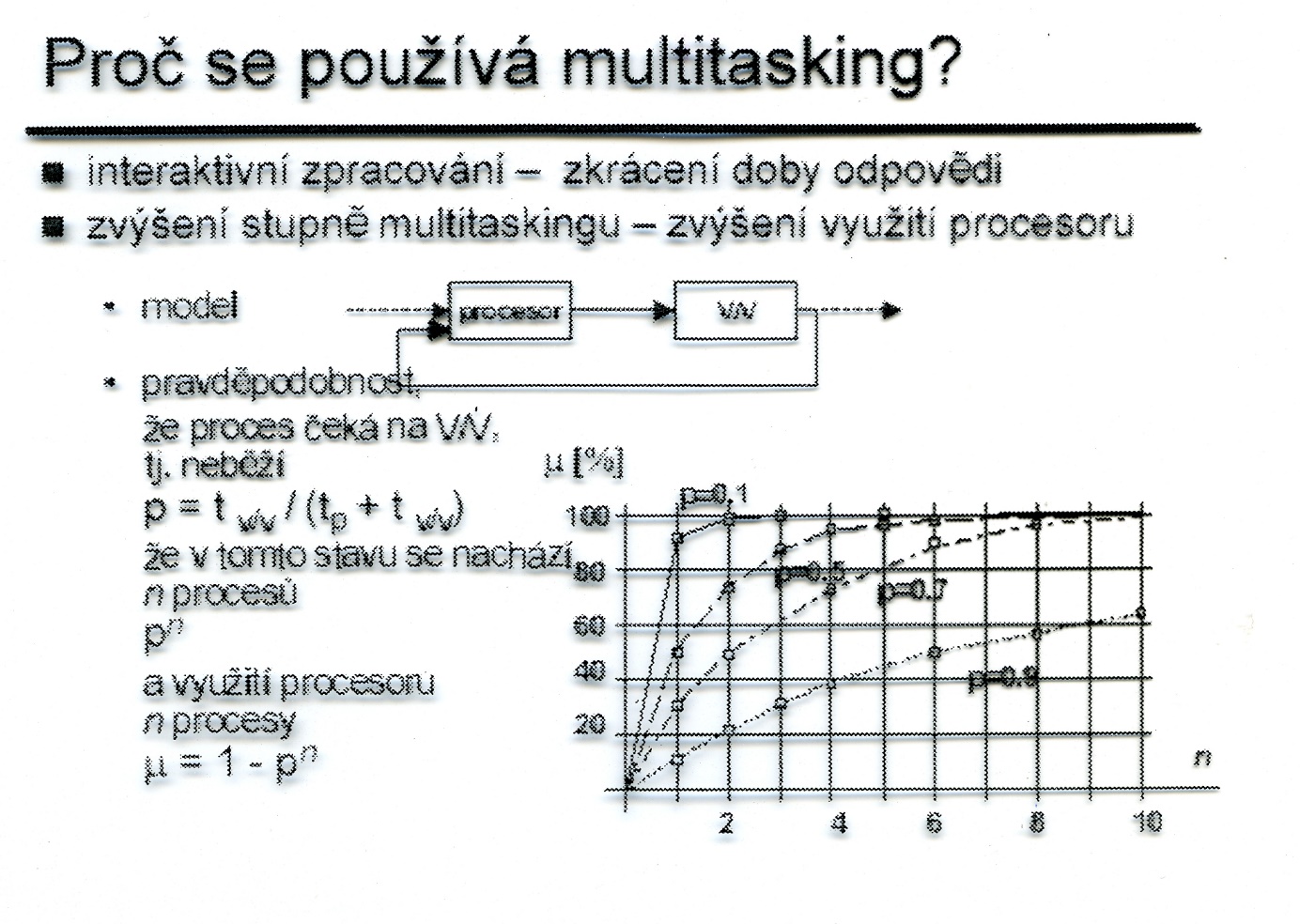
Q

P

společně

**obrázek 26 souběh**

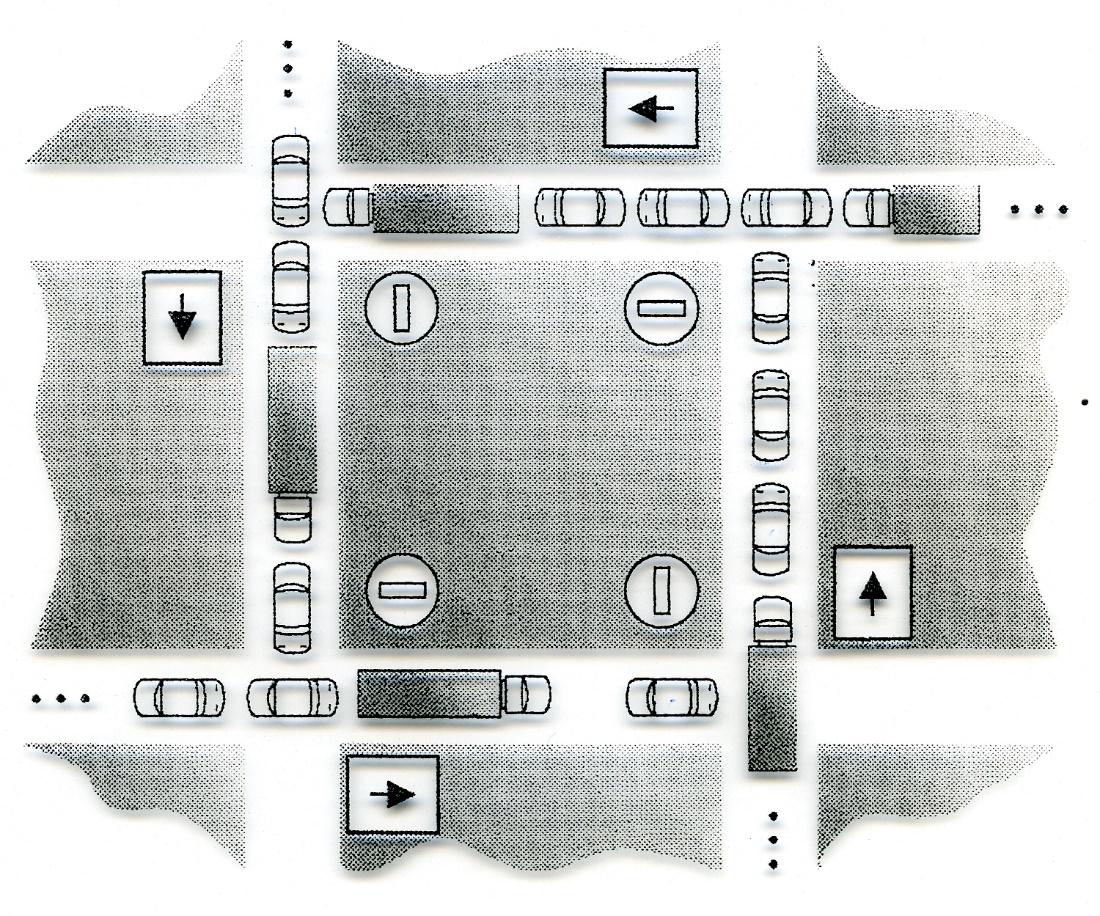
## Chyby v synchronizaci



**Stárnutí** – proces příliš dlouho čeká (a nemusí se dočkat)

**Uváznutí (**deadlock**)–** proces je zablokován a nemůže již dále pokračovat.

Množina procesů uvázla, jestliže každý z procesů v této množině čeká na událost, kterou může vyvolat pouze některý z procesů v této množině.



**Nutné podmínky uváznutí:**

1. Vzájemná výlučnost – prostředek smí používat nejvýše 1 proces a je jich méně než procesů.
2. Inkrementální požadavky – proces vlastnící prostředky požaduje další.
3. Nepředbíhatelnost – prostředky přidělené procesu musí on sám vrátit, nikdo mu je nemůže odebrat.
4. Cyklické čekání – vytvoří se cyklický řetěz posloupností procesů z hlediska čekání na prostředky.

1

2

3

A

B

C

1

2

3

proces

prostředek

obrázek nutné podmínky uváznutí

**Ochrana před chybami**

* 1. Můžeme v systému užit protokol, který zajistí, že *nikdy* deadlock nenastane.   
     (prevence – zrušení jedné z podmínek 1-4, obcházení – zabránění současného uplatnění 1-4)
  2. Můžeme systému dovolit, aby deadlock nastal a potom ho vyřešit (detekce a obnova).
  3. Můžeme celý problém deadlocku ignorovat a předstírat, že k němu nikdy nedojde. Toto řešení zatím užívá většina OS, včetně Unixu („pštrosí“ politika).

## Synchronizační nástroje

### Semafory

Používají se pro synchronizaci procesů pracujících nad stejnými daty. Základní operace jsou P (vstup) a V (výstup z kritické sekce).

Pokud jeden proces vstoupí do kritické sekce, nastaví se hodnota semaforu na 0 a ostatní procesy nemohou pokračovat. Semafor je proto řadí do fronty.

Při vystoupení procesu z kritické sekce se nastaví hodnota semaforu na nenulovou hodnotu. Do kritické sekce může vstoupit další proces. Přednost mají procesy ve frontě. Jejich vstupy a výstupy řídí semafor.

Řekněme, že máme mnoho spolupracujících procesů, které všechny čtou a zapisují záznamy

v jednom datovém souboru. Budeme zřejmě potřebovat přísně omezit přístup k souboru.

Můžeme použít semafor s počáteční hodnotou jedna a kolem kódu pro operaci se souborem umístit dvě semaforové operace: první bude testovat a dekrementovat hodnotu semaforu, druhá ji bude testovat a inkrementovat. První proces, který se pokusí o přístup k souboru, bude dekrementovat semafor a zdaří-li se mu to, bude mít semafor nově hodnotu 0. Tento proces nyní může pokračovat dále a používat datový soubor, pokud se však nyní pokusí o dekrementaci semaforu jiný proces, operace se nezdaří protože nová hodnota semaforu by byla -1. Druhý proces bude pozastaven do doby, než první proces inkrementuje hodnotu semaforu zpět na 1. Nyní je možno pozastavený proces probudit a tentokrát pokus o dekrementaci semaforu uspěje.

Každý objekt semaforu je popsán polem.

Příklad 1:

type Semafor = record S:word;

F:fronta procesů;

end;

procedure Init\_Sem(var SMF:Semafor;PH:word);

begin

SMF.S:=PH;

SMF.F:={prázdná};

end;

procedure **Wait**(var SMF:Semafor);

begin

if SMF.S=0 then {úmístění volajícího procesu do fronty SMF.F,

zde bude čekat}

else SMF.S:=SMF.S-1;

end;

-

+

SMF.S=0

úmístění volajícího procesu do fronty SMF.F,

zde bude čekat

SMF.S:=SMF.S-1

procedure Send(var SMF:Semafor);

begin

if SMF.F<>{prázdná} then {jeden proces vyjmout z fronty SMF.F

a uvolnit jej, dokončí pak operaci

Wait, ve které čeká}

else SMF.S:=SMF.S+1;

end;

-

+

SMF.F<>{prázdná}

jeden proces vyjmout z fronty SMF.F

a uvolnit jej, dokončí pak operaci

Wait, ve které čeká

SMF.S:=SMF.S+1

Příklad 2:

**Proces chce do kritické sekce**

Wait(Hlidej);

KSP;

3

Send(Hlidej); ;

3

2

kritická sekce

2

Vzájemné vyloučení procesů

1

1

0

0

315

44

konzument

producent

**Producent – Konzument**

const MaxPol=3;

var Buffer:array[0..MaxPol-1] of Typ\_polozky;

Obsazene, Volne:Semafor;

J, K:word;

Vst\_data, Vyst\_data:Typ\_polozky;

begin

J:=0;

K:=0;

Init\_Sem(Obsazene,0);

Init\_Sem(Volne,MaxPol);

cobegin

-- Producent

loop

{Produkuj položku Vst\_data}

Wait(Volne);

Buffer[J]:=Vst\_data;

J:=(J+1) mod MaxPol;

Send(Obsazene);

end loop;

-- Konzument

loop

Wait(Obsazene);

Vyst\_data:=Buffer[K];

K:=(K+1) mod MaxPol;

Send(Volne);

{Spotřebuj položku Vyst\_data}

end loop;

coend;

end;

### 

### Zasílání zpráv

Při zasílání vnikají tyto operace: **send\_Message(…)** – zaslání zprávy, kdy jeden proces zašle operaci (posloupnost bitů) – zpráva a druhý proces ji příjme **recive\_Message(…)** – přijetí zprávy.

**Adresace**

* symetrická – uvádí se jméno přijímacího i vysílacího procesu
* asymetrická – uvádí se pouze jméno přijímacího procesu, jméno vysílajícího se určí až po přijetí zprávy
* nepřímá – uvádí se jméno vyrovnávací paměti nebo komunikačního kanálu

**Asynchronní zasílání zpráv**

Recive\_Message

Send\_Message

Přijetí

Vysílání

Zapsáno v paměti

obrázek asynchronní zasílání zpráv

Při symetrické a asymetrické adresaci je vyrovnávací paměť poskytnuta systémovými prostředky (schránka). Může obsahovat i více zpráv, kapacita je ale omezena.

**Příklad:** Řešení úlohy Producent – Konzument pomocí asynchronního zaslání zpráv se symetrickou adresací.

-- Producent – konzument:

cobegin

-- Producent;

declare P1:Typ\_polozky;

begin

loop

Send\_Message(Konzument, konverze na zprávu (P1));

end loop;

end Producent

-- Konzument

declare P2:Typ\_polozky;

begin

loop

Recive\_Message(Producent, konverze na zprávu (P2));

end loop;

end Konzument

coend;

**Synchronní zasílání zpráv**

Příjem

Vysílání

Recive\_Message

Send\_Message

Čekej potvrzení

Vyšli potvrzení

obrázek synchronní zasílání zpráv

### Monitory

* Synchronizační nástroje obsažené ve vyšších programovacích jazycích, například MODULA.
* Monitor obsahuje deklaraci společné proměnné s definicí všech operací, které lze na ní provádět a jsou vzájemně vyloučeny.

M:monitor

Cislo,jmeno;

procedure PRG1(...);

begin

...

end;

procedure PRG2(...);

begin

...

end;

.

.

.

procedure PPRG(...);

begin

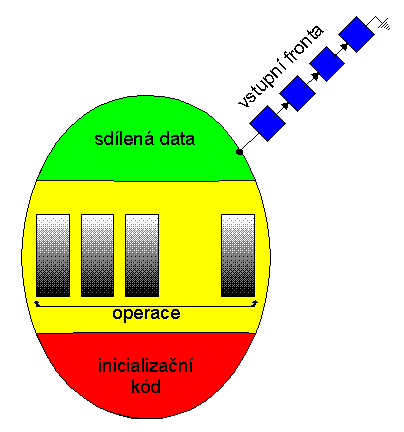
...

end;

begin

inicializace pr1...prm;

end;



Konstrukce monitoru zajišťuje, ze pouze jeden proces může být v daném čase na monitoru aktívní, takže programátor se nemusí zabývat synchronizačním kódem. Je patrné, že definice monitoru je v mnoha ohledech podobná kritické oblasti a stejně jako existují podminěné kritické oblasti, tak pro monitor je třeba definovat podmíněnost.