# Semafory

Synchronizační primitivum
Dále budeme mluvit o procesech (obecně paralelních entitách), ale kromě synchronizace procesů můžeme samozřejmě procesy použít pro synchronizaci vláken

## Z čeho se skládá semafor?

Celočíselný čítač **s**

*Hodnotu lze semaforu přiřadit pouze při deklaraci, pak se mění jen voláním operací **P** a **V***

- Binární semafor – může nabývat pouze hodnot 0, 1
  - Použití pro ***vzájemné vyloučení***
    výhodou – kontrola, aby semafor nenabyl hodnot 2,3,.. které by umožnili přístup vice procesům/vláknům zároveň
- Obecný semafor – hodnoty 0,1,2 ..
  - Navíc použití pro ***synchronizaci***, např. buffer omezené velikosti N (ProdKonzum)

Operace nad semaforem P, V

- Operace **P(s)**
  - Pokud je S > 0, sníží S o 1, operace P tím končí a náš proces, který ji vyvolal může pokračovat dále
  - Pokud je S = 0, náš proces se zablokuje. *V životním cyklu procesu se ze stavu **běžící** dostaneme do stavu **blokovaný**, dokud nás nějaká událost neodblokuje, pak přejdeme do stavu **připravený**.*
  - Pokud byl tedy náš proces zablokovaný, a dojde externí událostí časem k jeho odblokování (např. jiný proces zavolal V na stejném semaforu a vzbudil náš proces), operace **P** skončí a náš proces, který zavolal operaci **P** pokračuje dále; hodnota semaforu **s** zůstává na nule

- Operace **V(s)**
  - Pokud žádný proces není ve stavu **blokovaný nad semaforem s** (viz životní cyklus procesu), zvětšíme hodnotu semaforu **s** o *jedna* a operace **V** končí
  - Pokud nějaký proces byl zablokovaný nad semaforem **s**, dojde ke vzbuzení tohoto procesu (přesune se z blokovaný do připravený) a naše operace V končí
    u **spravedlivých** semaforů se vzbudí proces, který je první na řadě (nejdéle čeká na semafor s)

## Implementace

Z hlediska implementace semaforu si můžeme představit, že kromě čítače s (celé číslo), je semafor z hlediska datových struktur tvořen i **frontou procesů, čekajících na daný semafor**, která je ze začátku prázdná a blokováním při operaci **P** se může plnit. Stejně tak operace **V** z této fronty procesy může odebírat. Procesy by byly ve frontě reprezentovány svým PID (process id) – číslem, de facto klíčem do tabulky procesů, kde je o nich podrobnější informace.

## Příklad:

*Semafor s má hodnotu 1 a náš proces P1 zavolá operaci p(s).*
Operace p sníží hodnotu semaforu s na nula a skončí, tedy náš proces P1 může pokračovat dále.

*Semafor s má hodnotu 0 a nějaký proces P2 zavolá operaci p(s).*
Tentokrát operace p zjistí, že hodnota semaforu s je 0, a proces P2 se tedy zablokuje.
Pokud časem nějaký jiný proces zavolá operaci v(s), dojde k odblokování procesu P2, který bude moci pokračovat dále.

## Klasický způsob ošetření kritické sekce:

```
Semaphore s = 1;          // počáteční hodnota 1, aby vůbec někdo mohl vstoupit

P(s);   // vstup do kritické sekce

X++;   // kód představující kritickou sekci, kterou smí vykonávat pouze 1 proces současně

V(s);    // výstup z kritické sekce
```

*Mnemotechnická pomůcka:*
*Pokud se Vám pletou názvy operací P,V tak vězte, že pro ošetření kritické sekce je použijeme v abecedním pořadí ☺*

## Výhoda semaforu

Oproti *aktivnímu čekání*, kdy je proces neustále plánován jenom kvůli tomu, aby se podíval, že např. hodnota proměnné KUK je stále nula a nedělá nic užitečného, při využití semaforu dojde v operaci P k **zablokování** procesu, tento tedy není plánován, **šetří se zdroje** systém (např. čas procesoru) a až ve vhodný okamžik dojde k jeho vzbuzení (někdo zavolá operaci V(s)).

Srovnání s realitou:
*Představte si 24hodinovou pohotovostní službu lékaře. Pokud bude moci spát, a vzbudí jej, až je potřeba řešit případ, je to účelnější, než když stráví celých 24 hodin sledováním dveří, zda nepřichází pacient..*

## Více kritických sekcí

(tj. např. nad různými společnými proměnnými x, y, z) – můžeme využít vice semaforů.

Výhoda:

- Přístup do jedné kritické sekce (X) neblokuje přístup do jiné (Y)
- Tedy větší efektivita

Nevýhoda:

- Nesmíme se splést a v jednom procesu ošetřovat stejnou kritickou sekci X semaforem s1 a v druhém procesu semaforem s2 – přístup by nebyl chráněn

Stejná kritická sekce musí být v procesech chráněna stejným semaforem
KS1, např. přístup k proměnné x: semafor s1 v procesech P1 i P2 i P3
KS2, např. přístup k proměnné y: semafor s2 v procesech P1 i P2 i P3

## Kde se semafor vezme?

Semafor může být poskytován operačním systémem či programovou knihovnou.

Např. Java viz *java.util.concurrent*

Semaforem můžeme synchronizovat jak procesy, tak vlákna.

## A co uvíznutí?

I to se může stát, operace P a V nad různými semafory mohou být rozházené po celém kódu programu, stačí se přepsat, viz následující příklad:

```
Semaphore s = 1;

P(s);

  X++;   // kritická sekce

P(s);    // zde mělo být správně V(s); hodnota s je 0, pokud nikdo jiný nezavolá V(s), tak se z této
           funkce P nevrátíme
```

Zde jsou výhodou monitory – synchronizační věci jsou „na jednom místě“, snažší kontrola, že kód je napsán tak, jak bylo zamýšleno.

*Tento text by měl sloužit jako zopakování znalostí o monitorech, které byste měli mít.*
*Případné komentáře k textu prosím na moji e-mailovou adresu, děkuji, L. Pešička.*

*Verze: 1.0*