Semafory

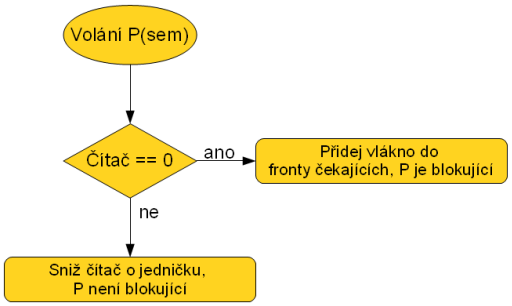
Semafor je v informatice široce používané synchronizační primitivum, které obsahuje celočíselný čítač. Semafor se využívá zejména jako ochrana proti souběhu tím, že chrání přístup do kritické sekce, k čemuž používá dvojici operací V (*up*) a P (*down*). (zdroj*: http://cs.wikipedia.org/wiki/Semafor_(synchronizace)*)

Atomická operace P(proberen, někdy označováno jako down)

Test stavu čítače:

1) rovno nule - operace P je blokující, vlákno, které volalo P se přidá do fronty čekajících,
2) různé od nuly – sniž čítač o jedničku a vpusť vlákno do kritické sekce.

Zakresleno pomocí vývojového diagramu



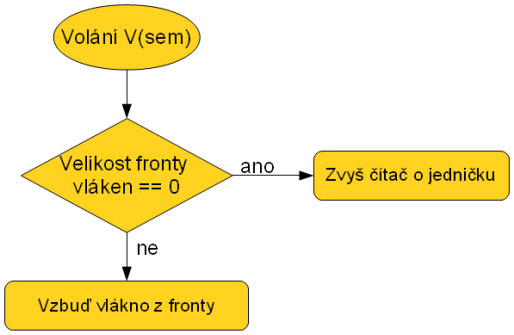
Atomická operace V (verhogen, někdy označováno jako up)

Test velikosti fronty čekajících vláken:

1) velikost rovna nule – operace V zvýší čítač o jedničku,
2) velikost různá od nuly – operace V „vzbudí“ vlákno čekající ve frontě. To může vstoupit do fronty.

Čítač je možné si představit jako omezení počtu procesů, které mohou zároveň vstoupit do kritické sekce nebo například jako počitadlo volných prostředků. Tato implementace neodstraňuje problém aktivního čekání.

Zakresleno pomocí vývojového diagramu



Př. Synchronizace (viz cvičení):

Vlákno č.1 umí vypsat řetězce "Ahoj, " a "je " (volání funkcí print("Ahoj ") a print("je")).

Vlákno č.2 umí vypsat řetězce "dnes, " a "krásně." (volání funkcí print("dnes ") a print("krásně.")).

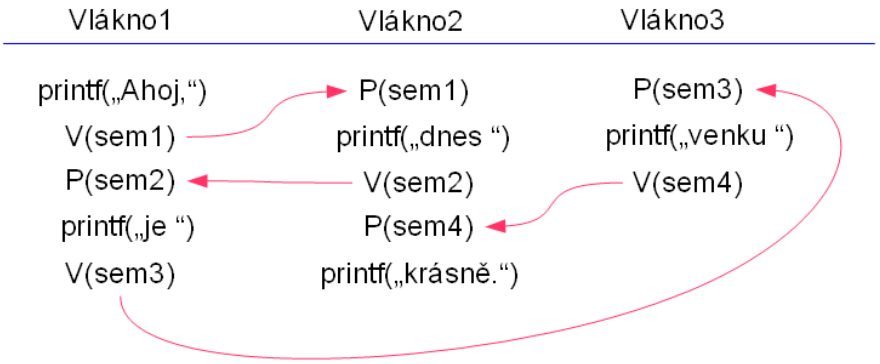
Vlákno č.3 umí vypsat řetězec "venku " (volání funkce print("venku ")).

Vlákna běží paralelně, ošetřete SEMAFORY, aby vždy byla vypsána správně věta:

Ahoj, dnes je venku krásně.

Nejprve pomocí 4 semaforů:

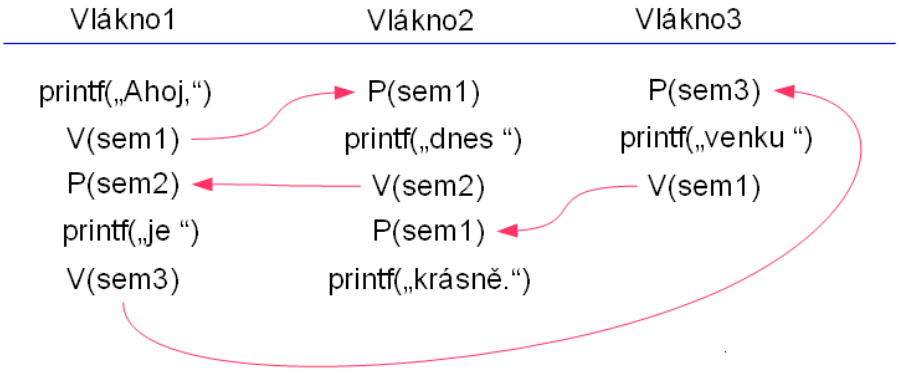
```
semaphore sem1 = 0; semaphore sem2 = 0; semaphore sem3 = 0;
semaphore sem4 = 0;
```



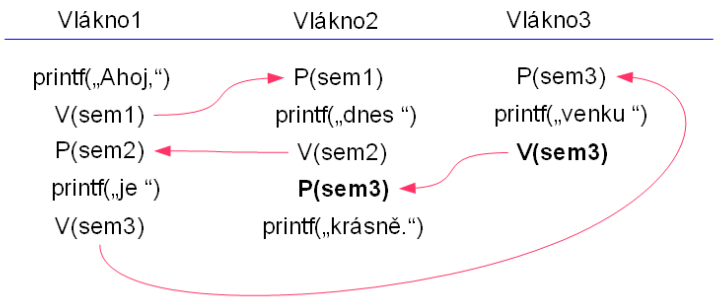
Dále popsané případy slouží k hlubšímu pochopení problematiky semaforů a synchronizace. Tyto případy jsou nad rámec předmětu ZOS, nicméně jejich pochopení by se mohlo hodit v dalším studiu.

Příklad lze řešit i pomocí 3 semaforů: (jeden semafor použijeme dvakrát)

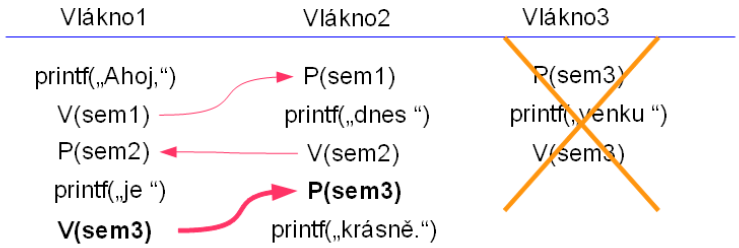
```
semaphore sem1 = 0; semaphore sem2 = 0; semaphore sem3 = 0;
```



Semafor sem4 jsme nahradili semaforem sem1. Zamysleme se nyní, zda není možné nahradit semafor sem4 semaforem sem3. Příklad by vypadal takto:

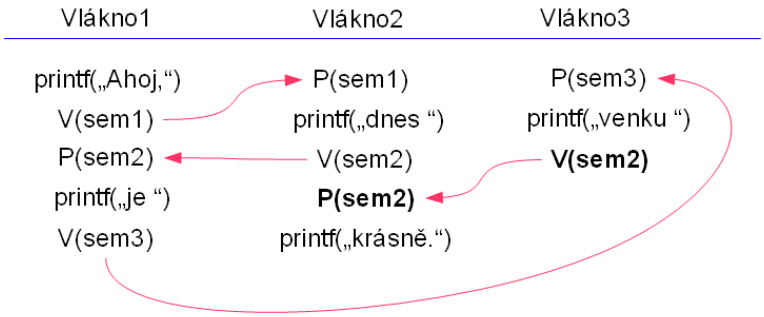


V tomto případě je chovaní vláken nepředvídatelné. Předpokládejme, že Vlákno3 nebude z nějakého důvodu naplánováno plánovačem (nejprve se provedou Vlákno1 a Vlákno2 a až po nějaké době kód Vlákna3). Obrázek takového chování by vypadal takto:

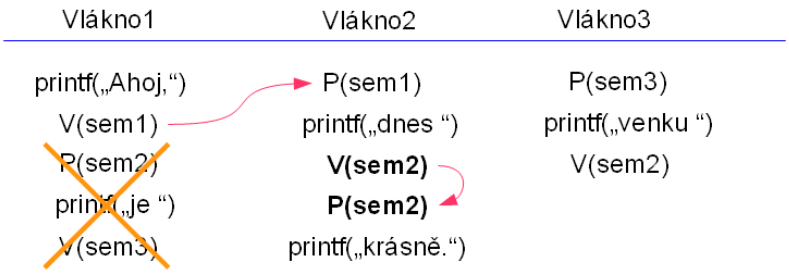


Jak je vidět, tak volání **V(sem3)** z Vlánka1 způsobilo probuzení Vlákna2, které bylo blokováno funkcí **P(sem3)**. Po naplánování Vlákna3 se toto vlákno uspí funkcí P(sem3), ovšem nikdo již nezavolá V(sem3). Dojde tedy k **uvíznutí.** Vlákno3 bude v tomto případě blokovat celý program.

Zamysleme se nyní, co by se stalo, pokud bychom nahradili původní semafor sem4 semaforem sem2. Příklad by vypadal takto:



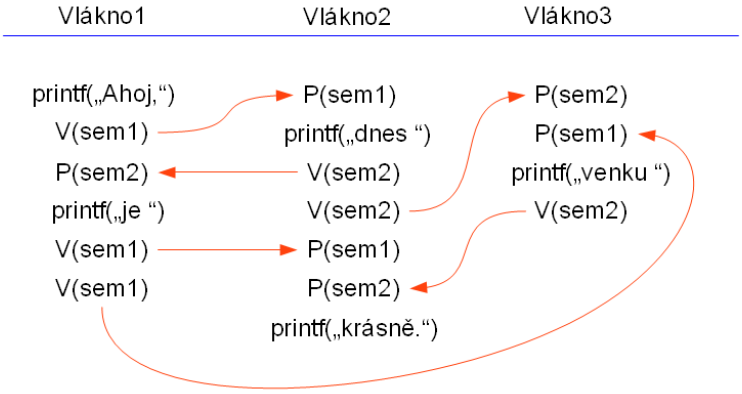
I zde může dojít k **uvíznutí**. Pro uvíznutí stačí předpoklad, že Vlákno1 bude uspáno po provedení druhého příkazu (konkrétně **V(sem1)**). Uspání bude na delší dobu, Vlákno2 a Vlákno3 mezi tím provedou své příkazy. Obrázek takového chování by vypadal takto:



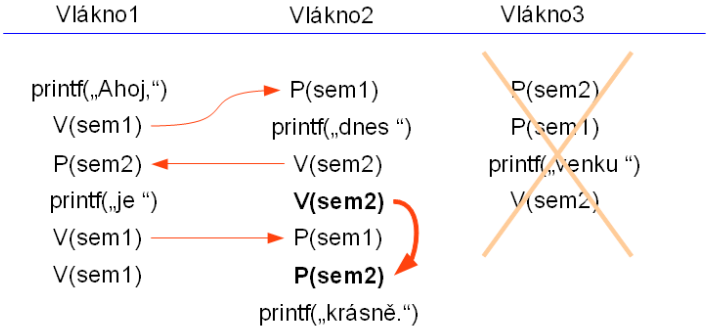
Jak je vidět z obrázku, tak pouze Vlákno2 vykonalo všechny své příkazy. Vlákno1 a Vlákno3 se po naplánování uspí a následně budou blokovat chod programu.

Ke stejným problémům dojdeme, pokud bychom chtěli celý příklad řešit pomocí 2 semaforů. Nástin řešení je vidět na následujícím obrázku. Předem podotýkám, že příklad zřejmě nelze řešit pomocí 2 semaforů, ke správnému chodu programu je potřeba nejméně 3 semaforů.

```
semaphore sem1 = 0; semaphore sem2 = 0;
```



Pokud v tomto příkladě budeme uvažovat, že Vlákno3 bude naplánováno později než Vlákno1 a Vlákno2 (Vlákno1 a Vlákno2 vykonají všechny své příkazy), dojde k **uvíznutí**. Obrázek tohoto stavu vypadá takto:



V tomto případě dojde k **uvíznutí**. Po naplánování Vlákna3 se toto vlákno uspí po volání P(sem2). Program nemůže dále pokračovat, protože vlákno se z tohoto stavu nedostane.