Synchronizace bez čekání

- pokud programujeme aplikaci pro multiprocesor, rádi bychom maximálně využili dostupný paralelismus - aplikaci strukturujeme jako **více vláken**

- potřebujeme, aby vlákna strávila co nejvíce času výpočtem, nikoli **čekáním na zámek** chránící sdílené datové struktury

- pokud se jedno z vláken zpozdí v kritické sekci (z důvodu výpadku stránky, přeplánování apod.), zdrží to i všechna ostatní vlákna čekající na zámek

- př. máme jednoduchou třídu, jako je sdílený čítač
  - dovolí nastavení hodnoty,
  - zjištění stavu
  - inkrementaci
- s HW podporou moderních CPU vytvořit spolehlivou wait-free implementaci i bez zamykání

Instrukce CAS – Compare and Swap

moderní CPU atomická instrukce typu compare-and-swap, CAS

- např. Intel - rodina instrukcí cmpxchg
- operace CAS má 3 operandy: místo v paměti (V), očekávanou původní hodnotu (A), novou hodnotu (B)
- pokud paměťové místo V obsahuje očekávanou hodnotu A, zapíše se do něj B; jinak nedělá nic
- vrací vždy původní hodnotu paměťového místa V

Použití instrukce CAS:

- přečteme hodnotu A z paměťového místa V
- na základě A spočteme novou hodnotu B
- pokud se hodnota A zatím nezměnila, pomocí CAS změníme hodnotu V z A na B

Pomocí instrukce CAS můžeme implementovat atomický čítač:

```
program waitfree;

atomic function
           CAS(var V:integer; A,B:integer): integer;
begin
     CAS:=V;
     If V=A then V:=B
end;


var x: integer := 0;

procedure a;
var i, a, r: integer;
begin
     for i:=1 to 100 do
        begin
          repeat
                a:=x;
                r := CAS(x, a, a+1);
          until r=a                { když r = a, končí }
        end
   end;


 begin
    cobegin
         a; a
    coend;
    writeln('x=', x)
 end.
```

implementace je wait-free, tj. zaručuje, že spočte správnou hodnotu v konečném počtu kroků bez ohledu na činnost ostatních vláken

- wait-free algoritmy existují pro práci s mnoha datovými strukturami
- výhody: nemůže nastat uvíznutí, nemůže nastat problém inverze priorit, při velkém soupeření o přístup k proměnné je levnější
- nevýhody: vyžadují HW podporu, implementace algoritmů je podstatně složitější (univerzální HW primitiva jsou drahá, levná primitiva nejsou univerzální)
- uživatelé většinou nepoužívají přímo, ale pomocí knihovních fcí apod. např. Java - třídy z java.util.concurrent,
  např ConcurrentLinkedQueue

http://java.sun.com/j2se/1.5.0/docs/api/java/util/concurrent/atomic/package-summary.html

package java.util.concurrent.atomic

A small toolkit of classes that support lock-free thread-safe programming on single variables. … provide an atomic conditional update operation of the form:

```
boolean compareAndSet(expectedValue, updateValue);
```

This method (which varies in argument types across different classes) atomically sets a variable to the `updateValue` if it currently holds the `expectedValue`, reporting `true` on success.