# Vlákna

## Co je to vlákno?

Hierarchie z pohledu operačního systému:

- Proces
  - největší výpočetní entita plánovače
  - vlastní prostředky, paměť a další zdroje
  - v závislosti na OS možnost preemptivního multitaskingu

- Vlákno Thread
  - každý proces má alespoň jeden, primární, thread
  - jeden proces může mít několik vláken
  - vlákna sdílí adresový prostor procesu a jeho zdroje
  - každé vlákno má svůj vlastní kontext (id, registry, prioritu, atd.)
  - v závislosti na OS možnost preemptivního multithreadingu

- Vlákno Fiber
  - Fiber – MS terminologie
  - Fiber je analogií threadu k procesu
  - Fiber plánuje některý thread procesu, ne plánovač operačního systému
  - Fiber běží v kontextu threadu, který ho naplánoval
  - Má pouze stav, zásobník a specifická data – např. prioritu má pouze thread
  - Fiber může ukončit běh vlákna v jehož kontextu běží
  - Ukončení aktivního fiberu jiným fiberem není OK – stack corruption => abnormal termination

- Fibre nevyužívá preemptivního multithreadingu OS, proces musí zajistit, že se fiber vzdá procesoru
  - kooperativní multithreading
  - Light-Weight Processes
    - Unix System V a Solaris
    - několik LWP běží v uživatelském adresovém prostoru ve strojovém času přiděleném vláknu jádra – tj. nad jedním vlákem jádra běží několik LWP
      - a nad jedním LWP může běžet několik user-level vláken
    - možné díky sdílenému bloku paměti, který vlastní proces
    - Odpadá režie přepínání úrovně oprávnění user-kernel => KIV/OS

- Může být, i nemusí, implementováno za podpory operačního systému
  - způsob, kterým aplikace může použít specifický systém plánování, který je pro ni výhodnější než ten, který poskytuje operační systém
  - obecně vzato, pro malý počet vláken použití fibers neposkytuje výhodu proti threadům
  - http://msdn2.microsoft.com/en-us/library/ms686919.aspx

- Implementace využívající podpory OS
  - Native Posix Thread Library pod Linuxem
  - Light-Weight Kernel Threads u některých verzí BSD
  - MS SQL Server 2000 http://msdn2.microsoft.com/en-us/library/Aa175393(SQL.80).aspx

- Implementace nepoužívající podpory OS
  - GNU Portable Threads
  - FSU PThreads – použití v Adě

- Hybridní
  - Native Posix Thread Library u NetBSD, některé Linuxové releasy a PM2 (Parallel Machine)
    - mechanismus „Scheduler/Linux Activations“
    - N:M – M skutečných vláken jádra a na každém z nich N aplikačních vláken

# Kdy se vlákna používají

- Obsluha periferních zařízení
  - U některých zařízení je třeba periodicky testovat stav hardware
  - Vláknu pak nemusí zbývat mnoho času na obsluhu uživatelského rozhraní
  - Jedno vlákno pro komunikaci s uživatelem a druhé obsluhuje hardware
  - Simulace časovače

- Síťová komunikace
  - Jedno vlákno akceptuje příchozí komunikaci
  - Jedno vlákno odesílá data
  - Jedno vlákno zpracovává data

- Virtualizace procesoru
  - Například jádro OS umožňující multitasking

- Vyvolání dojmu rychlé odezvy programu
  - Práce s velkým objemem dat
  - Hlavní vlákno pouze obsluhuje uživatelské rozhraní, další zpracovává data na pozadí

- Urychlení výpočtu
  - Lze-li spustit na víceprocesorovém stroji kooperující vlákna na několika procesorech
  - Pro dobře škálovatelné řešení u výpočetní úloh se už dnes ovšem nepoužívá kooperativní (fibers) ani preemptivní (thread) plánování vláken, ale task-stealing a abstrakce procesorového času do úloh (např. Intel Threading Building Blocks)

- Vhodné pro architekturu aplikace
  - Simulace – jedno vlákno počítá vlastní simulaci
  - Další vlákno periodicky vzorkuje stav simulace a zobrazuje ho
  - Primární vlákno obsluhuje uživatelské rozhraní

- Efektivita
  - Některé aplikace jsou ze své podstaty nevhodná pro jednovláknovou architekturu
  - Použití vláken může vést k výraznému zpřehlednění programového kódu
  - V moderním OS už beztak běží několik vláken, pár navíc nehraje roli
  - Každý OS má maximální strop na počet threadů, kdy je plánování procesu stále ještě efektivní

# Synchronizace

- Vlákna mohou přistupovat ke sdíleným prostředkům – např. úloha producent – konzument
- Aby byla zajištěna správná funkce programu, je třeba zajistit, aby si vlákna nepřepisovala data a četla pouze ta data, která jsou v konzistentním stavu
- Příklad
  - Máme připojenou sadu tlakoměrů a callback funkci, která vloží naměřený tlak a číslo tlakoměru do jednosměrného spojového seznamu a zaznamená čas měření jako počet tiků procesoru od jeho startu

  - Následující kód zobrazuje, kolik kódu je režie a jak málo kódu stačí na potřebnou synchronizaci
    - O tuhle výhodu ale přijdete, jakmile použijete dostatečně vysokoúrovňový jazyk, který se snaží programátorovi usnadnit práci
      - Ovšem za cenu efektivnosti jeho kódu…

```
typedef struct _tpressure{
   _tpressure *next;
  __int64 ticks;
   double pressure;
   __int16 deviceid;

} TPressure;

TPressure *head, *tail;

void __asm CallBack(double Pressure, __int16 DeviceID) {
    //ABI MS x64 konvence

     //alokace paměti
     move rax, sizeof(TPressure)
     push rax
     call malloc
     or rax, rax
     jz Error            //nepodařilo se alokovat paměť!
     mov rsi, rax        //v rsi je návratová hodnota malloc

    //uložení hodnot
    mov [rsi].TPressure.next, 0
    mov [rsi].TPressure.pressure,  ecx //pressure
    mov [rsi].TPressure.deviceid,  dx//DeviceID

    //zjištění času jako počet tiků procesoru
    xor eax, eax     //vyprázdní pipeline, aby to vrátilo
    cpuid            //po vykonání RDTSC hodnotu
    rdtsc            //výsledek je v edx:eax
    shl rdx, 32
    mov edx, eax
    mov [rsi].TPressure.ticks, rdx

    //až doteď jsme udělali vše, co šlo udělat bez zamykání
    //teď musíme updatovat *tail na rsi

  spin:                  //uděláme v podstatě spinlock
    mov rax, [tail]
    lock cmpxchgq [tail], rsi
    jnz spin

    or rax, rax     //je to první položka v seznamu?
    jnz finish      //aneb byl [tail] NULL?
    mov [head], rsi
finish:
     //teď ještě aktualizovat starou hodnotu a je hotovo
    mov [tail], rsi
    ret }
```

- Pokud by běžela alespoň dvě vlákna s výše uvedeným kódem na dvou procesorech se sdílenou pamětí, vlákna by si mohla přepisovat oblast paměti, *tail, pokud by tam nebyla obdoba spinlocku
  - IA32e mode – tj. 64-bitový, flat mód
    - Procesor bere segmentový registr jako rovný nule (jenom negeneruje výjimku jako v protected-mode)

- Řešením proto bylo uzamknutí přístupu ke sběrnici tak, aby k paměti mohlo pouze jedno vlákno – lock cmpxchgq

- A co kdybchom chtěli udržovat counter počtu prvků v seznamu?
  - Tak za každým spinlockem uděláme lock add [counter], 1

- A obousměrný spojový seznam?
  - V uvedeném příkladu jednoznačně víme i ukazatel na předka – klíčový je update tail a head

- Jenže… co dělat v případě, když nám nebudou atomické instrukce stačit? Co kdybychom chtěli kupříkladu ještě počítat i průměrný tlak?
  - To už bude nutné někde použít kritickou sekci…
  - Atomický add na plovoucí číslo totiž není

```
push qword ptr [CSHandle]
call EnterCriticalSectionshl rdx, 32

... komplexní činnost

push qword ptr [CSHandle]
call LeaveCriticalSectionshl rdx, 32
```

- A co když bude threadů více než dva a kritických sekcí bude existovat několik?
  - Zamykání v předem určeném pořadí

- Nedeterminismus
  - Thready plánuje operační systém a plánovač je pro aplikaci black-box
    - jednotlivé vlákna běží různě rychle a dopředu se neví, jak rychle
    - nelze předem určit množství přiděleného strojového času jednotlivému vláknu
    - u realtime systémů lze specifikovat nezbytné minimum
  - Celkový výpočetní čas vlákna určují i přístupy k hw – například zápis/čtení z disku – disková cache se plní v závislosti na všech běžících programech
  - Nutnost používat synchronizační primitiva, aby se zajistilo zpracování dat ve správném pořadí

# Přímá podpora vláken programovacím jazykem

- Většina programátorů není kompetentní k tomu, aby sestavila netriviální, paralelizovaný program, který nebude obsahovat chyby díky špatné synchronizaci, když na všechno nepoužije kritické sekce – např. Java synchronized

- Ale je to pohodlnější, rychleji se to učí – oběť možné optimalizaci, pokud opravdu víte, co děláte a proč
  - Pokud překladač inteligentně nepozná (jako že u některých programátorských kreací to nepozná ani autor), že daný blok kódu odpovídá např. funkci InterlockedCompareExchange, zbytečně se použije kód s mnohem vyšší režií

  - Nebo viz předchozí příklad
    - ve vysokoúrovňovém jazyku jako je např. Java by se rovnou vygenerovala kritická sekce na update spojového seznamu a počítadla prvků
    - naproti tomu v C++ by se spinlock dal realizovat tak, jak je v příkladu uvedeno

- Některým chybám lze preventivně zabránit už jenom tím, že syntaxe jazyka nedá programátorovi příležitost je udělat
  - Zkrátka omezuje programátora – tj. na jednu stranu větší bezpečnost, na druhou stranu nižší výkon