KIV Operační systémy

Vlákna

Verze 1.0

Vlákna

- Chceme-li v operačním systému souběžně spouštět několik procesů, musíme implementovat vlákna
- Vlákna zvyšují uživatelův dojem, že je systém responsivnější
  - V popředí běží aktuální vlákno, se kterým uživatel interaguje
  - V pozadí běží ostatní vlákna, která vyvíjí další činnost v době, kdy vlákno na popředí neběží
    - Nebo běží na dalších procesorech v systému

Vlákno

- Vlákno je sekvence instrukcí, která může být spravována plánovačem OS
  - Unit of CPU scheduling
- Z pohledu programátora je to často funkce, která se vykonává asynchroně k funkci main
  - Přičemž funkce main běží ve vlastním vláknu, které vytvořil OS po zavedení procesu do paměti, aby ho mohl spustit
  - Běžící proces je dynamická kolekce vláken s alespoň jedním vláknem
- Proces vlastní vlákna

Time slicing - uniprocessor

- Na jednom procesoru může v jeden okamžik běžet pouze jedno vlákno
- Chceme-li uživateli předstírat, že jich tam běží více, vlákna se musí střídat
- Time Slicing je technika, kdy se čas procesoru rozdělí na časová kvanta, která se postupně přidělují jednotlivým vláknům
  - Vždy běží vlákno, která má momentálně přidělené kvantum strojového času

Kooperativní multitasking

- Otázka zní, jak velké bude časové kvantum?
- Jednou z možností je, že vlákno poběží tak dlouho, dokud se dobrovolně nevzdá procesoru
  - Procesoru se vzdá systémovým voláním
  - Jádro OS pak vybere další vlákno, které má běžet a předá mu řízení
  - Výhodou je, že naplánování dalšího vlákna není časově kritické, protože se neděje v obsluze přerušení
  - Nevýhodou je, že vlákno může procesor uzurpovat příliš dlouho a OS se může jevit jako/být zaseknutý – Windows 3.1

Preemptivní multitasking

- Jádro OS má nainstalovanou obsluhu přerušení, které generují hodiny
- Během obsluhy přerušení hodin neběží vlákno, ale obsluha přerušení
  - => obsluha, tj. jádro, je schopné uložit stav aktuálního vlákna a nahradit ho stavem jiného vlákna
    - Tj. po návratu z obsluhy přerušení poběží jiné vlákno bez ohledu na to, jak dlouho by chtělo původní vlákno počítat
    - Celá akce však musí být rychlá, je to v obsluze přerušení

Kontext vlákna

- Při změně vlákna se vždy uloží stav aktuálního vlákna a obnoví se dříve uložený stav nově vybraného vlákna
- Kontext vlákna je dán jeho proměnnými
  - Některé proměnné jsou v registrech
  - Jiné proměnné jsou např. v zásobníku, kam ukazuje SS:RSP
  - Další proměnné jsou v paměti, kam mohou ukazovat další registry
  - Ukazatelem na instrukci, která se má vykonat – CS:RIP
  - A stavovou proměnnou OS, která říká, zda proces běží, je pozastaven, atd.

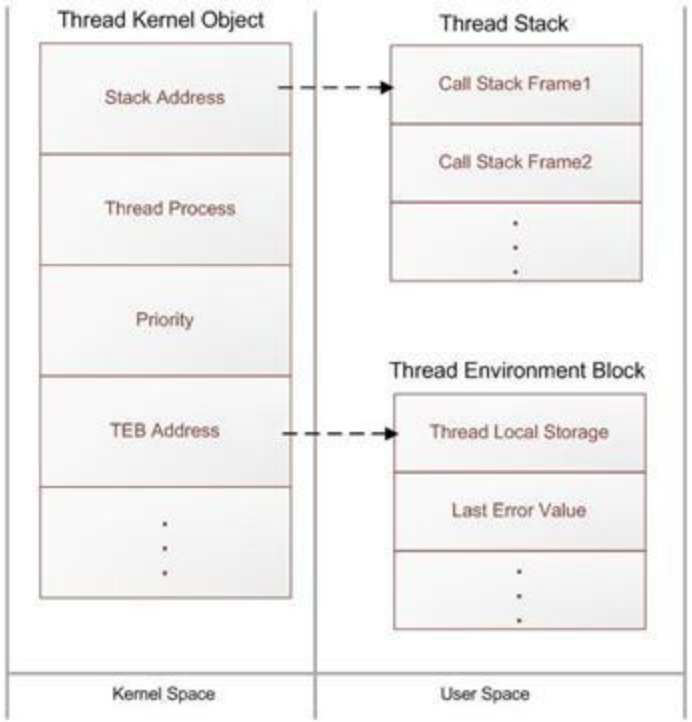
Thread Control Block

- Jádro spravuje thread podle jeho TCB
  - Pod Win/Linux přístupný přes fs/gs registry na x86, x86-64
- TCB obsahuje:
  - Uložené hodnoty registrů procesoru
  - Priorita
  - Stavovou proměnnou, čas dosavadního běhu vlákna
  - Ukazatel na Process Control Block
  - A další specifické info, jako jsou např. seznam obsluh vyjímek, skok na stránku s funkcí syscall, atd..

Windows User Thread

- Vlákno, které patří uživatelskému procesu, se sestává ze tří komponent:
  - Kernel object
    - Čas vytvořené, běhu, ukazatel na TEB, stav, priorita, počet změn kontextu, afinita, atd.
  - Zásobník
  - TCB, zde TEB aka Thread Environment Block
    - Thread Local Storage, obsluhy vyjímek, poslední chyba, impersonace, vlastněné kritické sekce, atd.

Windows User Thread

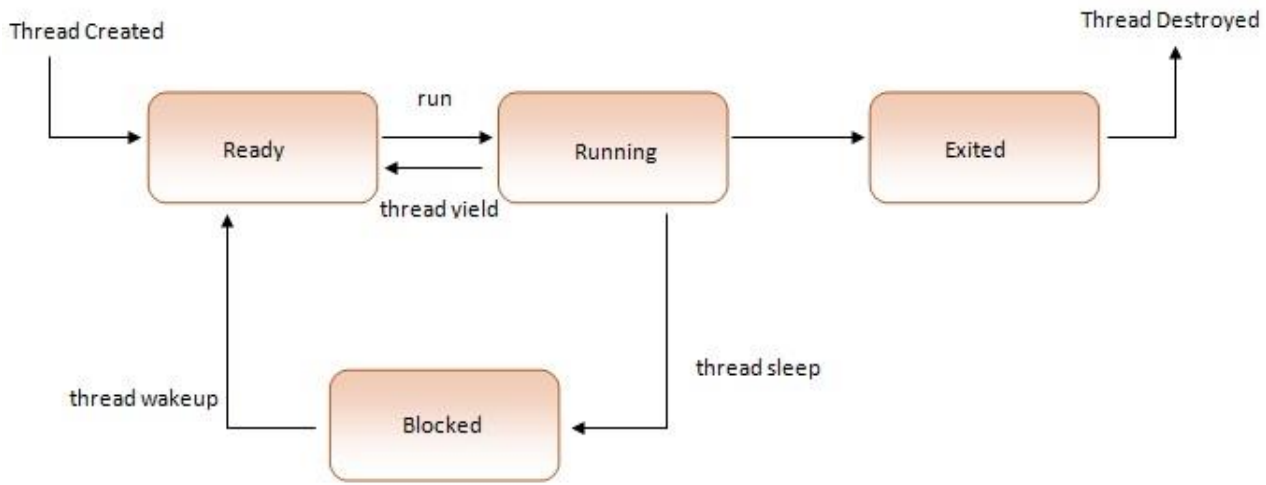


http://www.codeproject.com/Articles/662735/Internals-of-Windows-Thread

Stav vlákna

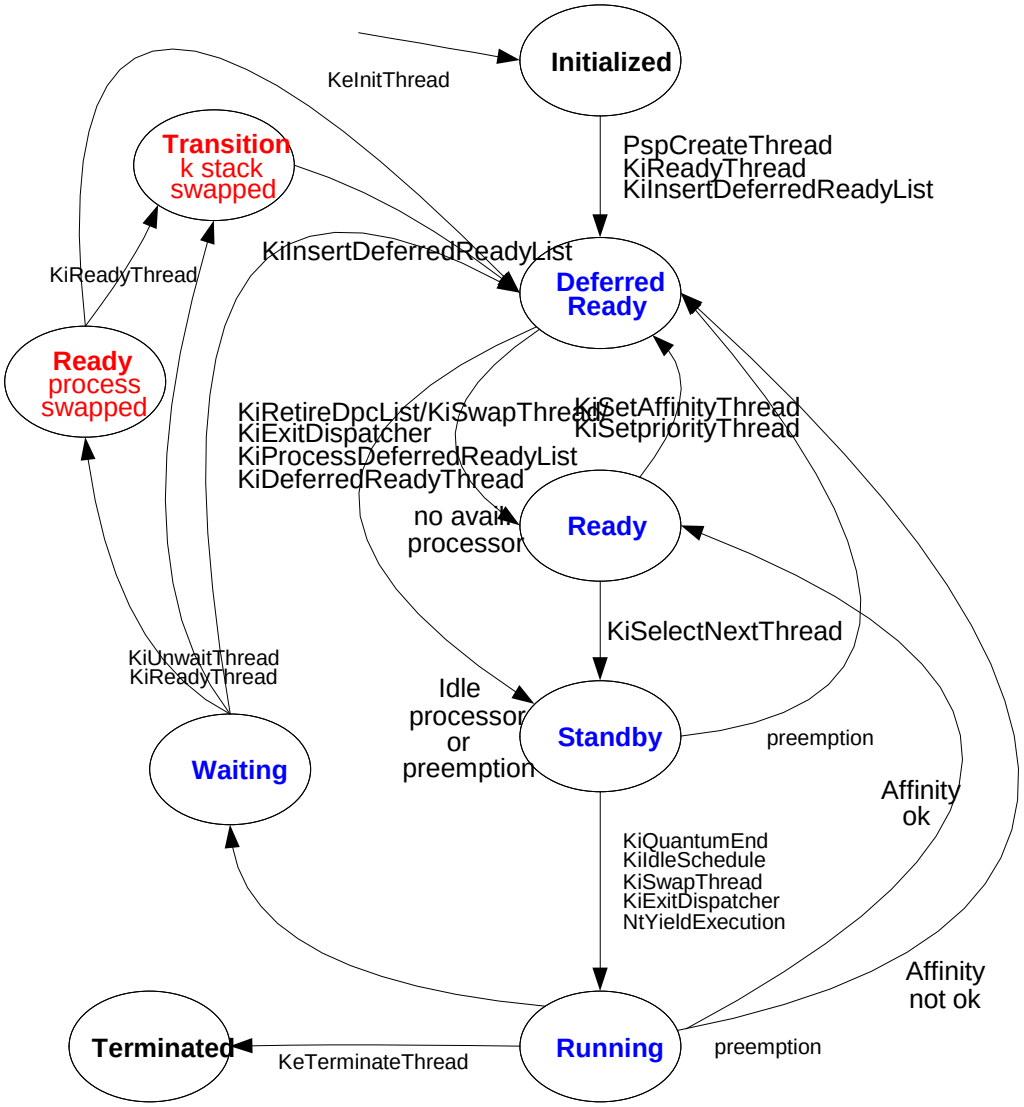
- Ve zjednodušeném pohledu je vlákno
  - Runnable – může být naplánováno a spuštěno
  - Running – vlákno běží
  - Blokované – buď se uspalo, nebo třeba čeká na podmínkové proměnné, nebo na dokončení I/O operace
  - Ukončené
- Ve skutečnosti je to složitější, protože OS potřebuje vědět, proč vlákno čeká
  - Určitě je rozdíl např. mezi PageFault a kritickou sekcí

Stav vlákna



http://www.codeproject.com/Articles/662735/Internals-of-Windows-Thread

Windows SMP Thread State



Kernel Thread Transition Diagram
DavePr@Microsoft.com
2003/04/06 v0.4b

http://i-web.i.u-tokyo.ac.jp/edu/training/ss/lecture/new-documents/Lectures/03-ThreadScheduling/ThreadScheduling.ppt

Plánování vláken

- OS neplánuje procesy, ale vlákna, protože proces je jenom kontajner
  - Pokud nejde např. o Linux, který nedělá rozdíl mezi procesem a vláknem; všechno je pro něj runnable task
- Vlákna s nejvyšší prioritou běží nějčastěji
  - Ale zároveň se občas musí ke slovu dostat i vlákna, která mají nízkou prioritu
  - Vlákno asociované s aktuálním vstupem, např. UI dialog, má dočasně zvýšenou prioritu, jako divadlo na uživatele, které mu zajistí větší dojem z responsivnosti systému

Round Robin

- Žádná priorita, jenom seznam vláken
  - Vláknům se přiděluje pouze časové kvantum 20 – 50 ms
  - Když se jedno vlákno přeruší, vezme se další runnable na seznamu
- Čím menší kvantum, tím více ztraceného času při přepínání vláken
- Čím větší kvantum, tím se zase OS zdá uživateli pomalejší
- Hodně procesů, které by měly mít nízkou prioritu může vyhladovět procesy, které by ji měly mít vysokou

Round Robin s prioritou

- Několik seznamů vláken
- Co seznam, to jedna priorita
- Nejprve běží vlákna ze seznamů s vyšší prioritou, dokud takový seznam není prázdný
  - Vlákno skončilo, nebo je blokováno
- Takže ve výsledku je možné vyhladovět procesy s nízkou prioritou

Plánovač Linuxu

- Používá velká časová kvanta pro důležité procesy
- Modifikuje velikost přidělovaných kvant podle využití CPU
- Snaží se držet procesy na stejném CPU
  - Každé CPU má svou fronton procesů, ze které plánuje procesy ke spuštění
  - Pokud má některý CPU frontu příliš dlouhou, přebývající procesy jsou přesunuty na jiný CPU
    - Každý proces má (affinity) masku, která říká, na kterých procesorech může běžet a na tom se nic nemění
- Completely Fair Scheduler, O(1)

Základ plánování

1. Vybere se fronta s největší prioriou a spustitelným procesem
   - V systému jsou dvě sady (active & expired) 140 front, každá pro jednu statickou prioritu
     - 0 – 99 jsou real-time procesy
     - 100 – 139 jsou normální procesy, nastavuje se pomocí nice()
     - 5 integerů tvoří bitmapu jejíž bity říkají, která fronta má spusitelný proces
2. Vybere se v ní první spustitelný proces a vypočítá se jeho časové kvantum
3. Spustí se a až vyčerpá své kvantum, dá se expire fronty
4. Zpět do prvního bodu

Výpočet kvanta

- SP – statická priorita
  - SP<120: Quantum = (140 – SP)*20
  - SP>=120: Quantum = (140-SP)*5
  - Proces s větší prioritou (tj. menším číslem) dostává větší časová kvanta

Priorita	SP	Nice	Quantum
Nejvyšší	100	-20	800 ms
Vysoká	110	-10	600 ms
Normální	120	0	100 ms
Nízká	130	+10	50 ms
Nejnižší	139	+20	5 ms

Dynamická priorita (DP)

- Vypočítává se ze statické priority a doby, po kterou proces neběžel
- Proces dostane bonus <0, 10>
  - 0 – sníží prioritu o (bonus) 5
  - 5 - neutrální (bonus 0)
  - 10 – zvýší prioritu o (bonus) 5
  - Pokud je ovšem process interaktivní, pak platí: bonus – 5 >= SP/4 - 28
- DP = max (100, min(SP - bonus + 5, 139))