Proces a thread - stavy, implementace, plánování, synchronizace.

Thursday, May 30, 2013 8:27 AM

Wiki: Thread je nejlehčí jednotka plánování, TXKoutný: Proces - *největší* výpočetní entita plánovače.

Je potřeba rozlišovat thread programátorský a thread z hlediska plánovače – dvě úplně odlišné věci, ale v literatuře se oboje jmenuje thread a pak jsou z toho zmatky.

- Viz přednáška 6 OS

Typy vláken dle OS
- jádrová
- lehké procesy
- uživatelská

Process

- Má svůj vlastní kontext a adresní prostor
- Může mít jedno nebo více vláken v jednom (svém) adresním prostoru
- Vlákna uvnitř procesu sdílejí prostředky procesu (otevřené soubory)

Thread

- Kernel thread: jednotka plánování plánovače jádra (činnost plánovače je založená na přepínání kernel threadů), jejich počet je nezávislý na počtu jader procesoru nebo počtu procesorů
  - pouze jádrová věc
- User thread: uživatelem vytvořený thread v rámci prostředků programovacího jazyka
  - Bez podpory plánovače jádra: fiber (100% user space)
  - S podporou plánovače jádra: lightweight proces (lehký protože nemá vlastní adr. prostor, ukazatele na soubory apod.) – fiber namapovaný na kernel thread
- Sdílená paměť a Thread-local storage (může mít a nemusí, v rámci adresního prostoru procesu)

Fiber

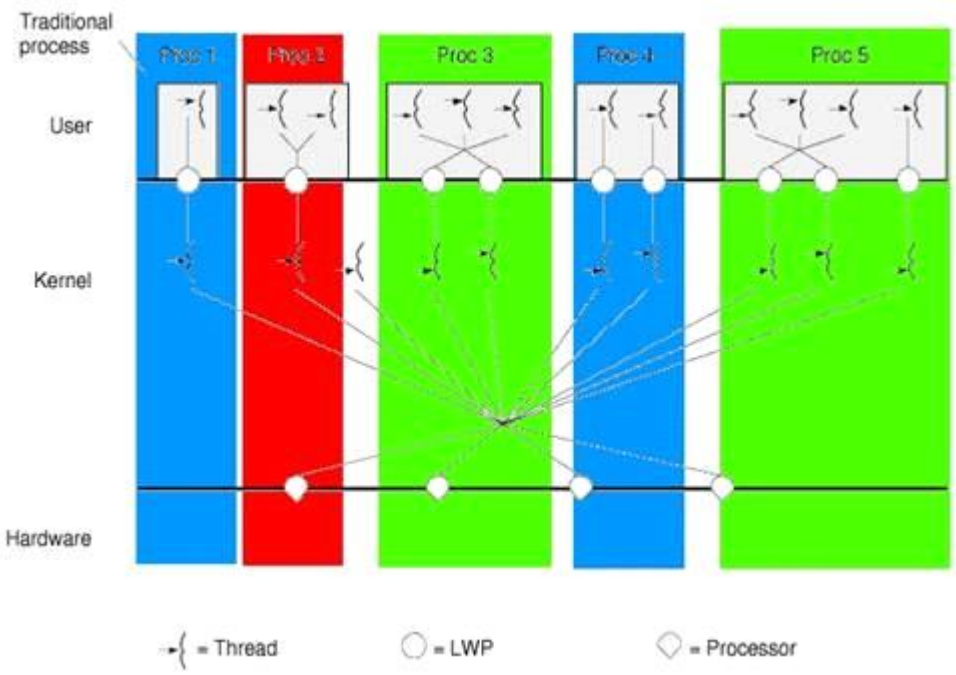
- Běží v uživatelském prostoru a při přepnutí fiberů se nepřepíná kontext – rychlé a levné
- Spolupracují kooperativně (ne preemptivně) – musí udělat yield, jsou **implicitně synchronizované**
- Méně problematická thread-safety: nejsou potřeba spinlocky a atomické operace
- Vyžaduje menší podporu od OS, nemusí požadovat vůbec žádnou (třeba podle výhodnosti plánování –OS může a nemusí mít „lepší“ plánovač). Podporují je Unixové systémy, Microsoft, Symbian…
- Nemohou využívat víc procesorů (všechny fibery jsou v jediném kernel threadu)
- Sdílená paměť a Fiber-local storage

LWP (Light-Weight Process)

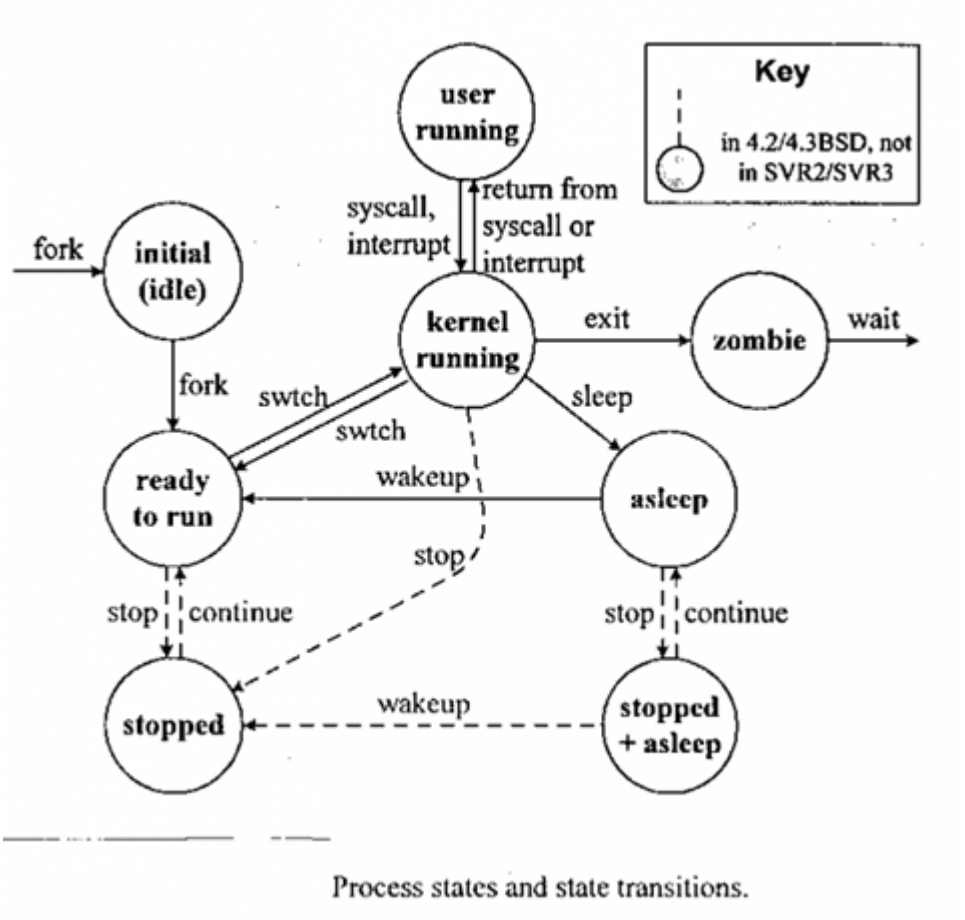
- Linux: http://www.linuxforu.com/2011/08/light-weight-processes-dissecting-linux-threads/
  - Kernel thread = LWP
  - Obsluha user-thread (u Linux = process) na LWP 1:1
- Na Windows: http://www.i.u-tokyo.ac.jp/edu/training/ss/lecture/new-documents/Lectures/03-ThreadScheduling/ThreadScheduling.pdf
  - Windows plánuje vlákna, ne procesy
  - Plánování je preemptivní, založeno na prioritě a používá round-robin pro nejvyšší priority
  - Každý thread má aktuální a základní prioritu - základní priorita je inicializována při spuštění procesu, aktuální pak závisí na chování threadu - priorita se snižuje, pokud thread vždy vyčerpá přidělené kvantum

Vláknové modely

- **1:1 (kernel vlákna)** Vlákna vytvořená uživatelem odpovídají počtem 1:1 entitám, které plánuje jádro. Nejjednodušší přístup, ale je třeba uvážit preempci a thread-safety
- **N:1 (uživatelská vlákna)** všechna aplikační vlákna jsou namapována na jedno jádrové vlákno, jádro nemá tušení o tom, že aplikace běží vláknově.
- **M:N (hybridní)** komplexní na implementaci (vyžaduje změny v jádře i uživ. prostoru) ale umožňuje zvýšení efektivity výpočtu (například přiřazení více uživatelských vláken více vláknům jádra – proces může běžet na více procesorech).



Stavy procesu



Process states and state transitions.

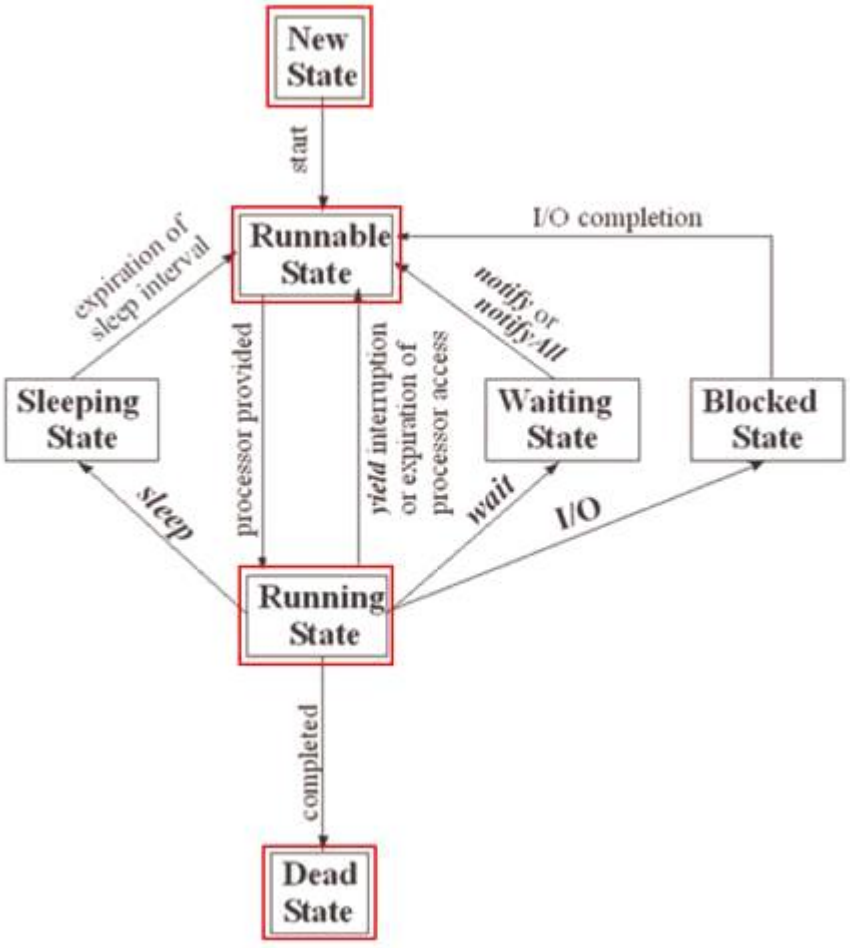
- **Počáteční** (initial/idle) – fork začal vytváření procesu.
- **Připraven na vykonání** (ready to run) – fork dokončil vytváření procesu, proces čeká až bude

naplánován na přidělení procesoru.

- **Běžící v jádře** (kernel running) – byl naplánován, vykoná se přepnutí kontextu, procedura jádra swtch() uloží HW kontext do registrů.
- **Běžící uživatelsky** (user running) – proces vykonává svůj programový kód. Může přejít do stavu běžící v jádře v důsledku volání služby jádra nebo přerušení, po skončení obsluhy se vrátí.
- **Spící** (asleep) – při vykonávání systémového volání se může stát, že je nutno čekat na nějakou událost nebo prostředek, proces (v jádře) proto zavolá proceduru sleep(). Když událost nastane, jádro vzbudí proces, proces se stane připraven na vykonání a po naplánování pokračuje obsluha systémového volání ve stavu běžící v jádře.

  Připraven na vykonání se může stát také, je-li běžící a uplyne mu přidělené časové kvantum - vykoná se preempce běžícího procesu a to ve stavu běžící uživatelsky nebo při návratu do něj. Jádro je nepreemptivní. Přerušení se může vyskytnout i ve stavu běžící v jádře, kdy po skončení obsluhy přerušení proces pokračuje ve stavu běžící v jádře.

- **Mátoha** (zombie) - proces končí voláním exit() anebo v důsledku signálu, dokud rodič nevykoná wait().
- **Zastaven** (stopped) - do něj přejde proces, je-li běžící nebo spící (+ asleep), po stop signálech:
  - SIGSTOP zastav proces
  - SIGTSTP CTRL-Z
  - SIGTTIN tty čtení procesu v pozadí
  - SIGTTOU tty psaní procesu v pozadí
  - SIGCONT převede proces do stavu připraven na vykonání nebo do stavu spící.



Plánování, plánovací třídy, inverze priority

Víceúlohové systémy:

- systémy reálného času
- interaktivní systémy

- dávkové systémy

  Ve víceúlohových systémech sdílení času je tradiční problém „vhodně“ a spravedlivě přidělovat čas jednotlivým procesům (adekvátně k typu – rt/dávkový) a zajišťovat vysokou průchodnost (tyhle tři požadavky jsou často v kontradikci). Je třeba hlídat aby nedošlo k vyhladovění a k inverzi priorit.

  Problém velikosti časového kvanta: v interaktivních systémech je třeba přepínat často, aby byl vytvořen dojem okamžité odezvy, ale přepínání HW kontextu je náročné a zdržuje = čím kratší čas, tím větší režie systému.

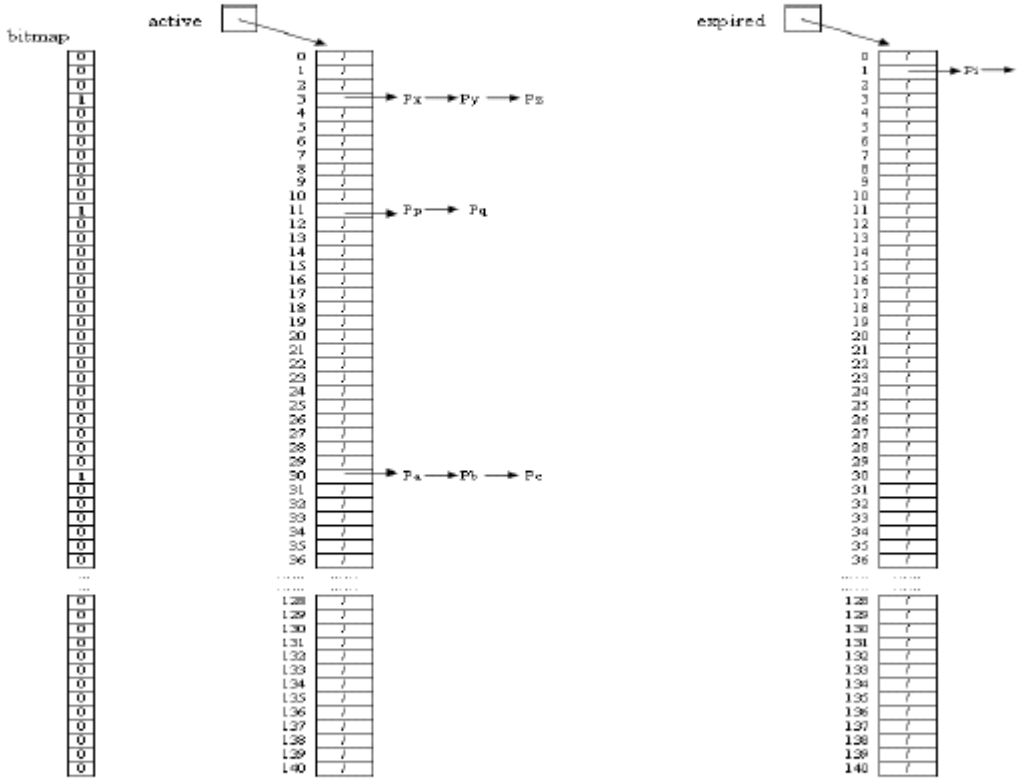
  Typy plánování:

- preemptivní – s předbíháním, proces je možno přerušit zvenčí, je potřeba zavést synchronizační primitiva
- nepreemptivní – proces se musí vzdát procesoru sám (fiber)

  Plánování:

- **Linux 2.6 – plánovač *O(1)* / Ingo Molnar**

  - fronta připravených procesů pro každý procesor
  - fronta je složena ze dvo polí o velikosti počtu priorit
    active – procesy s ještě nevyčerpaným časovým kvanten
    prioritní plánování, RR v rámci priority
    expired- procesy s vyčerpaným časovým kvantem

  nové časové kvantum a priorita se vypočte ihned po vyčerpání předcházejícího kvanta *O(1)* a proces se zařadí do fronty procesů s novou prioritou do pole **expired** na index odpovídající priorité
  je-li pole **active** prázdné „konec epochy“ pole se přepnou

- **FIFO** – nejjednodušší, neadaptabilní, málokdy spravedlivý. Když nějaký proces nedoběhne, může

dojít k vyhladovění

- **Shortest proces (job) first -** *nepreemptivní*, u dávkových úloh, předpokládá se znalost dob trvání procesů, vezme se proces/job s nejkratší předpokládanou dobou běhu, po jeho skončení se z fronty bere další proces; optimalizuje dobu obrátky - dávkové
- **shortest remaining time –** *preemptivní,* vybere úlohu, jejíž zbývající doba běhu je nejkratší - když je plánovaná úloha s 10 min do jejího dokončení a přijde úloha trvající 1 minutu, systém provede přeplánování a začne běžet ta nová úloha; dobrý pro krátký (hl. I/O vázaný) úlohy, může ale dojít k vyhladovění (v systému je hodně „krátkých" úloh, na ty dlouhodobější se nedostane řada)
- **Round robin** – procesy obdrží každý stejný čas a střídají se dokola, nedojde k vyhladovění protože není zavedena priorita
- **Prioritní plánování** – Každý proces má svou prioritu (procesy jádra nejvyšší, interaktivní vysokou, dávkové nízkou), procesy jsou podle priority roztříděny do tříd, ve kterých se periodicky střídají metodou Round Robin (vždy nejdřív první neprázdná třída od nejvyšší priority). Může dojít k vyhladovění procesů s nízkou prioritou (řešením je zvyšovat prioritu dlouho nenaplánovaným procesům).
    - priorita **statická** (při startu procesu) a **dynamická** (chování procesu v poslední době, snižuje ji u běžícího procesu při každém tiku plánovače; = 1/f, *f* je velikost částí kvanta, kterou proces naposledy použil)
- Prakticky ve všech dnešních OS, mnoho různých metod:
- **Fair Share** - Férové rozdělení času mezi uživatele, nikoli procesy. Rekurzivní aplikace Round Robin na každé úrovni abstrakce - nejprve na skupiny, pak na uživatele, pak na procesy.
- **Loterie** - plánovač má k dispozici pevný počet tiketů, každý proces obdrží určitý počet tiketů a plánovač pak vybere jeden tiket - **náhodně** vybere tiket a přidělí časové kvantum procesu, který ten tiket vlastní
- **Epochy** - čas procesoru je rozdělen do epoch
    a. každý proces má specifikováno časové kvantum v rámci epochy
    b. v jedné epoše proces může využívat své časové kvantum po částech
    c. epocha končí, když všechny běhu schopné procesy vyčerpaly svá časová kvanta
    d. Délka časového kvanta v epoše závisí na prioritě procesu.

Jaký je rozdíl mezi NoRealTime, SoftRealTime a HardRealTime

NoRealTime – Nemají žádný deadline.

SoftRealTime – překročení termínu se toleruje, systém reaguje zhoršenou kvalitou poskytovaných služeb – např. vypadne pár snímků, nebo přílet letadla se dozvíte s několikasekundovým zpožděním.

HardRealTime – Dokončení výpočtu po termínu se považuje za chybu a výsledek za bezcenný – strict deadline. Nedodržení termínu může vést k celkovému selhání systému (airbag, jaderná elektrárna…).

Z <https://d.docs.live.net/67cbdb2faf0647ea/Dokumenty/FAV/A11N0110P/Státní%20závěrečná%20zkouška/PPR/PPR_Karfik_v2.docx>

- Soft RT
- Hard RT

Viz níž.

Inverze priority

Inverzí priorit rozumíme situaci, která nastane, pokud vlákno s vyšší prioritou požaduje přístup k systémovým zdrojům, které v danou chvíli právě exkluzivně drží vlákno s nižší prioritou. V tomto případě dojde k preempci do vlákna s vyšší prioritou, které však nemůže běžet díky zablokovanému systémovému zdroji. To je velice nepříjemná situace, zejména v RTOS. Jediným řešením je umožnit vláknu, které systémový zdroj drží co nejrychleji doběhnout a umožnit tak i jiným vláknům pokračovat v jejich činnosti. **K vyřešení této situace se používá systém inverze priorit**, který umožní vláknu s nižší prioritou zdědit prioritu kritického vlákna, rychle vykonat potřebné operace až do chvíle uvolnění požadovaného systémového zdroje a dále pak nechat pokračovat v práci kritické vlákno.

Synchronizace

Kernel space (viz kapitola 9), vs. user space. a další (?)

Process Control Block (PCB)

Udržuje informace spojené s každým procesem, je využit při znovu rozběhnutí přerušeného procesu.

- Stav procesu
- Program counter
- CPU registry
- Informace o CPU plánování
- Info o správě paměti
- Info o status I/O (File descriptory, sockety atd.)

From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx>