

Operační systémy

Vlákna a procesy – Implementační aspekty

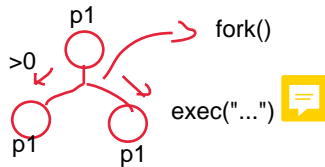
Petr Krajča



Katedra informatiky
Univerzita Palackého v Olomouci

- proces – původně základní entita vykonávající činnost
- procesy tvoří hierarchii, každý proces identifikován pomocí PID
- systém při inicializaci spustí první proces (`init`)
- nový proces (potomek) vytvořen voláním `fork()` – vytvoří kopii aktuálního procesu

```
pid_t n_pid = fork();  
if (n_pid < 0) { /* chyba */ }  
else if (n_pid == 0) { /* kod potomka */ }  
else { /* kod rodice */ }
```



- používá se společně s voláním `exec` – nahraje do paměti kód ze souboru a začne jej provádět
- v rámci vztahu rodič-potomek jsou sdílené některé zdroje (např. popisovače souborů)
- sirotci – pokud rodičovský proces skončí dříve, přejde pod `init`
- zombie – proces již skončil, ale existuje v systému
- priorita – nice (40 hodnot)

- vlákna přidána do Unixů později (dříve i ve formě knihoven)
- \Rightarrow není zcela konzistentní s původní koncepcí
- Jak se má zachovat `fork()`?
- oddělené mechanismy pro synchronizaci vláken a procesů
- vlákno – běžná procedura s jedním argumentem vracející jednu hodnotu

```
void *foo(void *arg) {  
    /* kod vlakna */  
    return (void *)42;  
}
```

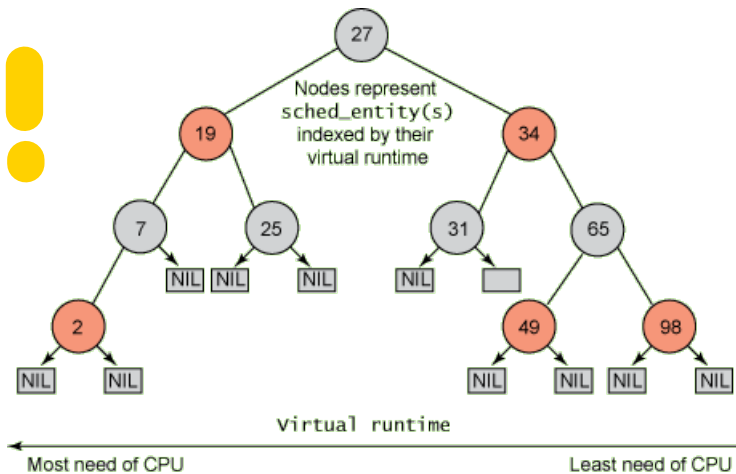
```
pthread_t thr;  
void * result;
```

```
pthread_create(&thr, NULL, foo, (void *)123);  
/* kod provadeny hlavnim vlaknem */  
pthread_join(thr, &result);
```

- interně jádro pracuje s vlákny i procesy stejně
- proces/vlákno \implies task (účastní se plánování)
- systémové volání `clone` – zobecněný `fork` (umožňuje definovat, které struktury se mají sdílet)
 - paměťový prostor
 - otevřené soubory
 - I/O
 - id rodiče
 - ...
- stavy úloh: běžící, připravené k běhu, uspané-přerušitelné – čeká na nějakou podmínku, uspané-nepřerušitelné (čeká na nějakou kritickou HW operaci), zastavené, skončené (zombie)
- je možné vybrat typ plánovače – obecný, dávkové úlohy, FIFO nebo RR (pro práci v reálném čase)

Completely Fair Scheduler

- varianta Guaranteed scheduleru
- tasky organizovány v RB-stromu (podle toho, kolik dostaly času)
- je zvolen ten, který získal nejméně času (tj. nejlevější list)
- priority řešeny pomocí koeficientů
- díky automatickému vyvažování má výběr tasku složitost $O(\log n)$.





- vlákno – základní jednotka vykonávající činnost (účastní se plánování)
- process – obsahuje jedno a více vláken (společné zdroje a paměť)
- job – slučuje několik procesů dohromady (společné správa, nastavení kvót, atd.)
- fiber – „odlehčené vlákna“ implementované v kontextu vlákna (kooperativní multitasking)

Vznik procesu

- není vyžadován vztah rodič-potomek
- `CreateProcess` – funkce vytvářející nový proces (10+ argumentů); příprava ve spolupráci s daným subsystémem, více verzí jednoho programu v jednom souboru
- `CreateThread` – funkce vytvářející nové vlákno v principu podobná `pthread_create`

```

BOOL WINAPI CreateProcess(
    __in_opt    LPCTSTR lpApplicationName,
    __inout_opt LPTSTR lpCommandLine,
    __in_opt    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    __in_opt    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in        BOOL bInheritHandles,
    __in        DWORD dwCreationFlags,
    __in_opt    LPVOID lpEnvironment,
    __in_opt    LPCTSTR lpCurrentDirectory,
    __in        LPSTARTUPINFO lpStartupInfo,
    __out       LPPROCESS_INFORMATION lpProcessInformation
);

HANDLE WINAPI CreateThread(
    __in_opt    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in        SIZE_T dwStackSize,
    __in        LPTHREAD_START_ROUTINE lpStartAddress,
    __in_opt    LPVOID lpParameter,
    __in        DWORD dwCreationFlags,
    __out_opt   LPDWORD lpThreadId
);

```




- plánování se účastní vlákna

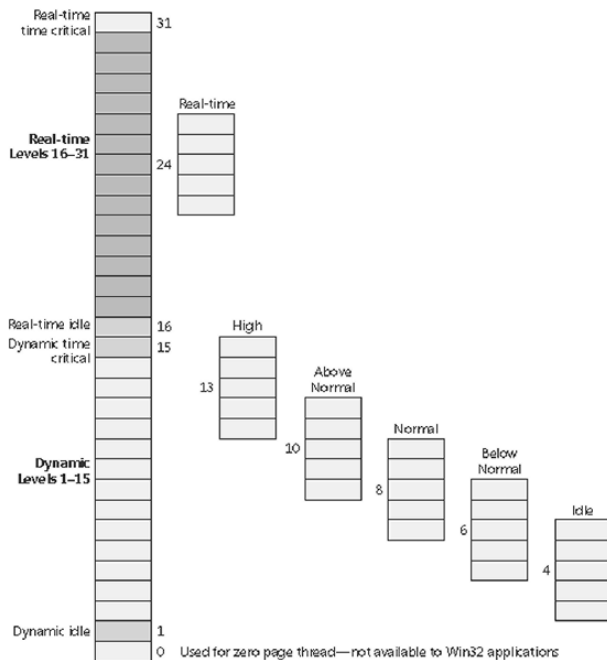
Stavy vláken

- **initiliazed** – během inicializace vlákna
- **ready** – čekající na běh (z těchto vláken vybírá scheduler další pro běh)
- **standby** – vlákno připraveno k běhu na konkrétním CPU
 - přechod do running
 - přechod do ready (pokud vlákno s vyšší prioritou přeslo do režimu standby)
- **running** – vlákno běží; možné přechody
 - vlákno s vyšší prioritou získalo CPU (návrat do standby nebo ready)
 - po vypršení kvanta \implies ready
 - čekání na událost \implies waiting
 - ukončení vlákna
- **waiting** – čeká na nějakou událost; přechod do ready, standby či running (v případě úloh s vysokou prioritou)
- **transition** – zásobník je mimo fyzickou paměť (přechod do ready)
- **terminated** – vlákno je ukončeno (lze změnit na initialized)

- pokud se objeví vlákno s vyšší prioritou ve stavu ready než má vlákno ve stavu running, dostane CPU; aktuálně běžící vlákno je přesunuto na začátek příslušné fronty

Priority

- priorita – hodnota 0–31 přiřazena vláknu (32 úroňová fronta)
- třída priority – vlastnost procesu udávající základní prioritu vláken
 - Real-time (24) 
 - High (13)
 - Above normal (10)
 - Normal (8)
 - Below normal (6)
- priority vláken – Time critical, highest, above normal, normal, below normal, lowest, idle
- priorita vlákna je dána relativně k prioritě procesu + další úpravy
- kategorie priorit
 - idle (0) – zero page thread 
 - dynamické úrovně (1–15) – běžné procesy
 - real-time úrovně (16–31) – systémové procesy (nejedná se o realtime systém)



Velikost kvanta

- závisí na verzi OS
 - workstation – 6 jednotek (2 tiky přerušení časovače)
 - server – 36 jednotek (12 tiků)
- velikost jde měnit (v nastavení nebo dočasně)
- při čekání, přepnutí, atd. se velikost kvanta mírně snižuje
- proces na popředí – všechna jeho vlákna mají 3× větší kvantum

Dočasné zvýšení priority (Priority Boost)

- u procesů s dynamickou úrovní
- po dokončení I/O zvýšena priorita o
 - +1 – disk, CD-ROM, grafická karta
 - +2 – síťová karta, ser. port
 - +6 – klávesnice, myš
 - +8 – zvuková karta
- po uplynutí kvanta se priorita snižuje o jedna až na základní hodnotu

- po čekání na událost nebo synchronizaci s jiným vláknem
 - na dobu jednoho kvanta zvýšena priorita o 1
 - při synchronizaci – vlákno může získat prioritu o jedna vyšší než mělo vlákno, na které se čekalo
- vlákno na popředí po dokončení čekací operace \implies priorita +2
- aktivita v GUI \implies priorita +4
- vlákno už dlouho neběželo (řádově sekundy) \implies priorita 15 + 2x delší kvantum

SMP

- proces i vlákno má nastavenou masku affinity – seznam povolených CPU, kde může běžet
- každé vlákno má ještě dvě hodnoty – ideální procesor a minulý procesor
- procesor pro vlákno je vybírán následovně
 - 1 nečinný CPU
 - 2 ideální CPU
 - 3 minulý CPU
 - 4 aktuální CPU
- každý procesor má svůj vlastní plánovač \implies lepší škálování

- koncepčně vychází z Unixu \implies pojetí procesů, vláken
- k plánování se používá víceúrovňová fronta

Grand Central Dispatch

- rozšíření OS a prog. jazyků
- umožňuje snadno provádět bloky kódu ve vláknech
- existuje několik front, kam se jednotlivé úlohy řadí (každá má thread-pool)
- globální fronty se umožňují přizpůsobit konkrétnímu HW
- soukromé fronty (úlohy zpracovávány sekvenčně), ale v samostatném vlákně
- rozšíření C \implies blok kódu:

```
x = ^{ printf("Foo!\n"); }  
y = ^(int a) { return a * 10; }  
x(); y(20);
```

- příklad použití:

```
dispatch_async(dispatch_get_main_queue(), ^{.... });
```