



Operační systémy 1

Procesy

Petr Krajča



Katedra informatiky
Univerzita Palackého v Olomouci



- neformálně: proces = běžící program (vykonává činnost)
- proces charakterizuje:
 - kód programu
 - paměťový prostor
 - data – statická a dynamická (halda)
 - zásobník
 - registry
- operační systém: organizace sekvenčních procesů
- oddělení jednotlivých úloh (abstrakce)
- multiprogramování: (zdánlivě) souběžný běh více procesů
- efektivní využití prostředků CPU (čekání na I/O)
- komunikace mezi procesy, sdílení zdrojů \implies synchronizace

Obecný životní cyklus procesu

- nový (new) – proces byl vytvořen
- připravený (ready) – proces čeká, až mu bude přidělen CPU
- běžící (running) – CPU byl přidělen procesor a právě provádí činnost
- čekající (waiting/blocked) – proces čeká na vnější událost (např. na vyřízení I/O požadavku, synchronizační primitiva)
- ukončený (terminated) – proces skončil svou činnost (dobrovolně × nedobrovolně)

Rozšíření

- suspend – proces byl odsunut do sekundární paměti (obr. Sta 123)
- ready/suspend + block/suspend – vylepšení předchozího mechanismu
- fronty pro přechod mezi stavy (obr. Sta 121)



- tabulka procesů \implies PCB: process control block – informace o procesu

Informace identifikující proces

- identifikátor procesu, uživatele, rodičovského procesu

Stavové informace

- stav uživatelských registrů
- stav řídicích registrů (IP, PSW)
- vrchol zásobníku(ů)

Řídící informace

- informace sloužící k plánování (stav procesu, priorita, odkazy na čekající události)
- informace o přidělené paměti
- informace o používaných I/O zařízeních, otevřených souborech, atd.
- oprávnění, atd.



- 1 uložení stavu CPU (kontextu, tj. registrů, IP, SP) do PCB aktuálního procesu
 - 2 aktualizace PCB (změna stavu, atd.)
 - 3 zařazení procesu do příslušné fronty
 - 4 volba nového procesu
 - 5 aktualizace datových struktur pro nový proces (nastavení paměti, atd.)
 - 6 načtení kontextu z PCB nového procesu
- \implies jde řešit softwarově nebo s podporou HW (různá náročnost na čas CPU)
 - kooperativní \times preemptivní přepínání

Důvody k přepínání

- vypršení přiděleného časového kvanta (nutná podpora HW)
- přerušení I/O (aktuální proces může pokračovat \times čekající proces může začít běžet)
- výpadek paměťové stránky, vyvolání výjimky (např. dělení nulou)



- potřeba efektivně plánovat procesorový čas
- časové kvantum: maximální čas přidělený procesu
- samotné přepnutí procesu má režii (uložení kontextu, vyprázdnění cache) \implies latence
- Jak zvolit velikost? \implies interaktivita \times odvedená práce
- CPU-I/O Burst cycle: pravidelné střídání požadavků na CPU a I/O
- \implies procesy náročné na CPU \times I/O

Typy plánování

- dlouhodobé – rozhoduje, zda bude přijat k běhu (změna stavu z new na ready)
- střednědobé – načtení/odložení procesu do sekundární paměti
- krátkodobé – vybírá mezi dostupnými procesy ty, které budou spuštěny na CPU (přechod ze stavu ready do running)
- I/O – rozhoduje jednotlivé požadavky na I/O



Různé typy úloh/systémů

- interaktivní
- dávkové zpracování
- pracující v reálném čase

Obecné požadavky na plánování procesů

- spravedlnost – každému procesu by v rozumné době měl být přidělen CPU
- vyváženost – celý systém běží
- efektivita – maximální využití CPU
- maximalizace odvedené práce (throughput)
- minimalizace doby odezvy
- minimalizace doby průchodu systémem (turnaround)



- vhodné pro dávkové zpracování

First-Come-First-Served

- první proces získá procesor
- nové procesy čekají ve frontě
- proces po skončení čekání zařazen na konec fronty
- nepreemptivní
- jednoduchý, neefektivní

Shortest Job First

- vybere se takový proces, který poběží nejkratší dobu
- nepreemptivní
- zlepšuje celkovou průchodnost systémem
- je potřeba znát (odhadnout) čas, který proces potřebuje

Shortest Remaining Time Next

- pokud nový proces potřebuje k dokončení činnosti méně času než aktuální, je spuštěn
- preemptivní



- vhodné pro interaktivní systémy

Round robin

- každý proces má pevně stanovené kvantum
- velikost kvanta? (\implies mírně větší než je typicky potřeba)
- připravené procesy jsou řazeny ve frontě a postupně dostávají CPU
- vhodný pro obecné použití (relativně spravedlivý)
- protežuje na CPU náročné procesy (\implies přidána další fronta pro procesy po zpracování I/O, Sta 406)

Prioritní fronta (obr. Sta 399)

- každý proces má definovanou prioritu
- statické \times dynamické nastavení priority (např. vyšší priorita po I/O)
- systém eviduje pro každou prioritu frontu (čekající procesy)
- riziko vyhladovění procesů s nízkou prioritou
- rozšíření: nastavení různých velikostí kvant pro jednotlivé priority (přesun mezi prioritami, nižší priorita \implies delší kvantum)



Shortest Process Next

- vhodný pro interaktivní systémy (krátká doba činnosti + čekání)
- používá se odhad, podle předchozí aktivity procesu

Guaranteed Scheduling

- reálně přiděluje stejný čas CPU
- máme-li n procesů, každý proces má získat $\frac{1}{n}$ CPU
- určí se poměr času, kolik získal a kolik má získat (< 1 – proces měl méně času)
- volí se proces s nejmenším poměrem

Lottery Scheduling

- proces dostane přiděl „losů“
- procesy voleny náhodně (proporcionální přidělování)
- možnost vzájemné výměny losů mezi procesy

Fair-share Scheduling

- plánování podle skupin procesů (např. podle uživatelů)

- nutné, aby systém zareagoval na požadavek v požadovaném intervalu (důležité pro řídicí systémy, např. v průmyslu)
- dva typy úloh:
 - hard real-time – požadavek je potřeba vyřešit do určité přesně dané doby (intervalu)
 - soft real-time – zpoždění vyřešení úlohy je tolerovatelné
- periodické × neperiodické úkoly
- systém nemusí být schopen všem požadavkům vyhovět

Varianty plánování

- statickou tabulkou – obsluha periodických úkolů je dána předem
- statické definice priorit – jednotlivým úlohám jsou nastaveny priority, aby byla splněna zadaná kritéria
- dynamické plánování – proces je spuštěn, pokud je možné splnit jeho požadavky
- dynamicky nejlepší možné – žádná omezení, pokud nebylo možné splnit všechny požadavky v systému, proces je odstraněn



- proces = sekvence vykonávaných instrukcí v jednom paměťovém prostoru
- procesy jsou od sebe izolovány \implies nemusí být vždy žádoucí
- obecnější přístup \implies proces = správa zdrojů (data, kód), vlákno = vykonávaný kód
- možnost více vláken v rámci jednoho procesu
- každé vlákno má své registry, zásobník, IP, stav (stejně jako proces); jinak jsou zdroje sdílené
- vlákna sdílí stejné globální proměnné (data) \implies žádná ochrana (předpokládá se, že není třeba \implies potřeba synchronizace)
- využití vláken
 - rozdělení běhu na popředí a na pozadí (CPU \times I/O)
 - asynchronní zpracování dat
 - víceprocesorové stroje
 - modulární architektura



Vztah proces-vlákno

- 1:1 – systémy, kde proces = vlákno
- 1:N – systémy, kde proces může mít více vláken (nejčastější řešení)
- N:1/M:N – více procesů pracuje s jedním vláknem (clustery, spíše hypotické řešení)

Implementace vláken

- jako knihovna v uživatelském prostoru
- součást jádra operačního systému
- kombinované řešení
- green threads



V uživatelském prostoru

- proces sám se stará o správu a přepínání vláken
- vlastní tabulka vláken
- nejde použít preempce \implies kooperativní přepínání (rychlé – není potřeba systémové volání)
- možnost použít plánovací algoritmus dle potřeby
- problém s plánováním v rámci operačního systému (OS neví nic o vláknech)
- problém s blokujícími systémovými voláními (jsou zablokována všechna vlákna)



V jádře

- jádro spravuje pro každé vlákno struktury podobně jako pro procesy (registry, stav, ...)
- řeší problém s blokujícími voláními
- vytvoření vlákna pomalejší (recyklace vláken procesu \implies pool vláken)
- přepínání mezi vlákny jednoho procesu rychlejší (než mezi procesy; ale pomalejší než u vláken v uživatelském prostoru)
- preemptivita

Hybridní

- proces má M vláken v jádře, které má každé N_i vláken uživatelském prostoru
- v OS ústup, renesance v Go



- 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 dřív, 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)
- \implies není zcela konzistentí 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);
```

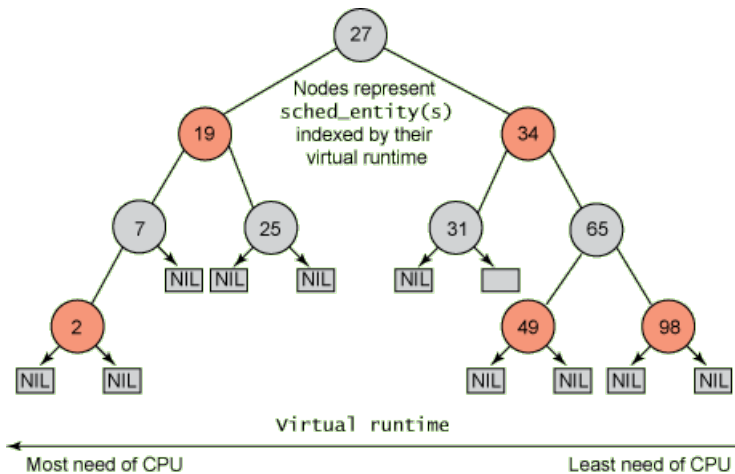


- interně jádro pracuje s vlákny i procesy stejně
- proces/vlákno \implies task (účastní se plánování)
- systémové volání `clone` – zobecněný `fork` (umožňuje definovat, které struktury se mají sdílet)
 - paměťový prostor
 - otevřené soubory
 - I/O
 - id rodiče
 - ...
- stavy úloh: běžící, připravené 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 $O(\log n)$.
- do verze 6.6





- od verze 6.6
- některé tasky nepotřebují moc procesorového času, ale potřebují jej co nejrychleji (vyžadují nízkou latenci); jiné procesy potřebují více času, ale není zase důležité, kdy jej dostanou)
- toto CFS neumí jednoduše podchytit
- EEVDF (poodobně jako CFS) se snaží jednotlivým taskům přiřadit odpovídající podíl procesorového času
- pro každý task je spočítán rozdíl mezi časem, který měl dostat a který dostal, tzv. *lag*
- pouze tasky, které mají nezáporný lag jsou označeny jako způsobilé (*eligible*) k běhu
- doba, kdy task bude způsobilý k běhu se označuje jako *eligible time*
- pro každý task je určena hodnota *virtual deadline* jako součet velikosti časového kvanta a *eligible time*
- je vybrán task s nejmenší hodnotou *virtual deadline*
- přirozeně řeší požadavky na latenci (jde nastavit pro každý task, jako `latency_nice` pomocí `shed_setattr`) a rovnoměrné dělení procesorového času



- 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

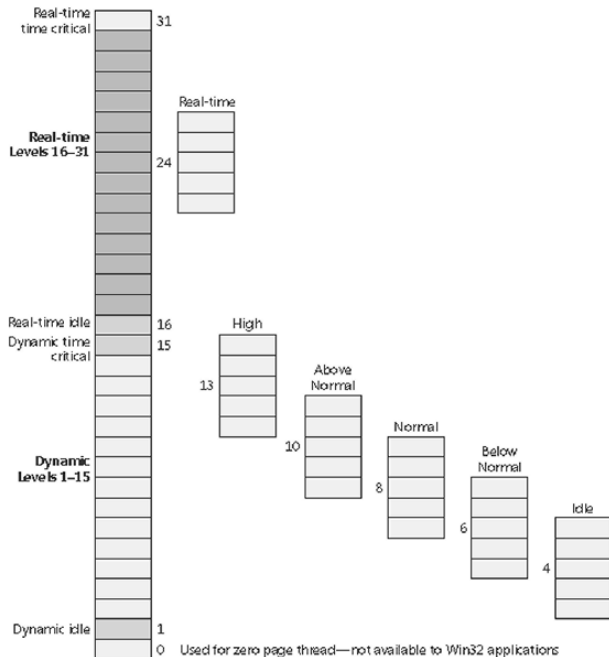
- initialized – během inicializace vlákna
- ready – čekající na běh (z těchto vláken vybírá scheduler další pro běh)
- standby – vlákno připraveno k běhu na konkrétním CPU
 - přechod do running
 - přechod do ready (pokud vlákno s vyšší prioritou přeslo do režimu standby)
- running – vlákno běží; možné 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



- symmetric multiprocessing (SMP) – dominantní řešení, sdílená paměť
- bootstrap processor – inicializace systému
- další procesory aktivovány podle potřeby
- každý procesor vykonává zadaný proces/vlákno odděleně
- úlohy přepínány na základě přerušení
- typické řešení – každé jádro má množinu procesů/vláken, které zpracovává (má vlastní plánovač), v delších intervalech (stovky ms) je provedeno vyvážení úloh mezi procesory
- není žádoucí migrovat úlohy mezi jádry (lepší využití cache)
- možné nastavit affinitu (procesu/vlákna), aby využívalo konkrétní jádra
- situaci komplikují hybridní architektury (různě výkonná a úsporná jádra)