



Operační systémy

# Procesy

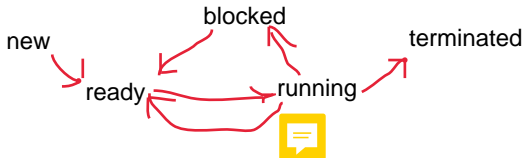
Petr Krajča



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 činností (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  $\Rightarrow$  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
- $\Rightarrow$  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)  $\Rightarrow$  latence
- Jak zvolit velikost?  $\Rightarrow$  interaktivita  $\times$  odvedená práce
- CPU-I/O Burst cycle: pravidelné střídání požadavků na CPU a I/O
- $\Rightarrow$  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
- u interaktivních systémů lze použít informace o využití CPU

## Shortest Remaining Time Next

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



# Algoritmy pro plánování procesů (2/3)

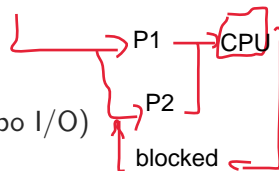
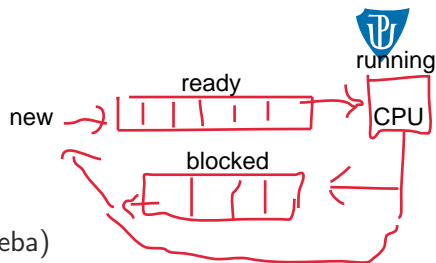
- vhodné pro interaktivní systémy

## 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
- dynamická nejlepší snaha – 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

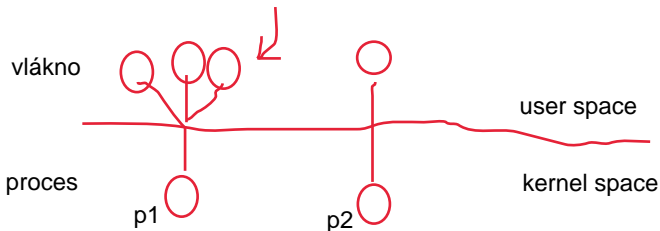
- 1:1 – systémy, kde proces = vlákno
- 1:N – systémy, kde proces může mít více vláken (nejčastější řešení)
- N:1/M:N – více procesů pracuje s jedním vláknem (clustery, spíše hypotické řešení)

## Implementace vláken

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

## V uživatelském prostoru

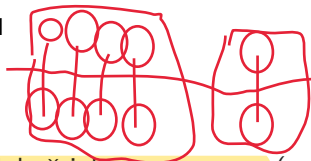
- proces sám se stará o správu a přepínání vláken
- vlastní tabulka vláken
- nejde použít preempece  $\Rightarrow$  kooperativní přepínání (rychlé – není potřeba systémové volání)
- možnost použít plánovací algoritmus dle potřeby
- problém s plánováním v rámci operačního systému (OS neví nic o vláknech)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



! dostanú rovnako času bez  
ohľadu, ktorý proces má  
koľko vláken



p1



p2



## 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  $\Rightarrow$  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