Plánování

PB 152 ◇ Operační systémy

Jan Staudek
http://www.fi.muni.cz/usr/staudek/vyuka/

Verze: jaro 2017

Osnova přednášky

Motivace:

- ✓ V multitaskingových systémech existuje více procesů připravených k běhu
- ✓ procesorů je ve výpočetním systému (prakticky vždy) méně než procesů
- ✓ OS musí rozhodovat, který proces poběží jako příští, tj. kterému procesu přidělí (příp. na omezenou dobu) procesor

Osnova:

- ✓ Základní pojmy
- ✓ Kritéria kvality plánování
- ✓ Plánovací algoritmus FCFS, režim fronty
- ✓ Plánovací algoritmus SPF, přednost mají krátké procesy
- ✓ Prioritní plánovací algoritmus, přednost mají prioritní procesy
- ✓ Plánovací algoritmus Round-Robin, spravedlivé čerpání kapacity CPU
- ✓ Plánování multiprocesorů

Klasifikace metod plánování

Podle cílových objektů

- □ plánování CPU – cíl přednášky
  - ✓ plánují se běhy procesů / vláken na CPU
- □ IO plánování
  - ✓ plánuje se pořadí plnění požadavků procesů na IO
  - ✓ předmět hlubšího studia v PV 062

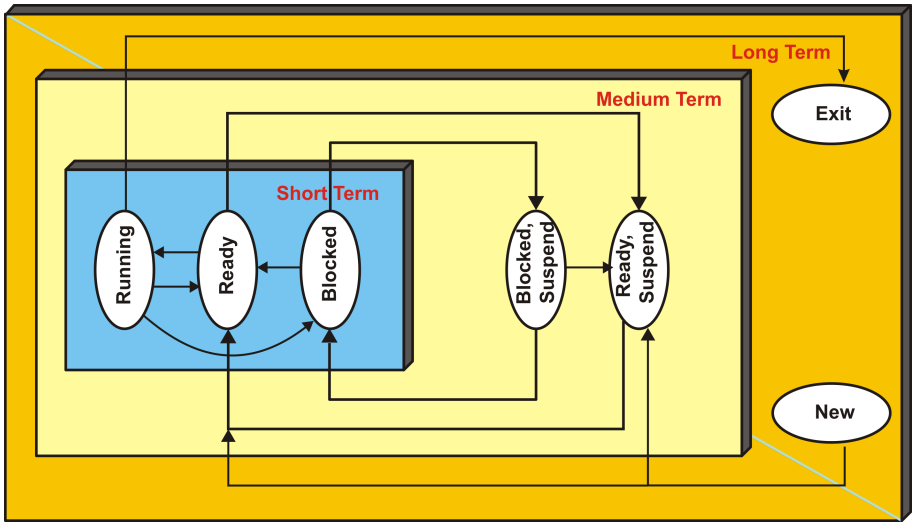
Klasifikace metod plánování podle času uplatnění

- □ krátkodobé, operativní plánování, hlavní cíl přednášky
  - ✓ krátkodobý plánovač (operační plánovač, dispečer, *dispatcher*):
  - ✓ rozhodování, kterému procesu /vláknu OS přidělí CPU
  - ✓ vyvoláván velmi často, desítky – stovky milisekund, musí být rychlý
  - ✓ samozřejmá součást správy procesoru
- □ střednědobé, taktické plánování
  - ✓ Střednědobý plánovač (taktický plánovač)
  - ✓ taktika využívání omezené kapacity FAP při multitaskingu, rozhodování, které procesy mohou využívat prostor hlavní paměti, logicky tudíž náleží do správy hlavní paměti
  - ✓ řídicí algoritmus techniky označované pojmem *swapping* – vybírá proces, který je nutné zařadit mezi odsunuté procesy (odebírá mu prostor ve FAP) a vybírá odsunutý proces, kterému lze opět přidělit prostor ve FAP

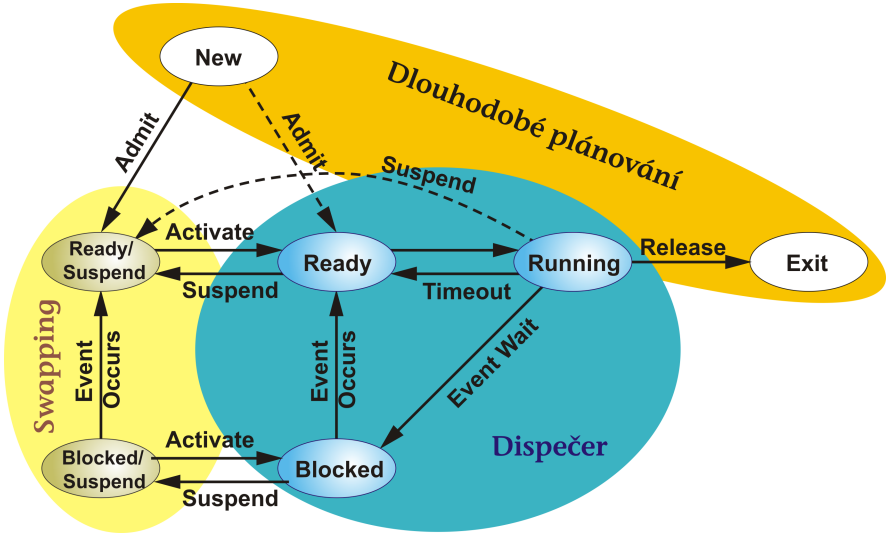
Klasifikace metod plánování podle času uplatnění

- dlouhodobé, strategické plánování
  - ✓ dlouhodobý plánovač (strategický plánovač, *job scheduler*)
  - ✓ má se nový proces zařadit mezi aktivní procesy ?
  - ✓ definuje stupeň multiprogramování
  - ✓ je vyvoláván řídce, nemusí být rychlý
  - ✓ může být součástí správy procesů, součásti rozhraní OS, . . .

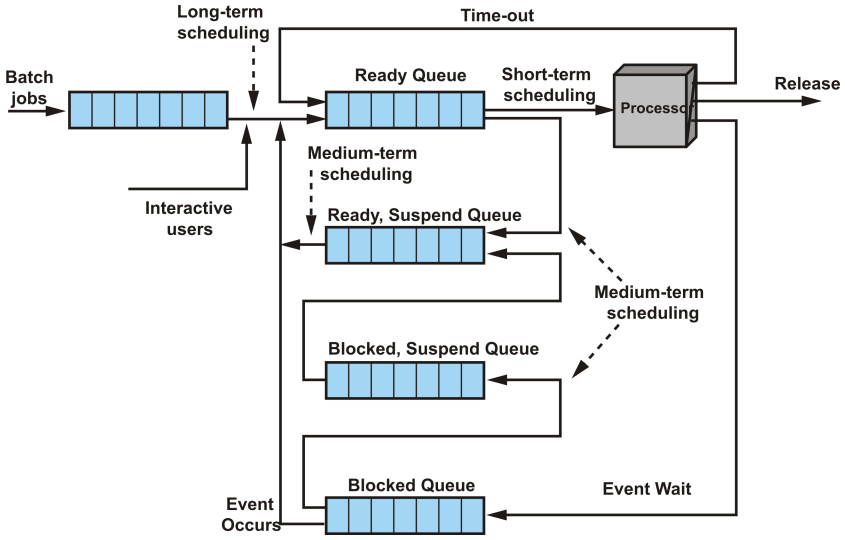
Klasifikace metod plánování podle času uplatnění



Pro připomenutí – stavový diagram procesů



Frontový model metod plánování



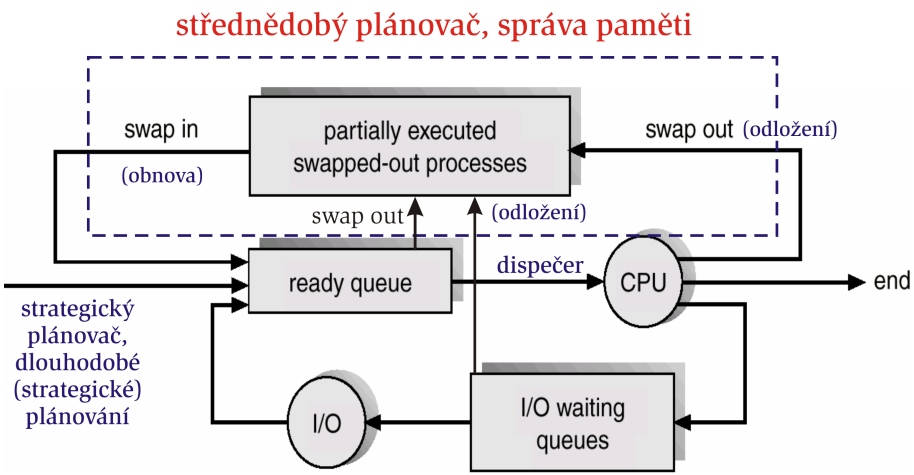
Dlouhodobé plánování

- Zadání úlohy ke zpracování / programu se stane procesem a předá se do fronty připravených procesů dispečerovi
- V některých OS se mohou zařazovat nové procesy mezi potlačené procesy, pak o jejich zařazení mezi připravené procesy rozhoduje střednědobý plánovač
- V OS s možností dávkového zpracování se nově zadané úlohy řadí do fronty úloh na disku a z ní dlouhodobý plánovač vybírá nově vytvářený proces
  - ✓ udržuje efektivní stupeň multitaskingu
  - ✓ pořadí výběru může být typu
    - FCSF (*first-come-first-served*)
    - prioritní
    - určené požadavkem na vyváženost IO a CPU činnosti, . . .

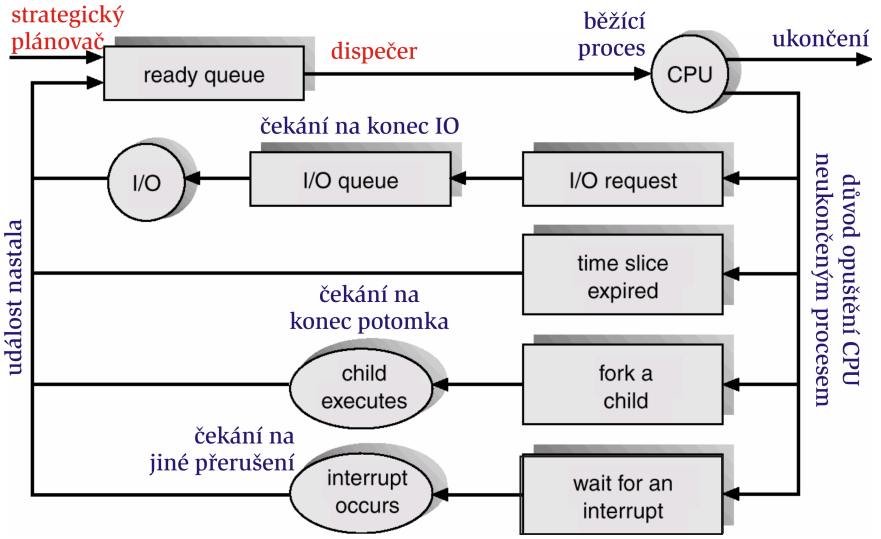
Dlouhodobé plánování

- V interaktivně orientovaných systémech dlouhodobý plánovač může určovat stav nasycenosti systému a novým uživatelům sdělovat nemožnost připojení

Střednědobý plánovač, odkládání (suspending) procesů



Krátkodové plánování – příčiny získání/odebrání CPU



Další možné klasifikace metod plánování

Podle charakteru interakce s procesy

- dávkové plánování
- plánování interaktivních procesů
- plánování procesů v real-time prostředí

Podle dynamiky aktualizace plánu

- nepreemptivní plánování, (plánování bez předbíhání)
  - ✓ provádí se po dokončení připraveného plánu
- preemptivní plánování, (plánování s předbíháním)
  - ✓ provádí se v okamžiku změny stavu některého z procesů
  - ✓ plán se dynamicky mění

Kritéria (metriky) kvality plánování

Uživatelsky orientovaná kritéria

- Doba obrátky, *Turnaround time*
  - ✓ doba běhu + doba čekání na zdroje vč. CPU
  - ✓ vhodná míra pro dávkové zpracování
  - ✓ přirozená je snaha o minimalizaci doby obrátky v dimenzi doby čekání
  - ✓ normalizovaná doba obrátky = doba obrátky / doba běhu
- Doba reakce, *Response time*
  - ✓ míra pro interaktivní systémy
  - ✓ doba od zadání požadavku do doby očekávané reakce
  - ✓ cíl – snaha o minimalizaci pro co největší komunitu uživatelů
  - ✓ v interaktivně orientovaných systémech mívají interaktivní úlohy přednost před dávkovými úlohami

Kritéria (metriky) kvality plánování

- Časové limity, *Deadlines*
  - ✓ pokud jsou zadané, plnění ostatních cílů se musí upozadit
  - ✓ vhodné pro real-time systémy
- časová proporcionalita
  - ✓ např. čekání 45 s na uzavření modemového spojení je akceptovatelné doba reakce na spuštění procesu z terminálu 45 s je neakceptovatelná

Kritéria / metriky kvality plánování

Systémově orientovaná kritéria

- Propustnost, *Throughput*
  - ✓ počet procesů dokončených za jednotku času
  - ✓ přirozená je snaha o maximalizaci propustnosti
  - ✓ požadavek dosažení vysoké propustnosti nebývá kompatibilní s požadavkem minimalizace doby obrátky
- Využití CPU
  - ✓ maximalizace ve víceuživatelských systémech
  - ✓ pro real-time systémy a 1-uživatelské systémy nepodstatné kritérium
- Spravedlivost, *Fairness*
  - ✓ porovnatelné procesy musí získat porovnatelnou obsluhu
  - ✓ pokud uživatel nebo systém neřekne jinak, mají všechny procesy stejnou šanci, vč, ochrany před stárnutím

Kritéria / metriky kvality plánování

- Prosazování priorit
  - ✓ upřednostňování procesů označených jako přednostní, prioritní
- Vyrovnávání zátěže
  - ✓ oblast střednědobého a dlouhodobého plánování
  - ✓ udržování využitelnosti systémových zdrojů
  - ✓ upřednostňování procesů řídce využívajících kritické, úzkoprofilové zdroje
  - ✓ budou-li se upřednostňovat procesy vázané na CPU, budou IO často v prostojích
  - ✓ budou-li se upřednostňovat procesy vázané na IO, bude CPU často v prostojích
  - ✓ ideál – multiprogramování se účastní vhodný mix procesů vázaných na CPU a na IO

Váha kritérií podle cílové oblasti

- Kritéria jsou nezávislá, nelze optimalizovat všechny současně.
- Interaktivní uživatelské systémy požadují
  - ✓ minimalizaci doby reakce – přepínání CPU mezi procesy musí být časté, to zvyšuje systémovou režii, takže se snižuje propustnost
  - ✓ minimalizaci doby obrátky
  - ✓ maximalizaci počtu interaktivních uživatelů (tj. i procesů)
  - ✓ proporcionalitu plnění očekávání uživatelů (spravedlivost) systémech
- Dávkové systémy požadují
  - ✓ maximalizaci propustnosti
  - ✓ maximalizaci využívání CPU

Váha kritérií podle cílové oblasti

- Real-time systémy požadují
  - ✓ dodržování časových limitů, zamezení ztrátám dat
  - ✓ přepověditelnost – zamezení degradace kvality v multimediálních systémech
- OS obecně požadují
  - ✓ maximalizaci propustnosti
  - ✓ minimalizaci dob obrátek
  - ✓ maximalizaci využívání CPU při zajištění proporčního využívání všech komponent počítače
  - ✓ spravedlivost odvozenou z prosazované politiky zpracování, dodržování priorit, ...
  - ✓ minimalizaci potřebného výkonu OS, minimalizaci systémové režie

Požadavky na plánování CPU

- plánování musí racionálně zohledňovat očekávání uživatelů
  - ✓ mějme 2 procesy:
    - korekce obrazovky po uzavření okna
    - odeslání e-mailu – zdržení o 2s je akceptovatelné
  - ✓ zdržení korekce obrazovky po uzavření okna kvůli odesílání mailu o 2s je neakceptovatelné
  - ✓ zdržení odesílání mailu kvůli korekci obrazovky po uzavření okna o 2s je akceptovatelné
- proces = střídání dávek CPU (běhu) a čekání na konec IO
- maximalizace využití CPU vyžaduje prokládání dávek CPU různých procesů, počet prokládání musí být minimální
  - ✓ přepnutí kontextu mezi procesy je složité (100 K instrukcí, ...)
    - přepnutí z uživatelského režimu do privilegovaného režimu
    - uchování stavu CPU
    - uchování stavu procesu
    - obnova stavu procesu, ...

Požadavky na plánování CPU

- je žádoucí upřednostňování procesů orientovaných na IO (disk)
  - ✓ trvalé urychlování CPU způsobuje, že většina procesů se stává více vázaná na IO (disk)
- kdy vydávat plánovací rozhodnutí ?
  - ✓ vytvořil se nový proces – má běžet rodič nebo potomek ?
  - ✓ proces skončil – který proces má běžet jak další ?
  - ✓ IO přerušení indikuje konec IO operace –
    - má dále běžet proces čekající na konec této IO operace ?
    - má dále běžet právě běžící proces ?
    - má dále běžet uplně jiný proces ?
  - ✓ uplynul plánovaný časový interval – co dál ?

Plánovač CPU, dispečer, součást jádra OS

- cíl funkcionality – alokace CPU konkrétnímu procesu / vláknu
- dispečer vybírá mezi procesy, které sídlí v hlavní paměti ty, které jsou připravené k běhu – *ready*
- Plánovací rozhodnutí vydává v okamžiku, kdy proces:
  - ✓ vzniká a řadí se mezi připravené procesy
  - ✓ přechází ze stavu běžící do stavu čekající
  - ✓ přechází ze stavu čekající do stavu připravený
  - ✓ končí
  - ✓ a nebo když okolní podmínky indikují potřebu změny alokace CPU, pak z rozhodnutí dispečera může nastat, že proces přechází ze stavu běžící do stavu připravený

Plánovač CPU, dispečer

- role v OS – po provedení vyžádané služby některým procesem nebo funkce aktivované přerušením se předává procesor procesu vybranému krátkodobým plánovačem
- algoritmus předání:
  - ✓ přepnutí kontextu z kontextu OS na kontext procesu
  - ✓ vč. přepnutí režimu procesoru na uživatelský režim
  - ✓ finále předání – skok na odpovídající místo v uživatelském programu pro restart procesu (určuje obraz čítače instrukcí v PCB)
- dispečerské zpoždění – obvyklá definice
  - ✓ doba, kterou potřebuje OS pro pozastavení běhu jednoho procesu a pro start běhu jiného procesu

Studované plánovací politiky / algoritmy

- Chování systému vymezuje politika, chování plánovacího systému vymezuje plánovací politika
- Plánovací politiku implementuje plánovací algoritmus
- Plánovací algoritmus je realizací výběrové funkce
- Výběrová funkce vybírá proces z fronty připravených procesů
  - ✓ Charakteristiky výběrové funkce
    - w – *waiting*, doba čekání, doba ve frontě připravených procesů
    - e – *execution*, doba běhu procesu na CPU
    - s – *process service time*, očekávaná doba realizace procesu, doba potřebná pro realizaci procesu určená / odhadnutá uživatelem, zahrnuje e
  - ✓ Např. *max (w)* je implementací politiky FCFS

Studované plánovací politiky / algoritmy

- Plánování monoprocesorových systémů
  - ✓ First-Come, First-Served (FCFS)
  - ✓ Round Robin, RR, cyklické plánování
  - ✓ Shortest-Proces-Next (SPN)
  - ✓ Shortest-Remaining-Time-First (SRT)
  - ✓ Prioritní plánování
  - ✓ Plánování s více frontami
  - ✓ Fair Share Scheduler (FSS)
- Plánování homogenních multiprocesorů

Studované plánovací politiky

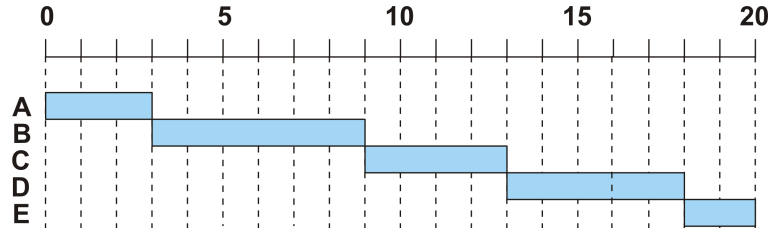
	FCFS	RR	SPN	SRT	HRRN	Feedback
Výběrová funkce	max[w]	constant	min[s]	min[s−e]	max[(w+s)/s]	(viz výklad)
Dynamika	Non-preemptive	Preemptive (po kvantech)	Non-preemptive	Preemptive (at arrival)	Non-preemptive	Preemptive (po kvantech)
Propustnost	Nemá význam	Malá, pokud je č. kvantum malé	Velká	Velká	Velká	Nemá význam
Doba reakce	Může být velká pokud se doby běhu procesů hodně liší	OK pro krátké procesy	OK pro krátké procesy	OK	OK	Nemá význam
Režie	Malá	Malá	Možná velká	Možná velká	Možná velká	Možná velká
Dopad na procesy	Penalizuje krátké procesy a procesy vázané na I/O	Fair treatment	Penalizes long processes	Penalizes long processes	Dobré vyrovnání	May favor I/O bound processes
Stárnutí	Ne	Ne	Možné	Možné	Ne	Možné

FCFS First-Come-First Served
RR Round Robin, cyklické plánování
SPN Shortest Process Next
SRT Shortest Remaining Time
HRRN Highest Response Ratio Next

Procesy použité při studování plánovacích politik

Proces	Čas příchodu	Doba realizce
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

First-Come, First-Served (FCFS)

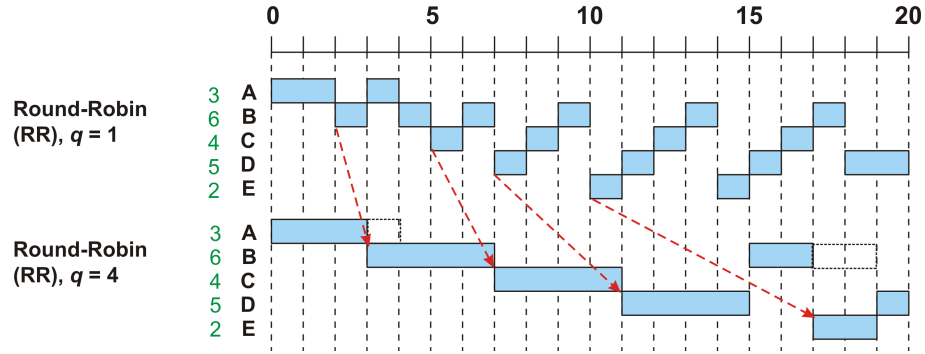


- také *first-in-first-out* (FIFO), resp. přísný frontový režim
- Nepreemptivní politika, uvolněný procesor se přiděluje procesu, který je ve frontě připravených procesů nejdéle
- Jednoduchá implementace
- Vhodné pro dlouhé procesy, upřednostňují se procesy orientované na CPU před procesy orientovanými na IO

Round Robin (RR), cyklické plánování

- preemptivní plánování typu FCFS založené na sledování časových intervalů
- Každý proces dostává CPU cyklicky na malou jednotku času – časové kvantum, q
  - ✓ q = desítky až stovky ms
- po uplynutí doby q
  - ✓ je běžící proces předběhnutý nejstarším procesem ve frontě připravených procesů a
  - ✓ dosud běžící proces se zařazuje na konec této fronty
- je-li ve frontě připravených procesů n procesů, pak každý proces získává $1/n$-tinu doby (výkonu) CPU, najednou získává CPU nejvýše na dobu délky q

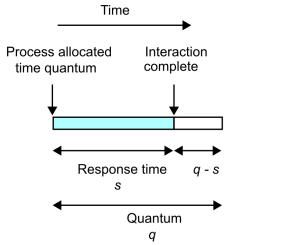
Round Robin (RR), cyklické plánování



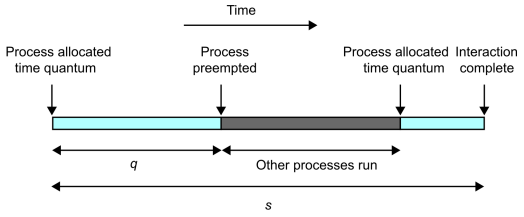
- Žádný proces nečeká na přidělení CPU déle než $(n-1)q$
- Výkonnostní hodnocení:
  - ✓ q velmi velké – plánování se blíží principu FCFS
  - ✓ q velmi malé – krátké procesy se budou rychleji ukončovat, ale CPU se věnuje převážně přepínání kontextů procesů

Round Robin (RR), cyklické plánování

- efektivní politika pro interaktivní víceuživatelské systémy a pro transakční zpracování
- Průměrná doba obrátky se může zlepšit, pokud většina procesu se době q ukončí
- zlaté pravidlo volby q – 80% dávek CPU by mělo být $< q$



kvantum větší než typická interakce



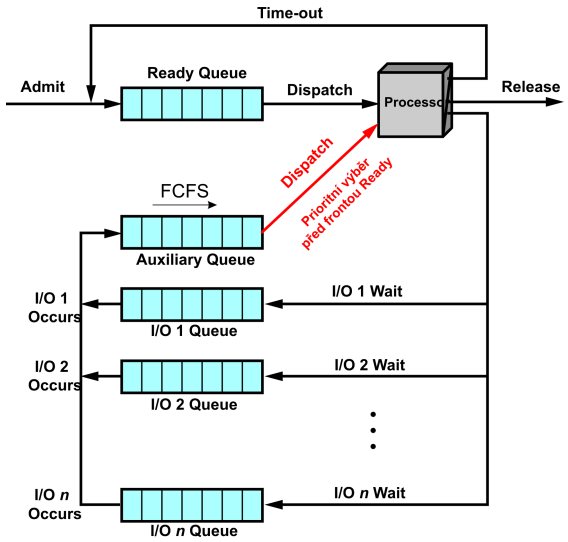
kvantum menší než typická interakce

Round Robin (RR), cyklické plánování

- procesy orientované na CPU získávají nefér výhodu, plně využívají přidělené kvantum
- procesy orientované na IO obvykle kvantum nevyužijí a po ukončení IO čekají ve frontě připravených procesů

  řešením je prioritní plánování procesů orientovaných na CPU

Round Robin (RR), cyklické plánování



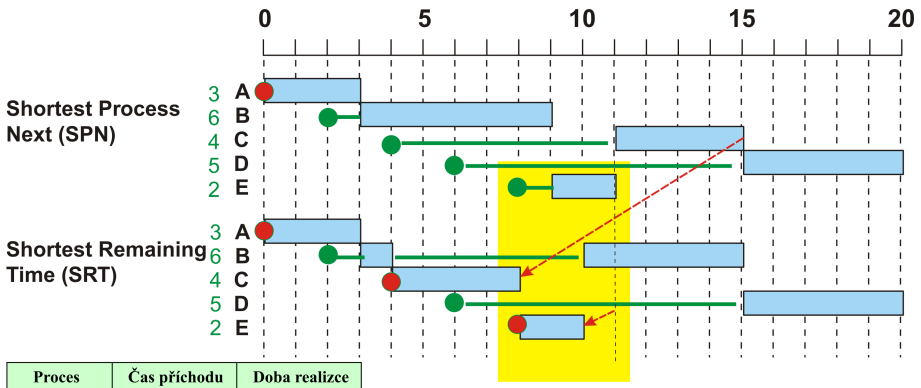
Shortest-Process-Next (SPN)

- Metoda jak redukovat nevhodné chování disciplíny FCFS
- K definici procesu se doplní délka jeho (příští) CPU dávky
- Vybírá se
  - proces s nejkratší (příští) dobou dávky CPU
  - resp. proces, který se ukončí nejdříve
- pravděpodobně mohou stárnout delší procesy
- dávka CPU se musí znát,
  - velikost může udávat vlastník procesu
  - velikost lze odhadovat na základě chování procesu

Shortest-Process-Next, Shortest-Remaining-Time-Next

- Používají se dvě varianty:
  - ✓ nonpreemptivní, bez předbíhání, Shortest-Process-Next (SPN)
    jakmile se CPU předá vybranému procesu, tento nemůže být předběhnutý žádným jiným procesem, dokud svoji dávku CPU nedokončí (nevyčerpá přidělené kvantum času procesoru)
  - ✓ preemptivní, s předbíháním, Shortest-Remaining-Time-First (SRT)
    jakmile se ve frontě *ready* objeví proces s délkou dávky CPU kratší než je doba zbývající k dokončení dávky právě běžícího procesu, nový proces ,,předběhne'' právě běžící proces
- pokud je kritériem kvality plánování průměrná doba čekání, je preemptivní varianta (tj. SRT) optimální algoritmus – pro danou množinu procesů zaručuje minimální průměrnou dobu čekání

Shortest-Process-Next, Shortest-Remaining-Time-Next



Proces	Čas příchodu	Doba realizce
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Jak určit (odhadnout) délku (příští) dávky CPU procesu

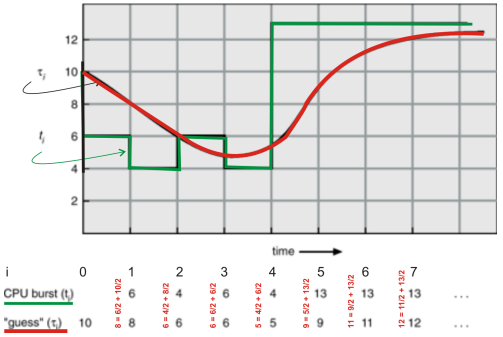
- Skutečný proces nemusí odhadovanou dávku CPU (přidělené kvantum času procesoru) využít (např. vyvolá I/O operaci a přidělenou dobu CPU nedočerpá, skončí dříve, ...)
- Délka příští dávky CPU skutečného procesu se zná přesně jen ve speciálních případech, délku příští dávky CPU skutečného procesu lze pouze odhadnout
- Zkušenostmi prověřená heuristika – odhad pravděpodobné délky příští dávky CPU se odvodí z historie chování procesu
  - ✓ musí se znát předchozí odhady délky dávek CPU
  - ✓ musí se znát jak proces využíval přidělená kvanta CPU
  - ✓ použije se exponenciální průměrování, klasické průměrování odhaduje budoucí chování nepřesně

Exponenciální průměrování

– t_n ... skutečná délka n-té dávky CPU
– τ_{n+1} ... odhad délky příští dávky CPU
– Klasické průměrování: $\tau_{n+1} = \frac{1}{n}\sum_{i=1}^{n} t_i$
– Exponenciální průměrování: $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$
– α, $0 \leq \alpha \leq 1$ ... parametr vlivu historie

– iniciální odhad: $\tau_0 = 10$
– vliv historie: $\alpha = 1/2$
– $\tau_{n+1} = 0.5t_n + 0.5\tau_n =$
$= 0.5(t_n + \tau_n)$

τ_0 se volí jako průměrná délka CPU_dávky v systému nebo se odvodí z typu programu



Exponenciální průměrování – analýza vlastností

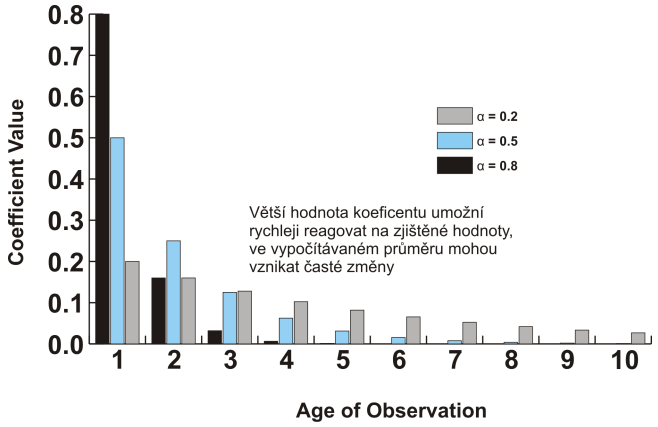
- čím je α menší, tím má historie na odhad menší vliv,
  - ✓ $\alpha = 0$, $\tau_{n+1} = \tau_n$, procesu se přidělují konstantní doby CPU
- čím je α větší, tím více se respektuje (krátká) historie
  - ✓ $\alpha = 1$, pro přidělení doby CPU je určující pouze skutečná poslední CPU dávka, $\tau_{n+1} = t_n$
- Když formuli $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$ rozvineme ($\tau_n = \alpha t_{n-1} + (1-\alpha)\tau_{n-1}$, ...) dostaneme pro obecné α

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \ldots + (1-\alpha)^j \alpha t_{n-j} + \ldots + (1-\alpha)^n \tau_0$$

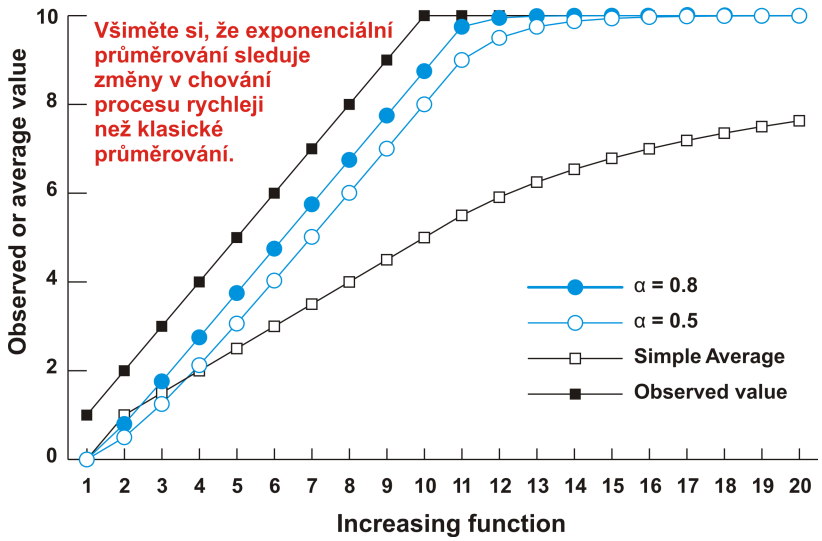
τ_0 se volí jako vhodná konstanta,
$\tau_0 = 0$ prioritizuje nové procesy

Exponenciální průměrování – analýza vlastností

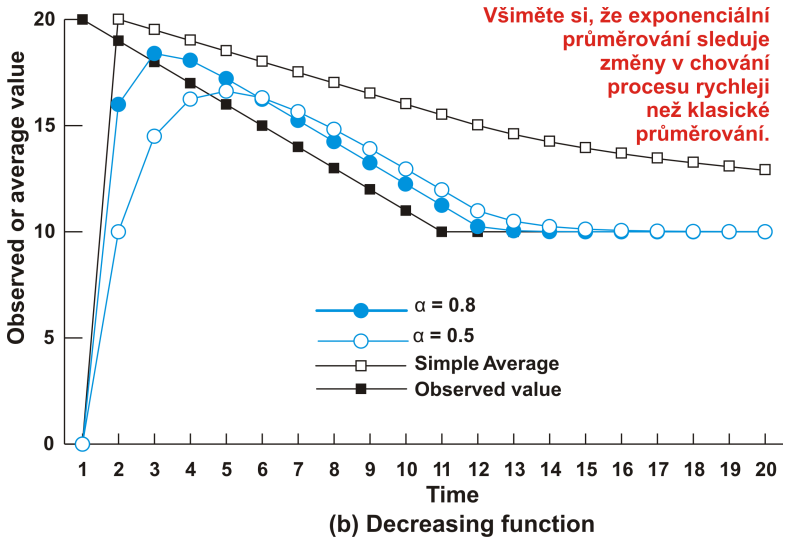
Poněvadž α a $(1-\alpha)$ jsou hodnoty ≤ 1,
každý další term má na τ_{n+1} menší vliv než jeho předchůdce



Exponenciální průměrování vs. jednoduché průměrování

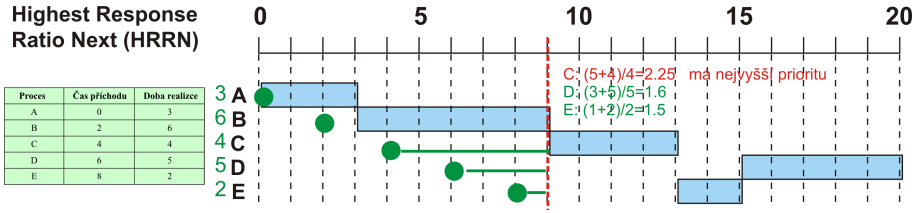


Exponenciální průměrování vs. jednoduché průměrování



Highest Response Ratio Next (HRRN)

- Vybírá se proces s největším poměrem
  skutečné a očekávané doby existence procesu R,
  normalizovaná doba obrátky, $R = max(\frac{w+s}{s})$
- Atraktivní algoritmus,
  protože počítá s dobou existence procesu
  - ✓ preference déle čekajících kratších procesů, ale
  - ✓ při zachování možnosti pozdějšího vítězství i delších procesů



Proces	Čas příchodu	Doba realizce
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Prioritní plánování

- s každým procesem je spojeno prioritní číslo (integer)
  - ✓ prioritní číslo – preference procesu při výběru příště běžícího procesu
  - ✓ CPU se přiděluje procesu s nejvyšší prioritou
  - ✓ nejvyšší prioritě obvykle odpovídá nejnižší prioritní číslo
- Používají se dvě varianty:
  - ✓ nonpreemptivní, bez předbíhání
    jakmile se CPU vybranému procesu předá, tento, dokud dávku CPU nedokončí, nemůže být předběhnut žádným jiným procesem
  - ✓ preemptivní, s předbíháním
    jakmile se ve frontě připravených objeví proces s prioritou vyšší, než je priorita právě běžícího procesu,
    nový proces předběhne právě běžící proces

Prioritní plánování

- SPN je prioritní plánování,
  prioritou je předpovídaná délka příští CPU dávky
- Problém: stárnutí
  - ✓ procesy s nižší prioritou se nemusí nikdy provést
- Řešení stárnutí: zrání procesů
  - ✓ např. priorita procesu se s postupem času (doby čekání, ...) zvyšuje, HRRN

Plánování s víceúrovňovými frontami

- frontu připravených procesů lze dělit např. do dvou front:
  - ✓ fronta přednostní (*foreground*) – interaktivní
  - ✓ fronta procesů na pozadí (*background*) – dávková
- každé frontě náleží specifický plánovací algoritmus, např.
  - ✓ přednostní fronta (interaktivní) – RR
  - ✓ fronta na pozadí (dávková) – FCFS
- jiné možné dělení fronty připravených procesů:
  - ✓ systémové procesy (OS)
  - ✓ interaktivní aplikační procesy cílové aplikace
  - ✓ interaktivní editační procesy cílové aplikace
    (příprava programů, dat, ...)
  - ✓ dávkové procesy cílové aplikace (týdenní plány, ...)
  - ✓ ostatní procesy (hry, studenské úlohy, ...)

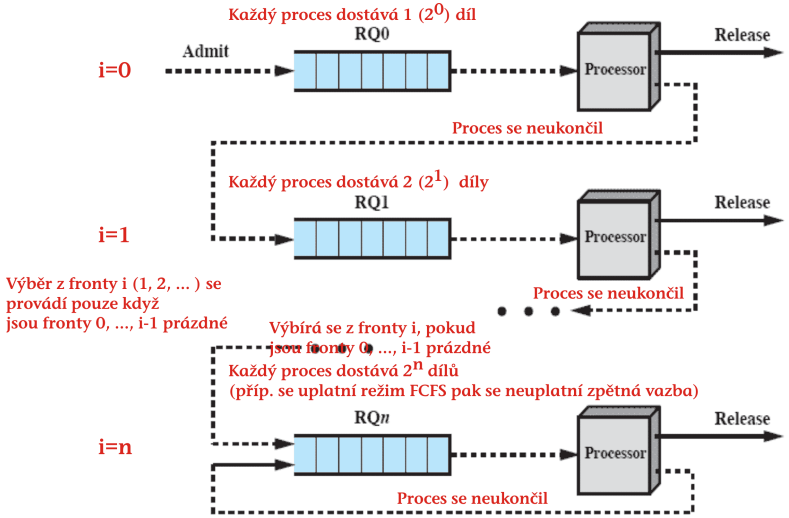
Plánování s víceúrovňovými frontami, 2

- musí se uplatnit vhodná politika plánování, tj. rozhodování, kdy se provádí výběr z jedné a kdy z druhé (další) fronty
  - ✓ pevné prioritní plánování
  - ✓ např. pro prvý příklad na minulé obrazovce –
    - dávková fronta se obsluhuje jen když je interaktivní fronta prázdná
    - ex. hrozba stárnutí procesů v dávkové frontě !
  - ✓ časové řezy
    - pro obsluhu každé fronty se věnuje jistý díl času CPU, po který plánovač procesy vybírá z té které fronty
    - např. 80 % času CPU pro interaktivní úlohy s plánováním typu RR
      20 % času CPU pro dávkové úlohy s plánováním typu FCFS

Zpětnovazební plánování s víceúrovňovými frontami

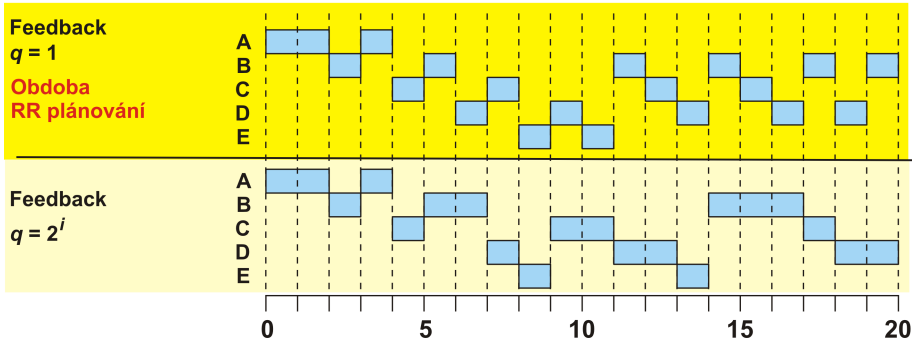
- plánovač může procesy mezi frontami přesouvat
- zpětná vazba – plánovač zná charakteristiky běhů procesů
- takto lze implementovat zrání procesu
- Plánovač s víceúrovňovými zpětnovazební frontami lze definovat např. následujícími parametry
  - ✓ počet front
  - ✓ plánovací algoritmus každé fronty
  - ✓ metoda použitá pro určení kdy proces přeložit mezi procesy s větší preferencí (např. po interakci)
  - ✓ metoda použitá pro určení kdy proces přeložit mezi procesy s menší preferencí (např. po plném vyčerpání časového kvanta)
  - ✓ metoda použitá pro určení do které fronty bude proces vstupovat když požaduje provést nějakou službu (např. vstoupí do fáze krizového řízení aplikace)

Zpětnovazební plánování s víceúrovňovými frontami



Zpětnovazební plánování s víceúrovňovými frontami

Proces	Čas příchodu	Doba realizce
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



Fair Share Scheduler (FSS)

- Plánování v některých OS typu Unix
- Cíl – dát spravedlivou šanci procesům na bázi příslušnosti procesů do skupin
  - ✓ Procesy se dělí do skupin např. na bázi příslušnosti k uživatelům
  - ✓ Každému uživateli je přidělována jistá část výkonu procesoru,
    kdo ho využívá více než je spravedlivé, bude dostávat méně,
    kdo ho využívá méně než je spravedlivé, bude dostávat více

Fair Share Scheduler (FSS)

- Princip
  - ✓ plánování je prioritní
  - ✓ skupina procesů k využívá díl výkonu procesoru W_k
  - ✓ koncept spravedlivosti –
    priorita procesu klesá s růstem doby používání procesoru
    – procesem a
    – skupinou, do které proces patří
  - ✓ skupině s větší vahou W_k, klesá používáním CPU priorita pomaleji
  - ✓ označování priority – vyšší prioritní číslo znamená nižší prioritu

Fair Share Scheduler (FSS)

- ✓ prioritní číslo P_j^i procesu j patřícího skupině k v časovém intervalu i je dané vztahem:

$$P_j^i = B_j + \frac{CPU_j^{i-1}}{2} + \frac{GCPU_k^{i-1}}{(4W_k)}$$

kde B_j je bázová priorita procesu j

CPU_j^{i-1} exponenciálně vážený průměr používání procesoru procesem j v časovém intervalu $i-1$

$GCPU_k^{i-1}$ exponenciálně vážený průměr používání procesoru skupinou k v časovém intervalu $i-1$

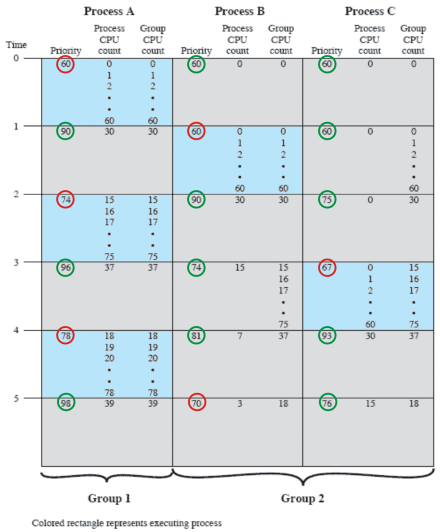
- ✓ Pro výpočty exponenciálně vážených průměrů se používá $\alpha = 1/2$:

$$CPU_j^i = \frac{U_j^{i-1}}{2} + \frac{CPU_j^{i-1}}{2}, \; GCPU_j^i = \frac{GU_j^{i-1}}{2} + \frac{GCPU_j^{i-1}}{2}$$

kde U_j^{i-1} je použití procesoru procesem j v intervalu $i-1$

GU_k^{i-1} je použití procesoru skupinou k v intervalu $i-1$

Zpětnovazební plánování s víceúrovňovými frontami



Váha obou skupin je 0,5

Bázová priorita všech procesů je 60

Měření výkonu CPU
-- 60 x / 1 s přerušení:
inkrementuje se využití CPU běžícím procesem

Každou sekundu se přepočítávají priority

Startuje proces A, po 1 s mají procesy B a C vyšší prioritu, proces A je předběhnut procesem B

Po další sekundě má nejvyšší prioritu proces A

Procesy poběží v pořadí A B A C A B A C ...

Polovinu výkonu dostává proces A (skupina 1)

Polovinu výkonu dostávají procesy B a C (skupina 2)

Klasifikace multiprocesorových systémů

- □ Multiprocesor – počítač vybavený více procesory (CPU, IO procesor, ...)
- □ Distribuovaný multiprocesor, cluster
  - ✓ kolekce relativně autonomních systémů, každý se svou hlavní pamětí a se svým IO podsystémem
  - ✓ propojení sítí, typicky model klient-server
- □ Funkčně specializované procesory, asymetrický multiprocesor
  - ✓ hlavní, univerzální procesor + specializované procesory realizující procesy poskytující služby (IO, ...) procesům hlavního procesoru
- □ Homogenní multiprocesor (HMP), úzce vázaný multiprocesor
  - ✓ symetrický multiprocesor
  - ✓ skupina procesorů sdílejících společnou hlavní paměť a integrovaně řízených operačním systémem

Plánování homogenního multiprocesoru (HMP)

- □ Předmět studia – nezávislý paralelismus, plánování souběžného řešení vzájemně nezávislých procesů na HMP
- □ Přirozené rozšíření monoprocesorového prostředí na multiprocesor
- □ Plánování HMP zahrnuje vzájemně závislé problémy
  - ✓ přidělování procesů k procesorům
  - ✓ použití multitaskingu na jednotlivých procesorech
  - ✓ výběr konkrétního procesu, jak vybírat z fronty připravených procesů ?

Plánování homogenního multiprocesoru (HMP)

Přidělování procesů procesorům v HMP

- Procesory tvoří bank a z banku se přidělují na žádost
- Statické přidělení procesu k procesoru
  - ✓ pro každý procesor se udržuje individuální fronta připravených procesů
  - ✓ vhodné pro skupinové (gangové) plánování, detaily později
- Dynamické přidělování procesu k procesoru
  - ✓ udržuje je globální fronta připravených procesů, společná pro všechny procesory
  - ✓ každý proces může střídavě běžet na kterémkoliv procesoru
  - ✓ vhodné pro dynamické vyrovnání zátěže (*load balancing*)

Plánování homogenního multiprocesoru (HMP)

Kdo o přidělení rozhoduje ?

- jediný, centrální (master) procesor
  - ✓ řeší opakovaně dostupnou plánovací službu z jádra OS na žádost generovanou uvolněním podřízeného procesoru
- kterýkoliv uvolněný procesor, symetrický multiprocesing
  - ✓ udržuje se jedna centrální fronta připravených procesů / sledů
  - ✓ každý volný procesor si sám vyhledává příští sled přesněji – kopie OS běžící na procesoru si sama vyhledává . . .
  - ✓ uvolněný procesor řeší násobně dostupnou plánovací službou jádra OS, což vyžaduje používat vzájemné vylučování v jádru

Plánování homogenního multiprocesoru (HMP)

- Provozovat multitasking na jednotlivých procesorech HMP ?
  - ✓ pokud je dostupných mnoho procesorů není důležité, aby byl každý procesor využíván co možná nejvíce
- Jak vybírat z fronty připravených procesů / vláken v HMP
  - ✓ při plánování na úrovni procesů co nejjednodušší výběr – FIFO
  - ✓ při plánování na úrovni vláken se používají specifické techniky, prioritní výběr či výběr na základě sledování historie nejsou pro plánování vláken výhodné politiky

Plánování vláken v HMP

- Sdílení zátěže, *load sharing*
  - ✓ každý procesor může realizovat kterékoliv vlákno, procesy nejsou přiděleny žádnému konkrétnímu procesoru
  - ✓ globální fronta *ready*, výběr z fronty – FIFO nebo prioritně (priorita klesá s růstem počtu vláken v procesu)
  - ✓ předběhnuté sledy pravděpodobně nebudou pokračovat na stejném procesoru – nelze proto používat ,,cache''paměti procesorů
  - ✓ jestliže jsou všechna vlákna procesu v jedné společné frontě *ready*, pravděpodobně nebudou spuštěná najednou (paralelně)
- Gangy
  - ✓ technika plánování zaručující současný běh více sledů 1 procesu na více procesorech
  - ✓ vhodné pro aplikace jejichž výkon klesá, pokud se neřeší paralelně – např. rozpoznávání scény

Plánování vláken v HMP

- Dedikované přidělení, opak sdílení zátěže
  - ✓ při vytboření procesu se přidělí každému vláknu procesor
  - ✓ při čekání vlákna na IO je procesor v prodlevě, žádný multitasking na přidělených procesorech
  - ✓ v systémech s tisíci či stovkami procesorů není využití procesoru metrikou efektivnosti
  - ✓ eliminace plánování v průběhu procesu může významně urychlit provedení procesu

Plánování v systému Windows

- V jádru neexistuje centrální plánovací vlákno
  - ✓ když vlákno nemůže pokračovat v běhu, vstupuje do režimu jádra a provádí program plánovače a zjišťuje kterému vláknu se předá řízení
- Vlákno nemůže pokračovat v běhu když
  - ✓ musí čekat na událost, semafor, mutex, IO, . . . ,
  - ✓ signalizuje událost (zvedá semafor, . . .)
  - ✓ vyčerpalo přidělené časové kvantum běhu na CPU (došlo k přerušení časovačem)
- Plánovač se rovněž vyvolává když
  - ✓ se dokončí IO operace
  - ✓ uplynul interval časově omezeného čekání

Windows – Plánování je prioritní

		Win32 process class priorities					
		Real-time	High	Above Normal	Normal	Below Normal	Idle
Win32 thread priorities	Time critical	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

- ✓ pro nastavení *Real-time* priority musí mít uživatel speciální oprávnění
- ✓ aplikační vlákna mají priority 15 – 1
- ✓ bázová priorita vlákna = priorita procesu
- ✓ běžná priorita vlákna = bázová priorita + relativní korekce priority

Windows – Plánování, korekce priorit

- Nejvyšší prioritní úroveň je plánovaná v režimu *round-robin*, cyklické plánování
- Navýšení priority vlákna, příklady
  - ✓ po dokončení očekávané IO operace + 1: disk, +2: komunikace, +6: klávesnice, +8: zvuková karta
  - ✓ po události, zvednutí semaforu, . . . : + 1
- Snižování priority vlákna
  - ✓ kdykoliv vlákno vyčerpá časové kvantum CPU
  - ✓ až do úrovně bázové priority

Příklad plánování – Linux, plánování procesů

Linux používá dva algoritmy pro plánování procesů:

- algoritmus pro spravedlivé časové sdílení umožňující předbíhání (fair preemptive scheduling algorithm)
  - ✓ každý proces získá jistý počet kreditů
  - ✓ při každém přerušení časovačem ztrácí běžící proces 1 kredit
  - ✓ proces s 0 kredity se vzdává CPU
  - ✓ jakmile neexistuje žádný připravený proces s kredity, provede se rekreditace, která přidá kredity všem procesům v systému, nejen připraveným, podle pravidla $\boldsymbol{kredity = kredity/2 + priorita}$
- real-time algoritmus pro řešení těch úkolů, pro které je mnohem důležitější absolutní priorita před spravedlivostí
  - ✓ neimperativní (soft) real-time

Příklad plánování – Linux, plánování procesů, 2

- O aplikaci toho kterého plánovacího algoritmu rozhoduje plánovací třída procesu (*process's scheduling class*)
- Linux implementuje FIFO plánovací třídu a round-robin real-time plánovací třídu
- V obou variantách má proces navíc i prioritu
- Plánovač spouští proces nejvyšší priority
- Na stejné prioritní úrovni se vybírá podle doby čekání (FIFO)
- Procesy *plánovací třídy FIFO* běží dokud neskončí nebo se nezablokují
- Procesy *round-robin real-time plánovací třídy* jsou po uplynutí časových kvant předbíhané a řadí se na konec plánovací fronty
  - ✓ round-robin procesy stejné priority se spravedlivě střídají v běhu automaticky

Příklad plánování – Linux, plánování procesů, 3

- V Linuxu se plánováním označuje i spouštění různých „procesů jádra“ (tasks)
  - ✓ spouštění procesů jádra požadovaná během normálních procesů
  - ✓ spouštění procesů jádra vynucená drivery zařízení

Plánování javovských sledů

- JVM používá prioritní preemptivní plánování
- na stejné prioritní úrovni se aplikuje princip FIFO
- JVM plánuje běh sledu když:
  - ✓ běžící sled se vzdává práva běžet běžet
    - konec sledu – sled vystupuje z metody typu *run()*,
    - čekání na událost – např. na konec I/O operace
  - ✓ se stane připraveným sled vyšší priority než běžící sled

Plánování javovských sledů

- sledům se přiděluje stejné kvantum času
- pokud podpůrný OS nepodporuje RR plánování,
  ex. nástroje pro sdělení JVM, že zbytek přiděleného kvanta nepotřebuje a lze proto plánovat další kvantum dalšímu sledu – *thread.yield()* – plánuje se běh dalšího sledu stejné priority
- sledu je priorita přidělena při jeho vytvoření
- JVM priority dynamicky nemění