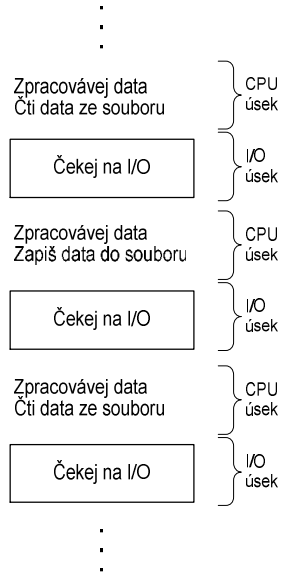
Téma 4 – Plánování práce procesorů

Obsah

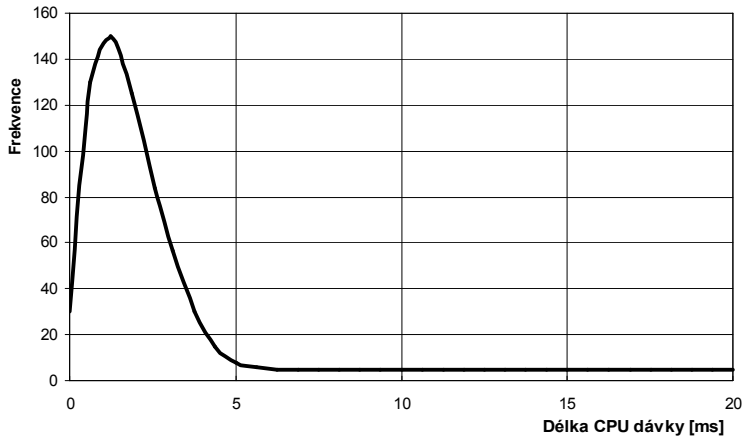
1. Plánování a jeho charakteristiky
2. Plánovací algoritmy a jejich vlastnosti
3. Zpětnovazební plánování
4. Obecný plánovač
5. Plánování v multiprocesorech
6. Systémy reálného času a plánování v nich
7. Algoritmy RMS a EDF

Motivace plánování CPU

- Maximálního využití CPU se dosáhne uplatněním multiprogramování
- Běh procesu = cykly alternujících úseků
  - **[: CPU úsek, I/O úsek :]**
- CPU úsek se může v čase překrývat s I/O úseky jiných procesů
  - místo „úsek“ se často nepřesně používá slovo „dávka“

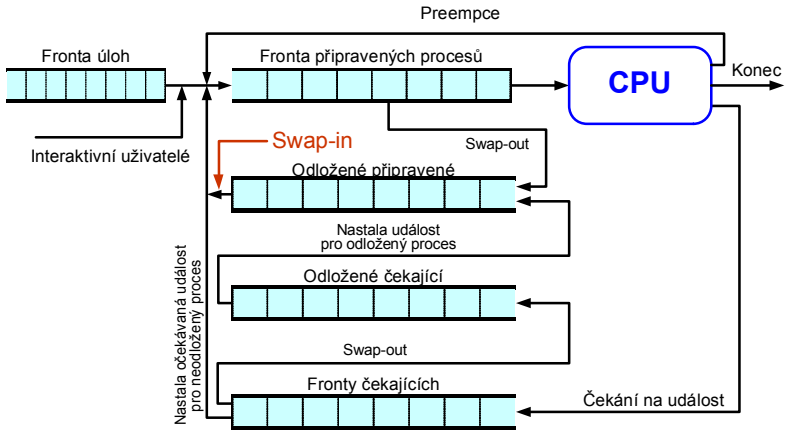


Typický histogram délek CPU dávek



V diagramu je podstatný jeho tvar, nikoliv konkrétní hodnoty

Frontový model plánování CPU



Srovnej se „sedmistavovým diagramem procesů“ v Tématu 03

Kriteria krátkodobého plánování

- Uživatelsky orientovaná
  - čas odezvy
    - doba od vzniku požadavku do reakce na něj
  - doba obrátky
    - doba od vzniku procesu do jeho dokončení
  - konečná lhůta (*deadline*)
    - požadavek dodržení stanoveného času dokončení
  - předvídatelnost
    - Uloha by měla být dokončena za zhruba stejnou dobu bez ohledu na celkovou zátěž systému
    - Je-li systém vytížen, prodloužení odezvy by mělo být rovnoměrně rozděleno mezi procesy
- Systémově orientovaná
  - průchodnost
    - počet procesů dokončených za jednotku času
  - využití procesoru
    - relativní čas procesoru věnovaný aplikačním procesům
  - spravedlivost
    - každý proces by měl dostat svůj čas (ne „hladovění“ či „stárnutí“)
  - vyvažování zátěže systémových prostředků
    - systémové prostředky (periferie, hlavní paměť) by měly být zatěžovány v čase rovnoměrně

Plánovač procesů

- Aktivace plánovače
  - Obslužná rutina přerušení na svém konci vyhlásí tzv. významnou událost v systému
    - např. dokončení přenosu dat, vyčerpání časového kvanta
  - Významná událost aktivuje plánovač, který rozhodne, co dále
  - Plánovač může přepnout kontext ⇒ přechod od jednoho procesu k jinému je VŽDY důsledkem nějakého PŘERUŠENÍ (nebo výjimky)
- Fronta připravených procesů
  - Plánovač rozhoduje, který proces aktivovat
  - Proces v čele fronty dostává procesor a může tak způsobit preempci. Ta může nastat kdykoliv (i bez „vědomí“ běžícího procesu)
  - Fronty nemusí být prosté (FIFO), může se v nich předbíhat dle priorit
  - Určení priority procesu
    - Klíč k dosažení cílů plánovače (spravedlivost, propustnost, ...)
    - Odhadují se měnící se charakteristiky procesu
    - Založeno na měření chování procesu

Plánovací algoritmy

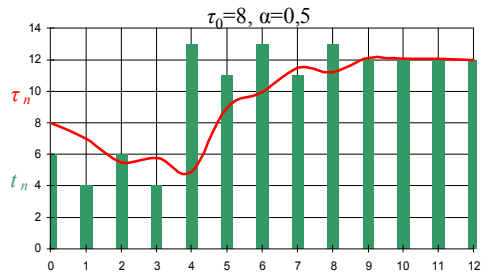
- Ukážeme plánování:
  - FCFS (*First-Come First-Served*)
  - SPN (SJF) (*Shortest Process Next*)
  - SRT (*Shortest Remaining Time*)
  - cyklické (*Round-Robin*)
  - zpětnovazební (*Feedback*)
- Příklad
  - používaný v tomto textu pro ilustraci algoritmů

Proces	Čas příchodu	Potřebný čas (délka CPU dávky)
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- Chování se ilustruje Ganttovými diagramy

Odhad délky příští CPU dávky procesu

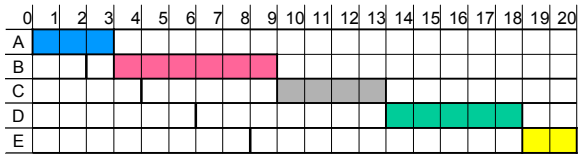
- Délka příští dávky CPU skutečného procesu je známa jen ve velmi speciálních případech
  - Délka dávky se odhaduje a predikuje se na základě nedávné historie procesu
- Nejčastěji se používá tzv. exponenciální průměrování
  - t_n skutečně změřená délka n-té dávky CPU
  - τ_{n+1} odhad délky příští dávky CPU
  - α, $0 \le \alpha \le 1$ parametr vlivu historie
  - $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$
  - τ_0 se obvykle volí jako průměrná délka CPU dávky v systému nebo se odvodí z typu programu
  - Příklad:
    - $\alpha = 0{,}5$
    - $\tau_{n+1} = 0{,}5t_n + 0{,}5\tau_n = 0{,}5(t_n + \tau_n)$



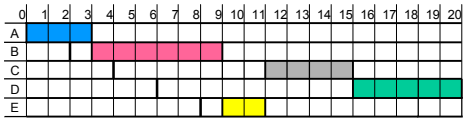
Plánování FCFS

- FCFS = *First Come First Served* – prostá fronta FIFO
- Nejjednodušší nepreemptivní plánování
- Nově příchozí proces se zařadí na konec fronty
- Průměrné čekání může být velmi dlouhé
- Příklad:

$$^{w}T_{Avg} = \frac{0+1+5+7+10}{5} = 4{,}6$$



- Průměrné čekání umíme zredukovat jednoduchým trikem:
  - Např. v čase 9 je procesor volný a máme na výběr procesy C, D a E



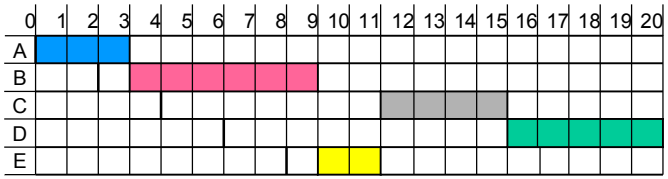
$$^{w}T_{Avg} = \frac{0+1+7+9+1}{5} = 3{,}6$$

Vlastnosti FCFS

- FCFS je primitivní nepreemptivní plánovací postup
- Průměrná doba čekání $^{w}T_{Avg}$ silně závisí na pořadí přicházejících dávek
- Krátké procesy, které se připravily po dlouhém procesu, vytváří tzv. konvojový efekt
  - Všechny procesy čekají, až skončí dlouhý proces
- Proto se pro krátkodobé plánování FCFS samostatně prakticky nepoužívá.
  - Používá se pouze jako složka složitějších plánovacích postupů

Plánování SPN

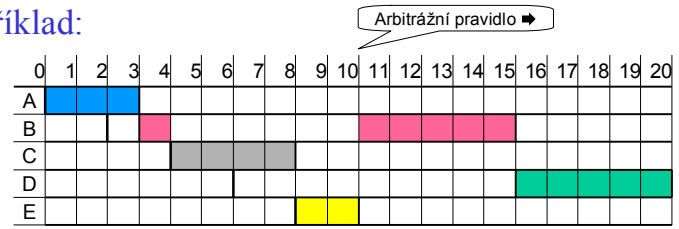
- SPN = *Shortest Process Next* (nejkratší proces jako příští); též nazýváno SJF = *Shortest Job First*
  - Opět nepreemptivní
  - Mezi připravenými procesy se vybírá ten s nejkratší příští dávkou CPU
  - Krátké procesy předbíhají delší, nebezpečí stárnutí dlouhých procesů
  - Je-li kritériem kvality plánování průměrná doba čekání, je SPN optimálním algoritmem, což se dá exaktně dokázat
- Příklad:



$$^{w}T_{Avg} = \frac{0+1+7+9+1}{5} = 3{,}6$$

Plánování SRT

- SRT = *Shortest Remaining Time* (nejkratší zbývající čas)
  - Preemptivní varianta SPN
  - CPU dostane proces, který potřebuje nejméně času do svého ukončení
  - Jestliže existuje proces, kterému zbývá k jeho dokončení čas kratší, než je čas zbývající do skončení procesu běžícího, dojde k preempci
- Příklad:



$$^{w}T_{Avg} = \frac{0+1+0+9+0}{5} = 2{,}0$$

Prioritní plánování (1)

- Každému procesu je přiřazeno prioritní číslo (integer)
  - Prioritní číslo – preference procesu při výběru procesu, kterému má být přiřazena CPU
  - CPU se přiděluje procesu s nejvyšší prioritou
  - Nejvyšší priorité obvykle odpovídá (obvykle) nejmenší prioritní číslo
    - Ve Windows je to obráceně
- Existují se opět dvě varianty:
  - Nepreemptivní
    - Jakmile se vybranému procesu procesor předá, procesor mu nebude odňat, dokud se jeho CPU dávka nedokončí
  - Preemptivní
    - Jakmile se ve frontě připravených objeví proces s prioritou vyšší, než je priorita právě běžícího procesu, nový proces předběhne právě běžící proces a odejme mu procesor
- SPN i SRT jsou vlastně případy prioritního plánování
  - Prioritním číslem je predikovaná délka příští CPU dávky
    - SPN je nepreemptivní prioritní plánování
    - SRT je preemptivní prioritní plánování

Prioritní plánování (2)

- Problém stárnutí (*starvation*):
  - Procesy s nízkou prioritou nikdy nepoběží; nikdy na ně nepřijde řada
    - Údajně: Když po řadě let vypínali v roce 1973 na M.I.T. svůj IBM 7094 (tehdy jeden z největších počítačů na světě), našli proces s nízkou prioritou, který čekal od roku 1967.
- Řešení problému stárnutí: zrání procesů (*aging*)
  - Je nutno zařídit, aby procesu rostla priorita na základě jeho historie a doby strávené mezi připravenými procesy
    - Během čekání na procesor se priorita procesu zvyšuje

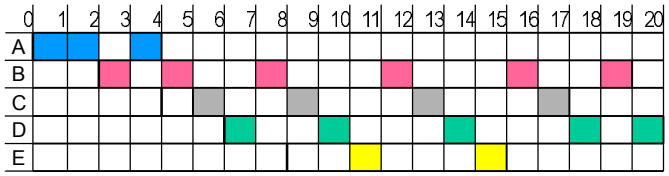
Cyklické plánování

- Cyklická obsluha (*Round-robin*) – RR
- Každý proces dostává CPU periodicky na určitý časový úsek, tzv. časové kvantum, délky q (~ desítky ms)
  - Shodná priorita všech procesů
  - Po vyčerpání kvanta je běžícímu procesu odňata CPU ve prospěch nejstaršího procesu ve frontě připravených 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 CPU
  - Žádný proces nedostane 2 kvanta za sebou
    - ledaže by byl jediný připravený
  - Žádný proces nečeká na přidělení CPU déle než $(n-1)q$ časových jednotek
- Z principu preemptivní plánování

Cyklické plánování (2)

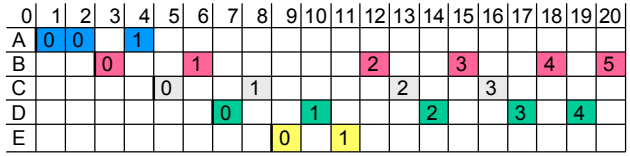
- Efektivita silně závisí na velikosti kvanta
  - Veliké kvantum – blíží se chování FCFS
    - Procesy dokončí svoji CPU dávku dříve, než jim vyprší kvantum.
  - Malé kvantum => časté přepínání kontextu => značná režie
- Typicky
  - Dosahuje se průměrné doby obrátky delší oproti plánování SRT
  - Výrazně lepší je čas odezvy
  - Průměrná doba obrátky se může zlepšit, pokud většina procesů se době q ukončí
  - Empirické pravidlo pro stanovení q: cca 80% procesů by nemělo vyčerpat kvantum

Příklad:

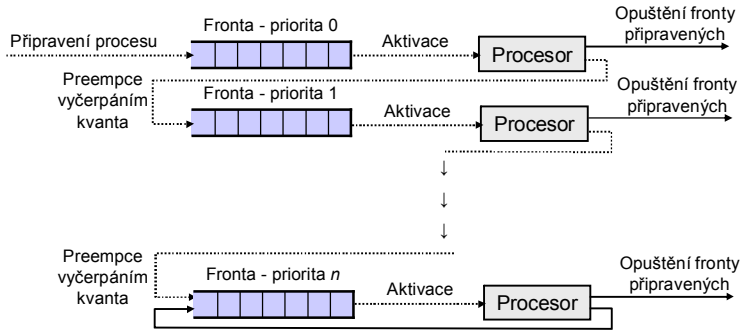


Zpětnovazební plánování

- Základní problém:
  - Neznáme předem časy, které budou procesy potřebovat
- Východisko:
  - Místo odhadování budeme penalizovat procesy, které běžely dlouho
- Řešení:
  - Dojde-li k preempci přečerpáním časového kvanta, procesu se snižuje priorita
  - Implementuje se obvykle pomocí víceúrovňových front
    - pro každou prioritu jedna
  - Nad každou frontou samostatně běží algoritmus určitého typu plánování
    - obvykle RR s různými kvanty a FCFS pro frontu s nejnižší prioritou
- Příklad

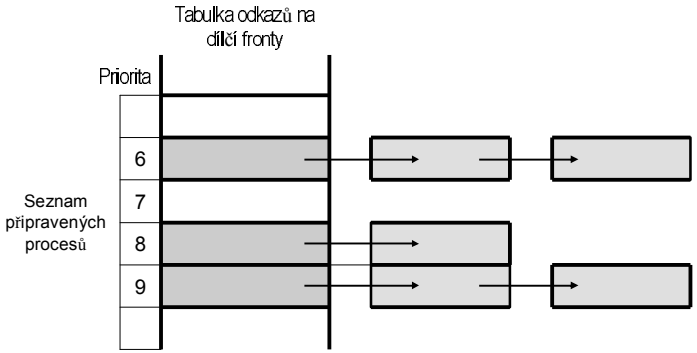


Víceúrovňové fronty



- Proces opouštějící procesor kvůli vyčerpání časového kvanta je přeřazen do fronty s nižší prioritou
- Fronty s nižší prioritou mohou mít delší kvanta
- Problém stárnutí ve frontě s nejnižší prioritou
  - Řeší se pomocí zrání (*aging*) – v jistých časových intervalech se zvyšuje procesům priorita, a tak se přemísťují do „vyšších“ front

Implementace víceúrovňových front



- Implementace JOS musí dbát na rychlost přístupu k datovým strukturám, aby přepínání kontextu bylo co nejrychlejší
- Fronta na procesor je rozdělena na dílčí fronty, pro každou prioritu jedna samostatně uspořádaná způsobem FIFO.

Obecný plánovač (1)

- Tři komponenty:
  1. Rozhodování
     - Kterému procesu přidělit procesor (resp. i který z více procesorů)
  2. Prioritní funkce
     - Všem připraveným procesům určit efektivní priority
  3. Arbitrážní pravidlo
     - Co činit, jsou-li dva procesy rovnocenné (tj. mají shodnou prioritu)
- Rozhodování
  - Pracuje nad (třeba i víceúrovňovou) frontou připravených
  - Rozhoduje se když:
    1. Nový proces se stane připraveným
    2. Běžící proces skončí
    3. Čekající proces změní svůj stav na připravený
    4. Běžící proces se začne čekat
    5. Běžící proces vyčerpá časové kvantum
    6. Připravenému procesu vzroste priorita nad prioritu procesu běžícího
  - Vždy důsledek nějaké výjimky (přerušení)

Obecný plánovač (2)

- Prioritní funkce
  - Určuje efektivní prioritu připravených procesů
  - Může záviset na dynamických vlastnostech procesů
  - Základní prioritní úroveň
    - Vysoká pro interaktivní procesy
    - Nízká pro dávkové zpracování „na pozadí“
  - Nároky na paměť (velká režie odkládání):
    - „Malý“ proces – rychlé odkládání, lze snáze obsluhovat mnoho malých procesů
  - Časové vlastnosti procesu
    - Relativní spotřeba časových kvant
    - Celkový spotřebovaný čas
    - Vyšší priorita krátkých procesů
- Arbitrážní pravidlo
  - Aplikuje se na připravené procesy se stejnou efektivní prioritou
  - Náhodná volba
    - Používá se zřídka – nedává příliš smysl
  - Chronologické řazení
    - Nejčastější – klasická fronta (FIFO); proces s touž prioritou se řadí na konec fronty

Plánování v multiprocesorech

- Přiřazování procesů (vláken) procesorům:
  - Architektura „master/slave“
    - Klíčové funkce jádra běží vždy na jednom konkrétním procesoru
    - Master odpovídá za plánování
    - Slave žádá o služby mastera
    - Nevýhoda: dedikace
      - Přetížený master se stává úzkým místem systému
  - Symetrický multiprocesing (SMP)
    - Všechny procesory jsou si navzájem rovny
    - Funkce jádra mohou běžet na kterémkoliv procesoru
  - SMP vyžaduje podporu vláken v jádře
- Proces musí být dělen na vlákna, aby SMP byl účinný
  - Aplikace je pak sada vláken pracujících paralelně do společného adresního prostoru
  - Vlákno běží nezávisle na ostatních vláknech procesu
  - Vlákna běžící na různých procesorech dramaticky zvyšují účinnost systému

Symetrický multiprocesing (SMP)

- Jedna společná (globální) fronta pro všechna vlákna
  - Fronta „napájí“ společnou sadu procesorů
    - Fronta může být víceúrovňová dle priorit
  - Každý procesor si sám vyhledává příští vlákno
    - Přesněji: instance plánovače běžící na procesoru si je sama vyhledává ...
- Fakta:
  - Plánovací politiky pro přidělování procesorů v multiprocesoru nemají takový význam jako v monoprocesoru
  - Možnost souběžného běhu vláken jednoho procesu na více procesorech zvyšuje potenciálně dostupný výkon pro běh aplikací
- Problémy
  - Jedna centrální fronta připravených sledů vyžaduje používání vzájemného vylučování v jádře
    - Kritické místo v okamžiku, kdy si hledá práci více procesorů
    - Předběhnutá (přerušená) vlákna nebudou nutně pokračovat na stejném procesoru – nelze proto plně využívat cache paměti procesorů
- Používáno ve
  - Windows (>NT), Linux, Mac OS X, Solaris, BSD4.4

Poznámky k plánování v multiprocesorech

- Dynamické vyvažování (*Load Balancing*) využití jednotlivých procesorů:
  - Používá se v systémech, kde každý procesor má svoji frontu připravených
  - V systémech s globální frontou není dynamické vyvažování potřebné
    - Problémem naopak je, že pokud jsou všechny sledy instrukcí (CPU dávky) v jedné společné frontě připravených, nemohou se jednoduše spustit současně a běžet paralelně
- Používají se různá (heuristická) pravidla (i při globální frontě):
  - Afinita vlákna k procesoru – použij procesor, kde vlákno již běželo (možná, že v cache procesoru budou ještě údaje z minulého běhu)
  - Použij nejméně využívaný procesor
- Mnohdy značně složité
  - při malém počtu procesorů (≤ 4) může přílišná snaha o optimalizaci plánování vést až k poklesu výkonu systému

Systémy reálného času (RT)

- Obvykle malé systémy se specializovaným použitím
  - Často vestavěné (*embedded*)
- Správná funkce systému závisí nejen na logickém (či numerickém) výsledku ale i na čase, kdy bude výsledek získán
  - Logicky či numericky správný výsledek dodaný pozdě je k ničemu
- Úlohy a procesy reagují na události pocházející zvenčí systému a navenek dodávají své výsledky
  - Nastávají v „reálném čase“ a potřebná reakce musí být včasná
- Příklady
  - Řízení laboratorních či průmyslových systémů
  - Robotika
  - Řízení letového provozu
  - Telekomunikační aplikace (digitální ústředny)
  - Vojenské systémy velení a řízení
  - ...

Charakteristiky OS RT

- Determinismus
  - Operace jsou prováděny ve fixovaných, předem určených časech nebo časových intervalech
  - Reakce na přerušení musí proběhnout tak, aby systém byl schopen obsluhy všech požadavků v požadovaném čase (včetně vnořených přerušení)
- Uživatelské řízení
  - Uživatel (návrhář systému) specifikuje:
    - Priority
    - Práva procesů
    - Co musí vždy zůstat v paměti
- Spolehlivost
  - Degradace chování může mít katastrofální důsledky
- Zabezpečení
  - Schopnost systému zachovat v případě chyby aspoň částečnou funkcionalitu a udržet maximální množství dat

Požadavky na OS RT

- Extrémně rychlé přepínání mezi procesy či vlákny
- OS musí být malý
- Multiprogramování s meziprocesními komunikačními nástroji
  - semafory, signály, události ➡
- Speciální souborové systémy s velkou rychlostí
  - RAM disky, souvislé soubory
- Plánování založené na prioritách
  - Pozor: preempce je ale časově náročná
- Minimalizace časových úseků, kdy je vypnut přerušovací systém
- Mnohdy zvláštní hardwarové vybavení
  - hlídací časovače (*watch-dog timers*) a alarmy

Plánování CPU v RT systémech

- Tabulkou řízené statické plánování
  - Určuje pevně, kdy bude který proces spuštěn tak, aby včas skončil
  - Nejčastější případ v uzavřených systémech s předem známými procesy a jejich vlastnostmi
- Preemptivní plánování se statickými prioritami
  - Používá klasický prioritní plánovač s fixními prioritami
    - Může být problematické kvůli velké režii spojené s preempcí, přesto nejčastější v "univerzálních" RT systémech – viz RMS ➡
- Dynamické plánování
  - Za běhu se určuje proveditelnost (splnitelnost požadavků)
  - V tzv. přísných RT systémech (*Hard real-time systems*) obtížně použitelné vlivem značné režie
    - *Hard real-time systems* musí přísně zaručovat dokončení časově kritických procesů do předepsaného termínu

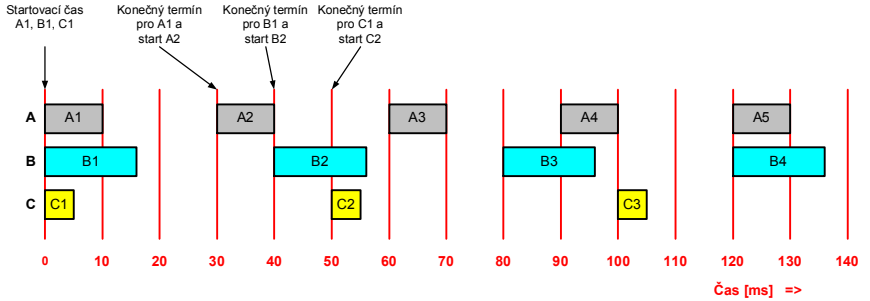
Periodické plánování dle konečného termínu

- Procesy se spouštějí periodicky, perioda je fixní a předem daná
  - Procesům jde zejména o včasné dokončení v rámci své periody
    - Např. periodicky je třeba vzorkovat údaje z čidel
- O každém procesu je znám
  - Potřebný čas (horní hranice délky jeho CPU úseku)
  - Termín začátku a nejzazšího konce každého běhu periodicky spouštěného procesu
- Zjednodušující předpoklady
  - Termín dokončení je identický s počátkem následující periody
  - Periody všech procesů začínají v čase 0
  - Požadavky na systémové prostředky nebudeme uvažovat
    - Předpokládáme, že procesy mají trvale vyčleněné systémové prostředky

Příklad 1

- 3 periodické procesy

Proces	Perioda p_i	Procesní čas T_i	T_i / p_i
A	30	10	1/3 = 0,333
B	40	15	3/8 = 0,375
C	50	5	1/10 = 0,100



Plánovatelnost v periodických úlohách

- Relativní využití strojového času
  - Proces *i* využije poměrnou část $r_i = \frac{T_i}{p_i}$ celkového strojového času, kde T_i je čas běhu a p_i je jeho perioda
  - Celkové využití je $r = \sum_{i=1}^{N} r_i = \sum_{i=1}^{N} \frac{T_i}{p_i}$
  - Aby vše mohlo pracovat musí platit $r = \sum_{i=1}^{N} \frac{T_i}{p_i} \leq 1$ (podmínka plánovatelnosti)
- Náš Příklad 1 $r = \sum_{i=1}^{3} \frac{T_i}{p_i} = \frac{10}{30} + \frac{15}{40} + \frac{5}{50} \cong 0{,}808 < 1$

Plánování RMS

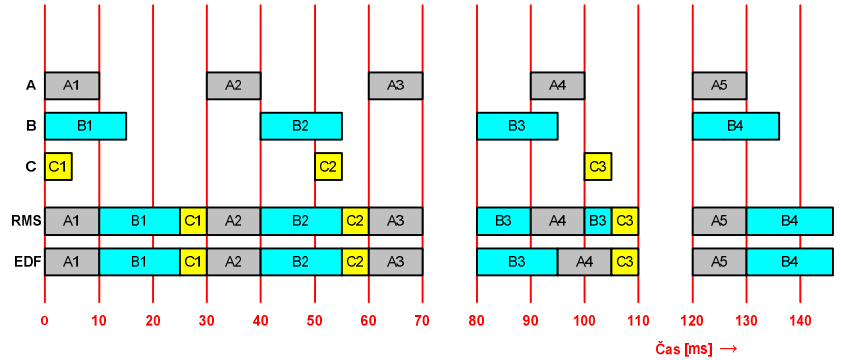
- RMS = *Rate Monotonic Scheduling*
- Statické priority pevně odvozené od periody procesů
  - $Prio_i \approx p_i$ (kratší perioda = menší číslo ~ vyšší priorita)
- Použitelné pro procesy s následujícími vlastnostmi
  - Periodický proces musí skončit během své periody
  - Procesy jsou vzájemně nezávislé
    - nesmí se vzájemně blokovat
  - Každý běh procesu (CPU dávka) spotřebuje konstantní čas
  - Preempce nastává okamžitě (zanedbatelná režie)
- Velmi podrobně prozkoumaný algoritmus
  - Je známo jeho chování i v případě, že nejsou splněny všechny zjednodušující předpoklady (⬅)

Plánování EDF

- EDF = *Earliest Deadline First*
  - Upřednostňuje proces s nejbližším termínem dokončení
- Dynamické priority
  - Plánovač vede seznam připravených procesů uspořádaný podle požadovaných časů dokončení a rozhodne vždy ve prospěch procesu s nejbližším požadovaným termínem dokončení
- Vhodné pro procesy s následujícími vlastnostmi
  - Procesy nemusí být přísně periodické ani nemusí mít konstantní dobu běhu
  - Pro každý proces musí být znám požadovaný termín dokončení (*deadline*)
- Vlastnosti algoritmu
  - Při plánování periodických procesů lze dodržet dokončovací termíny i při vytížení téměř 100%
  - Následky přetížení však nejsou známy a nejsou předvídatelné. Rovněž není známo chování v případech, kdy dokončovací termín a perioda jsou různé

Příklad 1 (pokračování)

Proces	Perioda p_i	Procesní čas T_i
A	30	10
B	40	15
C	50	5



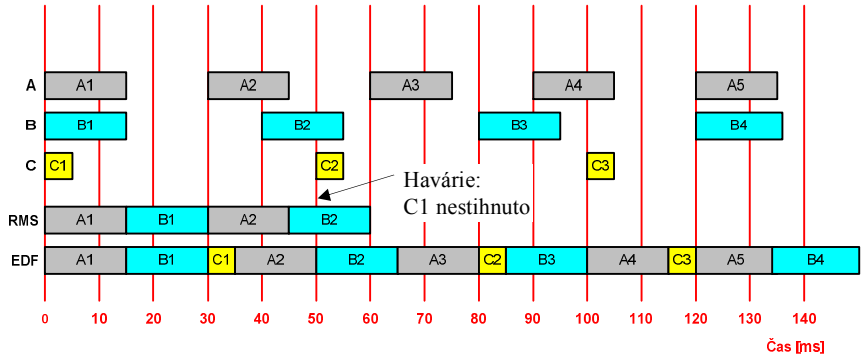
- Oba algoritmy fungují
  - Dokonce chvílemi zbývá volný čas k běhu nějakého procesu „na pozadí“

Příklad 2

- Opět 3 periodické procesy

Proces	Perioda p_i	Procesní čas T_i	T_i / p_i	Suma
A	30	15	0,500	r = 0,975 < 1
B	40	15	0,375	
C	50	5	0,100	

Plánovatelné



Plánování RMS podrobněji (1)

- Dobře analyticky zpracovaný algoritmus zaručující dodržení termínů dokončení, pokud při N procesech platí *postačující podmínka* [Liu & Layland 1973]:

$$r = \sum_{i=1}^{N} \frac{T_i}{p_i} \le N\left(\sqrt[N]{2} - 1\right);$$

$$\lim_{N \to \infty} N\left(\sqrt[N]{2} - 1\right) = \ln 2 \approx 0.693147$$

N	$N\left(\sqrt[N]{2} - 1\right)$
2	0,828427
3	0,779763
4	0,756828
5	0,743491
10	0,717734
20	0,705298

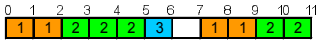
- Jsou vypracovány i způsoby spolupráce sdílených systémových prostředků
- Je známo i chování algoritmu při přechodném „přetížení“ systému
- Používáno v téměř všech komerčních RT OS

Plánování RMS podrobněji (2)

- Použitelnost algoritmu RMS
  - Bude použitelný, i když není splněna postačující podmínka?

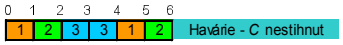
$3(\sqrt[3]{2}-1) = 0{,}7797$

P	i	p_i	T_i	T_i/p_i	$\Sigma(T_i/p_i)$
A	1	7	2	0,286	0,286
B	2	8	3	0,375	0,661
C	3	10	1	0,100	**0,761**

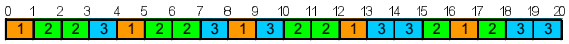


P	i	p_i	T_i	T_i/p_i	$\Sigma(T_i/p_i)$
A	1	6	2	0,333	0,333
B	2	8	3	0,375	0,708
C	3	10	1	0,100	**0,808**

P	i	p_i	T_i	T_i/p_i	$\Sigma(T_i/p_i)$
A	1	4	1	0,250	0,250
B	2	5	1	0,200	0,450
C	3	6	3	0,500	**0,950**



P	i	p_i	T_i	T_i/p_i	$\Sigma(T_i/p_i)$
A	1	4	1	0,250	0,250
B	2	5	2	0,400	0,650
C	3	20	7	0,350	**1,000**



  - Celkové vytížení tedy není zřejmě základním předpokladem pro použitelnost RMS

Plánování RMS podrobněji (3)

- Lehoczky, Sha & Ding [1989] podrobili RMS analýze znovu:
  - Mějme procesy $\{P_i, i = 1 \ldots N \mid p_i \le p_{i+1}, i = 1 \ldots N-1\}$
    - uspořádané dle priorit
  - Soustřeďme se na procesy 1...*i*, (*i*=1...*N*) a určeme vždy

$$W_i(t) = \sum_{j=1}^{i} T_j \lceil t / p_j \rceil, \quad L_i(t) = \frac{W_i(t)}{t}, \quad L_i = \min_{\{0<t\le T_i\}} L_i(t), \; L = \max_{\{1\le i\le N\}} L_i$$

$W_i(t)$ reprezentuje kumulativní potřeby procesů $P_1 \ldots P_i$ v časovém úseku $[0, t]$

  - Nutnou a postačující podmínkou pro spolehlivé použití algoritmu RMS je $L \le 1$. Pro určování $W_i(t)$ je nepříjemný „spojitý“ čas *t*.
  - Autoři ukázali, že stačí určovat $W_i(t)$ jen v časech *t* rovných násobku periody každého z procesů
    - Např. pro $\{p_1 = 4; p_2 = 5; p_3 = 13\}$ stačí počítat $W_i(t)$ a $L_i(t)$ pouze pro $t \in \{4, 5, 8, 10, 12, 13\}$
    - Z toho navíc plyne, že pokud RMS plán nezhavaruje během 1. periody procesu s nejnižší prioritou (nejdelší periodou), nezhavaruje ani v budoucnosti.

Plánování RMS podrobněji (4)

- Příklady použití uvedené teorie
  - RMS zhavaruje

i	p_i	T_i	T_i/p_i	$\Sigma(T_i/p_i)$
1	4	1	0,250	0,250
2	5	1	0,200	0,450
3	6	3	0,500	**0,950**

$L_i(4)$	$L_i(5)$	$L_i(6)$	L_i	L
0,250	0,400	0,333	0,250	1,167
0,500	0,600	0,667	0,500	
1,250	1,200	1,167	1,167	

  - RMS bude funkční

i	p_i	T_i	T_i/p_i	$\Sigma(T_i/p_i)$
1	4	1	0,250	0,250
2	5	2	0,400	0,650
3	20	7	0,350	**1,000**

$L_i(4)$	$L_i(5)$	$L_i(8)$	$L_i(10)$	$L_i(12)$	$L_i(15)$	$L_i(16)$	$L_i(20)$	L_i	L
0,250	0,400	0,250	0,300	0,250	0,267	0,250	0,250	0,250	1,000
0,750	0,800	0,750	0,700	0,750	0,667	0,750	0,650	0,650	
2,500	2,200	1,625	1,400	1,333	1,133	1,188	1,000	1,000	

Dotazy