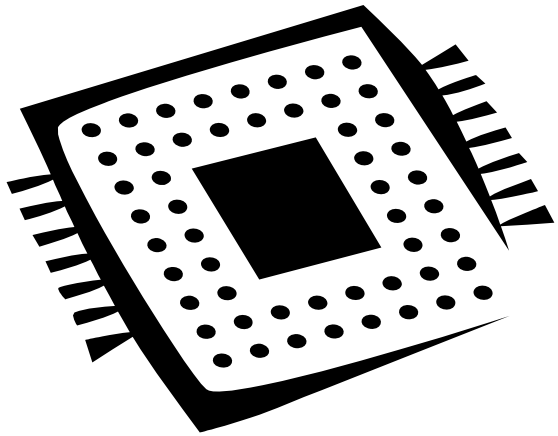
PB153
Operační systémy a jejich rozhraní

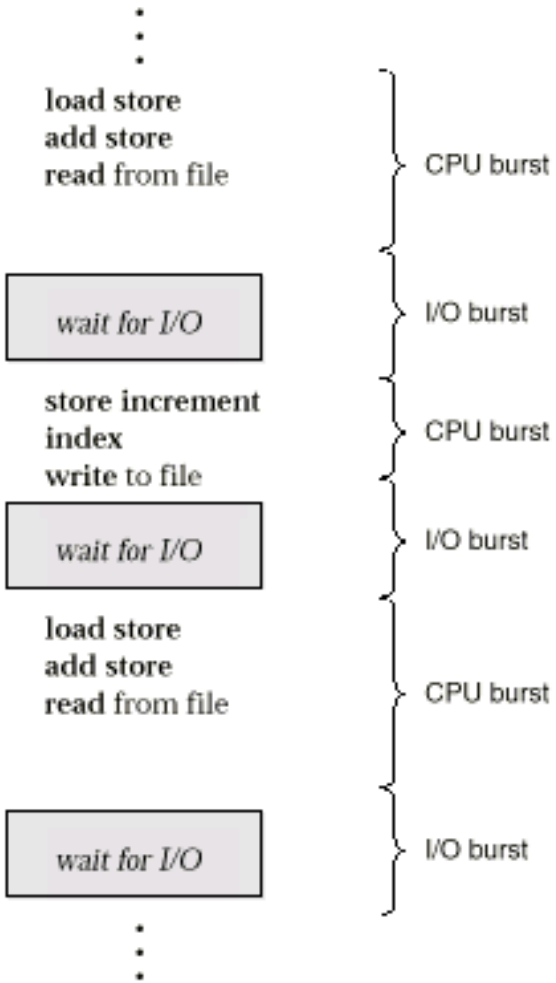
Plánování CPU



Multiprogramování

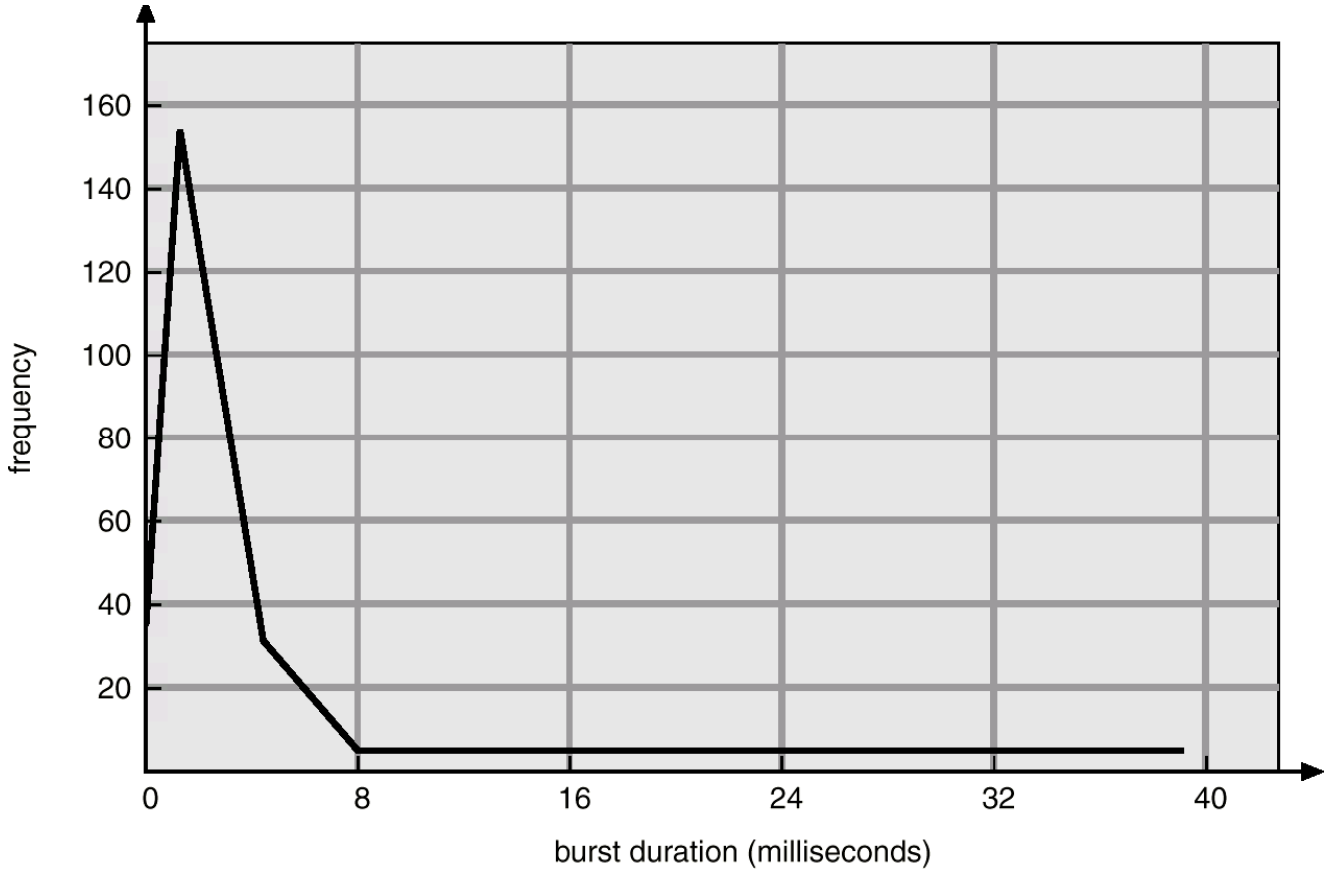
- Multiprogramování zvyšuje využití CPU
- Pokud jeden proces čeká na dokončení I/O operace může jiný proces CPU využít
- Nejlepšího výsledku dosáhneme při vhodné kombinaci procesů orientovaných na I/O a na využití CPU

Střídání využití CPU a I/O

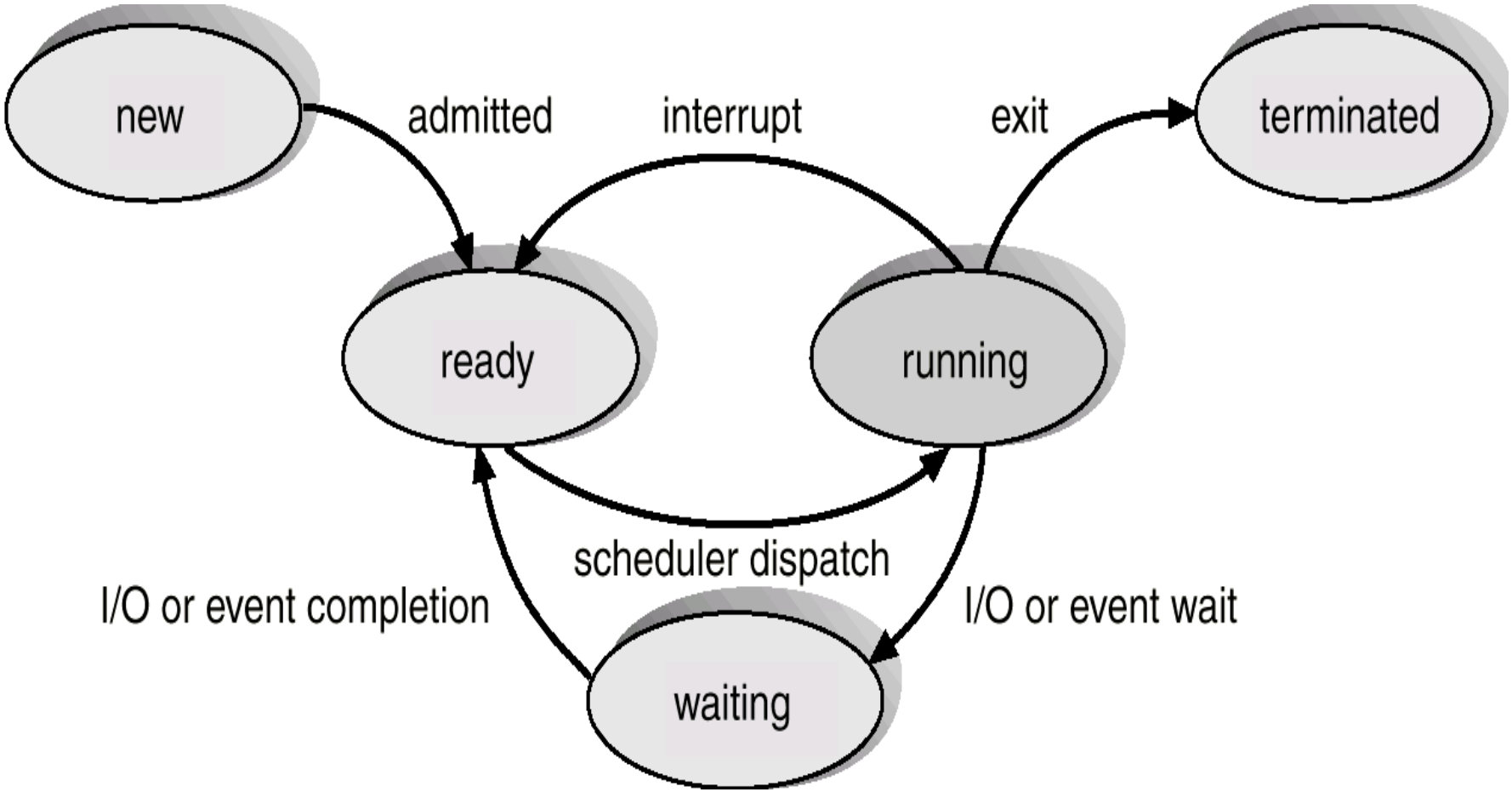


- Proces obvykle střídá části využívající CPU a části vyžadující I/O.
- Při zahájení I/O je proces zařazen mezi procesy čekající na událost.
- Teprve při ukončení I/O operace se proces opět dostává mezi procesy „připravené“.

Histrogram doby využití CPU



Stavy procesu



Plánování CPU

- Krátkodobý plánovač – dispečer
  - Vybírá proces, kterému bude přidělen CPU
  - Vybírá jeden z procesů, které jsou zavedeny operační paměti a které jsou „připravené“
  - Plánovací rozhodnutí může vydat v okamžiku, kdy proces:
    - 1. přechází ze stavu běžící do stavu čekající
    - 2. přechází ze stavu běžící do stavu připravený
    - 3. přechází ze stavu čekající do stavu připravený
    - 4. končí
  - Případy 1 a 4 se označují jako nepreemptivní plánování (plánování bez předbíhání)
  - Případy 2 a 3 se označují jako preemptivní plánování (plánování s předbíháním)

Dispečer

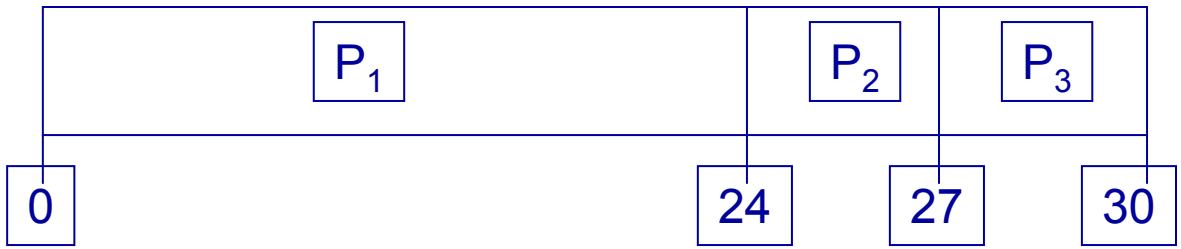
- Výstupní modul krátkodobého plánovače nebo plánovač sám, který předává procesor procesu vybranému krátkodobým plánovačem
- Předání zahrnuje:
  - přepnutí kontextu
  - přepnutí režimu procesoru na uživatelský režim
  - skok na odpovídající místo v uživatelském programu pro opětovné pokračování v běhu procesu
- Dispečerské zpoždění (Dispatch latency)
  - Doba, kterou potřebuje dispečer pro pozastavení běhu jednoho procesu a start běhu jiného procesu

Kritéria plánování [a optimalizace]

- Využití CPU [maximalizace]
  - cílem je udržení CPU v kontinuální užitečné činnosti
- Propustnost [maximalizace]
  - počet procesů, které dokončí svůj běh za jednotku času
- Doba obrátky [minimalizace]
  - doba potřebná pro provedení konkrétního procesu
- Doba čekání [minimalizace]
  - doba, po kterou proces čekal ve frontě „připravených“ procesů
- Doba odpovědi [minimalizace]
  - doba, která uplyne od okamžiku zadání požadavku do doby první reakce (první odpovědi, nikoli poskytnutí plného výstupu)

Algoritmus FCFS

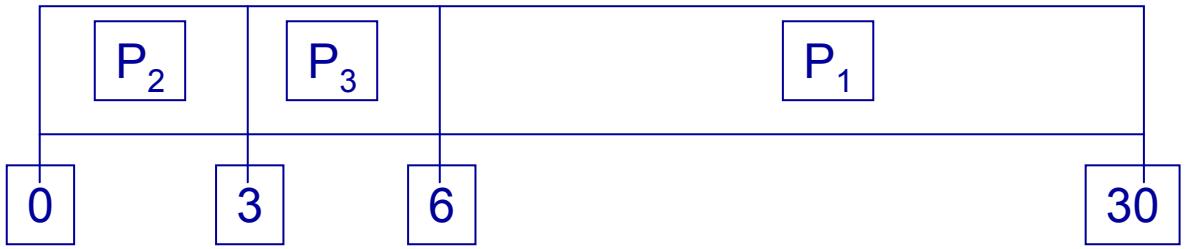
- Algoritmus „Kdo dřív přijde, ten dřív mele“ (First Come, First Served), FCFS
- Máme 3 procesy P1 (vyžaduje 24 dávek CPU), P2 (vyžaduje 3 dávky CPU), P3 (vyžaduje 3 dávky CPU)
- Procesy vznikly v pořadí: P1, P2, P3
- Ganttovo schématické vyjádření plánu:



- Doby čekání: P1 = 0, P2 = 24, P3 = 27
- Průměrná doba čekání: (0+24+27)/3 = 17

Algiritmus FCFS (2)

- Varianta jiná: procesy vznikly v pořadí P2, P3, P1
- Ganttovo schématické vyjádření plánu:



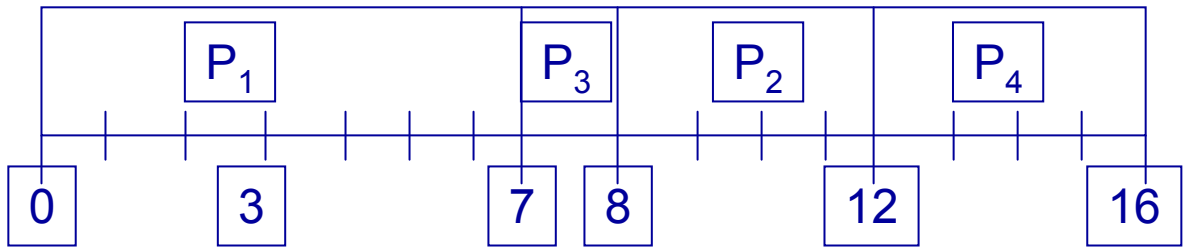
- Doby čekání: P2 = 0, P3 = 3, P1 = 6
- Průměrná doba čekání: (6+0+3)/3 = 3
- To je mnohem lepší výsledek než v předchozím případě, i když se jedná o stejné procesy a stejný plánovací algoritmus
- Krátké procesy následující po dlouhém procesu ovlivňuje „konvojový efekt“

Algoritmus SJF

- Algoritmus Shortest-Job-First
- Musíme znát délku příštího požadavku na dávku CPU pro každý proces
- Vybírá se proces s nejkratším požadavkem na CPU
- Dvě varianty:
  - nepreemptivní, bez předbíhání
    - jakmile se CPU předá vybranému procesu, tento nemůže být předběhnut žádným jiným procesem, dokud přidělenou dávku CPU nedokončí
  - preemptivní, s předbíháním
    - jakmile se ve frontě připravených procesů objeví proces s délkou dávky CPU kratší než je doba zbývající k dokončení dávky právě běžícího procesu, je právě běžící proces ve využívání CPU předběhnut novým procesem
    - tato varianta se rovněž nazývá Shortest-Remaining-Time-First (SRTF)
- SJF je optimální algoritmus (pro danou množinu procesů dává minimální **průměrnou** dobu čekání)

Příklad *nepreemptivního* algoritmu SJF

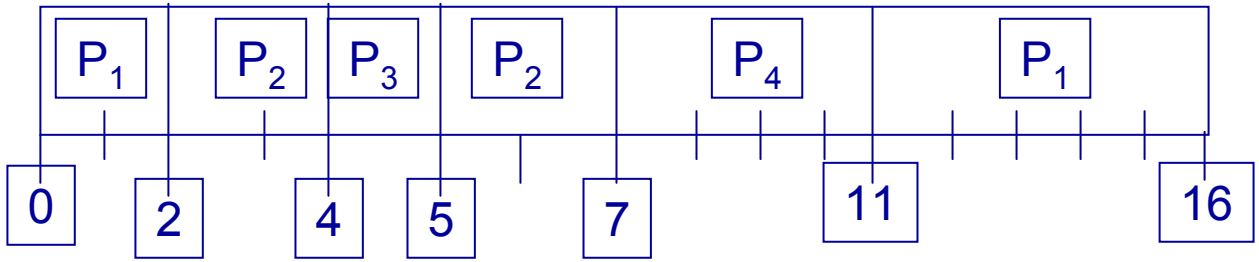
- Proces Doba příchodu Délka dávky CPU
  - P1 0.0 7
  - P2 2.0 4
  - P3 4.0 1
  - P4 5.0 4
- Ganttovo schématické vyjádření plánu:



- Průměrná doba čekání: (0+6+3+7)/4 = 4

Příklad *preemptivního* algoritmu SJF

- Proces Doba příchodu Délka dávky CPU
  - P1 0.0 7
  - P2 2.0 4
  - P3 4.0 1
  - P4 5.0 4
- Ganttovo schématické vyjádření plánu:



- Průměrná délka čekání: (9+1+0+2)/4 = 3

Určení délky příští dávky CPU procesu

- Délku příští dávky CPU procesu neznáme, můžeme ji pouze odhadovat
- To můžeme udělat na základě historie
  - Musíme znát délky předchozích dávek CPU
  - Použijeme exponenciální průměrování:

1. t_n = skutecna delka n - te davky
2. τ_{n+1} = predpokladana hodnota pro dalsi CPU davku
3. $\alpha, 0 \le \alpha \le 1$, koeficient historie
4. pak definujeme odhad jako :

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n .$$

Příklad

- $\alpha = 0$ (historii nebereme v úvahu)
  - $\tau_{n+1} = \tau_n$
- $\alpha = 1$ (budoucí odhad = skutečná minulá hodnota)
  - $\tau_{n+1} = t_n$
- Když formuli rozvineme, dostaneme
  $\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \ldots + (1-\alpha)^i \alpha t_{n-i} + \ldots + (1-\alpha)^n t_0$
- α a $(1-\alpha)$ jsou <= 1,
- Každý další člen výrazu má na výslednou hodnotu menší vliv než jeho předchůdce

Prioritní plánování

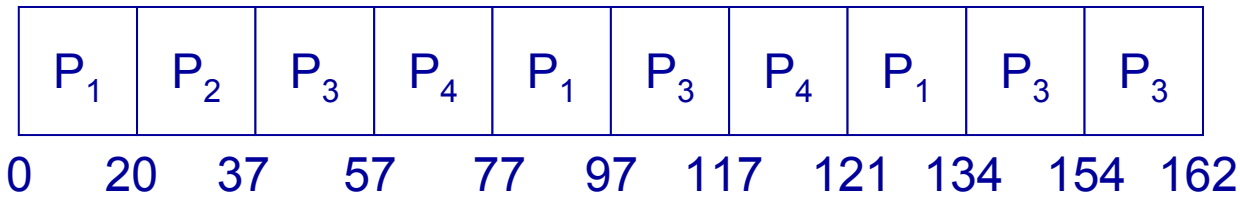
- S každým procesem je spojeno prioritní číslo
  - prioritní číslo – preference procesu pro výběr příště běžícího procesu
  - CPU se přiděluje procesu s největší prioritou
  - nejvyšší prioritě obvykle odpovídá nejnižší prioritní číslo ☺
- Opět dvě varianty
  - nepreemptivní, bez předbíhání
    - jakmile proces získá přístup k CPU nemůže být předběhnut jiným procesem dokud dávku neukončí
  - preemptivní, s předbíháním
    - jakmile se ve frontě připravených procesů objeví proces s vyšší prioritou než je priorita běžícího procesu, je běžící proces předběhnut
- SJF je prioritní plánování, prioritou je předpokládaná délka příští CPU dávky
- stárnutí
  - procesy s nižší prioritou se nemusí nikdy provést
  - řešení: zrání – priorita se s postupem času zvyšuje

Round Robin (RR)

- Každý proces dostává CPU na malou jednotku času – časové kvantum
  - Desítky až stovky ms
- Po uplynutí této doby je běžící proces předběhnut nejstarším procesem ve frontě připravených procesů a zařazuje se na konec této fronty
- Je-li ve frontě připravených procesů n procesů a časové kvantum je *q*, pak každý proces získává *1/n* doby CPU, najednou nejvýše *q* časových jednotek
- Žádný proces nečeká na přidělení CPU déle než *(n-1)q* časových jednotek
- Výkonnostní hodnocení
  - q velké → ekvivalent FIFO
  - q malé → velká režie; v praxi musí být q musí být dostatečně velké s ohledem na režii přepínání kontextu

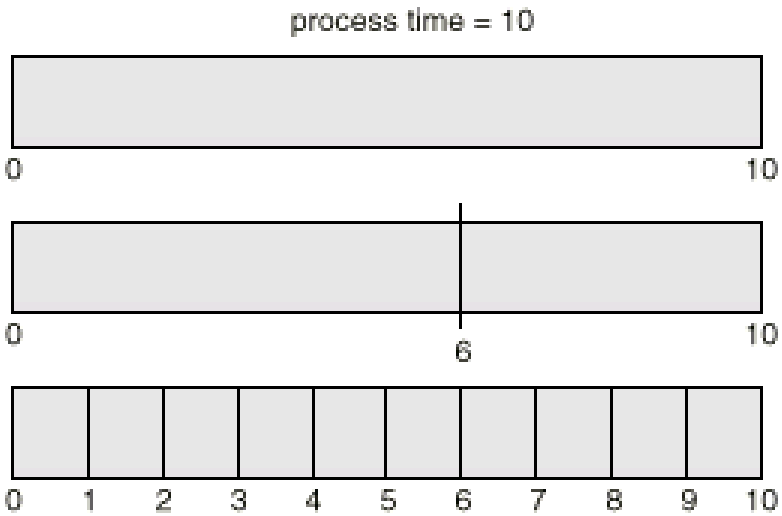
Příklad RR s časovým kvantem = 20

- Proces Délka dávky CPU
  - P1 53
  - P2 17
  - P3 68
  - P4 24
- Ganttovo schématické vyjádření plánu:



- Typicky se dosahuje delší *průměrné* doby obrátky než při plánování SJF, avšak doba odpovědi je výrazně nižší

Časové kvantum a doba přepnutí kontextu

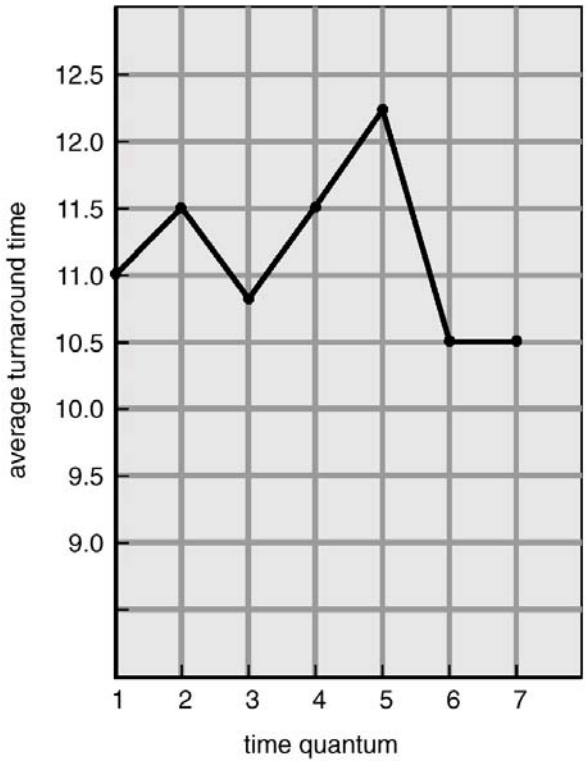


quantum	context switches
12	0
6	1
1	9

- Příklad: doba přepnutí kontextu = 0,01
- Ztráty související s režií OS při q = 12, 6 a 1 jsou 0,08; 0,16 a 1 %

Doba obrátky

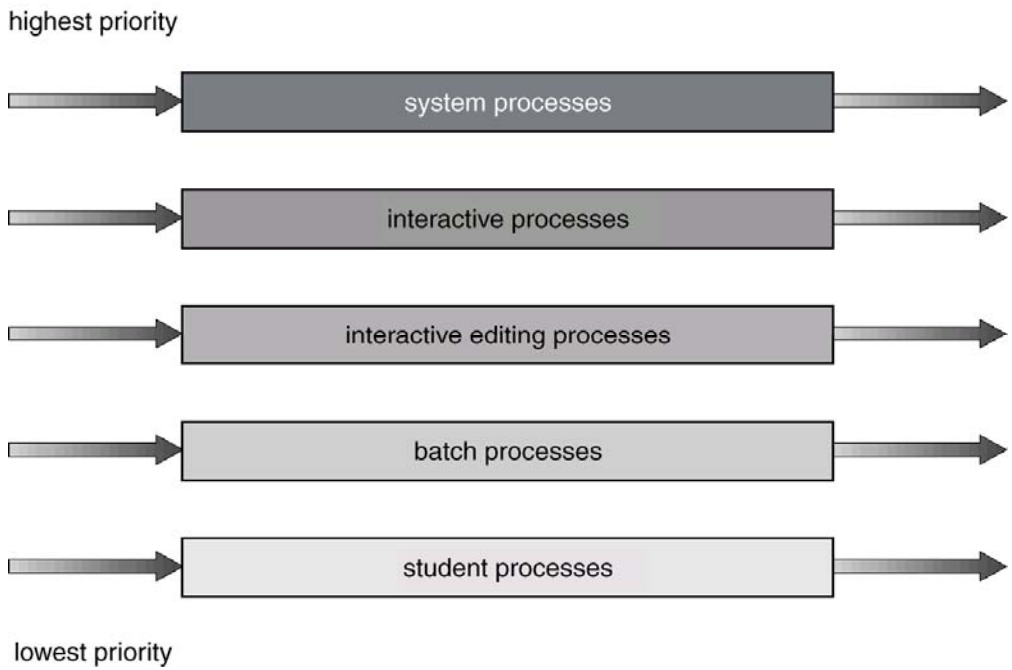
- Doba obrátky se mění se změnou délky časového kvanta



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Fronta procesů

- Fronta „připravených“ procesů nemusí být jediná
  - procesy můžeme dělit na interaktivní, dávkové, apod.
  - pro každou frontu můžeme použít jiný plánovací algoritmus



Příklad: Linux

- Plánovací algoritmus je součástí jádra OS
- Skládá se ze 2 funkcí
  - schedule() – plánování procesů
  - do_timer() – aktualizuje informace o procesech (spotřebovaný čas v uživatelském režimu, v režimu jádra, priority apod.)
- Časové kvantum je 1/100 sekundy
- Plánovací algoritmus byl předmětem vývoje

Příklad: Linux (2)

- Plánovací algoritmus
  - 3 kategorie procesů z hlediska plánování
    - SCHED_FIFO
    - SCHED_RR
    - SCHED_OTHER
  - první dva typy jsou procesy se zvláštními nároky na plánování (soft real time), mají přednost před ostatními procesy (časové kvantum 200ms bez preempce) a může je vytvářet pouze root
  - procesy SCHED_FIFO jsou plánovány metodou FIFO
  - procesy SCHED_RR jsou plánovány metodou RR
  - procesy SCHED_FIFO a SCHED_RR mají přirazenu prioritu (0-99), při plánování jsou vybírány procesy s vyšší prioritou, plánovací algoritmu je preemptivní (kvantum 10ms)

Příklad: Linux (3)

- Plánovací algoritmus – pokračování
  - do SCHED_OTHER patří všechny klasické timesharingové procesy
  - v rámci SCHED_OTHER jsou procesy plánovány na základě dynamických priorit
    - tzv. hodnota nice
    - zvyšována při stárnutí procesu
    - neadministrátorský proces může jen zhoršit prioritu
    - procesy se stejnou prioritou jsou plánovaný pomocí RR

Příklad: Linux (4)

- Plánovací algoritmus
  - Do jádra 2.4 algoritmus procházející globální (pro všechny procesory) frontu a hledající vhodný proces (složitost O(n) kde n počet čekajících procesů)
  - Od jádra 2.6 (resp 2.5) nový, tzv. O(1) algoritmus, fronty procesů per-CPU
  - Od 2.6.23 nahrazen algoritmem CFS (completely fair scheduler), který pro seznamy procesů používá červeno-černý strom. Výběr procesu pro běh na procesoru v konstantním čase, jeho znovuvložení O(log n).

Příklad: Linux (5)

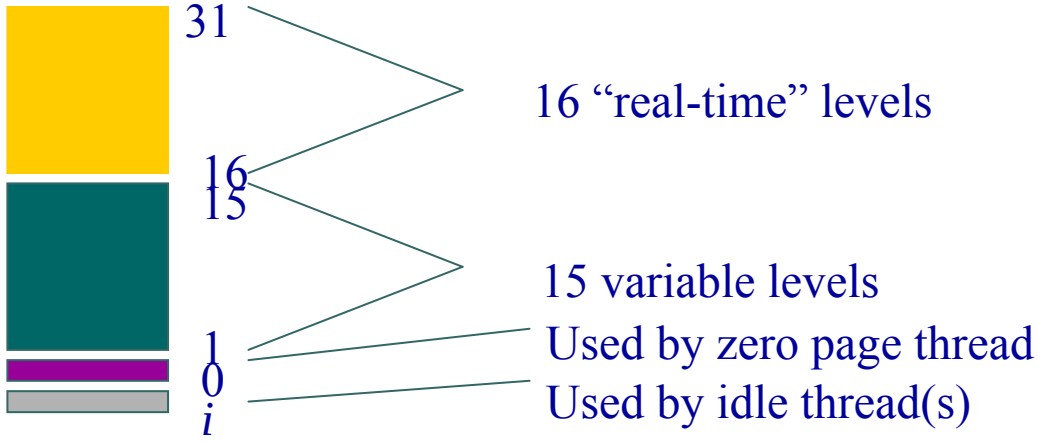
System Calls Related to Scheduling	
System Call	**Description**
nice( )	Change the priority of a conventional process.
getpriority( )	Get the maximum priority of a group of conventional processes.
setpriority( )	Set the priority of a group of conventional processes.
sched_getscheduler( )	Get the scheduling policy of a process.
sched_setscheduler( )	Set the scheduling policy and priority of a process.
sched_getparam( )	Get the scheduling priority of a process.
sched_setparam( )	Set the priority of a process.
sched_yield( )	Relinquish the processor voluntarily without blocking.
sched_get_ priority_min( )	Get the minimum priority value for a policy.
sched_get_ priority_max( )	Get the maximum priority value for a policy.
sched_rr_get_interval( )	Get the time quantum value for the Round Robin policy.

Příklad: Win32 (1)

- Z pohledu Win32:
  - procesy při vytvoření přiděleny do jedné z následujících tříd
    - Idle
    - Below Normal
    - Normal
    - Above Normal
    - High
    - Realtime
  - Vlákna dále mají relativní prioritu v rámci třídy, do které patří
    - Idle
    - Lowest
    - Below_Normal
    - Normal
    - Above_Normal
    - Highest
    - Time_Critical

Příklad: Win32/W2k (2)

- Plánovací algoritmu ve Windows 200
  - plánuje vlákna, ne procesy
  - vlákna mají priority 0 až 31



Příklad: Win32 (3)

- Přehled priorit ve Win32

	real-time	high	above normal	normal	below normal	idle priority
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

Příklad: Win32 (4)

- *Get/SetPriorityClass*
- *Get/SetThreadPriority* – relativní vůči základní prioritě procesu
- *Get/SetProcessAffinityMask*
- *SetThreadAffinityMask* – musí být podmožinou masky procesu
- *SetThreadIdealProcessor* – preferovaný procesor
- *Get/SetProcessPriorityBoost*
- *Suspend/ResumeThread*

Příklad: Win32 (5)

- Plánovací algoritmus je řízen především prioritami
  - 32 front (FIFO seznamů) vláken, která jsou „připravena“
    - pro každou úroveň priority jedna fronta
    - fronty jsou společné pro všechny procesory
  - když je vlákno „připraveno“
    - buď běží okamžitě
    - nebo je umístěno na konec fronty „připravených“ procesů ve své prioritě
  - na jednoprocesorovém stroji vždy běží vlákno s nejvyšší prioritou
- V rámci jedné prioritní skupiny se plánuje algoritmem round-robin pomocí časových kvant