# OS – Plánování procesů

Tomáš Hudec

Tomas.Hudec@upce.cz

http://fei-as.upceucebny.cz/usr/hudec/vyuka/os/

# Plánování – scheduling

- scheduler – plánovač
  - rozhoduje, který proces (vlákno) má CPU
  - řídí se plánovacím algoritmem
- historie:
  - dávkové systémy – jediná fronta, jediná úloha
    - potřeba multiprogramování, neboť 80 % času bylo CPU nevyužito – zdržování na V/V
  - více paměti – multiprogramming, timesharing
    - potřeba naplánovat běh procesů optimálně

# Typy plánování (dle času)

- dlouhodobé (long term)
  - které nové úlohy se mají zpracovávat, které ukončit
  - rozhoduje o množství procesů v systému
- střednědobé (medium term)
  - které procesy se mají odložit nebo vrátit do paměti
  - efektivní práce s omezenou operační pamětí
- krátkodobé (short term, dispatching)
  - který připravený proces dostane CPU, na jak dlouho

# Cíle plánování obecně a podle typu OS

- spravedlnost – každý proces stejně času
- dodržování strategie (priorit)
- efektivní využití zdrojů, rovnováha zatížení
  - snaha využívat všechny části systému současně
- interaktivní systémy
  - minimalizace odezvy (response time)
    - čas mezi zadáním příkazu a odezvou
  - proporcionalita
    - vyhovět očekáváním uživatelů

# Cíle plánování podle typu OS

- dávkové systémy
  - maximalizovat propustnost (throughput)
    - počet vykonaných úloh za jednotku času (h)
  - minimalizovat obrat (turnaround time)
    - průměrný čas na vykonání úlohy
  - využití CPU – využívat maximálně CPU
- systémy real-time
  - respektování lhůt – zabránění ztráty dat
  - předvídatelnost – zabránění degradace kvality
    - např. multimediální systémy

# Režimy plánování

- nepreemptivní
  - proces se musí sám vzdát CPU (nebo blokovat)
- preemptivní
  - plánovač rozhoduje, kdy který proces má CPU
  - (efektivně) plánovat lze pouze v případě, že je k dispozici přerušovací systém a časovač
  - časovač „tiká“ typicky na frekvenci 100 Hz
    - přerušení nastává tedy každých 10 ms
    - plánování spotřebovává také čas CPU – režie

# Typy procesů

- vstupně-výstupně orientovaný proces
  - většinu času čeká na dokončení operací V/V
  - typická je krátká výpočetní doba
  - časté používání blokujících systémových volání
  - typické pro interaktivní procesy
- výpočetně orientovaný proces
  - používá intenzivně procesor
  - blokující volání téměř nepoužívá

# Plánovací algoritmy

- historie, dávkové systémy
  - fronta jednotlivých úloh (FIFO), víceúlohová FIFO
  - podle odhadu doby běhu úlohy
- moderní plánovací algoritmy
  - round-robin – spravedlivé střídání úloh
  - prioritní – dle důležitosti úlohy
  - uživatelsky férové – spravedlivé mezi uživateli
  - termínové – dodržení lhůt na systémech real-time

# První přijde, první mele

- First-Come First-Served (fronta FIFO)
  - nepreemptivní
  - nové úlohy se zařadí do fronty
  - po ukončení aktuálního procesu se přidělí CPU procesu, který čekal ve frontě nejdéle
  - krátké procesy musejí zbytečně dlouho čekat
  - zvýhodňuje výpočtově orientované procesy
    - procesy bez V/V čekají pouze jednou
    - procesy s V/V čekají při každém dokončení operace

# Nejkratší úloha první

- Shortest Job First
  - nepreemptivní
  - spustí se proces s nejkratší očekávanou dobou provádění
  - krátké procesy mají přednost
  - závislé na dobrém odhadu délky běhu procesu
  - hrozí vyhladovění dlouhodobých procesů

# Nejkratší zbývající následuje

- Shortest Remaining Time Next
  - preemptivní varianta SJF
  - spustí se proces s nejkratší očekávanou dobou do dokončení
  - dále minimalizuje obrat (turnaround time)

# Cyklická obsluha

- Round-Robin
  - preempce založená na časovači
  - každý proces dostane časové kvantum na CPU
  - přepnutí je prováděno při vypršení kvanta nebo při volání blokujícího systémového volání
  - je třeba optimalizovat délku kvanta
    - Příklad:
    - kvantum 4 ms, context switch 1 ms
    - CPU pracuje produktivně jen 80 % času
  - typické nastavení frekvence časovače je 100 Hz

# Prioritní plánování

- priority based scheduling
- dává se přednost procesu s vyšší prioritou
- obvykle více front pro připravené procesy
- nízká priorita může mít za následek „vyhladovění“ procesu (starvation)
  - proces se již nedostane k CPU, „smrt hladem“
  - pro zabránení je třeba např. prioritu přizpůsobovat v závislosti na době čekání a historii běhu procesu

# Plánování se zárukou

- Guaranteed Scheduling – Fair-Share
  - zaručuje každému uživateli stejné podmínky
    - n uživatelů na systému
    - každý dostane časové kvantum 1/n
  - příklad:
    - uživatel A spustí 9 procesů
    - uživatel B spustí 1 proces
    - při RR má uživatel A 90 % času CPU, B pouze 10 %
    - při FS se využití CPU rozdělí mezi A a B na 50 %
      - B: jeden proces 50 % času CPU
      - A: devět procesů si rozdělí 50 % času, jeden má cca 5,56 %

# Loteriové plánování

- Lottery Scheduling
  - každý proces dostane tiket(y) a periodicky se losuje
  - „výherní“ proces získá čas CPU
  - důležité procesy mohou mít více tiketů
    - procesy jsou si rovny, ale některé jsou si „rovnější“ (parafráze Orwella)
  - kooperativní procesy si mohou předávat tikety
  - lze použít jako aproximace jiných algoritmů
    - snadná implementace

# Plánování na systémech reálného času (real-time)

- pro úlohy reálného času je důležité dokončení ve stanoveném termínu (nikoliv rychlost)
- jen periodické události:
  - systém je plánovatelný, je-li suma časů potřebných na obsloužení událostí dělená jejich periodami menší nebo rovna jedné

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

  - lze plánovat staticky (table-driven)
    - tabulky stanoví, kdy která úloha má být spuštěna
- aperiodické události: dynamické plánování

# Plánování na systémech reálného času a preempce

- preemptivní
  - běžící proces může být přerušen
    - procesem s vyšší prioritou
    - procesem s bližším termínem dokončení
- nepreemptivní
  - běžící proces nesmí být přerušen
    - proces se musí vzdát procesoru sám
  - př.: proces musí v reálném čase bez přerušení komunikovat s externím zařízením

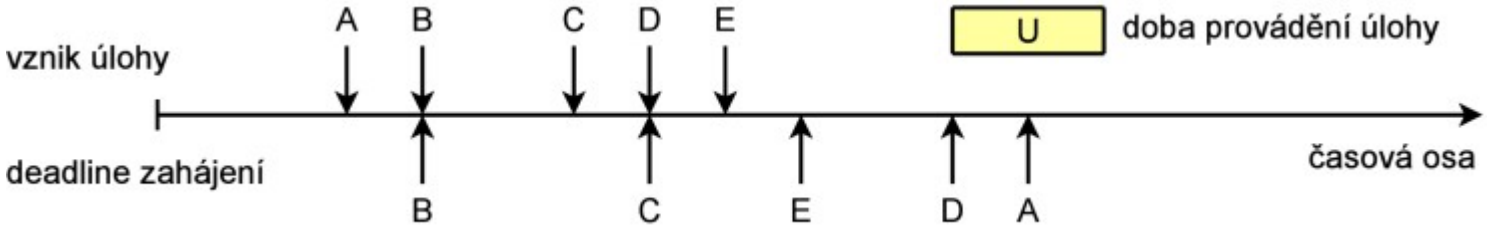
# Termínové plánování – EDF (Earliest Deadline First)

- dokončení všech úloh ve stanoveném termínu
- ke spuštění vybírá úlohu s nejbližším termínem (deadline) zahájení / ukončení
- minimalizuje se podíl úloh, které nejsou dokončeny v požadovaném termínu
- na plánovatelných jednoprocesorových systémech s preempcí je optimální

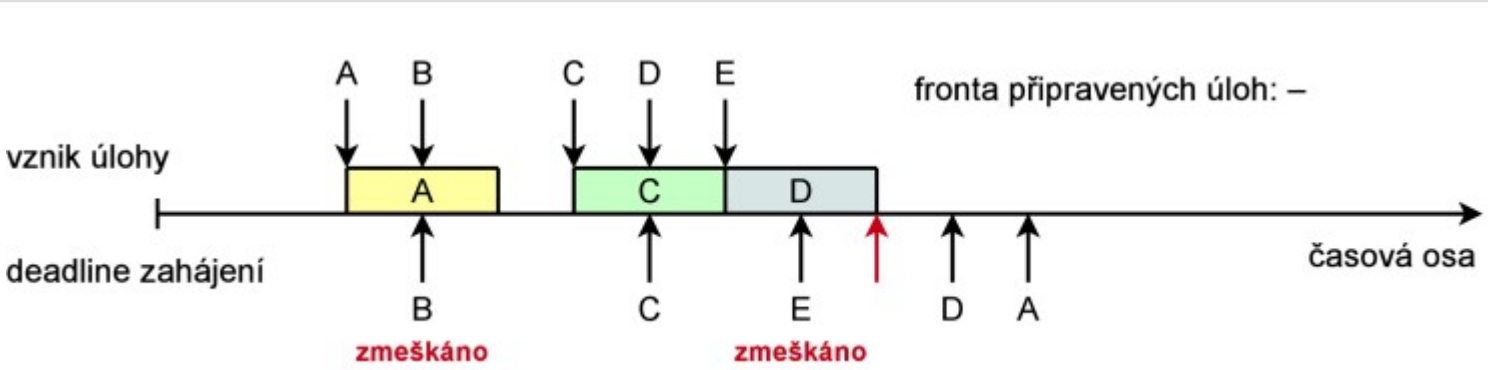
# Nevýhody termínového plánování

- při přetížení systému není předvídatelné
  - skupina ovlivněných procesů je závislá na čase, kdy nastalo přetížení
- implementace v HW je náročná (přesnost)
  - termíny je třeba reprezentovat konečnými čísly
- je třeba znát termíny a dobu zpracování úloh
- kritické sekce: např. úlohy A, B, C (dle termínů)
  - C je v KS, A požaduje přístup do KS (blokuje)
  - plánuje se B, C neuvolní KS a zmešká se termín A

# Plánování RT úloh – FIFO bez preempce (obrázek)

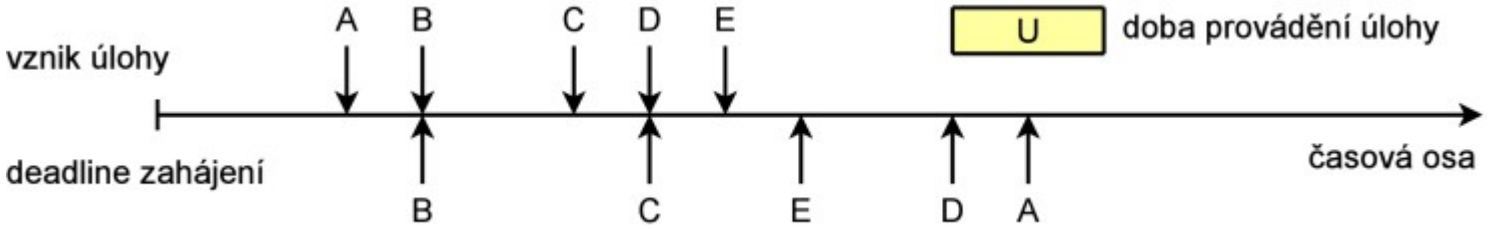



výchozí předpoklady

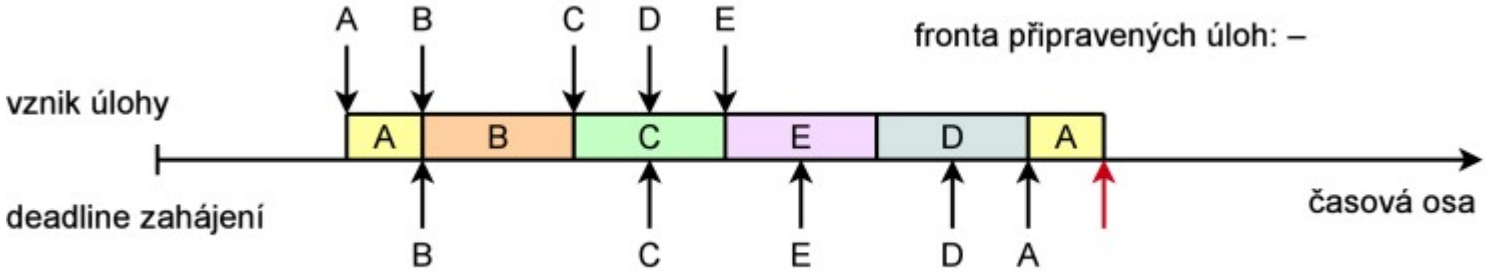



First-Come First-Served (nepreemptivní)

# Plánování RT úloh – EDF s preempcí (obrázek)

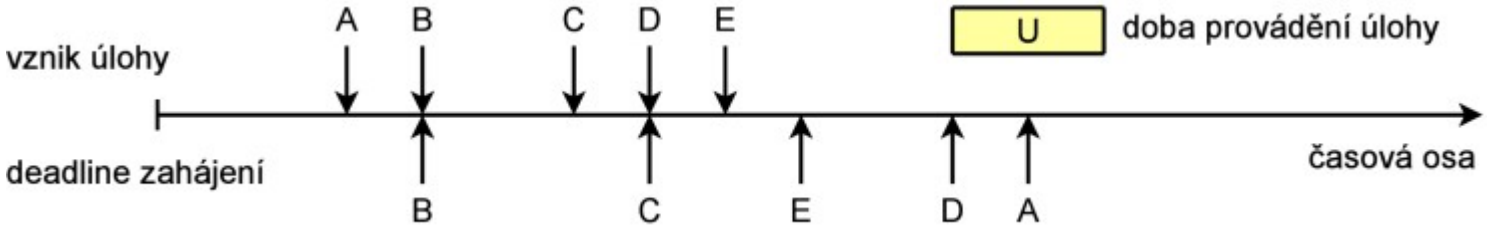



výchozí předpoklady

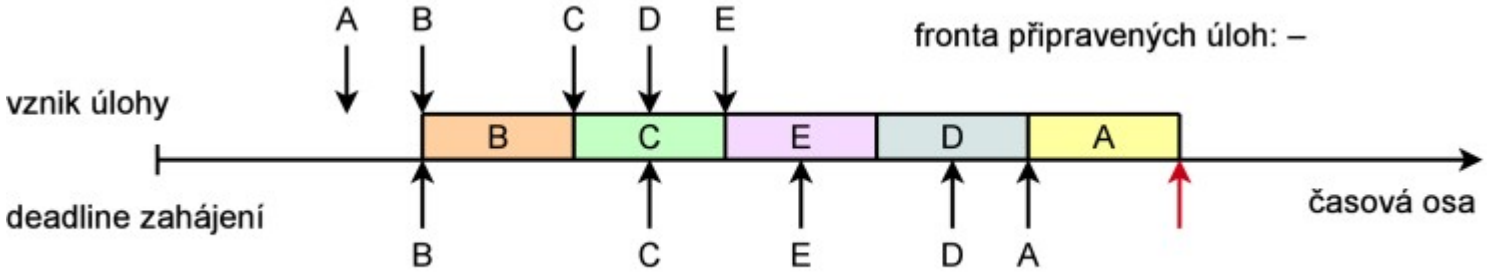



Earliest Deadline First (preemptivní)

# Plánování RT úloh – EDF bez preempce (obrázek)




výchozí předpoklady

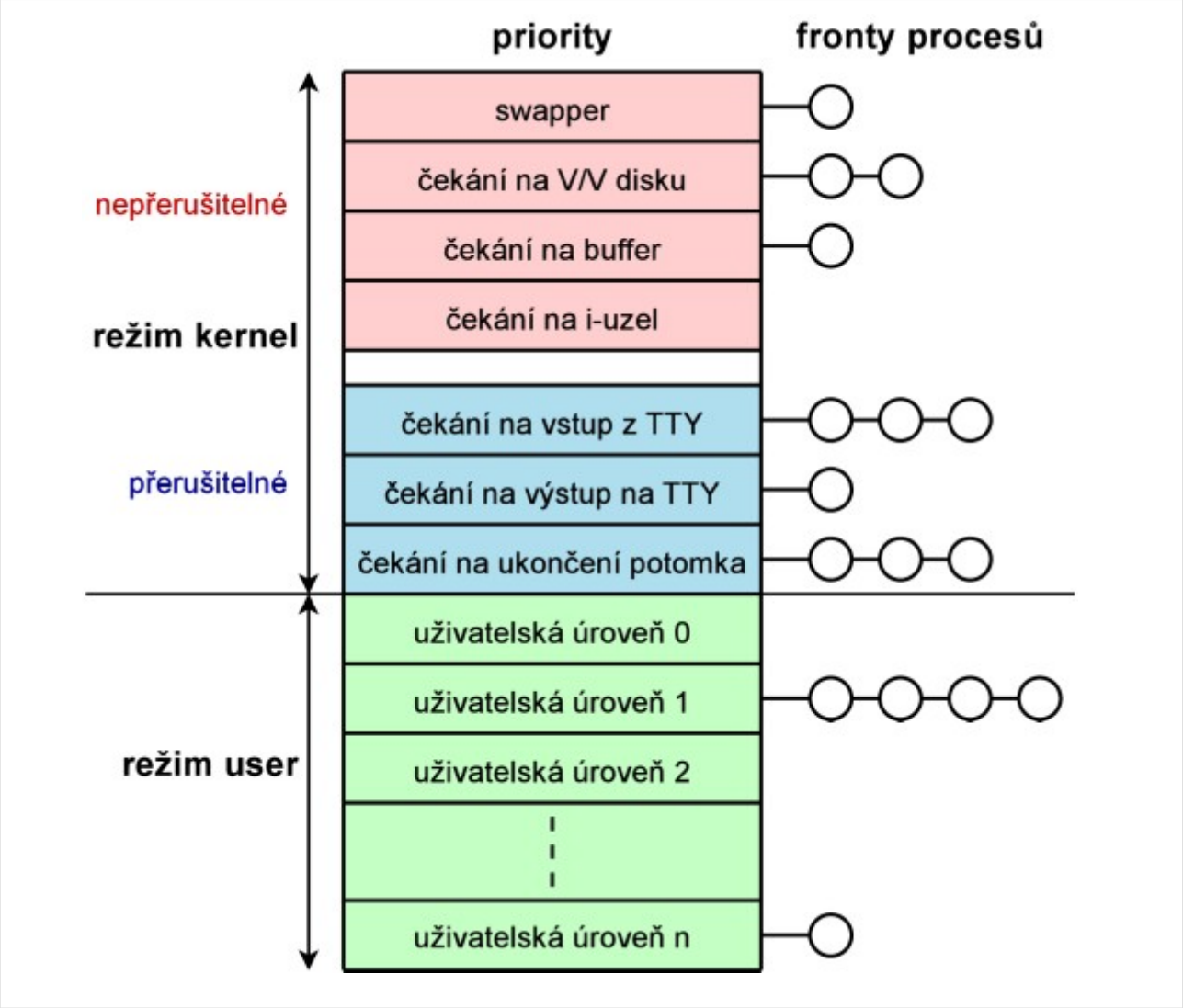



Earliest Deadline First (nepreemptivní)

# Plánování v Unixu

- založeno na prioritním plánování
- priorita pro proces je dvojí
  - pro běh v režimu jádra
    - přiřazuje se procesu, když přechází do spícího stavu
    - pevná priorita podle typu systémového volání
    - přerušitelná a nepřerušitelná
  - pro běh v uživatelském režimu
    - nastaví se po návratu z režimu jádra
    - dynamická priorita

# Plánování v Unixu (obrázek)



# Plánování v Linuxu – přehled

- verze 1.2 (1995) – round-robin
- 2.2 (1999) – class based scheduling
  - priority, podpora SMP, jediný seznam úloh
- 2.4 (2001) – plánovač $O(n)$
  - čas CPU rozdělen do epoch, jediný seznam úloh
- 2.6 (2003) – plánovač $O(1)$
  - samostaná fronta pro každé CPU, dvě pole úloh
- 2.6.23 (2007) – CFS (Completely Fair Scheduler)

# Dynamická priorita

- jádrem určený bonus dle historie běhu procesu
  - čekajícím se priorita zvyšuje, běžícím snižuje
- uživatelsky ovlivnitelná část – hodnota nice(2)
  - od −20 (maximální priorita) do 19 (minimální priorita)
    - výchozí hodnota je nula
  - hodnota říká, jak „milý“ (nice) je uživatel na ostatní
  - změna o ±1 znamená zhruba ±5 % času CPU
  - správce (root) smí hodnotu nastavovat libovolně
  - uživatel smí prioritu pouze snižovat (příp. obnovit)

# Linux 2.4 – plánovač *O*(*n*)

- procesorový čas je rozdělen do epoch
  - každý proces má vypočítané časové kvantum
    - v rámci epochy je může využívat po částech
  - když všechny běhuschopné procesy vyčerpaly svá kvanta, epocha končí a začíná epocha nová
    - přepočítají se časová kvanta **VŠECH** procesů – *O*(*n*)
  - časové kvantum je dynamické
    - základní frekvence je 100 · HZ / 1000,
      HZ = 100 Hz, tj.10 tiků ≈ 100ms kvantum
  - jediný seznam procesů pro všechny procesory

# Linux 2.6 – plánovač *O*(1)

- založeno na prioritách – 140 úrovní
  - priorita real-time (0–99) má vždy přednost
- rozlišuje interaktivní procesy
  - podle průměrné doby čekání na CPU
- fronta procesů (*runqueue*) pro každé CPU
  - má dvě struktury: *active* a *expired*, každá obsahuje
    - seznam procesů pro každou prioritu (140 front)
    - bitová mapa – neprázdnost seznamu pro každou prioritu
    - počet procesů

# Linux 2.6 – plánovač *O*(1) – preempce

- preempce (činnost plánovače):
  - právě přerušený proces:
    - (dynamická) priorita a časové kvantum jsou přepočítány
    - přesunut na konec příslušného seznamu (dle priority)
    - interaktivní a real-time zůstává v *active*, jinak do *expired*
  - aktivace procesu s nejvyšší prioritou v *active*
    - nejvyšší nastavený bit v bitmapě určí seznam – *O*(1) využití instrukce typu *find-first-bit-set*
  - prázdné *active* → prohození s *expired*
    - neexistuje-li běhuschopný proces, HLT (čekání na IRQ)

# Priorita a časové kvantum *O*(1)

- priorita p v jádře: 0–139 (0 je maximální priorita)
  - procesy real-time: rt_priority > 0 (99 je max. priorita)

    $p = 99 - \text{rt\_priority}$ statická hodnota (0–98)
  - ostatní procesy: rt_priority = 0, dynamická 100–139

    $p = \max(100, \min(139, 120 + nice - b + 5))$ $b \in \langle 0, 10 \rangle$
    - bonus $b$ = (průměrná doba čekání na CPU v ms) / 100
    - proces je interaktivní, pokud $b - 5 \geq (120 + nice) / 4 - 28$
- základní časové kvantum pro p ≥ 100

  $t = (140 - p) \cdot 20$ pro $p < 120$ 420 – 800 ms

  $t = (140 - p) \cdot 5$ pro $p \geq 120$ 5–100 ms

# Priorita a plánovací třídy

- třídy plánovače – sched_setscheduler(2), chrt(1)
  - procesy real-time (statická priorita 0–99)
    - absolutní přednost před procesy s dynamickou prioritou
    - třída SCHED_FIFO (nepreemptivní)
    - SCHED_RR (round-robin)
  - ostatní (dynamická priorita, základní je 120 + *nice*)
    - čekání na CPU (blokování) zvyšuje prioritu
    - třída SCHED_OTHER (SCHED_NORMAL), později navíc
      - SCHED_BATCH (od v. 2.6.16, 2006)
      - SCHED_IDLE (od v. 2.6.23, 2007)

# Plánovací třídy real-time

- SCHED_FIFO (nepreemptivní), SCHED_RR
- statická priorita – hodnota `rt_priority` > 0
- preempce pouze v těchto případech:
  - preempce procesem s vyšší prioritou
  - blokující systémové volání
  - zavolání sched_yield(2)
    - proces je vložen na konec fronty pro svou prioritu
  - SCHED_RR navíc po vypršení časového kvanta
    - sched_rr_get_interval(2), typicky 100 ms

# Ostatní plánovací třídy

- výchozí SCHED_NORMAL, `rt_priority` = 0
  - může běžet pouze tehdy, když neexistuje běhuschopný proces s prioritou real-time
- SCHED_BATCH (od jádra 2.6.16)
  - jádro vždy předpokládá, že proces je výpočetní
    - proces dostane malou penalizaci
  - vhodné pro neinteraktivní výpočetní procesy
- SCHED_IDLE (od 2.6.23, součást CFS)
  - nice nemá význam, větší penalizace než nice +19

# Completely Fair Scheduler

- jádro 2.6.23 (2007) – autor Ingo Molnár
  - autorem konceptu Con Kolivas – RSDS (SD)
    - Rotating Staircase Deadline Scheduler
- zohledňuje odlišné požadavky systémů
  - desktopové – minimální odezva
  - serverové – maximální výkon
  - lze za běhu přepínat
- nepoužívá klasickou frontu procesů
  - fronty nahradil strom (Red-Black Tree) + váhy

# CFS – Red-Black Tree

- lepší výkon než samovyvažovací stromy (AVL)
  - nejdelší cesta z kořene do listu není více než dvakrát delší než kterákoli jiná
  - nedokonale vyvážený strom, ale s nízkou režií
- klíčem je vážená virtuální doba běhu (VRT)
  - nízké hodnoty vlevo od kořene, vysoké vpravo
    - nejlevější úloha dostane CPU
  - uzlem může být skupina procesů (společná VRT)
    - př. procesy stejného uživatele

# Plánovací třída deadline

- SCHED_DEADLINE od v. 3.14 (2014)
  - nejvyšší priorita (vyšší než real-time)
  - založeno na EDF (Eealiest Deadline First) a CBS (Constant Bandwidth Server)
    - podle lhůty dokončení (EDF) s podporou rezervací (CBS)
  - nelze nastavit, pokud by systém nebyl plánovatelný
  - parametry: runtime ≤ deadline ≤ period (v ns)
    - runtime – obvykle > průměrná (nebo nejdelší) doba běhu
    - deadline – nejzazší doba dokončení (od začátku periody)

# CFS + Real-Time + Deadline

- limit pro RT a DL: max. 95 % času CPU
  - brání vyhladovění ne-RT procesů
  - lze měnit v **`/proc/sys/kernel/sched_*`**
    - sched_rt_runtime_us – max. pro RT úlohy: 950 000 µs
    - sched_rt_period_us – 100 % času CPU: 1 000 000 µs
- runqueue pro každé CPU
  - DL: Red-Black Tree, klíč: termín dokončení
  - RT: pole 100 seznamů (front) pro jednotlivé priority
  - ostatní: Red-Black Tree, klíč: virtual-runtime