08.
Čtenáři – písaři
Plánování procesů

ZOS 2024, L. PEŠIČKA

Problém čtenářů a písařů

- modeluje přístup do databáze
- rezervační systém (místenky, letenky)
- množina procesů, které chtějí přistupovat
  - souběžné čtení lze
  - výhradní zápis (žádný další čtenář ani písař)

Častá praktická úloha, lze realizovat s předností čtenářů, nebo s předností písařů.
Pro komerční aplikace je samozřejmě vhodnější přednost písařů.

```
// pseudokód čtenáři – písaři, přednost čtenářů
 semaphore m=1 ;          {chrání čítač rc}
 semaphore w=1;           {přístup pro zápis }
 int rc = 0;              { počet čtenářů }

void writer()
{
 P(w);
 …       // zapisuj
 V(w);
}
```

```
void reader()
{
 P(m);
 rc = rc + 1;
 if (rc == 1) P(w);    //1. čtenář blok. písaře
 V(m);
 …       // čti
 P(m);
 rc = rc - 1;
 if (rc==0) V(w);      // poslední čtenář odblokuje písaře
 V(m) ;
}
```

Čtenáři – písaři popis

- čtenáři
  - první čtenář provede P(w) – blokuje písaře
  - další čtenáři zvětšují čítač rc
  - po “přečtení” čtenáři zmenšují rc
  - poslední čtenář provede V(w) – odblokuje písaře
- semafor w
  - zabrání vstupu písaře, jsou-li čtenáři
  - zabrání vstupu čtenářům při běhu písaře:
    - prvnímu zabrání P(w)
    - ostatním brání P(m)
- toto řešení je s předností čtenářů
  - písaři musí čekat, až všichni čtenáři skončí

Implementace zámků v operačních a databázových systémech

- přístup procesu k souboru nebo záznamu databázi

- výhradní zámek (pro zápis)
  - nikdo další nesmí přistupovat
- sdílený zámek (pro čtení)
  - mohou o něj žádat další procesy
  - více jich může číst, ale nikdo nesmí zapisovat

- granularita zamykání
  - celý soubor x část souboru
  - tabulka x řádka v tabulce

Implementace zámků v OS

Linux, UNIX lze zamknout část souboru funkcí

fcntl (fd, F_SETLK, struct flock)

```
int fd;
struct flock fl;
fd = open("testfile", O_RDWR);
fl.l_type = F_WRLCK;                    - zámek pro zápis
fl.l_whence = SEEK_SET;                 - pozice od začátku souboru
fl.l_start = 100; fl,l_len = 10;        - pozice, kolik
fcntl (fd,F_SETLK, &fl);                - zamkneme pro zápis
                                          // vrací -1 pokud se nepovede
```

Implementace zámků v OS

Odemknutí

fl.l_type = F_UNLCK;	- odemknutí
fl.l_whence = SEEK_SET;	- pozice od začátku souboru
fl.l_start = 100; fl,l_len = 10;	- pozice v souboru, kolik bytů
fcntl (fd,F_SETLK, &fl);	- nastavíme

Dostupné operace

F_SETLK	- set / clear lock, nečeká
F_GETLK	- info o zámku
F_SETLKW	- nastavení zámku, čeká když je zamčený

Implementace zámků v OS

- zámky poradní (advisory)
  - nejsou vynucené
  - pro kooperující procesy
  - Linux - defaultní chování
- zámky mandatory
  - Linux – připojení fs přes volbu mount s –o mand
  - Linux – nastavené atributy souboru g-x,g+s
- různé způsoby zamykání:
  fcntl(), flock(), lockf()

Poznámky

mandatory vs. advisory locks:

http://stackoverflow.com/questions/575328/fcntl-lockf-which-is-better-to-use-for-file-locking

Locking in unix/linux is by default **advisory**, meaning other processes don't need to follow the locking rules that are set. So it doesn't matter which way you lock, as long as your co-operating processes also use the same convention.

Linux does support **mandatory** locking, but only if your file system is mounted with the option on and the file special attributes set. You can use mount -o mand to mount the file system and set the file attributes g-x,g+s to enable mandatory locks, then use fcntl or lockf. For more information on how mandatory locks work see here.

Note that locks are applied not to the individual file, but to the inode. This means that 2 filenames that point to the same file data will share the same lock status.

In Windows on the other hand, you can actively exclusively open a file, and that will block other processes from opening it completely. Even if they want to. I.e. The locks are mandatory. The same goes for Windows and file locks. Any process with an open file handle with appropriate access can lock a portion of the file and no other process will be able to access that portion.

čtenáři – písaři s předností písařů

funkce:

db_lock_r(x)	zámek pro čtení (sdílený, více procesů může číst)
db_lock_w(x)	zámek pro zápis (exkluzivní)
db_unlock_r(x)	odemčení záznamu x pro čtení
db_unlock_w(x)	odemčení záznamu x pro zápis

Algoritmus je složitější, ale v praxi více použitelný
uveden jen na ukázku

čtenáři – písaři s přednostní písařů

```
type zamek = record
 int wc = 0; int rc = 0;          // počet písařů a čtenářů
 semaphore mutw = 1;              // chrání přístup k čítači wc
 semaphore mutr = 1;              // chrání přístup k čítači rc
 semaphore wsem = 1;              // blokování písařů
 semaphore rsem = 1;              // blokuje 1. čtenáře
 semaphore rdel  = 1;             // blokování ostatních čtenářů
end;
```

```
void db_lock_w(var x: zamek)

 // uzamčení záznamu pro zápis
{
 P(x.mutw);
 x.wc = x.wc+1;                    // zvětšit počet písařů
 if (x.wc==1) P(x.rsem);           // 1.písař zablokuje 1. čtenáře
 V(x.mutw);
 P(x.wsem);                        // blokování písařů
}
```

```
void db_unlock_w(var x: zamek) {

// odemčení zápisů pro zápis
// sníží počet písařů, poslední písař odblokuje čtenáře

 V(x.wsem);                  // odblokování písařů
 P(x.mutw);
 x.wc = x.wc-1;
 if (x.wc==0)
          V(x.rsem);         // poslední písař pustí 1.čtenáře
 V(x.mutw);
}
```

```
void db_lock_r(var x: zamek)
{
 P(x.rdel);                    // nejsou blokováni ostatní čtenáři
 P(x.rsem);                    // není blokován 1. čtenář
 P(x.mutr);
 x.rc = x.rc+1;
 if (x.rc==1)  P(x.wsem);      // 1. čtenář zablokuje písaře
 V(x.mutr);
 V(x.rsem);
 V(x.rdel);
}
```

```
void db_unlock_r(var x: zamek)
{
 P(x.mutr);
 x.rc = x.rc-1;
 if (x.rc==0)   V(x.wsem);        // poslední čtenář odblokuje
                                  // písaře
 V(x.mutr) ;
}
```

Plánování procesů

Základní stavy procesu:

- běžící – může běžet tolik, kolik je jader procesů (a hyperthreading)
- připraven – čeká na CPU
- blokován – čeká na zdroj nebo zprávu
- nový (new) – proces byl právě vytvořen
- zombie – ukončený proces, ale stále má záznam v PCB
- ukončený (terminated) – proces byl ukončen

Správce procesů – udržuje tabulku procesů

Záznam o konkrétním procesu – PCB (Process Control Block)
– souhrn dat potřebných k řízení procesů

opakování (!!)

v Linuxu je datová struktura task_struct, která obsahuje informace o procesu (tj. představuje PCB)

- každý proces má záznam (řádku) v **tabulce procesů**
- tomuto záznamu se říká **PCB** (process control block)

- PCB obsahuje všechny potřebné informace (tzv. kontext procesu) k tomu, abychom mohli proces kdykoliv pozastavit (odejmout mu procesor) a znovu jej spustit od tohoto místa přerušení (Program Counter - PC: CS:EIP)
- proces po opětovném přidělení CPU pokračuje ve své činnosti, jako by k žádnému přerušení vykonávání jeho kódu nedošlo, je to z jeho pohledu transparentní

opakování (!!)

- kde leží tabulka procesů?
  v paměti RAM, je to datová struktura jádra OS
- kde leží informace o PIDu procesu?
  v tabulce procesů -> v PCB (řádce tabulky) tohoto procesu
- jak procesor ví, kterou instrukci procesu (vlákna) má vykonávat?
  podle program counteru (PC, typicky CS:EIP), ukazuje na oblast v paměti, kde leží vykonávaná instrukce;
  obsah CS:EIP, stejně jako dalších registrů je součástí PCB

opakování (!!)

- jak vytvořím nový proces?
  - systémovým voláním fork()
- jak vytvořím nové vlákno?
  - voláním pthread_create()
- jak spustím jiný program?
  - systémovým voláním execve()
  - začne vykonávat kód jiného programu v rámci existujícího procesu

Běh programu v C

- Pohled na program C
  z hlediska programátora a z hlediska OS

- Spuštění funkce main
  - Může mít argumenty argc, argv
- Návrat z funkce main – return
  - Ukončení programu
  - Předání návratové hodnoty,
    např. z bashe otestovat echo $?

Běh programu v C (z pohledu OS)

- OS vytvoří virtuální paměťový prostor pro daný proces
  - Kód, data, zásobník
- Nahraje program do paměťového prostoru, včetně knihoven
- OS předá řízení danému programu

- Crt0 – C runtime 0
  - Startovací rutina přidaná k C programu, provede potřebné inicializace, než se zvolá funkce main

Plánovač x dispatcher

- dispatcher předává řízení procesu, který byl vybrán short time plánovačem:
  - přepnutí kontextu
  - přepnutí do user modu
  - skok na uloženou adresu instrukce pouštěného programu, aby pokračovalo jeho vykonávání

- více připravených procesů k běhu – plánovač vybere, který spustí jako první
- plánovač procesů (scheduler) používá plánovací algoritmus (scheduling algorithm)

Pamatuj

Plánovač určí, který proces (vlákno) by měl běžet nyní.

Dispatcher provede vlastní přepnutí z aktuálního běžícího procesu na nově vybraný proces.

Plánování procesů - vývoj

- dávkové systémy
  - Úkol: spustit další úlohu v pořadí, nechat ji běžet do konce
  - Uživatel s úlohou nekomunikuje, zadá program plus vstupní data např. v souboru
  - O výsledku je uživatel informován, např. e-mailem aj.
- systémy se sdílením času
  - Můžeme mít procesy běžící na pozadí
  - interaktivní procesy – komunikují s uživatelem
- kombinace obou systémů (dávky, interaktivní procesy)

- chceme: přednost interaktivních procesů
  - Srovnejte: odesílání pošty x zavírání okna

Střídání CPU a I/O aktivit procesu

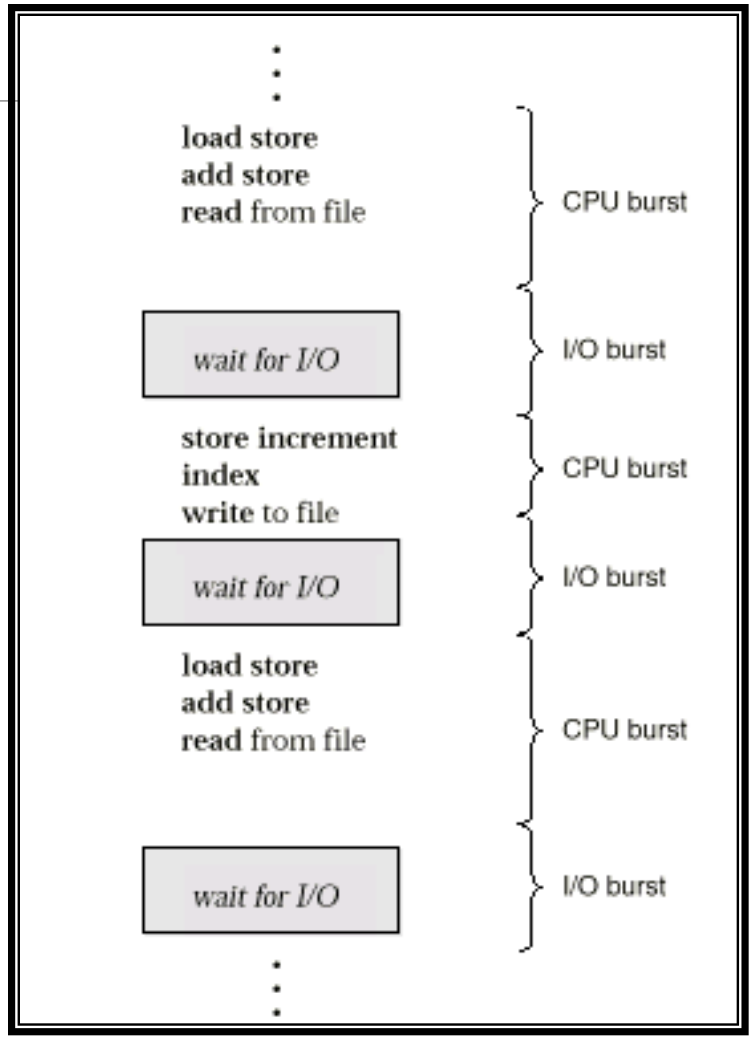
Během vykonávání procesu se střídají úseky:

- CPU burst (vykonávání kódu)
- I/O burst (čekání)

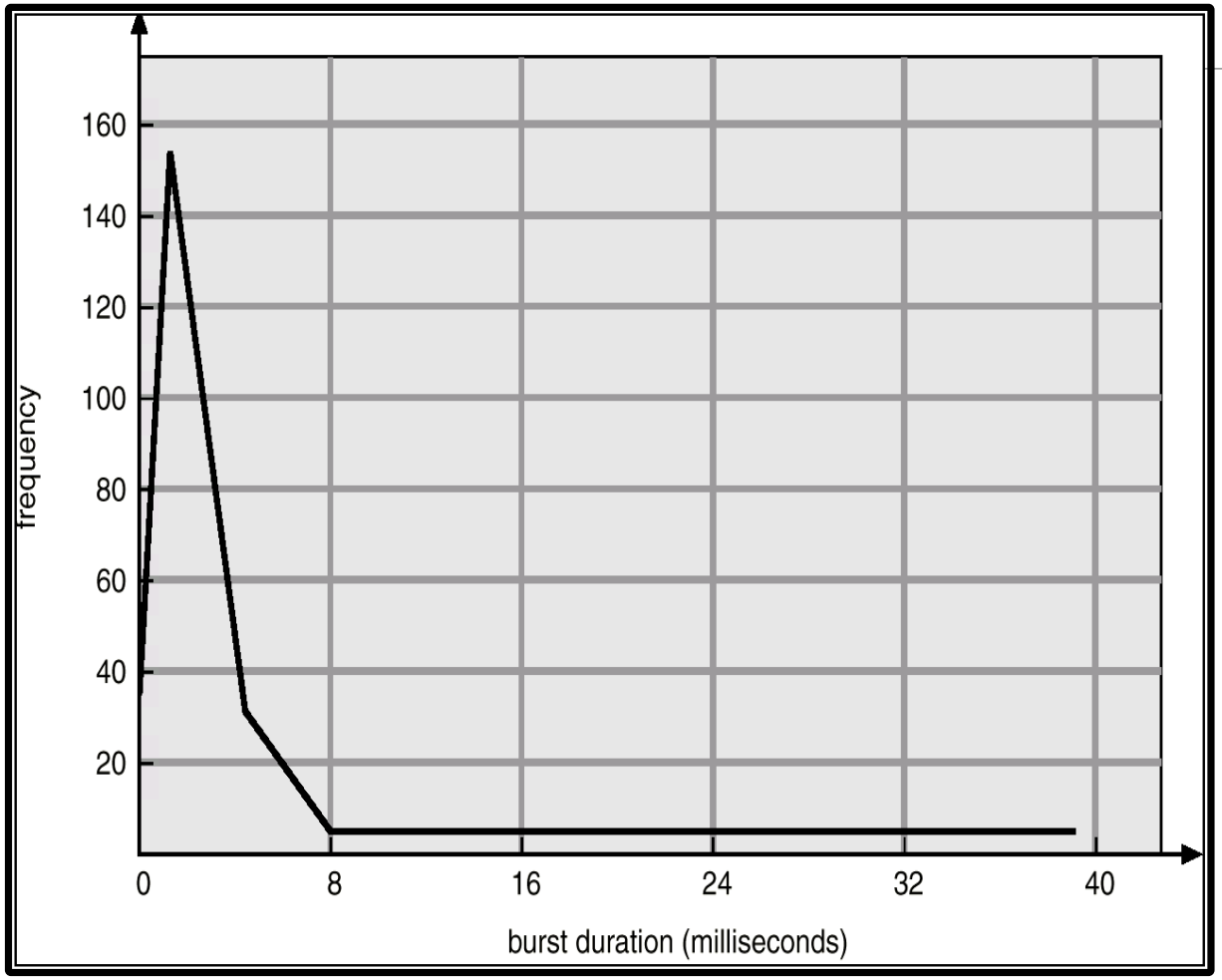
střídání těchto fází

končí CPU burstem (není na obrázku)

Pro CPU bursty obvykle platí:
hodně krátkých burstů
a málo dlouhých



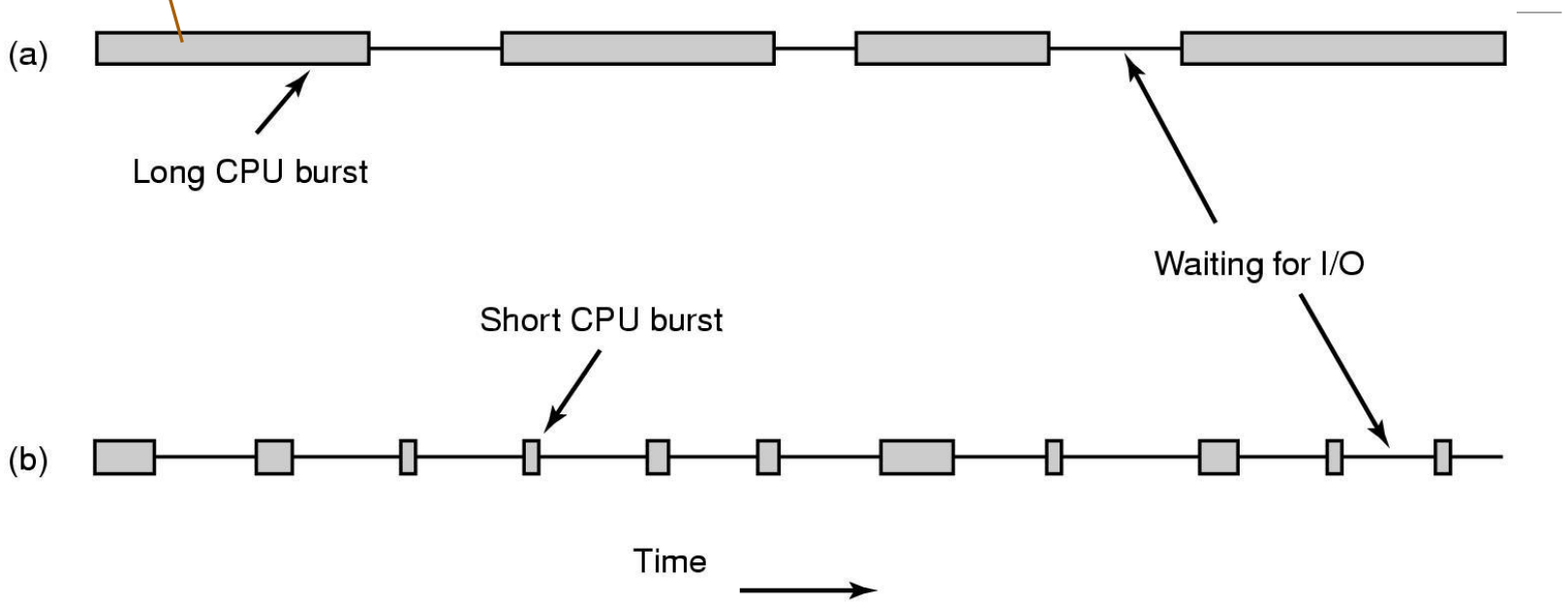
Histogram CPU burstů



Hodně krátkých burstů

počítám

Plánování



a) CPU-vázaný proces („hodně času tráví výpočtem“)
b) I/O vázaný proces („hodně času tráví čekáním na I/O“)

Uveďte příklady CPU vázaného a I/O vázaného procesu

Nepreemptivní plánování

Nepreemptivní (chybí přechod z běžící do připravený)

- každý proces dokončí svůj CPU burst (!!!!)
- proces si podrží kontrolu nad CPU, dokud se jí sám nevzdá (I/O čekání, ukončení)
- lze použít v dávkových systémech, není ale vhodné pro time sharingové systémy (se sdílením času)
- Win 3.x non-preemptivní (kooperativní) plánování, od Win95 preemptivní

Jaký má vliv non-preemptivnost systému na obsluhu kritické sekce u jednojádrového CPU?

Preemptivní plánování

Preemptivní plánování (je přechod z běžící do připravený)

- proces lze přerušit **KDYKOLIV** během CPU burstu a naplánovat jiný (-> problém kritických sekcí i na jednojádru !!!)
- dražší implementace kvůli přepínání procesů (režie)
- Vyžaduje speciální hardware – timer (časovač)
  časovač je na základní desce počítače, pravidelně generuje hardwarová přerušení systému od časovače

Část výkonu systému spotřebuje režie nutná na přepínání procesů.
K přepnutí na jiný proces také může dojít v nevhodný čas (ošetření KS).
Preemptivnost je ale u současných systémů důležitá, pokud potřebujeme interaktivní odezvu systému.
Časovač tiká (generuje přerušení), a po určitém množství tiků se určí, zda procesu nevypršelo jeho časové kvantum.

Preempce jádra OS

- preempce jádra OS
  - přeplánování ve chvíli, kdy se manipuluje s daty (I/O fronty) používanými jinými funkcemi jádra..
  - UNIX (když bylo jádro dříve nepreemptivní)
    - čekání na dokončení systémového volání
    - nebo na dokončení I/O
    - výhodou jednoduchost jádra
    - nevýhodou výkon v RT a na multiprocesorech

Preempce se může týkat nejen uživatelských procesů, ale i jádra OS. Linux umožňuje zkompilovat a dnes běžně používá preemptivní jádro.

Cíle plánování - společné

- Spravedlivost (Fairness)
  - Srovnatelné procesy srovnatelně obsloužené
- Vynucení politiky (Policy enforncement)
  - Bude vyžadováno dodržení stanovených pravidel
- Balance (Balance)
  - Snaha, aby všechny části systému (CPU, RAM, periferie) byly vytížené
- Nízká režie plánování
  - Výkon je třeba věnovat procesům, ne plánovači ☺

Cíle plánování – **dávkové syst.**

- Propustnost (Throughput)
  - maximalizovat počet jobů za jednotku času (hodinu)
- Doba obrátky (Turnaround time)
  - minimalizovat čas mezi přijetím úlohy do systému a jejím dokončením
- CPU využití
  - snaha mít CPU pořád vytížené

Cíle plánování

All systems

Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems

Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and termination
CPU utilization - keep the CPU busy all the time

Interactive systems

Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems

Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

Některé cíle jsou společné, jiné se liší dle typu systému

Zajímavosti

V roce 1973 provedli na MITu
shut-down systému IBM 7094
a našli low priority proces,
který nebyl dosud spuštěný
a přitom byl založený …..

..v roce 1967 ..

Plánovač (!!!)

- **rozhodovací mód**
  - okamžik, kdy jsou vyhodnoceny priority procesu a vybrán proces pro běh (KDY)
- **prioritní funkce**
  - určí prioritu procesu v systému
- **rozhodovací pravidla**
  - jak rozhodnout při stejné prioritě

Tři zásadní údaje, které charakterizují plánovač

Plánovač – **Rozhodovací mód**

- Nepremptivní systém
  - Proces využívá CPU, dokud se jej sám nevzdá (např. I/O)
  - jednoduchá implementace
  - vhodné pro dávkové systémy
  - nevhodné pro interaktivní a RT systémy

- Preemptivní systém
  - **Kdy** dojde k vybrání dalšího procesu pro běh?
    - periodicky – časové kvantum (interaktivní systémy) - nejčastější
    - přijde nový proces (algoritmus SRT u dávkových systémů)
    - priorita připraveného > běžícího (RT systémy)
  - Náklady (režie)
    - přepínání procesů, logika plánovače

Plánovač – **Prioritní funkce**

- určuje prioritu procesu v systému
- funkce, bere v úvahu parametry procesu a systémové parametry
- externí priority
  - třídy uživatelů(„VIP procesy“), systémové procesy
- priority odvozené z chování procesu
  - dlouho čeká na CPU, nebo dlouho čeká na I/O aj.
- Většinou dvě složky – statická a dynamická priorita
  - Statická – přiřazena při startu procesu
  - Dynamická – dle chování procesu (dlouho čekal, aj.)

Prioritní funkce (!)

priorita: statická a dynamická

proč 2 složky?
pokud by chyběla:

- statická – nemohl by uživatel např. při startu označit proces jako důležitější než jiný
- dynamická – proces by mohl vyhladovět, mohl by být neustále předbíhán v plánování jinými procesy s lepší prioritou

Plánovač – Prioritní funkce

Co všechno **může** vzít v úvahu prioritní funkce:

- čas, jak dlouho proces využíval CPU
- aktuální zatížení systému
- paměťové požadavky procesu
- čas, který proces strávil v systému
- celková doba provádění úlohy (limit)
- urgence (RT systémy)

Plánovač – **Rozhodovací pravidlo**

- pokud je malá pravděpodobnost stejné priority
  - náhodný výběr

- velká pravděpodobnost stejné priority
  - cyklické přidělování kvanta
  - chronologický výběr (FIFO)

Prioritní funkce může být navržena tak, že málokdy vygeneruje stejné priority, nebo naopak může být taková, že často (nebo když se nepoužívá vždy) určí stejnou hodnotu.
Pak nastupuje rozhodovací pravidlo.

Cíle plánovacích algoritmů

Každý algoritmus nutně upřednostňuje nějakou třídu úloh na úkor ostatních.

- dávkové systémy
  - dlouhý čas na CPU, omezí se přepínání úloh (režie)
- interaktivní systémy
  - interakce s uživatelem, tj. I/O úlohy
- systémy reálného času
  - dodržení deadlines

Dávkové systémy (!)

- průchodnost (throughput)
  - počet úloh dokončených za časovou jednotku
- průměrná doba obrátky (turnaround time)
  - průměrná doba od zadání úlohy do systému do dokončení úlohy
- využití CPU

Průchodnost a průměrná doba obrátky jsou různé údaje ! Někdy snaha vylepšit jednu hodnotu může zhoršit druhou z nich.

Dávkové systémy

- maximalizace průchodnosti nemusí nutně minimalizovat dobu obrátky

- modelový příklad:
  - dlouhé úlohy následované krátkými
  - upřednostňování krátkých
  - bude tedy dobrá **průchodnost**
  - dlouhé úlohy se nevykonají
    -> **doba obrátky** bude nekonečná

Interaktivní systémy

Chceme:

Minimalizaci doby odpovědi

ale je třeba dbát na:

efektivitu – drahé přepínání mezi procesy

Realtimové systémy

Dodržení deadlines

- termín, do kdy musí být daný proces obsloužen

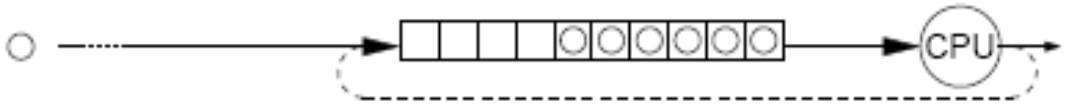
Předvídatelnost

- Některé akce pravidelné, periodické (např. generování zvuku)
- Příklad procesu – např. musí běžet každé 2s a vždy spotřebuje 0.1s
- Vytvoříme si kalendář, zda je daný proces plánovatelný

Plánování úloh v dávkových systémech

- FCFS (First Come First Served)
- SJF (Shortest Job First)
- SRT (Shortest Remaining Time)
  jediný z nich je preemptivní – vychází z SJF
- Multilevel Feedback

FCFS (First Come First Served)



Nepreemptivní FIFO

- Nově příchozí na konec fronty připravených
- Úloha běží, když opustí CPU vybrána další ve frontě

Co když úloha provádí I/O operaci? Přístupy:

1. Zablokována, CPU se nevyužívá (prvotní varianta) – není vůbec efektivní
2. Do stavu blokovaný, běží jiná úloha po dokončení I/O je naše úloha zařazena na **konec** fronty připravených (častá varianta !!!)
   - I/O vázané úlohy znevýhodněny před výpočetně vázanými
3. Po dokončení I/O na začátek fronty připravených

Poznámka

V následujících příkladech předpokládáme, že se uvažovaná úloha skládá jen z 1 dlouhého CPU burstu,
tj. nečeká na I/O, tj. jen počítá

-> aby šlo lépe nakreslit diagramy

U dávkových úloh můžeme odhadnout dobu provádění úlohy (třeba z dřívějších běhů) a tento čas může být velmi významný pro rozhodnutí plánovače.

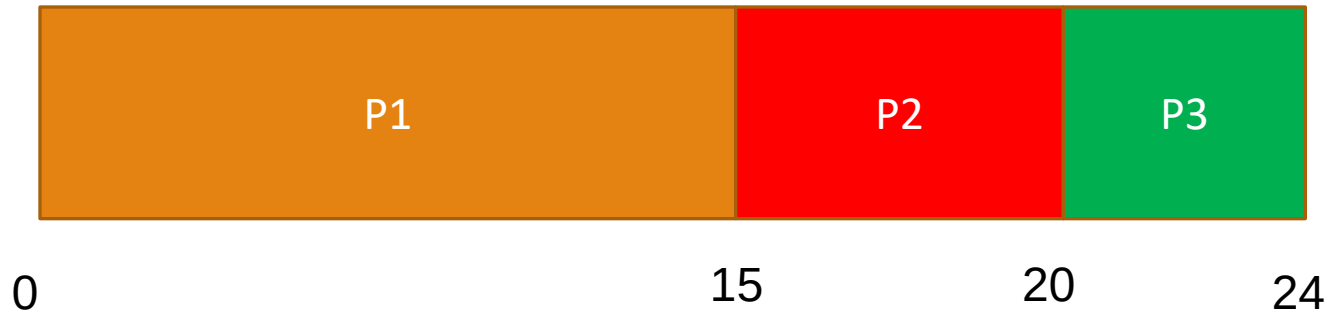
FCFS příklad

V čase nula budou v systému procesy P1, P2, P3 přišlé v tomto pořadí:

proces	Doba trvání (s)
P1	15
P2	5
P3	4

doba obrátky:
odešel
-
přišel

tedy:
P1: 15-0 = 15
P2: 20-0 = 20
P3: 24-0 = 24



průměrná doba obrátky: (15+20+24) / 3 = 19,666

SJF (Shortest Job First)

- Nejkratší úloha jako první
- Předpoklad – známe přibližně dobu trvání úloh
- Nepreemptivní
  - Jedna fronta příchozích úloh
  - Plánovač vybere vždy úlohu s nejkratší dobou běhu

Optimalizuje dobu obrátky

průměrná doba obrátky: (4+9+24) / 3 = 12,3

Výpočet průměrné doby obrátky

Do systému přijdou v čase nula úlohy A,B,C,D s dobou běhu: 8, 4, 4, 4 minut. Úlohy budou tvořeny jedním CPU burstem.

FCFS

-Spustí v pořadí A, B, C, D dle strategie FCFS

-Doba obrátky:

- A 8 minut
- B 8+4 = 12 minut
- C 8+4+4 = 16 minut
- D 8+4+4 +4 = 20 minut

-Průměrná doba obrátky:
(8+12+16+20) / 4 = 14 minut

Výpočet průměrné doby obrátky

strategie plánovače **SJF**

V pořadí B, C, D, A – od nejkratší

- B 4 minuty
- C 4+4 = 8 minut
- D 4+4+4 = 12 minut
- A 4+4+4+8 = 20 minut

Průměrná doba obrátky
(4+8+12+20) / 4 = 11 minut

Průměrná doba obrátky je v tomto případě lepší

SRT (Shortest Remaining Time)

Úlohy samozřejmě můžou přicházet **kdykoliv *(nejen v čase nula)***

Preemptivní (má přechod běžící - připravený)

- Plánovač vždy vybere úlohu, jejíž **zbývající** doba běhu je nejkratší (!!!)

KDY dojde k preempci:
Právě prováděné úloze zbývá 10 minut, do systému právě teď přijde úloha s dobou běhu 1 minutu – systém prováděnou úlohu pozastaví a nechá běžet novou úlohu

i když by byla tvořena jen CPU burstem

Možnost vyhladovění dlouhých úloh (!) => neustále předbíhány krátkými

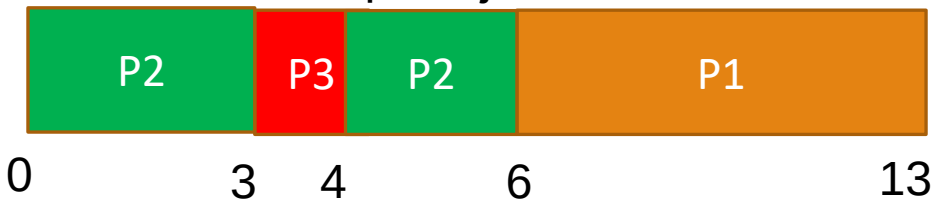
SRT příklad

Čas příchodu	Název úlohy	Doba úlohy (s)
0	P1	7
0	P2	5
3	P3	1

V čase 0 máme na výběr P1, P2. Naplánujeme P2 s kratší dobou běhu

V čase 3 přijde do systému nová úloha. Zkontrolujeme zbývající doby běhu úloh: P1(7), P2 (2), P3(1). Naplánujeme P3.

Jakmile skončí P3, naplánujeme P2, až doběhne, naplánujeme P1, …

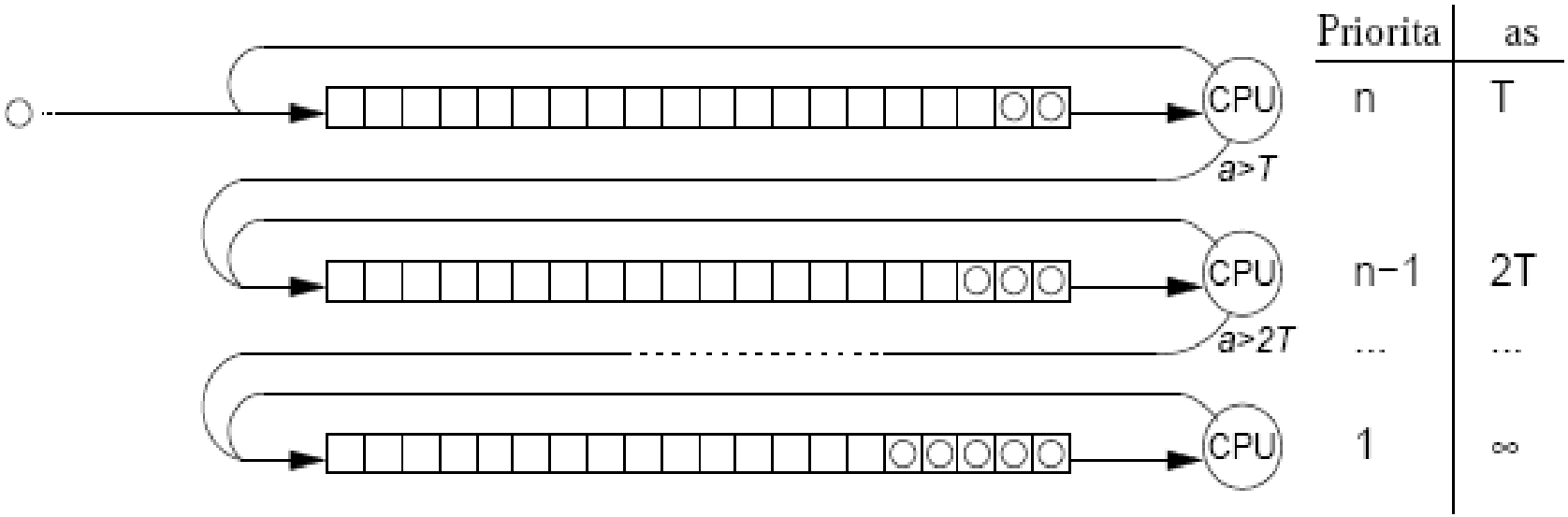


Multilevel feedback

- **N** různých úrovní – front (představují „prioritu“)
- Každá úroveň má svojí frontu úloh
- Úloha vstoupí do systému do **nejvyšší fronty** (!)
- Na každé prioritní úrovni
  - Stanoveno maximum času CPU, který může úloha obdržet
  - Např.: T na úrovni n, 2T na úrovni n-1 atd.
  - Pokud úloha překročí tento limit, její priorita se sníží
  - Na nejnižší prioritní úrovni může úloha běžet neustále nebo lze překročení určitého času považovat za chybu
- Procesor obsluhuje nejvyšší neprázdnou frontu (!!)

I když v MF mluvíme o prioritě front, úlohy zde žádnou explicitní prioritu nemají

Multilevel feedback



První fronta – dokud nemá na CPU celkem stráven čas T
Druhá fronta – dokud nemá na CPU celkem stráven čas T+T=2T

Výhoda – upřednostňuje I/O vázané úlohy – déle se drží ve vysokých frontách („úlohy co ještě moc času nenapočítaly“)

Shrnutí – dávkové systémy

Algoritmus	Rozh. mód	Prioritní funkce	Rozh. pravidlo
FCFS	nepreemptivní	P(r) = r	Náhodně
SJF	nepreemptivní	P(t) = -t	Náhodně
SRT	**Preemptivní (při příchodu nové úlohy)**	P(a,t) = a-t	FIFO nebo náhodně
MLF	nepreemptivní	viz popis	FIFO v rámci fronty

r celkový čas strávený úlohou v systému
t předpokládaná délka běhu úlohy
a čas strávený během úlohy v systému

Plánování procesů v interaktivních systémech

- potřeba docílit, aby proces neběžel „příliš dlouho“
  - možnost obsloužit další procesy – na každého se dostalo
- nevíme jak dlouhý bude CPU burst procesu
  - Nelze spoléhat, že se proces sám brzy zablokuje (nad I/O, semaforem, ...)

- vestavěný systémový časovač v počítači
  - provádí pravidelně přerušení (tiky časovače, clock ticks)
  - vyvolá se obslužný podprogram v jádře
  - rozhodnutí, zda proces bude pokračovat, nebo se spustí jiný (preemptivní plánování) – po několika tikách časovače

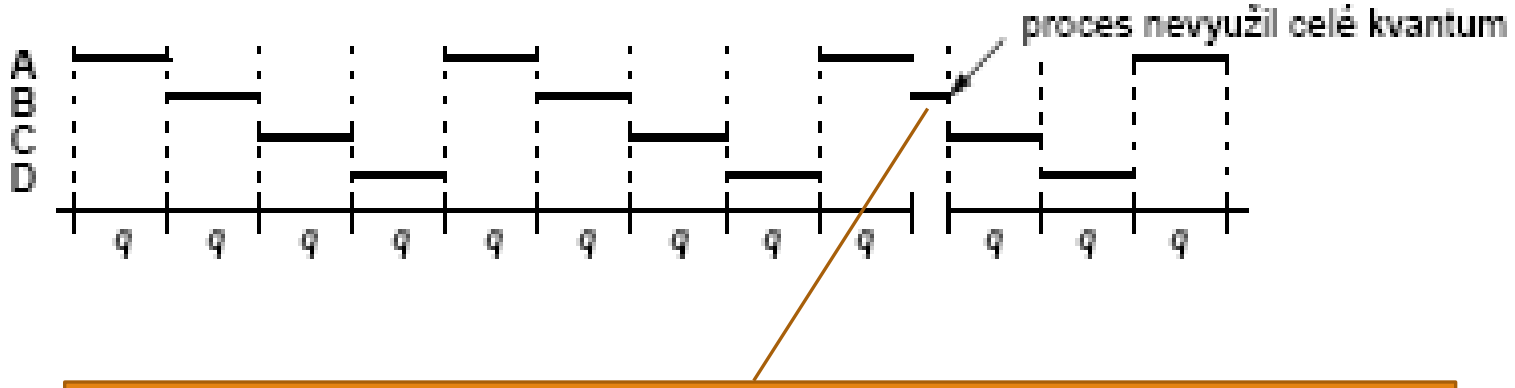
Algoritmus cyklické obsluhy – Round Robin (RR)

- jeden z nejstarších a nejpoužívanějších
- každému procesu přiřazen
  časový interval = časové kvantum, po které může proces běžet

- proces běží na konci svého kvanta
  - preemce, spuštěn je další připravený proces

- proces skončí nebo se zablokuje před uplynutím kvanta
  - hned je spuštěn další připravený proces

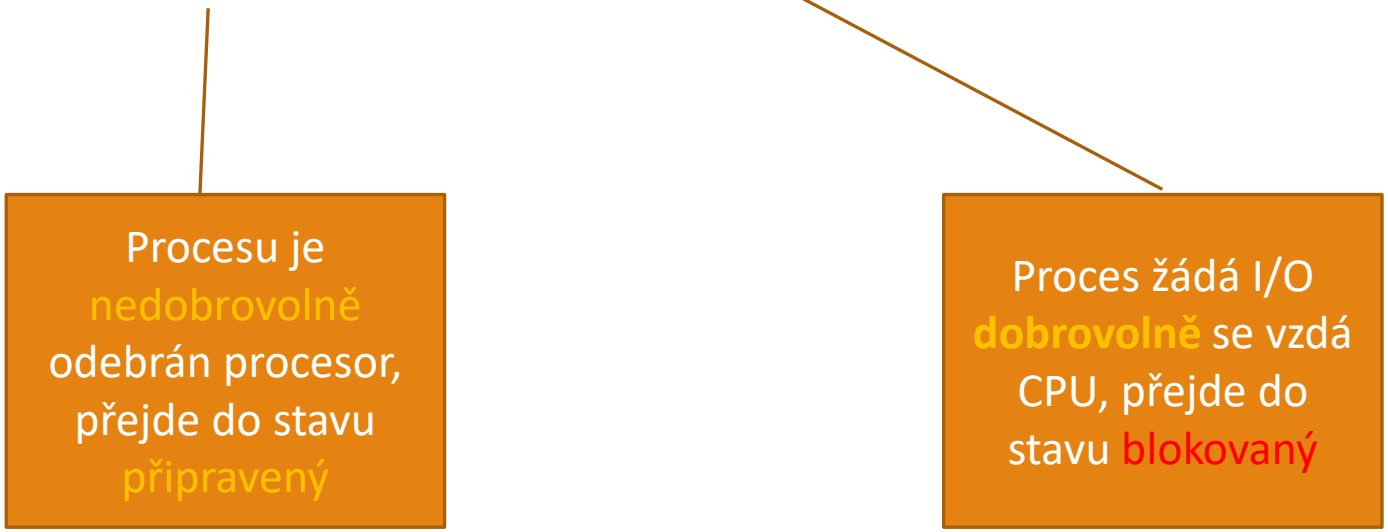
Round Robin



Pokud proces nevyužije celé časové kvantum,
okamžitě se naplánuje další proces, na nic se nečeká
(je třeba max. využít procesor)

Round Robin

- plánovač udržuje seznam připravených procesů
  - Při vypršení kvanta nebo zablokování nebo ukončení procesu –> vybere další proces



Obslužný program přerušení časovače

- v jádře

- nastavuje interní časovače systému
- shromažďuje statistiky systému
  - kolik času využíval CPU který proces, ...
- po uplynutí kvanta (resp. v případě potřeby) zavolá plánovač

1 časové kvantum – odpovídá více přerušením časovače

Časovač může proces přerušit vícekrát v průběhu časového kvanta.

Pokud bude přerušení 100x za sekundu, tj. každých 10ms

a časové kvantum bude mít např. hodnotu 50ms

=> přeplánování každý pátý tik

vhodná délka časového kvanta

- krátké
  - přepnutí procesů chvíli trvá (uložení a načtení registrů, přemapování paměti, ...)
  - pokud by trvalo přepnutí kontextu 1ms, kvantum 4ms – 20% velká režie
- dlouhé
  - vyšší efektivita; kvantum 1s – menší režie
  - pokud kvantum delší než průměrná doba držení CPU procesem – preempce je třeba zřídka
  - problém interaktivních procesů – kvantum 1s, 10 uživatelů stiskne klávesu, odezva posledního procesu až 10s

vhodná délka kvanta - shrnutí

- krátké kvantum – snižuje efektivitu (režie)
- dlouhé – zhoršuje dobu odpovědi na interaktivní požadavky
- kompromis ☺
- pro algoritmus cyklické obsluhy obvykle 20 až 50 ms
- kvantum nemusí být konstantní
  - Je to jen číslo, co jde změnit

Problém s algoritmem cyklické obsluhy

- v systému výpočetně vázané i I/O vázané úlohy

- výpočetně vázané – většinou kvantum spotřebují
- I/O vázané – pouze malá část kvanta se využije a zablokují se
- => výpočetně vázané získají nespravedlivě vysokou část času CPU

- modifikace VRR (Virtual RR, 1991)
  - procesy po dokončení I/O mají přednost před ostatními
  - jeden z možných návrhů řešení

Prioritní plánování

- (RoundRobin: všechny procesy stejně důležité)
- ale:
  - vyšší priorita zákazníkům, kteří si „připlatí“
  - interaktivní procesy vs. procesy běžící na pozadí
- prioritu lze přiřadit staticky nebo dynamicky:
- staticky
  - při startu procesu, např. Linux – příkaz nice
- dynamicky
  - přiradit I/O procesům větší prioritu, použití CPU a zablokování

Priorita

priorita = statická + dynamická

- obsahuje obě složky – výsledná jejich součtem
- statická (při startu procesu)
- dynamická (chování procesu v poslední době)

- kdyby pouze statická složka a plánování jen podle priorit – běží pouze připravené s nejvyšší prioritou
- plánovač snižuje dynamickou prioritu běžícího procesu při každém tiku časovače
  - klesne pod prioritu jiného -> přeplánování

Dynamická priorita (!!)

V kvantově orientovaných plánovacích algoritmech:

dynamická priorita např. dle vzorce: 1 / f (!)

f – velikost části kvanta, kterou proces naposledy použil

zvýhodní I/O vázané x CPU vázaným

Pokud proces nevyužil celé kvantum, jeho dynamická priorita se zvyšuje, např. pokud využil posledně jen 0.5 kvanta, tak 1/0,5 = 2, pokud celé kvantum využil 1/1=1

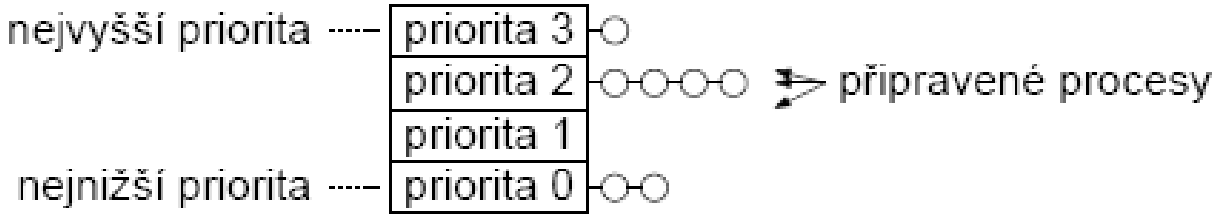
Spojení cyklického a prioritního plánování

- prioritní třídy
  - v každé třídě procesy se stejnou prioritou
- prioritní plánování mezi třídami
  - Bude obsluhována třída s nejvyšší prioritou
- cyklická obsluha uvnitř třídy
  - V rámci dané třídy se procesy cyklicky střídají
- obsluhovány jsou pouze připravené procesy v nejvyšší neprázdné prioritní třídě

A kdy se dostane na další fronty?

Prioritní třídy

Máme zde
priority, třídy i časová kvanta



- 4 prioritní třídy
- dokud procesy v třídě 3 – spustit cyklicky každý na 1 kvantum
- pokud třída 3 prázdná – obsluhujeme třídu 2
  - (prázdná => žádný proces danou prioritu nemá, nebo má, ale je ve stavu blokovaný, čeká např. na I/O)
- jednou za čas – přepočítání priorit
  - procesům, které využívaly CPU se sníží priorita

Prioritní třídy

- dynamické přiřazování priority
  - dle využití CPU **v poslední době**
  - priorita procesu
    - snižuje se při běhu
    - zvyšuje při nečinnosti
- cyklické střídání procesů

- OS typu Unix
  - Mají typicky cca 30 až 50 prioritních tříd (ale i více)

Plánovač spravedlivého sdílení (!)

problém:

- čas přidělován každému procesu nezávisle
- Pokud uživatel má více procesů než jiný uživatel
  -> dostane více času celkově

spravedlivé sdílení

- přidělovat čas každému uživateli (či jinak definované skupině procesů) proporcionálně, bez ohledu na to, kolik má procesů
- máme-li N uživatelů, každý dostane 1/N času

= spravedlnost vůči uživateli
jde o modifikaci plánovače, aby byl spravedlivý vůči uživatelům

Spravedlivé sdílení

- nová položka: priorita skupiny spravedlivého plánování
  - Zavedena pro každého uživatele

- obsah položky
  - započítává se do priority každého procesu uživatele
  - odráží poslední využití procesoru všemi procesy daného uživatele

Má-li uživatel Pepa procesy p1, p2, p3
a pokud proces p3 bude využívat CPU hodně často, budou touto položkou penalizovány i další procesy uživatele Pepa

Spravedlivé sdílení – implementace (!)

- každý uživatel – položka g
- obsluha přerušení časovače – inkrementuje g uživatele, kterému patří právě běžící proces
- jednou za sekundu rozklad: g=g/2
  - Aby odrážel chování **v poslední době**, vzdálená minulost nás nezajímá
- priorita P (p,g) = p – g
- pokud procesy uživatele využívaly CPU v poslední době – položka g je vysoká, vysoká penalizace

Plánování pomocí loterie

- Lottery Scheduling (Waldspurger & Weihl, 1994)
- cílem – poskytnout procesům příslušnou proporci času CPU
- základní princip:
  - procesy obdrží tikety (losy)
  - plánovač vybere náhodně jeden tiket
  - vítězný proces obdrží cenu – 1 kvantum času CPU
  - důležitější procesy – více tiketů, aby se zvýšila šance na výhru (např. celkem 100 losů, proces má 20 – v dlouhodobém průměru dostane 20% času)

Loterie - výhody

řešení problémů, které jsou v jiných plán. algoritmech obtížné

- spolupracující procesy – mohou si předávat losy
  - klient posílá zprávu serveru a blokuje se
  - klient může serveru propůjčit všechny své tikety
  - server běží s prioritou (počtem losů) daného klienta
  - po vykonání požadavku server tikety vrátí
  - nejsou-li požadavky, server žádné tikety nepotřebuje

Loterie - výhody

- rozdělení času mezi procesy v určitém poměru
  - to bychom těžko realizovali u prioritního plánování, co znamená, že jeden proces má prioritu např. 30 a jiný 10?
  - proces – množství ticketů – velikost šance vyhrát

zatím spíše experimentální algoritmus

Loterie - nevýhody

- není deterministický
- Nemáme zaručeno, že budou naše losy „vylosovány“ v konečném čase.
- Nelze použít tam, kde jsme na determinismu závislí -> řídící aplikace.

Shrnutí

Algoritmus	Rozhodovací mód	Prioritní funkce	Rozhodovací pravidlo
RR	Preemptivní vyprší kvantum	P() = 1	cyklicky
prioritní	Preemptivní P jiný > P	Viz text	Náhodně, cyklicky
spravedlivé	Preemptivní P jiný > P	P(p,g)=p-g	cyklicky
loterie	Preemptivní vyprší kvant.	P() = 1	Dle výsledku loterie

Interaktivní systémy (shrnutí)

- Základem je round robin
  - Pojem časové kvantum
- Prioritní plánování
  - Statická a dynamická složka priority
- Spojení RR + priority => prioritní třídy
- Spravedlivé sdílení
  - Modifikace plánovače pro spravedlnost vůči uživatelům
- Loterie
  - Experimentální, zajímavé vlastnosti
  - Nelze použít pro řídící systémy - nedeterminismus

Odkaz – plánování Windows

Příklad – Windows 2000/XP/…

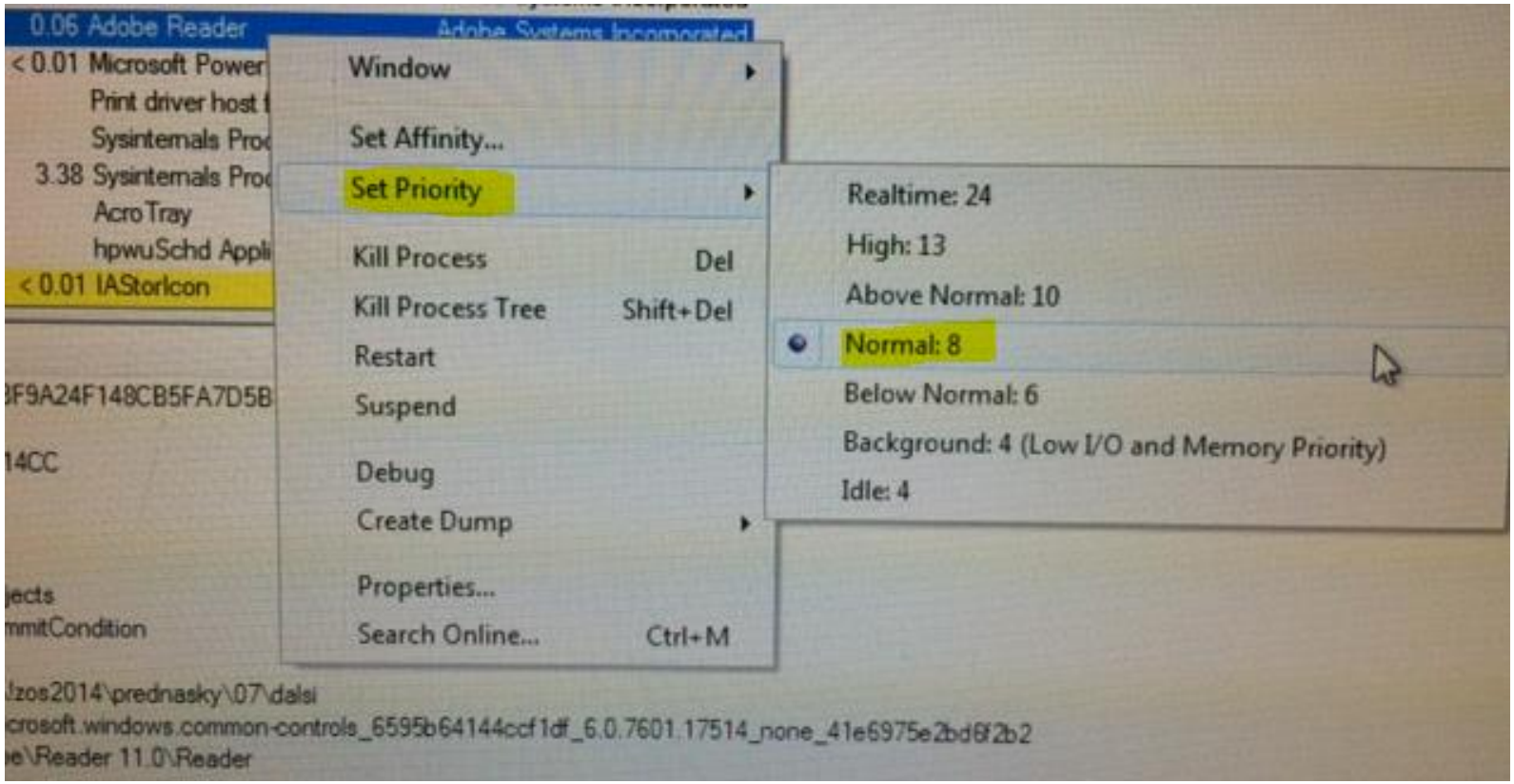
- 32 prioritních úrovní, 0 až 31 (nejvyšší)
- pole 32 položek
  - každá položka – ukazatel na seznam připravených procesů
- plánovací algoritmus – prohledává pole od 31 po 0
  - nalezne neprázdnou frontu
  - naplánuje první proces, nechá ho běžet 1 kvantum
  - po uplynutí kvanta – proces na konec fronty na příslušné prioritní úrovni

viz https://docs.microsoft.com/cs-cz/windows/win32/procthread/scheduling?redirectedfrom=MSDN

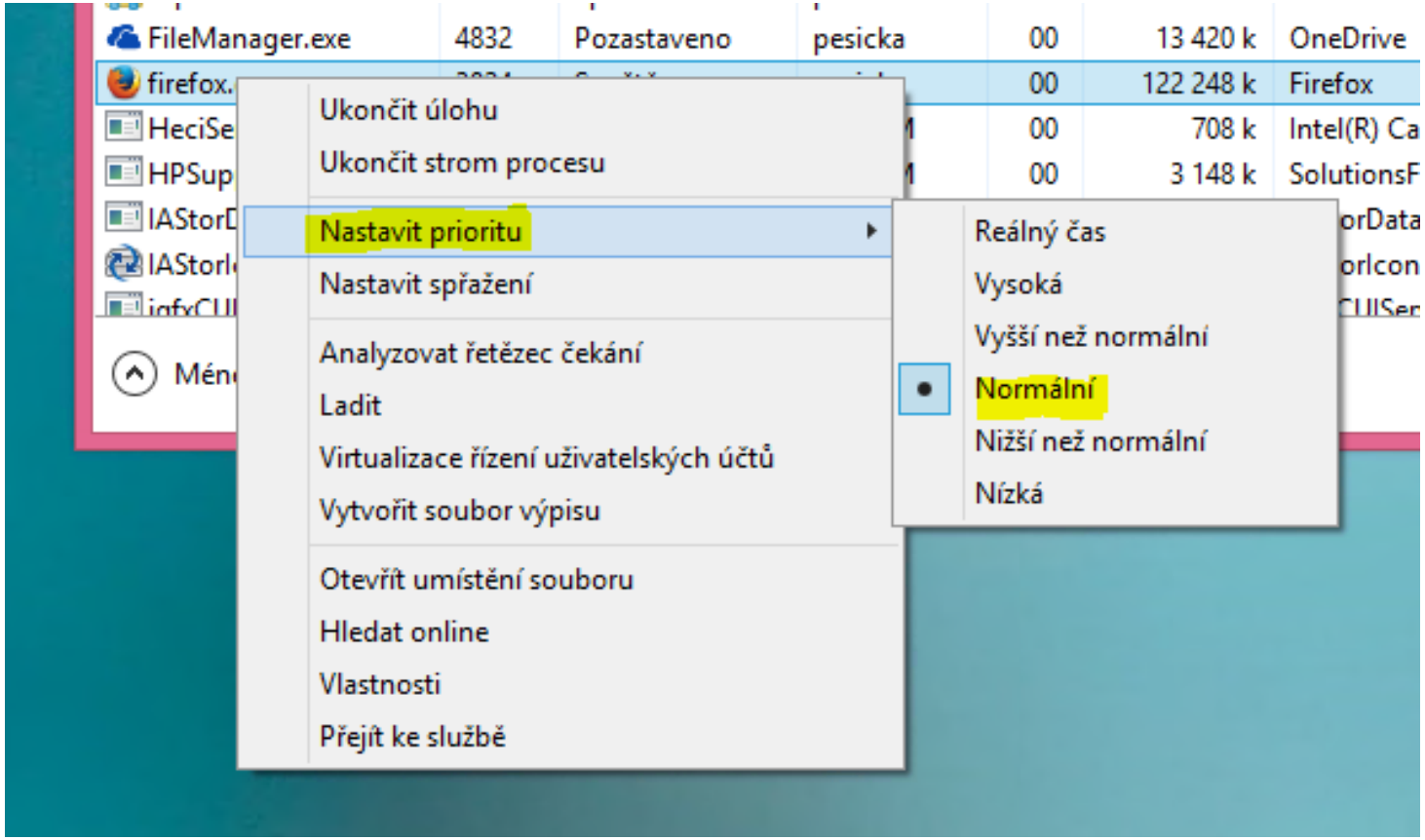
Windows – skupiny priorit

priorita	popis
0	Nulování stránek pro správce paměti
1 .. 15	Obyčejné procesy
16 .. 31	Systémové procesy

Process explorer ze sysinternals sady programů



Windows 10 správce úloh



Windows - priority

- 0 .. pokud není nic jiného na práci
- 1 .. 15 – obyčejné procesy
- aktuální priorita se mění
- bázová priorita – základní, může ji určit uživatel voláním *SetPriorityClass*
- procesy se plánují přísně podle priorit, tj. obyčejné pouze tehdy, pokud není žádný systémový proces připraven

Windows – změna akt. priority

- dokončení I/O zvyšuje prioritu o
  - 1 – disk, 2 – sériový port, 6 – klávesnice, 8 – zvuková karta
- vzbuzení po čekání na semafor, mutex zvýší o
  - 2 - pokud je proces na popředí
    (řídí okno, do kterého je posílán vstup z klávesnice)
  - 1 – jinak
- proces využil celé kvantum
  - sníží se priorita o 1
- proces neběžel dlouhou dobu
  - na 2 kvanta priorita zvýšena na 15 (zabránit inverzi priorit)

Windows – plánování na vláknech

proces A = 10 spustitelných vláken

proces B = 2 spustitelná vlákna

předpokládáme - stejná priorita

každé vlákno cca 1/12 CPU času

NENÍ 50% A, 50% B

nedělí rovnoměrně mezi procesy, ale mezi vlákna

Idle threads

- Jeden pro každý CPU

- „pod prioritou 0“
- Naplánován, když ve frontě připravených nikdo není

- umožní nastavit vhodný power management
  - volá HAL (hardware abstraction layer)
  - šetří systémové zdroje
  - NOP (no operation instrukce)
  - HALT (zastavení daného CPU jádra do interruptu)

Zero page thread

- Jeden pro celý systém
- Běží na úrovni priority 0
- Nuluje nepoužívané stránky paměti

Bezpečnostní opatření, když nějakému procesu přidělíme stránku paměti, aby v ní nezůstali data jiného procesu „z dřívějška“, aby se nedostal k informacím, ke kterým se dostat nemá

Kvantum, stretching

- kvantum stretching
  - none 2 tiky (základ)
  - middle 4 tiky (2x proti základu)
  - maximum 6 tiků (3x proti základu)

- na **desktopu** je defaultní kvantum 2 ticky (tj. 20 – 30 ms)
  - u vlákna na popředí – může být **stretching**
- na **serveru** je kvantum vždy 12 ticků
  - není kvantum stretching
- standardní clock tick je 10 nebo 15 ms

Zjištění hodnoty časovače

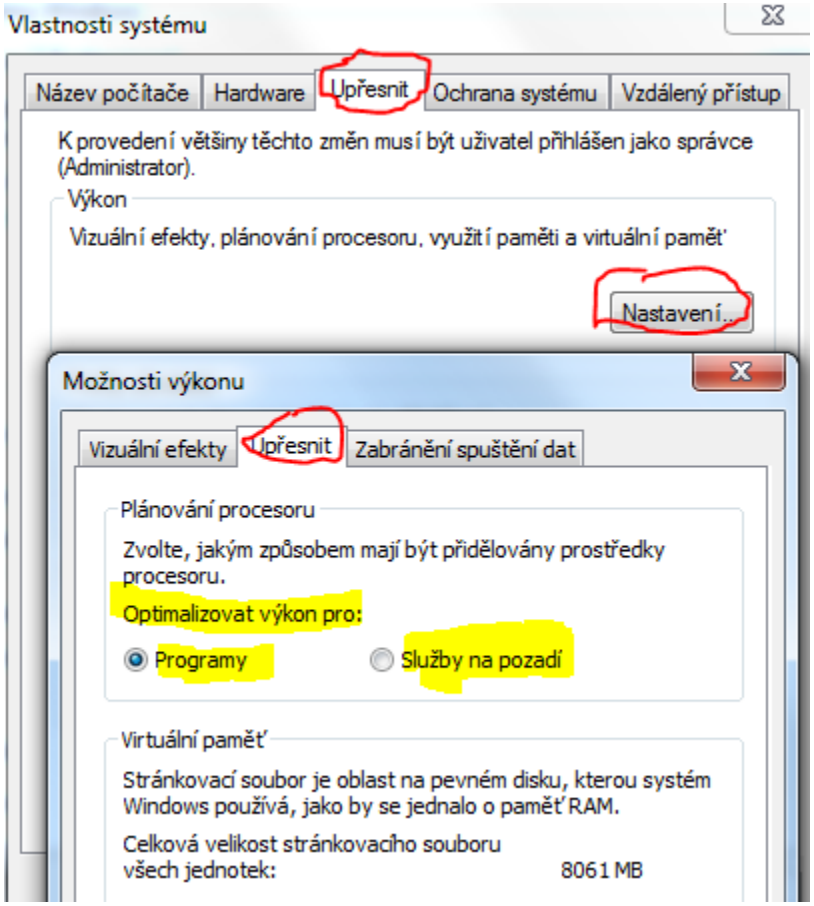
Program clockres ze sady Sysinternal

```
C:\!zos2012\prednasky\07\dalsi\SysinternalsSuite>clockres

ClockRes v2.0 - View the system clock resolution
Copyright (C) 2009 Mark Russinovich
SysInternals - www.sysinternals.com

Maximum timer interval: 15.600 ms
Minimum timer interval: 0.500 ms
Current timer interval: 10.000 ms
```

Windows 10
Systém – upřesnit – optimalizovat výkon pro



registrový klíč:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\PriorityControl

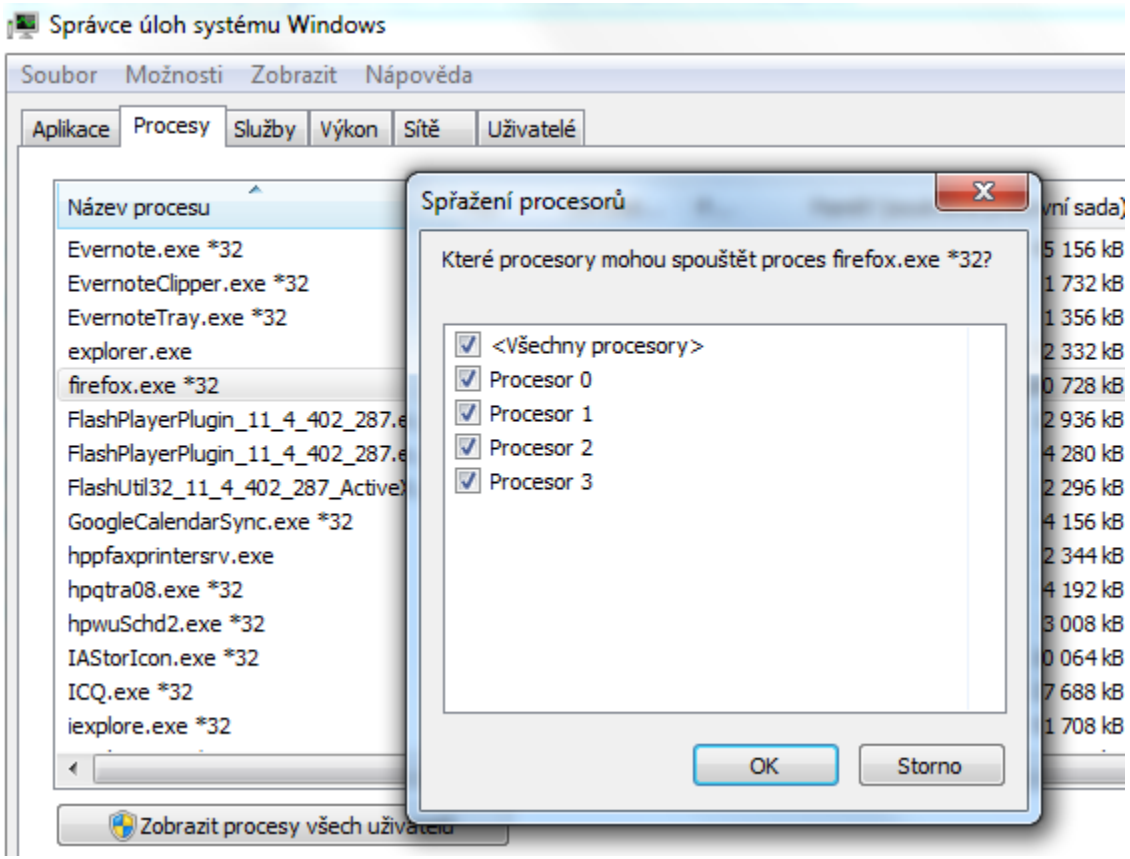
Win32PrioritySeparation 2
6bitů: XX XX XX
kvantum
- krátké, dlouhé
- proměnné, pevné
- navýšení pro procesy na popředí: 2x, 3x)
viz
http://technet.microsoft.com/library/Cc976120

Windows: vlákénka (fibers)

- kromě vláken i fibers
- fibers plánuje vlastní aplikace, nikoliv centrální plánovač jádra
- vytvoření fiberu: CreateFiber
- nepreemptivní plánování – odevzdá řízení jinému vlákenku přes SwitchToFiber

příklad:
http://msdn.microsoft.com/en-us/library/windows/desktop/ms686919%28v=vs.85%29.aspx

Windows - Afinita



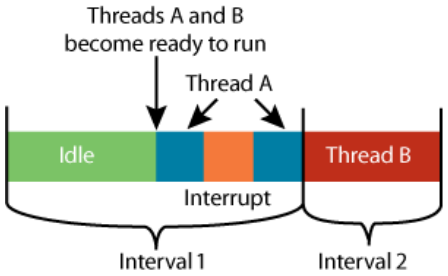
afinita
určení CPU (jádra CPU), na kterých může proces běžet

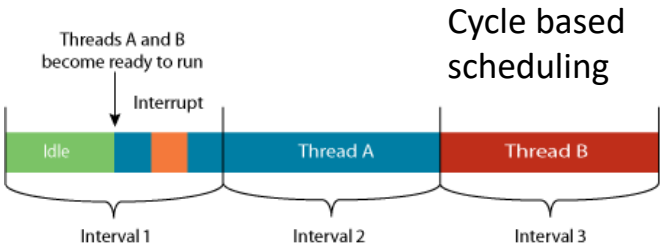
hard afinity
seznam povolených jader

soft afinity
vlákno přednostně plánováno na procesor, kde běželo naposledy

Windows Vista – modifikace

- plánovač používá cycle counter registr moderních CPU
  - Počítá, kolik CPU cyklů vlákno vykonalo (než jen používat intervalový časovač)





- prioritní plánování I/O fronty
  - Defragmentace tolik neovlivňuje procesy na popředí

- Viz: https://learn.microsoft.com/en-us/previous-versions/technet-magazine/cc162494(v=msdn.10)?redirectedfrom=MSDN

Další informace

http://cs.wikipedia.org/wiki/Plánování_procesů
http://en.wikipedia.org/wiki/Scheduling_%28computing%29

shrnutí – vhodné pro zopakování

http://cs.wikipedia.org/wiki/Preempce_%28informatika%29

http://cs.wikipedia.org/wiki/Změna_kontextu
http://cs.wikipedia.org/wiki/Mikrojádro

http://cs.wikipedia.org/wiki/Round-robin_scheduling

http://cs.wikipedia.org/wiki/Priority_scheduling

http://cs.wikipedia.org/wiki/Earliest_deadline_first (RTOS)

http://cs.wikipedia.org/wiki/Completely_Fair_Scheduler (CFS)