Virtuální adresový prostor.

Thursday, May 30, 2013 8:26 AM

ZOS

Viz přednáška 09_2012 ZOS. Program větší než dostupná fyzická paměť => mechanismus překrývání (overlays). Jako řešení tohoto problému se dnes nejčastěji používá právě virtuální paměť.

Překrývání (overlays)

Program – rozdělen na moduly tak, aby se daly postupně zavádět jednotlivé části do paměti, která je menší než celková paměť potřebná pro běh aplikace. (analogie s koberci)

Start – spuštěna část 0, při skončení zavede část 1…

Časté zavádění některých modulů

- Více překryvných modulů + data v paměti současně
- Moduly zaváděny dle potřeby (nejen 0, 1, 2…)
- Mechanismus odkládání (jako odkládání procesů)

Zavádění modulů zařizuje OS

Rozdělení programů i dat na části – navrhuje programátor (Např. vytváření DLL)

- Vliv rozdělení na výkonnost, komplikované
- Pro každou úlohu nové rozdělení

Virtuální paměť

- Potřebujeme rozsáhlý adresový prostor
- Ve skutečné paměti je pouze část adresového prostoru
  - Jinak by to bylo příliš drahé
- Zbytek může být odložen na disku

Virtuální adresy

- Fyzická paměť slouží jako cache virtuálního adresního prostoru procesů
- Proces – používá virtuální adresy
- Pokud požadovaná část VA prostoru je ve fyzické paměti, tak MMU převede VA => FA, přístup k paměti
- Pokud požadovaná část není ve fyzické paměti, OS si ji musí přečíst z disku I/O operace – přidělení CPU jinému procesu
- Většina systémů virtuální paměti používá stránkování

Mechanismus stránkování (paging)

- Program používá virtuální adresy
- Musíme rychle zjistit, zda je požadovaná adresa v paměti – pokud ano, převedeme VA na FA
- Musí být co nejrychlejší, děje se při každém přístupu do paměti

VAP – stránky (pages) pevné délky. Délka je obvykle mocnina 2, nejčastěji se jedná o 4KB, běžně 512B – 8KB

Fyzická paměť – rámce (page frames) stejné délky. **Rámec** může obsahovat **PRÁVĚ JEDNU stránku**. Na **známém místě v paměti** pak musí být **TABULKA STRÁNEK**. Ta poskytuje mapování virtuálních stránek na rámce.

Tabulka stránek

- Součástí PCB (tabulka procesů) – určuje, kde leží jeho tabulka stránek

- Velikost záznamu v tabulce stránek je 32 bitů kde vyšších 20 bitů určuje číslo stránky a nižších 12 bitů určuje offset
- Číslo rámce má pak 20 bitů (takže max. 2^20 stránek)

Výpočet adresy

Velikost stránky = 4096B.

Je dána VA(p1)=100. Určete FA. Tabulka stránek je:

Číslo stránky	Rámec
0	**1**
1	**2**
2	**--**
3	**0**

Máme-li více procesů, každý má svou vlastní tabulku stránek.

Virtuální adresu rozdělíme na číslo stránky a offset.

Str = VA div 4096 (dělení)

Offset = VA mod 4096 (zbytek po dělení)

Převod pomocí tabulky stránek – převedeme číslo stránky na číslo rámce

- tab_str[0] = 1 (pro stránku 0 je číslo rámce 1)
- tab_str[1] = 2
- tab_str[2] = -- stránka není namapována
- tab_str[3] = 0
- Pro VA = 100 je stránka 0, offset 100 => tedy rámec 1

Z čísla rámce a offsetu sestavíme fyzickou adresu:

- FA = rámec * 4096 + offset
- FA = 1 * 4096 + 100
- FA = 4196 v daném případě
- V reálném systému dělení znamená rozdělení na vyšší a nižší bity adresy (proto mocnina dvou)
- Nižší bity – offset
- Vyšší bity – číslo stránky

Výpadek stránky

- stránka není mapována
- Výpadek stránky způsobí výjimku, zachycena OS pomocí přerušení
- OS iniciuje zavádění stránky a přepne na jiný proces
- Po zavedení stránky OS upraví mapování (tabulku stránek)
- Proces může pokračovat
- Pokud daná stránka procesu není namapována na určitý rámec ve fyzické paměti a chceme k ní přistoupit, dojde k výpadku stránky – vyvolání přerušení operačního systému. Operační systém se postará o to, aby danou stránku zavedl do nějakého rámce ve fyzické paměti, nastavil mapování a poté může přístup proběhnout.
- Vnitřní fragmentace (část přidělené paměti je nevyužita), vnější fragmentace (souvislý paměťový prostor mapován do nesouvislých částí paměti)…
- Tabulka stránek procesu – mapuje číslo stránky na číslo fyzického rámce, obsahuje i další informace jako např. příznaky ochrany.
- Relokace = mapování VA na FA
- Ochrana – v tabulce stránek jsou pouze ty stránky, ke kterým má proces přístup. Při přepnutí na jiný proces přepne MMU na jinou tabulku stránek.
- Problémy

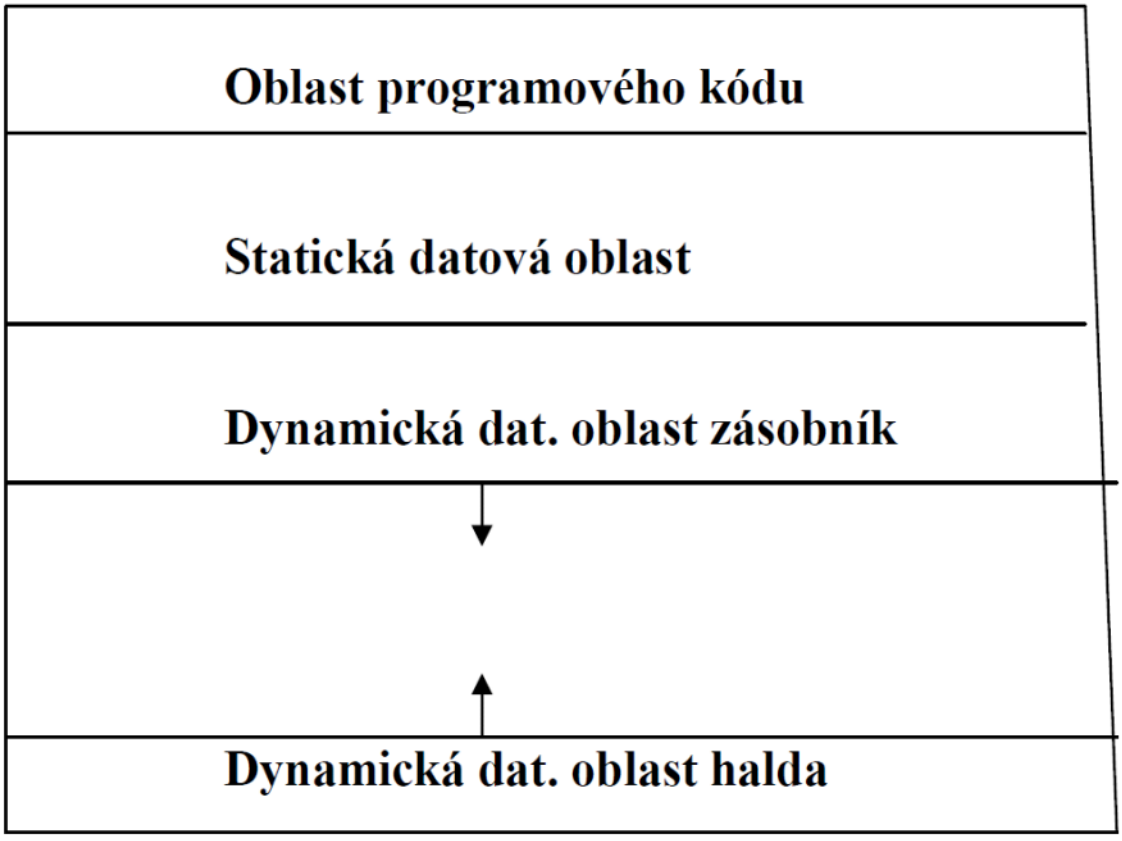
- velikost tabulky stránek – pomůže víceúrovňová struktura
- rychlost převodu VA -> FA – pomůže TLB (Transaction Look-aside Buffer)
- HW cache
- dosáhneme zpomalení jen 5 až 10%
- Přepnutí kontextu na jiný proces – problém (vymazání cache, …); než se TLB opět zaplní – pomalý přístup

Invertovaná tabulka stránek – řešení problému velikosti celé tabulky, obsahuje položky pro každý fyzický rámec. Omezený počet – dán velikostí RAM – VA je 64 bitů, 4KB stránky, 256MB RAM – 65536 položek. Forma položky: (id procesu, číslo stránky).

Při výpadku stránek nastupují algoritmy nahrazování stránek: FIFO, …
OPT, LRU, NRU, second chance, aging, clock

Each active process must have a Page Directory assigned to it. However, there is no need to allocate RAM for all Page Tables of a process at once; it is more efficient to allocate RAM for a Page Table only when the process effectively needs it.
 The physical address of the Page Directory in use is stored in a control register named **cr3** .
 The Directory field within the linear address determines the entry in the Page Directory that points to the proper Page Table. The address's Table field, in turn, determines the entry in the Page Table that contains the physical address of the page frame containing the page. The Offset field determines the relative position within the page frame (see Figure 2-7). Because it is 12 bits long, each page consists of 4096 bytes of data.

Z FJP o rozdělení paměti:



From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx>

Algoritmy nahrazování stránek

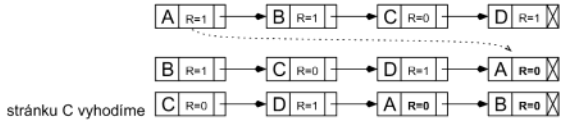
Thursday, May 30, 2013 10:35 AM

```
KIV/ZOS 2003/2004
Přednáška 10

Algoritmus Not-Recently-Used (NRU, NUR)
.......................................

* OS se snaží zjistit, které stránky se používají a nepoužívané vyhazovat
* systémy s VM poskytují HW podporu - stavové bity Referenced (R) a Dirty
  (zde M jako Modified) v tabulce stránek
* bity nastavované HW podle způsobu přístupu ke stránce
 - bit R - nastaven na 1 při čtení nebo zápisu do stránky
 - bit M - nastaven na 1 při zápisu do stránky; označuje, že
           se stránka má při vyhození zapsat na disk
 - po nastavení bitu zůstane na 1 dokud ho SW nenastaví zpět na 0

* algoritmus NRU:
 - na začátku mají všechny stránky nastaveny R=0, M=0
 - bit R je OS nastavován periodicky na 0 (např. při přerušení
   časovače) - tím se rozliší, které stránky byly referencovány
   v poslední době
 - OS rozlišuje 4 kategorie stránek:

   třída 0: R=0, M=0
   třída 1: R=0, M=1  ;; vznikne z třídy 3 po tiku, který nastaví R=0
   třída 2: R=1, M=0
   třída 3: R=1, M=1

 - algoritmus NRU vyhodí stránku z nejnižší neprázdné třídy, výběr mezi
   stránkami ve stejné třídě je náhodný

* algoritmus předpokládá, že je lepší vyhodit modifikovanou stránku
  která nebyla použita 1 tik než nemodifikovanou stránku, která se
  právě používá

* výhody algoritmu NRU:
 - jednoduchost, srozumitelnost
 - efektivně implementovatelný

* nevýhody:
 - výkonnost (jsou i lepší algoritmy)

Pokud by HW neměl bity R a M, můžeme je simulovat následujícím způsobem:

* při startu procesu se všechny jeho stránky označí jako nepřítomné
  v paměti
* při odkazu na stránku nastane výpadek stránky - OS interně nastaví
  R=1 a nastaví mapování v režimu READ ONLY
* při pokusu o zápis do stránky nastane výjimka - OS výjimku zachytí, nastaví
  M=1 a změní režim přístupu do stránky na READ/WRITE.

Algoritmy "Second Chance" a "Clock"
...................................

* algoritmy "Second Chance" a "Clock" vycházejí z algoritmu FIFO

Obchod: V algoritmu FIFO jsme vyhazovali zboží, které bylo zavedeno před
nejdelší dobou (bez ohledu na to, jestli ho někdo chce nebo ne). V algoritmu
"Second Chance" začneme evidovat, jestli zboží někdo v poslední době koupil
(pokud ano, prohlásíme ho za čerstvě zavedené zboží).

* jak modifikovat FIFO, abychom zabránili vyhození často používané stránky?
* algoritmus (Second Chance - vyhledání stránky pro vyhození):
 - podívat se na bit R nejstarší stránky
 - pokud R=0, stránka je nejstarší a zároveň nepoužívaná => vyhodíme
 - pokud R=1, nastavíme R na 0 a přesuneme na konec seznamu stránek
   (jako by byla nově zavedena)
```

Příklad:

Stránky uchováváme v seznamu uspořádaném podle času příchodu. V paměti budeme mít např. stránky A, B, C, D (viz obrázek); algoritmus "Second Chance" bude probíhat v následujících krocích:

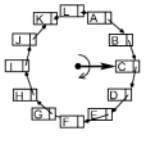
1. krok: Nejstarší je A; má R=1 => nastavíme R na 0 a přesuneme na konec seznamu;
2. krok: Druhá nejstarší je B; má také R=1 => nastavíme R na 0 a opět přesuneme na konec seznamu;
3. krok: Další nejstarší je C, R=0 => vyhodíme jí.



[]

* algoritmus "Second Chance" vyhledává nejstarší stránku, která nebyla referencována "v poslední době"
* pokud byly všechny referencovány, degeneruje na čisté FIFO:
 - postupně všem stránkám nastavíme bit R na 0 a přesuneme je na konec seznamu
 - dostaneme se opět na stránku A, tentokrát má R=0 => vyhodíme jí
 => algoritmus končí nejvýše po $počet_rámců + 1$ krocích

* algoritmus "Clock" = optimalizace datových struktur algoritmu Second Chance:
 - stránky udržovány v kruhovém seznamu
 - ukazatel na nejstarší stránku ("ručička hodin")

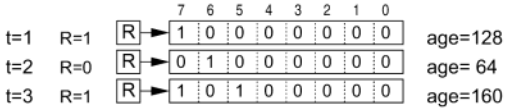


* výpadek stránky -> vyhledáváme stránku k vyhození
 - stránka kam ukazuje "ručička":
  . má-li R=0, stránku vyhodíme a ručičku posuneme o 1 pozici
  . má-li R=1, nastavíme R na 0, ručičku posuneme o 1 pozici; opakujeme dokud nenalezneme stránku s R=0
* od algoritmu Second Chance se liší pouze implementací
* varianty algoritmu Clock používají např. systémy BSD UNIX

Softwarová aproximace LRU
.....

* algoritmus LRU vždy vyhazuje nejdéle nepoužitou stránku

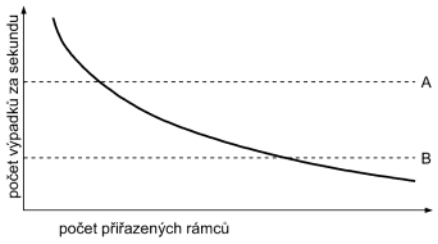
* algoritmus Aging:
 - každá položka v tabulce stránek má pole "stáří" age, N bitů (např. N=8)
 - na počátku age=0
 - při každém přerušení časovače pro každou stránku:
  . posun pole "stáří" o 1 bit vpravo
  . zleva se přidá hodnota bitu R
  . nastavení R na 0



```
* to odpovídá následujícímu kódu (v Turbo Pascalu):

    age := age shr 1;           { posun o 1 bit vpravo }
    age := age or (R shl N-1);  { zleva se přidá hodnota bitu R }
    R := 0;                     { nastavení R na 0 }

* při výpadku stránky se vyhodí stránka, jejíž pole age je má nejnižší
  hodnotu

Dva rozdíly od LRU:

* několik stránek může mít stejnou hodnotu pole age a nevíme která stránka
  byla odkazovaná dříve (u LRU to víme vždy)
 - rozlišení je "hrubé" (= po tricích časovače)
* pole age se může snížit na 0 - nevíme, zda stránka byla naposledy
  odkazovaná před 9 nebo před 1000 tiky časovače
 - uchovává pouze omezenou historii
 - v praxi není problém: pokud je tik časovače po 20 ms a N=8, nebyla
   odkazována 160 ms => nejspíš není tak důležitá, můžeme jí vyhodit
* pokud se musíme rozhodovat mezi dvěma stránkami se stejnou hodnotou
  age, vybíráme náhodně

Shrnutí algoritmů pro nahrazování stránek
.........................................

* optimální algoritmus (MIN čili OPT)
 - není implementovatelný, ale je užitečný pro srovnání
* FIFO
 - vyhazuje nejstarší stránku
 - jednoduchý, ale je chopen vyhodit důležité stránky a trpí Beladyho
   anomálií
* LRU (Least Recently Used)
 - výborný
 - implementace vyžaduje speciální HW, proto prakticky používán zřídka
* NRU (Not Recently Used)
 - rozděluje stránky do 4 kategorií podle bitů R a M
 - efektivita nic moc, přesto občas používán
* "Second chance" a "Clock"
 - vycházejí z FIFO, před vyhozením zkontrolují zda se stránka používala
 - mnohem lepší než FIFO
 - používané algoritmy (např. některé varianty UNIXu)
* Aging
 - dobře aproximuje LRU => efektivní
 - často prakticky používaný algoritmus


Ostatní problémy stránkované virtuální paměti
---------------------------------------------

Alokace fyzických rámců
.......................

* 2 základní metody - globální a lokální alokace:
 - globální alokace - pro vyhození se uvažují všechny rámce
 - lokální alokace - pro vyhození se uvažují pouze rámce alokované
   procesem (tj. obsahující stránky procesu, jehož výpadek stránky se
   obsluhuje)
  . počet stránek alokovaných pro proces se nemění
  . program se vzhledem ke stránkování chová přibližně stejně při
    každém běhu
 - u globální alokace vybírá ze všech rámců
  . lepší průchodnost systému - proto globální alokace častější
  . na běh procesu má vliv chování ostatních procesů

* při lokální alokaci - kolik rámců dát kterému procesu?
 - nejjednodušší - všem procesům dáme stejně, ale potřeby procesů mohou
   být různé
 - proporcionální - dáme každému proporcionální díl podle velikosti procesu
```

- nejlepší metoda - podle frekvence výpadků stránek (Page Fault Frequency, PFF)

Pro většinu rozumných algoritmů se PFF snižuje s množstvím přidělených rámců:



PFF se snažíme udržet v rozumných mezích:

* pokud je PFF větší než A, přidáme procesu rámce
* pokud je PFF menší než B, proces má asi příliš paměti, rámce mu mohou být odebrány

Zahlcení
.....

* proces pro svůj rozumný běh potřebuje pracovní množinu stránek
* pokud se pracovní množiny stránek aktivních procesů nevejdou do paměti, nastane tzv. zahlcení (angl. trashing)

* jak to vypadá:
 - v procesu nastane výpadek stránky
 - paměť je plná (není volný rámec) => je třeba některou stránku vyhodit
 - stránka pro vyhození bude ale brzy zapotřebí, takže se bude muset vyhodit jiná používaná stránka
 - z uživatelského hlediska se to projeví tak, že systém pracuje intenzivně s diskem a běh procesů řádově zpomalí (stráví víc času stránkováním než během)

* řešení - při zahlcení snížit úroveň multiprogramování (zahlcení lze detekovat pomocí PFF)

Zhodnocení mechanismu virtuální paměti
.....

Virtuální paměť má podstatné výhody oproti předchozím mechanismům:

* rozsah virtuální paměti (např. 2 GB pro proces - NT nebo Linux na i386)
 - adresový prostor úlohy není omezen velikostí fyzické paměti
 - multiprogramování (= počet procesů) není zásadně omezeno rozsahem fyzické paměti
* efektivnější využití fyzické paměti
 - není vnější fragmentace paměti
 - nepoužívané části adresového prostoru úlohy nemusejí být ve fyzické paměti

Nevýhody:

* režie při převodu virtuálních adres na fyzické adresy
* režie procesoru (údržba tabulek stránek a rámců, výběr stránek pro vyhození, plánování I/O)
* režie I/O při čtení/zápisu stránky
* paměťový prostor pro tabulky stránek
* vnitřní fragmentace

```
Segmentace
----------

* dosud diskutovaná virtuální paměť byla jednorozměrná:
  od adresy 0 do nějaké maximální virtuální adresy
* pro mnoho programů by bylo výhodnější mít víc samostatných virtuálních
  adresových prostorů
* příklad - mám několik tabulek a chci, aby jejich velikost mohla růst
=> paměť nejlépe mnoho nezávislých adresových prostorů = segmenty

* segment = logické seskupení informací
* každý segment lineární posloupnost adres, začínající od adresy 0
* programátor o segmentech ví, používá explicitně (adresuje konkrétní
  segment)
* příklad - překladač Pascalu může používat samostatné segmenty pro:
  - kód přeloženého programu
  - globální proměnné
  - hromadu
  - zásobník návratových adres
  - je možné i jemnější dělení (segment pro každou proceduru/fci)
* často se používá také pro implementaci:
  - přístupu k souborům (1 soubor = 1 segment)
    . není třeba open, read...
  - sdílených knihoven:
    . dnešní programy využívají rozsáhlé knihovny - knihovnu potřebuje prakticky každý
      program
    . myšlenka vložit knihovnu do segmentu a sdílet mezi více programy

* každý segment je logická entita - má smysl, aby měl samostatnou ochranu

Čistá segmentace
................

* každý odkaz do paměti se skládá z dvojice: (selektor, offset)
 - selektor: číslo segmentu, určuje segment
 - offset: relativní adresa v rámci segmentu
* technické prostředky musí přemapovat dvojici (selektor, offset) na fyzickou
  (= lineární) adresu
* k tomu slouží tabulka segmentů, každá položka tabulky obsahuje:
  - počáteční adresu segmentu (bázi)
  - rozsah segmentu (limit)
  - příznaky ochrany segmentu (nejčastěji čtení, zápis, provádění = rwx)
* postup při převodu na fyzickou adresu:
  - PCB obsahuje odkaz na tabulku segmentů procesu
  - odkaz do paměti má tvar (selektor, offset)
    např. v důsledku instrukce LD R, selektor:offset
  - selektor = index do tabulky segmentů
  - zkontroluje se zda je offset<limit; ne => chyba porušení ochrany paměti
  - zkontroluje se, zda dovolen způsob použití; ne => chyba porušení
    ochrany paměti
  - adresa = báze + offset
* často možnost sdílet segment mezi více procesy

* mnoho věcí podobných jako přidělování paměti po sekcích, ale rozdíl:
  - po sekcích - pro procesy
  - segmenty - pro části procesu
* stejné problémy jako přidělování paměti po sekcích: externí
  fragmentace paměti, mohou zůstávat malé díry (tj. dále již prakticky
  nepoužitelné)

Segmentace na žádost
....................

* segment může být zavedený v paměti nebo odložený na disku
* pokus o adresování segmentu, který není v paměti způsobí výpadek segmentu
* OS zavede segment do paměti
* není-li místo, je některý jiný segment odložen na disk

* HW podpora - v tabulce segmentů bity:
```