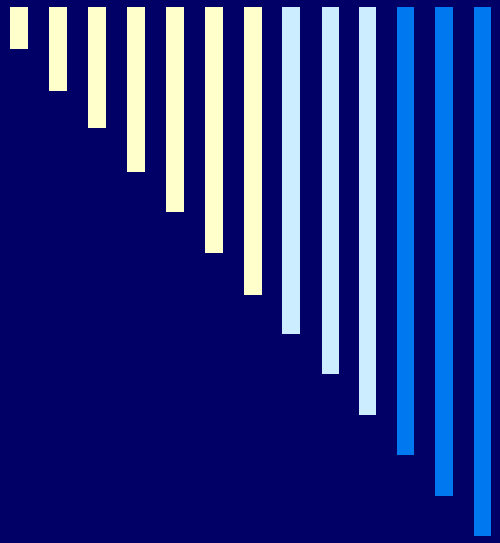
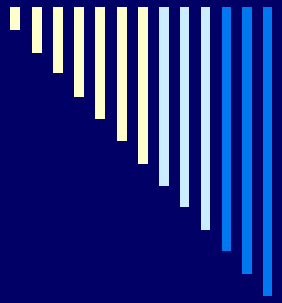


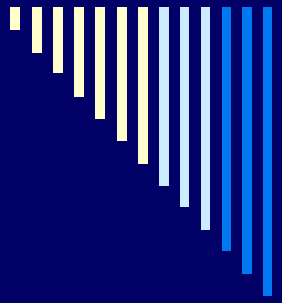
10. Memory management, I/O

ZOS 2006, L. Pešička



Informace – 2. zápočtový test

- Látka z přednášek do 9. týdne
  - Požadované znalosti
    - Meziprocesová komunikace
    - Synchronizace
    - Základ MM
    - Řešení konkrétních příkladů v BACI
- **12.12.2006** (úterý) od **18:30** v **EP130**

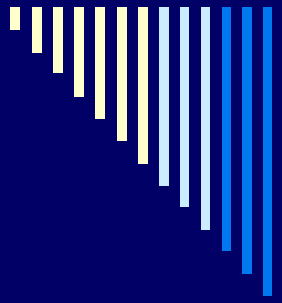


Obsah

- MIN / OPT
- LRU
- NRU
- Second Chance, Clock
- Aging

- Segmentování

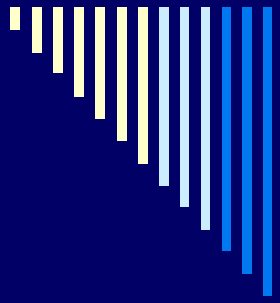
- I/O



Algoritmus MIN / OPT

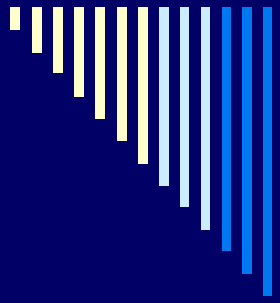
- optimální – nejmenší možný výpadek stránek
- Vyhodíme zboží, které nejdelší dobu nikdo nebude požadovat.

- stránka označena počtem instrukcí, po který se k ní nebude přistupovat
- p[0] = 5, p[1] = 20, p[3] = 100
- výpadek stránky – vybere s nejvyšším označením
- vybere se stránka, která bude zapotřebí nejpozději v budoucnosti



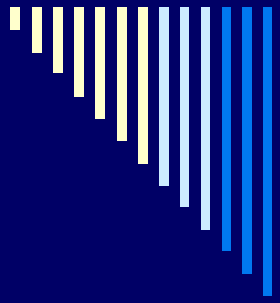
MIN / OPT

- není realizovatelný (křišťálová kole)
  - jak zjistit dopředu která stránka bude potřeba?
- algoritmus pro srovnání s realizovatelnými
- běh programu v simulátoru
  - uchovávají se odkazy na stránky
  - spočte se počet výpadků pro MIN/OPT
  - srovnání



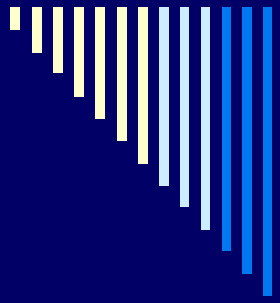
Least Recently Used (LRU, LUR)

- nejdéle nepoužitá (pohled do minulosti)
- princip lokality
  - stránky používané v posledních instrukcích se budou pravděpodobně používat i v následujících
  - pokud se stránka dlouho nepoužívala, pravděpodobně nebude brzy zapotřebí
- Vyhazovat zboží, na kterém je v prodejně nejvíce prachu = nejdéle nebylo požadováno



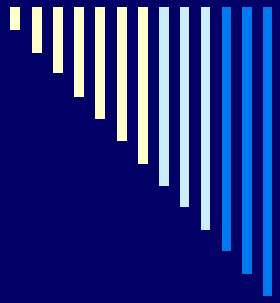
LRU

- obtížná implementace
- sw řešení (není použitelné)
  - seznam stránek v pořadí referencí
  - výpadek – vyhození stránky ze začátku seznamu
  - zpomalení cca 10x, nutná podpora hw



LRU – hw řešení - čítač

- hw řešení – čítač
  - MMU obsahuje čítač (64bit), při každém přístupu do paměti zvětšen
  - každá položka v tabulce stránek – pole pro uložení čítače
  - odkaz do paměti
    - obsah čítače se zapíše do položky pro odkazovanou stránku
  - výpadek stránky
    - vyhodí se stránka s nejnižším číslem



LRU – HW řešení - matice

- MMU udržuje matici n * n bitů
  - n – počet rámců
- všechny prvky 0
- odkaz na stránku odpovídající k-tému rámci
  - všechny bity k-tého řádku matice na 1
  - všechny bity k-tého sloupce matice na 0
- řádek s nejnižší binární hodnotou
  - nejdéle nepoužitá stránka

LRU – matice - příklad

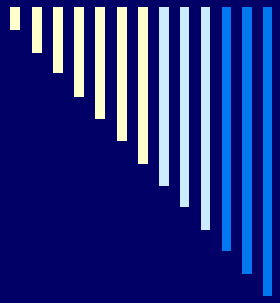
reference v pořadí: 3 2 1 0

	0	1	2	3
0.	0	0	0	0
1.	0	0	0	0
2.	0	0	0	0
3.	1	1	1	0

	0	1	2	3
0.	0	0	0	0
1.	0	0	0	0
2.	1	1	0	1
3.	1	1	0	0

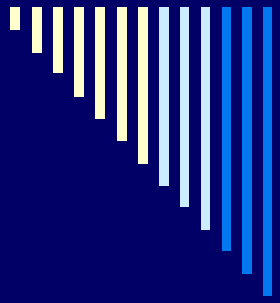
	0	1	2	3
0.	0	0	0	0
1.	1	0	1	1
2.	1	0	0	1
3.	1	0	0	0

	0	1	2	3
0.	0	1	1	1
1.	0	0	1	1
2.	0	0	0	1
3.	0	0	0	0



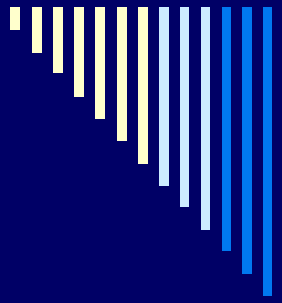
LRU - vlastnosti

- výhody
  - z časově založených (realizovatelných) nejlepší
  - Beladyho anomálie nemůže nastat
- nevýhody
  - každý odkaz na stránku – aktualizace záznamu (zpomalení)
    - položka v tab. stránek
    - řádek a sloupec v matici
- LRU se pro stránkovanou virtuální paměť moc nepoužívá
- LRU např. pro blokovou cache souborů



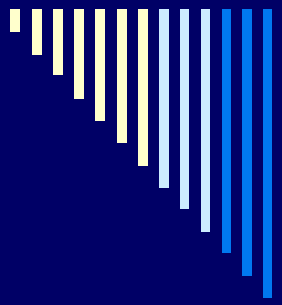
Not-Recently-Used (NRU, NUR)

- snaha vyhazovat nepoužívané stránky
- HW podpora u systémů s VM
  - stavové bity Referenced (R) a Dirty (M = modified)
  - v tabulce stránek
- bity nastavované HW dle způsobu přístupu ke stránce
- bit R – nastaven na 1 při čtení nebo zápisu do stránky
- bit M – na 1 při zápisu do stránky
  - stránku je třeba při vyhození zapsat na disk
- bit zůstane na 1, dokud ho SW nenastaví zpět na 0



algoritmus NRU

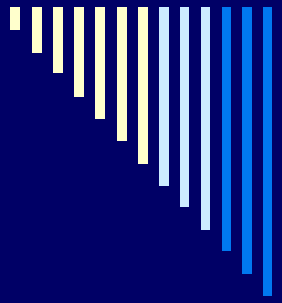
- začátek – všechny stránky R=0, M=0
- bit R nastavován OS periodicky na 0 (přerušení čas.)
  - odliší stránky referencované **v poslední době**
- 4 kategorie stránek
- *třída 0: R = 0, M = 0*
- *třída 1: R = 0, M = 1* -- z třídy 3 po ticku nulujícím R
- *třída 2: R = 1, M = 0*
- *třída 3: R = 1, M = 1*
- NRU vyhodí stránku z nejnižší neprázdné třídy
- výběr mezi stránkami ve stejné třídě je náhodný



NRU

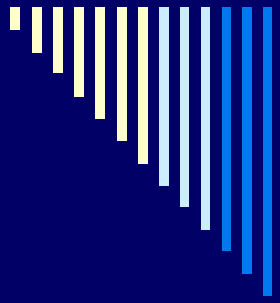
- NRU předpokládá – lepší je vyhodit modifikovanou stránku, která nebyla použita 1 tik, než nemodifikovanou stránku, která se právě používá

- výhody
  - jednoduchost, srozumitelnost
  - efektivně implementovaný
- nevýhody
  - výkonnost (jsou i lepší algoritmy)



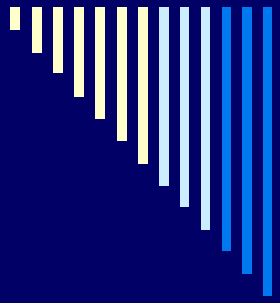
Náhrada bitů R a M - simulace

- start procesu – všechny stránky jako nepřítomné v paměti
- odkaz na stránku – výpadek
  - OS interně nastaví R=1
  - nastaví mapování v READ ONLY režimu
- pokus o zápis do stránky – výjimka
  - OS zachytí a nastaví M=1,
  - změní přístup na READ WRITE



Algoritmy Second Chance a Clock

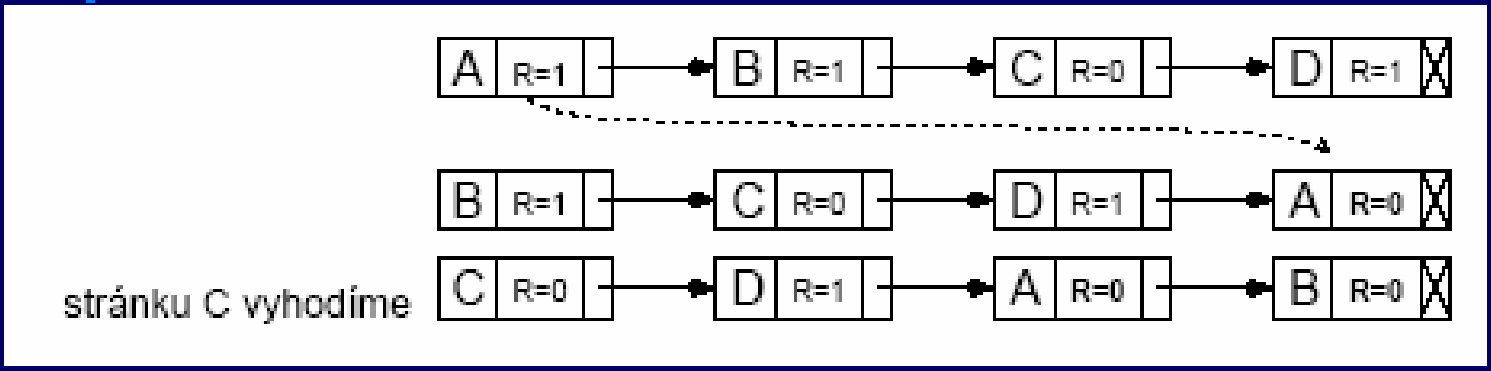
- vycházejí z FIFO
  - FIFO – obchod vyhazuje zboží zavedené před nejdelší dobou, ať už ho někdo chce nebo ne
  - Second Chance – evidovat, jestli zboží v poslední době někdo koupil (ano – prohlásíme za čerstvé zboží)
- modifikace FIFO – zabránit vyhození často používané



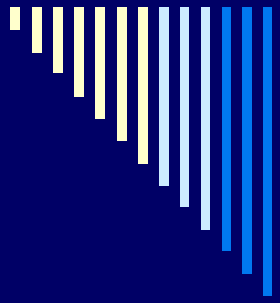
Second Chance

- algoritmus Second Chance
- dle bitu R nejstarší stránky
  - R = 0 ... stránka je nejstarší, nepoužívaná – vyhodíme
  - R = 1 ... nastavíme R=0, přesuneme na konec seznamu stránek (jako by byla nově zavedena)

Příklad Second Chance



1. Krok – nejstarší je A, má R = 1 – nastavíme R na 0 a přesuneme na konec seznamu
2. Druhá nejstarší je B, má R = 1 – nastavíme R na 0 a opět přesuneme na konec seznamu
3. Další nejstarší je C, R = 0 – vyhodíme ji

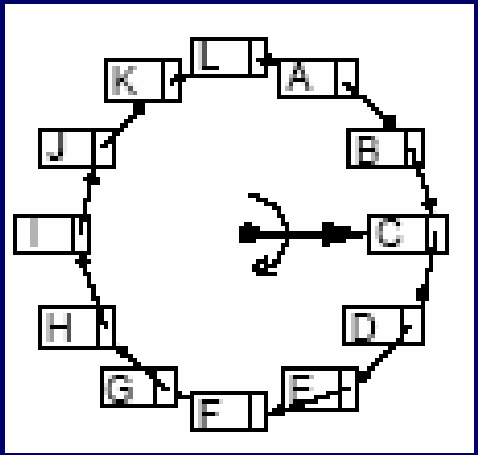


Second Chance

- SC vyhledá nejstarší stránku, která nebyla referencována v poslední době
- Pokud všechny referencovány – čisté FIFO
  - Všem se postupně nastaví R na 0 a na konec seznamu
  - Dostaneme se opět na A, nyní s R = 0, vyhodíme ji
- 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“



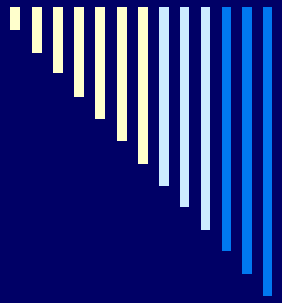
Výpadek stránky – najít stránku k vyhození

Stránka kam ukazuje ručička

- má-li R=0, stránku vyhodíme a ručičku posuneme o jednu pozici
- má-li R=1, nastavíme R na 0, ručičku posuneme o 1 pozici, opakování,..

Od SC se liší pouze implementací

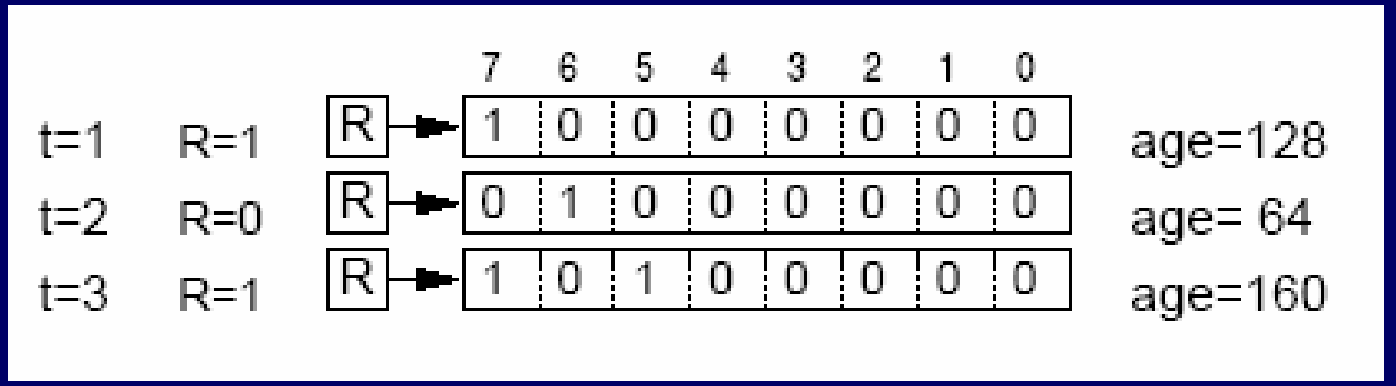
Varianty Clock používají např. BSD UNIX



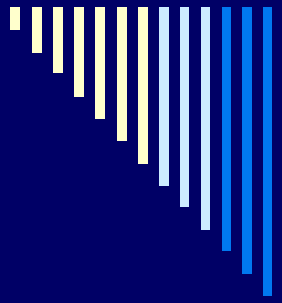
SW aproximace LRU - Aging

- LRU vyhazuje vždy nejdéle nepoužitou stránku
- Algoritmus Aging
  - Každá položka tabulky stránek – pole stáří (age), N bitů (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
- Při výpadku se vyhodí stránka, jejíž pole age má nejnižší hodnotu

Aging

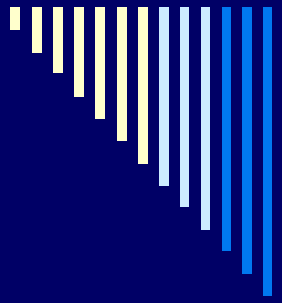


Age := age shr 1;	posun o 1 bit vpravo
Age := age or (R shl N-1);	zleva se přidá hodnota bitu R
R := 0;	nastavení R na 0



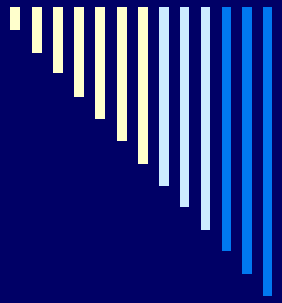
Aging x LRU

- Několik stránek může mít stejnou hodnotu age a nevíme, která byla odkazovaná dříve (u LRU jasné vždy) – hrubé rozlišení (po ticích časovače)
- Age se může snížit na 0
  - ? Odkazovaná před 9ti nebo 1000ci tiky časovače
    - Uchovává pouze omezenou historii
    - V praxi není problém – tik 20ms, N=8, nebyla odkazována 160ms – nejspíše není tak důležitá, můžeme jí vyhodit
- 2 stránky se stejnou hodnotou age – vybereme náhodně



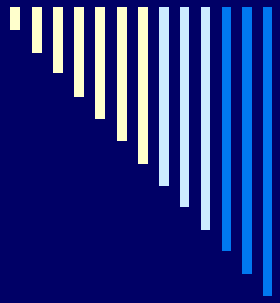
Shrnutí algoritmů

- Optimální algoritmus (MIN čili OPT)
  - Nelze implementovat, vhodný pro srovnání
- FIFO
  - Vyhazuje nejstarší stránku
  - Jednoduchý, ale je schopen vyhodit důležité stránky
  - Trpí Beladyho anomálií
- LRU (Least Recently Used)
  - Výborný
  - Implementace vyžaduje spec. hardware, proto používán zřídka



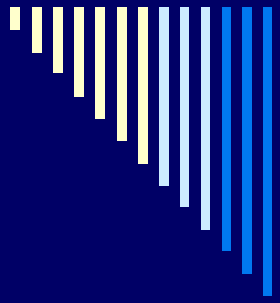
Shrnutí algoritmů II.

- NRU (Not Recently Used)
  - Rozděluje stránky do 4 kategorií dle bitů R a M
  - Efektivita není příliš velká, přesto používán
- Second Chance a Clock
  - Vycházejí z FIFO, před vyhození zkotrolují, zda se stránka používala
  - Mnohem lepší než FIFO
  - Používané algoritmy (některé varianty UNIXu)
- Aging
  - Dobře aproximuje LRU – efektivní
  - Často prakticky používaný algoritmus



Ostatní problémy stránkované VP

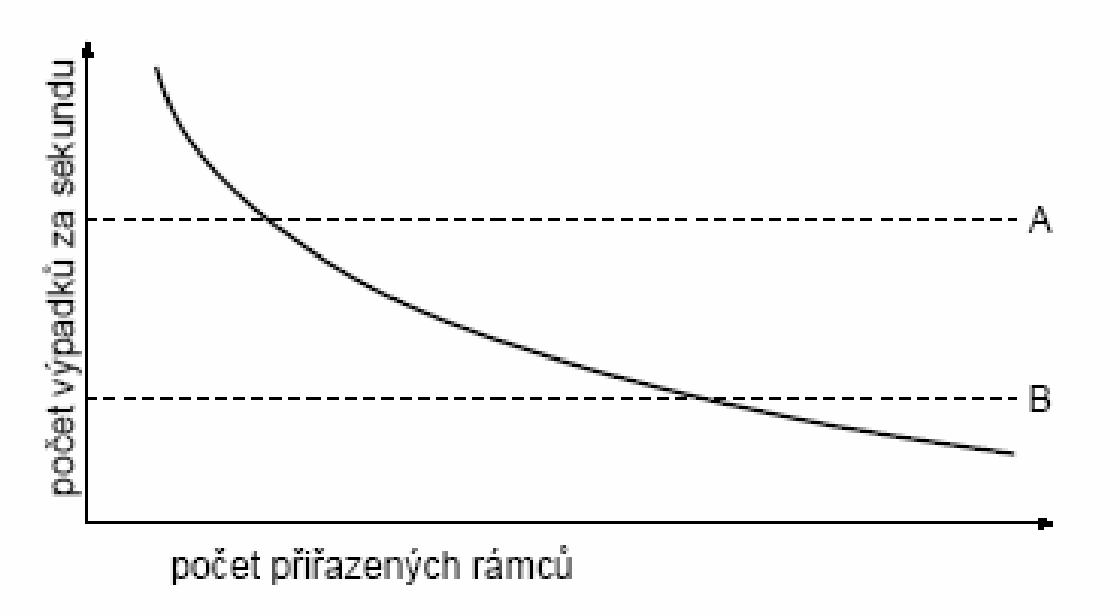
- Alokace fyzických rámců
  - Globální a lokální alokace
  - Globální – pro vyhození se uvažují všechny rámce
    - Lepší průchodnost systému – častější
    - Na běh procesu má vliv chování ostatních procesů
  - Lokální – uvažují se pouze rámce alokované procesem (tj. obsahující stránky procesu, jehož výpadek stránky je obsluhuje)
    - Počet stránek alokovaných pro proces se nemění
    - Program se vzhledem k stránkování chová přibližně stejně při každém běhu



Lokální alokace

- Kolik rámců dát každému procesu?
- Nejjednodušší – všem procesům dát stejně
  - Ale potřeby procesů jsou různé
- Proprocionální – každému proporcionální díl podle velikosti procesu
- Nejlepší – podle frekvence výpadků stránek (Page Fault Frequency, PFF)
  - Pro většinu rozumných algoritmů se PFF snižuje s množstvím přidělených rámců

Page Fault Frequency (PFF)



PFF udržet v roz. mezích

PFF > A

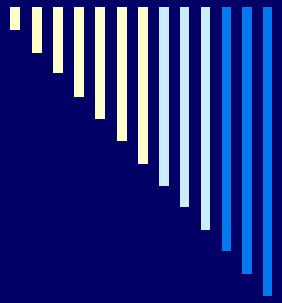
přidáme procesu rámce

PFF < B

proces má asi příliš paměti

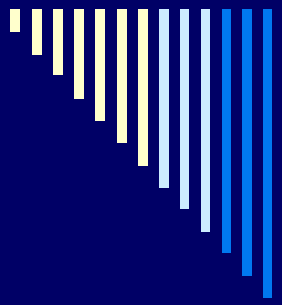
rámce mu mohou být

odebrány



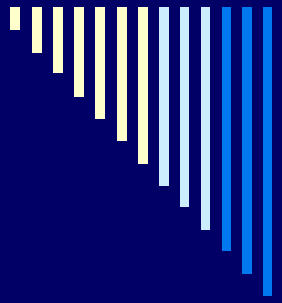
Zahlcení

- Proces pro svůj rozumný běh potřebuje pracovní množinu stránek
- Pokus se pracovní množiny stránek aktivních procesů nevejdou do paměti, nastane zahlcení (trashing)
- Zahlcení
  - V procesu nastane výpadek stránky
  - Paměť je plná (není volný rámec) – je třeba nějakou stránku vyhodit, stránka pro vyhození bude ale brzo zapotřebí, bude se muset vyhodit jiná používaná stránka …
- Uživatel – systém intenzivně pracuje s diskem a běh procesů se řádově zpomalí (víc času stránkování než běh)
- Řešení – při zahlcení snížit úroveň multiprogramování (zahlcení lze detekovat pomocí PFF)



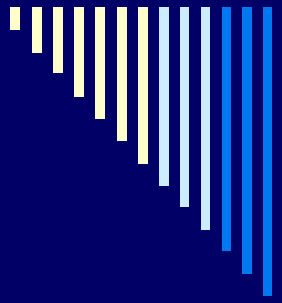
Zhodnocení mechanismu virtuální paměti - výhody

- Rozsah virtuální paměti
  - (2GB pro proces – NT nebo Linux na i386)
  - Adresový prostor úlohy není omezen velikostí fyzické paměti
  - Multiprogramování – není omezeno rozsahem fyz. paměti
- Efektivnější využití fyzické paměti
  - Není vnější fragmentace paměti
  - Nepoužívané části adresního prostoru úlohy nemusejí být ve fyzické paměti



Mechanismus VP - nevýhody

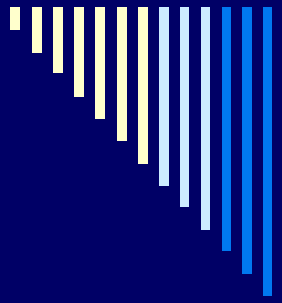
- Režie při převodu virt. adres na fyzické adresy
- Režie procesoru
  - Údržba tabulek stránek a rámců
  - výběr stránky pro vyhození, plánování I/O
- Režie I/O při čtení/zápisu stránky
- Paměťový prostor pro tabulky stránek
- Vnitřní fragmentace



Segmentace

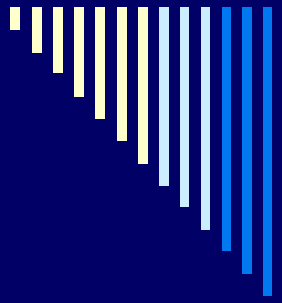
- Dosud diskutovaná VP – jednorozměrná
  - Od adresy 0 do nějaké maximální virtuální adresy
- Často výhodnější – více samostatných virtuálních adresových prostorů
- Př. – máme několik tabulek a chceme, aby jejich velikost mohla růst

- Paměť nejlépe mnoho nezávislých adresových prostorů - segmenty



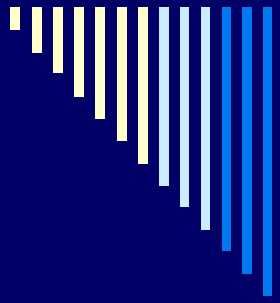
Segmentace

- Segment – logické seskupení informací
- Každý segment – lineární posloupnost adres od 0
- Programátor o segmentech ví, používá je explicitně (adresuje konkrétní segment)
- Např. překladač jazyka – samostatné segmenty pro
  - Kód přeloženého programu
  - Globální proměnné
  - Hromada
  - Zásobník návratových adres
  - Možné i jemnější dělení – segment pro každou funkci



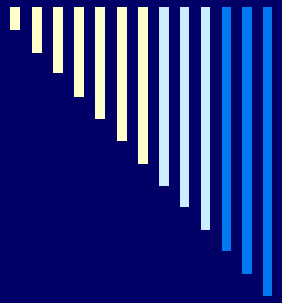
Segmentace

- Lze použít pro implementaci
  - Přístup k souborům
    - 1 soubor = 1 segment
    - Není třeba open, read ..
  - Sdílené knihovny
    - Programy využívají rozsáhlé knihovny
    - Vložit knihovnu do segmentu a sdílet mezi více programy
- Každý segment – logická entita – má smysl, aby měl samostatnou ochranu



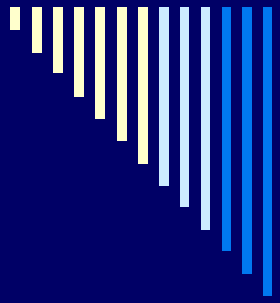
Čistá segmentace

- Každý odkaz do paměti – dvojice (selektor, offset)
  - Selektor – číslo segmentu, určuje segment
  - Offset – relativní adresa v rámci segmentu
- Technické prostředky musí umět přemapovat dvojici (selektor, offset) na fyzickou adresu – lineární
- Tabulka segmentů – každá položka má
  - Počáteční adresa segmentu (báze)
  - Rozsah segmentu (limit)
  - Příznaky ochrany segmentu (čtení,zápis, provádění – rwx)



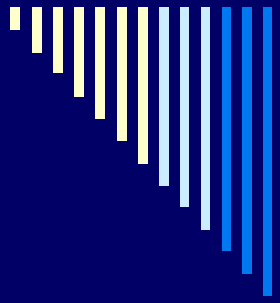
Převod na fyzickou adresu

- PCB obsahuje odkaz na tabulku segmentů procesu
- Odkaz do paměti má tvar (selektor, offset)
- Např. v důsledku instrukce LD R, sel:offset
- Selektor – index do tabulky segmentů
- Kontrola offset < limit, ne – porušení ochrany paměti
- Kontrola zda dovolený způsob použití; ne – chyba
- Adresa = báze + offset
- Často možnost sdílet segment mezi více procesy



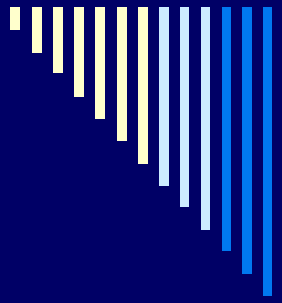
Segmentace

- 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



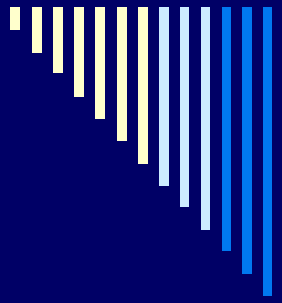
Segmentace na žádost

- Segment – zavedený v paměti nebo odložený na disku
- Adresování segmentu co není v paměti – výpadek segmentu – zavede do paměti – není-li místo – jiný segment odložen na disk
- HW podpora – bity v tabulce segmentů
  - Bit segment je zaveden v paměti (Present / absent)
  - Bit referenced
- Používal např. systém OS/2 pro i80286 – pro výběr segmentu k odložení algoritmus Second Chance



Segmentace se stránkováním

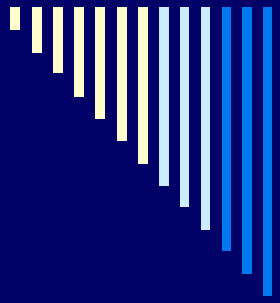
- velké segmenty – nepraktické celé udržovat v paměti
- Myšlenka stránkování segmentů
  - V paměti pouze potřebné stránky
- Implementace – např. každý segment vlastní tabulka stránek



Ukázka – Intel Pentium

- segmentace
- stránkování
- segmentace se stránkováním

- tabulka LDT (Local Descriptor Table)
  - každý proces má svojí
  - segmenty lokální pro program (kód,data,zásobník)
- tabulka GDT (Global Descriptor Table)
  - pouze jedna, sdílená všemi procesy
  - systémové segmenty, včetně OS



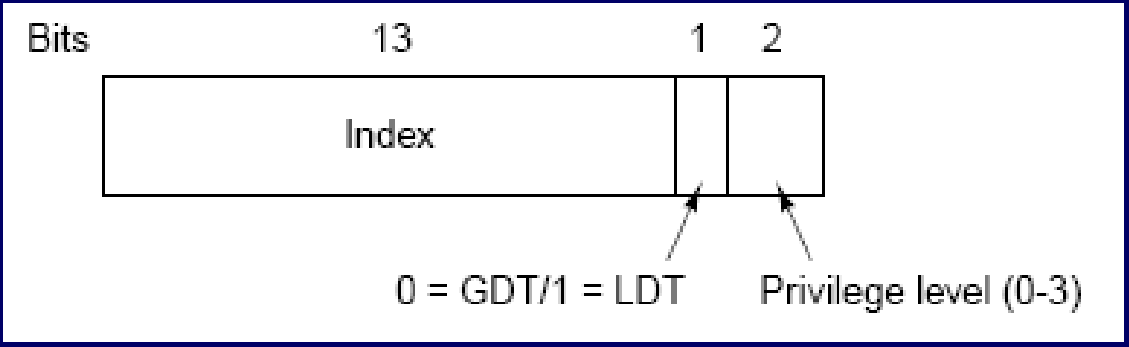
Ukázka – CPU Pentium

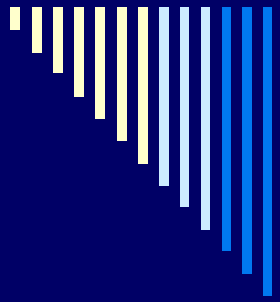
- Pentium má 6 segmentových registrů, např.
  - CS (Code Segment)
  - DS (Data Segment)
  - SS (Stack Segment)
- přístup do segmentu – do segmentového registru se zavede selektor segmentu

Selektor segmentu

- Selektor – 16bitový

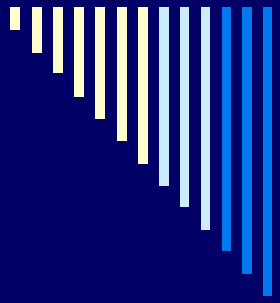
- 13bitů – index to GDT nebo LDT
- 1 bit – 0=GDT, 1=LDT
- 2 bity – úroveň privilegovanosti (0-3)





Selektor segmentu

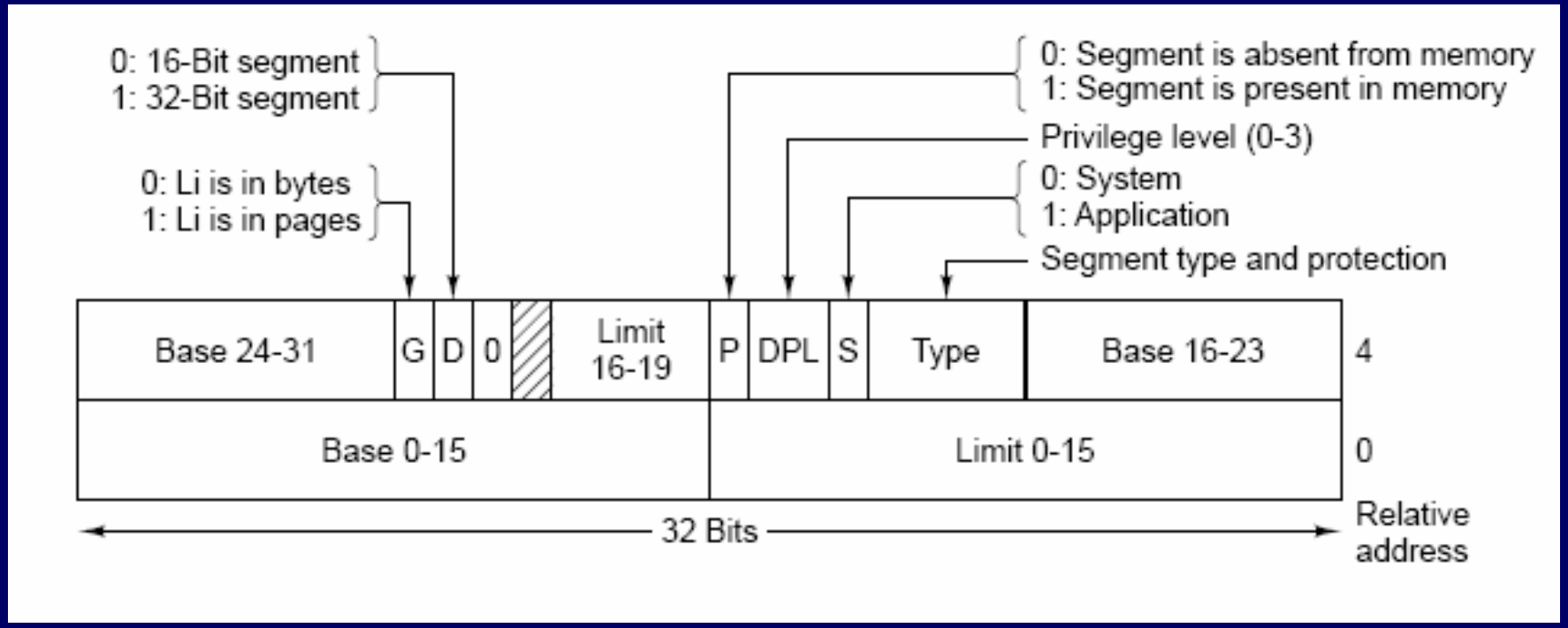
- 13 bitů – index, tj. max 2^13 = 8192 položek
  - selektor 0 – indikace nedostupnosti segmentu
- v době zavedení selektoru do segmentového registru CPU také zavede odpovídající popisovač z LDT nebo GDT do vnitřních registrů CPU
  - bit 2 – LDT nebo GDT
  - popisovač segmentu na adrese (selektor and 0fff8h) + zač. tabulky

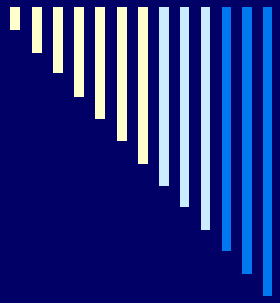


Popisovač segmentu

- 64bitů
  - 32 bitů báze
  - 20 bitů limit
    - v bytech, do 1MB (2^20)
    - v 4K stránkách (do 2^32) (2^12 = 4096)
  - příznaky
    - typ a ochrana segmentu
    - segment přítomen v paměti..

Pentium code segment descriptor

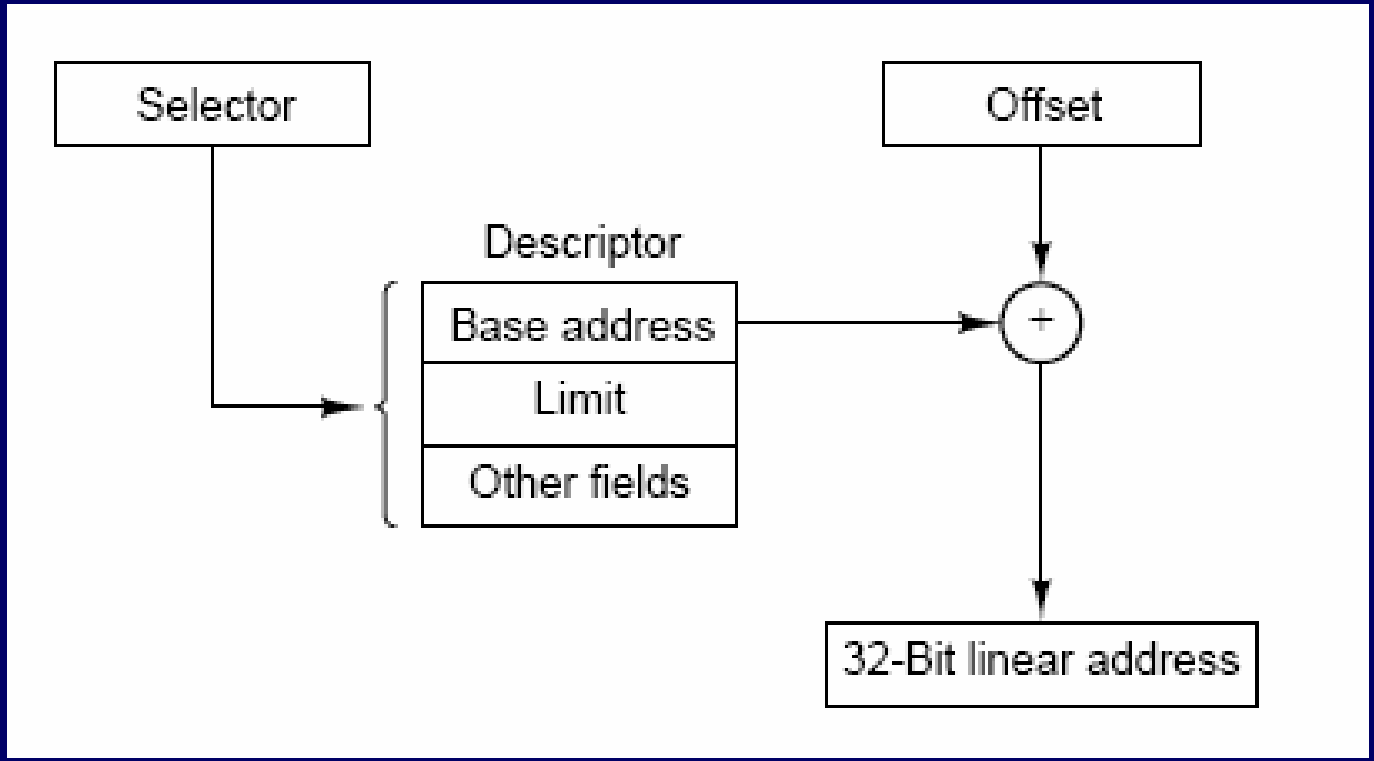


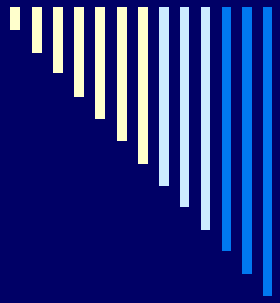


Konverze na fyzickou adresu

- Proces adresuje pamět pomocí segmentového registru
- CPU použije odpovídající popisovač segmentu v interních registrech
- pokud segment není – výjimka
- kontrola ofset > limit – výjimka
- 32bit. lineární adresa = báze + offset
- není-li stránkování – jde o fyzickou adresu
- jinak dále ..

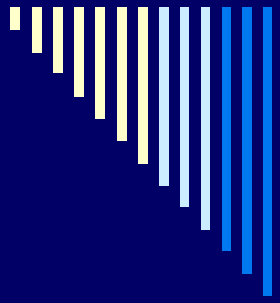
Konverze na fyzickou adresu





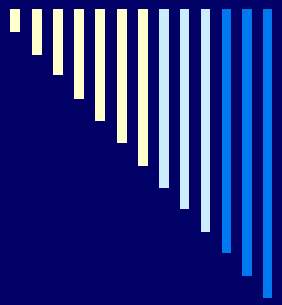
Konverze na fyzickou adresu

- stránkování – lineární adresa je VA, mapuje se na fyzickou pomocí tabulek stránek
- dvoúrovňové mapování



Pokračování

- Viz paio.pdf



Odkazy

http://www.zive.cz/h/Testcentrum/Ar.asp?ARI=110138&CHID=4&EXPS=&EXPA=

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