12. Stránkování a virtualizace paměti

zobrazení LAP na FAP nespojité = tabulka stránek

Tabulka stránek = pole položek indexované číslem stránky, obsah položky = číslo fyzického rámce + příznaky
Všechny stránky nejsou v paměti – **virtualizace paměti**. Přístup do stránky, která není v paměti, způsobí přerušení - přechod do systémového režimu, jádro může stránku zavést a pokračovat v provádění procesu.

Organizace tabulky stránek

Problém realizace tabulky stránek v HW

1. Transformace LAP na FAP se musí uskutečnit při každé instrukci a adresaci operandu (načtení instrukce, operandy)
   - rychlost provádění instrukcí < 1 ns
   - rychlost přístupu do paměti = **50 ns**
   - adresu nelze transformovat indexováním a čtením tabulky stránek z operační paměti!
2. Zobrazení plného 32bitového adresového prostoru - 1M položek (4 MB pro stránku 4 KB)
   - malý program by vyžadoval plnou tabuku stránek (kód na začátku adresového prostoru, zásobník na konci)
   - takto velkou tabulku stránek nelze umístit do rychlé interní paměti na procesoru (nehledě na problém jejího nastavování po přepnutí kontextu)
   - pro 64bitový adresový prostor nereálné i uložení v paměti

Řešení 1) TLB - Translation Lookaside Buffer (vyrovnávací paměť překladů adres)
Poslední (nebo nejčastěji používané) překlady adres jsou v rychlé asociativní paměti na procesoru.

Průběh transformace adresy:

- je transformace v TLB?
  ano, doba transformace t_t (<1 ns)
  ne, čtení položky tabulky stránek z operační paměti a případně uložení do TLB, doba transformace t_m (50 ns)

Průměrná doba transformace adresy:
úspěšnost TLB (hit ratio) p = 85 - 95 %

$$t_a = p.t_t + (1-p).t_m \qquad (3 - 8 \text{ ns})$$

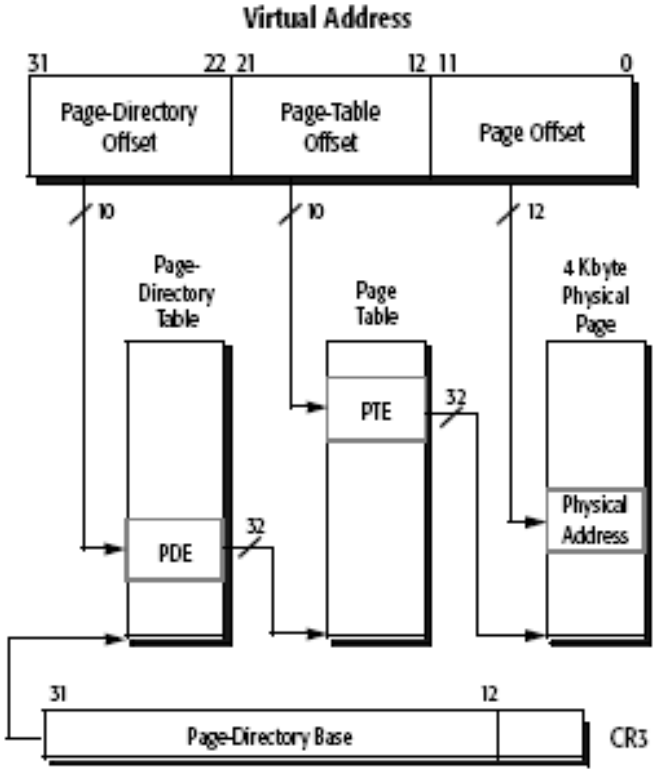
čím více se úspěšnost TLB blíží 100 %, tím menší zpomalení, může se skrýt díky řetězenému zpracování (pipeline).

Problémy:

- Realizace velké asociativní paměti (max. ~64 položek) → přímo adresovaná paměť s asociativitou stupně *n* (pro stejně mapovaný index lze uložit *n* údajů), při hledání se vybaví celý řádek a hledá se shoda indexu – mohou nastávat konflikty při přístupu na různé stránky mapované do stejné položky.
- Přepínání kontextu - jiný adresový prostor, nutné vyprázdnění TLB → doplnění čísla procesu (adresového prostoru) do položky TLB (SPARC, Intel IA-32e).
- Plnění TLB – page walk (průchod hierarchií tabulek), realizuje buď mikrokód nebo jádro (speciální přerušení TLB miss).
- Uvolňování položek TLB (LRU algoritmus) při plnění TLB.
- Synchronizace stavu mapování mezi procesory po změně tabulky stránek – každý procesor má svou TLB, pokud více procesorů používá stejný adresový prostor (jádro, sdílená paměť, vícevláknový program), musí se změna LAP projevit ve všech (invalidace celé nebo části TLB, nutné použít IPI – Inter-Processor Interrupt), tzv. *TLB shootdown*.

Řešení 2) Víceúrovňová (hierarchická) organizace tabulek
Tabulka stránek je hierarchická, logická adresa se dělí na více částí, každá část se použije samostatně jako index do tabulky první úrovně, druhé úrovně, ... Tabulka první úrovně popisuje celý adresový prostor, položky této tabulky obsahují báze tabulek 2. úrovně, atd. Teprve tabulka poslední úrovně obsahuje indexy fyzických rámců.

SPARC V8 - 3 úrovně, 8, 6, 6 bitů
68030 - 0 až 4 úrovně, programovatelné dělení
MIPS 3000, Sparc V9 - 0 úrovní (procesor používá pouze TLB, plnění TLB musí zajistit jádro systému)
Intel x86-32 - 2 úrovně po 10 bitech (max. 4GB FAP):



obě úrovně tabulek mají velikost 4KB = 1024 položek/4 byte
Aktivní tabulka stránek - nastavena privilegovanou instrukcí

```
movl ptbl,%cr3
```
fyzická adresa tabulky do CR3

číslo rámce			D	A	CD	WT	U	W	P

P (Present) - stránka je v paměti

W (Write) - stránka má povolen zápis

U (User) - uživatelská stránka

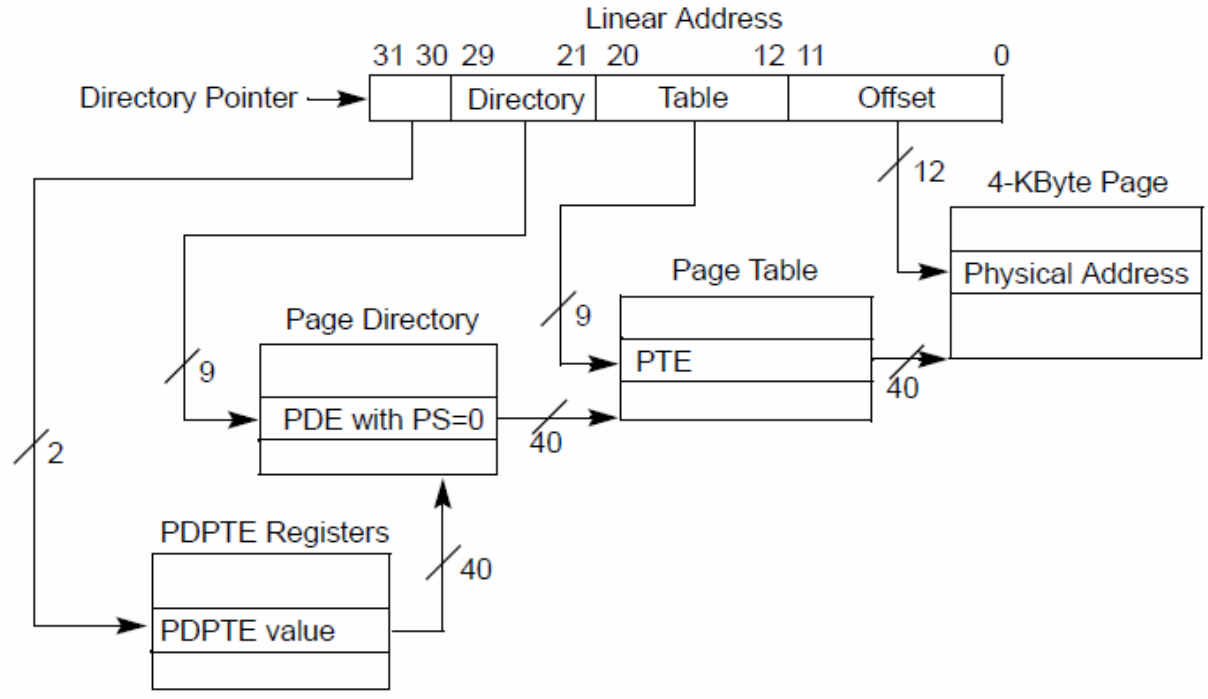
WT (Write Through) - data zapisovat paralelně do cache a hlavní paměti

CD (Cache Disable) - nepoužívat cache

A (Accessed) - stránka byla použita

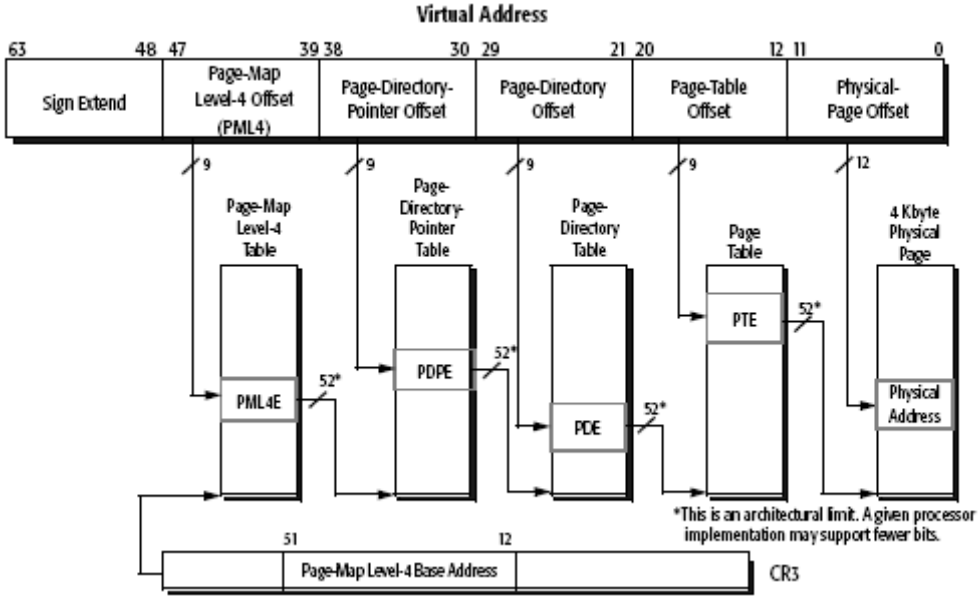
D (Dirty) - stránka byla modifikována

Pro širší fyzickou adresu (PAE, Pentium Pro 36 bitů, max. až 52 bitů) je nutné zvětšení položky na 64 bitů, tabulky pak obsahují jen 512 údajů – doplněna PDPTE se 4 položkami. PAE podporují všechny Unixy, 32bitové systémy Microsoft Windows jen serverové (Win XP, Vista, Win7, 8 mají limit max. 4GB fyzické paměti):

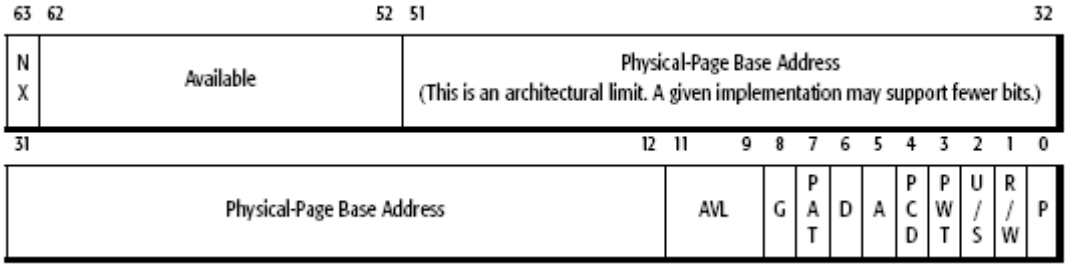


Pozn.: 40 bitů je číslo rámce, dolních 12 bitů je vždy 0, celkem lze tedy adresovat 52 bitů FAP (praktický limit je dán implementací procesoru, např. Intel Xeon 5600 má limit 40 bitů, tj. 1024 GB FAP, Xeon E5-26XX/SP má 46 bitů, tj. 64 TB).

AMD/Intel 64bitové architektura x86-64 (IA-32e):



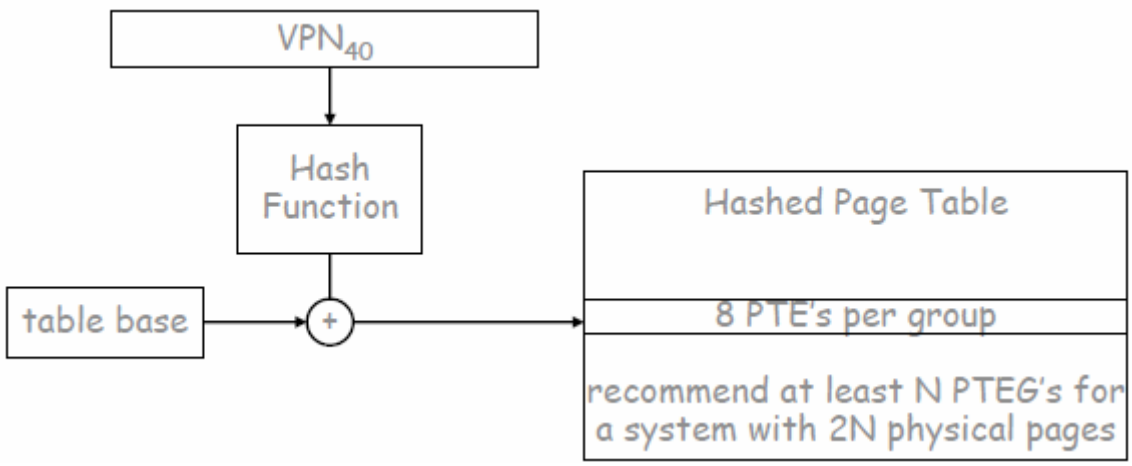
63-48 – znaménko z bitu 47, adresa LAP má 48 bitů,
všechny úrovně tabulek tabulky mají 512 položek/8 byte
(pokud je nastaven příznak PDE.PS – bit 7, je tabulka poslední úrovně vynechána a stránka má velikost 2MB)



PAT/PS = 1, další hierarchie tabulky nepoužita, adresuje se 1GB (PDPT), resp. 2 MB (PDT) souvislé paměti (bez virtualizace). Šířka fyzické adresy je stejná jako u PAE.

3. Inverzní (hashovaná) tabulka stránek

HP PA RISC, IBM PowerPC, Intel Itanium



Tabulka organizovaná podle rámců - obsahuje pouze tolik položek, kolik je rámců. Obsah položky - logická adresa stránky a číslo procesu

- minimalizuje režii tabulky stránek
- komplikované hledání (hashování)
- je třeba řešit konflikty při hashování (PowerPC dovoluje 8+8)
- nelze zobrazit aliasy (1 fyzická adresa zobrazená na různé logické adresy v různých procesech - sdílené knihovny)

Volba optimální velikosti stránky p

cílem je minimalizovat režii

interní fragmentace $\frac{p}{2}$

velikost tabulky stránek
s = průměrná velikost procesu $\frac{s}{p}.e$
e = velikost položky tabulky

minimum - derivace podle p = 0 $-\frac{s}{p^2}.e+\frac{1}{2}=0$

výsledek

$$p=\sqrt{2se}$$

e=4
s = 128 Kb, p = 1 KB; s = 1 MB, p = 4 KB; s = 8 MB, p = 8 KB

Virtualizace paměti

Při běhu procesu nemusí být celý obsah adresového prostoru trvale v paměti

- segmentování – komplikace při programování
- stránkování – menší režie, lepší odezva, odpadá externí fragmentace

LAP	stránky	$N = \{1, 2, \ldots, n\}$
FAP	rámce	$M = \{1, 2, \ldots, m\}$, $n \geq m \geq 1$
		$n > m$ virtuální paměť

Tabulka stránek: f_t: $N \rightarrow M \cup \{ 0 \}$

příznak zavedení stránky

$f_t(x) = \{$ *y*, pokud je stránka *x* v rámci *y* v čase *t*

0, pokud není (výpadek stránky)

Výpadek stránky – odkaz na platnou stránku, která není v paměti → přerušení a přechod do systémového režimu

Zpracování:

1. Výběr volného rámce.
2. Pokud není žádný rámec volný, výběr stránky, která bude odstraněna (nahrazena) – **nahrazovací algoritmus.** Pokud byla stránka modifikována, odložení stránky.
3. Zavedení požadované stránky a nastavení tabulky stránek.
4. Restart instrukce, která způsobila výpadek.

Vliv virtualizace na rychlost přístupu k paměti:

Pravděpodobnost výpadku stránky: $0 \leq p \leq 1$

Efektivní přístupová doba: $(1-p).(t_a+t_m)+p.$(doba výpadku)

100 ns 10 ms

Je nutno minimalizovat počet výpadků!

Stránkovací algoritmus

Sled stránek: posloupnost odkazů na stránky při provádění

$\omega = r_1, r_2, \ldots, r_t, \ldots \qquad r_t \in N, t \geq 1$

Stav paměti: množina stránek zavedených v paměti v čase *t*

$S_t = \{p_1, \ldots, p_k\}, \; p_i \in N$

$|S_t| \leq m$

Stránkovací algoritmus: zpracovává ω a generuje $S_1, \ldots, S_t$

$S_0 = \varnothing$

$S_t = S_{t-1} \cup X_t - Y_t$ $\quad X_t$ zavedené stránky, Y_t odstraněné stránky

Implementace stránkovacího algoritmu:

1. Výběr zaváděných stránek (kdy, kolik) - X_t
2. Výběr rámců, do kterých jsou stránky zaváděny
3. Výběr stránek, které mají být nahrazeny (odloženy) - Y_t

1. Výběr zaváděných stránek

a. Dopředné zavádění (prefetching)

b. **Zavádění na žádost** (demand paging) při výpadku stránky x:

$S_t = S_{t-1} \cup \{x\} - Y_t$ pro $f_t(x) = 0$, jinak $S_t = S_{t-1}$

Počet zaváděných stránek:

a. celý LAP

b. pouze žádaná stránka (demand paging)

c. předvídání (okolí žádané stránky)

Příklad:

```
move.l  (a0)++,(a1)++       #(až 6 výpadků)
```

2. Umísťování zaváděných stránek – nemá vliv na počet výpadků, může mít vliv na rychlost odkládání (clustering)

3. Nahrazovací algoritmus – výběr stránky, která má být odložena s cílem minimalizovat počet výpadků

Problém výběru – odstraněná stránka může být vzápětí potřebná, ale následující sled stránek většinou neznáme.
Praktické algoritmy: $Y_t = R(S_{t-1}, q, x)$
kde q je stav algoritmu

Klasifikace nahrazovacích algoritmů

- **lokální** – pouze stránky procesu, ve kterém nastal výpadek
- **globální** – všechny stránky bez ohledu na proces

Podle počtu rámců:

- **pevný počet rámců** – počet stránek v paměti zůstává stejný
- **proměnný počet rámců** – počet stránek v paměti se mění

Princip lokality

Při běhu programu nejsou třeba najednou všechny stránky LAP
Prostorová lokalita - je vysoká pravděpodobnost, že následující odkaz bude směřovat do stejné nebo sousední stránky

- sekvenční kód
- globální, lokální proměnné
- datové struktury (pole)

Časová lokalita - jistý časový interval jsou používány opakovaně stejné proměnné, stejný kód

- funkce, procedury s lokálními proměnnými
- cykly

Nahrazovací algoritmy jsou založeny na principu lokality, pokud neplatí, sebelepší algoritmus bez předvídání bude poskytovat špatné výsledky → optimalizace kódu, uložení dat

1. Optimální

- vybírá stránku, která bude nejdéle nepoužívána
- vyžaduje znalost budoucnosti
- minimální počet výpadků stránek

k, r_{k+t} je první výskyt x v $r_{t+1}, r_{t+2}, \ldots$

$d_t(x) = \{$ dopředná vzdálenost

∞

$R(S_t, q, x) = y, \; d_t(y) = \max_{z \in S_t}(d_t(z))$

m=4, n = 8, R = OPT, obsah rámců

ω	**2**	**1**	**3**	**5**	**4**	**6**	**3**	**7**	**4**	**7**	**3**	**3**	**5**	**5**	**3**	**1**
1	2	2	2	2	4	4	4	4	4	4	4	4	4	4	4	1
2		1	1	1	1	6	6	7	7	7	7	7	7	7	7	7
3			3	3	3	3	3	3	3	3	3	3	3	3	3	3
4				5	5	5	5	5	5	5	5	5	5	5	5	5

počet výpadků C = 8

2. LRU (Least Recently Used)

- vybírá stránku, která byla nejdéle nepoužita
- aproximace optimálního díky lokalitě odkazů

k, kde r_{t-k} je poslední výskyt x v $r_1, \ldots, r_t$

$b_t(x) = \{$ zpětná vzdálenost

∞

$R(S_t, q, x) = y, \; b_t(y) = \max(b_t(z))$

Implementace:

- pro každý rámec časová značka aktualizovaná při každém přístupu
- zásobník - při použití přesun nahoru, spodní odstranit

m=4, n = 8, R = LRU, obsah rámců

ω	**2**	**1**	**3**	**5**	**4**	**6**	**3**	**7**	**4**	**7**	**3**	**3**	**5**	**5**	**3**	**1**
1	2	2	2	2	4	4	4	4	4	4	4	4	4	4	4	1
2		1	1	1	1	6	6	6	6	6	6	6	5	5	5	5
3			3	3	3	3	3	3	3	3	3	3	3	3	3	3
4				5	5	5	5	7	7	7	7	7	7	7	7	7

počet výpadků C = 9

3. NRU (Not Recently Used)

- aproximace LRU s jedním bitem
- příznak použití stránky, při zavedení a pravidelně nulován, při použití nastaven (MMU)
- vybírá stránku, která nebyla použita v intervalu t-τ, …, t

4. LFU (Least Frequently Used)

- nejméně používaná stránka, při rovnosti použití LRU
- může odstranit právě zavedenou stránku

$f_t(x)$ = je počet výskytů x v r_1, …, r_t

$R(S_t, q, x) = y, \qquad f_t(y) = \min(f_t(z))$

5. FIFO

- vybírá stránku, která je nejdéle v paměti
- může být odstraněna často používaná stránka

$q_t(x)$ = je poslední výpadek x v r_1, …, r_t

$R(S_t, q, x) = y, \qquad q_t(y) = \min(q_t(z))$

6. LIFO

- stránka je nejkratší dobu v paměti

Hodnocení stránkovacích algoritmů A

$C(A, m, \omega) = \Sigma |X_t|$ celkový počet výpadků stránek

Uvažujeme pouze algoritmy **stránkování na žádost**:
A: $S_t = S_{t-1} \cup \{x\} - R(S_{t-1}, q, x)$ pro $f_t(x) = 0$, jinak $S_t = S_{t-1}$
Stránkovací algoritmus je definován nahrazovacím algoritmem

Věta: Algoritmus A=OPT minimalizuje C pro libovolné *m* a ω

Beladyho anomálie

Předpoklad: s rostoucím *m* bude C klesat

$$C(A, m, \omega) > C(A, m+1, \omega)$$

Příklad:
R=FIFO, N = {1, 2, 3, 4 , 5}, ω =1,2,3,4,1,2,5,1,2,3,4,5

m=3, stav paměti FIFO

ω	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
1	[1]	1	1	2	3	4	1	1	1	2	5	5
2		[2]	2	3	4	1	2	2	2	5	3	3
3			[3]	[4]	[1]	[2]	[5]	5	5	[3]	[4]	4

C = 9

m=4

ω	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
1	[1]	1	1	1	1	1	2	3	4	5	1	2
2		[2]	2	2	2	2	3	4	5	1	2	3
3			[3]	3	3	3	4	5	1	2	3	4
4				[4]	4	4	[5]	[1]	[2]	[3]	[4]	[5]

C = 10

Věta: $C(FIFO, m, \omega) < 2.C(FIFO, m+n, \omega)$

Hledáme algoritmy, které mají vlastnost:

$$C(A, m, \omega) \geq C(A, m+1, \omega)$$

Zásobníkové algoritmy

$$S(m, \omega) \subseteq S(m+1, \omega)$$

Stavy paměti tvoří pro rostoucí *m* systém podmnožin

Zásobník – stavy paměti lze zapsat ve tvaru zásobníku, kde je stav paměti dán *m* stránkami na vrcholu zásobníku

R=LRU, m=4, n = 8

ω	**2**	**1**	**3**	**5**	**4**	**6**	**3**	**7**	**4**	**7**	**3**	**3**	**5**	**5**	**3**	**1**
1	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1
2		2	1	3	5	4	6	3	7	4	7	7	3	3	5	3
3			2	1	3	5	4	6	3	3	4	4	7	7	7	5
4				2	1	3	5	4	6	6	6	6	4	4	4	7
5					2	1	1	5	5	5	5	5	6	6	6	4
6						2	2	1	1	1	1	1	1	1	1	6
7								2	2	2	2	2	2	2	2	2
D	∞	∞	∞	∞	∞	∞	4	∞	4	2	3	1	5	1	2	6

$C(LRU, 4, \omega) = 9$

Při rostoucí hloubce zásobníku zůstávají horní vrstvy zásobníku neměnné.

Zásobníková vzdálenost stránky

$D_x(\omega)$ = pozice stránky *x* v zásobníku nebo ∞, pokud není stránka zavedena

Vlastnosti zásobníkových algoritmů:

1. $D_x(\omega x) = 1$ Odkazovaná stránka je vždy na vrcholu musí platit $x \in S(1, \omega x)$
2. $D_y(\omega) \leq D_y(\omega x), y \neq x$ Neodkazovaná stránka nejde nahoru (omezení dané stránkováním na žádost)
3. $D_y(\omega) = D_y(\omega x)$ pro $y \neq x$ a $D_y(\omega) > D_x(\omega)$
   Zásobník pod odkazovanou stránkou před přesunem na vrchol se nemění

Vložení nové stránky nebo přesun stránky z hloubky na vrchol posune celý zásobník dolů → stránka na pozici *m* vypadne z aktuálního stavu paměti (pokud to není *x*).

4. $R(S \cup \{y\}, q, x) = R(S, q, x)$ nebo *y*

Věta: $C(STACK, m, \omega) \geq C(STACK, m+1, \omega)$
(nenástává Beladyho anomálie)

Organizace zásobníku - lze sestavit různé metody, nový stav může být konstruován permutacemi splňujícími podmínky 1-3. Algoritmy OPT, LRU, LFU, LIFO jsou zásobníkové.

Věta: Počet výpadků lze určit pro libovolné *m* a ω při znalosti zásobníkových vzdáleností pro konkrétní *m*:

$$C(STACK, m, \omega) = \sum_{k=m+1}^{\infty} a_k(\omega)$$

kde $a_k(\omega) = \| D_x(r_1\ r_2\ \ldots\ x) = k \|$ pro $\forall x \in \omega$
(počet výskytů vzdálenosti *k* v zásobníkových vzdálenostech)

Příklad: (D z posledního řádku předchozí tabulky)

$a_1 = 2$ $C_1 = a_2 + a_3 + \ldots + a_\infty = 14$
$a_2 = 2$ $C_2 = a_3 + \ldots + a_\infty = 12$
$a_3 = 1$ $C_3 = 11$
$a_4 = 2$ $C_4 = 9$
$a_5 = 1$ $C_5 = 8$
$a_6 = 1, a_\infty = 7$ $C_6 = 7$

Varianty LRU, NRU:

Clock

- cyklický seznam stránek, pro každou jednobitový příznak použití
- aktuální ukazatel do seznamu
- při hledání nahrazované stránky se testuje příznak použití
- pokud je nastaven, vynuluje se a ukazatel posune
- pokud je nulový, stránka se použije pro nahrazení a ukazatel posune

Second Chance (NRU)

Jako Clock, navíc příznak modifikace

$u = 0 \wedge w = 0$	nahrazovaná stránka, posun na další
$u = 1 \wedge w = 0$	vynuluje se u a posun na další
$u = 1 \wedge w = 1$	vynuluje se u a posun na další
$u = 0 \wedge w = 1$	zapíše se modifikovaná stránka, pak vynuluje w a posune na další

Zahlcení při stránkování (thrashing)

Pokud nemá proces v paměti určitý minimální počet stránek v paměti (pracovní množinu), je počet výpadků příliš velký.
Lokální + pevný počet rámců vyžaduje správný odhad pracovní množiny procesu (rovnoměrné dělení, podle priority)
Globální - v paměti může být více procesů, než je přípustné

Thrashing – stav, kdy počet výpadků překračuje přípustnou mez

- nízké využití procesoru - čekání na V/V při stránkování
- vede ke zvýšení multiprogramování → další zhoršení situace

Řešení

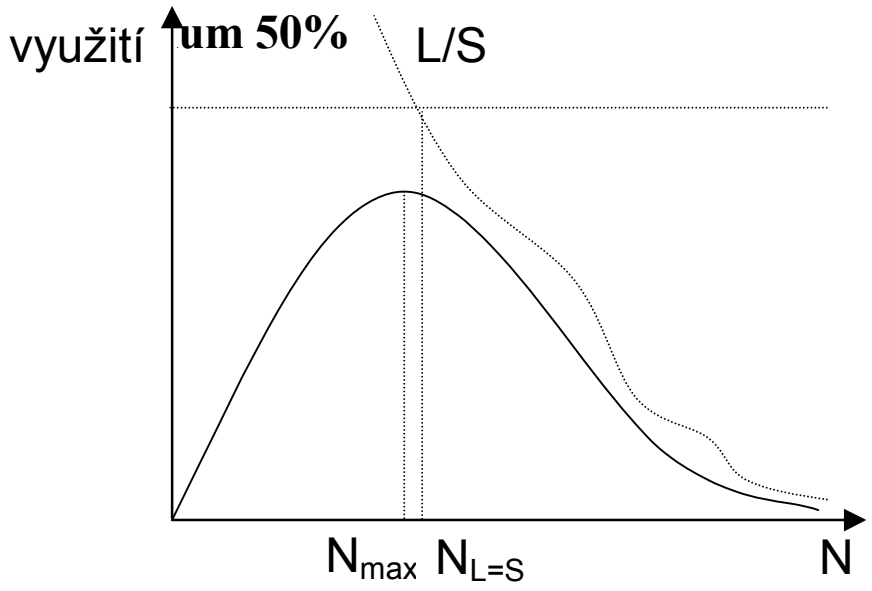
- volba stupně multiprogramování (odkládání)
- dynamická volba velikosti pracovní množiny pro proces

Volba stupně multiprogramování

nepřímé odvození - četnost výpadků stránek

1. Kritérium L = S

střední doba mezi výpadky (L) = střední doba obsluhy (S)



2. Stránkovací periferní zařízení využité na 50% (Denning)

3. Modifikovaný Clock (Carr)

Četnost plných průchodů cyklickým seznamem rámců:
a) pokud malá, lze zvýšit multiprogramování
b) pokud překročí limit, musí se začít odkládat

Nahrazovací algoritmy s proměnným počtem rámců

1. VMIN – optimální algoritmus
- udržuje v paměti pouze stránky, které budou potřebné

$$S_t = S_{t-1} \cup X_t - Y_t$$

$Y_t = \{y\},\ y \in S_{t-1} \wedge y \notin \{r_{t+1}, \ldots, r_{t+\tau}\}$ odstranění
$X_t = \{x\},\ x \notin S_{t-1}$ zavedení

Stránka, která nebude v budoucnu v intervalu $(t+1, t+\tau)$ je po použití uvolněna z paměti
τ = klouzavé okno, v paměti jsou pouze stránky, které budou použity v tomto okně

2. WS (Working Set)
$Y_t = \{y\},\ y \in S_{t-1} \wedge y \notin \{r_{t-1}, \ldots, r_{t-\tau}\}$ odstranění
$X_t = \{x\},\ x \notin S_{t-1}$ zavedení

V paměti je pouze množina stránek použitých v okně τ = **pracovní množina**
Problém implementace – při každém odkazu (nejen při výpadku stránky) se musí aktualizovat S_t a stránky, které do S_t nepatří uvolnit.

3. Page Fault Frequency
Nepřímá volba velikosti pracovní množiny:
- měří interval mezi výpadky
- pokud je interval menší než stanovené τ, lze při výpadku stránky přidat další stránku (zvětšit pracovní množinu)
- pokud je interval větší než τ, jsou odebrány všechny stránky nepoužité od posledního výpadku.

Odkládání procesů (swapping)

- počet běžících procesů > počet procesů v paměti
- odloží se všechny stránky procesu najednou

1. Spuštění odkládání = volba stupně multiprogramování

2. Výběr procesu pro odložení:

- pozastavený proces
- zavedený nejdelší dobu
- podle priority

3. Alokace odkládacího prostoru = alokace úseků

- předem, při alokaci paměti – zbytečně zabírá *swap*
- při odložení – nemusí být dostatek prostoru – uváznutí

4. Výběr odloženého procesu:

- připravený (ukončeno čekání)
- nejdéle odložen
- podle priority