Operační systémy

Segmentace, algoritmy pro náhradu stránek, návrh stránkovacích systémů.

Jan Trdlička

České vysoké učení technické v Praze, Fakulta informačních technologií
Katedra počítačových systémů

https://courses.fit.cvut.cz/BI-OSY

Obsah přednášky

Jednorozměrný VAS x Segmentace

- **Jednorozměrný VAS procesu**
  - Obsahuje datové struktury s různými vlastnostmi
    - ★ velikost: statická (TEXT, DATA) x dynamická (halda, zásobník),
    - ★ typ přístupu: privátní (zásobník,...) x sdílený (knihovny, sdílená paměť,...),

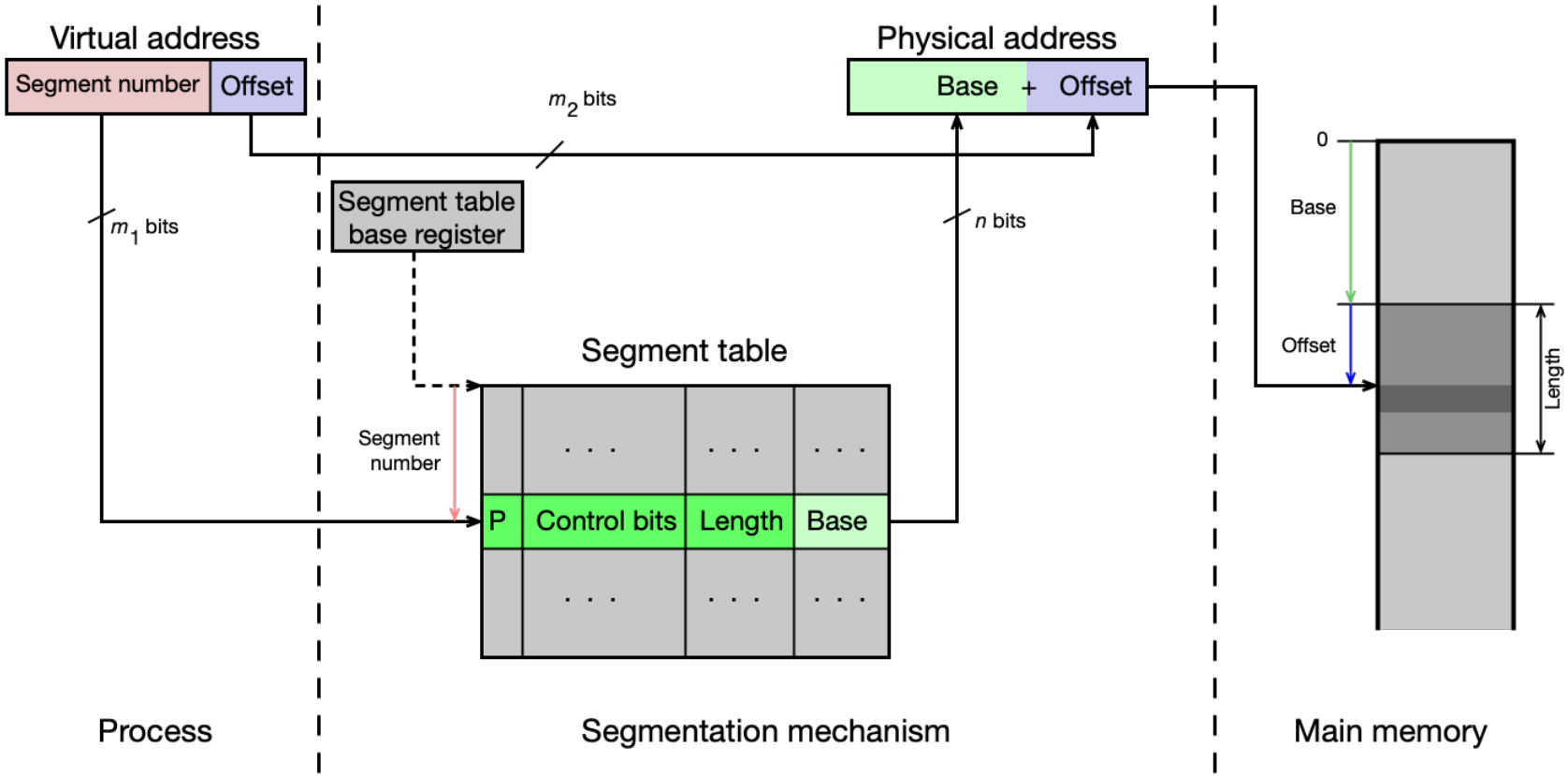
    $\Rightarrow$ mapování těchto struktur do jednorozměrného VAS je umělé,

    $\Rightarrow$ problémy při implementaci pomocí dynamických oblastí
    - ★ fragmentace fyzické paměti (problém s nalezením volné souvislé oblasti pro nový proces),
    - ★ rezervace místa ve VAS pro dat. struktury s dynamickou velikostí,
    - ★ sdílení struktur mezi více procesy.
- **Řešení**
  - Stránkování: řeší problémy v rámci jednorozměrného VAS.
    - ★ Podpora v CPU: x86-64, ARM, UltraSparc,...
    - ★ Používané v OS: MS Windows, Linux, Solaris,...
  - Segmentace: rozdělení VAS do několika segmentů (jednorozměrných oblastí) s různými vlastnostmi a v rámci segmentů se používá stránkování.
    - ★ Podpora v CPU: POWER od IBM,...
    - ★ Používané v OS: AIX (Unix od IBM),...

Čistá segmentace

- VAS procesu je rozdělen do několika segmentů (jednorozměrných oblastí) s různými vlastnostmi implementovaných pomocí dynamických oblastí.
- Programátor/překladač může definovat vlastnosti jednotlivých segmentů.
- Pro překlad virtuálních adres se používá tabulka segmentů.
- Položka této tabulky obsahuje následující informace
  - kontrolní bity,
  - velikost segmentu (Length),
  - počáteční adresu segmentu (Base).
- Číslo segmentu (nejvýznamnější bity virtuální adresy) funguje jako index do tabulky segmentů.
- OS si musí udržovat pro každý proces jednu tabulku segmentů.

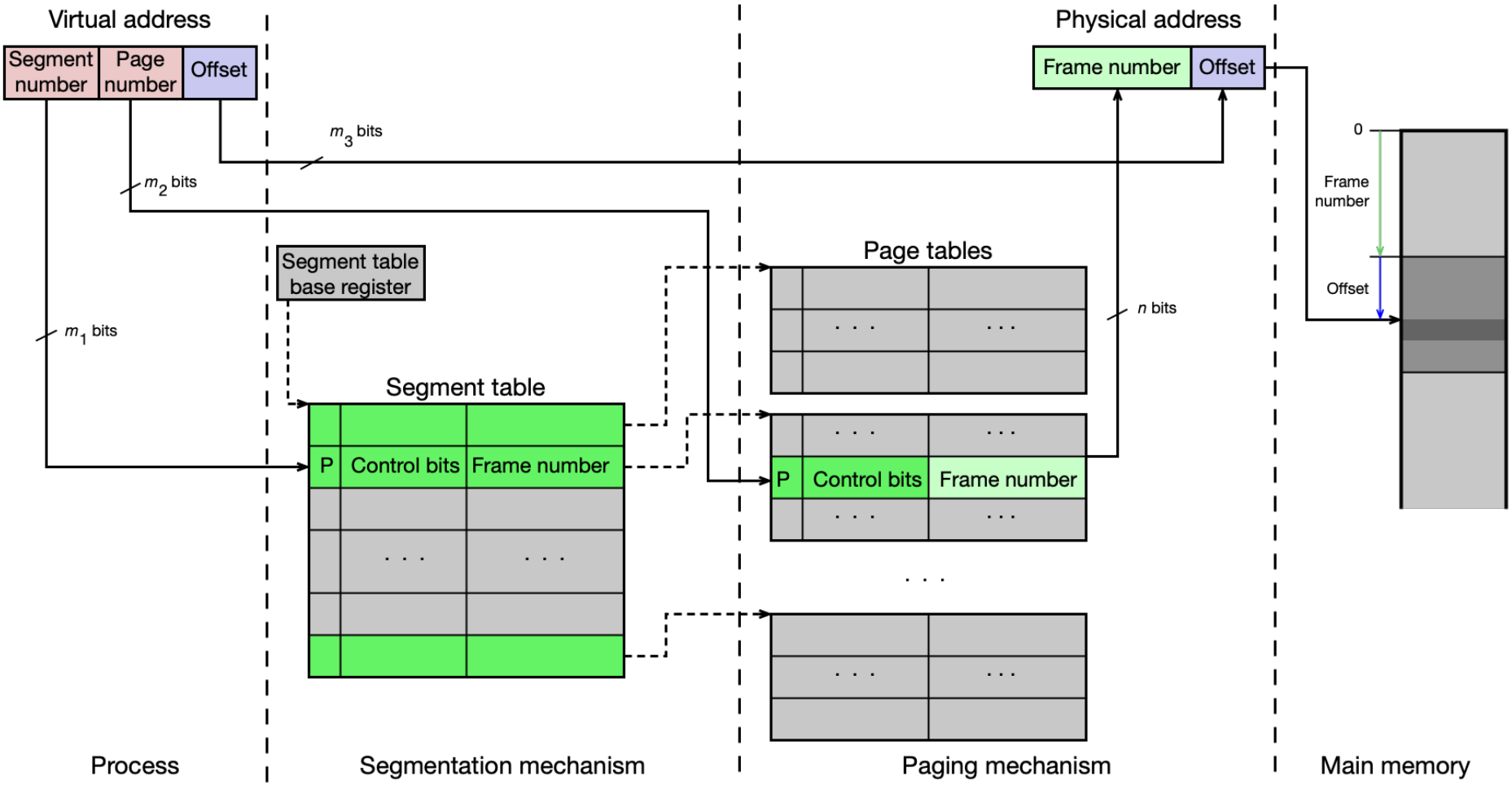
Čistá segmentace



Segmentace se stránkováním

- Při implementaci čisté segmentace vzniká podobný problém jak u dynamických oblastí
  $\Rightarrow$ problém nalezení volné oblasti pro nový segment,
  $\Rightarrow$ **segmentace se kombinuje se stránkováním a TLB**.
- **Mechanismus segmentace**
  - Číslo segmentu (nejvíce významné bity virtuální adresy) reprezentuje index do tabulky segmentů.
  - Položka v tabulce segmentů obsahuje číslo rámce, kde je uložená tabulka stránek/víceúrovňová tabulka stránek/invertovaná tabulka stránek.
- **Mechanismus stránkování**
  - Číslo stránky (méně významné bity virtuální adresy) reprezentuje index do tabulky stránek, ve které najdeme číslo rámce, kde je uložená příslušná stránka ve fyzické paměti.
  - Nejméně významné bity virtuální adresy reprezentují offset.
- **TLB**
  - Položka TLB obsahuje navíc číslo segmentu a jeho atributy.

Segmentace se stránkováním



Algoritmy pro náhradu stránek

- V okamžiku kdy většina/všechny rámce fyzické (hlavní) paměti jsou obsazené, je úkolem OS najít vhodný rámec, jehož obsah (stránka) se uvolní. K tomu slouží algoritmy pro náhradu stránek.
- **Požadavky na algoritmy**
  - Minimalizovat počet výpadků stránek.
  - Rychlost.
  - Jednoduchá implementace.
- **Princip algoritmů**
  - K instrukcím/datům ve VAS procesu se nepřistupuje náhodně
    - ⋆ Instrukce se většinou vykonávají sekvenčně, pouze občas pokračujeme instrukcí na vzdálené adrese, která je uložena na jiné stránce (větvení výpočtu, cyklus, skok,...).
    - ⋆ Data jsou uložena blízko sebe (halda, zásobník, pole, zřetězený seznam,...) na jedné nebo několika stránkách a přistupujeme k nim často sekvenčně (procházení pole/seznamu, push/pop na zásobník,...).

    $\Rightarrow$ Platím princip prostorové a časové lokality.

Optimální algoritmus

- **Princip**
  - Nahradí se stránka, která má čas příštího přístupu nejdelší (bude se k ní přistupovat za nejdelší dobu).
- **Vlastnosti**
  - Lze dokázat, že tento algoritmus generuje minimální počet výpadků stránek.
  - Nelze použít v praxi protože neznáme budoucnost
    $\Rightarrow$ ale lze použít pro porovnání kvality reálných algoritmů.
- **Příklad**
  - Ke stránkám se přistupovalo v pořadí: 2,3,2,1,5,2,4,5,3,2,5,2.
  - Fyzická paměť se skládá ze tří prázdných rámců a, b, c.
  - Jak se bude měnit obsazení rámců v průběhu času?

Číslo stránky		2	3	2	1	5	2	4	5	3	2	5	2
Rámec	a	2		2			2	4			2		2
	b		3							3			
	c				1	5			5			5	
Výpadek stránky		x	x		x	x		x			x		

  - ★ Přístup bez výpadku stránky/přístup s výpadkem stránky.
  - ★ Pokud je více možností, zvolíme rámec ze začátku abecedy.
  - ★ Počet výpadků: 6.

NRU algoritmus (Not Recently Used)

- **Princip**
  - Většina systémů se stránkováním si pro každou stránku pamatuje R bit (reference) a M bit (modified).
  - Při načtení stránky do paměti jsou bity nastaveny na hodnotu 0.
  - Tyto bity jsou nastavovány automaticky hardwarem při každém přístupu ke stránce.
    - R bit se nastaví, pokud se ke stránce přistupuje (čtení nebo zápis).
    - M bit se nastaví, pokud se změnil obsah stránky (zápis).
  - Abychom získali informaci, kdy se ke stránce přistupovalo (před dlouhou/krátkou dobou), je nutné, aby OS periodicky resetovat hodnotu R bitu na nulu.
  - Na základě hodnot R a M bitů můžeme stránky rozdělit do čtyř tříd.
    - Class 0: R=0, M=0,
    - Class 1: R=0, M=1,
    - Class 2: R=1, M=0,
    - Class 3: R=1, M=1.
  - Algoritmus NRU nahradí stránku z neprázdné třídy s nejnižším číslem.

NRU algoritmus (Not Recently Used)

- **Vlastnosti**
  - Jednoduchý na pochopení.
  - Rozumně složitá implementace.
  - Relativně malý počet výpadků stránek.

NRU algoritmus (Not Recently Used)

- **Příklad**
  - Ke stránkám se přistupovalo v pořadí: 2,3,2,1,5,2,4,5,3,2,5,2.
  - Fyzická paměť se skládá z tří prázdných rámců a, b, c.
  - OS resetuje R bit v časech $10 \times i$, kde $i \in \{0, 1, 2, \ldots\}$.
  - Jak se bude měnit obsazení rámců v průběhu času?

Číslo stránky			2	3	2		1	5		2	4	5	3		2	5	2
Typ přístupu			r	r	w		r	r		r	r	r	r		w	w	r
Čas			1	4	9	10	15	17	20	21	22	25	27	30	31	32	37
Rámce	a	Stránka	2		2					2					2		2
		R	1		1	0			0	1				0	1		1
		M	0		1					1					1		1
	b	Stránka		3				5			4		3				
		R		1		0		1	0		1		1	0			
		M		0				0			0		0				
	c	Stránka					1					5				5	
		R					1		0			1		0		1	
		M					0					0				1	
Výpadek stránky			x	x			x	x			x	x	x				

  - ★ Přístup bez výpadku stránky/přístup s výpadkem stránky.
  - ★ Reset R bitu.
  - ★ Pokud je více možností, zvolíme rámec ze začátku abecedy.
  - ★ Počet výpadků: 7.

FIFO algoritmus (First-In First-Out)

- **Princip**
  - OS si udržuje seznam všech stránek, které se aktuálně nachází v hlavní paměti.
  - V okamžiku, kdy se stránka nahraje do hlavní paměti, přidá se její záznam na konec seznamu.
  - FIFO algoritmus vybere první stránku ze seznamu jako vhodného kandidáta pro náhradu.
- **Vlastnosti**
  - Jednoduchý na pochopení a implementaci.
  - Nahrazuje se stránka, které je v paměti nejdéle.
  - Algoritmus nezohledňuje, kdy se ke stránce přistupovalo, ale pouze kdy se stránka nahrála do hlavní paměti
    $\Rightarrow$ způsobuje relativně velký počet výpadků stránek.

FIFO algoritmus (First-In First-Out)

- **Příklad**
  - Ke stránkám se přistupovalo v pořadí: 2,3,2,1,5,2,4,5,3,2,5,2.
  - Fyzická paměť se skládá ze tří prázdných rámců a, b, c.
  - Jak se bude měnit obsazení rámců v průběhu času?

Číslo stránky		2	3	2	1	5	2	4	5	3	2	5	2
Rámec	a	2		2		5			5	3			
	b		3				2				2	5	
	c				1			4					2
Začátek seznamu		a	a	a	a	b	c	a	a	b	b	c	a
Výpadek stránky		x	x		x	x	x	x		x		x	x

  - ★ Přístup bez výpadku stránky/přístup s výpadkem stránky.
  - ★ Pokud je více možností, zvolíme rámec ze začátku abecedy.
  - ★ Počet výpadků: 9.

Clock algoritmus

- **Princip**
  - Modifikovaný FIFO algoritmus.
  - Seznam stránek je implementován jako kruhová fronta.
  - Na počátku ručička (ukazatel) ukazuje na první položku fronty.
  - Pro každou stránku si pamatujeme její R bit (reference).
    - Když se stránka nahraje do paměti, OS nastaví R bit na hodnotu 1.
    - Při každém přístupu (čtení/zápis) ke stránce se nastaví R bit na hodnotu 1.
  - Postup při hledání vhodné stránky pro náhradu
    - Pokud ručka ukazuje na stránku, jejíž R bit má hodnotu 1, potom se resetuje R bit na hodnotu 0 a ručička se posune na následující stránku (položku) ve frontě.
    - Předchozí krok se bude opakovat, dokud ručička nebude ukazovat na stránku s R bitem rovným hodnotě 0.
      $\Rightarrow$ Tato stránka se nahradí a ručička se nastaví na následující stránku ve frontě.
  - **Vlastnosti**
    - Rozumně složitá implementace.
    - Algoritmus generuje nízký počet výpadků stránek.

Clock algoritmus

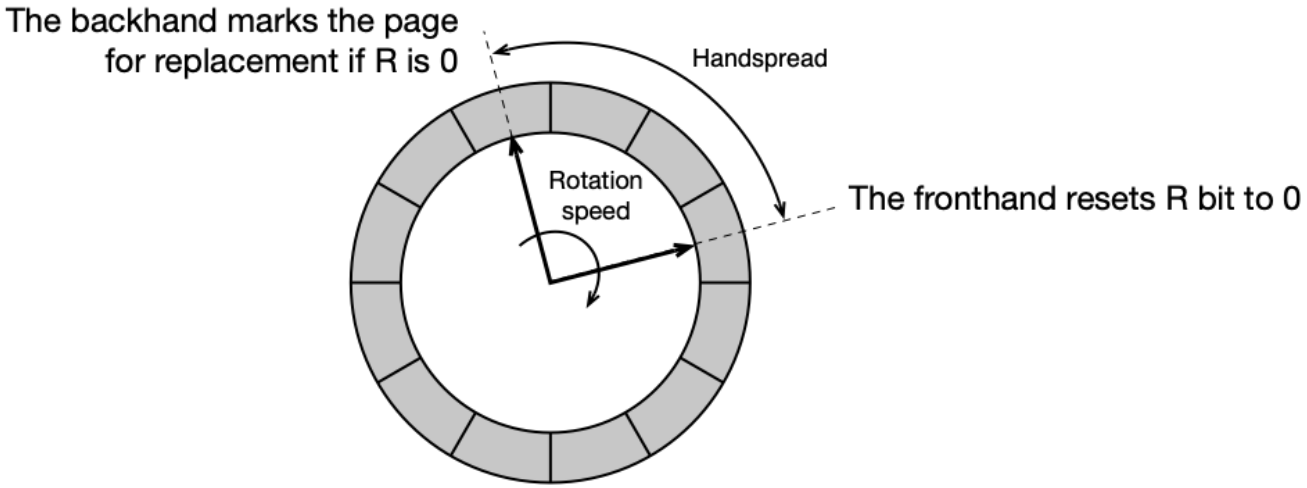
- **Příklad**
  - Ke stránkám se přistupovalo v pořadí: 2,3,2,1,5,2,4,5,3,2,5,2.
  - Fyzická paměť se skládá z tří prázdných rámců a, b, c.
  - Jak se bude měnit obsazení rámců v průběhu času?

Číslo stránky			2	3	2	1	5	2	4	5	3	2	5	2
Rámce	a	Stránka	2		2		5			5	3			
		R	1		1		1			1	1			
	b	Stránka		3				2				2		2
		R		1			0	1			0	1	0	1
	c	Stránka				1			4				5	
		R				1	0		1		0		1	
Ručička ukazuje na			a	a	a	a	b	c	a	a	b	b	a	
Výpadek stránky			x	x		x	x	x	x		x		x	

  - ★ Přístup bez výpadku stránky/přístup s výpadkem stránky.
  - ★ Reset R bitu.
  - ★ Pokud je více možností, zvolíme rámec ze začátku abecedy.
  - ★ Počet výpadků: 8.

Two-handed clock algoritmus

- Různé varianty Clock algortimu jsou používány v reálných OS.
- Jako příklad může sloužit varianta clock algoritmu se dvěma ručičkami, který byl používán v Unixu SVR4 a v současné době ho můžeme najít v jeho nástupcích.



- **Princip**
  - Algoritmus používá opět kruhovou frontu stránek a R bity jednotlivých stránek.
  - Obě ručičky se otáčejí stejnou rychlostí, která se mění podle aktuálního množství volné paměti.
  - U stránky, na kterou ukazuje první ručička (fronthand), se resetuje R bit na nulu.
  - Pokud stránka, na kterou ukazuje druhá ručička (backhand) má stále R bit nulový, pak je vybrána pro náhradu.
  - Rozevření ručiček (handspread) společně s rychlostí definuje časové okno, na základě kterého poznáme, zda se ke stránce nedávno přistupovalo.

LRU algoritmus (Least Recent Used)

- **Princip**
  - Vhodným kandidátem pro náhradu je stránka, ke které se nepřistupovalo po nejdelší dobu.
- **Vlastnosti**
  - Dobrá aproximace optimálního algoritmu.
  - Problematická implementace.
    - ★ Při každém přístupu ke stránce je nutné si zapamatovat informaci o "čase" přístupu.
    - ★ Vhodným kandidátem pro náhradu je stránka s nejmenším časem (musí se porovnat "časy" všech stránek).
- **Implementace pomocí speciálního hardwarového čítače**
  - Každá položka v tabulce stránek bude obsahovat navíc položku "time-of-used".
  - Hodnota čítače bude reprezentovat logický čas a bude se inkrementovat při každém přístupu do paměti.
  - Při přístupu ke stránce se uloží aktuální hodnota čítače do položky "time-of-used" v tabulce stránek.
  - Vhodným kandidátem pro náhradu je stránka s nejmenší hodnotou "time-of-used".

LRU algoritmus (Least Recent Used)

- **Příklad**
  - Ke stránkám se přistupovalo v pořadí: 2,3,2,1,5,2,4,5,3,2,5,2.
  - Fyzická paměť se skládá ze tří prázdných rámců a, b, c.
  - Jak se bude měnit obsazení rámců v průběhu času?

Číslo stránky		2	3	2	1	5	2	4	5	3	2	5	2
Rámec	a	2		2			2			3			
	b		3			5			5			5	
	c				1			4			2		2
Výpadek stránky		x	x		x	x		x		x	x		

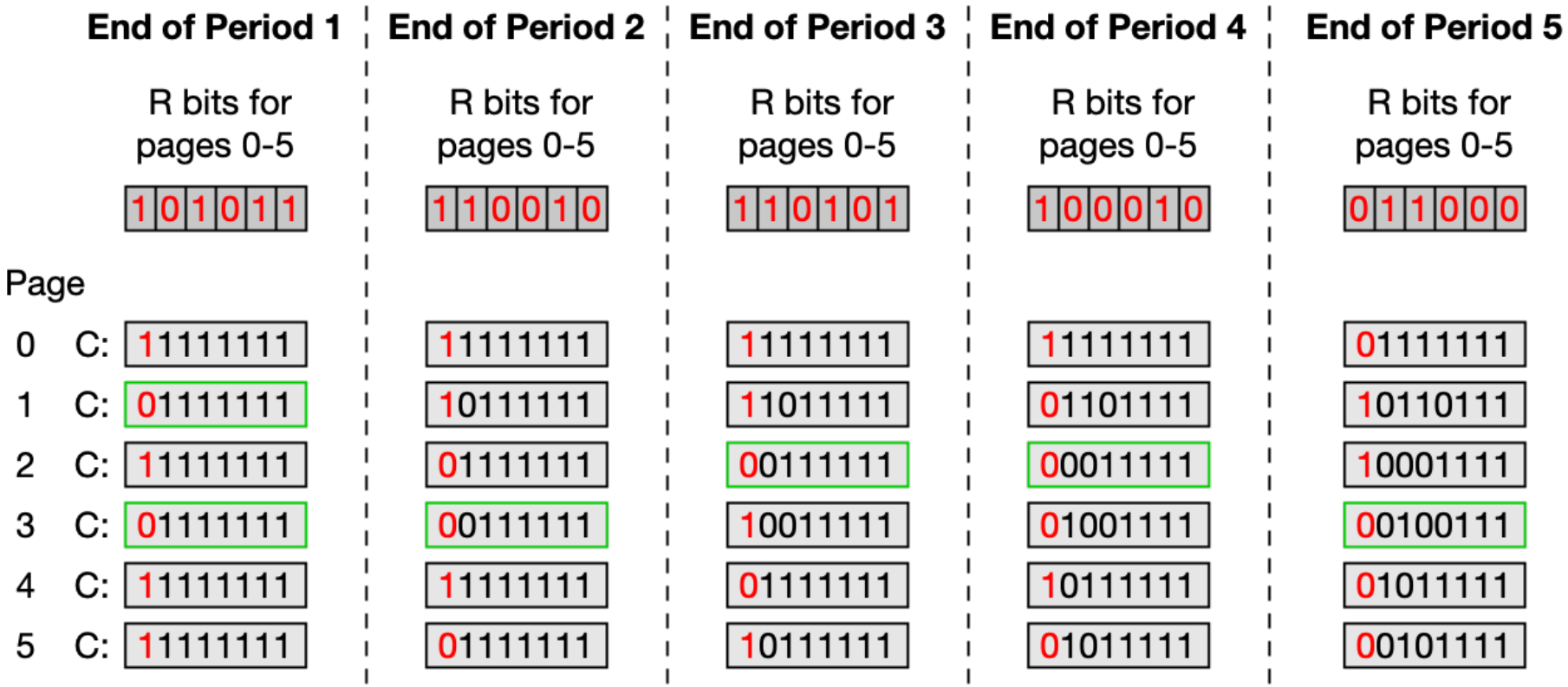
  - ★ Přístup bez výpadku stránky/přístup s výpadkem stránky.
  - ★ Pokud je více možností, zvolíme rámec ze začátku abecedy.
  - ★ Počet výpadků: 7.

Aging algoritmus

- Softwarová simulace LRU algoritmu.
- **Princip**
  - Pro každou stránku si systém pamatuje
    - ★ R bit (reference), který se nastaví při přístupu (čtení/zápis) ke stránce,
    - ★ *n*-bitový čítač *C*, který má všechny bity nastavené na 1 při načtení stránky do paměti.
  - Systém periodicky pro každou stránku
    1. posune obsah čítače *C* doprava o jeden bit,
    2. nastaví hodnotu nejvýznamejšího bitu čítače *C* na hodnotu R bitu,
    3. resetuje hodnotu R bitu na hodnotu 0.
  - Vhodným kandidátem pro náhradu bude stránka jejíž čítač *C* má nejmenší hodnotu.
- **Vlastnosti**
  - Implementace tohoto algoritmu má menší režii než LRU.
  - Algoritmus není tak přesný jako LRU.
    - ★ Pro každou stránku si nepamatuje přesný čas přístupu, ale pouze "interval", ve kterém se ke stránce přistupovalo.
    - ★ O každé stránce si pamatujeme pouze omezenou historii díky omezenému počet bitů čítače *C*.

Aging algoritmus

- **Příklad**
  - Systém používá pouze 6 stránek $(0, \ldots, 5)$, které se načetly do paměti v periodě 0, a 8-bitový čítač C.
  - Zeleně orámované jsou stránky, které jsou vhodné pro náhradu na konci dané periody.



Aging algoritmus

- **Příklad**
  - Ke stránkám se přistupovalo v pořadí: 2,3,2,1,5,2,4,5,3,2,5,2.
  - Fyzická paměť se skládá z tří prázdných rámců a, b, c.
  - OS resetuje R bit v časech $10 \times i$, kde $i \in \{0, 1, 2, \ldots\}$.
  - Čítač C bude reprezentován pouze 3 bity.
  - Jak se bude měnit obsazení rámců v průběhu času?

Číslo stránky			2	3	2		1	5		2	4	5	3		2	5	2
Čas			1	4	9	10	15	17	20	21	22	25	27	30	31	32	37
Rámce	a	Stránka	2		2			5			4	5	3				
		R	1		1	0		1	0		1	1	1	0			
		C	7		7	7		7	7		7	7	7	7			
	b	Stránka		3						2					2		2
		R		1		0			0	1				0	1		1
		C		7		7			3	7				7	7		7
	c	Stránka					1									5	
		R					1		0					0		1	
		C					7		7					3		7	
Výpadek stránky			x	x			x	x		x	x	x	x			x	

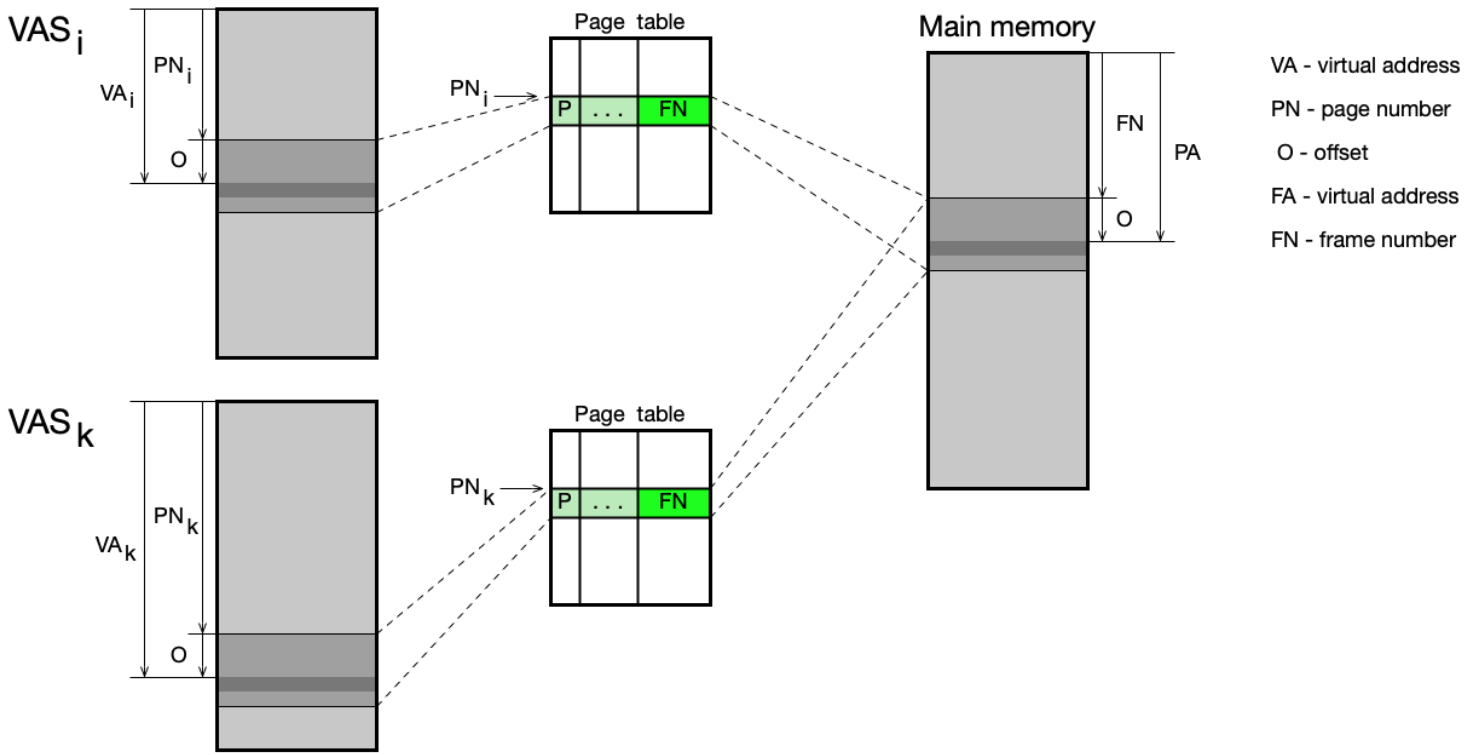
  - ★ Přístup bez výpadku stránky/přístup s výpadkem stránky.
  - ★ Reset R bitu.
  - ★ Pokud je více možností, zvolíme rámec ze začátku abecedy.
  - ★ Počet výpadků: 9.

Návrh systémů se stránkováním

- Správa paměti je daleko komplexnější téma než jenom samotná architektura paměti (virtuální paměť, stránkování,...) a algoritmy pro náhradu stránek.
- Při návrhu OS je nutné vyřešit celou řadu dalších otázek, tak aby OS splňoval požadavky, které na něj klademe.
  - Jak zajistit sdílení paměti mezi více procesů?
  - Jak implementovat adresový prostor jádra OS?
  - Kdy se stránka nahraje do hlavní paměti?
  - Kde v hlavní paměti by stránka měla být umístěna?
  - Kolik fyzické paměti má být přiděleno konkrétnímu procesu?
  - Kolik procesů by mělo být maximálně ve fyzické paměti?
  - Jaký bude postup OS, když začne docházet fyzická paměť?
  - . . .

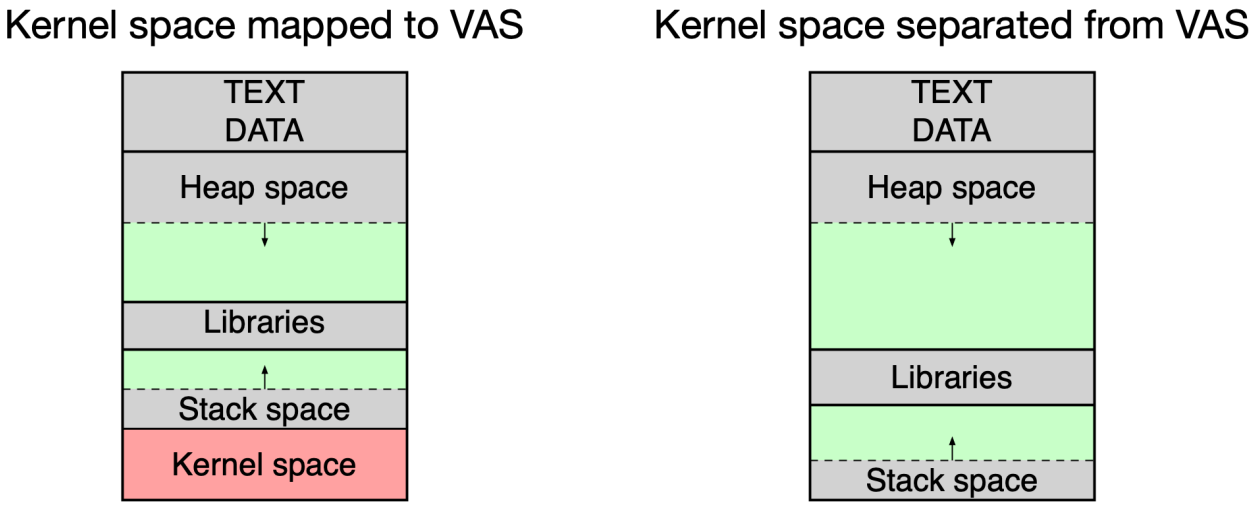
Návrh systémů se stránkováním

- **Jak zajistit sdílení paměti mezi více procesů?**
  - Procesy používají soubory/struktury, které je vhodné sdílet s ostatními procesy
    - ★ spustitelné binární programy,
    - ★ knihovny,
    - ★ soubory mapované do paměti (memory mapped files),
    - ★ sdílená paměť.
  - Pokud OS používá stránkování/segmentaci, pak sdílení lze implementovat relativně jednoduše.



Návrh systémů se stránkováním

- **Jak implementovat adresový prostor jádra OS?**



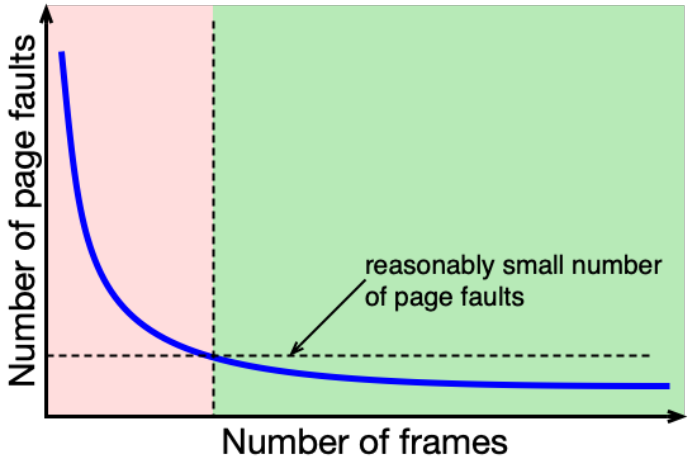
  - Adresový prostor jádra OS mapovaný do VAS procesu
    - ★ U starších většinou 32-bitových procesorů (např. x86, SPARC V7,...) byl adresový prostor jádra mapovaný do VAS každého procesu.
    - ★ Při přechodu z "user modu" do "kernel modu" nebylo nutné měnit nastavení adresového prostoru.
  - Oddělený prostor jádra
    - ★ U novějších typicky 64-bitových procesorů (např. x86-64, SPARC V9,...) je adresový prostor jádra většinou oddělený od VAS procesů.

Návrh systémů se stránkováním

- **Kdy se stránka nahraje do hlavní paměti?**
  - Používají se dvě základní strategie (fetch strategies).
  - Stránkování na žádost (demand paging)
    - ★ Stránka je nahrána z disku do hlavní paměti až v okamžiku, když se proces pokouší k ní přistoupit.
  - Před-stránkování (prepaging)
    - ★ Tato strategie je založená na principu prostorové lokality.
    - ★ Když se proces pokusí přistoupit ke stránce, která ještě není v hlavní paměti, tak jádro načte tuto stránku a několik následujících stránek současně.
      ⇒ minimalizuje se počet přenosů z disku,
      ⇒ do paměti se mohou nahrát stránky, které se nebudou používat.
    - ★ Většina OS používá tuto strategii.
- **Kde v hlavní paměti by stránka měla být umístěna?**
  - V případě používání dynamických oblastí nebo čisté segmentace se používaly různé algoritmy (např. best-fit/worst-fit,...) jejichž úkolem bylo minimalizovat fragmentaci paměti.
  - Pokud systém používá stránkování, pak je nová stránka nahrána do libovolného volného rámce.

Návrh systémů se stránkováním

- **Kolik fyzické paměti má být přiděleno konkrétnímu procesu?**



  - Graf zobrazuje závislost mezi počtem přidělených rámců fyzické paměti procesu a počtem jím generovaných výpadků stránek.
    - ★ Pokud procesu přidělíme málo rámců fyzické paměti, tak bude generovat hodně výpadků stránek (růžová oblast grafu).
    - ★ Naopak od určitého počtu přidělených rámců se počet výpadků stránek příliš nezmění i když procesu přidáme další rámce.

  $\Rightarrow$ Pokud každému procesu P_i přidělíme "rozumný" počet rámců WS_i (Working Set), pak bude generovat "rozumný" počet výpadků stránek.

  - **Variable-allocation strategy**
    - ★ Většina OS přiděluje rámce fyzické paměti jednotlivým procesům podle jejich aktuálních potřeb.

Návrh systémů se stránkováním

- **Kolik procesů by mělo být maximálně ve fyzické paměti?**
  - V systému by měl být takový počet aktivních procesů (procesy, kde aspoň jedno vlákno je ve stavu "Ready"/"Running"), aby součet jejich WS_i byl menší než velikost hlavní paměti
    $\Rightarrow$ jinak budou procesy soupeřit o paměť a fyzická paměť nebude efektivně využívána.
  - Úkolem administrátora je, aby toto zajistil
    - omezením počtu aplikací běžících v systému,
    - omezením požadavků na paměť od jednotlivých aplikací.

Návrh systémů se stránkováním

- **Jaký bude postup OS, když začne docházet fyzická paměť?**
  - V hlavní paměti se můžou nacházet různá data
    - ★ jádro OS a části VAS jednotlivých procesů,
    - ★ části/celý obsah bývalých nebo aktuálně otevřených souborů,
    - ★ obsah některých pseudo FS (např. obsah adresáře `/tmp`),...
  - OS si obvykle udržuje množinu (pool) volných rámců fyzické paměti, tak aby byl schopný okamžitě uspokojit požadavek na alokaci/načtení nové stránky.
  - Pokud klesne počet volných rámců pod kritickou mez (např. definována jako parametr jádra OS), pak OS začne aktivně uvolňovat hlavní paměť.
  - Pro uvolňování hlavní paměti používá OS dva mechanismy
  - **Paging-out strategy**
    - ★ OS začne uvolňovat jednotlivé stránky, které byly vybrány pomocí příslušného algoritmu pro náhradu stránek.
    - ★ Modifikované stránky uloží na disk a rámce všech vybraných stránek si přidá do množiny volných rámců.
  - **Swaping strategy**
    - ★ Pokud předchozí strategie nezajistí dostatečné navýšení počtu volných rámců, tak OS začne odkládat celé procesy na disk (disková oblast/soubor).
    - ★ Přednost mají procesy, kde ani jedno vlákno není ve stavu "Ready"/"Running".
    - ★ Do speciální odkládací diskové oblasti/souboru uloží pouze "anonymní" stránky, které neexistují jinde ve FS (např. halda, zásobník,...).

Použité zdroje

1. A. S. Tanenbaum, H. Bos: *Modern Operating Systems (4th edition)*, Pearson, 2014.
2. W. Stallings: *Operating Systems: Internals and Design Principles (9th edition)*, Pearson, 2017.
3. A. Silberschatz, P. B. Galvin, G. Gagne: *Operating System Concepts (9th edition)*, Wiley, 2012.
4. R. McDougall, J. Mauro: *Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture (2nd edition)*, Prentice Hall, 2006.