Výhoda:

- využívá časovou lokalitu => zvyšuje hit rate
- při N=2 je velmi jednoduché udržovat informaci o nejméně používané položce v setu (1 LRU bit, protože v setu jsou jen 2 bloky)

Nevýhoda:

- při N>2 je HW komplikovanější, časová složitost pro udržení LRU informace také roste

Náhodně [Random]

Při větších velikostech cache je na tato metoda téměř stejně úspěšná, jako LRU, ale nevyžaduje další logiku.

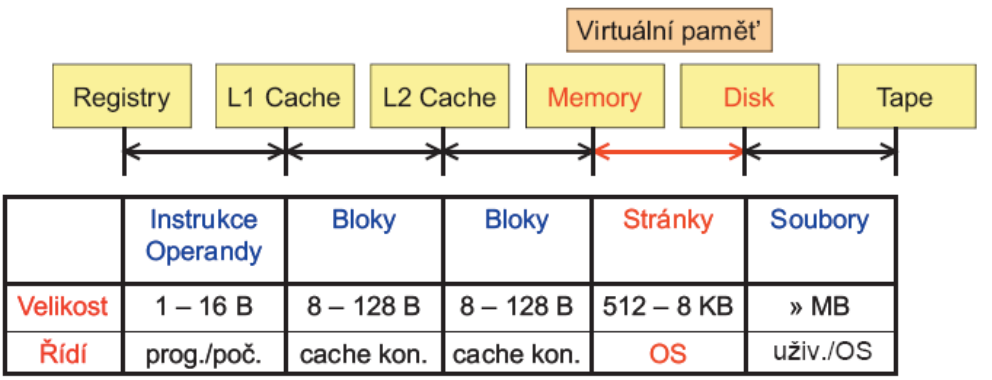
Materiály

- X36APS, přednáška 6 (*http://service.felk.cvut.cz/courses/36APS/lectures/aps7_2.pdf*)
- X36APS, přednáška 7 (*http://service.felk.cvut.cz/courses/36APS/lectures/aps8_1.pdf*)
- http://www.pcguide.com/ref/mbsys/cache/funcComparison-c.html
- http://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/CACHE.HTML

Virtuální paměť a dynamický překlad adres

Virtuální paměť

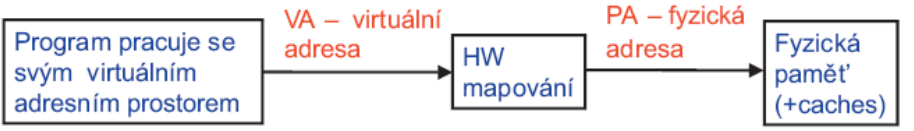
Situace vypadá takto:



	Instrukce Operandy	Bloky	Bloky	Stránky	Soubory
Velikost	1 – 16 B	8 – 128 B	8 – 128 B	512 – 8 KB	» MB
Řídí	prog./poč.	cache kon.	cache kon.	OS	uživ./OS

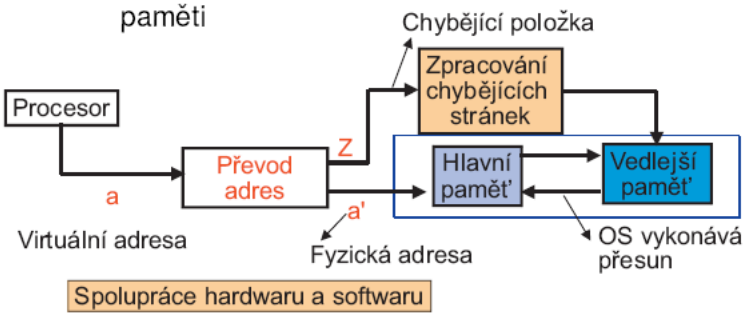
Tedy virtuální paměť jsou stránky obsažené v operační paměti a na disku. Vlastnosti virtuální paměti:

- Umožňuje sdílení paměti (operačním systémem)
- Vzájemná ochana programů (v současnosti je důležitější ochrana dat než využití principu lokalit), tzn. to aby jeden program nepřepisoval druhému programu jeho data a tak.
- Každý běžící program pracuje se svým virtuálním adresním prostorem

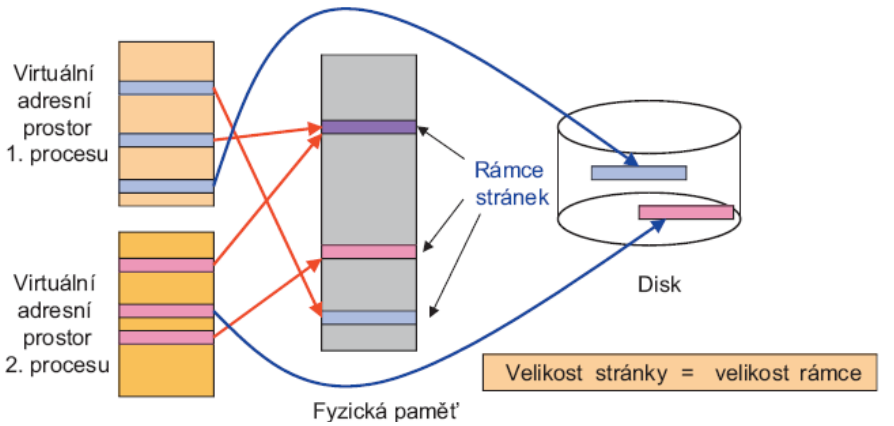


- OS rozhoduje o umístění daných běžících programů v paměti
- Hardware provádí mapování virtuální adresy na fyzickou – máme *viruální adresní prostor V*={0,1,...,n} a *fyzický adresní prostor M*={0,1,...,m}, kde n>>m. Mapování V→M provádí mapovací funkce.

$MAP(a) = a'$, když data na adrese a VA jsou reprezentována daty fyzické paměti PA na adrese $a' \in M$
$= Z$ když data na místě VA a nejsou obsažena ve fyzické paměti



- Virtuální prostor je rozdělena na stejně velké "stránky" (pages), které se přiřazují jednotlivým běžícím procesům
- Fyzická paměť je rozdělena na stejně velké rámce (frames)

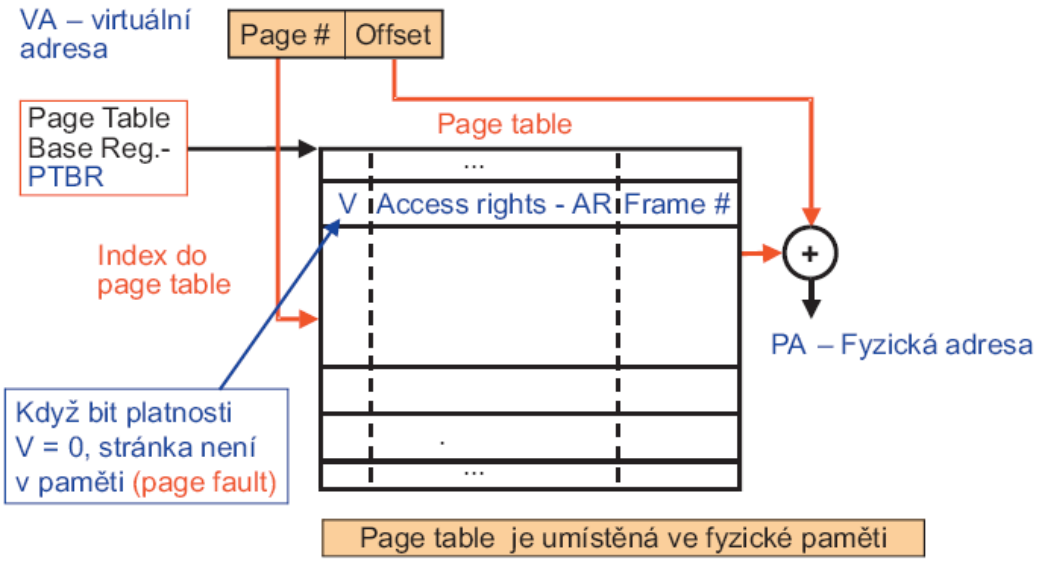


- Stránky tvoří souvislý VA prostor a jsou mapované do rámců, avšak:
  - Korespondující rámce jsou libovolně umístěny ve fyzické paměti (nejsou za sebou)
  - Ne všechny stránky jsou v každém okamžiku mapovány do nějakého rámce
- Stránka je jednotkou mapování a současně to je jednotka přenosu mezi diskem a hlavní pamětí - fyzickou pamětí

Tabulka stránek

Protože není možné nalézt dostatečně jednoduchou mapovací funkci, pomáháme si tabulkou stránek. Tabulka stránek je vyhledávací tabulka umístěná v hlavní paměti.

Virtuální adresa je pak rozdělena na 2 části: na číslo stránky a offset (posun) v rámci této stránky. Pomocí čísla stránky najdeme v tabulce stránek příslušný záznam (pokud tam je a pokud je tento záznam platný a pokud máme příslušná právo pro přístup k této stránce) - tím získáme adresu rámce, do kterého je požadovaná stránka právě namapována. Přičtením offset najdeme požadované místo.



Formát položky tabulky stránek

- Struktura tabulky stránek je dána strukturou OS (obsahuje mapování virtuálních adres na fyzické)
- Každý proces běžící v OS má vlastní tabulku stránek
- Pokud se někam ukládá stav procesu (kontext), myslí se tím PC, všechny registry a tabulka stránek
- OS mění tabulky stránek změnou PTBR (Page Table Base Register) - PTBR obsahuje bázovou adresu tabulky stránek
- Pokud není požadovaný záznam v tabulce stránek platný, OS přenese příslušnou stránku z disku do hlavní paměti
- Stránky mají stejnou velikost => každý rámec je využit => nedochází k fragmentaci
- OS rezervuje Swap Space na disku pro každý proces

Porovnání Virtuální paměti a Cache

Cache	Virtuální paměť
Blok/řádek	Stránka
Miss	Page fault
Velikost bloku: 8 – 128B	Velikost stránky: 512B – 8KB
Typ: DM, N-way, asociativní	asociativní
Výběr oběti: LRU/Random	Výběr oběti: LRU
Write Thru/Write back	Write back

Výpadek stránky

tj. hledáme stránku v hlavní paměti, ale ona tam není – prostě to samé jako Cache Miss ale ve Virtuální paměti.

Pokud jsou data na disku:

- Načte se požadovaná stránka z disku do prázdného rámce v paměti
- Přenos prostřednictvím DMA a přepnutí na proces, který čeká
- Když je DMA ukončen, dojde k vyvolání přerušení a aktualizaci tabulky stránek procesu
- Při přepnutí zpět na původní úlohu jsou požadovaná data v hlavní paměti (načtené stránce)

Když dochází paměť:

- Stránky s Dirty bitem náležící programu zapíšeme na disk (Dirty bit říká, se obsah stránky změnil, ale na disku je stále zapsaná její stará verze)
- Pomocí některé ze strategií výběru oběti (např. LRU) vybereme rámec obsazený stránkou a uvolníme ho
- Aktualizuje se tabulka stránek programu

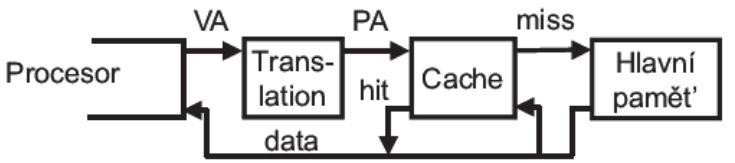
Problem 1

Je málo fyzické paměti

- mějme 64 MB fyzické paměti
- *N* procesů a každý z nich má 4 GB virtuální paměti
- můžeme mít až 1000 virtuálních stránek na 1 fyzickou stránku

Řešení: Princip prostorové lokality

- velikost stránky $\approx$ 4 KB $\Rightarrow$ množství blízkých referencí
- i veliký program požaduje v určitém čase jen málo stránek
- pracovní set: "právě" používané stránky
- mapování každé adresy $\Rightarrow$ další přístup do paměti navíc
- pozorování: platí princíp lokality uvnitř stránky $\Rightarrow$ musí platit ve virtuálních adresách těchto stránek
- proč nepoužít "cache – TLB" pro převod VA $\rightarrow$ PA pro urychlení převodu?

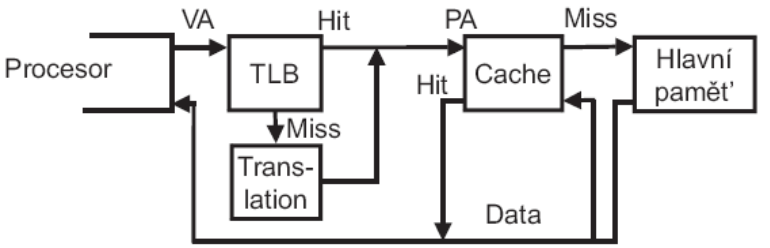


- cache typicky pracuje s fyzickými adresami
- tabulka stránek představuje další přístup do paměti pro každý programový přístup do paměti
- jak řešit tento problém?

TLB

Translation Lookaside Buffer je cache pro položky tabulky stránek (položkou tabulky stránek je vlastně překlad virtuální adresy na fyzickou). Doba přístupu do něj je srovnatelná s dobou přístupu do cache paměti a typicky má 128 - 256 položek. Jakožto cache může být přímo mapovaná, s omezeným stupněm asociativity nebo plně asociativní. Jedna položka TLB obvykle obsahuje :

- Příznak platnosti
- Virtuální adresu
- Fyzickou adresu, na níž je virtuální adresa namapována
- Přístupová práva
- Dirty bit (TLB pracuje v režimu Write-Back)
- Referenční příznak (používaný při hledání oběti LRU)
- ASID (identifikace procesu / uživatele)



Co když je Miss TLB? Pak se HW podívá do tabulky stránek a načte nové PTE (Page Table Entry) do TLB.

Problem 2
Tabulka stránek je příliš velká!

- 4 GB virtuální paměti, 4 KB stránka
- $\sim$ 1 milión PTE
- 4 MB jen pro tabulku stránek jednoho procesu
- 25 procesů $\Rightarrow$ 100 MB pro tabuky stránek

Pokud proces běží, tak jeho tabulka stránek musí být celá ve fyzické paměti.

Řešení:

- víceúrovňové stránkování
- inverzní tabulka stránek
- segmentace

Víceúrovňové stránkování

Při víceúrovňovém stránkování je virtuální adresa rozdělena na více částí. Například pro dvouúrovňové stránkování je virtuální adresa rozdělena na "super" číslo stránky, offset1 a offset2.

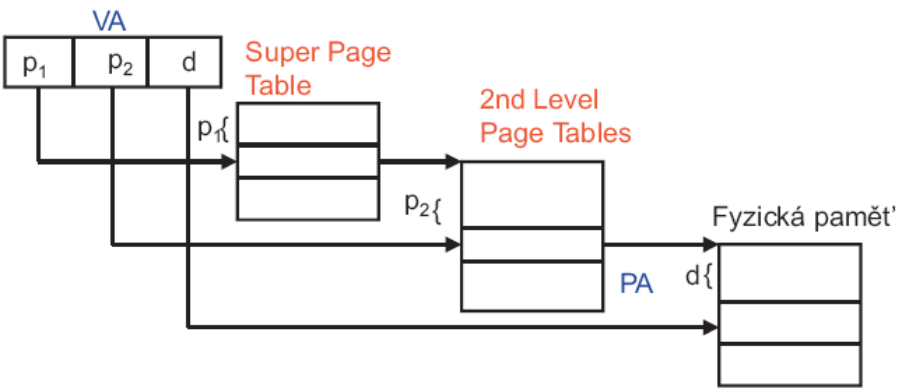
Jednoúrovňová tabulka stránek:

Číslo stránky p = 20 b	Offset d = 12 b

Víceúrovňová tabulka stránek:

Super Číslo stránky p_1 = 10 b	Offset p_2 = 10 b	Offset d = 12 b

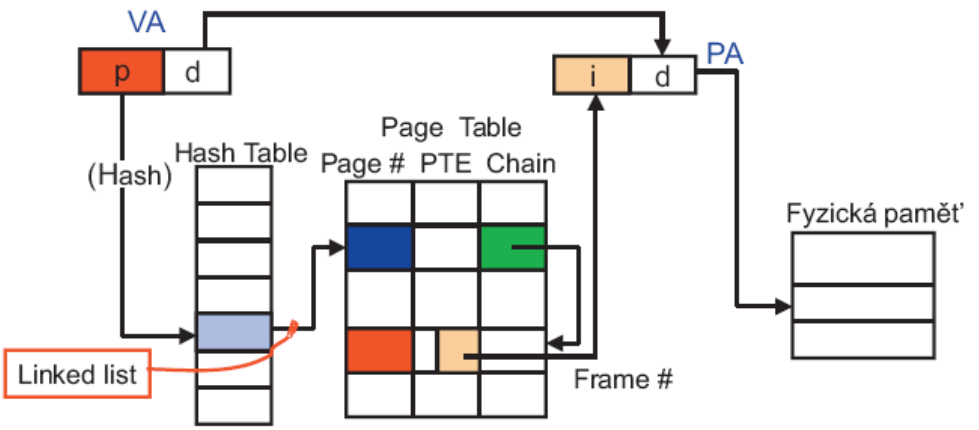
"Super" číslo stránky slouží k vyhledání záznamu v "super" tabulce stránek. Ta obsahuje PTBR jednotlivých tabulek stránek druhé úrovně. K nalezenému PTBR se přičte offset1 a na tomto místě v tabulce stránek druhé úrovně nalezneme záznam o rámci, do kterého je stránka namapována. Nakonec k adrese rámce přičteme offset2 a získáme přesnou adresu ve fyzické paměti.



- Tabulky stránek 2. úrovně existují jen pro *platné položky* "super" tabulky stránek (to je ten hlavní fígl). Pokud máme jenom 10 % platných vstupů super tabulky stránek, potom pro všechny tabulky stránek potřebujeme přibližně 1/10 prostoru stránek jednoduchého stránkovaní!
- Při víceúrovňovém stránkování jsou jednotlivé tabulky menší než při jednoúrovňovém a mohou být různě rozmístěny po paměti
- Dvouúrovňové stránkování vyžaduje 3 přístupy do paměti

Inverzní tabulka stránek

Vypadá takto:

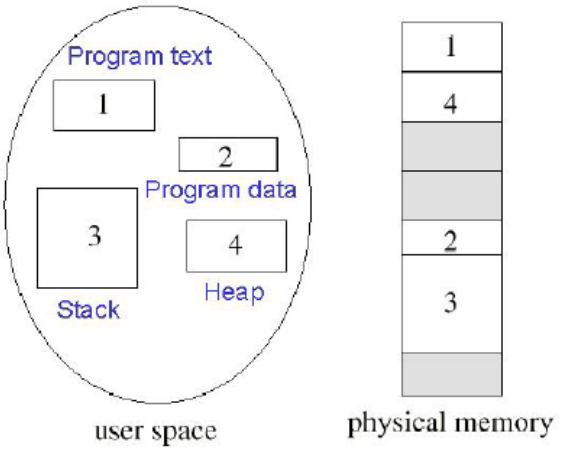


- Jeden vstup pro každý rámec
- Hledání pomocí hashovací tabulky nebo hashovací funkce

- Vzhledem k tomu, že více virtuálních adres může být mapováno do stejného vstupu, je pro nalezení správné položky použit "chaining" (záznamy, které mají stejný kontrolní součet (hash), jsou zřetězeny)

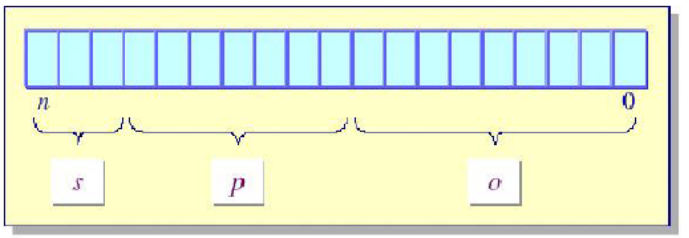
Segmentace

Segmentace se původně ke stránkování nevztahuje, ale v mnoha systémech je se stránkováním spojena. Každý uživatelský prostor může totiž být rozdělen na různé segmenty. Tyto segmenty jsou pak určitým způsobem uloženy ve fyzické paměti.



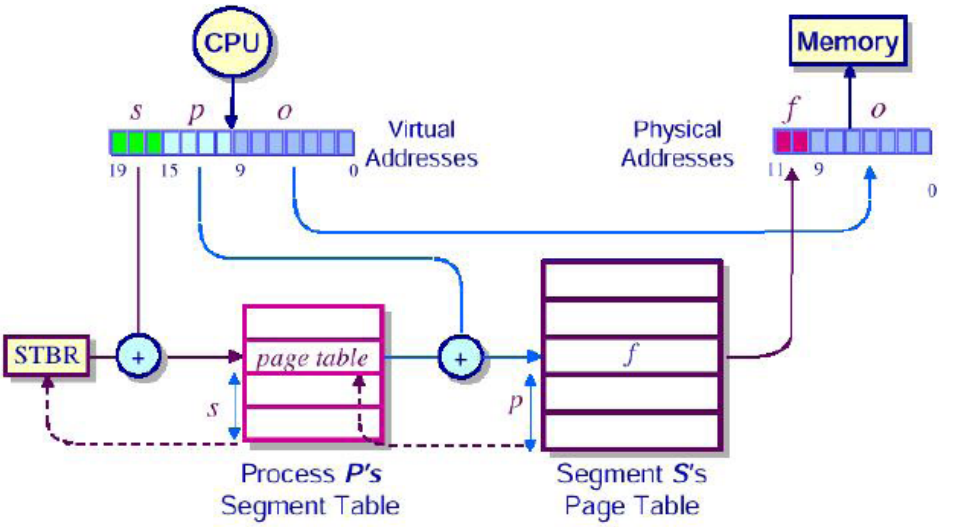
Nakonec jsou jednotlivé segmenty implementovány jako stránkový virtuální adresní prostor. Virtuální adresa je rozdělena na 3 části: číslo segmentu, číslo stránky a offset.

- s — segment number
- p — page number (p_{max} pages)
- o — page offset (o_{max} bytes/pages)



$$\text{Virtual address} = (s \times p_{max} + p) \times o_{max} + o$$

Nejprve se v segmentové tabulce příslušného procesu na základě čísla segmentu najde PTBR tabulky stránek pro daný proces a segment. K němu se přičte číslo stránky a na této pozici v tabulce stránek nalezneme záznam s adresou hledného rámce. K této adrese přičteme offset a nalezeneme požadované místo ve fyzické paměti.



Fragmentace paměťového prostoru

- Externí fragmentace (typická pro segmentaci) - celý paměťový prostor je k dispozici (po splnění zrčitých požadavků), ale není souvisle obsazován
- Interní fragmentace (typická pro stránkování) - alokovaná část paměti může být větší, než je požadováno

Materiály

- Wikipedia - Virtuální paměť (*http://cs.wikipedia.org/wiki/Virtu%C3%A1ln%C3%AD_pam%C4%9B%C5%A5*)
- X36APS, přednáška 8

Hlavní paměť s prokládaným cyklem

Paměť s prokládanými cykly (interleaved memory) se používá ve výkonných počítačích ke snížení cyklu paměti. Hlavní paměť je rozdělená do několika bloků (bank) pracujících samostatně a schopných provádět nezávisle čtecí nebo zápisový cyklus. Jestliže procesor komunikuje střídavě s různými paměťovými bloky, mohou přenosy probíhat paralelně a komunikace mezi pamětí a procesorem se celkově zrychlí. Současně lze číst z n-paměťových bank naráz, rychlost čtení je tedy cca n-násobná. Rychlost čtení je také ovlivněna *střídou* (co to je viz dále…)

Příklad: Uvažujme paměť se 4 bankami s prokládaným cyklem obsahující vektor V se střídou (stride) $s = 7$.

1. Naznačte umístění prvních čtyř složek vektoru v paměťových bankách, pokud je složka $V[0]$ umístěna v modulu č. 2.
2. Napište formuli určující číslo paměťového modulu, ve kterém bude umístěna složka $V[i]$ (moduly jsou číslovány od 0).