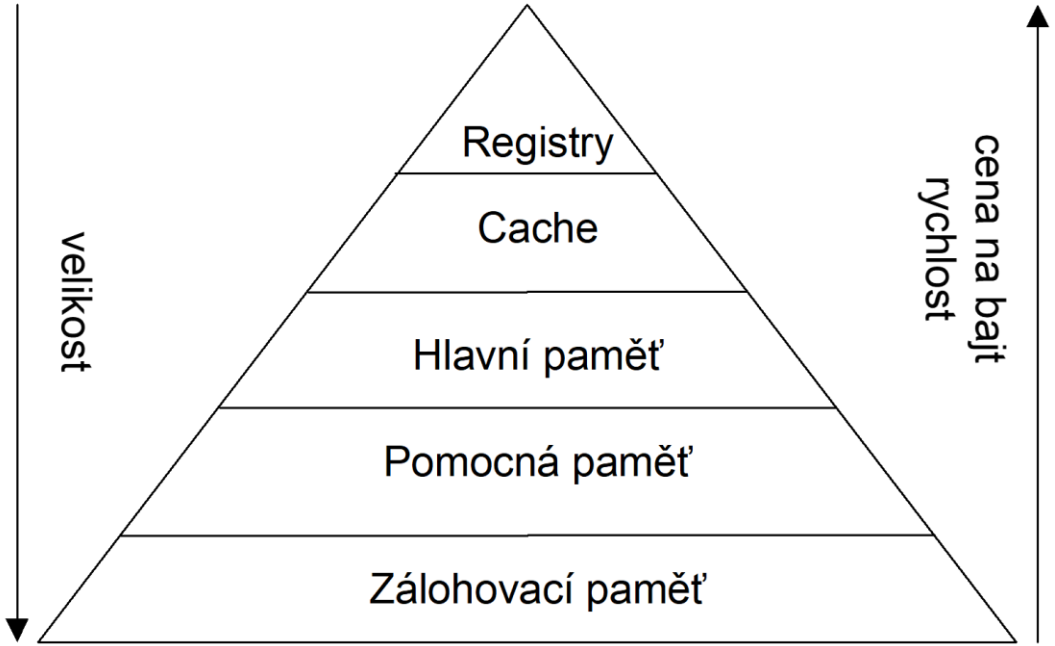
OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA

Správa paměti

Ing. Pavel Smolka, Ph.D.

Hierarchie paměti



Požadavky na správu paměti

- Potřeba relokace programů
  - Programátor nemůže vědět, ze které části paměti bude jeho program procesorem interpretován
  - Odkládání (*swapping*) umožňuje OS udržovat velkou sadu rozpracovaných procesů
  - Procesu může být dynamicky přidělována jiná (aspoň zdánlivě souvislá) oblast paměti – relokace
- Odkazy na paměť uvedené v programu (v LAP) se musí dynamicky překládat na skutečné adresy ve FAP
- Potřeba ochrany paměti
  - Procesy nesmí být schopné odkazovat se bez povolení na paměťová místa přidělená jiným procesům nebo OS
  - Relokace neumožňuje, aby se adresy kontrolovaly během překladu či sestavování
  - Odkazy na adresy se musí kontrolovat při běhu procesu hardwarem
    - softwarové kontroly příliš pomalé a drahé

Požadavky na správu paměti

- Logická organizace
  - Programy jsou sady sekcí s různými vlastnostmi
    - Sekce s instrukcemi jsou ,,*execute-only*“
    - Datové sekce jsou ,,*read-only*“ nebo ,,*read/write*“
    - Některé sekce jsou ,,soukromé“ (*private*), jiné jsou ,,veřejné” (*public*)
  - OS a HW musí podporovat práci se sekcemi tak, aby se dosáhlo požadované ochrany a sdílení

- Požadavky na sdílení
  - Více procesů může sdílet společné úseky (sdílené struktury), aniž by tím docházelo k narušení ochrany paměti
    - Sdílený přístup ke společné datové struktuře je efektivnější než udržování konzistence násobných kopií vlastněných jednotlivými procesy

Základní fakta

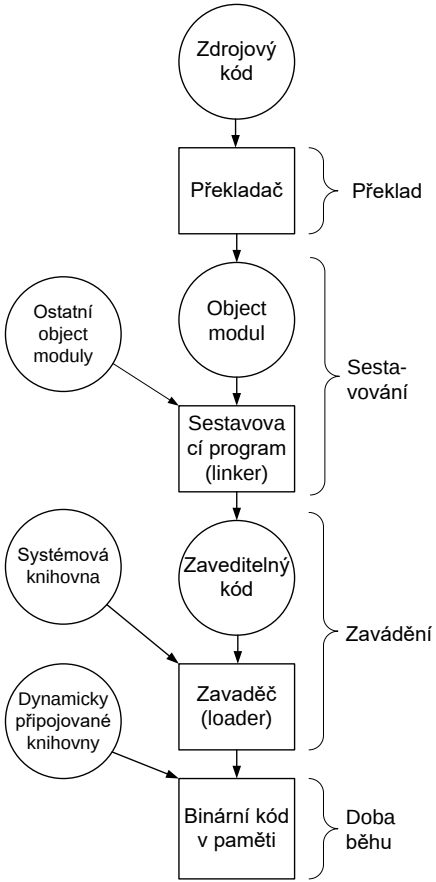
- Procesor umí vykonat pouze instrukci, která je uložena ve vnitřní paměti a jejíž data jsou také uložena v vnitřní paměti počítače
- FAP – fyzická adresa paměti je adresa vnitřní paměti počítače
  - Rozsah FAP je dán architekturou počítače
    - kolik „drátů“ má adresní sekce sběrnice
  - Velikost vnitřní paměti bývá i menší než je rozsah FAP
    - záleží na tom, kolik peněz jsme ochotni za paměť dát a kolik paměti se fyzicky do počítače vejde, případně na paměťových řadičích apod.
- LAP – logická adresa paměti je adresa hypotetické paměti, jejíž obsah je buď uložen ve fyzické paměti, nebo ve vnější paměti (pevný disk), případně nebyla ještě použita, a proto ani neexistuje
  - Rozsah LAP je dán architekturou procesoru
    - „šířkou“ dat, s nimiž je CPU schopen pracovat

Základní fakta

- Pro běh procesu je nutné, aby program, který ho řídí byl umístěn v operační paměti
- Program se přetváří do formy schopné interpretace procesorem ve více krocích
  - V jistém okamžiku se musí rozhodnout, **kde** budou kód a data umístěna ve vnitřní paměti
  - Cíl: **Vazba adres** instrukcí a dat na skutečné adresy v operační paměti
- Vnitřní paměť uchovává data a programy právě běžící, popř. připravené procesy
  - Roli dlouhodobé paměti programů a dat plní vnější paměť
- Správa paměti je nutně předmětem činnosti OS
  - Aplikační procesy nemají přístup k prostředkům pro řízení paměti
    - Privilegované akce
  - Nelze ji svěřit na aplikačnímu programování
    - Bylo by to velmi neefektivní a nebezpečné

Vázání adres (*Address Binding*)

- Při překladu
  - Je-li umístění ve FAP známé před překladem kompilátor může generovat absolutní kód
    - Použitelné jen v uzavřených systémech
    - Při přemístění je nutný nový překlad
- Při zavádění programu
  - Použitelné, je-li umístění ve FAP je známé při sestavování nebo při zavádění programu
  - Překladač generuje sestavovatelný modul (*object module*), cílovým adresním prostorem je LAP
  - Sestavovací program (*linker*, *linkage editor*) generuje zaváděcí modul (*load module*) a doplňuje do něj knihovní moduly
    - Vazbu na adresy FAP provede až zavaděč
  - Zaváděcí modul může být absolutní nebo přemístitelný
    - Absolutní modul zavádí absolutní zavaděč
    - Např. části JOS musí být absolutní
    - Přemístitelný modul zavádí přemísťující zavaděč



Vázání adres za běhu

- Za běhu procesu
  - Může-li proces měnit své umístění v paměti, vázání adres se musí odsunout až na dobu běhu
  - Cílovým prostorem pro sestavování je LAP
  - Program se zavede do FAP ve tvaru pro LAP
    - Např. začíná od adresy 0
- Musí být k dispozici hardwarová podpora
  - MMU
    - Např. bázové a limitní registry s privilegovanými přístupy
  - Často se kombinuje se základní ochranou paměti
  - Při vázání za běhu se LAP a FAP liší
  - Dochází k dynamickému překladu adres DAT – *Dynamic Address Translation*
- Používáno v prakticky všech moderních systémech

Dynamické zavádění (*Dynamic Loading*)

OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA

- Podprogram (*routine*) se zavádí, až když je volán
  - Na disku se uchovává jako přemístitelný modul
  - Volající program nejprve prověří, zda volaný podprogram je již zaveden v paměti
  - Pokud ne zavádí ho a koriguje stav zavedených programů
  - Dosahuje se lepšího využití prostoru ve FAP
    - nepoužívané moduly se nikdy nezavádí
  - Užitečná technika v případech, kdy se velkými programovými moduly řídí jen zřídka se vyskytující alternativy
    - např. při spuštění programu se uživatel rozhodne, zda bude dělat analýzu nebo syntézu
  - Běh programu nepotřebuje žádnou speciální podporu od operačního systému
    - Programátorské řízení dynamického zavádění však může být složité

Dynamické sestavování (*Dynamic Linking*)

- Základ pro dynamicky vázané knihovny, DLL
- Statické sestavování
  - Jednorázová činnost sestavujícího programu, kdy se vytvoří zaváděcí modul
- Dynamické sestavování
  - Technika vhodná zejména pro připojování knihovních podprogramů
  - Připojování podprogramu – rutiny – (tj. sestavení a zavedení, včetně vázání adres) je odloženo až na dobu běhu (*run-time*)
  - Pro vyhledání příslušných podprogramů se používá malý kousek kódu – ***stub*** (=pahýl, pařez, zbytek, …), který za běhu vyvolá přerušení
  - OS zkontroluje, zda je rutina zavedena v operační paměti a pokud ne, zprostředkuje její zavedení
  - „Stub" sám sebe nahradí instrukcí s adresou volané rutiny, a tím jí předá řízení (provede ji)
    - Samo o sobě obtížné bez podpory OS, neboť kód je chráněn proti přepisu a zde se předpokládá „sebemodifikace" kódu
    - Náhrada se děje jen jednou; příště se už rutina volá přímo

Strategie přidělování paměti

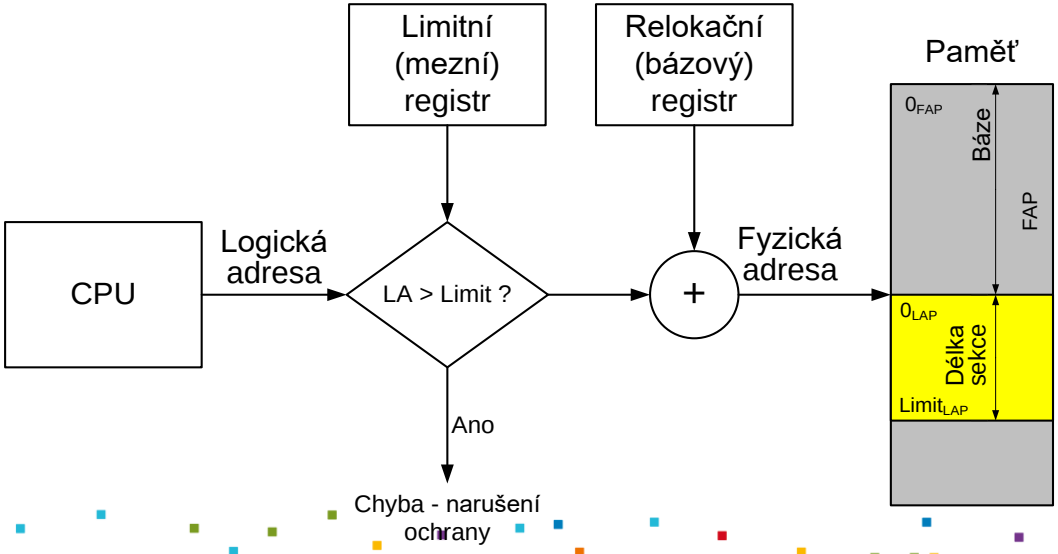
- Spojité přidělování
  - Přidělování veškeré volné paměti
  - Přidělování pevných bloků paměti
  - Přidělování bloků paměti proměnné velikosti
- Nespojité přidělování
  - Segmentace paměti
  - Stránkování paměti
  - Stránkování na žádost (demand paging)
  - Segmentace se stránkováním na žádost

Historie správy paměti

- První počítače bez správy paměti – přímý přístup do paměti
- Výhody systému bez správy paměti
  - Rychlost přístupu do paměti
  - Jednoduchost implementace
  - Lze používat i bez operačního systému – robustnost
- Nevýhody systému bez správy paměti
  - Nelze kontrolovat přístup do paměti
  - Omezení paměti vlastnostmi HW
- Použití
  - První počítače
  - Osmibitové počítače (procesory Intel 8080, Z80, apod.) – 8-mi bitová datová sběrnice, 16-ti bitová adresová sběrnice, možnost využít maximálně 64 kB paměti
  - Řídicí počítače – embedded (v současné době již jen ty nejjednodušší řídicí počítače)

Přidělování souvislých oblastí

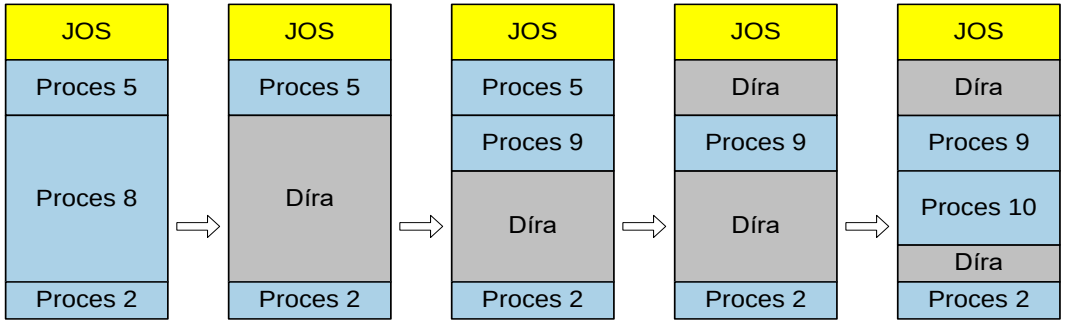
- Operační paměť se dělí do dvou typů sekcí
  - sekce pro rezidentní JOS
    - obvykle na počátku nebo na konci FAP (hardwarově závislé)
  - sekce pro uživatelské procesy
- Ochrana paměti při přidělování sekce
  - Schéma s limitním (mezním) a relokačním (bázovým) registrem



Dynamické přidělování více souvislých sekcí

- Díra – blok dostupné paměti
  - Díry jsou roztroušeny po FAP
  - Procesu se přiděluje díra, která jeho požadavek uspokojí
  - Evidenci o sekcích udržuje JOS
  - Kde přidělit oblast délky *n*, když je volná paměť rozmístěna ve více souvislých nesousedních sekcích?
    - First fit – první volná oblast dostatečné velikosti – rychlé, nejčastější
    - Best fit – nejmenší volná oblast dostatečné velikosti – neplýtvá velkými děrami
    - Worst fit – největší volná oblast – zanechává velké volné díry

Nákladné – nutno prohledat celý seznam volných děr

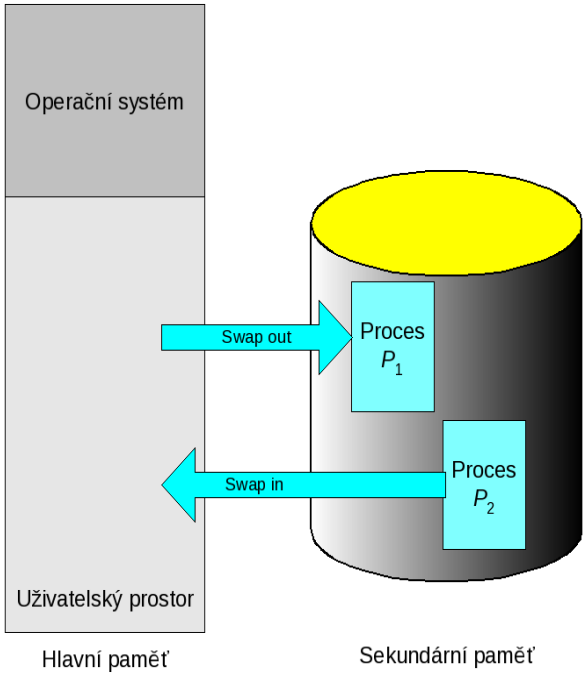


Problém fragmentace

- **Externí** (vnější) **fragmentace**
  - Celkové množství volné paměti je sice dostatečné, aby uspokojilo požadavek procesu, avšak prostor není souvislý, takže ho nelze přidělit
- **Interní** (vnitřní) **fragmentace**
  - Přidělená díra v paměti je o málo větší než potřebná, avšak zbytek nelze využít
- Redukce externí fragmentace pomocí setřásání
  - Přesouvají se obsahy úseků paměti s cílem vytvořit (jeden) velký souvislý volný blok
  - Použitelné *pouze* při dynamické relokaci
    - Při absolutních adresách v paměti by bylo nutno přepočítat a upravit všechny adresy v instrukcích
  - Problém s I/O:
    - S vyrovnávacími pamětmi plněnými z periferií (zejména přes DMA) nelze autonomně hýbat, umisťují se proto do prostoru JOS

Výměny, odkládání (*Swapping*)

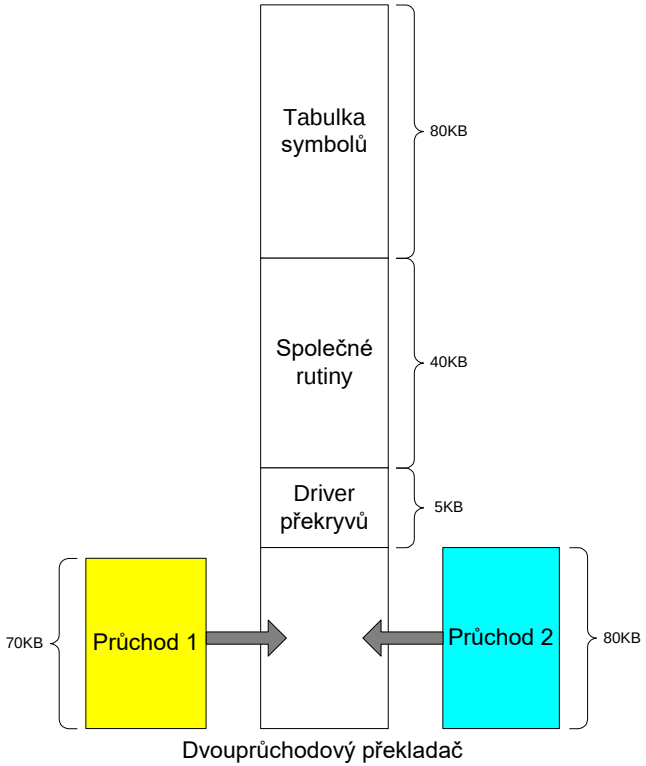
- Sekce FAP přidělená procesu je vyměňovaná mezi vnitřní a vnější pamětí oběma směry
  - Potřebujeme rychlou sekundární paměť s přímým přístupem
- *Swap out, swap in* (*roll out, roll in*)
  - výpis na disk, načtení z disku
- Trvání výměn je z podstatné části tvořena dobou přenosu
  - je úměrná objemu vyměňované paměti
- Princip používaný v mnoha OS
  - pokud nepodporují virtualizaci
- Základ virtualizace



Odkládá se celý proces!

Správa paměti s překryvy (*Overlays*)

- V operační paměti se uchovávají pouze ty instrukce a data, která jsou v daném časovém intervalu potřebná
  - První technika umožňující „zvětšit“ paměť. Tedy, přidělený paměťový prostor je menší než je souhrn potřeb procesu.
- Vyvolávání překryvů je navrženo uživatelem
  - od operačního systému se požaduje jen minimální podpora
  - programátorský návrh překryvné struktury může být složitý



Dvouprůchodový překladač

Virtualizace paměti

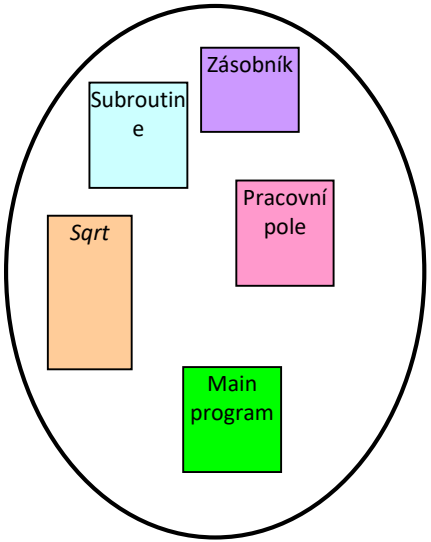
- Požadavek na větší chráněnou paměť, kterou má na starosti systém
  - tedy někdo jiný než programátor
- Řešením je princip virtuální logické paměti, která se „nějakým“ způsobem ukládá v fyzické paměti počítače
- 1959-1962 první počítač s virtuální pamětí Atlas Computer v Manchesteru
  - velikost paměti byla 576 kB
  - implementováno stránkování ➡
- 1961 - Burroughs vytvořil první komerční počítač B5000, který využíval k virtualizaci segmenty
- Intel
  - 1978 procesor 8086 – základ prvního PC – jednoduché segmenty
  - 1982 procesor 80286 – protected mode – skutečná segmentace
  - 1985 procesor 80386 – plná virtualizace – stránkování i segmenty

Jednoduché segmenty – Intel 8086

- Procesor 8086 má 16 bitů datovou sběrnici a navenek 20 bitů adresní.
  - 20 bitů je ale problém. Co s tím?
- Řešením jsou jednoduché segmenty
  - Adresa je tvořena adresou segmentu 16-bitů a adresou uvnitř segmentu (offset) 16-bitů.
  - Výsledek se složí podle pevného pravidla:

    (segment<<4)+ offset
- Nejedná se o virtualizaci, pouze o prostředek, jak používat větší paměť než nám dovoluje systém
  - Někdy se hovoří o mapování, tj. malý logický adresní prostor se promítá do většího FAP
- Dva způsoby adresace
  - uvnitř segmentu, krátký odkaz (*near pointer*) obsahuje pouze odkaz uvnitř segmentu, segment je určen nastavením segmentových registrů
  - mezi segmenty, dlouhý odkaz (*far pointer*) obsahuje popis segmentu a odkaz uvnitř segmentu

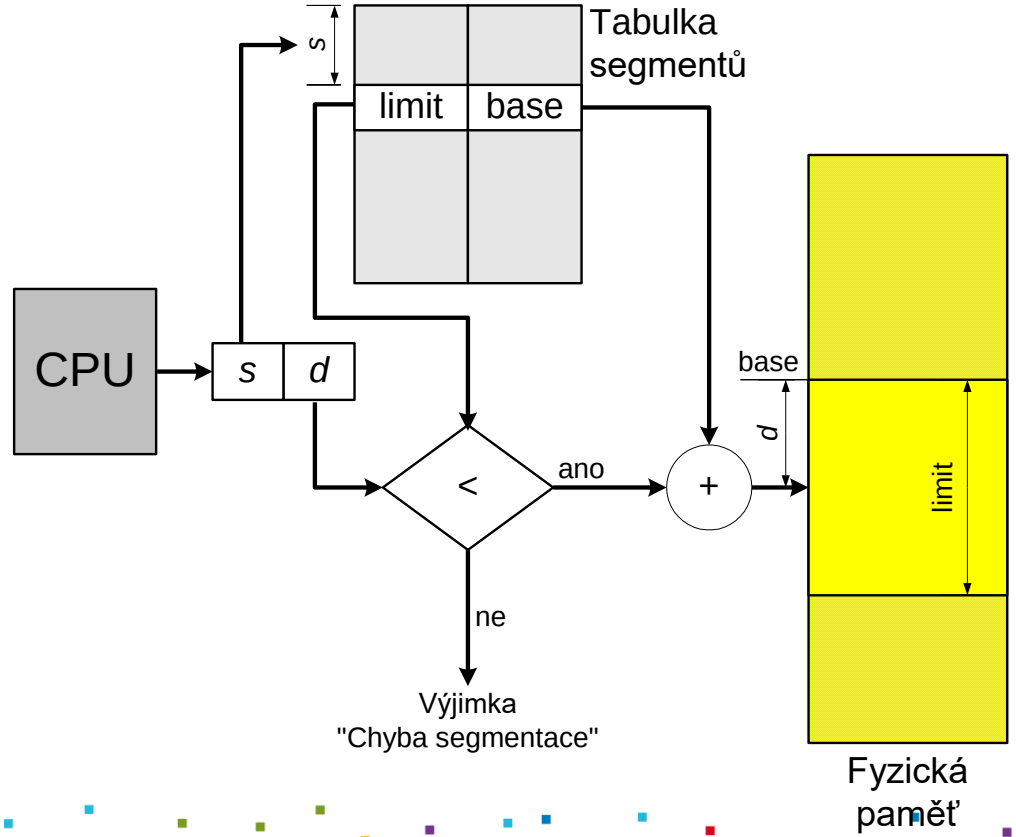
Segmentace – obecný princip



- Podpora uživatelského pohledu na LAP
  - Program je kolekce segmentů
  - Každý segment má svůj logický význam:
    hlavní program, procedura, funkce,
    objekt a jeho metoda, proměnné, pole, ...
- Základní úkol – převést adresu typu (segment, offset) na adresu FAP

- Tabulka segmentů – Segment table, ST
  - Je uložena v paměti
  - Zobrazení 2-D (segment, offset) LAP do 1-D (adresa) FAP
  - Položka ST:
    - base – počáteční adresa umístění segmentu ve FAP, limit – délka segmentu
  - Segment-table base register (STBR) – umístění ST v paměti
  - Segment-table length register (STLR) – počet segmentů procesu

Hardwarová podpora segmentace

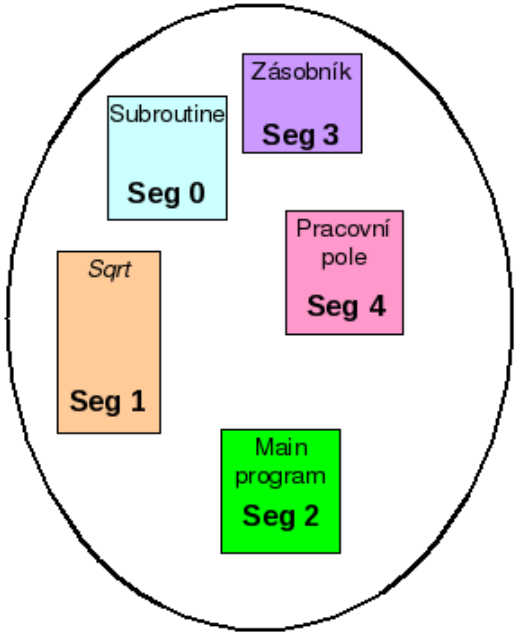


Segmentace

OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA

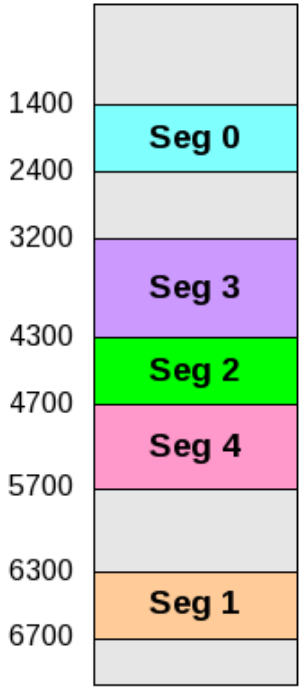
- Výhody segmentace
  - Segment má definovanou délku
  - Lze detekovat přístup mimo segment, který způsobí chybu segmentace – výjimku „*segmentation fault*“
  - Lze nastavovat práva k přístupu do segmentu
    - Operační systém požívá větší ochrany než aplikační proces
    - Uživatel nemůže ohrozit operační systém
  - Lze pohybovat s daty i programem v fyzické paměti, aniž by si toho někdo mohl všimnout (posun počátku segmentu je pro aplikační proces neviditelný a nedetekovatelný)
- Nevýhody segmentace
  - Problém jak umísťovat segmenty do paměti. Segmenty mají různé délky. Při běhu více procesů se segmenty ruší a vznikají nové.
  - Režie při přístupu do paměti
    - převod na lineární adresu se opírá o tabulku segmentů a ta je také v paměti – nutné 2 přístupy

Segmentace – příklad



Tabulka segmentů

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



Fyzická paměť

- Alokace segmentů v paměti je netriviální úloha
  - Segmenty mají proměnnou délku
  - Problém s externí fragmentací

Stránkování

OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA

- Jiným řešením virtuální paměti je stránkování
- Stránkování odstraňuje základní problém segmentace – různé velikosti segmentů
- Stránky mají pevně stanovenou velikost, danou architekturou procesoru
- Fragmentace – neúplné zaplnění paměti je možné pouze uvnitř stránky a jeho velikost je tedy omezena velikostí stránky

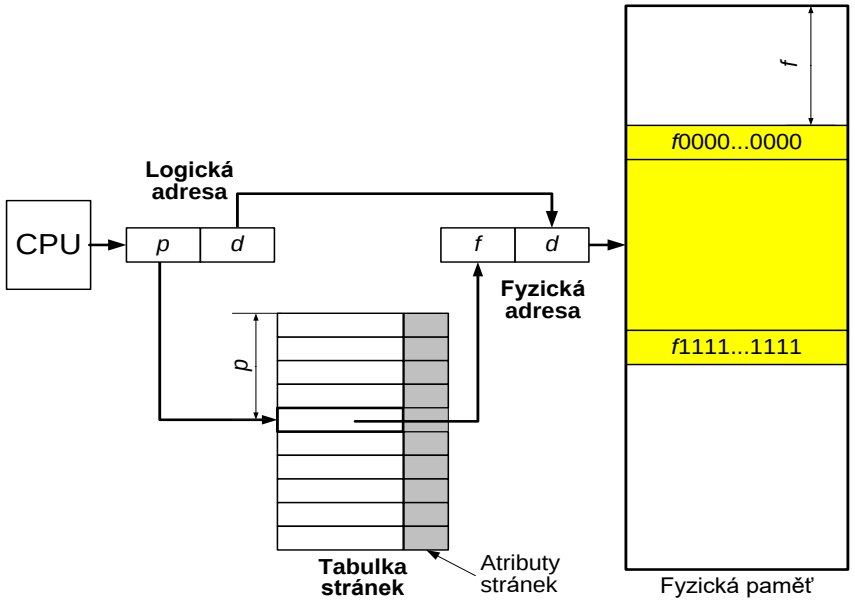
Stránkování

OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA

- Souvislý LAP procesu není zobrazován jako jediná souvislá oblast FAP
  - Zobrazuje se do po částech do sady volných sekcí FAP
- FAP se dělí na sekce zvané **rámce**
  - Pevná délka, zpravidla v celistvých mocninách 2 (512 až 8.192 B, nejčastěji 4KiB, ale někdy i 4 MiB)
- LAP se dělí na sekce zvané **stránky**
  - Pevná délka, shodná s délkou rámců
- OS udržuje seznam volných rámců
- Proces o délce *n* stránek se umístí do *n* rámců, které ale nemusí být v paměti za sebou
- Překlad logická adresa → fyzická adresa
  - PT = *Page Table*, **tabulka stránek**
- Vzniká vnitřní fragmentace
  - stránky nemusí být zcela zaplněny)

Stránkování – překlad adres

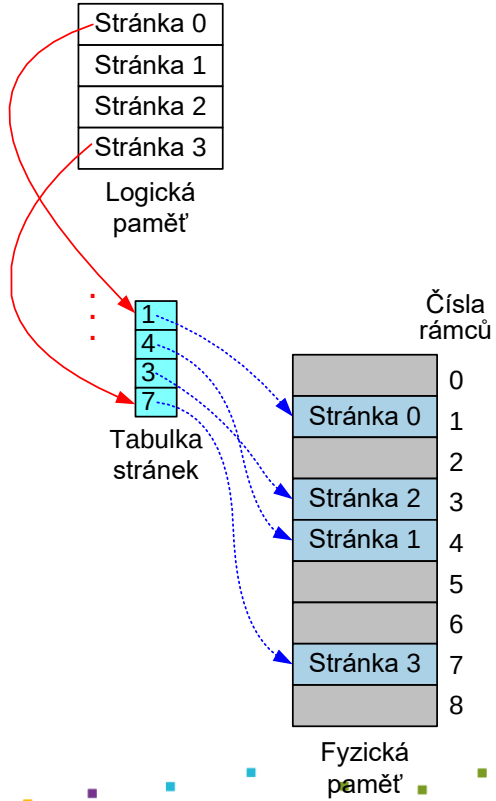
- Logická adresa (generovaná CPU) se dělí na
  - číslo stránky, *p*
    - index do tabulky stránek, obsahující bázové adresy rámců přidělených stránkám
  - offset ve stránce, *d*
    - relativní adresa (posunutí = *offset, displacement*) ve stránce/v rámci



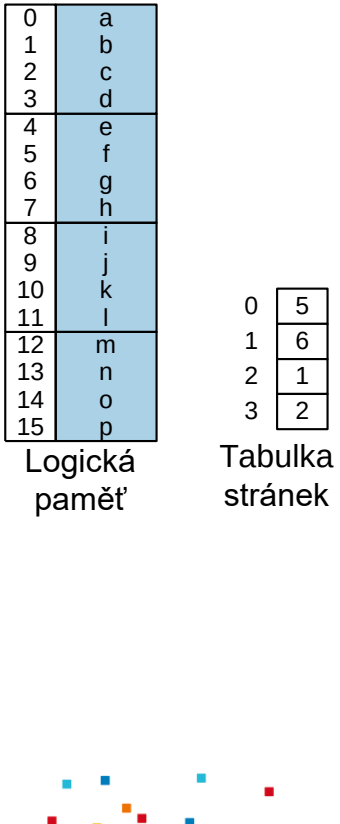
- Atributy stránek
  - Ochranné příznaky
    - *r/w* = *read/write*
  - Virtualizační příznaky
    - *v/i* = *valid/invalid* indikuje přítomnost stránky ve FAP
    - *a* = *accessed* značí, že stránka byla použita
    - *d* = *dirty* indikuje, že obsah stánky byl modifikován

Příklady stránkování

Příklad 1



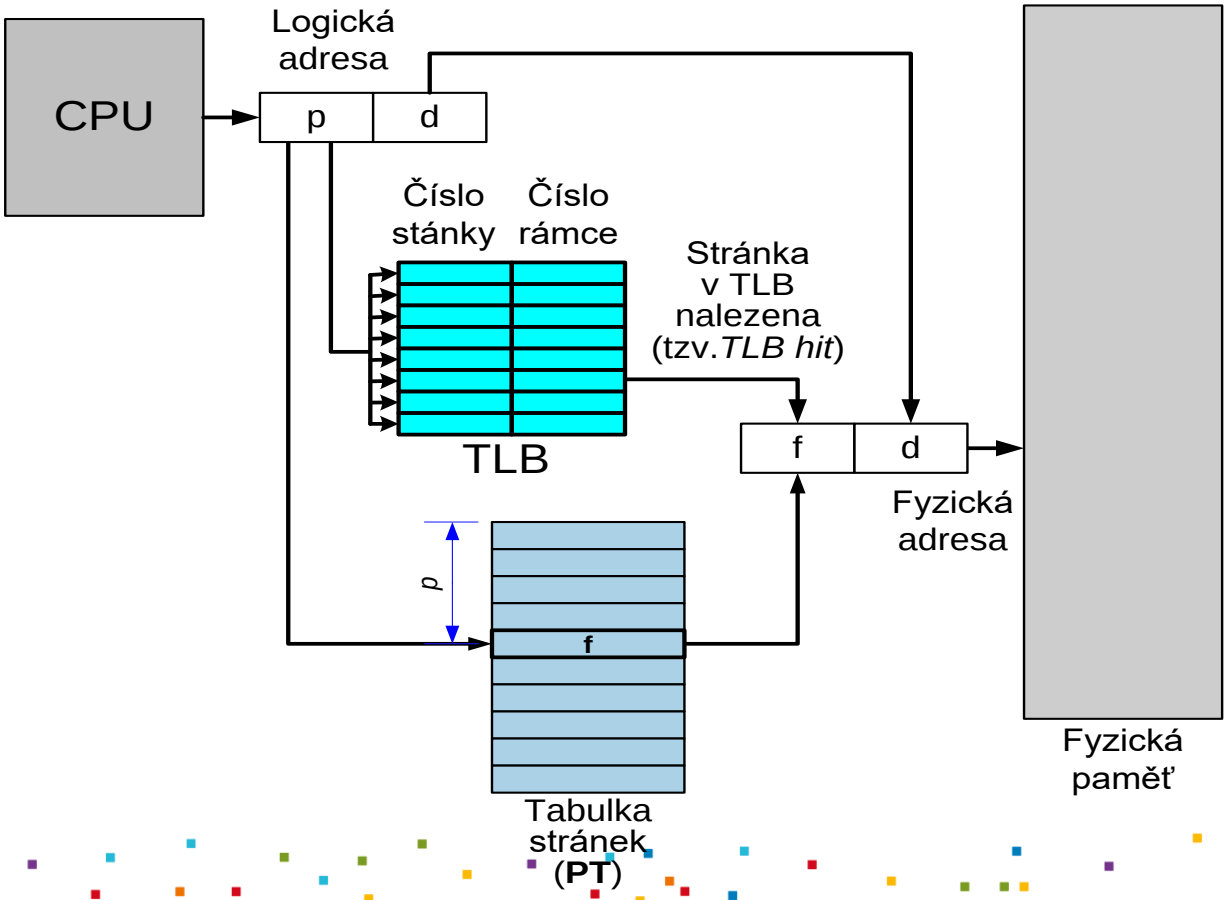
Příklad 2



Implementace tabulky stránek

- Tabulka stránek je uložena v hlavní paměti
  - Její začátek je odkazován registrem „*Page-table base register*“ (PTBR) a její délka je v registru „*Page-table length register*“ (PTLR)
    - Význam PTLR ukážeme později
- Každý přístup do paměti vyžaduje přístupy dva:
  - přístup do tabulky stránek a vlastní přístup do paměti pro údaj/instrukci – časově náročné
- Problém dvojího přístupu lze řešit speciální rychlou cache pamětí s využitím principu lokality
  - asociativní paměť, *Translation Look-aside Buffer* (TLB)
- TLB je asociativní paměť „adresovaná“ částí obsahu
  - relativně malá kapacita, vysoká rychlost
  - Překlad $p \rightarrow f$: Jestliže se p nachází v asociativní paměti, z paměti se získá i hodnota f, jinak se f vyhledá v tabulce stránek (PT) v paměti a obsah TLB se zaktualizuje (HW nebo SW prostředky)

Stránkování s TLB



Doba přístupu do paměti s TLB

- Skutečná přístupová doba – *Effective Access Time* (**EAT**)
  - Přístupová doba do fyzické paměti = t
  - Přístup to TLB = ε
  - „Hit ratio“ α – pravděpodobnost, že se stránka nalezne v TLB

$$EAT = (\varepsilon + t)\alpha + (\varepsilon + 2t)(1-\alpha)$$
$$= (2-\alpha)t + \varepsilon$$

Příklad pro t = 100 *ns*

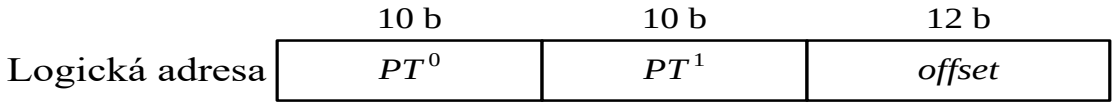
Jen *PT* bez *TLB*		*EAT* = 200 *ns*	Snížení rychlosti na polovinu oproti případu bez stránkování
ε = 10 *ns*	α = 60 %	*EAT* = 150 *ns*	*TLB* způsobila významné zrychlení průměrného přístupu do paměti
ε = 10 *ns*	α = 80 %	*EAT* = 130 *ns*	
ε = 10 *ns*	α = 98 %	*EAT* = 112 *ns*	

Struktura stránkovacích tabulek (PT)

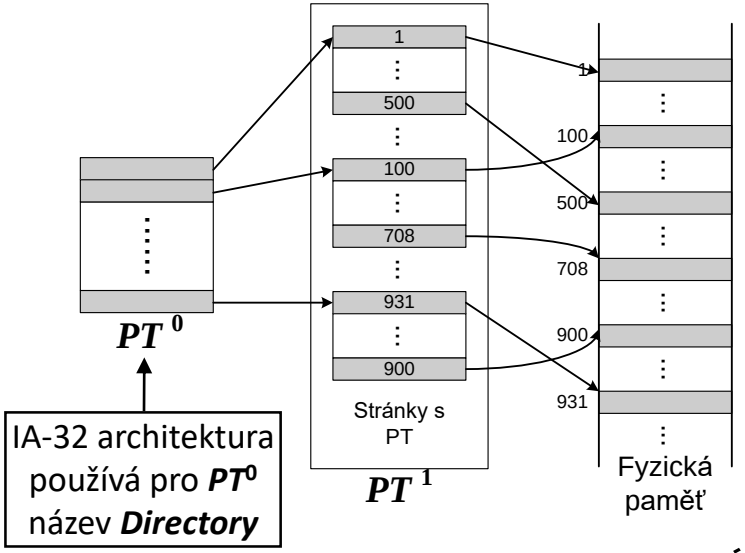
- Problém s velikostí PT
  - Každý proces má svoji PT
  - 32-bitový LAP, 4 KiB stránky → PT má až 4 MiB
    - musí být stále v hlavní paměti
- Hierarchické stránkování
  - Zobrazování LAP se dělí mezi více PT
  - Pro 32-bitový LAP typicky dvouúrovňové PT
  - $\boldsymbol{PT^0}$ obsahuje definice (odkazy) vlastních tabulek $\boldsymbol{PT^1}$
  - „Pravé“ tabulky stránek $\boldsymbol{PT^1}$ podléhají stránkování, takže v RAM lze zobrazovat jen skutečně využité části/stránky s „pravými” PT
- Hašovaná PT
  - Náhrada přímého indexování číslem ***p*** v PT hašovací funkcí *hash*(*p*)
- Invertovaná PT
  - Jediná PT pro všechny koexistující procesy
  - Počet položek je dán počtem fyzických rámců
  - Vyhledávání pomocí hašovací funkce *hash*(*pid, p*)

Dvouúrovňové stránkování – příklad (1)

- 32-bitový stroj se stránkou o velikosti 4 KB
  - Např. IA-32 (Pentium): stránka 4 KiB, tj. 12 bitů adresy ve stránce
  - 20 bitů na číslo stránky → 2^{20} stránek
  - Bude-li položka PT mít 4 byty, pak celá PT bude mít 4 MiB
- Východisko: Samotná PT je stránkovaná
- Logická adresa 32 bitů
  - Dělí se na 20 bitů čísla stránky a 12 bitů adresy ve stránce (offset)
  - Číslo stránky se dále dělí na
    - Index do ***PT*^{0}** – tzv. „**vnější PT**“ – 10 bitů (p_1) a
    - Offset v ***PT*^{1}** – 10 bitů (p_2)

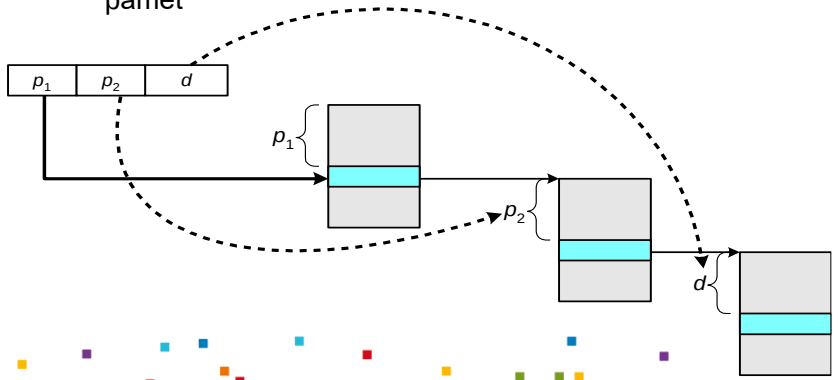


Dvouúrovňové stránkování – příklad (2)



Struktura dvouúrovňového stránkování

Překlad LA → FA



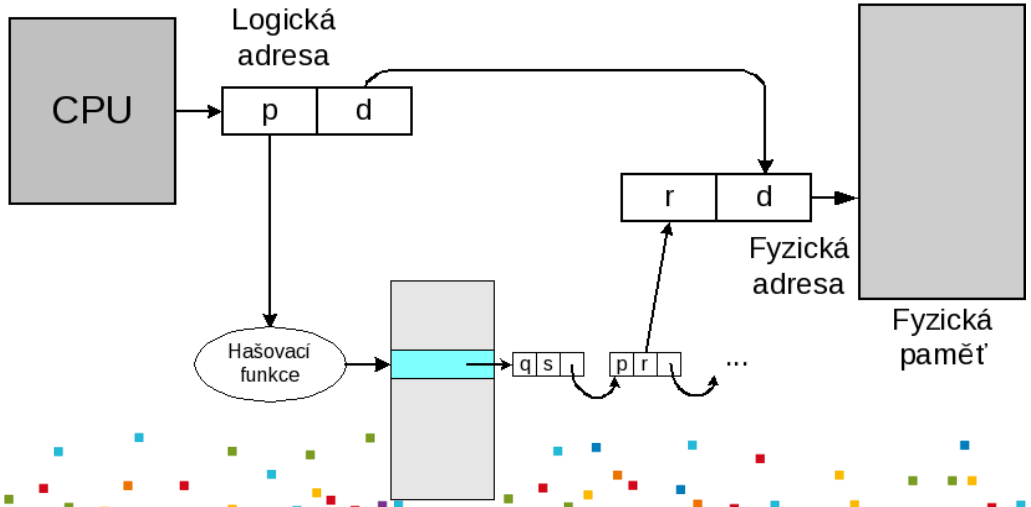
Víceúrovňové stránkování

OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA

- 64-bitový procesor se stránkou o velikosti 8 KiB
  - 51 bitů na číslo stránky → 2 Peta (2048 Tera) stránek
- Základní problém víceúrovňového stránkování:
  - Každá další úroveň znamená další přístup do paměti a zpomalení
- UltraSparc – 64 bitů ~ 7 úrovní → neúnosné
- Linux – 64 bitů
  - Trik: používá pouze 43 bitů ostatní ignoruje
  - Velikost LAP je sice „pouhých“ 8 TB, ale zatím si nikdo nestěžoval
  - 3 úrovně tabulek po 10 bitech
  - 13 bitů offset ve stránce
  - ještě únosná režie

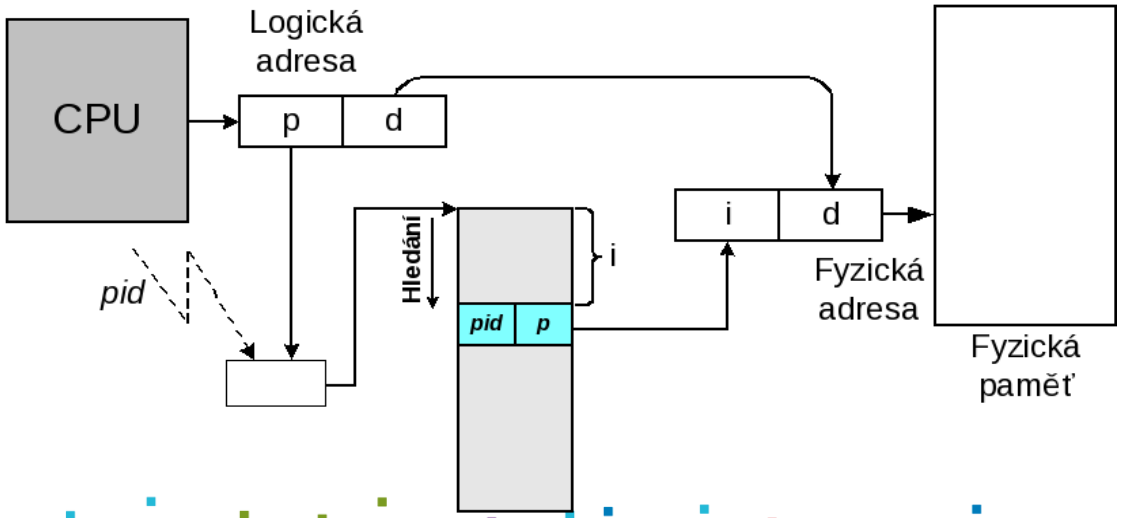
Hašovaná stránkovací tabulka

- Používá se zejména pro veliké adresní prostory
  - šíře adresy více než 32 bitů (např. Intel Itanium 64 bit)
- Číslo stránky se hašuje do tabulky stránek
  - ta obsahuje zřetězené prvky hašované do stejného místa
  - prvek řetězce = (číslo stránky, číslo fyzického rámce)
  - v řetězci se hledá číslo stránky a odtud se získá číslo rámce
  - Hašovací funkce i prohledávání řetězce by mělo být z důvodů efektivity realizováno v HW



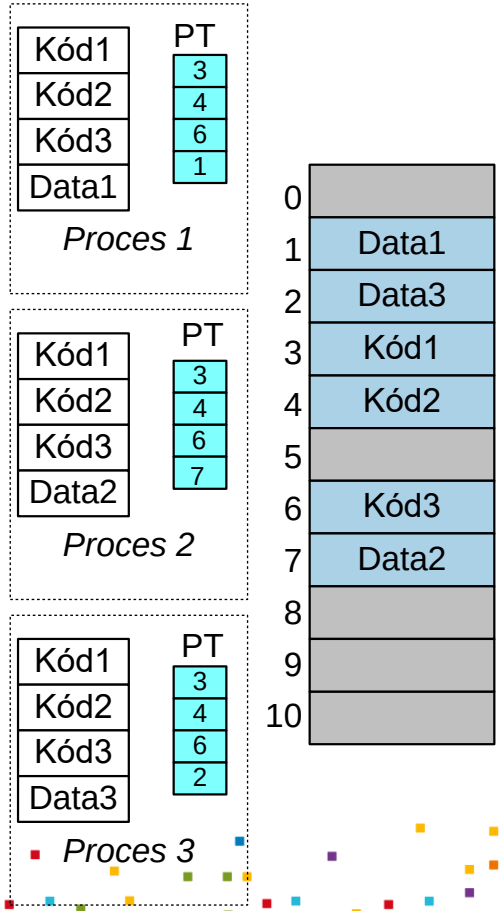
Invertovaná stránkovací tabulka (IPT)

- Index v invertované PT je *číslo rámce* (nikoliv č. stránky)
- V IPT jsou položky pro skutečně alokované rámce ve fyzické paměti (např. PowerPC)
  - jediná IPT pro všechny procesy
  - řádek obsahuje ***pid*** a ***p***
- Šetří se místo v paměti za cenu delšího prohledávání
  - Urychlit prohledávání lze hašováním



Sdílení stránek

- Sdílený kód
  - Jediná *read-only* kopie (reentrantního) kódu ve FAP sdílená více procesy
    - více instancí editoru, shellů, oknový systém
- Privátní kód a data
  - Každý proces si udržuje svoji vlastní kopii kódu a dat
  - Stránky s privátním kódem a daty mohou být kdekoliv v LAP
- Sdílená data
  - Potřebná pro implementaci meziprocesní komunikace
    - Např. mailboxy



Segmentace se stránkováním

- Kombinace obou výše uvedených metod
- Ponechává výhody segmentace, např. možnost přesného omezení paměťového prostoru
- Přináší výhodu jednoduchého umísťování segmentu do fyzické paměti. Ve fyzické paměti je pouze ta část segmentu, která se používá.

Segmentace se stránkováním

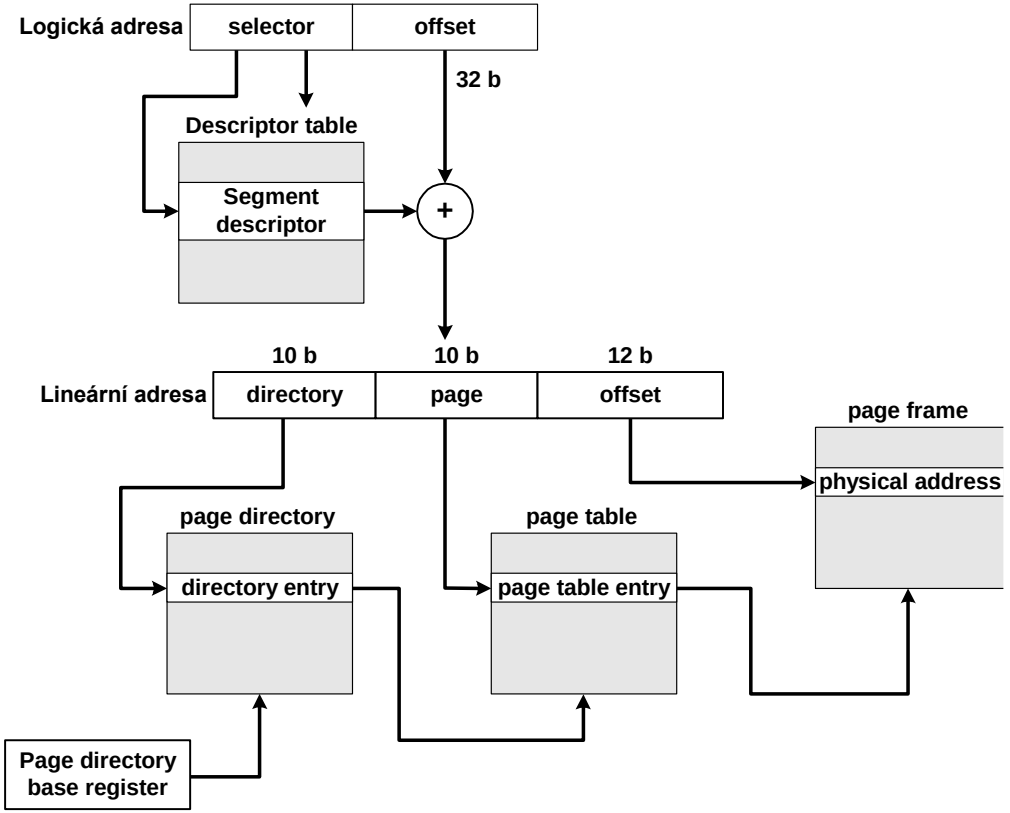
- Tabulka segmentů ST obsahuje místo báze segmentu
  - Buď adresu stránkovací tabulky PT
  - Nebo tzv. lineární adresu používanou pro přepočet na fyzickou adresu
- Segmentace se stránkováním je používána architekturou IA-32 (např. INTEL-Pentium)
- IA-32 transformace LAP → FAP
  - LAP lineární (4 GiB), transformace **identita**
    - používají pouze řadiče, apod.
  - LAP lineární (4 GiB), **stránkování**,
    - 1024 oblastí à 4 MiB, délka stránky 4 KiB, 1024 tabulek stránek, každá tabulka stránek má 1024 řádků
    - Používají implementace UNIX na INTEL-Pentium
  - LAP segmentovaný, **segmentace**
    - 16 Ki segmentů à 4 GiB ~ 64 TiB
  - LAP segmentovaný stránkovaný, **segmentace se stránkováním**
    - Segmentace vybírá části LAP, stránkování zobrazuje LAP do FAP
    - Používají Windows

Segmentace se stránkováním IA-32

OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA

- LAP: 16 K segmentů s délkou až 4 GB každý
- Dva logické podprostory (popisovač TI = 0 / 1)
  - 8 Ki privátních segmentů procesu – popisuje tabulka segmentů Local Description Table, LDT
  - 8 Ki segmentů sdílených procesy – popisuje tabulka segmentů Global Description Table, GDT
- Logická adresa = (popisovač segmentu, offset)
  - offset = 32-bitová adresa v segmentu, segment je stránkován
  - popisovač segmentu
    - 13 bitů *číslo segmentu*,
    - 1 bit *popisovač TI*,
    - 2 bity *úroveň ochrany*: segment jádra, ..., segment aplikace
    - práva *r/w* až na úrovni stránek
- Lineární adresní prostor uvnitř segmentu se stránkuje s použitím dvouúrovňového mechanismu stránkování
  - délka stránky 4 KiB, offset ve stránce 12 bitů, číslo stránky 2x10 bitů

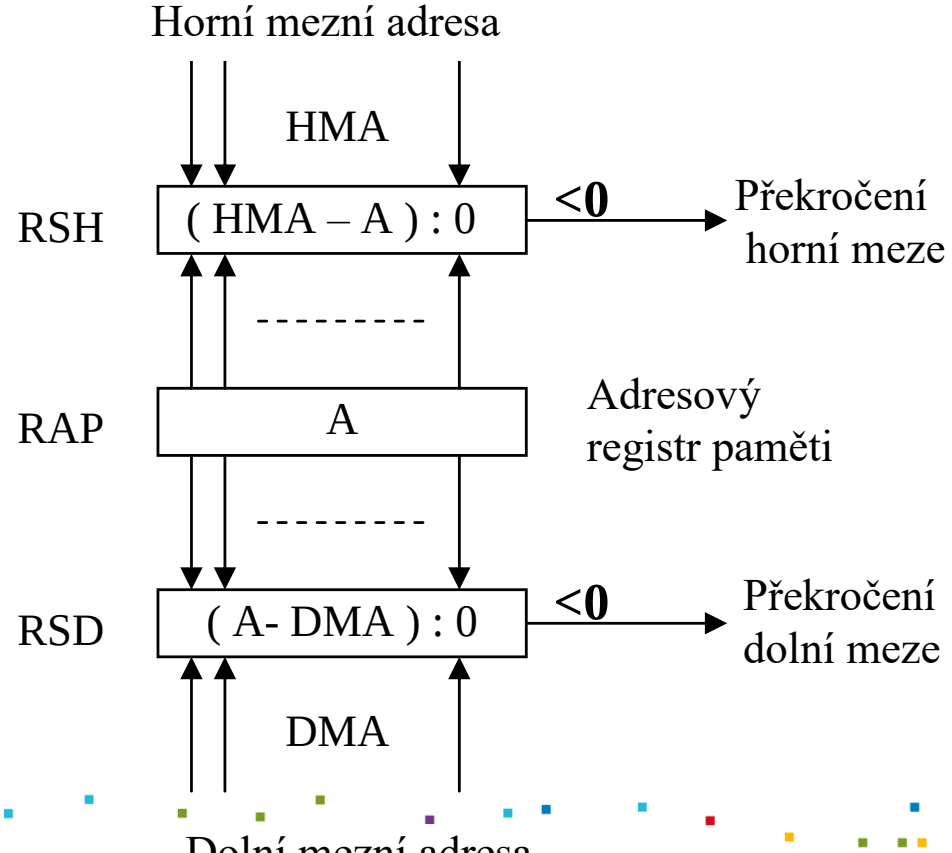
Segmentace se stránkováním Pentium



Ochrana paměti

- pomocí mezních registrů
  - Dva mezní registry udávají nejnižší a nejvyšší dostupnou adresu. Nastavuje je OS, když předává řízení procesu. Odkaz na paměť mimo způsobí vnitřní přerušení "porušení ochrany paměti". Nastavení mezních registrů musí být privilegovaná instrukce, jinak může program napsaný se špatným úmyslem číst nebo měnit paměťové oblasti jiných procesů.
- pomocí mechanismu zámků a klíčů
  - Paměť je rozdělena na stránky pevné velikosti (např. 4 KB). Každé stránce paměti je přiřazen zámek (= celé číslo). Procesor má speciální registr, který slouží jako klíč. Proces může používat pouze ty stránky paměti, které mají zámek nastavený na stejnou hodnotu, jako je klíč. OS může používat univerzální klíč číslo 0, který umožňuje přístup k libovolné stránce paměti

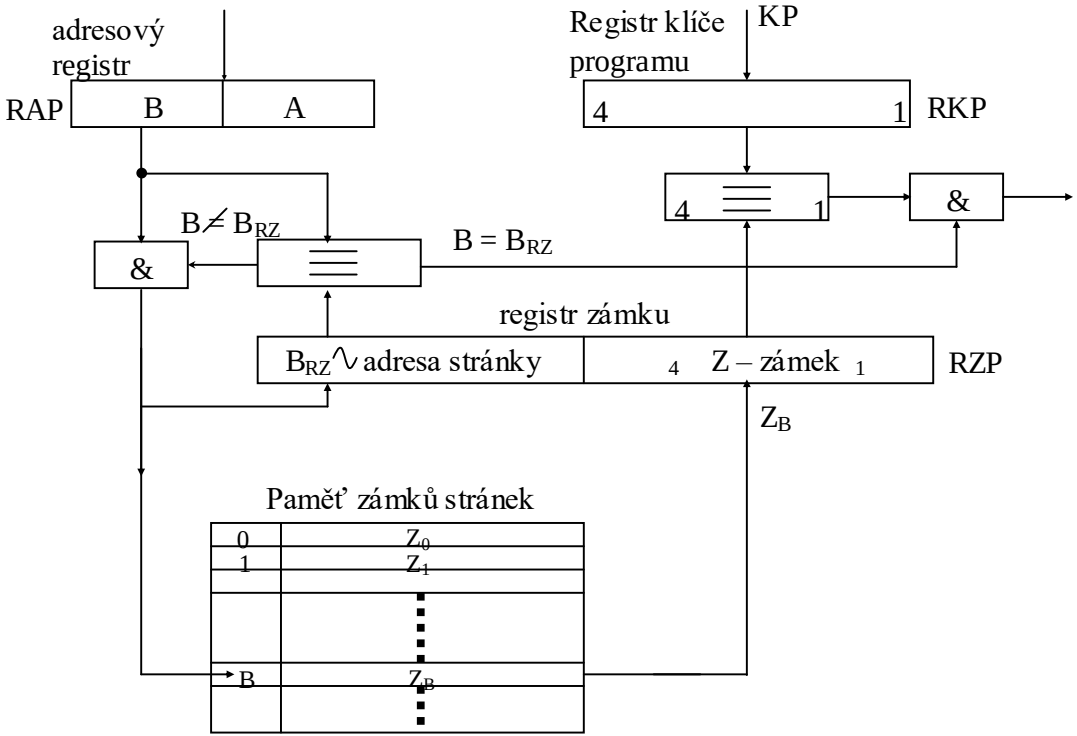
Ochrana mezními registry



Ochrana paměti

- znemožnění použít adresy mimo adresní prostor procesu -> mezní registr obsahuje max. číslo stránky pro příslušný proces
- procesu není dovoleno měnit obsah tabulky stránek

Ochrana paměti klíči



OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA

Děkuji za pozornost