Správa hlavní paměti

PB 152 ◇ Operační systémy

Jan Staudek
http://www.fi.muni.cz/usr/staudek/vyuka/

Verze: jaro 2017

Osnova přednášky

- Obecné principy správy hlavní paměti
- stránkování, *paging* hlavní paměti,
- segmentování, *segmentation* hlavní paměti
- stránkované segmentování hlavní paměti
- stránkování (segmentování) hlavní paměti na žádost, *demand paging / segmentation*,

Základnípojmy

Frame	A fixed length block of main memory.	Rámec
Stránka Page	A fixed length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.	
Segment Segment	A variable length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).	

Pokud neřekneme jinak, pak při výkladu správy hlavní paměti a virtualizaci paměti, tj. paměti přímo dostupné z CPU, používáme termíny paměť a hlavní paměti alternativně

Požadavky na správu paměti

- ✓ Roli dlouhodobé paměti programů a dat plní energeticky nezávislá vnější paměť (disky, . . .)
- ✓ Program řídicí běh procesu a zpracovávaná data musí být umístěny v (energeticky závislé) hlavní (operační, vnitřní) paměti, každé metodě a proměnné musí být přidělena adresa v této paměti
- ✓ Někdo musí určit kde v hlavní paměti budou program a data umístěny
- ✓ Proces netuší kde v hlavní paměti bude umístěn jeho program, dynamicky vytvářená data data mohou být v hlavní paměti kdekoliv, někdo musí svázat instrukce a data s konkrétními adresami paměti, address binding
- ✓ ,,Někdo'' v multitaskingových systémech = správa hlavní paměti
- ✓ Správa hlavní paměti musí zajistit, aby sdílení hlavní paměti procesy bylo transparentní a efektivní a přitom bezpečné
- ✓ Správa hlavní paměti je předmětem činnosti OS, nelze ji nechat na aplikačním programování, výkon jejich funkcí, by byl neefektivní spíše však škodlivý

Adresový prostor, AP

- generické chápání – vymezení adres cílových objektů
  - ✓ AP telefonního seznamu je interval čísel 0 – 999 999 999
  - ✓ AP nemusí být nutně numerický, Internetové domény jsou rovněž adresovým prostorem
- Adresový prostor instrukcí interpretovaných CPU je škála adres buněk hlavní paměti, fyzický adresový prostor
- V hlavní paměti kooexistuje OS a (spousta) procesů
- Procesy jsou řízeny programy a v ideálním případě se každý proces vč. OS realizuje ve svém vlastním AP, v AP procesu
- Cílové řešení v soudobých počítačích
  - ✓ umožnit každé entitě typu proces (OS) používat svůj vlastní AP procesu implementovaný jako abstrakce fyzického adresového prostoru počítače – logický adresový prostor

Logický / Fyzický adresový prostor, LAP / FAP

- Logický adresový prostor, LAP, mnohdy také řečený virtuální adresový prostor
  - ✓ je vymezený šířkou a formou adresy v instrukci ve strojovém jazyku,
  - ✓ adresa v LAP – logická adresa, také virtuální adresa
  - ✓ kapacita a struktura LAP je daná bitovou šířkou a strukturou adresy v instrukci
  - ✓ muže být jednodimenzionální nebo dvoudimenzionální
- Fyzický adresový prostor, FAP, resp. reálný adresový prostor
  - ✓ je dán škálou adres hlavní pamětí, je lineární, jednodimenzionální
  - ✓ adresa ve FAP – fyzická adresa, také reálná adresa
  - ✓ kapacita FAP je daná bitovou šířkou adresové sběrnice hlavní paměti, resp. bitovou šířkou registru adresy hlavní paměti

Vázání LAP na FAP

- vázání LAP na FAP,
  resp. vázání instrukcí a dat na adresy hlavní paměti,
  je bázový koncept správy paměti
  - ✓ při vázání LAP–FAP v době kompilace
    - LAP a FAP se shodují (velikostí, strukturou)
    - abstrakce AP se neuplatňuje,
    - přesná konkrétní umístění se musí znát *a priori*
  - ✓ při vázání LAP–FAP v době zavádění
    - LAP a FAP se shodují (velikostí, strukturou)
    - abstrakce AP se neuplatňuje,
    - aplikuje se relativní adresování (báze + offset)
  - ✓ při vázání LAP–FAP v době běhu procesu
    - LAP a FAP mohou mít rozdílné délky a/nebo i struktury
    - uplatňuje se abstrakce AP, vč. virtualizace paměti
    - aplikace DLL (Dynamic Linking Library) modulů, sdílené knihovny

Zavádění, sestavování

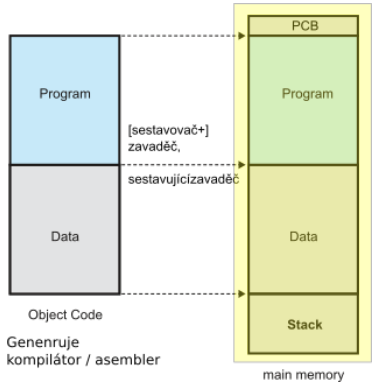
- První krok vytváření aktivního procesu
  - ✓ zavedení programu a iniciálních dat do hlavní paměti a vytvoření obrazu procesu v PCB

Co to je ,,object code" ?

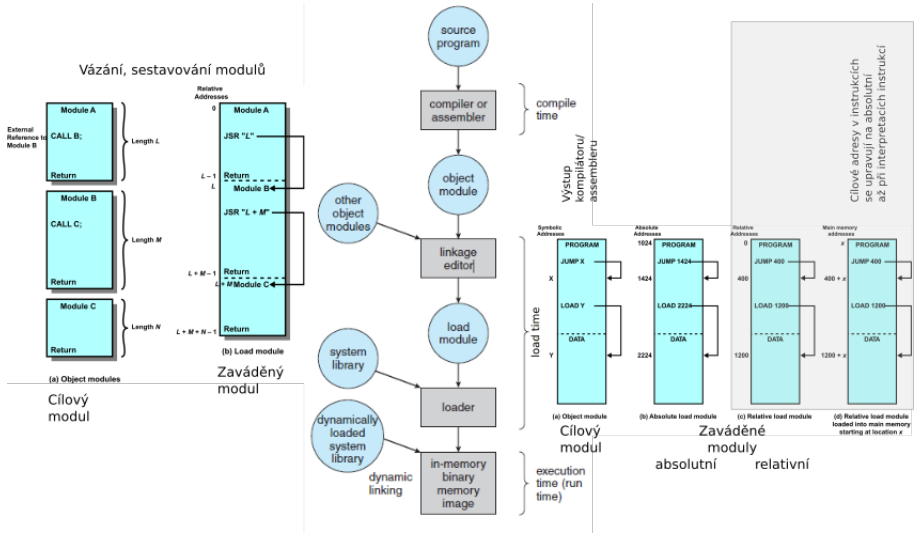
Cílový kód --
produkt kompilátoru,asembleru, ...

Obvykle strojový kód,
případně s možností
přemístění (relokace) ve FAP,
a/nebo sestavení s jinými moduly

Modul s cílovým kódem -- cílový (object) modul



Typický scénář zavádění a sestavování



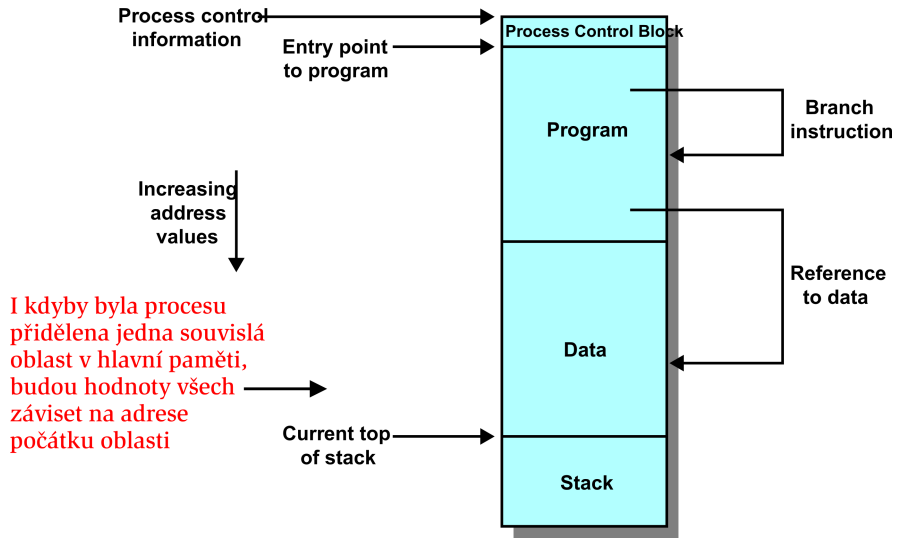
Studovaný model správy paměti

- Předmětem studia jsou operační systémy multitaskingových systémů, ve kterých se vázání LAP–FAP řeší v době běhu procesu
  - ✓ Program (posléze řídicí proces) je uložený na vnější paměti a je vypracovaný pro AP procesu
  - ✓ Do hlavní paměti jej OS (případně po částech) zavádí dynamicky, v hlavní paměti je stále zobrazený v AP procesu a tento AP proces je mapovaný do FAP až při interpretaci jeho instrukcí v CPU
- Vázání LAP–FAP v době tvorby programu, překladu, zavádění, … je typické pro vestavěné počítače apod.

Požadavky na správu paměti

- Možnost relokace programů (přemísťování v hlavní paměti)
  - ✓ programátor nemůže vědět, ve které části hlavní paměti bude bude jeho program umístěný
  - ✓ při výměnách částí programů procesů mezi hlavní a vnější pamětí (swapping) může být těmto částem dynamicky přidělena jiná oblast FAP, než kterou opustily
  - ✓ swapping umožňuje OS udržovat velký bank připravených procesů
  - ✓ odkazy na paměťová místa v instrukci interpretované CPU musí odrážet okamžité skutečné adresy operandů v hlavní paměti (ve FAP)

Adresování v rámci procesu musí umožnit relokaci



Vázání LAP na FAP při kompilaci – staticky

- Abstrakce AP se nepoužívá, $LAP \equiv FAP$
- umístění programu ve FAP je známé a priori (před překladem)
- kompilátor generuje absolutní program přímo pro FAP
- obraz programu ve FAP – absolutní zaváděný modul
- absolutní zaváděný modul zavádí do paměti absolutní zavaděč
- při změně umístění programu ve FAP se musí překlad opakovat

Vázání LAP na FAP při sestavování / zavádění – staticky

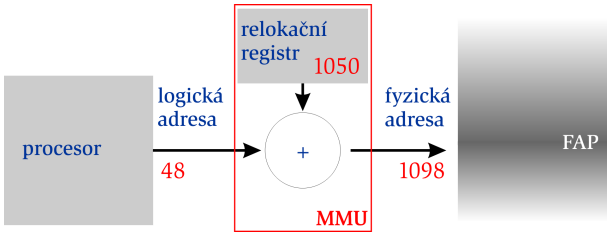
- Abstrakce AP se nepoužívá, $LAP \equiv FAP$
- umístění programu ve FAP je známé při sestavování / zavádění programu
- překladač generuje sestavovatelný modul (*object module*), jehož cílovým adresovým prostorem je LAP
- sestavovač, *linkage editor*, generuje doplňuje do něj knihovní moduly a vytváří zaváděný modul (*load module*)
- zaváděný modul může být absolutní nebo přemístitelný
  - ✓ absolutní modul – obraz FAP, zavádí ho absolutní zavaděč
  - ✓ přemístitelný modul – obsahuje identifikace umístění dat závislých na umístění ve FAP, zavádí ho přemísťující zavaděč zajišťující zobrazování LAP na FAP

Vázání LAP na FAP při běhu – dynamická relokace

- cílovým prostorem programování a sestavení je LAP
- program se zavede do FAP ve tvaru připraveném pro LAP
- vázání adres LAP na adresy FAP se odkládá na dobu běhu – při interpretaci instrukce
- proces může měnit svoji polohu ve FAP mezi různými fázemi běhu
- musí být tudíž dostupná hardwarová podpora – *Memory Management Unit*, *MMU* nebo *Dynamic Address Translation*, *DAT*
- nejjednodušší formou MMU je relokační / bázový registr, viz dále

Vázání LAP na FAP při běhu – dynamická relokace

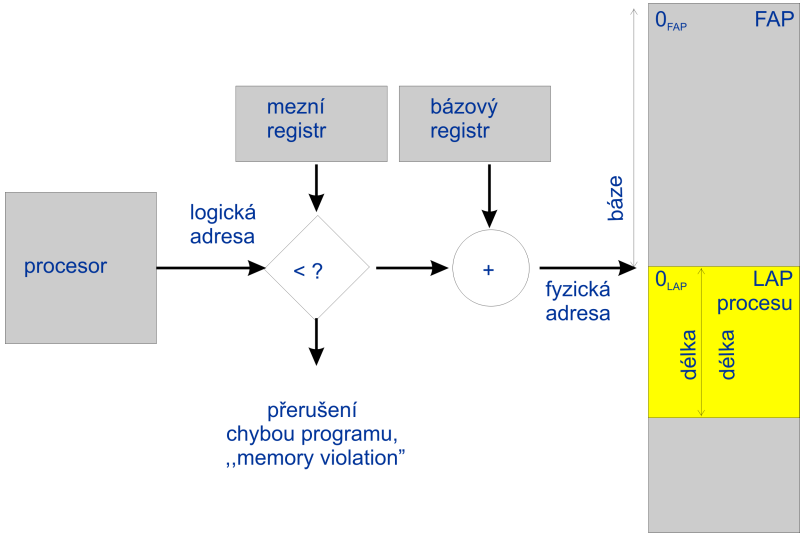
- práce s relokačním registrem
  - ✓ počátečně se nastaví hodnotou dolní adresy oblasti ve FAP přidělené procesu
  - ✓ jeho obsah CPU připočítává k adresám v interpretovaných instrukcí v okamžiku, kdy je tato adresa používaná jako ukazatel do hlavní paměti
  - ✓ relokační registr je privilegovaný, pro nahrání je dostupný pouze OS



Přidělování souvislých oblastí, mezní registr

- hlavní paměť (FAP) se typicky dělí do dvou typů oblastí
  - ✓ oblast pro rezidentní část OS, obvykle na počátku FAP
  - ✓ 1, 2, . . . oblast pro uživatelské procesy
- přidělování oblast procesům
  - ✓ pro ochranu procesů uživatelů mezi sebou a OS lze použít schéma s relokačním a mezním registrem
  - ✓ relokační registr – nejnižší (bázová) adresa oblast ve FAP
  - ✓ buďto mezní registr udává poslední adresu LAP použitou v procesu
    – logická adresa použitá v procesu musí být $<$ obsah mezního registru
  - ✓ nebo mezní registr udává poslední adresu FAP v oblasti FAP přidělené procesu
    – fyzická adresa použitá v procesu musí být $<$ obsah mezního registru

Přidělování souvislých oblastí, mezní registr pro LAP



Dynamické zavádění, Dynamic Loading

- (pod)program – routine – se zavádí až je vyvolán
  - ✓ na disku se uchovává jako přemístitelný zaváděný modul
  - ✓ volající pogram nejprve prověří, zda volaný program je/má být zavedený
  - ✓ a případně ho zavádí a koriguje stav zavedených programů
- dosahuje se lepšího využití prostoru ve FAP
  - ✓ nevolané moduly se nikdy nezavádí
- užitečná technika v případech, kdy se musí velkými programovými moduly řídit řídce se vyskytující alternativy
- Návrh programu nepožaduje žádnou speciální podporu od operačního systému

Dynamické sestavování, Dynamic Linking Library, DLL

- statické sestavování – jednorázová činnost sestavovače
- dynamické sestavování – (pod)program – routine – se sestavuje (a zavádí) až je vyvolán
- pro indikaci kde leží příslušný knihovní program v hlavní paměti, resp. jak ho zavést, pokud ještě zavedený není, se používá malý program – stub (madlo, pahýl, pařez, zbytek, . . .)
- *Stub* při získání řízení nahradí sám sebe voláním (pod)programu – *routine* – a předá mu řízení (provede ho)
- příště se už volaný podprogram volá přímo

Požadavky na správu paměti

- Nutnost ochrany
  - ✓ procesy nesmí být schopné se bez povolení odkazovat na paměťová místa FAP přidělená jiným procesům nebo OS
  - ✓ možnost relokace vyžaduje, aby se adresy kontrolovaly při běhu procesu hardwarem CPU
- Možnost sdílení
  - ✓ více procesů může řízeně sdílet společnou část FAP, aniž by se tím porušovala ochrana paměti
    - sdílený přístup ke společné datové struktuře je lepší řešení, než udržování konzistence jejích násobných kopií vlastněných jednotlivými procesy

Požadavky na správu paměti

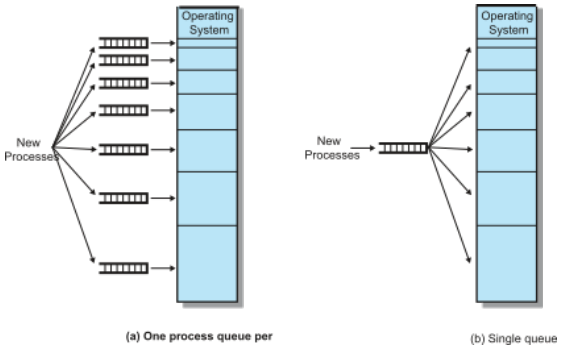
- Možnost logická organizace struktur programu
  - ✓ uživatelé tvoří programy jako moduly se vzájemně odlišnými vlastnostmi
    - moduly s instrukcemi jsou mnohdy *execute-only*
    - datové moduly jsou buď to *read-only* nebo *read/write*
    - některé moduly jsou privátní (*private*) jiné jsou veřejné (*public*)
  - ✓ OS a HW musí podporovat práci s moduly tak, aby se dosáhla požadovaná úroveň ochrany a sdílení

Základní techniky správy paměti

- historické techniky přidělování paměti procesům
  - ✓ souvislé oblasti ve FAP pro běh procesů (*Partitioning*)
  - ✓ formy – fixní oblasti statické délky, trpí vnitřní fragmentací
    – proměnné oblasti dynamicky určované délky, trpí vnější fragmentací
  - ✓ vnější fragmentace – v paměti je dostatečně velký neobsazený prostor rozdělených do více volných oblastí a žádná z nich není schopna uspokojit požadavek na přidělení paměti
  - ✓ vnitřní fragmentace – proces nevyužívá celý přidělený adresový prostor
  - ✓ vesměs používají pro zobrazování $LAP \rightarrow FAP$ dynamickou relokaci bázovým registrem
  - ✓ v současnosti se užívají ve specializovaných OS, typicky v RT-OS, žádná podpora virtualizace paměti
  - ✓ pro zvýšení efektivity (stupně multiprogramování) vesměs umožňují obsah alokovaných oblastí přesouvat mezi hlavní a vnější pamětí – výměny (swapping)

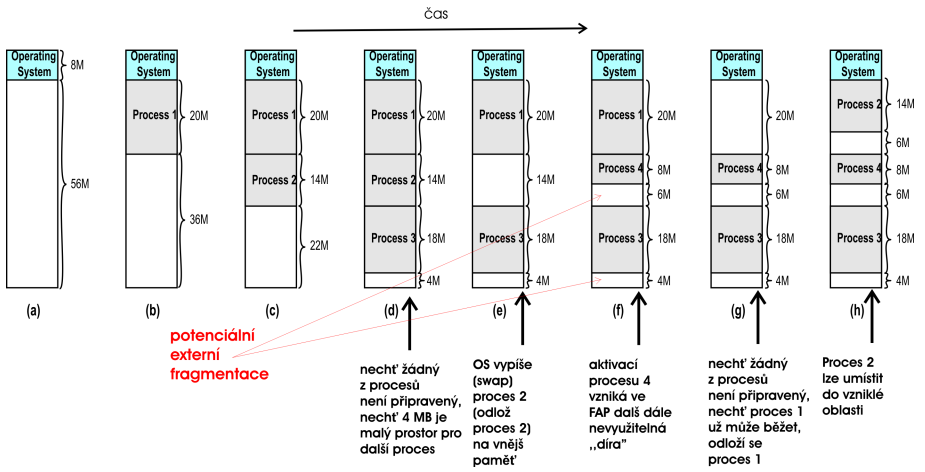
Multiprogramování s více pevnými souvislými oblastmi FAP

- obvykle dávkové systémy, dnes už vesměs historie
- FAP sdílí OS a n procesů
- pro vazbu LAP-FAP se typicky použije relokace využívající bázový registr (+ mezní registr)



(a) One process queue per (b) Single queue

Multiprogramování s dynamickými souvislými oblastmi FAP

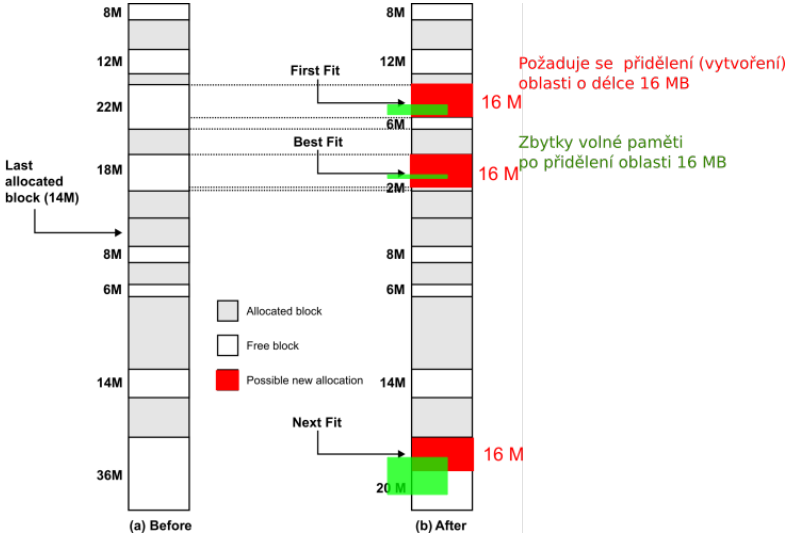


- Oblasti se vytváří a zanikají tak jak vznikají a zanikají procesy

Přidělování souvislých oblastí proměnné délky

- problém
  - ✓ dynamicky vznikají úseky dostupné paměti roztroušené po FAP
  - ✓ procesu se přiděluje oblast v úseku volné paměti ve FAP, který jeho požadavek uspokojí
  - ✓ evidenci o přidělených a volných úseků udržuje OS
- Kde přidělit oblast dané délky, když je volná paměť rozmístěna ve více souvislých nesousedních úsecích?
  - ✓ First-fit – v prvním dostatečně dlouhém úseku volné paměti
  - ✓ Best-fit – v nejmenším dostatečně dlouhém úseku volné paměti, vznikají se velmi malé (nejmenší možné) volné úseky volné paměti
  - ✓ Worst-fit – v největším úseku volné paměti, ponechávají se největší volné úseky volné paměti
  - ✓ Z hlediska rychlosti a kvality využití pamětu jsou *First-fit* a *Best-fit* lepší techniky než technika *Worst-fit*
  - ✓ nejčastěji se používá *First-fit*

Přidělování souvislých oblastí proměnné délky



Zlo fragmentace

- Vnější fragmentace
  - ✓ souhrn volné, tj. neobsazené, paměti ve FAP je dostatečný, nikoli v dostatečně velkém souvislém úseku pro zaváděnou část programu
- Vnitřní fragmentace
  - ✓ Oblasti paměti FAP se přidělují po úsecích, jejichž délka je větší než požadovaná velikost pro zaváděnou část programu
  - ✓ Přebytek v přidělené oblasti FAP je nevyužitelná část paměti
- snižování vnější fragmentace setřásáním
  - ✓ přesouvají se obsahy obsazených oblastí s cílem vytvořit (jeden) velký úsek volné paměti
  - ✓ použitelné jen když je možná dynamická relokace (viz MMU)
  - ✓ provádí se v době běhu – problém I/O – s vyrovnávacími pamětmi plněnými z periférií, autonomně nelze hýbat, umísťují se proto do prostoru OS

Základní techniky správy paměti

- historické techniky přidělování paměti procesům
  - ✓ další možné formy – stránkování/segmentování (*paging / segmenting*)
    - – rozptylování dílčích oblastí LAP po FAP po částech (stránky/segmenty) do nesousedních oblastí FAP,
  - ✓ minimalizace vnější fragmentace (stránkování) a/nebo vnitřní fragmentaci (segmentace), detatily později
- soudobé techniky přidělování paměti procesům
  - ✓ stránkování/segmentování na žádost (*demand paging / segmenting*), tj. virtualizace paměti,
    - – rozptylování oblastí LAP po FAP po částech (stránky/segmenty) do nesousedních oblastí FAP
    - – zaváděni LAP do FAP po částech, na žádost procesu, z obrazu LAP procesu uchovávaného ve vnější paměti

Základní techniky správy paměti, přehled

	+	−
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily continuos frames	No external fragmentation.	A small amount of internal fragmentation.

Základní techniky správy paměti, přehled

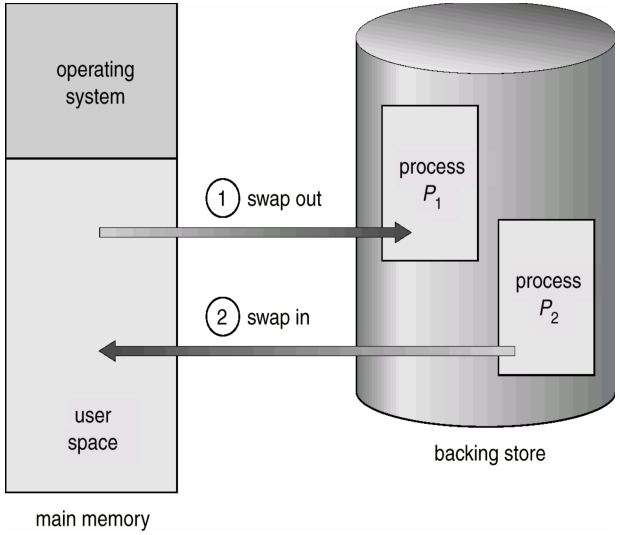
	+	−
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

stránkování / segmentován na žádost

Výměny, Swapping

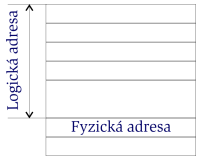
- Obsah oblasti FAP přidělené procesu je vyměňovaný mezi vnitřní a vnější pamětí oběma směry
- *Roll out*, *roll in*
  - ✓ výpis na disk , načtení z disku
- Proces nemusí být přidělena při návratu do hlavní paměti tatáž oblast, kterou uvolnil
- mnohdy používané při prioritním plánování procesů
- Majoritní doba výměn je doba přenosu obsahu oblasti
- Princip používaný mnoha OS ve verzích nepodporujících virtualizaci paměti – v prehistorii: UNIX, Linux, Window

Výměny, Swapping, ve FAP jediný aplikační program



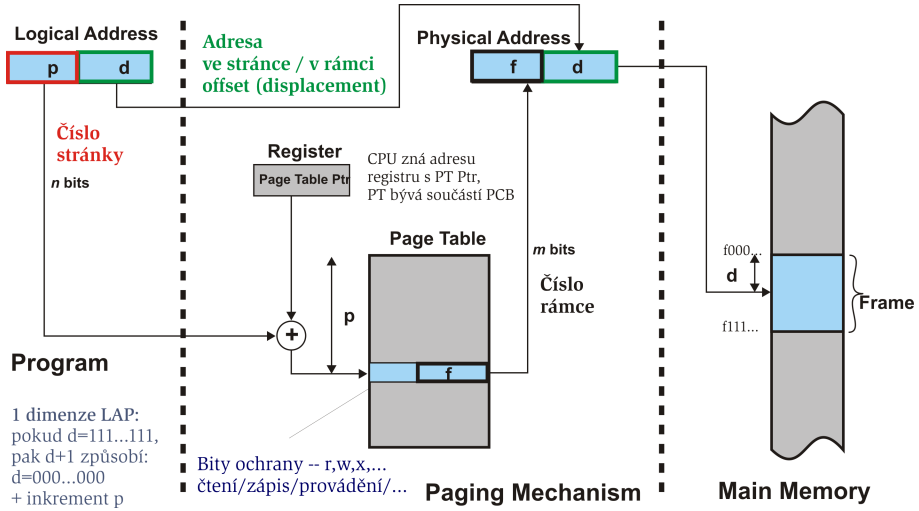
Stránkování, překlad logické adresy na fyzickou adresu

- Překlad
  logická adresa (adresa LAP) → fyzická adresa (adresa FAP)
  se realizuje tabulkou PT, *Page Table*, tabulka stránek
- idea:

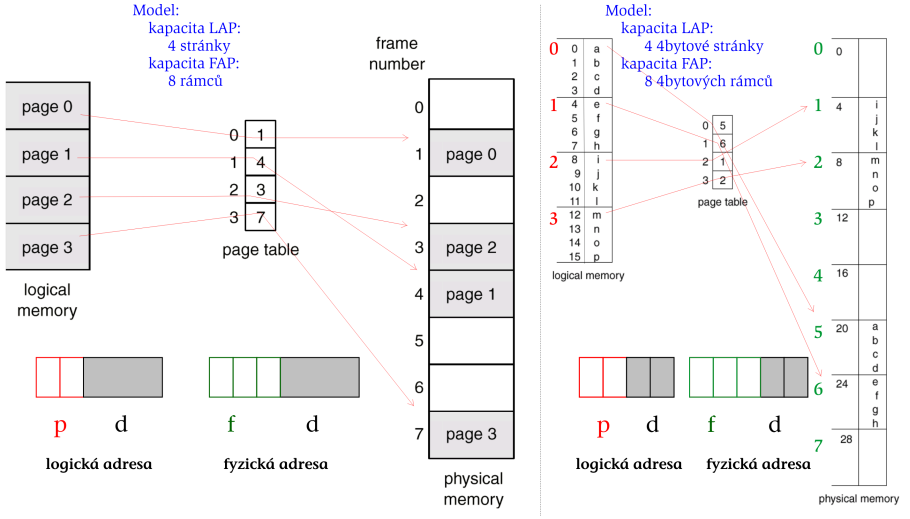


- každý proces má svoji PT (součást PCB),
  obsah PT nastavuje OS
- obsah PT interpretuje MMU
  při interpretaci instrukce procesorem
  - ✓ umístění PT v hlavní paměti mikroprogram CPU zná

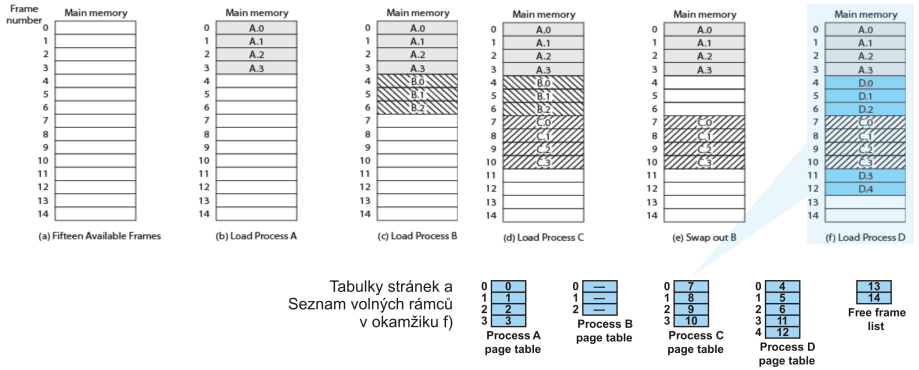
Stránkování, překlad logické adresy na fyzickou adresu



Stránkování, příklad 1



Stránkování, příklad 2, dynamické obsazování FAP



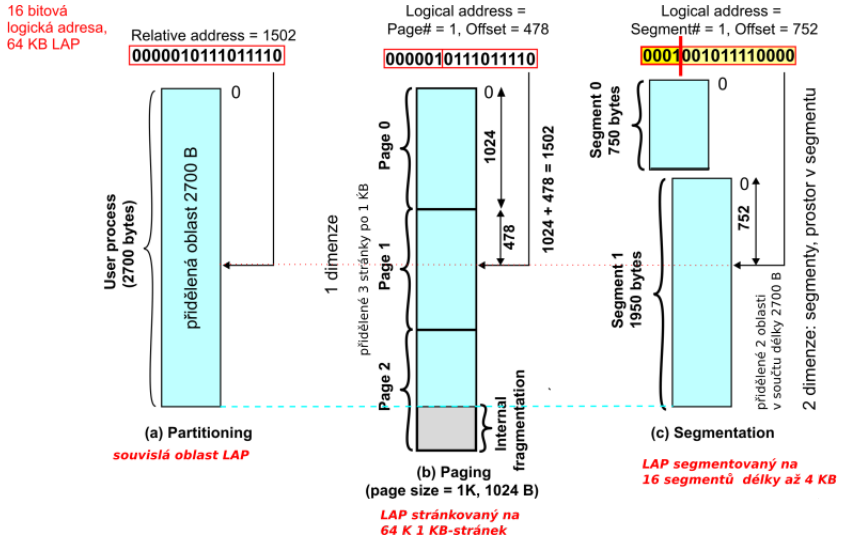
Segmentování

- prostor ve FAP se přiděluje po oblastech proměnné délky
- LAP se dělí do dvou dimenzí – na oblasti zvané segmenty, je definovaný max počet segmentů a max délka segmentu
- segmentům se přidělují oblasti ve FAP
- protože skutečná délka segmentu může být menší než jeho maximální délka, ke každému segmentu se uvádí jeho skutečná délka
- OS si udržuje seznam volných oblastí ve FAP a CPU pomocí tabulky segmentů ví, ve kterém oblasti je umístěný ten který segment

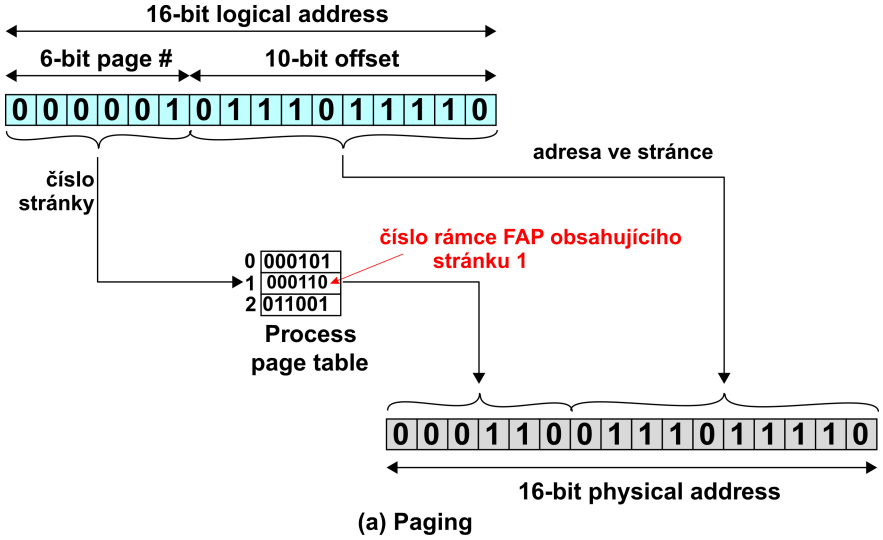
Segmentování

- pořadí přidělených oblast ve FAP nesouvisí s pořadím segmentů v LAP, pro programátora je alokace segmentů do FAP transparentním rysem počítače
- Účel jednotlivých segmentů LAP může být daný jednak architekturou počítače a jednak rozhodnutím programátora

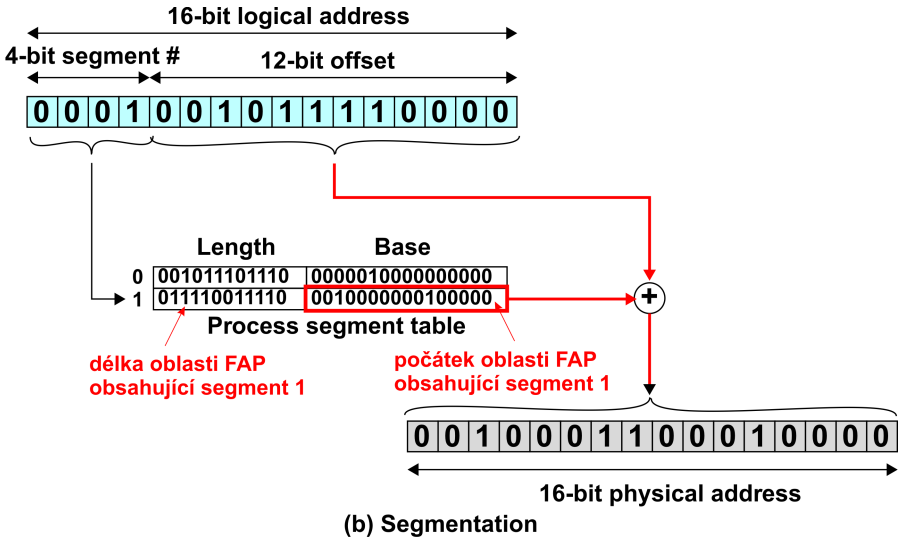
Logická adresa v oblastech, při stránkování a při segmentaci



Překlad logické adresy na fyzickou adresu, stránkování



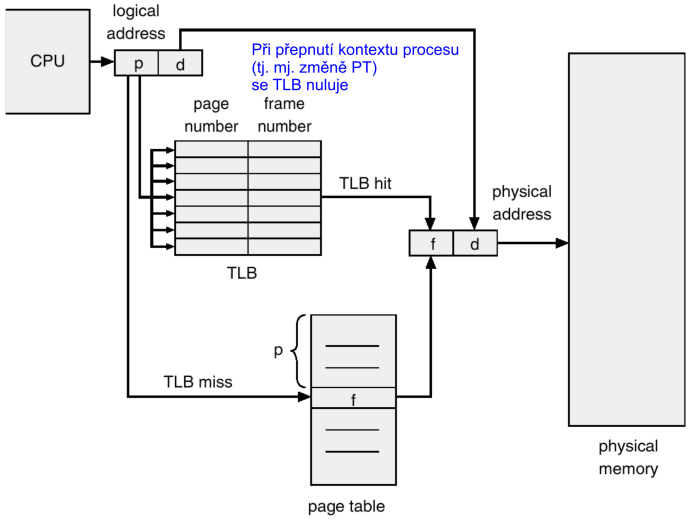
Překlad logické adresy na fyzickou adresu, segmentace



Implementace tabulky stránek

- je uložena v hlavní paměti
- je odkazovaná registrem PTBR (*Page-table base register*)
- zpřístupnění údaje/instrukce v hlavní paměti vyžaduje dva přístupy do hlavní paměti
  - ✓ jednou do tabulky stránek
  - ✓ jednou pro operand
- problém snížení efektivnosti dvojím přístupem lze řešit speciální rychlou hardwarovou cache pamětí
  - ✓ asociativní paměť, *Translation Look-aside Buffers* (TLB)
  - ✓ obsah: k dvojic $\{p, f\}$ použitých v nejbližší historii běhu
  - ✓ překlad $p \rightarrow f$:
    – jestliže se p nachází v TLB získává se hodnota f z TLB
    – v opačném případě se f získává z PT

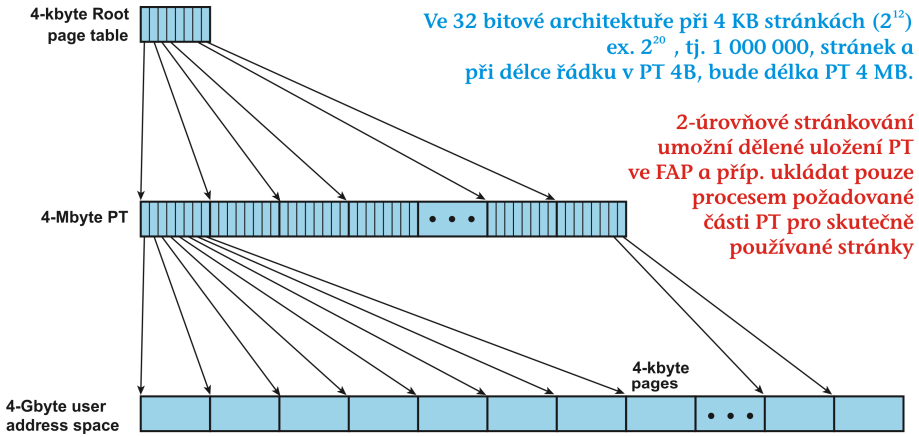
Stránkování, Translation Look-aside Buffers, TLB



TLB, Skutečná doba přístupu do vnitřní paměti

- EAT, (*Effective Access Time*)
- zpřístupnění TLB = ε časových jednotek
- doba přístupu hlavní paměti = t μs
- Hit ratio – α
  - ✓ pravděpodobnost, že se číslo stránky nalezne v TLB
- $EAT_{s\ TLB} = (TLB + operand)\alpha + (TLB + PT + operand)(1-\alpha)$
- $EAT_{s\ TLB} = (\varepsilon + t)\alpha + (2t + \varepsilon)(1-\alpha) = (2-\alpha)t + \varepsilon$
- $EAT_{jen\ PT} = 2t$, pro $t = 100\ ns \rightarrow EAT_{jen\ PT} = 200\ ns$
- $\varepsilon = 20\ ns, \alpha = 80\%\ (0,8), t = 100\ ns \rightarrow$
  $EAT_{s\ TLB} = 120 \times 0.8 + 220 \times 0,2 = 140\ ns$
- $\varepsilon = 20\ ns, \alpha = 98\%\ (0,98), t = 100\ ns \rightarrow$
  $EAT_{s\ TLB} = 120 \times 0.98 + 220 \times 0,02 = 122\ ns$

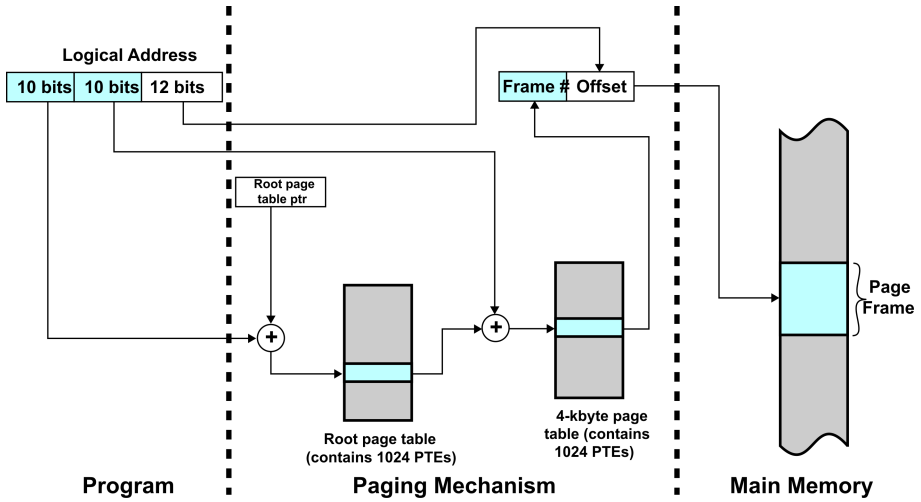
2-úrovňové stránkování (Pentium)



Ve 32 bitové architektuře při 4 KB stránkách (2^{12})
ex. 2^{20} , tj. 1 000 000, stránek a
při délce řádku v PT 4B, bude délka PT 4 MB.

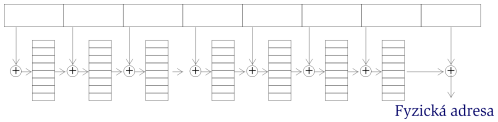
2-úrovňové stránkování umožní dělené uložení PT ve FAP a příp. ukládat pouze procesem požadované části PT pro skutečně používané stránky

2-úrovňové stránkování (Pentium)



Více-úrovňové stránkování, výkon

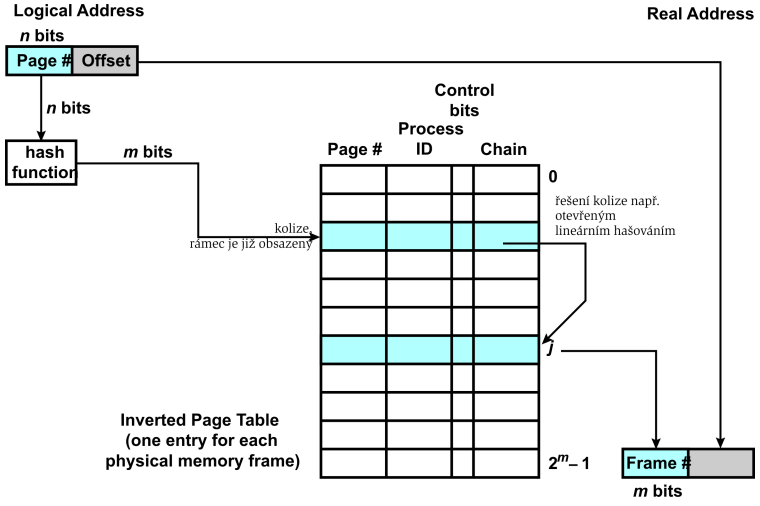
- Pentium, 32 b architektura – 2 úrovně
- Sparc, 32 b architektura – 3 úrovně
- Motorola (68030), 32 b atrchitektura – 4 úrovně
- pro 64 b architekturu je 2-úrovňové schéma nepostačující
  – UltraSparc, 64 b architektura – až 7 úrovní – neúnosné



- Každá úroveň je uchovávaná v paměti v samostatné tabulce, takže zobrazení logické adresy na fyzickou může spotřebovávat mnoho přístupů do vnitřní paměti
  - ✓ možné řešení: viz dále – hašovaná invertovaná PT

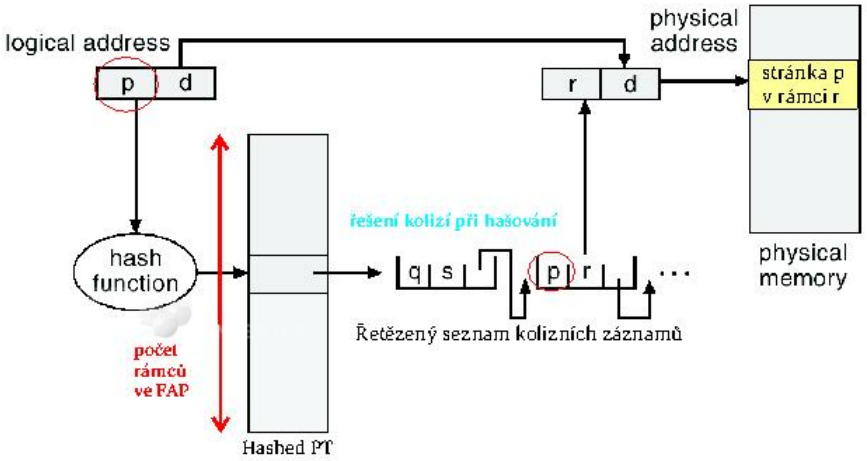
Invertovaná PT

- ✓ např. AS400 (IBM), UltraSPARC, PowerPC, IA-64

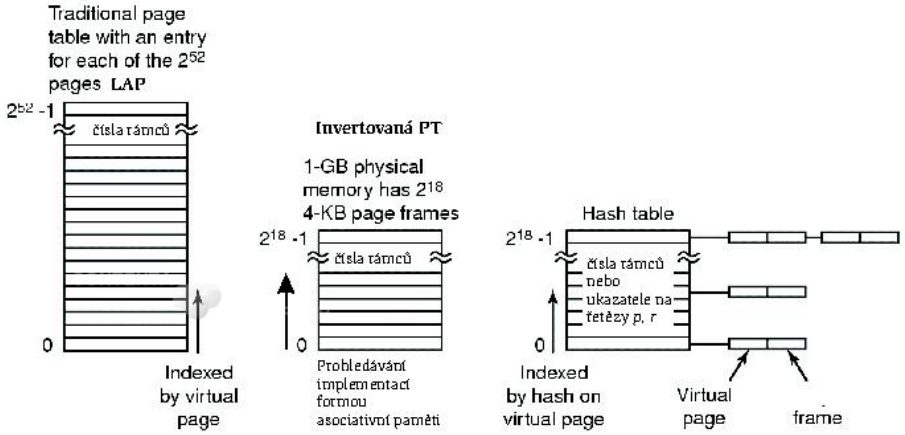


Hašovaná PT

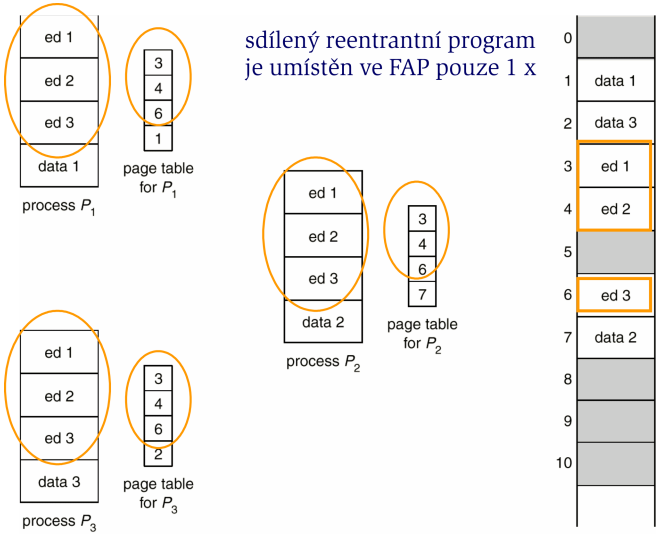
✓ Microsoft SQL Server 2000 (64-bit): Intel Itanium Processor



Porovnání technik implementace PT



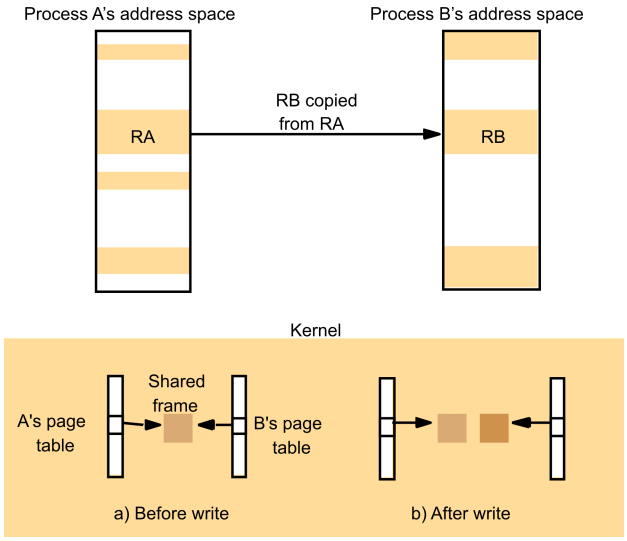
Sdílení stránek



Vytvoření procesu stylem Copy on Write

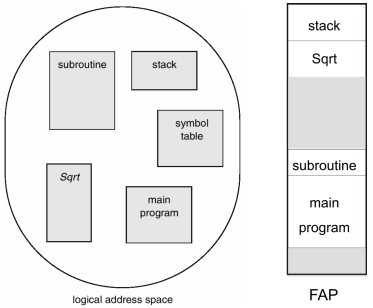
- Nově vytvářený proces požaduje vytvoření nového prostředí běhu
- Tradiční forma vytvoření procesu unixového typu
  - ✓ služba OS *fork* vytvoří nové prostředí běhu kopií prostředí žádajícího procesu + sdělení novému procesu, že je potomkem vytvářejícího procesu
  - ✓ služba *exec* umožní volajícímu procesu definovat nový program řídicí proces kopií z udaného souboru
- Vytvoření procesu způsobem *Copy on Write*
  - ✓ iniciálně nový proces sdílí stránky s původním procesem
  - ✓ při zápisu do stránky novým procesem se vytvoří pro nový proces samostatná kopie modifikované stránky

Vytvoření procesu způsobem *Copy on Write*



Segmentování

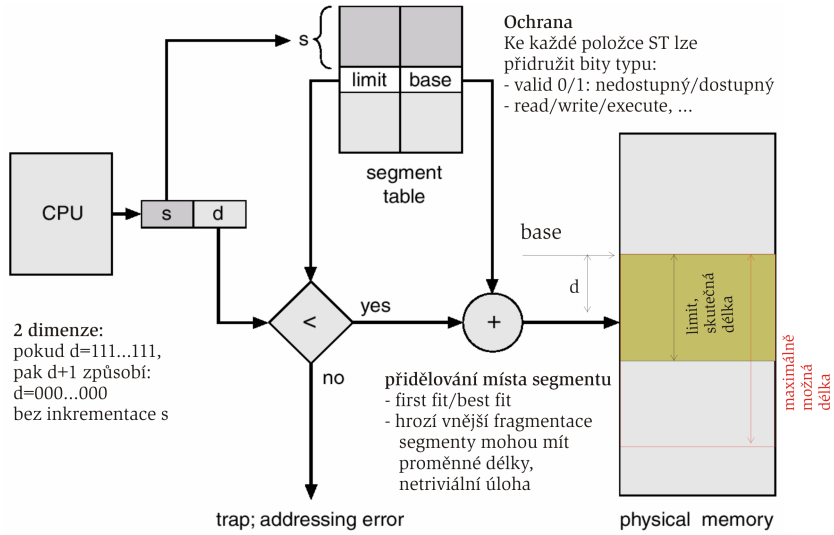
- Podpora uživatelského pohledu na LAP
- program je kolekce (lienárních, samostatných) segmentů (modulů)
- každý segment má programátorem přisouzenou roli



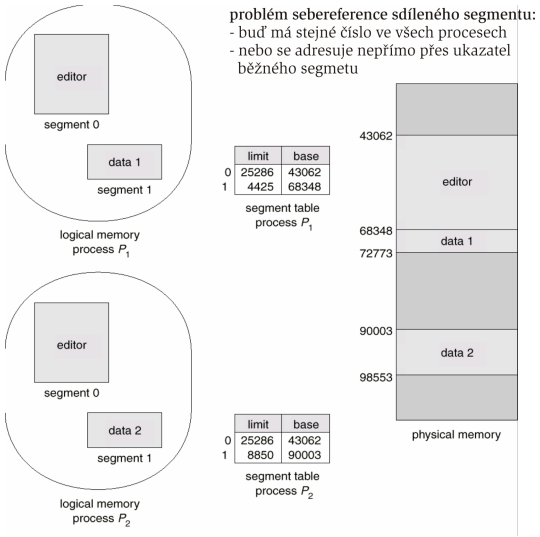
Segmentování

- FAP má 1-dimensionální charakter, je lineární
- LAP má 2-dimensionální charakter
  - ✓ Logická adresa = dvojice {*segment-number s, offset d*}
- Transformace LAP → FAP se řeší dynamicky určovaným bázovým registrem pomocí tabulky segmentů, *Segment Table*, ST
- položka ST
  - ✓ *base* – počáteční adresa umístění segmentu ve FAP
  - ✓ *limit* – skutečná délka segmentu

Segmentování, překlad logické adresy na fyzickou adresu



Sdílení segmentů



Segmentování se stránkováním, stránkování segmentů

- řešení problému vnější fragmentace segmentování stránkováním segmentů
- řešení problému velikosti PT ve FAP
  - ✓ ve vnitřní paměti se uchovávají PT pouze zavedených segmentů
- ST obsahuje adresu PT segmentu, ne bázi segmentu

Segmentování se stránkováním, INTEL/PENTIUM

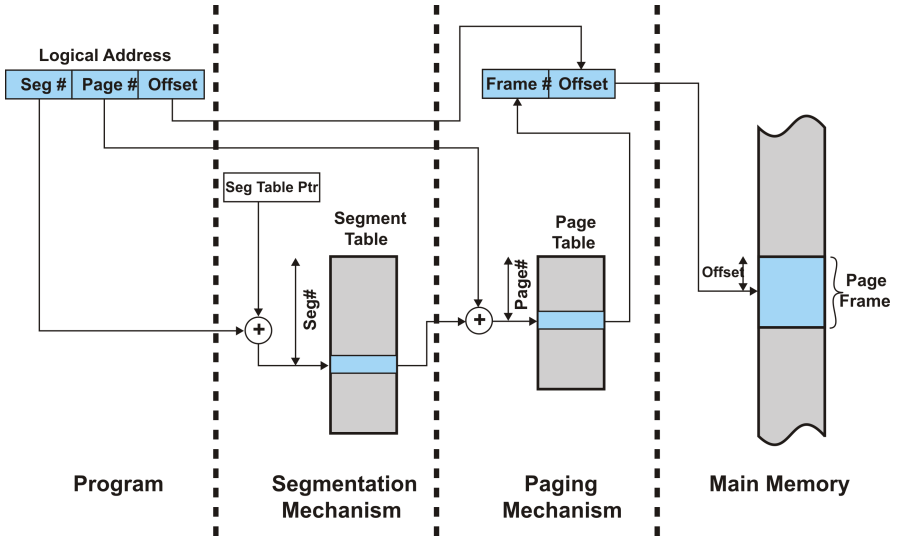
- FAP: 32 bitů adresa, $|FAP| = 4$ GB
- lineární nestránkovaný LAP – řadiče, vestavěné počítače, . . .
  - ✓ 32 bitů adresa, $|LAP| = 4$ GB, $LAP \rightarrow FAP$: identita;
- lineární stránkovaný LAP – původní Unixy
  - ✓ 32 bitů adresa, $|LAP| = 4$ GB,
  - ✓ $LAP \rightarrow FAP$: 2-úrovňové stránkování, 4 KB stránka
  - ✓ 1024 4 MB oblastí stránek – 1024 PT, každá PT pro 1024 stránek
- segmentovaný stránkovaný LAP – Windows
  - ✓ 32 bitů adresa v segmentu, $|segment| = 4$ GB,
  - ✓ 16 K segmentů, $|LAP| = 16$ K x 4 GB = 64 TB,
  - ✓ $LAP \rightarrow FAP$: segmentování (výběr z LAP) + 2-úrovňové stránkování segmentů (zobrazování LAP do FAP)

Segmentování se stránkováním, INTEL/PENTIUM

- LAP = 16K 4GB-segmentů
- dva logické podprostory LAP, TI = 0 / 1
  - ✓ 8 K segmentů pro proces
  - ✓ 8 K segmentů sdílených (OS, . . .)
- ochrana segmentu, RPL (*Requested Privilege Level*)
  - ✓ klasifikace / oprávnění

    0 – správa paměti, bázová bezpečnost
    1 – zbytek OS
    2 – aplikační bezpečnost
    3 – aplikace
  - ✓ read / write omezení na úrovni stránek (v PT)

Segmentování se stránkováním, princip



Segmentování se stránkováním, INTEL/PENTIUM

V jednom okamžiku proces
může pracovat až s 6 segmenty

Linux např.:
1. program jádra OS
2. data jádra OS
3. program procesu
4. data procesu
5. stavový segment (TSS, Task-State Segment)
   pro ukládání stavu při přepínání kontextu
6. volitelně sdílený segment mezi procesu