11. Správa paměti

Cite from 1981: "Nobody will ever need more than 640kByte RAM for a PC"

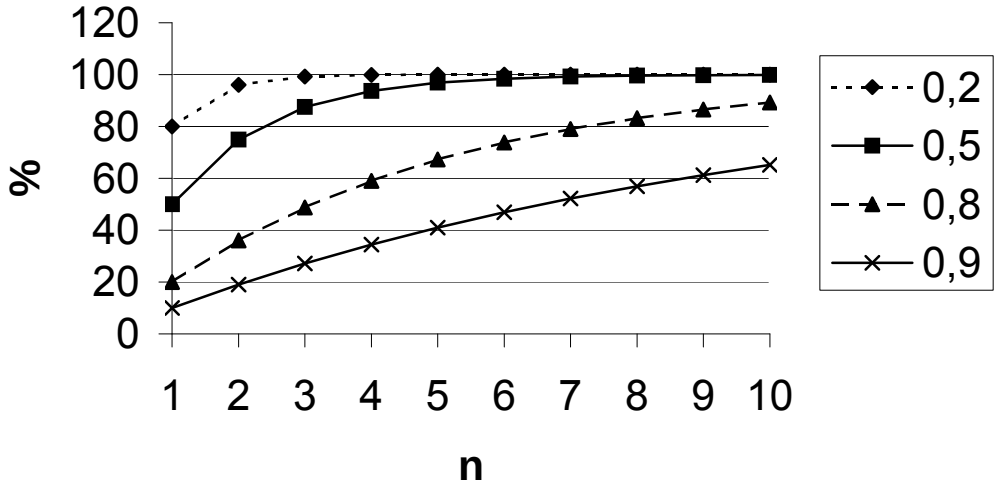
- organizace paměti
- přidělování paměti
- virtualizace paměti

Multiprogramování - více procesů v operační paměti

Využití procesoru

p relativní část provádění strávená čekáním na V/V

p^n pravděpodobnost, že n procesů čeká na V/V



$1- p^n$ využití procesoru

Interaktivní procesy - 99 % stráví čekáním

Při multiprogramování je žádoucí udržovat všechny procesy v paměti.

Organizace paměti

A. organizace LAP
   1. monoprogramové systémy
   2. multiprogramové systémy
      a. společný LAP
      b. oddělený LAP

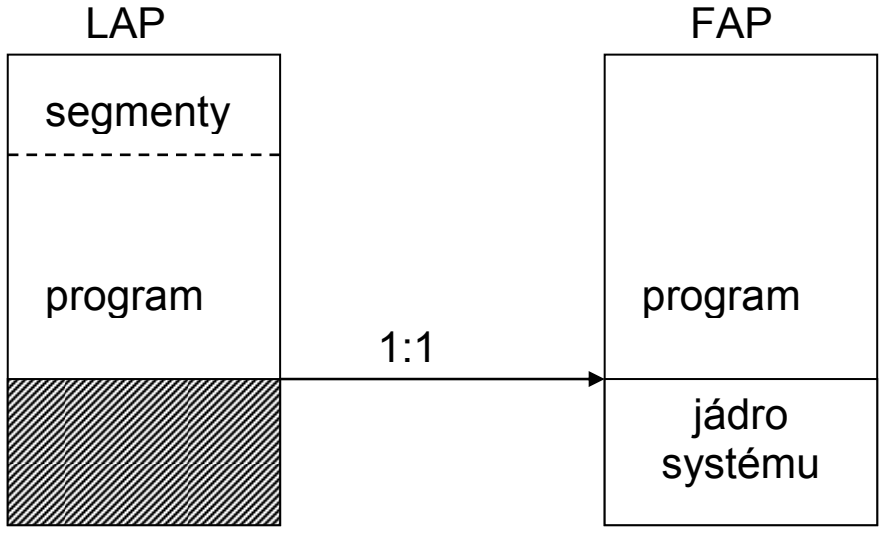
B. mapování LAP na FAP
   1. úseky
   2. segmenty
   3. stránky

A1) Jeden souvislý úsek paměti

LAP(P_i) = FAP, jeden program v paměti
Monoprogramování bez ochrany paměti - jeden adresový prostor, program je zaváděn na pevnou nebo proměnnou adresu

Jádro systému - pevně přidělena část paměti, zbytek dostupný pro procesy (MS-DOS), ochrana mezním registrem (HP RTE-II)

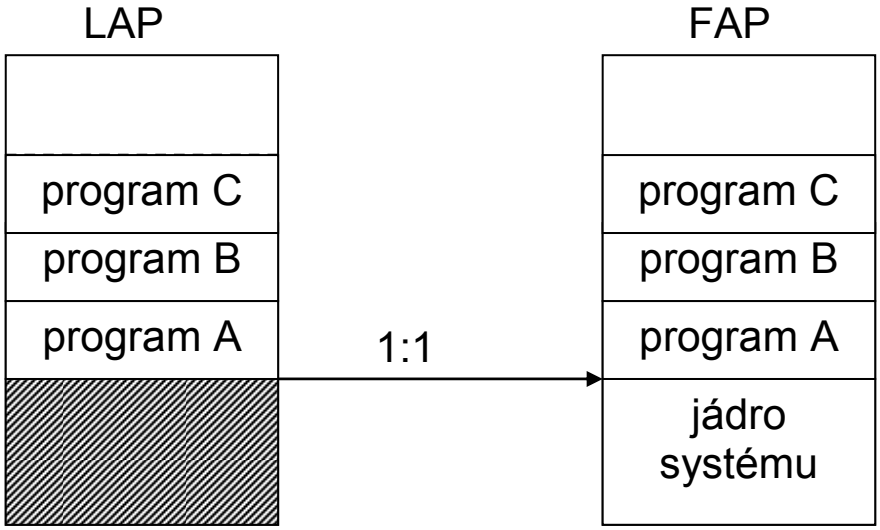
Segmentování (překryvné segmenty, *overlay*) - zavádění programu po částech, podle potřeby



A2a) Společný adresový prostor procesů

$LAP(P_i) = LAP(P_j) = FAP$ pro všechny programy v paměti
Jeden adresový prostor, více programů v paměti na různých adresách, lze kombinovat s následujícími technikami zobrazování LAP do FAP

- \- vyžaduje dynamickou relokaci programů na danou adresu
- \- omezený sdílený LAP (ale pro 64bitové procesory nevadí)
- \+ jednoduchá správa
- \+ jediná tabulka stránek při kombinaci se stránkováním
- \+ přidělování paměti viz přidělování úseků, stránkování



Multiprogramování - nutná ochrana paměti:

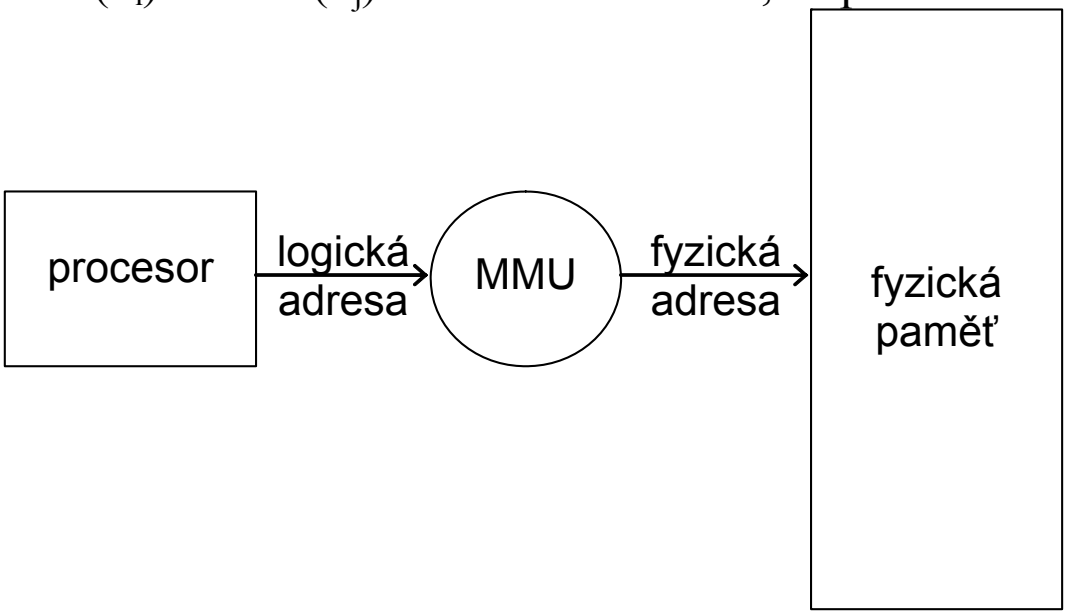
- ochrana uživatelských programů navzájem
- ochrana jádra systému

Implementace ochrany:

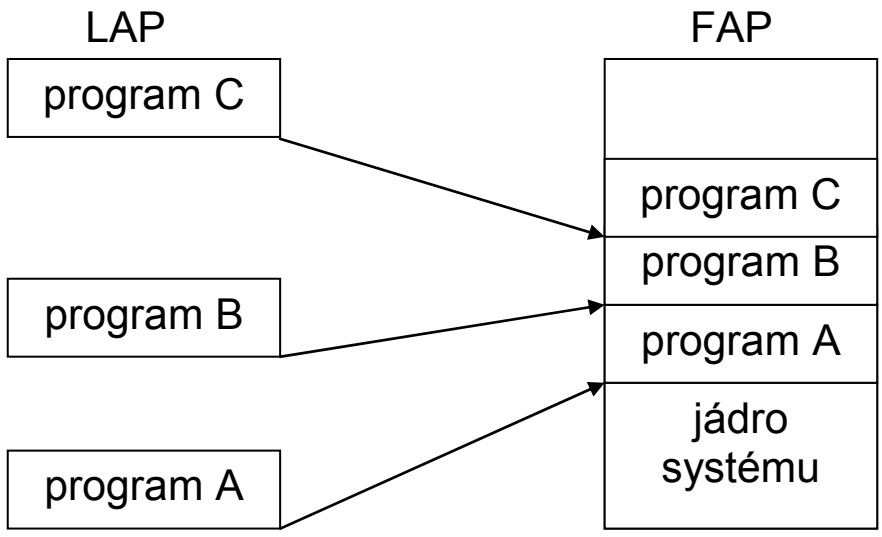
- mezní registry
- chráněný režim činnosti procesoru

A2b) Oddělené adresové prostory procesů

$LAP(P_i)$!= $LAP(P_j)$!= FAP - zobrazení, mapování

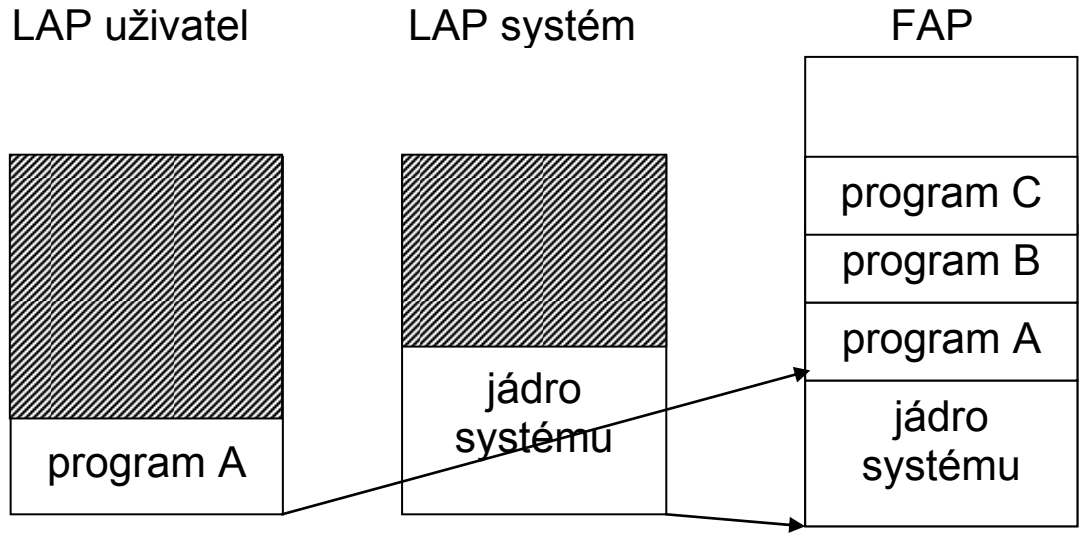


Každý proces má k dispozici celý logický adresový prostor. Logický adresový prostor je tranformován (mapován) na fyzický některou z následujících transformací. Nutná podpora hardware (převod logické adresy na fyzickou je nutný pro každou instrukci).



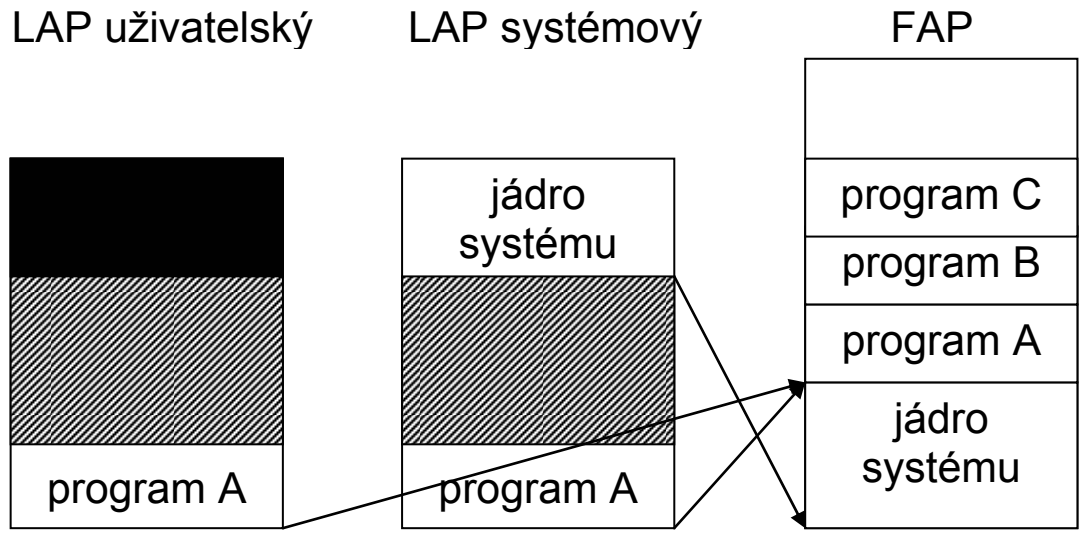
Adresový prostor jádra:

a) oddělený - nutný přepočet adres parametrů volání jádra, přístup do jiného adresového prostoru pro čtení/zápis parametrů a dat (bufferů) každého volání jádra (velká režie).



b) sdílený s procesem, který volá jádro - horní část LAP rezervovaná pro jádro, v uživatelském režimu chráněná (i proti čtení), v systémovém režimu zde běží jádro (Unix, Windows/NT)

- problém 32bitového adresového prostoru, rezerva pro jádro omezuje maximální velikost uživatelských dat, jádro nemá moc prostoru pro buffery (obvykle 512-1024 MB),
- není problém u 64bitových architektur.



B1. Úseky pevné velikosti

$FA = LA + f(P_i)$ mapování

- Fyzická paměť dělena na souvislé úseky (partition) při startu systému nebo rekonfiguraci (HP RTE, OS/360 MFT)
- Programy mají pevnou deklarovanou velikost
- Programy jsou zaváděny do vhodného úseku:
  a) Fronta procesů pro každý úsek (optimální volba úseku)
  b) Společná fronta, přidělen nejmenší postačující volný

Interní fragmentace - nevyužitá část přiděleného úseku

Odkládání (swapping)

- pozastavený proces může být dočasně odložen do odkládacího prostoru (swap)

čas odložení - vystavení + přenos (lineárně závislý na velikosti)
min. doba setrvání v paměti - min. čas odložení
doba odložení - dolní, horní hranice
přidělování odkládacího prostoru - po úsecích (viz dále)
problém V/V bufferů - program může čekat na dokončení přenosu dat systémem, nelze odložit paměť v požadavku

B2. Úseky proměnné velikosti

- Fyzická paměť dělena dynamicky na souvislé úseky podle požadavků (HP RTE-A, OS/360 MFT)
- Problém - co s uvolněnými úseky:
  a) spojování sousedících - vzniká **externí fragmentace:** volná paměť je dostatečná, ale není souvislá. Nevyužitelné úseky, které nelze spojit, ani využít k uspokojení požadavků, protože jsou menší než je požadováno.
  b) setřásání - spojování s přesunem obsazených úseků
- problematika správy úseků viz dále

B3. Segmentace

$FA = LA_{ofset} + f(P_i, LA_{segment})$

Princip úseků proměnné velikosti, ale pro jeden proces více úseků = *segmentů*.

Adresa segmentu - součástí logické adresy $<LA_{segment}, LA_{ofset}>$

- explicitní - součástí každé adresy
- implicitní - segmentový registr

Tabulka segmentů - mapuje segmenty na souvislé úseky fyzické paměti, každá položka obsahuje dvojici: **(báze, limit)**

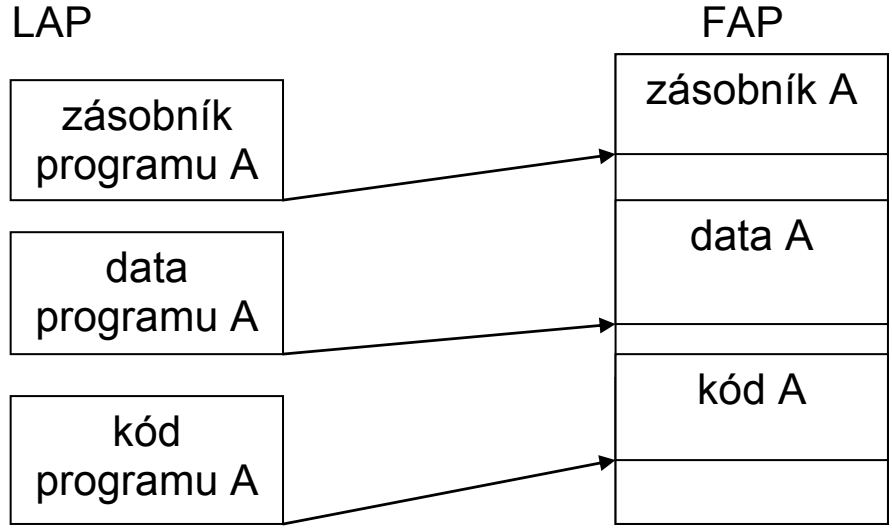
báze – bázová fyzická adresa segmentu

limit – délka segmentu

lokální - každý proces má svou

globální - jediná pro systém, segmenty musí být chráněné

Program = soubor segmentů (kódové, datové, zásobník)



Výhody:

- Segmentace umožňuje logicky rozdělit adresový prostor a zároveň vymezit stejně chráněné celky adresového prostoru.
- Segmenty lze sdílet mezi procesy (např. sdílené knihovny).
- Segmenty pro dynamická data a zásobník mohou bez problému narůstat (problém u jednoho úseku pro celý proces).

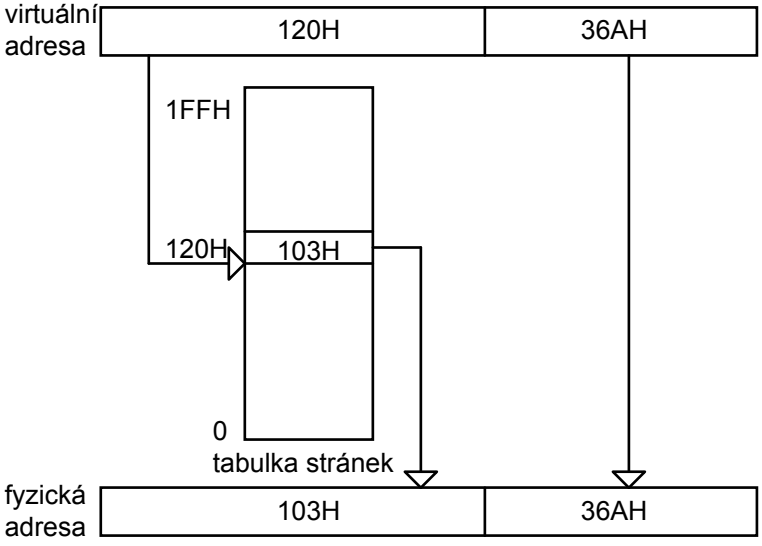
Nevýhody:

- Zůstává problém externí fragmentace - přidělování paměti po úsecích proměnné velikosti
- Strukturovaná logická adresa (segid,ofset) – praktické problémy při použití ukazatelů, nelze spojovat segmenty pro získání většího prostoru (aritmetika ukazatelů)
- Velikost segmentu nemůže překročit velikost dostupné paměti – nelze prakticky virtualizovat paměť (segment musí být přítomný buď celý nebo vůbec).

B4. Stránkování

$FA = LA + f(P_i, LA >> \log_2(\text{velikost\_rámce}))$

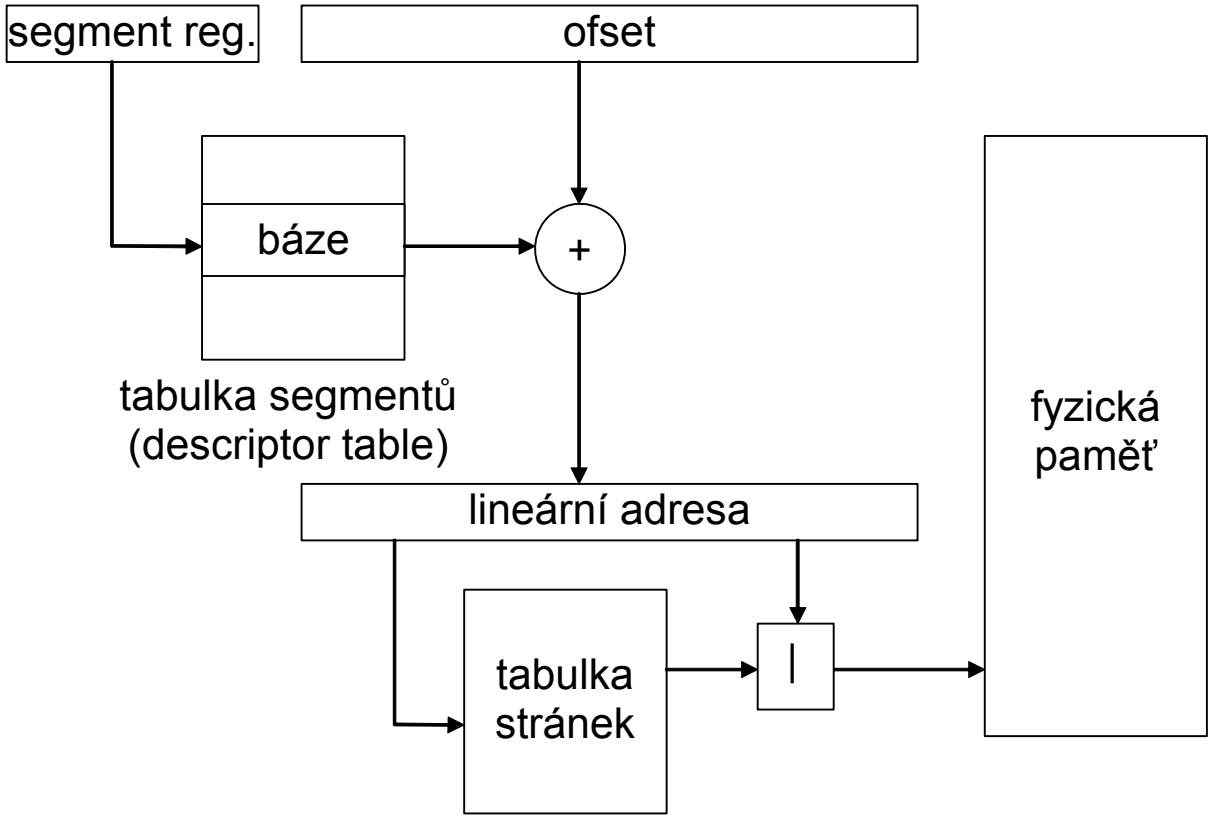
- Fyzická paměť dělena na *rámce* - úseky stejné velikosti
- Logická paměť dělena na *stránky* stejné velikosti
- Velikost rámce=stránky obvykle 4 KB až 16 KB
- Transformace LAP tabulkou stránek (zobrazení LA na FA)
- Logická adresa rozdělena na dvě části: <číslo stránky;ofset>
- Paměť přidělována po rámcích, lze přidělit libovolné rámce, nemusí být vedle sebe:
  - nenastává externí fragmentace
  - interní fragmentace je, ale průměrně (velikost_stránky/2).
- Problém je ochrana paměti, musí být na úrovni stránek.



B5. Segmentace se stránkováním

Segmenty podléhají stránkování - lokální (každý segment má svou tabulku stránek - Multics), globální (všechny segmenty sdílí stejnou tabulku stránek - Intel)
Spojení výhod segmentace – nastavení ochrany paměti na úrovni segmentů a stránkování – bez externí fragmentace, optimální využití paměti, virtualizace paměti.

Architektura Intel X86 je netypická způsobem využití segmentace (pouze 4 implicitní segmenty), proto se segmentace v 32bitovém režimu nepoužívá (všechny segmenty mají stejnou bázi 0 a stejný limit).



Segmentace a stránkování Intel >= 80386

Správa úseků o proměnné velikosti

Zdroj má omezenou kapacitu, je přidělován po úsecích proměnné velikosti. Při uvolňování vznikají volné úseky.

Cíle:

- Efektivní využití zdroje o omezené kapacitě – pokud není zdroj zcela využit, musí být volné úseky použitelné k uspokojení požadavků.
- Rychlá alokace zdroje – minimální čas nutný pro nalezení dostatatečně velkého volného úseku.

Použití – přidělování systémové paměti, swap, LAP, apod.

Problém: Který volný úsek použít pro uspokojení požadavku o velikosti N? Pokud je větší, přidělit jej celý nebo rozdělit?

First Fit - přiděluje první postačující úsek
Next Fit - přiděluje první postačující úsek, příště pokračuje od místa posledního přidělení
Best Fit - přiděluje minimální postačující volný úsek, minimální interní fragmentace
Worst Fit - přiděluje vždy největší volný úsek, menší externí fragmentace malých úseků, ale nepříjemná fragmentace velkých úseků

Při uvolňování vznikají prázdné úseky o různých velikostech - problém evidence volných úseků:

- seznam řazený podle pořadí uvolnění – FIFO, LIFO (nejjednodušší)
- společný seznam organizovaný podle adres (vhodné pro spojování)
- samostatný seznam volných bloků
- seznam organizovaný podle velikosti (náročné zařazování)

Spojování volných úseků:
- nikdy
- při alokaci
- při uvolnění
- při neúspěšném přidělení (garbage collection)

Problém – přidělení znamená prohledávání lineárního seznamu, což je časově náročné. Stejně tak spojování (pokud není seznam podle adres).

Vlastnosti:
Z hlediska správy paměti jsou při různých vzorcích požadavků srovnatelné, žádný není optimální (Knuth, 1968).
Z hlediska rychlosti přidělování uvolňování je vhodná kombinace first-fit a společný seznam podle adres se spojováním při přidělování, naopak jakékoli varianty best/worst fit jsou pomalejší.

Varianty optimalizované na rychlost:
Alokace úseků o velikosti 2^n - nejpoužívanější metoda *malloc()*
- přidělují se pouze úseky o velikosti 2^n (nejmenší postačující)
- seznamy volných úseků jsou nezávislé pro každou velikost
- ze seznamu volných úseků se přidělí první volný
- pokud není k dispozici volný úsek dané velikosti, alokuje se blok pro K úseků a ty se zařadí do seznamu volných úseků
- není třeba spojovat
- jednoduchá organizace seznamu volných úseků
- možná paralelizace přidělování úseků o různých velikostech
- větší interní fragmentace

Buddy systém (SVR4, Linux - systémová paměť)

- varianta předchozího se spojováním
- přidělí se vždy nejmenší postačující úsek o velikosti $2^n >= N$
- velikost volné paměti známá dopředu = 2^m
- seznam volných bloků pro paměť o velikosti 2^m se skládá z *m* seznamů pro velikosti 2^i, i = 1..m
- pokud není volný blok velikosti 2^i, vezme se nejbližší větší dělením na poloviny vytvoří blok o velikosti 2^i, zbylé volné bloky se zařadí do odpovídajících seznamů
- adresa každého přiděleného bloku je násobkem 2^i
- při uvolnění se sousední bloky spojují - hledání blížence pro *x*:

  buddy(*x*) = $x+2^i$ pro ($x \bmod 2^{i+1}$) = 0

  = $x-2^j$ pro ($x \bmod 2^{i+1}$) = 2^i
- pro evidenci volných bloků se používá bitová mapa