INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA STROJNÍ

OPERAČNÍ SYSTÉMY

PŘIDĚLOVÁNÍ PAMĚTI

doc. Dr. Ing. Oldřich Kodym

Ostrava 2013

ISBN 978-80-248-3053-7

Tento studijní materiál vznikl za finanční podpory Evropského sociálního fondu (ESF) a rozpočtu České republiky v rámci řešení projektu: CZ.1.07/2.2.00/15.0463, MODERNIZACE VÝUKOVÝCH MATERIÁLŮ A DIDAKTICKÝCH METOD

OBSAH

2. PŘIDĚLOVÁNÍ PAMĚTI

OBSAH KAPITOLY:

Přidělování jediné souvislé oblasti paměti.

Přidělování paměti po sekcích.

Dynamické přemisťování sekcí.

Stránkování.

Stránkování na žádost.

Segmentace.

Segmentace a stránkování na žádost.

MOTIVACE:

Způsob přidělování operační paměti je jedním klíčových parametrů operačního systému, který je určující pro jeho vlastnosti, funkce a využití. V následujícím textu jsou představeny různé strategie přidělování paměti, důležité datové struktury operačního systému a kritéria hodnocení jednotlivých způsobů přidělování operační paměti.

CÍL:

Přidělování paměti – kritéria hodnocení, jedna souvislá oblast, statické sekce, výhody, nevýhody.

Dynamické přidělování sekcí – principy, výhody, nevýhody.

Dynamické přemisťování sekcí – principy, relokační registr, výhody, nevýhody.

Stránkování paměti – principy, podpůrné datové struktury, výhody, nevýhody.

Stránkování na žádost – principy, podpůrné datové struktury, výhody, nevýhody.

Segmentace paměti – principy, sdílení segmentů, výhody, nevýhody.

1. ÚVOD

Paměť = *operační paměť* je paměť, kterou přímo využívají procesory při zpracování instrukcí a dat. Instrukce a data umístěna kdekoliv jinde (ve virtuální paměti, v souboru na odkládacím zařízení) jsou pro procesor nedostupná a zpracována být nemohou.

Funkce modulu přidělování paměti:

1. **Sledování stavu** každého paměťového místa v operační paměti, tj. zda bylo přiděleno, nebo je volné, k dispozici.
2. **Určování strategie** přidělování paměti, tj. komu bude paměť přidělena, která její část, kdy a v jakém rozsahu. Pokud má být operační paměť sdílena více procesy, musí určit, které z požadavků procesu budou kdy splněny.
3. **Realizace přidělení** paměti. Jakmile je o přidělení paměti rozhodnuto, musí se zvolit příslušná paměťová místa a aktualizovat informace o přidělení paměti.
4. **Realizace uvolnění** paměti. Proces může paměť uvolnit sám, nebo mu může být modulem přidělování paměti odebrána (dle strategie přidělování paměti). Po uvolnění je třeba aktualizovat odpovídající informace o přidělení paměti.

Prostudujeme tyto techniky přidělování paměti:

1. Přidělování jediné souvislé oblasti paměti.
2. Přidělování paměti po sekcích.
3. Dynamické přemisťování sekcí.
4. Stránkování.
5. Stránkování na žádost.
6. Segmentace.
7. Segmentace a stránkování na žádost.

2. PŘIDĚLOVÁNÍ JEDINÉ SOUVISLÉ OBLASTI PAMĚTI

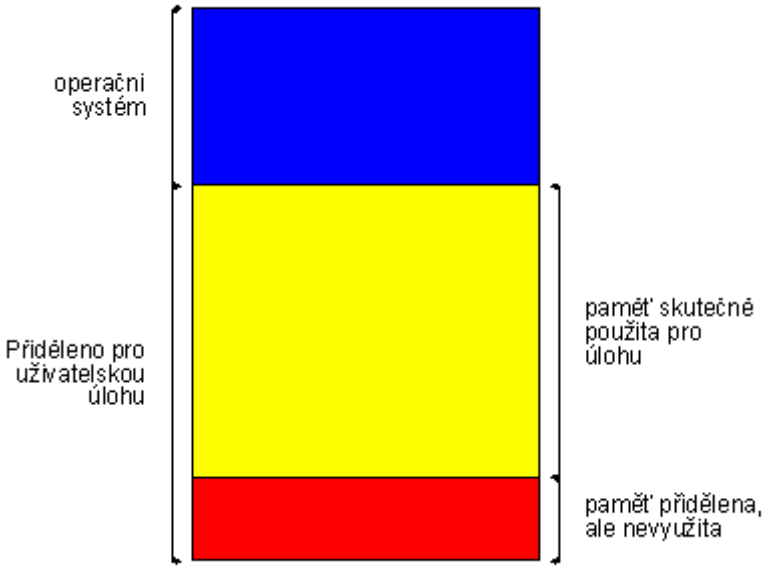
Tento způsob práce je velmi jednoduchý, pochází z doby, kdy na osobních počítačích byl provozován operační systém MS DOS, resp. systémy podobného charakteru. Uživatelským rozhraním zde byl pouze příkazový řádek. Základní charakteristiky:

- nevyžaduje žádné zvláštní technické prostředky
- neumožňuje multiprogramování - mezi uživatelem, úlohou, krokem úlohy a procesem existuje vzájemně jednoznačná korespondence. Pojmy uživatel, úloha, krok úlohy a proces jsou zde v podstatě zaměnitelné.
- 3 souvislé úseky paměti (viz obr. 1):
  - jeden je trvale obsazen operačním systémem
  - celá zbývající paměť je k dispozici, a také přidělena jediné úloze, která ve skutečnosti zase využívá jen část tohoto úseku. Ta zbývající je sice úloze přidělena, ale zůstává nevyužita

Principiální výhoda této koncepce tkví v jednoduchosti:

- Sledování paměti – celá paměť je přidělena jediné úloze
- Strategie přidělování paměti – celá paměť je přidělena jediné úloze

- Přidělení paměti – celá paměť je přidělena jediné úloze
- Uvolnění paměti – po dokončení úlohy se celá paměť uvolňuje a je k dispozici pro další použití



Obrázek 1 - Přidělování jediné souvislé oblasti

2.1 Požadavky na technické vybavení

Tato technika nevyžaduje zvláštní technické vybavení. Vhodný je mechanismus ochrany paměti, která je přidělena operačnímu systému - např. mezní registr a dva stavy procesoru. **Mezní registr** obsahuje nejnižší adresu obsazenou OS a v **problémovém (uživatelském) stavu** procesoru není dovolen zápis kamkoli nad ni. **V privilegovaném stavu** procesoru ano.

2.2 Výhody

- Jednoduchost:
  - Dokáže pracovat i s velmi malou pamětí.
  - K pochopení a použití takového systému není třeba velkých vědomostí.

2.3 Nevýhody

- Nevyužívá plně paměť:
  - Část pamětí není využita (viz obr. 1).
  - Paměť obsahující uživatelský program není využita, je-li úloha ve stavu *čekající*. Tato doba může zahrnovat i 65 až 70 % celkového času. To je doba, kdy není využita ani paměť, ani procesor.
  - Paměť může obsahovat informace, které nebudou využity (např. alternativní algoritmy zpracování, naddimenzovaná pole…). Proč je zatahovat do pamětí?
- Nedostatečná flexibilita - úloha nemůže byt vykonána, je-li její požadovaný adresový prostor vetší než ten, který je k dispozici.

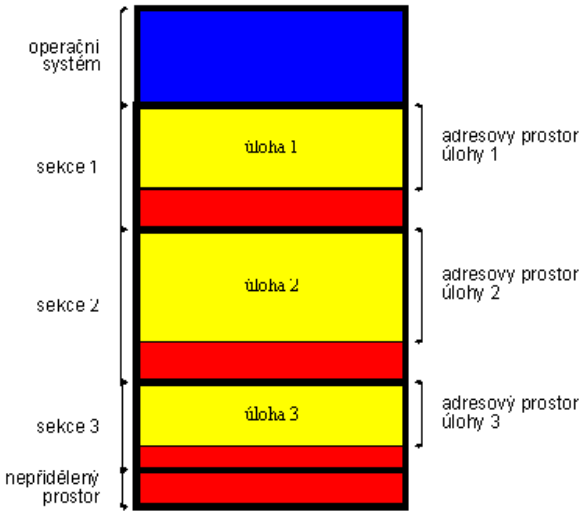
Shrnutí:

- Nedostatečné využití paměti.

- Nedostatečné využití procesoru.
- Omezení uživatelské úlohy rozsahem operační paměti.

3. PŘIDĚLOVÁNÍ PAMĚTI PO SEKCÍCH

Jde o jednu z nejjednodušších technik přidělování pamětí v multiprogramových systémech. Paměť se rozdělí na samostatné úseky – **sekce** (memory partions), z nichž každá obsahuje paměťový prostor jedné úlohy (viz obr. 2).



Obrázek 2 - Přidělování sekcí paměti

Čtyři funkce modulu přidělování paměti lze realizovat takto:

- Sledování stavu každé sekce – ("používá se" / "nepoužívá se", rozsah).
- Strategie přidělování paměti – řeší plánovač úloh.
- Přidělení paměti – ze sekcí, které jsou k dispozici, se přidělí sekce dostatečného rozsahu.
- Uvolnění paměti - po ukončení úlohy se sekce označí jako "nepoužívá se" a je k dispozici.

3.1 Požadavky na technické vybavení

Požadavky na technické vybavení jsou minimální. Je vhodné zajistit technickými prostředky ochrany paměti, aby úloha nemohla porušit OS nebo jinou úlohu ať už náhodně nebo v důsledku selhání technického vybavení. Lze to provést několika způsoby, např. dvěma mezními registry. Pokud by došlo při běhu programu k pokusu o užití paměti vně sekce, nastalo by přerušení z důvodu porušení ochrany paměti.

3.2 Nevýhody

- Častá změna hodnot stavových registru (při každém přidělení procesoru)
- Obtížná ochrana paměti u I/O kanálu – není únosné, aby OS kontroloval všechny mezní registry před každou I/O operací.

Dokonalejší řešení je metodou **ochrany paměti klíčem**. Každé sekci je přidělen klíč (přirozené číslo; klíč 0 je vyhrazen pro OS). Tyto klíče se technickými prostředky přiřadí jednotlivým blokům paměti (o rozsahu např. 2 kB – rozsah každé sekce musí pak byt násobkem 2 kB a všechny klíče v sekci musí mít stejnou hodnotu). Při každém přidělení

procesoru se položka klíč ve stavovém slově nastaví na příslušnou hodnotu. Obdobně využívají ochrany paměti i I/O kanály.

3.3 Statické přidělování sekcí

- Paměť je pevně rozdělena na sekce už při spuštěni OS.
- V každém kroku úlohy musí být udán max. rozsah paměti – vyhledá se dostatečné velká sekce (nejmenší z těch, do kterých lze úlohu umístit) a ta je úloze přidělena.
- Metoda je vhodná zejména u systémů s omezeným rozsahem úloh při jejich známých typických paměťových nárocích a známé frekvenci jejich zadávání. Pak se může vhodně zvolit velikost jednotlivých sekcí. Jinak většinou zůstane značná část paměti nevyužita.

4. DYNAMICKÉ PŘIDĚLOVÁNÍ SEKCÍ

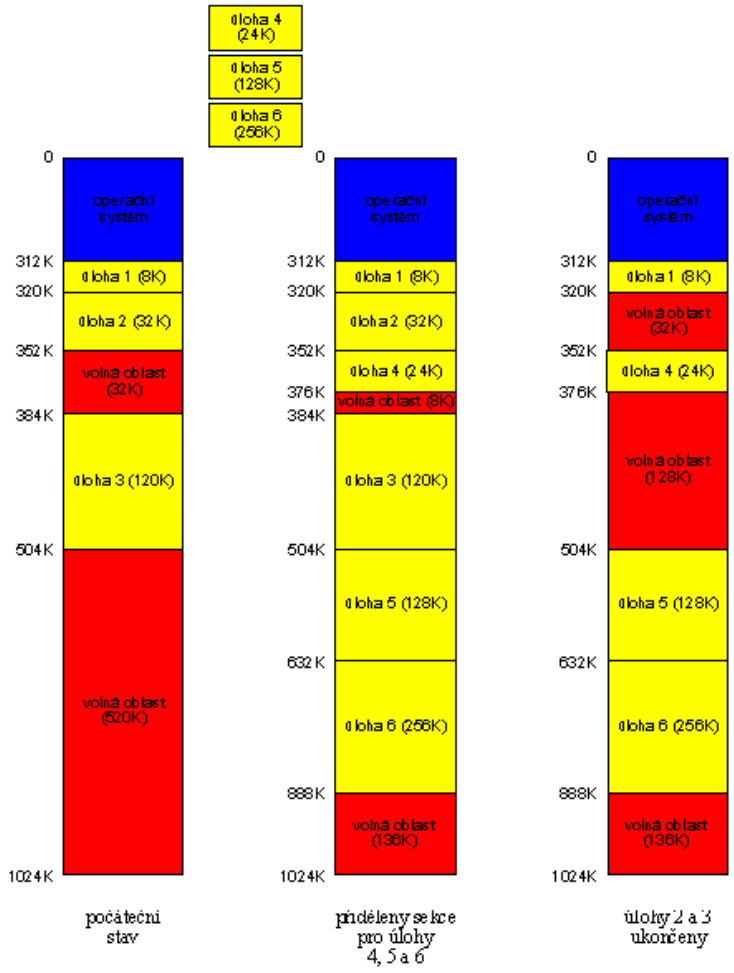
- Sekce se vytvářejí za běhu úlohy tak, aby jejich rozsahy přesně odpovídaly paměťovým nárokům jednotlivých úloh.
- Způsobu realizace dynamického přidělování sekcí je mnoho. Vždy je třeba vytvořit tabulky se záznamy o každé sekci - rozsah, umístění v paměti a vymezeni přístupu a tabulku pro zbylou volnou oblast. Pro tyto tabulky se využívají dynamické datové struktury operačního systému.
- Před přidělením paměti je nutné:
  - nalézt volnou oblast dostatečného rozsahu; je-li oblast vetší je nutno ji rozdělit na dvě – jedna se obsadí úlohou a druhá zůstane nadále volnou oblastí.
  - je-li sekce uvolněna, je vhodné ji spojit se sousedními volnými oblastmi, aby vznikla co největší volná oblast.

4.1 Výhody

- Umožňuje multiprogramování a tím se dosahuje efektivnějšího využití procesoru a I/O zařízení.
- Nevyžaduje nákladné speciální technické řešení.
- Použité algoritmy jsou relativně jednoduché a snadno implementovatelné.

4.2 Nevýhody

- Fragmentace – může dojít k takové posloupnosti zavádění úloh tak, že celkové využití paměti bude velmi nízké (i méně než 10 %). Mira fragmentace závisí na typické posloupnosti úloh a použitém algoritmu přidělování sekcí.
- I když nedojde k fragmentaci, může se stát, že pro vytvoření sekce není žádná volná oblast dost velká.
- Vyžaduje větší rozsah paměti a složitější operační systém než technika přidělování souvislého úseku paměti (díky multiprogramování).
- Paměť může obsahovat informace, které nebudou nikdy použity, a maximální velikost sekce je limitována rozsahem operační paměti.



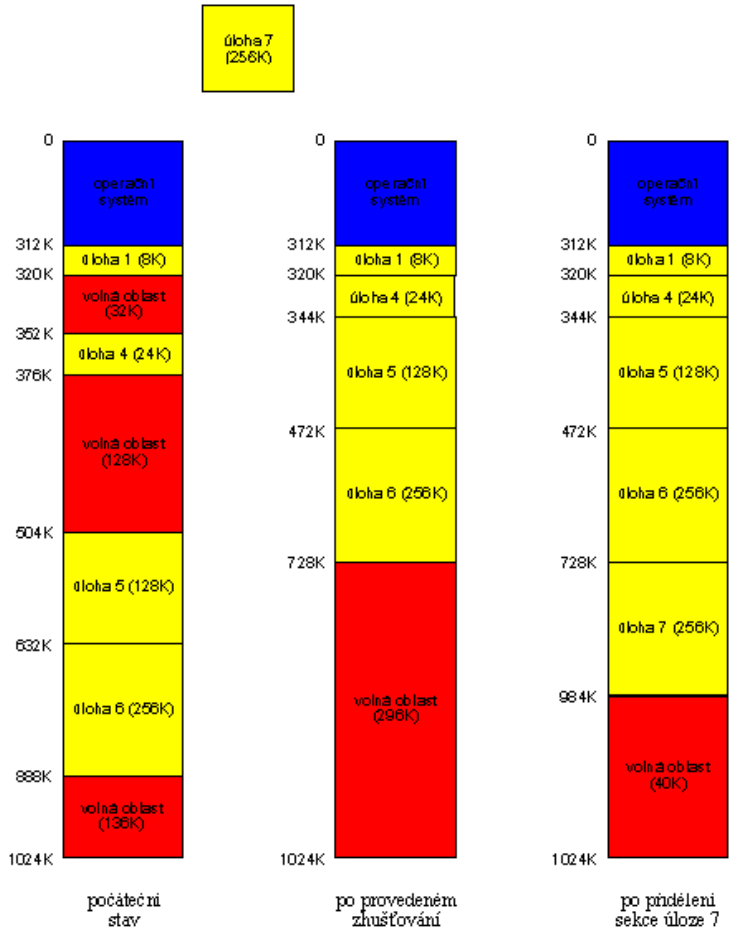
Obrázek 3 - Přidávání a uvolňování sekcí

5. DYNAMICKÉ PŘEMISŤOVÁNÍ SEKCÍ

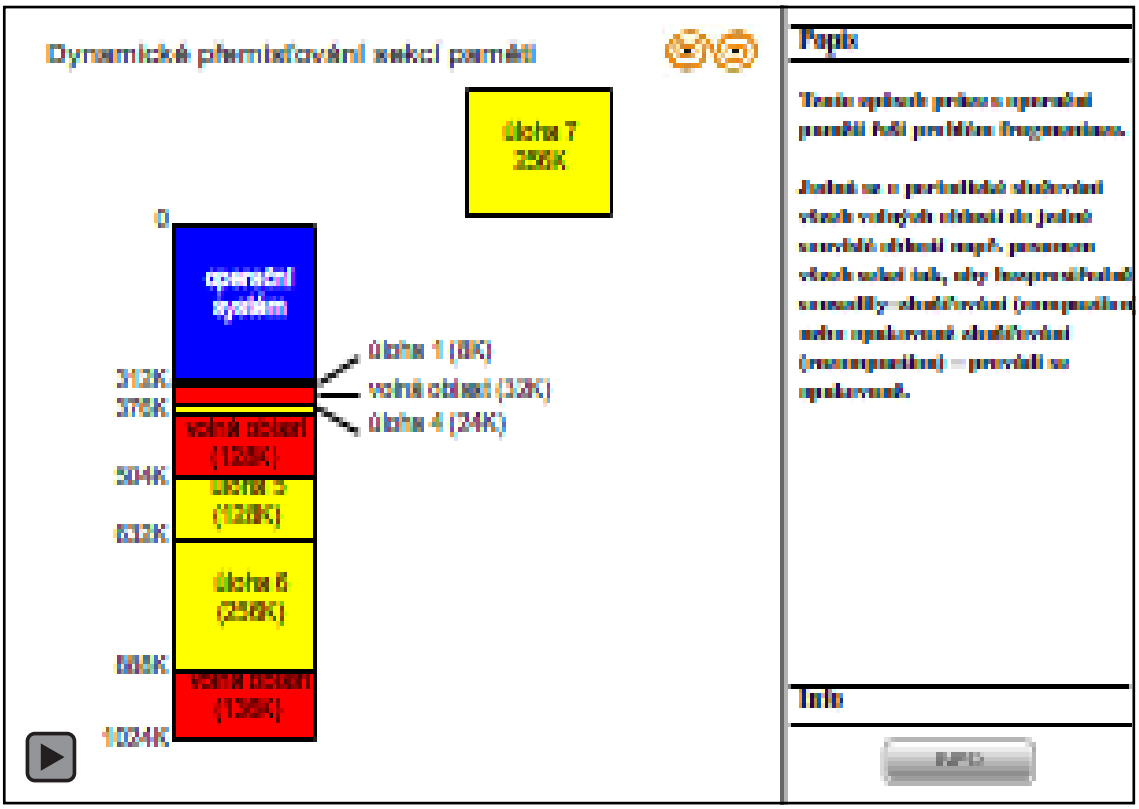
Tento způsob práce s operační pamětí řeší problém fragmentace. Jedná se o periodické slučování všech volných oblastí do jedné souvislé oblasti např. posunem všech sekcí tak, aby bezprostředně sousedily – zhušťování (compaction) nebo opakované zhušťování (recompaction) – provádí se opakovaně.

Dynamické přemisťování sekcí je principiálně jednoduché, není ovšem zaručeno, že po přesunu bude úloha dál probíhat korektně. Je třeba ošetřit části programu závislé na jeho umístění:

- obsahy bázových registrů
- instrukce s absolutními adresami
- seznamy parametrů
- datové struktury využívající adresových ukazatelů (seznamy apod.)



Obrázek 4 - Dynamické přemisťování sekcí paměti



Animce 2.1 Dynamické přemisťování sekcí paměti

Tyto informace je nutno po přesunu vždy modifikovat – dynamické přemístění (relocation). Zjištění adres nutných pro přesměrování je značně náročné, neefektivní a omezující.

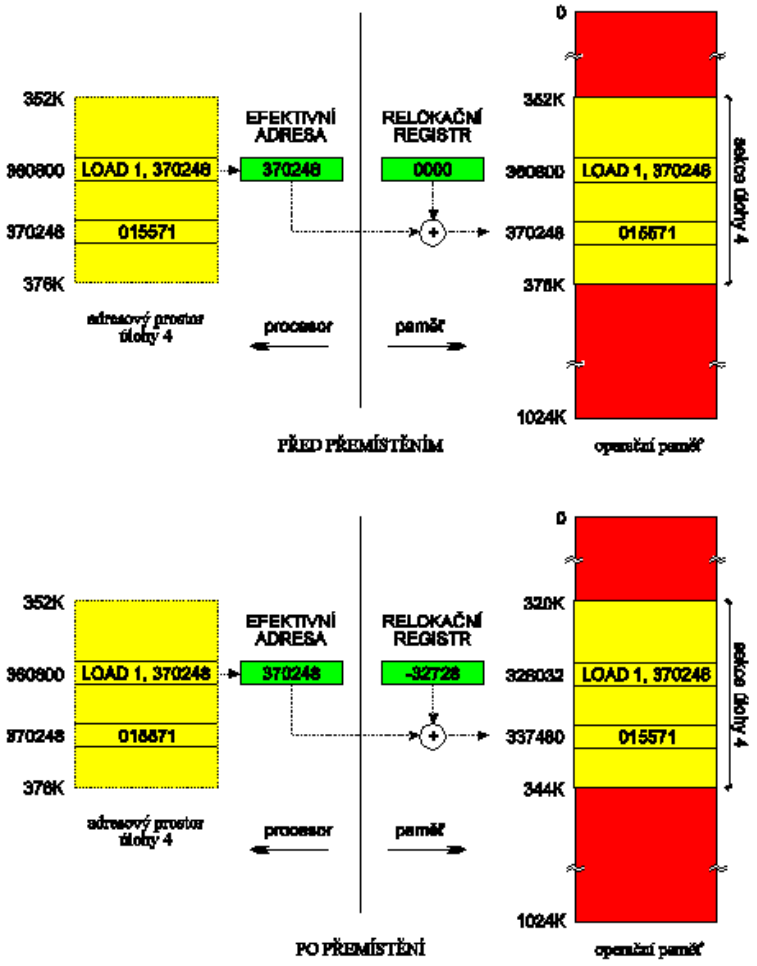
Problém dynamického přidělování sekcí lze řešit i jinak. Všechny úlohy, které mají být přemístěny vždy znovu zavést – možnost opakování výpočtu, možnost zhroucení, pokud ve výpočtu došlo k nějaké nevratné změně.

5.1 Požadavky na technické vybavení

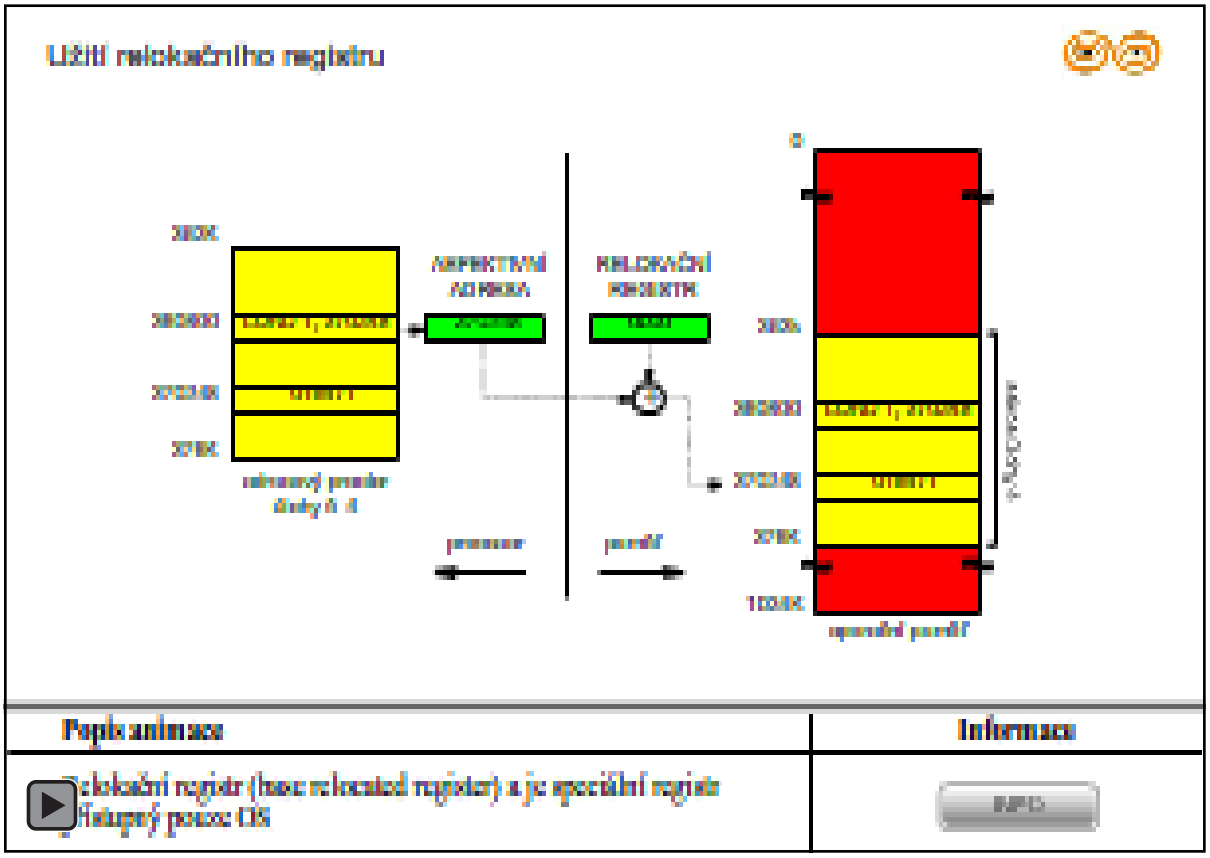
Typování dat – jedna z možností, jak řešit dynamické přemístění. Současně s uložením hodnoty do paměťového místa se uloží i její typ. Např. ke každému slovu se přidají dva bity pro zápis typu slova:

- 00 ... integer
- 10 ... string
- 01 ... real
- 11 ... adresa

Tyto bity nejsou uživatelskému programu přístupné, pracuje s nimi management paměti (při každém přiřazení se přiřazuje i typ apod.). Při přemístění sekce pak OS celkem snadno pozná, která slova jsou adresy a je třeba je změnit.



Obrázek 5 - Užití relokačního registru



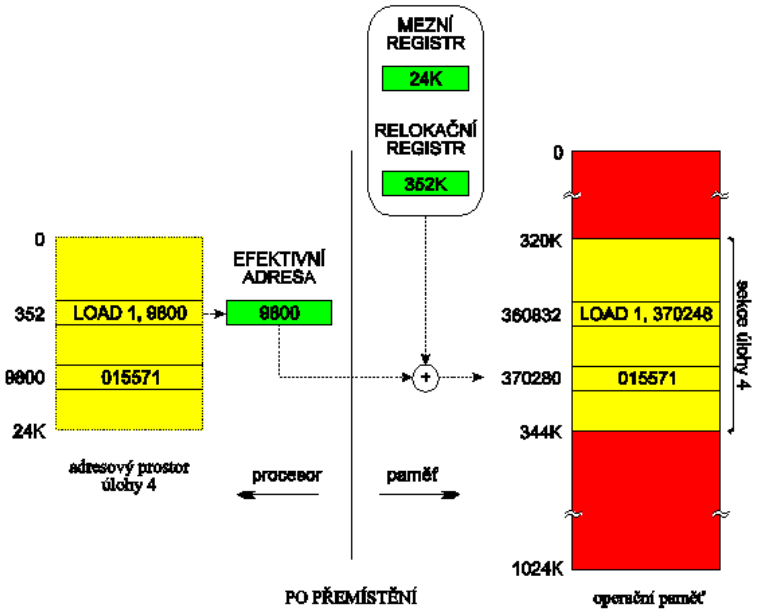
Animce 2.2 Užití relokačního registru

Vlastnosti dynamického přesouvání sekcí můžeme shrnout do následujících bodů:

- Zvyšuje paměťové nároky, zdržuje, vymyká se koncepci většiny počítačů.
- **Relokace**. Registr začátku programu – *relokační registr* (base relocated register) a *mezní registr* (bounds register) jsou dva speciální registry přístupné pouze OS. Při každém přístupu do paměti se k efektivní adrese připočítává obsah relokačního registru (efektivní adresa je adresa z adresového prostoru úlohy, určena procesorem z adresové části instrukce (viz obr. 5).
- Program tak vlastně nezná své skutečné umístění v paměti – nepotřebuje je. Každá instrukce se chová, jako by sekce po přesunu začínala na témže místě.
- Ochrana paměti se provádí nastavením mezního registru.
- Začátek paměťového prostoru úlohy je vždy vhodné volit 0 (na fyzickém umístění v paměti to nezáleží). Všechny adresy relokačních registrů jsou kladné a je zajištěna ochrana paměti, neboť lze snadno detekovat chybnou efektivní adresu – je záporná nebo větší než mezní registr.
- Stačí jedna dvojice registrů, která se vždy při přidělení procesoru jiné úloze aktualizuje a stávající hodnota se uloží, stejně jako PSW apod.

5.2 Výhody

- Eliminuje fragmentaci.
- Umožňuje vytvořit více sekcí (ve srovnání s dříve uvedenými technikami).
- Zlepšuje využití paměti i procesoru.



Obrázek 6 - Paměťový prostor úlohy 4, zavedené od adresy 0

5.3 Nevýhody

- Technické vybavení pro dynamické přemisťování zvyšuje cenu počítače a může zmenšit jeho rychlost. Dnes je však běžnou součástí i standardních procesorů.
- Část paměti může zůstat nevyužita, protože i po zhuštění může být rozsah volné oblasti menší, než je požadovaná velikost sekce.
- Jako u minulých technik může paměť obsahovat informaci, která nikdy nebude použita. Rozsah sekce je limitován rozsahem operační paměti.

6. STRÁNKOVÁNÍ PAMĚTI

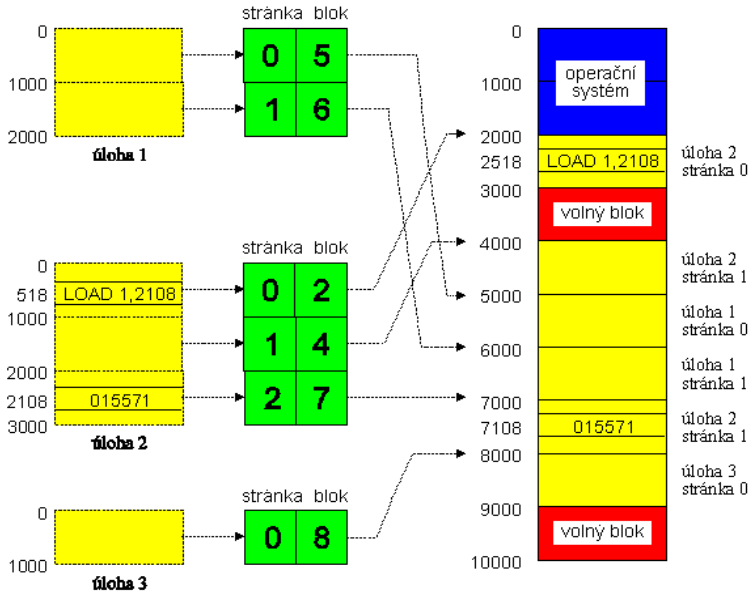
Metody přidělování sekcí vycházejí z předpokladu, že sekce musí tvořit **souvislou oblast**. Stránkování paměti umožňuje tento požadavek obejít.

- Adresový prostor každé úlohy se rozdělí na stejné úseky – *stránky*. Na stejně velké díly se rozdělí i operační paměť – *bloky, fyzické stránky, stránkové rámy.*
- Pomocí technického vybavení pro transformaci adres lze pak každou stránku vložit do libovolného bloku.
- Stránky jsou logicky souvislé vzhledem k uživatelskému programu, ale odpovídající bloky na sebe nemusí navazovat.
- Stejně jako u sekcí nemají transformace žádný vliv na uživatelskou úlohu.
- Pro každou stránku musí být vyhrazen samostatný registr. Registry – tabulky stránek – PMT tj. Page Map Tables. Registry jsou buď součástí technického vybavení, nebo jsou ve vyhrazeném úseku paměti.
- Efektivnost celé technologie značně ovlivňuje velikost stránky. Velká stránka vede k chování podobnému jako přidělování sekcí – fragmentace. Malá stránka vede k nadměrnému nárůstu položek tabulky stránek úlohy, což zvyšuje cenu výpočetního systému a zpomaluje jeho chod. Optimální velikost stránky je mezi 1kB až 4kB.

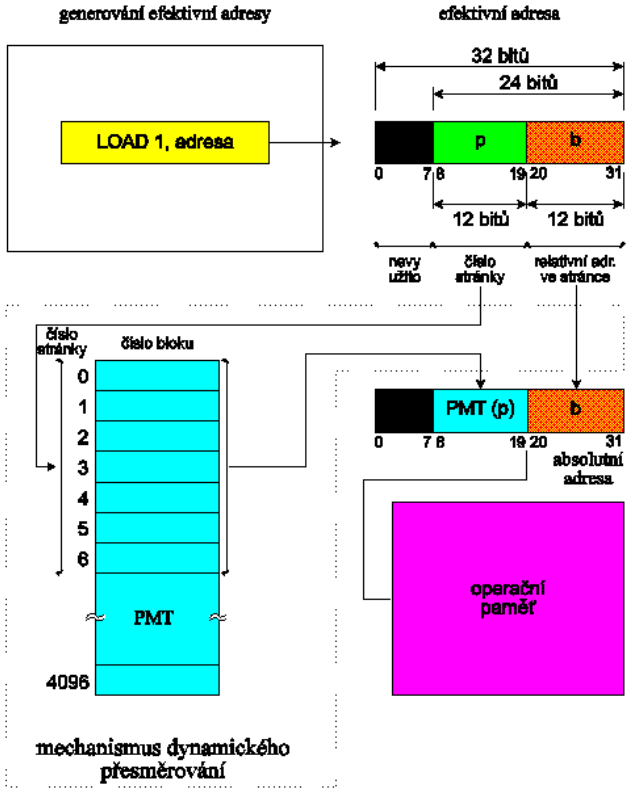
- Stránkování řeší problém fragmentace bez fyzických přesunů, snižuje tedy režii operačního systému.

Čtyři funkce modulu přidělování paměti lze realizovat takto:

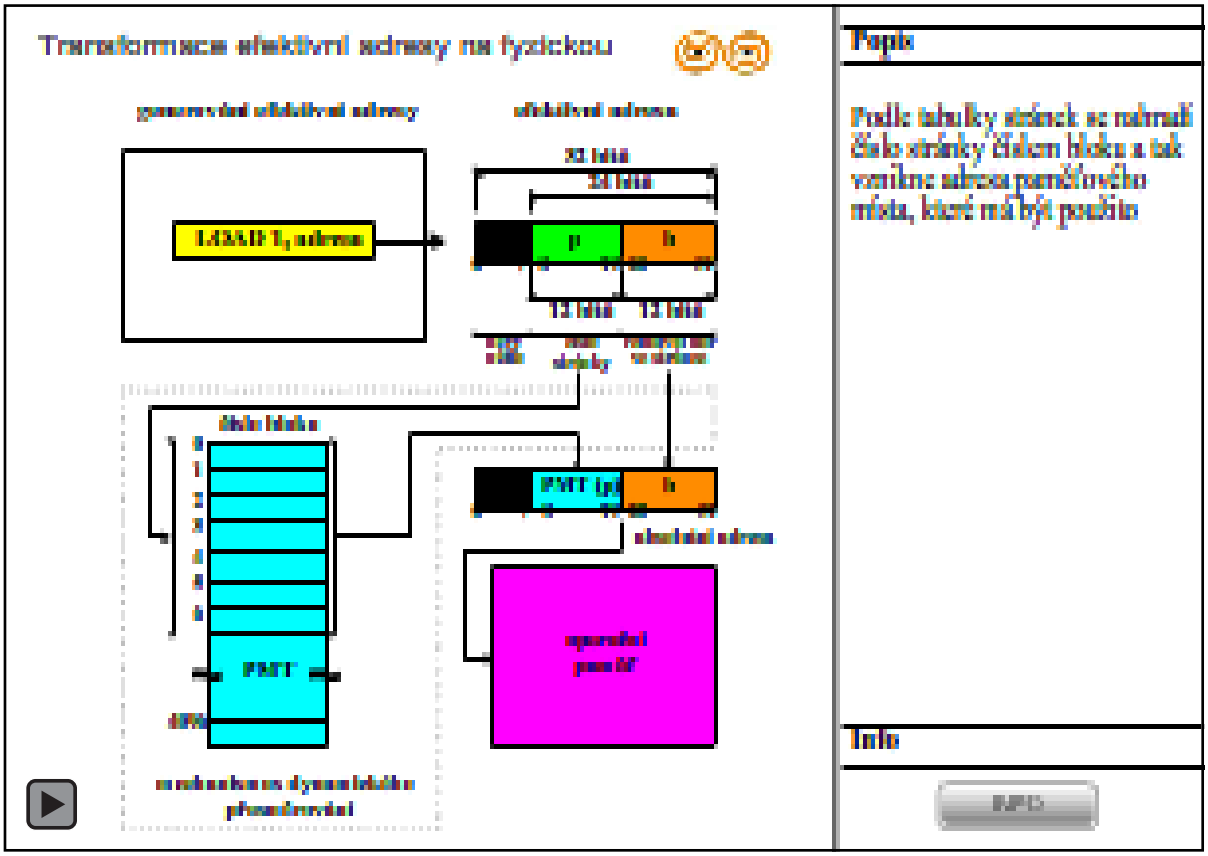
- Sledování stavu se provádí pomoci tabulek.
  - Pro každou úlohu jedna tabulka stránek – pro každou stránku jeden záznam (stránka, blok).
  - Systémová tabulka bloků paměti (Memory Block Table) – pro každý blok paměti záznam zda je "volný" nebo "užitý".
- Rozhodování o přidělování paměti provádí plánovač úloh – přidělí se množina volných bloků paměti, které se naleznou nejdříve.
- Přidělení paměti – stránky se zavedou do přiřazených bloků a aktualizují se záznamy v tabulce stránek a tabulce bloků.
- Uvolnění paměti – v tabulce bloků se odpovídající záznamy uvedou do stavu "volný"



Obrázek 7 – Stránkování paměti



Obrázek 8 - Transformace efektivní adresy na fyzickou



Animce 2.3 Transformace efektivní adresy na fyzickou

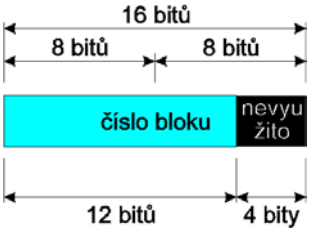
Na obr. 7 je jednoduchý příklad stránkování paměti. Rozsah stránky je $1000_{(16)}$ slabik. Všechny adresy budou uváděny v šestnáctkové soustavě. Adresový prostor úlohy 2 (3000

slabik) je rozdělen do tří stránek: str. 0 v bloku 2, str. 1 v bloku 4 a str. 2 v bloku 7. Např. instrukce *LOAD 1,2108* na adrese 0518 (str. 0, slabika 518) je ve skutečnosti na absolutní adrese 2518 (str. 2, slabika 518). Stejně tak konstanta 015571 na logické adrese 2108 je v operační paměti na adrese 7108.

K dispozici je 2000 slabik => je možno ještě uspokojit úlohu s paměťovými nároky max. 2000 slabik, a to **bez nutnosti přesunu** v paměti. V systému dynamického přidělování sekcí by bylo nutné provést zhušťování, aby vznikla jediná oblast požadovaného rozsahu. V systému stránkování se jednoduše úloze přiřadí dva volné bloky - např. blok 3 pro stránku 0 a blok 9 pro stránku 1.

Příklad

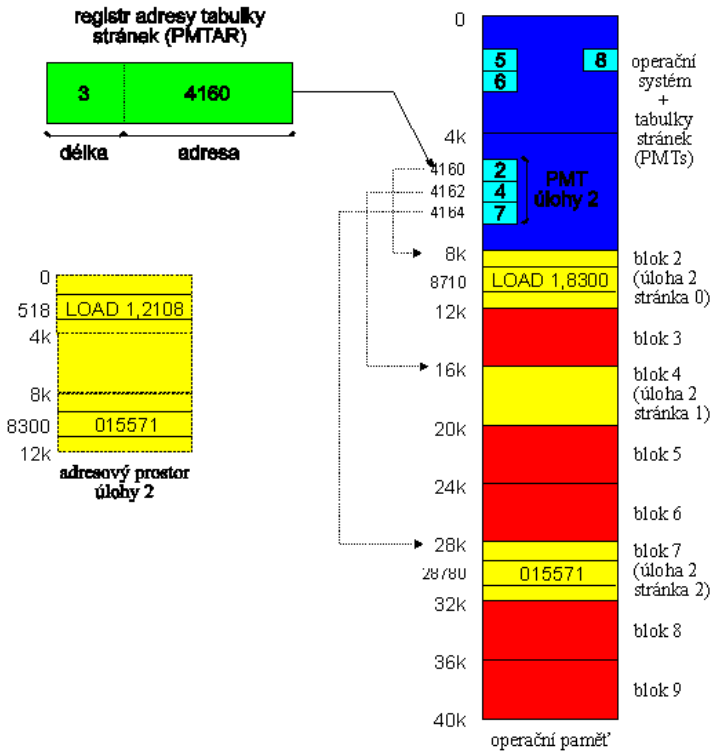
V systému IBM/370 je rozsah stránky 4 kB = 4096 B = 2^{12} B – pro adresování v rámci jedné stránky je třeba 12 bitu adresy. Pro co nejjednodušší transformaci adres se délka stránky volí jako mocnina 2. Celá efektivní adresa má deklu 24 b – systém adresuje maximálně 4096 bloků paměti a celá paměť systému může být max. 16777216 B = 16 MB. Každá položka v tabulce stránek (každé číslo bloku) musí byt uloženo ve dvou bytech – bity 0–11 obsahují číslo stránky, bity 12–15 nejsou využity.



Obrázek 9 - Číslo bloku

Adresa je rozdělena na 2 části:

- Bity 8–19 obsahuji číslo stránky.
- Bity 20–31 obsahují doplněk (relativní adresu ve stránce – offset).



Obrázek 10 - Stránka, blok, tabulka stránek, registr tabulky stránek

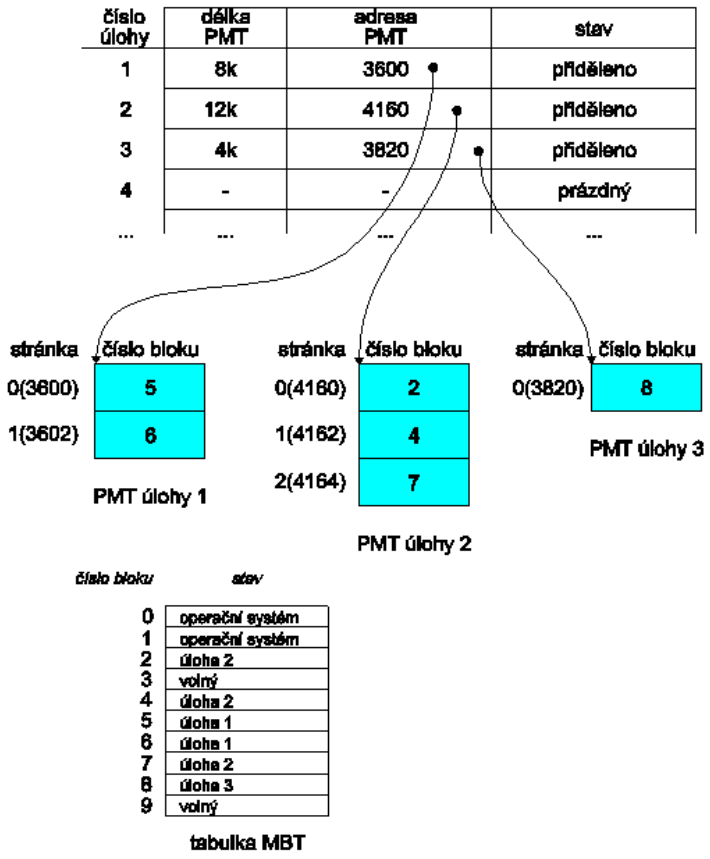
Podle tabulky stránek se nahradí číslo stránky číslem bloku a tak vznikne adresa paměťového místa, které má být použito. (viz obr. 8).

6.1 Tabulky stránek

Vyhradit pevné místo v paměti pro tabulku stránek aktuální úlohy je nevýhodné, protože by bylo nutno měnit tabulku pokaždé, když se procesor přiděluje jiné úloze. Lepší je ponechat možnost umístění tabulky na libovolném místě paměti (většinou se pro tabulky vyhradí část paměti určená pro OS) a toto místo určit pomocí tzv. registru adresy tabulky stránek – PMTAR (Page Map Table Register).

Když se procesor přidělí jiné úloze, je třeba změnit pouze obsah registru PMTAR tak, aby udával umístění tabulky stránek aktuální úlohy. Adresy stránek jdou v paměti souvisle za sebou, v PMTAR je umístění první adresy a jejich počet. (viz obr. 10).

Při technické realizaci stránkování tak jak bylo uvedeno, by se snížil výkon výpočetního systému na polovinu (každý přístup do paměti by vlastně znamenal dva – jeden pro zjištění absolutní adresy). Aby se tomuto zamezilo, užívají se tzv. **hybridní tabulky stránek**. Část tabulky je v paměti a část v superrychlých registrech, kterých není mnoho. Kdykoli je to možné, užijí se údaje z registru – přenos údajů do registru se provádí technickými prostředky tak, aby v nich byly vždy ty nejpoužívanější údaje. Tak lze obvykle redukovat režii stránkování paměti na méně než 5 %.



Obrázek 11 - Tabulky užívané při stránkování paměti

OS se stará o 3 základní tabulky:

- tabulka úloh (Job Table)
- tabulka bloků (Memory block table)
- tabulka stránek (Page Memory Table)

V tabulce úloh je pro každou úlohu položka obsahující údaje o umístění a délce její tabulky stránek a další stavové informace. Tabulka bloků udává stav každého bloku paměti (volný/užitý). (viz obr. 11).

Pokud některou stránku využívá více procesů (např. část OS) volí se technika překrývání stránek.

6.2 Výhody

- Eliminuje fragmentaci a umožňuje zvýšit počet úloh, pro které lze vytvořit paměťové prostory současně – lepší využití procesoru a operační paměti.
- Eliminuje se režie zhušťování, které je nezbytné při dynamickém přidělování sekcí.

6.3 Nevýhody

- Technické prostředky pro transformaci stránek zvyšují obvykle cenu výpočetního systému a současně snižují jeho rychlost.
- Uchovávaní tabulek (zejm. PMTs) zabírá část operační paměti. Současně se zvyšuje režie (čas procesoru) o dobu potřebnou k udržování těchto tabulek.

- Fragmentace jako taková je eliminována, vyskytuje se ale tzv. *vnitřní fragmentace*: 4kB stránka, úloha vyžaduje 5 kB paměti, tudíž dostane přiděleny 2 bloky. 3 kB druhého bloku zůstanou nevyužity. Průměrně připadá na každou úlohu nevyužita asi polovina (poslední) stránky. Problém zmiňuje zmenšení stránek, které ale zároveň zvyšuje režii správy paměti. Při tvorbě OS se stránkováním paměti je vhodné znát průměrný rozsah úloh.
- Nevyužita zůstává i paměť, která je sice volná, ale nestačí k pokrytí paměťových nároků přišedší úlohy. Jsou volné 2 stránky (8 kB) a úloha požaduje 8200 B. Kvuli 800 B zůstává nevyužito 8 kB operační paměti. Rozsah paměti nevyužité z těchto důvodů bývá přibližně polovina adresového prostoru průměrné úlohy.
- Stejně jako v minulých technikách i tady mohou být v paměti po celou dobu výpočtu zavedeny údaje, které se použijí buď jen zřídka, nebo dokonce nikdy.
- Adresový prostor je limitován fyzickým rozsahem operační paměti.

7. STRÁNKOVÁNÍ NA ŽÁDOST

Dosud vždy mohla být úloha zpracována jen tehdy, byla-li jí přidělena paměť pro celý adresový prostor. To často vedlo k nevyužití volné oblasti. Řešení je možné dvěma způsoby:

- Obrovské paměti – ekonomicky (i technicky) více méně nemožné.
- Simulace obrovské paměti – **virtuální (zdánlivá) paměť** – dvě hlavní techniky vedoucí k vytvoření virtuální paměti jsou stránkování na žádost a segmentace paměti.

Upustíme-li od požadavku umístění celého adresového prostoru úlohy do operační paměti, pak součet všech adresových prostorů jednotlivých úloh může překročit celkovou kapacitu operační paměti.

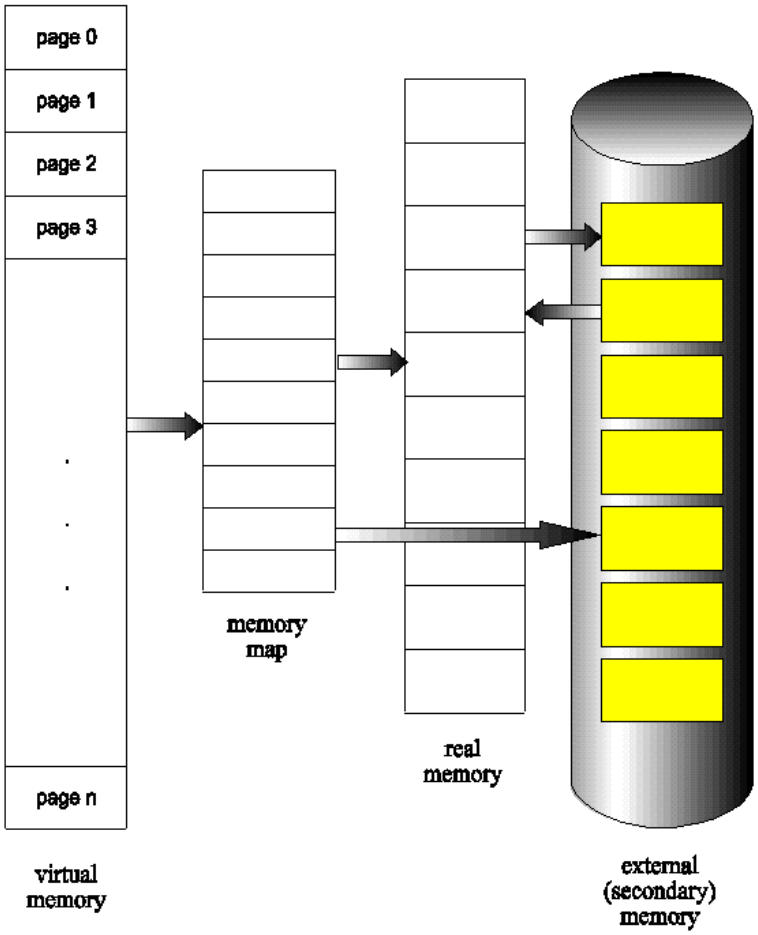
V paměti je jen právě používaná část úlohy.

Většina programů během svého konkrétního průběhu využívá jen malou část svého adresového prostoru, protože:

- Uživatelské programy pro ošetření chyb se užijí, jen když k chybě dojde (většinou je to snad výjimka).
- Logické větve vylučují současný průběh alternativních částí programu.
- Mnoha tabulkám (statickým paměťovým strukturám) je přiděleno pevné (maximální) množství adresového prostoru, které se obvykle nevyužije.
- Souběh mnoha podprogramů se časově vylučuje "vstupní fáze", "výpočetní fáze" a "výstupní fáze" apod.

Toto řešení ale přináší některé nové problémy:

- Situace, kdy se program odvolává na oblast adresového prostoru, která není zavedena v operační paměti.
- Strategie rozhodování, které stránky mají být uchovány v operační paměti.



Obrázek 12 - Princip vytvoření virtuální paměti

Technické řešení:

Rozšířit tabulku stránek o stavový bit, obsahující false – stránka je v operační paměti, nebo true – stránka není v operační paměti. Pokud technické prostředky pro transformaci adresy zjistí, že stavový bit požadované stránky je 1, vyvolá se **page interrupt – výpadek stránky.** V tomto případě se nejedná o chybu uživatelského programu (na rozdíl např. od přerušení v důsledku porušení ochrany paměti).

OS ošetří page interrupt zavedením požadované stránky do operační paměti a aktualizací stavového bitu této stránky. Poté se znovu zahájí provádění instrukce uživatelského programu. Stránka byla do paměti **zavedena na žádost**. Při naplánování zpracování úlohy je do operační paměti zavedena pouze první "startovací" stránka. Ostatní jsou pak zaváděny na žádost – nepotřebné stránky se do operační paměti vůbec nedostanou. Kopie celého adresového prostoru úlohy se uchovává v záložní paměti (obvykle zařízení typu disk) a odtud se čtou požadované stránky. Záložní paměť bývá realizována buď jako speciální soubor v rámci souborového systému nebo jako samostatná oblast (partition) disku.

Možné komplikace:

Co v okamžiku, kdy není paměťový prostor pro zavedení další stránky? Řeší se pomocí **výměny stránek (page swapping)**. Obětovaná stránka (ta, která byla určena k vyřazení z operační paměti) se nejprve překopíruje do záložní paměti a pak se místo ní zavede stránka nová. Optimální algoritmus nahrazování stránek je předmětem intenzivního výzkumu. Může se stát, že vyhozená stránka je bezprostředně na to požadována a je nutno ji znovu zavést – toto je nežádoucí stav snižující efektivitu operačního systému!

Čtyři funkce modulu přidělováni paměti jsou při stránkování na žádost složitější a flexibilnější:

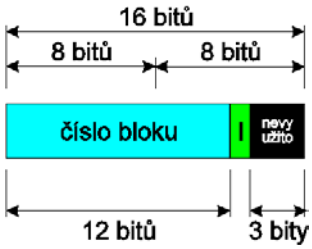
- Sledování stavu paměti – tabulky:
  - tabulky stránek PMTs – jedna tabulka pro adresový prostor každé úlohy,
  - tabulka bloků paměti MBT – jedna v systému,
  - tabulky souborů FMT (File Map Tables) – pro každý adresový prostor úlohy.
- Rozhodování o přidělení paměti – částečně provádí plánovač úloh. Přidělení je však determinováno také výpadky stránek.
- Přidělování paměti – je nutné najit "vhodný" blok paměti pro startovací stránku a změnit stavový bit bloku.
- Uvolňování paměti – není-li vhodný blok paměti k dispozici, musí být některý z obsazených uvolněn. Je-li úloha dokončena, všechny bloky, které obsadila, se uvolní.

7.1 Technické prostředky

U stránkování na žádost je třeba oproti obyčejnému stránkování rozšířit technické vybavení o tři důležité funkce:

- Stavový bit v tabulce stránek – indikace zda stránka je či není v operační paměti.
- Rozšíření mechanismu přerušení o "výpadek stránky" (tj. byl učiněn pokus o přístup ke stránce, která není přítomna v operační paměti), aby mohl OS tuto situaci ošetřit.
- Záznam o používání stránky (tj. počet přístupů do stránky, počet čtení, resp. zápisu do stránky apod.) pro strategii rozhodování, kterou stránku je v případě potřeby možné odstranit z operační paměti.

Možné řešení u systému IBM/370 je uvedeno na následujícím obrázku. Položky v PMT se liší pouze tím, že jeden z nevyužitých bitů se použije jako stavový bit.



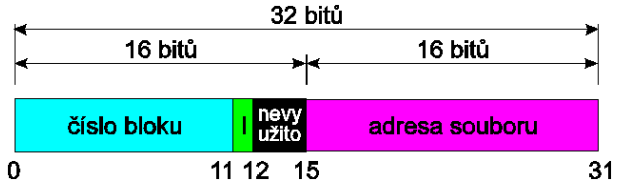
Obrázek 13 - Stavový bit stránky v čísle bloku

Každý blok paměti popisují navíc další dva bity:

- *bit indikace paměti* (reference bit) – při čtení libovolného místa z bloku se nastavuje na 1,
- *bit indikace změny* (change bit) – při zápisu do libovolného místa v bloku se nastavuje na 1.

Oba bity může nastavit jádro v privilegovaném režimu.

7.2 Algoritmy programového vybavení



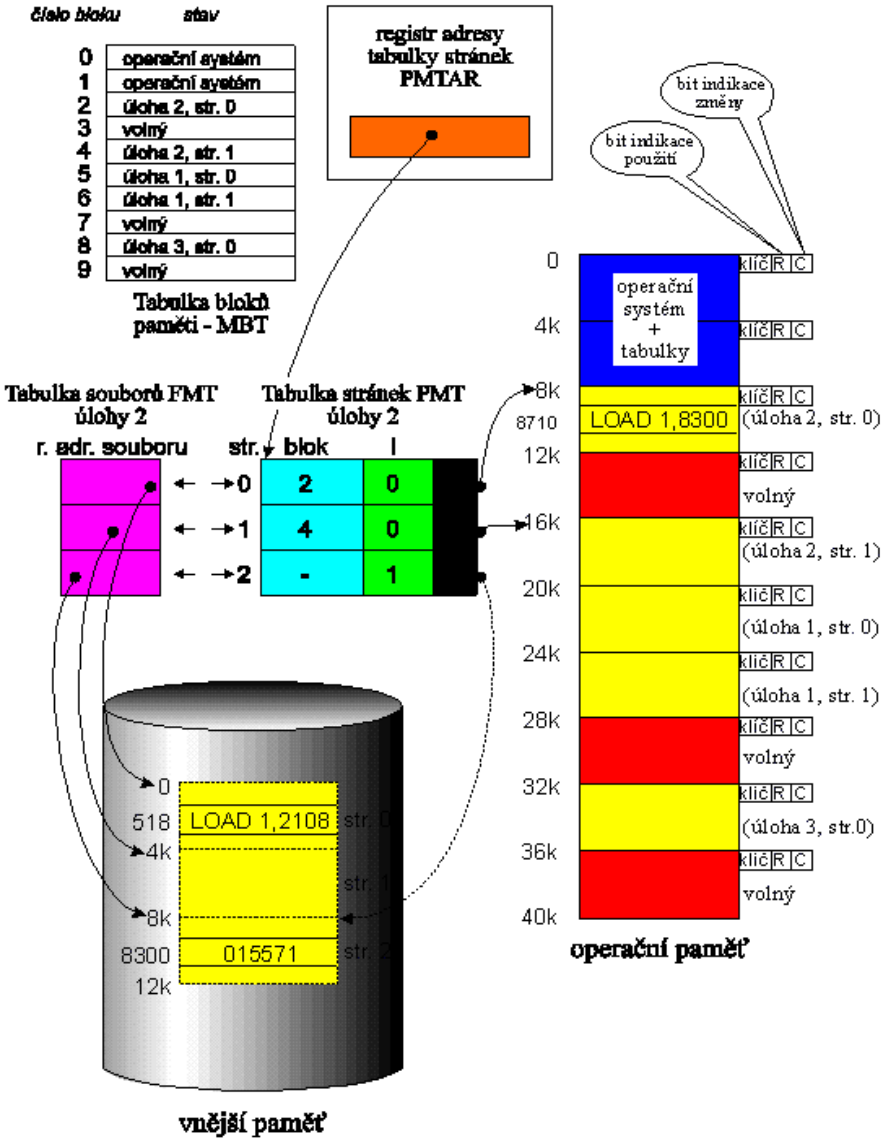
Obrázek 14 - Kompletní adresa bloku

Modul přidělování paměti musí spolupracovat se správou souboru (přístup k záložní paměti). Každá stránka má jednoznačně přiraženou adresu v záložní paměti – *adresu souboru* (file address). Tato adresa pak logicky náleží ke každé tabulce stránek, viz obr. 14.

V předchozím obrázku se předpokládá, že adresa souboru má délku 16 bitů.

7.2.1 Ošetření výpadku stránky

Paměť je přidělována a odebírána i během provádění úlohy pomocí mechanismu přerušení. Při stránkování na žádost velmi úzce spolupracuje programové a technické vybavení (se systémem souborů pracuje programové vybavení, viz obr. 16).



Obrázek 15 - Vztah tabulky souboru a tabulky stránek

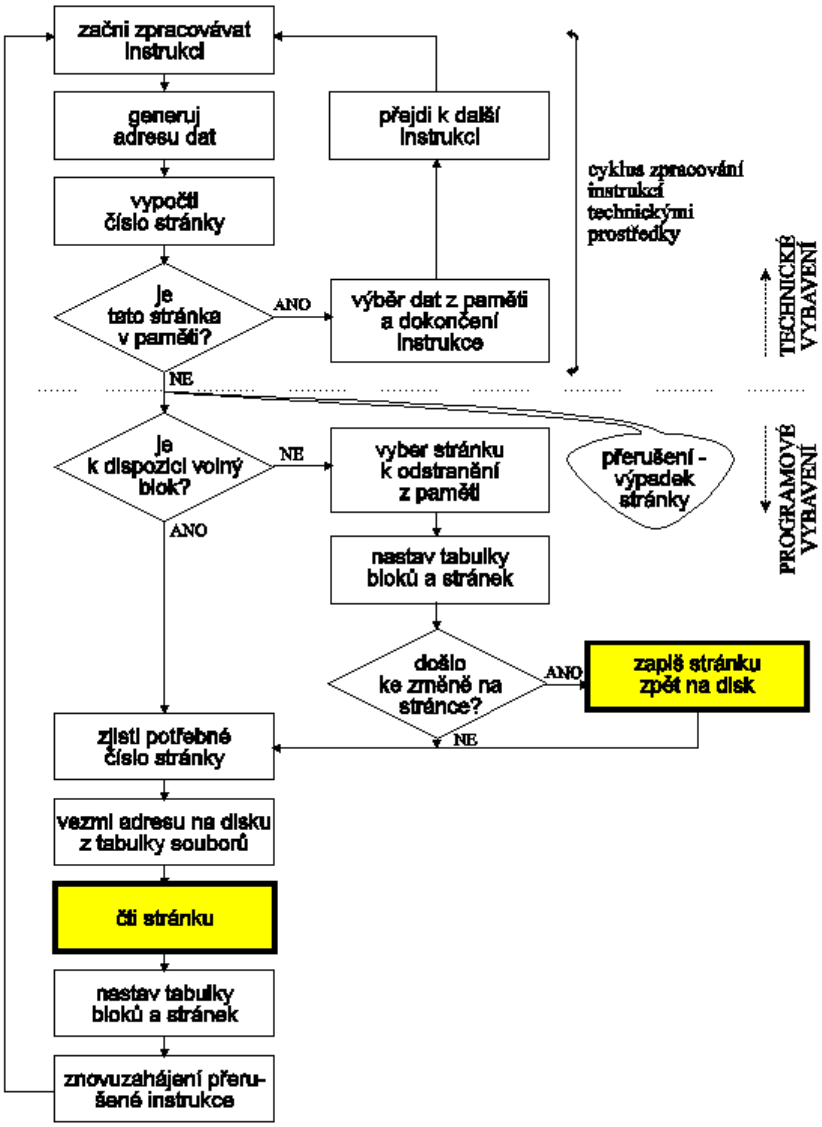
První část vývojového diagramu se provádí nejčastěji, proto je obvykle implementovaná jako součást technického vybaveni pro transformaci adres. Druhá část je přerušovací modul v OS.

Přítomnost stránky v paměti se zjišťuje podle indikačního bitu v odpovídající položce tabulky stránek (0 – stránka je v paměti; 1 – stránka není v paměti). Vždy je nutno prověřit všechny adresy v instrukci obsažené a adresu, na které je instrukce uložena – spouštění instrukcí pomoci instrukce EX tj. proveď instrukci na uvedené adrese (nepřímé adresování instrukce) apod.

U systémů s nepřímým adresováním na několika úrovních může nastat v podstatě neomezený počet výpadků stránek během vykonání jediné instrukce.

Je-li v paměti volný blok, zjistí se z k tomu určené speciální adresy v paměti číslo požadované stránky – odpovídající adresa stránky se zjistí z tabulky souborů – stránka se zavede do paměti – aktualizuje se tabulka stránek a tabulka bloků paměti – a znovu se zahájí přerušená instrukce.

Není-li žádný volný blok k dispozici, je nutno některou stránku z paměti odstranit. Po výběru takové stránky se musí ošetřit její *bit změny*. Má-li hodnotu 0, je kopie stránky v záložní paměti identická s tou v operační paměti a stránku není třeba znovu zapisovat do záložní paměti před jejím odstraněním z paměti operační. Jinak se stránka zapíše do záložní paměti a blok se uvolní.



Obrázek 16 - Interakce mezi programovým a softwarovým vybavením

Součástí ošetření výpadku stránky mohou být až 2 I/O operace. Během nich je možné přidělit procesor jiné úloze. To ale vyžaduje rozšířit stavy bloků o další položku *přenos* (in transit), protože je třeba ošetřit situaci kdy je do paměti zaváděna stránka úlohy 1, a proto je procesor přidělen úloze 2. Přitom v úloze 2 nastane výpadek stránky a tato úloha potřebuje volný blok paměti. Pokud by blok úlohy 1 byl označen jako **přidělený**, pak by se přerušovací modul mohl pokusit o jeho uvolnění, ale zápis stránky úlohy 1 nebyl dosud dokončen. Pokud by blok úlohy 1 byl označen jako **volný**, pak by sice mohl byt přidělen úloze 2, ale potom by obě úlohy vyžadovaly mít svou stránku v témže bloku. Pokud je blok označen jako **přenos**, je v daném okamžiku nepřidělitelný a teprve po dokončení I/O operace může být jeho stav změněn na **přidělený** nebo **volný**.

V systémech se značnou frekvenci I/O stránek se využívá rychlá cache paměť a chytré přidělování periferií.

7.2.2 Algoritmy nahrazování stránek v operační paměti

Řešení problému „vyber stránku k odstranění z paměti“:

- Nahradí se vždy stránka v bloku 3. (tj. první blok za OS) – velmi jednoduché a velmi neefektivní – mohl by vést až k zahlcení systému. Uvedeno jen pro nastínění problému.
- FIFO (First In – First Out) – první tam, první ven – uvolněna je nejstarší stránka.
- LRU (Least Recently Used) – uvolněna je nejdéle nepoužitá stránka.

Příklad:

Problém je analogický problému prodejny s plnými regály, která musí zavést nový výrobek. Je nutné určit algoritmus, podle kterého se vybere zboží k odstranění do skladu. Je možné zvolit to, které je v prodejně nejdéle (FIFO) – mnohdy úspěšná strategie, která vede k odstranění zboží s prošlou záruční dobou. U zboží s dlouhodobou trvanlivosti (čaj, cukr apod.) však vede k problémům.

Lépe je nahradit zboží, na kterém je v prodejně „nejvíce prachu“ – to, které nebylo dlouho požadováno (LRU). Pokud je algoritmus zvolen vhodně, do skladu se pro odložené zboží musí jen zřídka.

7.2.3 Simulace algoritmu nahrazování stránek

Ohodnoťme nahrazovací algoritmus řetězcem volání paměti a spočítejme počet výpadků stránky. Řetězec nazveme *referenční řetězec* (reference string). Sled adres nechť je generován uměle (např. generátorem náhodných čísel) nebo přímým záznamem každého volání paměti skutečného programu. To dává přibližně 1 milion adres za sekundu.

K vytvoření *referenčního řetězce* užijme čísla stránek, na kterých se jednotlivé instrukce nachází.

Pokud je volána stránka *p*, potom žádné bezprostředně následující volání stránky *p* nevyvolá výpadek stránky, a proto se tato volání v řetězci neprojeví.

Při sledování procesu je možné získat např. následující sled adres:

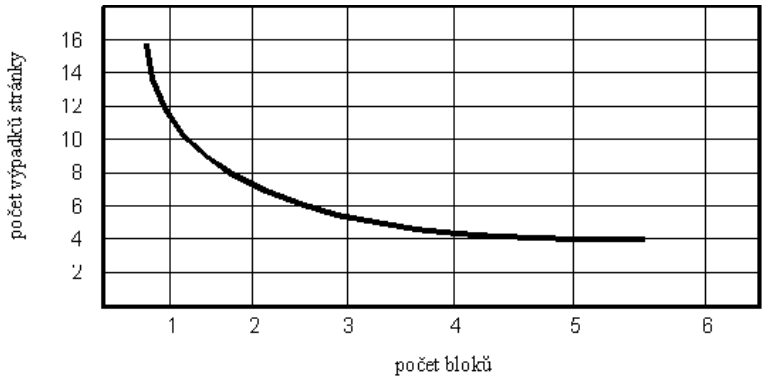
```
0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102,
0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102,
0105
```

Pokud má blok paměti velikost 100 bytů, je výsledný referenční řetězec:

```
1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1
```

Pro určení počtu výpadků stránky je nutno znát počet přístupných bloků. S klesajícím počtem volných bloků roste počet výpadků stránky. V horním případě, máme-li tři nebo více volných

bloků, dojde pouze ke třem výpadkům stránky nutným k zavedení jednotlivých stránek (1, 4 a 6). Je-li volný toliko jeden blok, vyvolá výměnu každé volání stránky a výpadků bude 11.



Obrázek 17 - Graf závislosti počtu výpadků stránky na počtu dostupných bloků

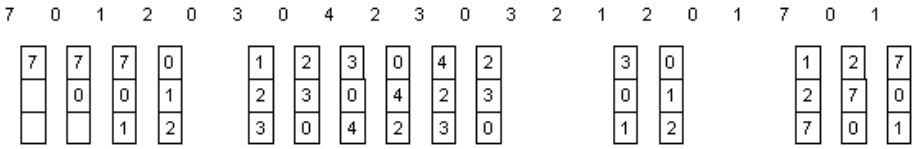
Pro demonstraci algoritmu výměny stránek užijme referenční řetězec:

`7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1`

pro paměť se 3 volnými bloky.

FIFO algoritmus

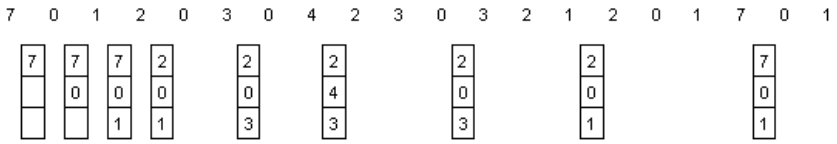
Pro výměnu vybírá stránku, která je v paměti nejdéle. Není nutno zaznamenávat čas zavedení každé stránky. Stačí vytvořit FIFO frontu stránek zavedených v paměti a nahradit vždy tu na vrcholu fronty. Každá zavedená stránka se zařadí na konec fronty. Pro náš referenční řetězec a paměť se 3 volnými bloky, by sled výměn v paměti byl následující:



Obrázek 18 - FIFO algoritmus výměny stránek

FIFO algoritmus výměny je jednoduchý na pochopení i programováni, nepracuje však vždy dobře – nová stránka může nahradit inicializační část programu, která už nebude nikdy použita, nebo naopak stránka s velmi často užívanými proměnnými, která se bude muset brzy zavádět znovu.

Principem „lepších“ algoritmů je nahradit stránku, která nebude v budoucnu dlouho používaná (viz obr. 19). Tyto algoritmy jsou však velmi těžko implementovatelné, protože vyžadují dopředu referenční řetězec výměn. Jejich použiti je v teoretické rovině pro srovnání optimálnosti toho kterého algoritmu. V analogii s dříve uváděným příkladem by se problém přemisťování zboží ze skladu do prodejny řešil pomocí dlouhodobé přesné znalosti vývoje poptávky.



Obrázek 19 - Optimální algoritmus výměny stránek

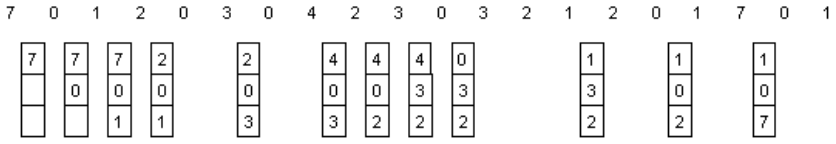
Optimální algoritmus není implementovatelný, ale je možno implementovat jeho aproximaci. Touto aproximací je

LRU algoritmus

Vychází z toho, že stránka, na kterou se dlouho nevyskytl žádný odkaz, nebude pravděpodobně potřeba ani nadále a naopak – základní myšlenka *teorie lokality*. Tato strategie je nejefektivnější ze všech, které pracují s uplynulým časem na rozdíl od OPT a MIN.

Strategie LRU patří do tzv. zásobníkových algoritmů, tj. algoritmy, u nichž nikdy nemůže vést zvýšení počtu bloků paměti ke zvýšení výpadků stránky. LRU strategie asociuje každou stránku s časem, kdy byla naposledy použita a pokud musí být stránka z paměti vyřazena, LRU vybere tu, která nebyla použita nejdéle.

Aplikaci LRU strategie na náš referenční řetězec ukazuje obr. 20. Během LRU dochází ke 12 výpadkům stránek; prvních 5 výpadků je stejných jako u algoritmu OPT. V okamžiku, kdy chybí stránka 4, LRU zjistí, že nejdéle nepoužitá je v paměti stránka 2, a tudíž bude nahrazena stránkou 4 atd.



Obrázek 20 - LRU algoritmus výměny stránek

Počet výpadků stránky: 9(OPT) < 12(LRU) < 15(FIFO)

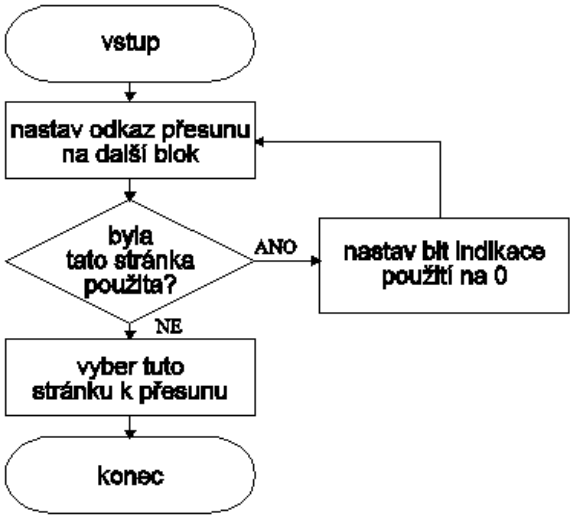
LRU algoritmus je často implementován a funguje celkem dobře. Největší problém je, **jak** ho implementovat, protože tento algoritmus vyžaduje velkou hardwarovou podporu. Problém je v tom, jak udržet pořadí užití jednotlivých stránek. 2 implementace jsou proveditelné:

- **Čítače:** Každý blok paměti se asociuje s polem *cas_uziti* a do CPU se přidá logický časovač, který je inkrementován při každém použití paměti. Pokaždé, když je operace s rámcem dokončena, zkopíruje se obsah logického časovače do pole *cas_uziti* dané stránky. Takto máme uchovaný „čas“ posledního použiti každé stránky a nahrazovat se bude vždy stránka s nejmenší hodnotou v registru *cas_uziti*. Tato technologie vyžaduje zápis do paměti (do registru *cas_uziti*) při každém přístupu do paměti, časy musí být aktualizovány při každé změně obsahu bloku a musí být ošetřeno přetečení časovače.
- **Zásobník:** Druhou cestou je vytvoření zásobníku čísel bloků. Vždy, když je na některý blok odkazováno, je jeho číslo smazáno z předešlého výskytu v zásobníku a uloženo na vrchol. V tomto případě je na vrcholu zásobníku nejčastěji používaná stránka a na konci ta nejméně. Implementace zásobníku musí umožňovat odebírat položky z jeho těla. Vybrání položky z těla zásobníku a její uložení na vrchol způsobí při nejhorším změnu 6 ukazatelů. Tento přístup je pro implementaci přiměřený.

Nevýhodou LRU algoritmu je nutnost aktualizace záznamu o užití stránky po každém odkazu na stránku (u FIFO jen v případě, že byla stránka zaváděna do paměti) => zpomalení a zdražení výpočetního systému. Proto se tato implementace v podstatě nepoužívá a využívá se aproximace LRU algoritmu.

Aproximace LRU algoritmu nevyužívá frontu žádostí o jednotlivé stránky, ale svou činnost staví na *referenčních bitech - bitech indikace paměti* jednotlivých stránek (viz 3. kap. 5.1). Podle nich algoritmus neví přesný sled užití jednotlivých stránek, ví ale, které stránky v

minulosti použity byly (jejich *referenční bit* má hodnotu 1) a které ne. Fungování této aproximace algoritmu znázorňuje následující diagram:



Obrázek 21 - Aproximace LRU algoritmu

Aproximace využívá ukazatel přesunu, který se cyklicky pohybuje po jednotlivých stránkách. V okamžiku výpadku stránky je projíždí jednu po druhé, dokud nenarazí na tu, která má referenční bit 0. Ta se použije pro výměnu. U těch, které mají referenční bit roven jedné, je tento vynulován, aby se zjistilo, zda do dalšího průchodu ukazatele byl nebo nebyl blok použít.

7.3 Výhody

- Všechny výhody až dosud uváděné, zejména eliminace fragmentace bez zhušťování.
- Rozsáhlá virtuální paměť – adresový prostor úlohy není omezen fyzickým rozsahem paměti.
- Efektivnější využití operační paměti – málo, příp. vůbec nevyužité části adresového prostoru se nezavádějí do operační paměti (často 25 % i vice).
- Neomezené multiprogramování – není omezeno fyzickým rozsahem operační paměti

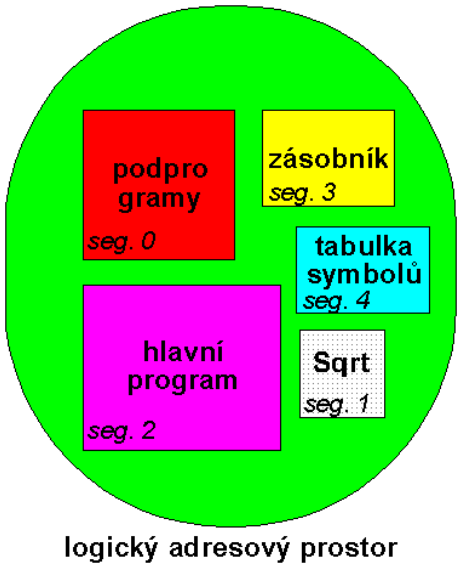
7.4 Nevýhody

- Nákladné technické prostředky, vnitřní fragmentace, režie procesoru, paměťový prostor pro tabulky – náročnější než u běžného stránkování.
- Nutno vytvořit prostředky pro prevenci zahlcení systému (případ, kdy čas procesoru pro ošetření výpadků stránek neúměrně stoupne na úkor uživatelských úloh.

8. SEGMENTACE PAMĚTI

Segmentace paměti přináší zcela odlišný přístup k práci s pamětí než u všech předešlých, kde se pracovalo technickými prostředky s pamětí tak, že uživatelská úloha o ničem nevěděla. Segmentace paměť lépe využívá a usnadňuje programování.

Jaké je vidění paměti uživatelem/programátorem? Jako lineární pole bytů, nebo jako množina různě velkých bloků různých účelů? Druhý pohled je jistě efektivnější (viz obr. 22). Nazvat ony bloky paměti segmenty je pak už jen formální krok.



Obrázek 22 - Uživatelský pohled na program

Segment lze definovat jako logické seskupení informací tj. např. hlavní program, podprogramy, datová oblast apod.

Adresový prostor každé úlohy je potom tvořen několika segmenty, jak ukazuje např. obr. 22. Technika přidělování paměti segmentům, se nazývá *segmentace*

Při *segmentaci paměti* musí každý odkaz do ní obsahovat

- číslo segmentu – (segment) a
- paměťové místo v segmentu – (offset).

I když je segmentace pro programátory viditelná, rozdíly mezi běžným a segmentovaným paměťovým prostorem program neovlivní. Pascalský překladač např. může vytvořit segmenty pro (1) globální proměnné; (2) zásobník adres volání procedur; (3) kódovou část každé procedury nebo funkce; a (4) lokální proměnné každé procedury nebo funkce. Loader potom přiřadí každému segmentu jeho číslo.

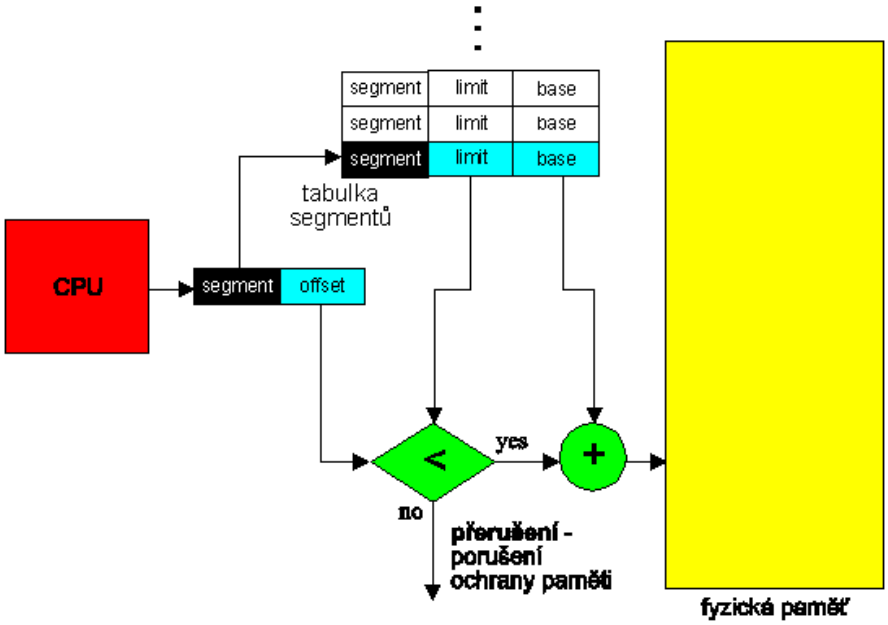
8.1 Technické prostředky

Technické prostředky musí umět překapovat dvourozměrnou adresu (segment, offset) na jednorozměrnou fyzickou adresu daného paměťového místa. Stejně jako u stránkování se k tomu využívá převodní tabulka - *tabulka segmentů* (segment table). Každá položka tabulky obsahuje *počáteční adresu segmentu* (segment base) a *rozsah segmentu* (segment limit). Užití tabulky segmentů ilustruje obr. 23. Každý proces má svou tabulku segmentů asociovanou s PCB (process control block), podle kterého se orientuje dispečer.

Logická adresa se skládá z čísla segmentu (segment) a relativní adresy slova v rámci segmentu (offset). Segment se použije jako index v tabulce segmentů. Offset, jakožto relativní adresa v rámci segmentu nesmí být vyšší než limit (délka segmentu). Pokud tomu tak je, nastává přerušení v důsledku porušení ochrany paměti. Je-li vše v pořádku, sečte se offset a base segment, čímž vznikne fyzická adresa, která se použije pro adresaci paměti.

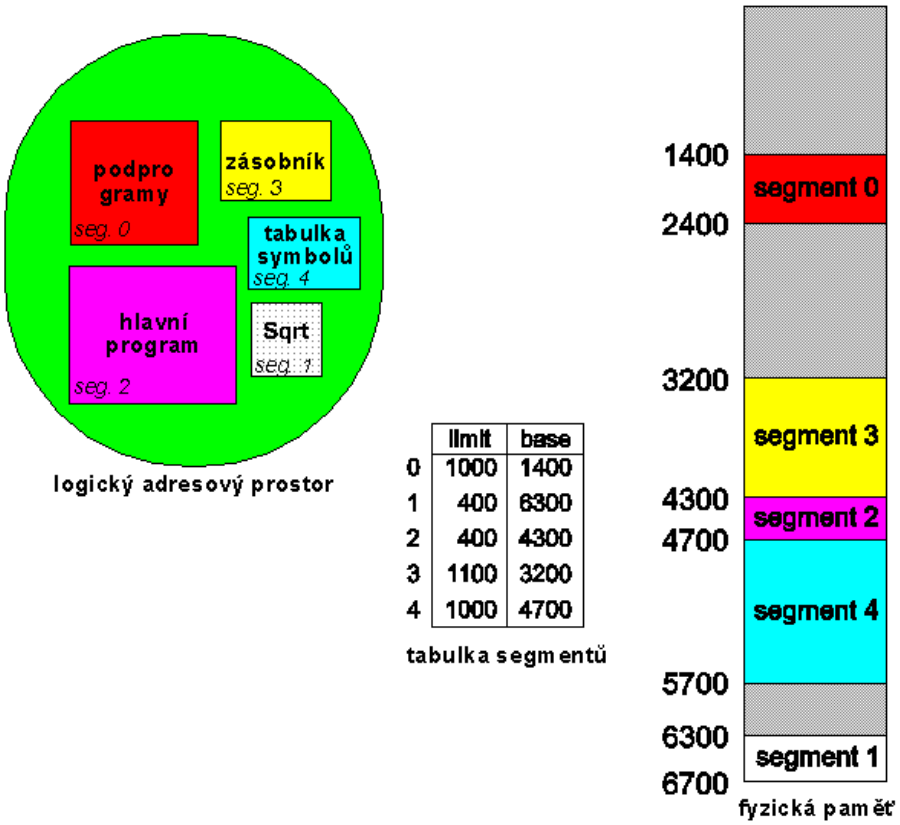
Pro ukázku reálné situace segmentace paměti poslouží příklad z obr. 22. Výsledná situace po zavedení úlohy do paměti je vidět na Obr. 24. 5 segmentů je očíslováno od 0 do 4. Všechny jsou uloženy ve fyzické paměti. Tabulka segmentů má vlastní položku pro každý segment a obsahuje počáteční adresu a délku segmentu. Např. segment (2) má délku 400 bytů a je uložen v paměti od adresy 4300. Odkaz na 53. byte segmentu (offset) je přemapován na 4300

+ 53 = 4353. Odkaz na byte 1222 segmentu 0 vyvolá přerušení v důsledku porušení ochrany paměti, neboť tento segment má délku pouze 1000 bytů.



Obrázek 23 - Technické řešeni segmentace paměti

Stejně jako u stránkování mohou být tabulky segmentů uloženy buď v paměti, nebo v rychlých registrech. V případě uložení v registrech je odezva rychlejší a součet offsetu s počáteční adresou segmentu a porovnání s délkou segmentu může byt provedeno souběžně s ukládáním předešlých dat.



Obrázek 24 - Segmentace paměti

V případě velkého množství segmentů není uložení tabulek v registrech proveditelné a je nutno je ukládat do paměti. Ukazatelem do tabulky segmentů je v Segment-table base point registr (STBR).

Vzhledem k proměnlivému počtu segmentů jednotlivých úloh je implementován Segment table length registr (STLR) obsahující počet segmentů aktuální úlohy.

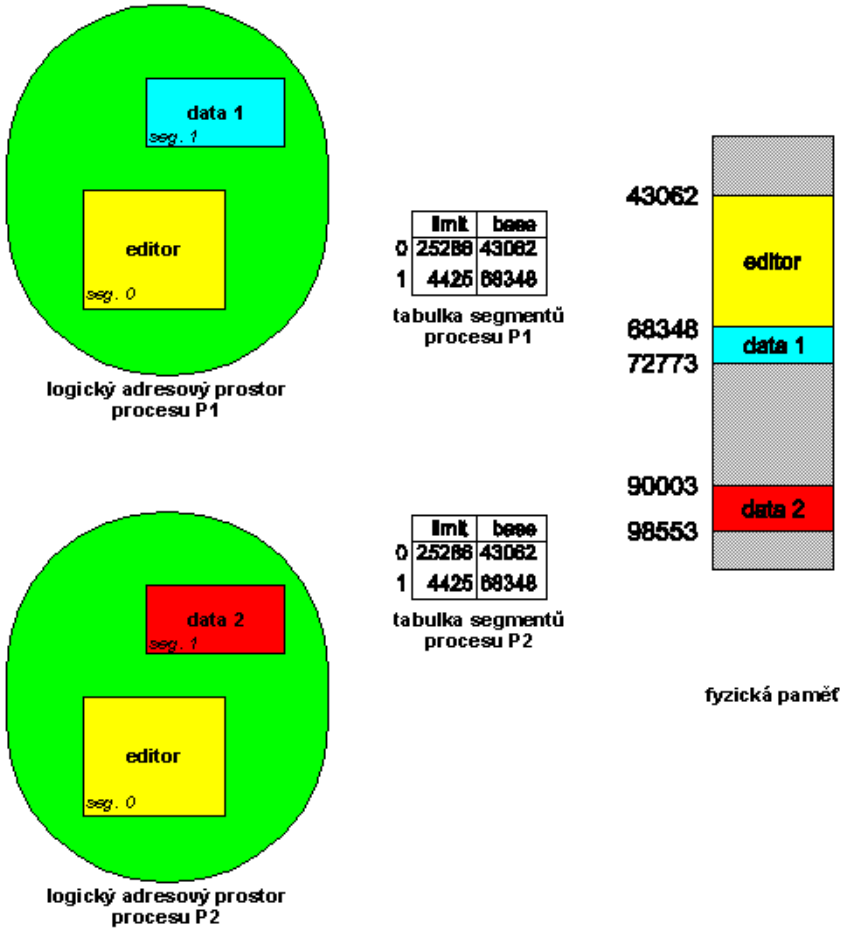
Pro každou logickou adresu (S, O) se nejdříve kontroluje číslo segmentu (S < STRL). Je-li S v pořádku, připočte se k STBR (STBR + S) a výsledkem je ukazatel na správný záznam v tabulce segmentu, se kterým je naloženo tak, jak uvádí Obr. 23.

Při umístění registru v paměti znamená každý přístup k paměti přístupy dva (jeden do tabulky segmentů). Aby se tento dopad zmírnil, využívají se opět rychlé registry jako cache pro umístění těch položek tabulky segmentů, na které je nejčastěji odkazováno. Celkem malý počet registrů dokáže snížit počet přístupů do paměti tak, že je jen o 10 – 15 % vyšší než u nemapované paměti.

8.2 Ochrana a sdílení segmentů

Každý segment je zabezpečen zvlášť. Protože segmenty představují sémanticky definované bloky dat, přistupuje se k jednotlivým bytům segmentu stejným způsobem – segmenty obsahující instrukce, segmenty obsahující data.

V moderních systémech lze instrukční segmenty definovat jako non-self-modifying, takže jsou definovány jako read-only nebo execute-only. Signalizační bity uvedené ochrany jsou v každé položce tabulky segmentů a memory-mapping hardware je kontroluje před každým přístupem do paměti, aby předešel neoprávněným zásahům, jako např. zápis do read-only segmentu, nebo užití execute-only segmentu jako dat.



Obrázek 25 - Sdílení segmentu při segmentaci paměti

Při uložení pole do segmentu memory-managment hardware automatika kontroluje indexy pole, jestli jsou v pořádku a nepřekračují hranici segmentu. Mnoho softwarových chyb je takto odhaleno hardwarem dříve, než stačí napáchat škody.

Další výhodou segmentace je možnost *sdílení* (sharing) kódu nebo dat jednotlivými úlohami.

Uvažujme užití textového editoru v systému sdílení času. Adresový prostor editoru tvořený segmenty může zabírat v paměti hodně místa. Tyto segmenty mohou být sdíleny více uživateli, než aby byl celý adresový prostor editoru v paměti n krát. Pro každého uživatele musí být k dispozici pouze segment pro uložení lokálních proměnných. Tyto segmenty samozřejmě sdíleny nebudou.

Vhodné je také samozřejmě sdílet i části programu, které obsahuji např. často používané rutiny (funkce Sqrt apod.). Musejí být proto v segmentech označených jako sdílitelné a read-only.

Sdílení je možné pouze u pečlivě propracovaných programů, které jsou vytvořeny tak, aby to umožňovaly (reentrantní). Např. podmíněný skok obsahuje cílovou adresu. Cílová adresa se skládá ze segmentu a offsetu. Segment bude číslo kódového segmentu programu. Pokud bude tento segment sdílen, musí tedy mít ve všech procesech stejné číslo apod.

Mnohem jednodušší je sdílet segmenty, které neobsahuji fyzické adresy. Tam potřeba stejného čísla segmentu u všech procesů, které ho sdílí, odpadá. Lépe je tedy adresovat relativně, např. vzhledem k aktuální pozici ukazatele programu apod.

8.3 Fragmentace

Plánovač úloh musí najít a alokovat paměťový prostor pro všechny segmenty uživatelského programu. Tato situace je totožná se stránkováním. Jediný rozdíl je v proměnlivé délce segmentu.

Stejně jako u stránkování stojíme před problémem možné fragmentace paměťového prostoru tj. situace kdy je velmi mnoho velmi malých volných oblastí v paměti, a tak nelze zavést novou úlohu.

Nejjednodušším řešením fragmentace, které je vždy možné, je počkat, až některá úloha uvolní svůj paměťový prostor. Příliš efektivní ovšem toto rosení není.

Další možnou cestou je zhušťování. Protože při segmentaci je paměťový prostor rozdělen logicky a celý algoritmus segmentace je ve své podstatě dynamický, můžeme zhušťování provádět kdykoli se nám zamane. Příkladně můžeme zhušťovat v době, kdy CPU musí čekat na lib. proces. Zhušťování v takovém případě může být spuštěno (nebo nemusí) jako samostatný proces s nízkou prioritou.

Závažnost problému fragmentace je závislá na velikosti segmentu. Pokud bychom systém definovali tak, že každý byte bude uložen ve vlastním segmentu, problém fragmentace by nás minul. Nastaly by však nejspíše problémy jiné a závažnější. V podstatě se dá říci, že pokud jsou segmenty „rozumně“ malé, je i fragmentace únosná.

9. SEGMENTACE SE STRÁNKOVÁNÍM

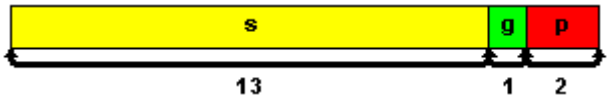
Kombinace segmentace a stránkování přináší vylepšení obou těchto technik. Využívají ji např. procesory Motorola 68000 i Intel 80X86.

9.1 OS/2 32-bit version

OS/2 ve 32 bitové verzi na platformě Intel 80X86 využívá kombinaci segmentace a stránkování pro management paměti. Maximální počet segmentů na proces je 16384 (16k) a každý segment může být velký max. 4 GB. Velikost stránky je 4kB.

Adresový prostor každého procesu je rozdělen na 2 části – 8192 segmentů může být privátních pro každý proces a 8192 segmentů může být sdílených. Informace o první skupině segmentů jsou uloženy v *local descriptor table (LDT)* o druhé potom v *global descriptor table (GDT).* Každá položka těchto tabulek zabírá 8 bytů podrobných informací o každém segmentu včetně délky a počáteční adresy segmentu.

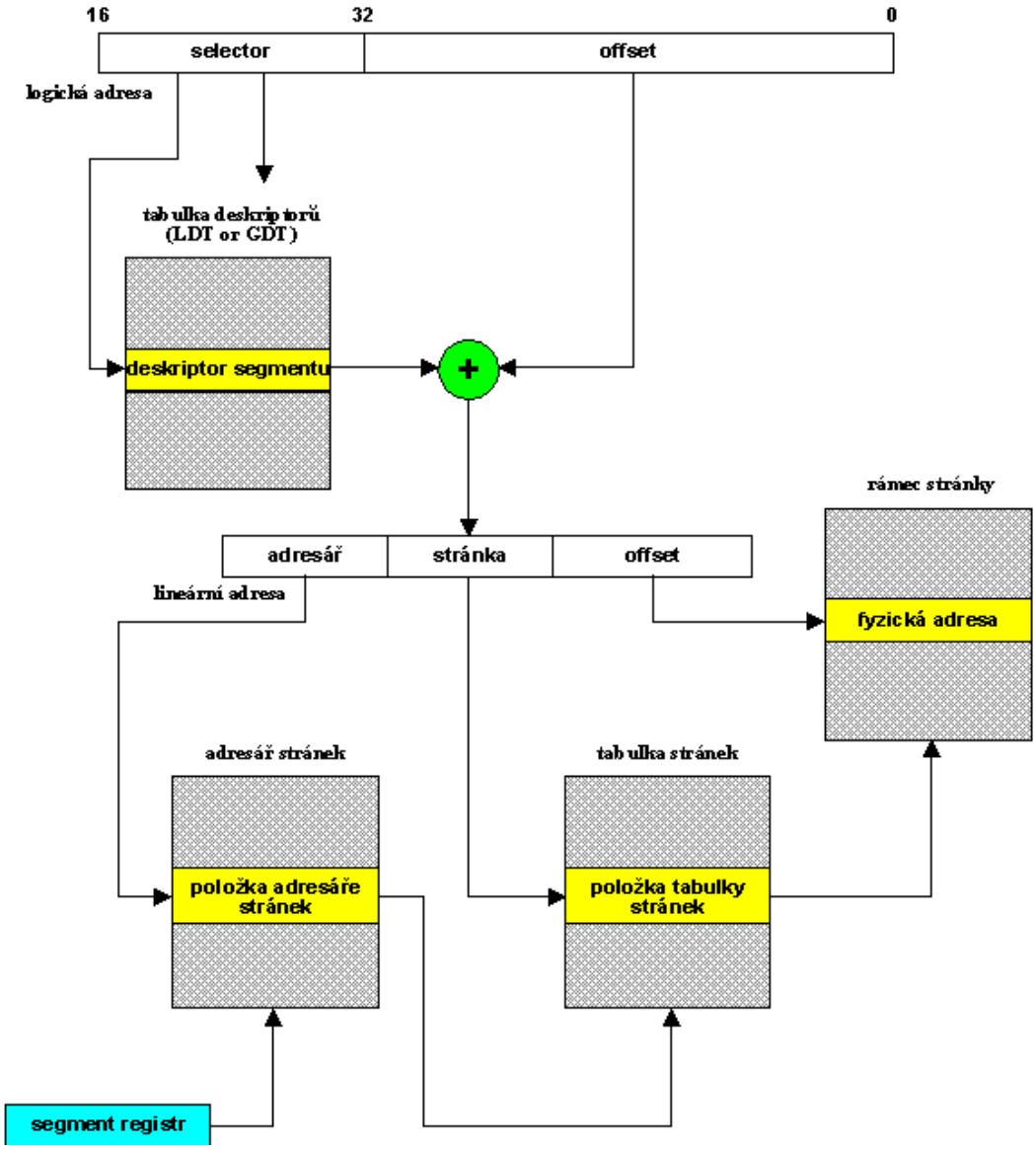
Logická adresa je dvojice (selector, offset), kde selektorem je 16-bitové číslo:



Obrázek 26 - Logická adresa v systému OS/2(s – číslo segmentu; g – indikuje, je-li segment v GDT nebo LDT; p – informace pro ochranu paměti)

Offset je 32bitové číslo udávající relativní adresu slova v rámci daného segmentu.

Výpočetní systém I386 obsahuje 6 segmentových registrů adresujících 6 segmentů, které mohou být okamžitě užity procesorem. Ke každému z registrů je přidělen 8bytový registr pro příslušné záznamy z LDT nebo GDT.

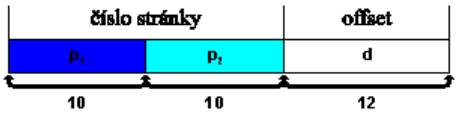


Obrázek 27 - Transformace adres u I386

Fyzická adresa u I386 je 32bitové číslo. Select registr ukazuje na příslušnou položku v LDT nebo GDT. Z ní jsou užity délka a počáteční adresa aktuálního segmentu pro vytvoření *lineární adresy*. Nejdříve je délka segmentu užita pro ověření regulérnosti odkazu na paměťové místo. Pokud je adresa mimo daný segment, je vyvoláno přerušení (memory fault). Je-li vše v pořádku, připočte se počáteční adresa segmentu k offsetu a vznikne *32bitová lineární adresa*. Tato adresa je modifikována na fyzickou.

Jak již bylo uvedeno, každý segment je stránkován a velikost stránky je 4kB. Tabulka stránek musí tedy obsahovat více než 1 mil. položek. Každá položka zabírá 4B, každý proces může požadovat až 4MB adresového prostoru pro svou tabulku stránek. Takto rozsáhlé tabulky je možno implementovat pouze stěží a I386 to řeší dvouúrovňovým schématem stránkování.

Lineární adresa je rozdělena na dvě části – číslo stránky (20 bitů) a offset (12 bitů). Číslo stránky je rozděleno na ukazatel do adresáře stránek (10 bitů) a ukazatel na stránku (10 bitů). Transformaci adresy z logické na fyzickou ukazuje obr. 28



Obrázek 28 - *Lineární adresa v systému OS/2*

Pro zvýšení rozsahu fyzické paměti může I386 swapovat tabulky stránek na disk. V tom případě je jeden nepoužitý bit v položkách ukazatel do adresáře stránek použít pro indikaci toho, zda je daná položka jen na disku nebo i v paměti. Je-li na disku, OS užívá 31b číslo pro adresaci tabulky na disku. Tabulka může být vložena do paměti na žádost.

10. SEGMENTACE NA ŽÁDOST

Stránkování na žádost je všeobecně uznávané jako nejlepší mechanismus pro vytvoření virtuální paměti. Vyžaduje ovšem silnou hardwarovou podporu. Pokud takový hardware není k dispozici, lze vytvořit virtuální paměť pomocí segmentace na žádost.

I286 nepodporuje stránkování paměti, ale je schopen vytvářet segmenty. Systém OS/2 instalovaný na tuto platformu využívá segmentaci na žádost jako prostředek pro vytvoření virtuální paměti.

Systém opět využívá tabulek segmentů pro jednotlivé procesy, kde jsou informace o uložení, ochraně a velikosti segmentu. Jelikož se segment ve fyzické paměti vyskytovat může i nemusí, má každá tabulka navíc indikační bit, který o tom informuje.

Je-li adresován segment, který není ve fyzické paměti, je vyvoláno přerušení (segment fault) stejně jako u stránkování na žádost. Není-li ve fyzické paměti dost místa pro zatažení nového segmentu (i po případně provedeném zhušťování), je některý ze segmentů v paměti odložen do virtuální paměti na disk.

Pro určení segmentu, který má být přesunut do záložní paměti využívá OS/2 další bit z tabulky segmentů – *accessed bit*. Jeho funkce je stejná jako funkce referenčního bitu u stránkování na žádost. Je nastaven do hodnoty 1 v okamžiku, je-li čteno libovolné slovo ze segmentu, nebo pokud je do něj zapisováno. Systém udržuje frontu obsahující položku pro každý segment v paměti. V krátkých časových intervalech systém ukládá na vrchol fronty segmenty s nastaveným *accessed bit*em a potom ho nastaví na 0. Tímto mechanismem si udržuje OS/2 frontu segmentů, kde na vrcholu jsou ty nejpoužívanější. Pořadí ve frontě může měnit i OS a zajistit tak neustálou přítomnost některých segmentů v paměti.

Je-li třeba vybrat segment pro odložení na disk, je vybrán ten na konci fronty. Je-li pro zatažení segmentu volné místo v paměti, je do něj segment zatažen, aktualizují se záznamy v tabulce segmentů a identifikace segmentu je zařazena na vrchol fronty.

I segmentace na žádost má značné hardwarové nároky, takže s málo vyspělým hardwarem nelze virtuální paměť budovat. Segmentace na žádost je pouze rozumným kompromisem nad ještě náročnějším hardwarem potřebným k provozu stránkování na žádost.