Správa pamäte

Portál:	edu.ukf.sk - Vzdelávací portál - Univerzita Konštantína Filozofa, Nitra
Kurz:	Operačné systémy (KI/OS/15)
Kniha:	Správa pamäte
Vytlačil(a):	Zuzana Pavlendová
Dátum:	Streda, 1 december 2021, 17:58

Opis

3 a) Správa pamäte

3.1 Stratégia prideľovania pamäte

3.1.1 Prideľovanie všetkej voľnej pamäte

3.1.2 Prideľovanie pevných blokov pamäte

3.1.2.1 Informácie o blokoch

3.1.2.2 Fragmentácia pamäte

3.1.2.3 Alokačné stratégie

3.1.2.4 Presúvanie blokov

3.1.2.5 Adresovanie

3.1.2.6 Ochrana pamäte

3.1.3 Prideľovanie blokov pamäte premennej veľkosti

3.1.4 Segmentácia pamäte

3.1.4.1 Odstránenie fragmentácie

3.1.5 Stránkovanie pamäti

3.1.6 Stránkovanie na žiadosť (demand paging)

3.1.6.1 Algoritmy nahradzovania stránok

3.1.7 Segmentácia so stránkovaním na žiadosť

Obsah

Úvod

Správca pamäte patrí medzi ďalšie veľmi dôležité, i keď nie najkomplikovanejšie moduly každého operačného systému.

Hlavnou úlohou správcu pamäte je:

- Prideľovať operačnú pamäť jednotlivým procesom, keď si ju vyžiadajú.
- Udržiavať informácie o pamäti, ktorá časť je voľná, a ktorá pridelená (a komu).
- Zaraďovať pamäť, ktorú procesy uvoľnia, opäť do voľnej časti.
- Odoberať pamäť procesom, ak je to potrebné.
- Zaistiť ochranu pamäte (ak to umožňuje technické vybavenie) - žiadny proces by nemal mať prístup k pamäti iného procesu alebo operačného systému, ak mu to 'vlastník' pamäte explicitne nepovolí.

Požiadavky na správu pamäte:

- Možnosť relokácie - programátor nemôže vedieť, z ktorej časti pamäte bude jeho program vykonávaný. Relokácia neumožňuje, aby sa adresy kontrolovali behom kompilácie – odkazy na adresy sa musia kontrolovať pri behu procesu hardvérom.

- Procesu môže byť dynamicky pri výmenách (odoberanie a vracanie prostriedkov procesu) prideľovaná iná oblasť pamäte – swapping.
- Odkazy na pamäť v LAP (logickom adresovom priestore) sa musia dynamicky prekladať na skutočné vo FAP (fyzickom adresovom priestore.
- Nutnosť ochrany – procesy sa nesmú byť schopné bez povolenia odkazovať na pamäťové miesta, pridelené iným procesom.
- Logická organizácia - užívatelia tvoria programy ako moduly s odlišnými vlastnosťami. Moduly s programami – execute, dátové read-only, read/write, niektoré moduly súkromné (private) a iné verejné (public).
- Možnosť zdieľania - viac procesov môže zdieľať spoločnú časť pamäte bez toho, aby sa tým porušovala ochrana pamäte. Zdieľaný prístup k spoločnej dátovej štruktúre – zdieľanie jediného exemplára dátovej štruktúry je lepšie riešenie, než udržovanie konzistencie ich násobných kópií vlastnených jednotlivými procesmi.
- Riadenie pamäte - ako sa organizácia pamäte chová v rôznych prípadoch.

Existujú tri rôzne kategórie riadenia pamäte:

- Načítanie (fetch) - chovanie pri získavaní ďalších častí procesov alebo dát, staršie na žiadosť (on demand), teraz v predstihu (anticipátory).

- Umiestňovanie (placement) - kde v pamäti umiestniť požadovaný blok dát.

- Výmena (replacement) - čo z pamäte odstrániť pre novo prichádzajúce dáta.

Požiadavky, ktoré sa v počítačovom systéme kladú na pamäte sa zatiaľ nedajú ekonomicky splniť jedinou pamäťou. Čím väčšia je kapacita pamäte, tým väčšia je totiž i jej pomerná cena. Podobne rastie cena i so skracovaním vybavovacej doby. Preto rýchle pamäte (s krátkou vybavovacou dobou) majú obvykle malú kapacitou, pomalé pamäte (s dlhšou vybavovacou dobou) majú veľkú kapacitu. Začlenenie jednotlivých úrovní pamäte je naznačené na obrázku.

Z pohľadu uvedenej hierarchie pamäte sa správa pamäte operačného systému zaoberá prevážne riadením hlavnej (operačnej) pamäte.

Požiadavky

V tejto kapitole sa dozviete:

• Aké sú hlavné úlohy správcu pamäte

• Aké sú stratégie prideľovania pamäte?

• Čo je to virtuálna pamäť

• Ako sa chráni pamäť proti neoprávneným prístupom

Po jej preštudovaní by ste mali byť schopní:

• Charakterizovať základné funkcie správcu pamäte.

• Poznať stratégie prideľovania pamäte viacerým procesom.

• Porozumieť princípom vytvárania virtuálnej pamäte.

• Popísať spôsoby ochrany pamäte.

Kľúčové slová tejto kapitoly:

Správca pamäte, virtuálna pamäť, fragmentácia pamäte, ochrana medznými registrami, ochrana pomocou zámkov a kľúčov.

Doba potrebná k štúdiu: 5 hodín

Sprievodca štúdiom:

. V prípade, že ste pochopili predchádzajúcu kapitolu o procesoch bude sa Vám ľahšie študovať táto kapitola. Štúdium tejto kapitoly je tiež pomerne náročné

Na štúdium tejto časti si vyhraďte aspoň 5 hodín. Odporúčame študovať s prestávkami vždy po pochopení jednotlivých podkapitol. Po celkovom preštudovaní a vyriešení všetkých príkladov odporúčame dať si prestávku, hoci 1 deň a potom sa pustiť do vypracovania korešpondenčných úloh.

3.1 Stratégia prideľovania pamäte

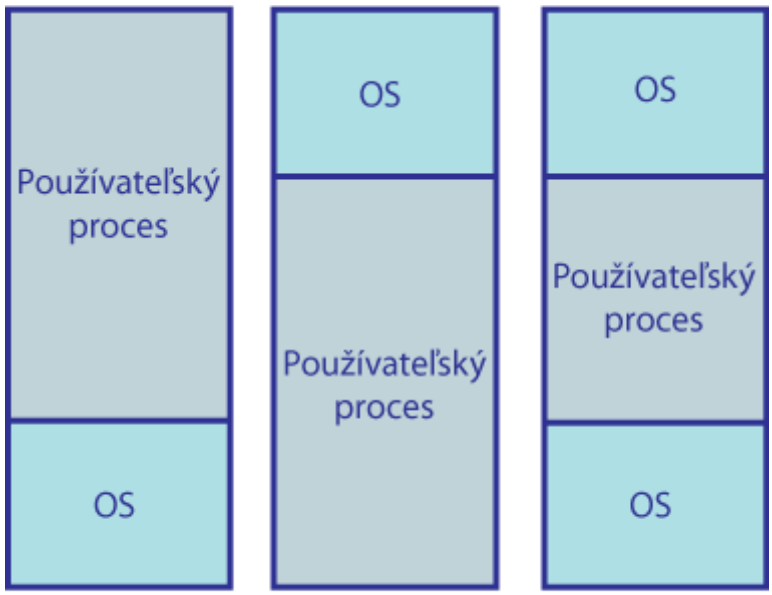
Existujú rôzne stratégie prideľovania pamäte:

- prideľovanie všetkej voľnej pamäte,
- prideľovanie pevných blokov pamäte,
- prideľovanie blokov pamäte premennej veľkosti,
- segmentácia pamäte,
- stránkovanie pamäte,
- stránkovanie na žiadosť (demand paging),
- segmentácia so stránkovaním na žiadosť.

3.1.1 Prideľovanie všetkej voľnej pamäte

Časť pamäte RAM je obsadená operačným systémom (kód, premenné, vyrovnávacie pamäte), zvyšok je k dispozícii pre používateľský program. V každom okamihu je teda v pamäti nanajvýš jeden používateľský program.

V jednoužívateľských operačných systémoch je prideľovanie pamäte naznačené na nasledujúcom obrázku.



Prideľovanie pamäte v jednopoužívateľských OS

Táto stratégia bola používaná na začiatku 60. rokov pre OS osembitové mikropočítače (CP/M, ISIS-2 od Intelu).

Operačná pamäť je v operačnom systéme CP/M rozdelená nasledovne:

· Prvých 100 byte je určených operačnému systému (údaje o bežiacom programe, parametre).

· V ďalšej časti je pamäť pre program (a voľné miesto).

· Na konci pamäte premennej systému a buffery , BDOS a BIOS (v ROM).

· Adresovanie pre programy začína vždy na adrese väčšej než 100. Tu sú ukladané preložené programy pripravené pre spustenie.

· Veľkosť operačnej pamäte je maximálne 64 KB a preto operačný systém a programy musia byť prispôsobené veľkosti pamäte.

Operačná pamäť v operačnom systéme ISIS-2 je rozložená nasledovne:

· Na začiatku pamäte je operačný systém, potom premenné a buffery.

· Potom nasleduje voľné miesto pre program.

Podľa veľkosti dostupnej pamäte, veľkosti jadra OS a počtu bufferov je nutné:

· v CP/M preadresovávať operačný systém,

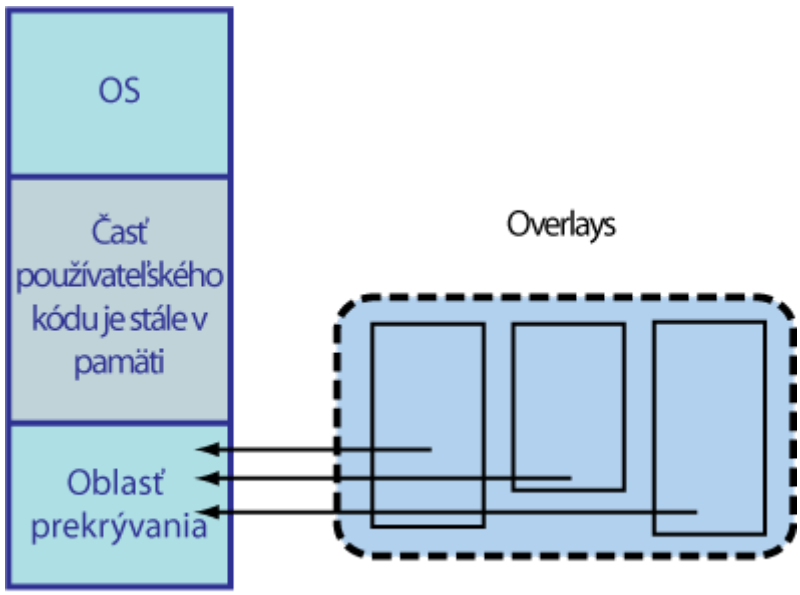
· v ISIS-2 preadresovávať používateľské programy.

Preadresovanie vykonáva program nazývaný Locator. Locator podľa tabuľky adries v relatívnom programe zmení (podľa umiestnenia) príslušnú adresu na absolútnu .

Väčšina procesorov, na ktorých sa vykonávajú systémy s touto stratégiou prideľovanie pamäte, neumožňuje žiadnu ochranu pamäte. Inak iba ochrana OS pred prepísaním používateľským programom (pomocou medzného registra; zmena iba v privilegovanom stave procesora). V používateľskom stave nie je možné zapisovať na adresy väčšie alebo rovné obsahu medzného registra ani meniť obsah medzného registra.

V obmedzenej miere je možné i pri tejto stratégii prideľovania pamäte používať multitasking. Pri prepnutí kontextu, je nutné nahrať celý obsah používateľskej oblasti pamäte na disk a zaviesť z disku obsah adresného priestoru druhého procesu.

V niektorých prípadoch je možné používateľský program rozdeliť na časť trvalo umiestnenú v operačnej pamäti a časti, ktoré sú na sebe nezávislé a preto môžu byť prekrývané. Tieto časti sú potom umiestnené na disku a zavádza sa čo? do prekrývanej oblasti pamäte v prípade ich vykonávania, ako je naznačené na nasledujúcom obrázku.

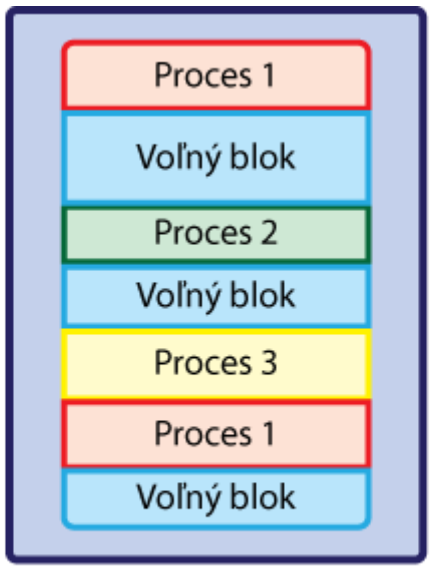


Prideľovanie pamäte v jednoužívateľských OS

3.1.2 Prideľovanie pevných blokov pamäte

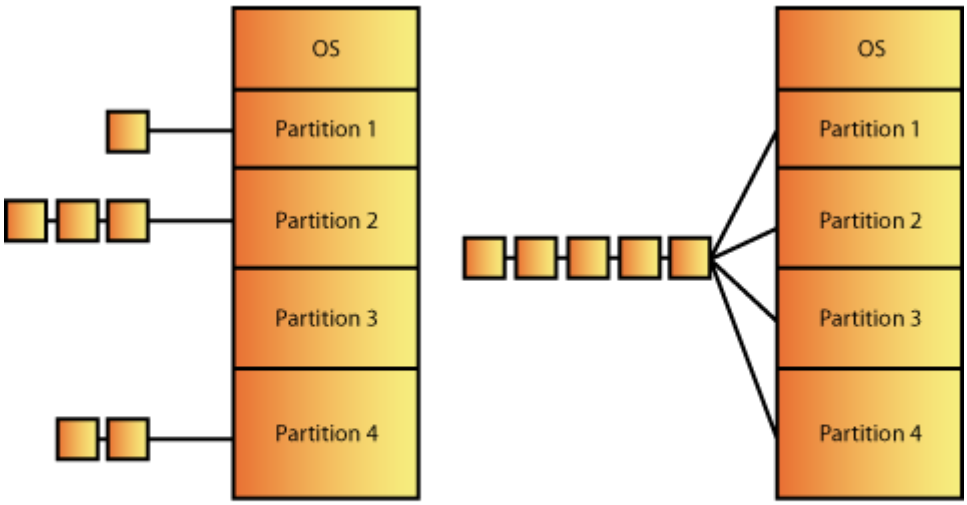
Väčšina jednoduchých operačných systémov využíva práve túto stratégiu - ako príklady môžeme menovať MS DOS. Jej základný princíp je veľmi jednoduchý: každý proces musí vedieť, koľko operačnej pamäte bude potrebovať a musí si túto pamäť od operačného systému explicitne vyžiadať. Operačný systém, správca pamäte, takúto požiadavku alebo splní pridelením bloku požadovanej veľkosti, alebo zamietne. V tom prípade je úlohou procesu vzniknutý problém vyriešiť (napr. predčasným ukončením práce).

Na nasledujúcom obrázku vidíme príklad operačnej pamäte rozdelenej na niekoľko blokov. Operačnú pamäť používajú tri procesy: proces 1 má pridelené dva bloky pamäte, zostávajúce dva bloky majú len po jednom bloku.



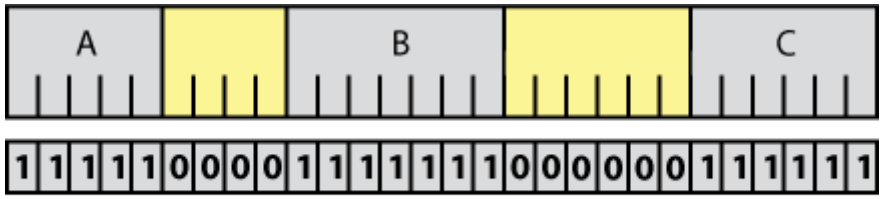
Operačná pamäť rozdelená na niekoľko blokov

Pamäť je rozdelená na bloky pevnej veľkosti a procesy sú vo frontách podľa požadovanej veľkosti. Veľkou nevýhodou je nutnosť poznať maximálnu potrebnú pamäť pred spustením a nevyužitím častí oddielov pamäte.



Operačná pamäť rozdelená do niekoľkých blokov

Procesy získavajú pamäť podľa potreby (pokiaľ je dostupná). Nevýhodou je vytvorenie nepoužiteľných dier medzi použitými blokmi pamäte. Túto nevýhodu môžeme odstrániť riadením pamäte pomocou bitovej mapy. Pamäť rozdelená na bloky rovnakej dĺžky (napr. 16 bajtov) ako je naznačené na nasledujúcom obrázku.



Riadenie pamäti pomocou bitovej mapy

Ďalším riešením je riadenie pamäti pomocou spojového zoznamu. Operačný systém má spojový zoznam všetkých blokov (voľných i použitých). Inou variantov sú dva spojové zoznamy, jeden pre voľné bloky, druhý pre bloky použité. Operačný systém potom musí riešiť dva problémy:

· rozhodnúť, ktorý z voľných blokov pridelí pri novej požiadavke procesu na pamäť (stratégia prideľovania), pri uvoľnení bloku zlúčiť nový voľný blok s okolitými voľnými blokmi.

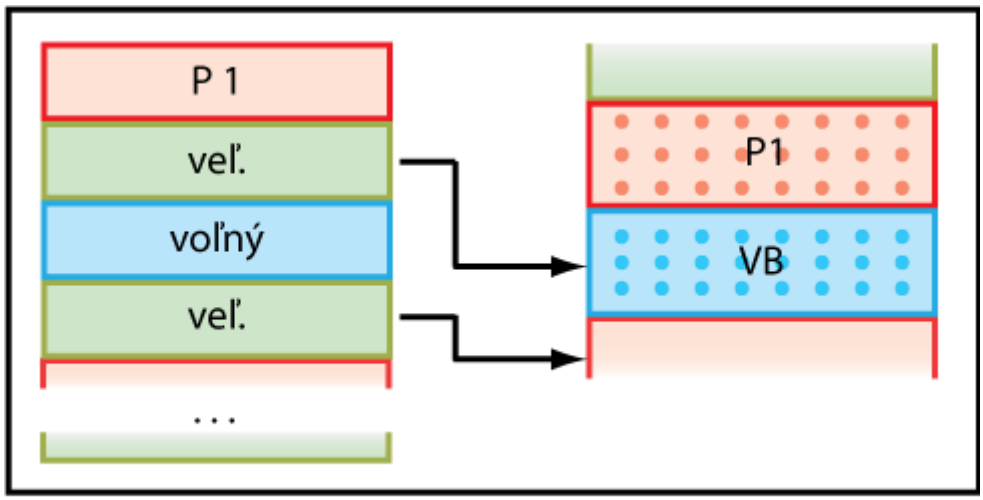
3.1.2.1 Informácie o blokoch

Správca pamäti musí o každom bloku vedieť najmenej dve veci:

· jeho dĺžku,

· jeho vlastníka.

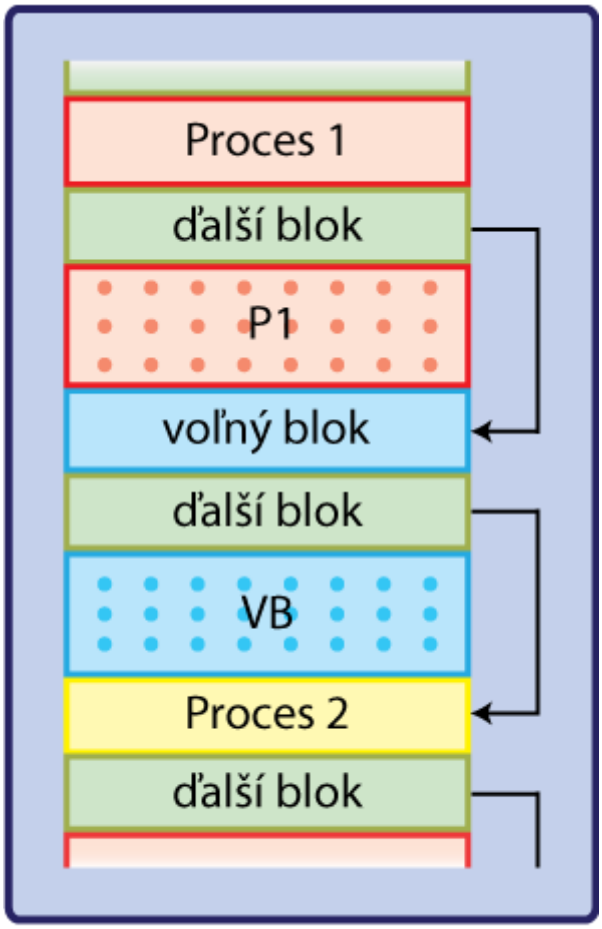
Ak sú informácie o vlastníkovi pamäťového bloku prázdne, znamená to, že blok je voľný a môže byť pridelený. Adresu ďalšieho bloku nemusíme explicitne uvádzať, pretože bloky na sebe v operačnej pamäti naväzujú; adresu nasledujúceho bloku tak ľahko spočítame na základe dĺžky a adresy aktuálneho bloku. Správca pamäte potom potrebuje len znalosť adresy prvého bloku, potom môže ľahko so spojovým zoznamom bloku pracovať. Správca pamäte môže samozrejme tieto údaje udržiavať vo vnútri vlastných dátových štruktúr (ilustráciu tohto spôsobu vidíme na nasledujúcom obrázku).



Údaje o blokoch, prvý spôsob

Toto riešenie však má jednu podstatnú nevýhodu. Musíme pre údaje vyhradiť dostatok pamäte. Veľkosť potrebnej pamäte je úmerná počtu pridelených blokov pamäte. Ich počet však pochopiteľne nie je možné určiť dopredu. S istotou samozrejme vieme, že blokov pamäte bude menej, než je v pamäti k dispozícii bytov; toľko priestoru pochopiteľne nie je možné pre systémové tabuľky vyhradiť. Ak však vyhradíme menej miesta riskujeme, že vo výnimočnom prípade, kedy budú procesy požadovať veľké množstvo malých blokov, vyhradená pamäť nepostačí.

Obvyklým a pomerne efektívnym riešením tohto problému je, prideliť na žiadosť procesu vždy blok o niekoľko bytov väčší a do tohto voľného miesta uložiť informácie o bloku samotnom, i o adrese nasledujúceho bloku. V operačnej pamäti tak vznikne spojový zoznam voľných i alokovaných blokov; správca pamäte potom pri každej požiadavke tento zoznam prechádza a spracováva jeho položky - jednotlivé bloky. Malý úsek takého spojového zoznamu vidíme na nasledujúcom obrázku.



Spojový zoznam

Vybodkované oblasti znázorňujú vlastné pamäťové bloky (tj. tú časť blokov, ktorú dostanú procesy pridelenú ako pamäť), 'P1' označuje blok pridelený procesu číslo 1. 'VB' označuje voľný blok.

Obrázok ilustruje i to, že tieto pomocné údaje bývajú uložené pred vlastným blokom, a nikdy nie za ním. Proces, ktorý požadoval pridelenie pamäte, teda dostane ukazovateľ na začiatok vybodkovaného priestoru a vôbec „nevie“, že pamäťový blok vlastne zaberá i niekoľko bytov pred týmto ukazovateľom.

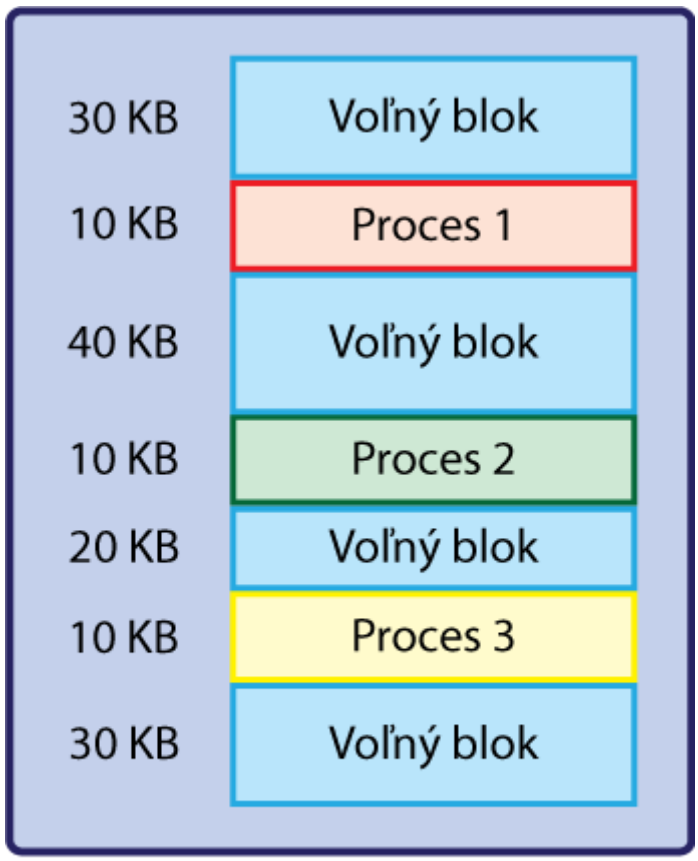
Toto riešenie má samozrejme tiež svoju nevýhodu. Systémové informácie o blokoch pamäte sú „zamiešané“ medzi pamäťové bloky pridelené jednotlivým procesom. Ak technické vybavenie počítača, na ktorom operačný systém vykonávame, neumožňuje zabezpečenie ochrany pamäte, je pomerne veľké riziko, že niektorý chybný program systémovej informácie poškodí . V takom prípade prakticky nie je možné pokračovať v činnosti operačného systému, pretože poškodením spojového zoznamu blokov pamäti je vyradený vlastne celý správca pamäte. Uvedomme si, že bez informácií z hlavičiek blokov nie je možné ani bloky postupne prejsť; tým menej potom sa dá alokovať ďalšia pamäť alebo uvoľniť niektorý z pridelených blokov.

3.1.2.2 Fragmentácia pamäte

Systém prideľovania pamäte po blokoch však má ešte jednu principiálnu nevýhodu, ktorá sa prejaví i v prípade, že žiadny z používaných programov neobsahuje chybu. Dochádza pri ňom totiž k tzv. fragmentácii pamäte.

Už sme sa zmienili o tom, že pri behu operačného systému s prideľovaním pamäte po blokoch, sa voľný úsek pamäte postupne rozpadá na viac oddelených blokov. Správca pamäte samozrejme dokáže spojiť dva voľné bloky, ktoré na seba nadväzujú, do jediného; to však nestačí. Často dochádza k situácii, kedy sú dva voľné bloky oddelené blokom, prideleným niektorému procesu.

Správca pamäte potom pochopiteľne nie je schopný splniť požiadavku na pridelenie väčšieho množstva pamäte, než je veľkosť najväčšieho z voľných blokov, bez ohľadu na to, že celková veľkosť voľnej pamäte (tj. súčet všetkých voľných blokov) môže byť i niekoľkonásobne väčší. Tento problém ilustruje nasledujúci obrázok, na ktorom vidíme pridelené i voľné bloky v operačnej pamäti veľkosti 150KB.



Fragmentácia pamäte

Na prvý pohľad je zrejmé, že dokiaľ niektorý z troch procesov, ktoré majú pridelenú pamäť, svoj blok pamäte neuvoľní, nemôže správca pamäte nikomu prideliť viac než 40KB. Rozsah voľnej pamäte je pritom ďaleko väčší - celých 120KB.

Fragmentáciu môžeme do istej miery obmedziť voľbou vhodnej alokačnej stratégie. Nemôžeme sa jej však nikdy zbaviť úplne, pokiaľ zachováme jednoduchý systém prideľovania blokov pamäte.

3.1.2.3 Alokačné stratégie

Ak sa zamyslíme nad problémom fragmentácie pamäte, napadne nás asi veľmi jednoduchá stratégia, ktorá by dokázala fragmentáciu úplne zamedziť. Stačilo by uvoľňovať bloky v opačnom poradí, než v ktorom boli prideľované procesom, tj. ako prvý vždy uvoľniť ten blok, ktorý bol alokovaný ako posledný. Na prvý pohľad je samozrejme vidieť že táto stratégia nie je v praxi použiteľná. Predstavme si, že posledný alokovaný blok bude potrebný ešte dlho, zatiaľ čo všetky predchádzajúce, by už dávno mohli byť voľné. Je však vhodné si uvedomiť, že zásadný nedostatok tejto stratégie spočíva v tom, že stratégia predpisovala jednotlivým procesom, ako sa smie a nesmie chovať, naviac v závislosti jeden na druhom.

Také závislosti sú však mimoriadne nešťastné. Procesy sa totiž v operačnom systéme objavujú celkom náhodne a ich požiadavky sú tiež náhodné. Ak vnesieme preto medzi rôzne procesy akúkoľvek vzájomnú náväznosť (a v ďalších kapitolách uvidíme, že niekedy sa tomu bohužiaľ nedá vyhnúť), procesy budú musieť nutne na seba navzájom čakať a priechodnosť operačného systému sa drastickým spôsobom zníži. A to si samozrejme neprajeme. Alokačné stratégie preto musia byť vnútornou vecou správcu pamäte.

Je možné vypracovať množstvo alokačných stratégií. V priebehu rokov sa však ako životaschopné ukázali iba dve: tzv. first fit (výber prvého dostatočne veľkého bloku), a tzv. best fit (výber bloku, jeho veľkosť najlepšie zodpovedá požadovanej veľkosti). Stratégia first fit je celkom priamočiara: správca pamäte prechádza postupne voľné bloky a porovnáva ich veľkosť s veľkosťou požadovanej pamäti. Akonáhle narazí na blok, ktorý je dostatočne veľký, uspokojí jeho požiadavky. Zvyšok bloku samozrejme zostane v zozname ako menší voľný blok. Výhodou tejto stratégie je jednoduchosť, rýchlosť a že nijako neobmedzuje fragmentáciu.

Pri stratégii best fit tomu správca pamäte prejde všetky voľné bloky, a z tých, ktoré sú dostatočne veľké, vyhľadá najmenší. Z voľných blokov potom už štandardným spôsobom uspokojí požiadavky. Účelom stratégie best fit je zachovať veľké voľné bloky čo najdlhšie 'nerozdrobené' a deliť predovšetkým tie menšie. Vďaka tomu sa fragmentécia skutočne trochu zníži. Pre úplnosť dodajme, že v literatúre sa často môžeme stretnúť i s pojmom 'stratégia last fit'. Pri tejto stratégii sa prechádzajú všetky voľné pamäťové bloky podobne ako pri stratégii best fit, ale požiadavky sa uspokoja z posledného bloku, ktorý je dostatočne veľký. Z hľadiska fragmentácie pamäte je táto stratégia dokonale rovnocenná stratégii first fit, je však o niečo málo zložitejšia a o dosť pomalšia. Používa sa len preto, že pamäťové bloky, ktoré sú jej pomocou alokované, sú pokiaľ je to možné na čo najvyšších adresách, čo môže byť za určitých podmienok výhodné. Ak potrebujeme napr. v operačnom systéme bez podpory multitaskingu blok pre zásobník (ktorý obvykle rastie smerom dole), môžeme použiť stratégiu last fit. Potom je veľká pravdepodobnosť, že medzi zásobníkom a ostatnými dátami zostane nejaký úsek voľnej pamäte a prípadné pretečenie zásobníka ešte nemusí viesť k zrúteniu programu.

3.1.2.4 Presúvanie blokov

Pozrime sa znova na nasledujúci obrázok, ktorý ukazuje fragmentovanú operačnú pamäť. Na prvý pohľad je zrejmé, že by nám pomohlo, keby sme mohli obsadené bloky premiestniť tak, že by ležali tesne vedľa seba. Voľná pamäť by sa potom mohla spojiť do jediného bloku zaberajúceho celých 120KB. Obsah pamäte po takom prerovnaní blokov, hovoríme mu často striasanie, vidíme na nasledujúcom obrázku. Správca pamäte môže bez problémov prideliť ľubovoľne veľký blok pamäte, až do veľkosti celej voľnej pamäte, tj. do 120KB.

Problémov s fragmentáciou sme sa teda zbavili; zato však musíme zaplatiť pomerne vysokú cenu. Správca pamäte musí pamäťové bloky presúvať, čo je samozrejme časovo náročné (zvlášť ak sa jedná o väčšie úseky pamäti) a jednotlivé procesy musia počítať s tým, že pridelená pamäť nemusí zostať stále na rovnakej adrese. Ako uvidíme, jedná sa o pomerne zložitý problém. Presúvanie blokov samo o sebe samozrejme zaberie nejaký čas, to však nie je zasa príliš tragické. Na rade počítačov má správca pamäte k dispozícii špecializovaný mikroprocesor (najčastejšie ho nazývame blitter), ktorý zaisťuje veľmi rýchle presuny dát v pamäti, takže časová strata nie je tak veľká. Ani na systémoch, kde nie je blitter k dispozícii, nehrozia vážnejšie problémy, pretože striasanie pamäte je potrebné vykonávať iba vo chvíli, keď niektorý proces požaduje príliš veľký blok a k tomu nedochádza tak často.

Ďaleko horším problémom je automatické presúvanie blokov z hľadiska procesov. V ideálnom prípade by samozrejme malo byť presúvanie blokov pre procesy celkom transparentné. To sa však dá zaistiť iba vtedy, ak technické vybavenie počítača umožňuje tzv. preklad adries. Ak však máme k dispozícii preklad adries, môžeme implementovať virtuálnu pamäť, ktorá nielen zamedzuje fragmentácii, ale prináša i rad ďalších výhod.

Procesy, ktoré pracujú pod operačným systémom, využíva jeho správca pamäte na automatické presúvanie blokov, preto obvykle musí spĺňať určité konvencie, ktoré zaistia, že presun bloku pamäte procesu „neuškodí“. Máme k dispozícii v zásade tri možnosti riešenia tohto problému:

· Procesy, pre prístup do pamäte musia dodržať určité adresovacie konvencie, ktoré zaistia premiestniteľnosť bloku (typicky sa bude jednať o povinné bázovanie vhodným registrom, konkrétne riešenie samozrejme závisí na adresovacích módoch použitého procesoru).

· Procesy musia dodržiavať určité konvencie na vyššej úrovni, na úrovni vlastného algoritmu. Proces môže byť napr. povinný pred prístupom do pamäti zistiť dotazom u správcu systému momentálnu adresu bloku a potom po celú dobu práce s týmto blokom pamäť ' zamknúť' - tj. zakázať jeho premiestnenie.

· Operačný systém môže procesu zaslať správu vo chvíli, kedy blok pamäti premiestňuje. Proces potom na základe tejto správy prepočíta všetky svoje ukazovatele, ktoré do bloku mieria, na správne hodnoty podľa novej adresy bloku.

Každá z možností má svoje výhody i nevýhody.

Prvá metóda je nepochybne najlepšia, závisí však do značnej miery od technického vybavenia, u niektorých procesorov môže veľmi podstatným spôsobom redukovať možnosti adresovania v pamäťových blokoch. Musí jej byť tiež prispôsobený použitý prekladač pri tvorbe procesov vo vyššom jazyku.

Druhým spôsobom kladie najmenšie nároky na správcu pamäte, o to viac práce však majú programátori, ktorí vytvárajú procesy.

Tretia metóda nie je príliš vhodná pre bežné aplikácie, dobre však poslúži ako doplnok prvej metódy pre programy, ktoré z nejakých dôvodov musí nutne spracovať s absolútnymi adresami (typicky sa jedná o ovládače periférnych zariadení). V prípade jej využitia je však nutné vhodným spôsobom zvoliť spôsob, ktorým správca pamäte procesu oznámi presunutie bloku pamäte - klasický aparát správ, s ktorým sa zoznámime neskôr, nie je vhodný, pretože správa by mohla prísť príliš neskoro (teda po pokuse o prácu s pamäťou na pôvodnom mieste).

Dodajme, že správca pamäte môže voliť jednu z dvoch variant:

- bloky môžu byť vyhradzované ľubovoľným spôsobom tak, ako sme predpokladali až dotiaľ, dokiaľ?
- alokovať bloky jedného procesu 'vedľa seba', bez prerušenia 'cudzím' blokom. V takom prípade budeme celú skupinu blokov jedného procesu nazývať sekcie.

Na záver si predveďme zhrnutie stratégie ukladanie do pevných blokov:

- First fit - predchádzajú sa bloky pamäte a pridelí sa prvý blok, dĺžka je väčšia alebo rovná požadovanej pamäti.
- Last fit - vyberie sa posledný vyhovujúci čo?
- Worst fit - vyberie sa najväčší vyhovujúci u prideľovaných blokov premennej veľkosti (používa sa u stratégie prideľovania blokov pamäti premennej veľkosti pre obmedzenie fragmentácie - viď ďalej)
- Best fit - vyberie sa najmenší vyhovujúci blok (pre prideľovaní blokov pevnej veľkosti sa spravidla používa táto stratégia)

Aké sú výhody prideľovania pevných blokov pamäti:

- vo vnútornej pamäti môže byť niekoľko procesov súčasne,
- jednoduchosť.

Nevýhody prideľovania pevných blokov pamäte:

- zlé využitie pamäte (fragmentácia),
- je nutné dopredu poznať pamäťové nároky,

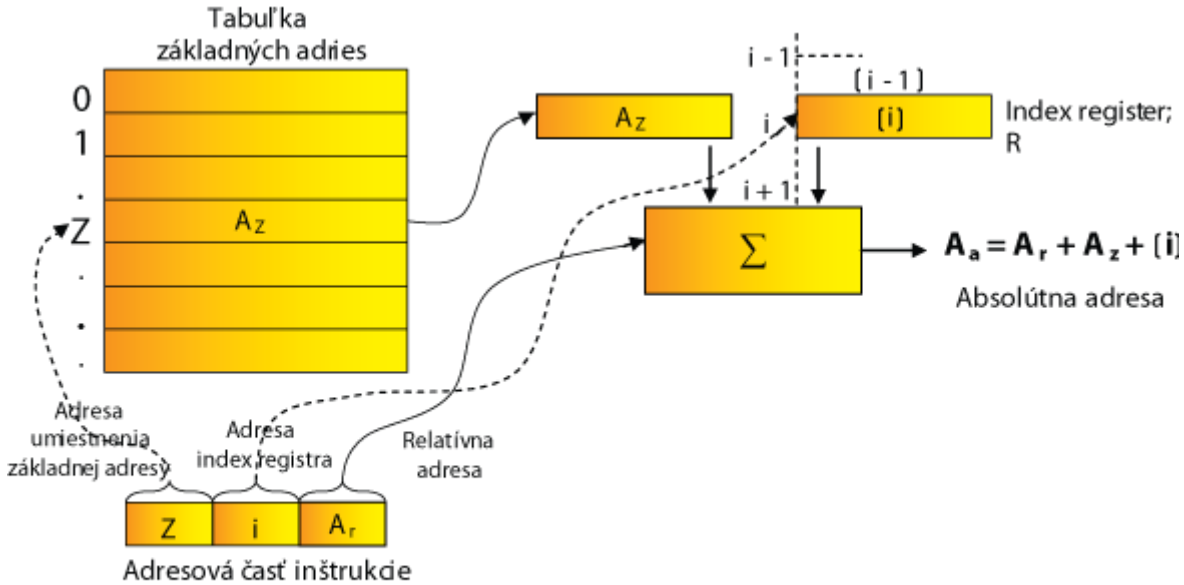
- pamäťové nároky sú väčšie než veľkosť najväčšieho bloku má smolu (nedá sa ho odštartovať - jednému programu nie je možné prideliť dva susedné bloky).

3.1.2.5 Adresovanie

Nie je možné dopredu stanoviť na akej adrese bude program uložený, to znamená, že program musí byť relokatibilný. Máme tri možnosti:

- Relokačná tabuľka v programe, tj. tabuľka s informáciami, kde sú v programe adresné konštanty. Pri zavádzaní programu sa všetky adresné konštanty upravia podľa skutočnej adresy, na ktorej je program.
- Program, v ktorom sú iba relatívne adresy. U skokov to sú autorelatívne adresy (skočiť o + alebo - koľko bytov). U premenných sa používa bázovanie, tzn. že určitý register sa naplní skutočnou adresou napr. na začiatku programu a pre odkazy na premenné sa používa miesto pevnej adresy hodnota bázového registra + posunutie (offset). Je veľmi žiadúce, aby procesor podporoval bázovanie a musí mať relatívne skoky.
- Dynamické prideľovanie pamäte, s využitím bázovej adresy.

Na nasledujúcom obrázku je naznačené určenie výslednej absolútnej adresy na základe súčtu relatívnej adresy, obsahu index registra a bázovej adresy, určenej v tabuľke základných adries.



Určenie výslednej adresy

3.1.2.6 Ochrana pamäte

Najčastejšie sa používa jedna z týchto metód:

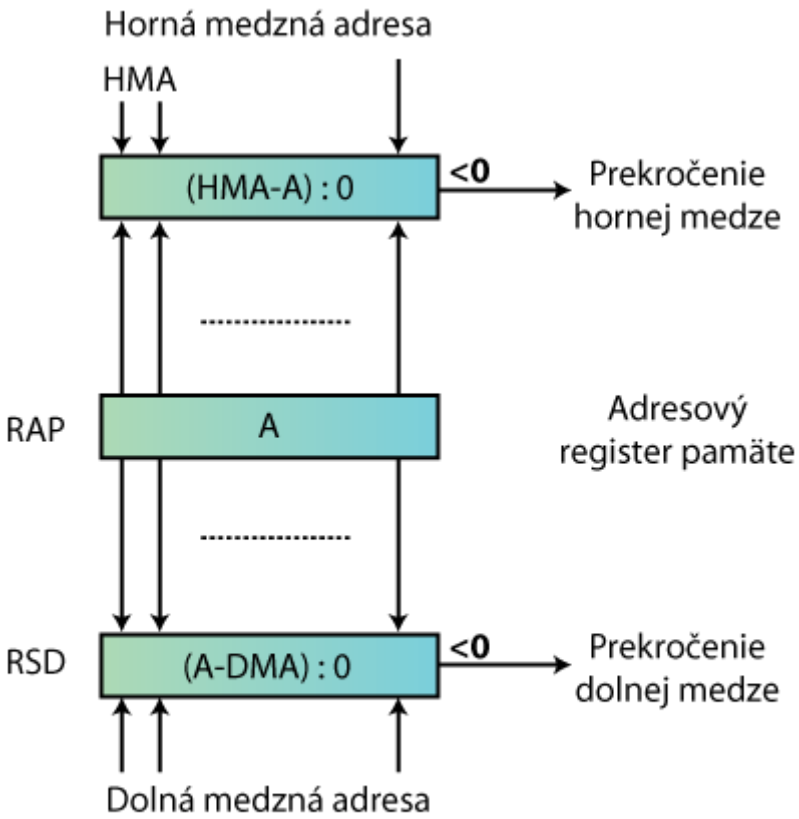
· medzný register,

· mechanizmus zámky a kľúča.

Pre použitie každej z týchto metód je nutná podpora hardvérom, pričom sa používa tá metóda, ktorú podporuje daný procesor.

3.1.2.6.1 Ochrana pamäte pomocou medzných registrov

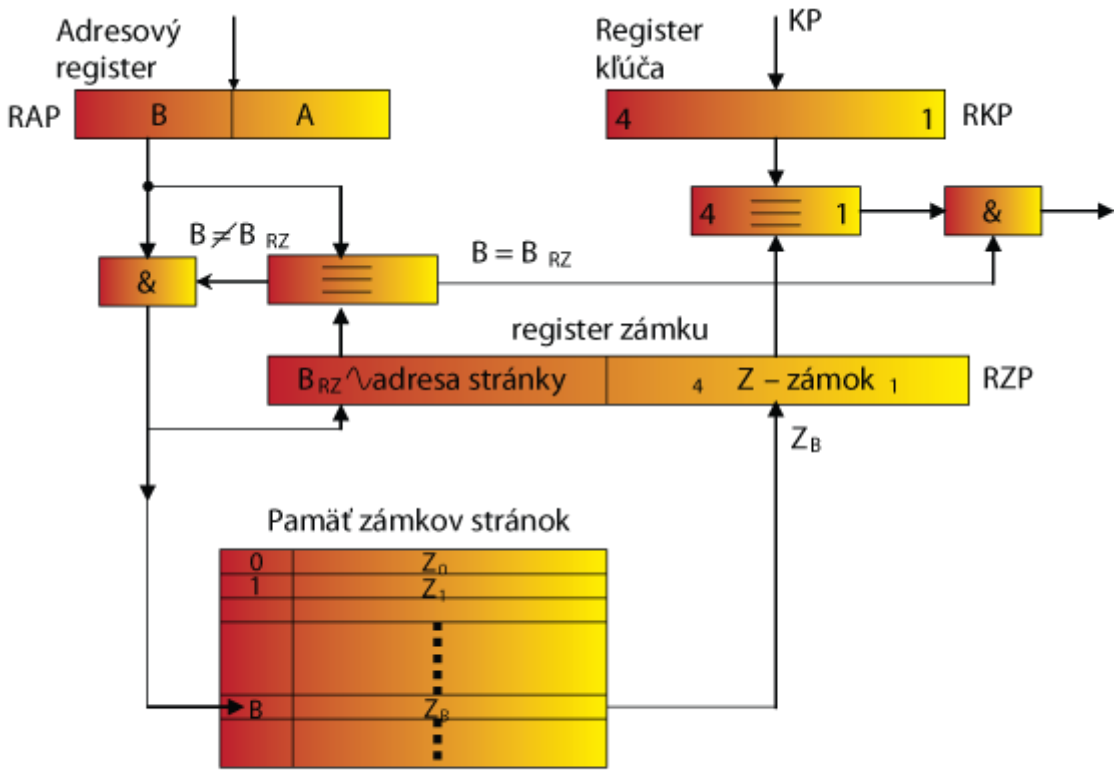
Dva medzné registre udávajú najnižšie a najvyššie dostupnú adresu. Nastavuje ich operačný systém, keď odovzdáva riadenie procesu. Odkaz na pamäť mimo rozsahu spôsobí vnútorné prerušenie (porušenie ochrany pamäti). Nastavenie medzných registrov musí byť privilegovaná inštrukcia, inak môže program napísaný so zlým úmyslom čítať alebo meniť pamäťové oblasti iných procesov.



Ochrana pamäte pomocou medzných registrov

3.1.2.6.2 Ochrana pamäte pomocou mechanizmu zámkov a kľúčov

Pamäť je rozdelená na stránky pevnej veľkosti (napr. 4 KB). Každej stránke pamäte je priradený zámok (= celé číslo). Procesor má špeciálny register, ktorý slúži ako kľúč. Proces môže používať iba tie stránky pamäte, ktoré majú zámok nastavený na rovnakú hodnotu ako je kľúč. Operačný systém môže používať univerzálny kľúč číslo 0, ktorý umožňuje prístup k ľubovoľnej stránke pamäte.



Ochrana pamäte pomocou zámkov a kľúčov

3.1.3 Prideľovanie blokov pamäte premennej veľkosti

Voľná pamäť nie je pevne rozdelená, ale pri štarte programu sa pridelí pamäť podľa nárokov programu (resp. pridelí sa celý voľný blok a program vráti, čo nepotrebuje). Nájdeme ju u MS-DOS, OS-MVT (Multitasking with Variable nuber of Tasks).

Operačná pamäť na počítačoch PC kompatibilných je rozčlenená nasledovne:

· na začiatku pamäte je umiestnený operačný systém, buffery, premenné,

· až do 640kB voľno (do tohto priestoru sa zavádzajú programy),

· VRAM (128kB) Video RAM,

· VROM (32kB) Video ROM (u adaptérov EGA, MCGA, VGA, SuperVGA, inak voľné),

· voľné miesto (toto miesto môžu využívať prídavné adaptéry pre svoje pamäte RAM alebo ROM),

· ROM BIOS (64 kB).

Výhodou, pri prideľovaní blokov premennej veľkosti oproti pevným blokom, je lepšie využitie pamäte.

Hlavné nevýhody môžeme charakterizovať nasledovne:

· Súčet pamäťových nárokov, ktoré sú súčasne v pamäti musia byť menšie alebo rovné veľkosti pamäti.

· Fragmentácie pamäte alebo procesy vyžadujú súvislé úseky pamäti. Môže sa stať, že nie je možné spustiť ďalší proces, pretože má väčšie nároky na pamäť, než je veľkosť najväčšieho voľného bloku i napriek tomu, že súčet veľkostí voľných blokov je väčší než pamäťové nároky procesu

Pre odstránenie fragmentácie sa preto používajú ďalšie stratégie:

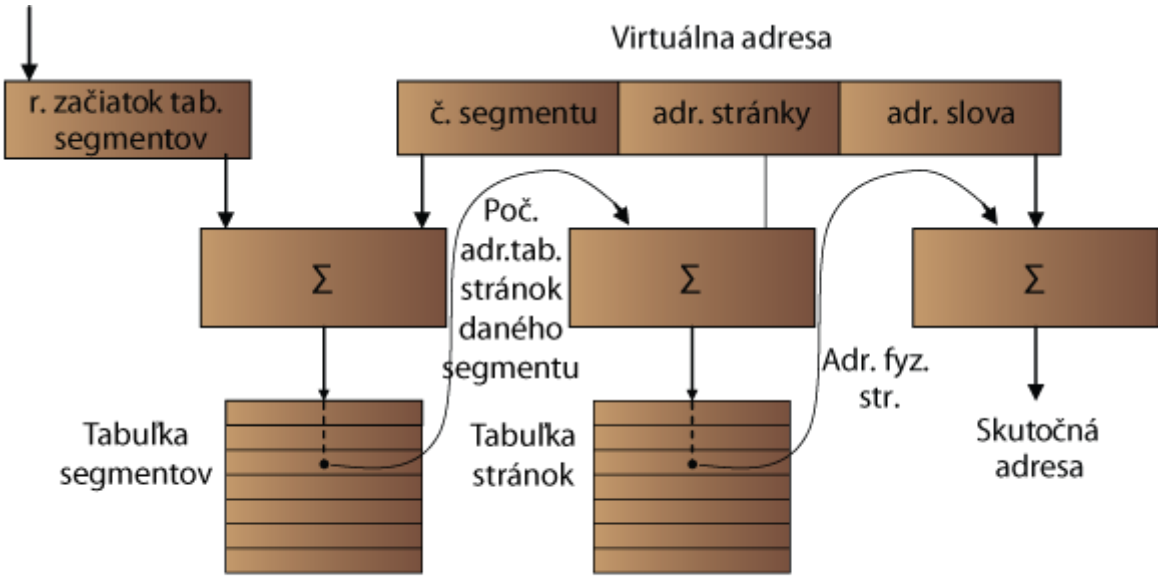
· Striasanie blokov – tu však môžu nastať problémy s adresnými konštantami, čo je možné odstrániť použitím segmentácie.

· Stránkovanie – kedy sa programu javí adresný priestor ako súvislý, hoci v skutočnosti súvislý nie je.

Ochrana pamäte sa prevádza obdobne ako u prideľovaní pevných blokov pamäte, tj. pomocou medzných registrov alebo mechanizmom zámkov a kľúčov.

3.1.4 Segmentácia pamäte

Fyzická (skutočná) adresa v pamäti sa získava prirátaním obsahu registra segmentu k logickej adrese (= adresa použitá v programe). Obsah registra segmentu nastavuje operačný systém a pre používateľský program je neprístupný. Vďaka tomu adresný priestor každého procesu začína na adrese 0 a odpadajú problémy s relokáciou programu. Väčšina systémov, ktoré používajú segmentáciu pamäte, dovoľuje procesom použiť viac segmentov.



Segmentácia pamäte

Rozdelenie na segmenty spravidla zodpovedá štruktúre pamäťového priestoru procesu:

· kód programu - nemenný (obsah ani dĺžka),

· konštanty - nemenné,

· inicializované statické premenné - dĺžka sa nemení, obsah je nastavený pri štarte procesu (načíta sa zo súboru, ktorý obsahuje program), pri behu procesu sa ich obsah môže meniť, dĺžka sa nemení,

· neinicializované statické premenné, dĺžka sa nemení, obsah áno,

· zásobník (návratové adresy z procedúr, lokálne premenné a parametre), premennej veľkosti a obsahu, neobsahuje diery,

· halda , premennej veľkosti a obsahu, môže obsahovať diery,

· pamäť pre overlaye (prekryvné segmenty), dynamické knihovne.

Toto rozdelenie umožňuje, aby procesy, ktoré sú riadené rovnakým programom zdieľali kód programu a konštanty (úspora vnútornej pamäte). Pri systémoch so zdieľanou pamäťou je vhodné zdieľané dáta uložiť do zvláštneho segmentu a ten sprístupniť všetkým procesom, ktoré ich používajú. Tým sa zlepší ochrana pamäte - procesy, ktoré používajú zdieľanú pamäť, nemajú prístup k súkromným oblastiam adresného priestoru iných procesov.

Výhody segmentácie pamäte:

· je možné dynamické premiestňovanie segmentov za behu procesu, pre odstránenie fragmentácie (môže spájať voľné bloky pamäte presunutím prekážajúceho bloku),

· možnosť dodatočne zväčšovať adresný priestor procesu,

· nie je nutné vykonávať relokáciu programu,

· možnosť zdieľania segmentov.

Nevýhody segmentácie pamäti:

· súčet nárokov procesov v pamäti musí byť menší alebo rovný veľkosti pamäti (môže obísť odkladaním segmentov na disk, čo môže byť časovo náročné), nutná hardvérová podpora segmentácie.

3.1.4.1 Odstránenie fragmentácie

V prípade, že procesy potrebujú súvislý kus pamäte, je nutné vykonať striasanie segmentov. Vykonáva sa spravidla v okamihu, kedy pri štarte nového procesu nie je žiadny voľný blok pamäti dostatočnej veľkosti, ale súčet voľných blokov stačí (niektoré systémy v prípade, že nie je dostatok voľného miesta, odkladajú adresné priestory niektorých procesov na disk, vykonávajú tzv. swapping). Striasanie segmentov je realizované prekopírovaním adresného priestoru niektorých procesov a zmenou premennej, ktorá slúži pre naplnenie registra segmentu. Existujú rôzne algoritmy, ktoré sa snažia minimalizovať veľkosť kopírovanej pamäte:

· niektoré procesory umožňujú, aby každý proces mohol používať viac segmentov,

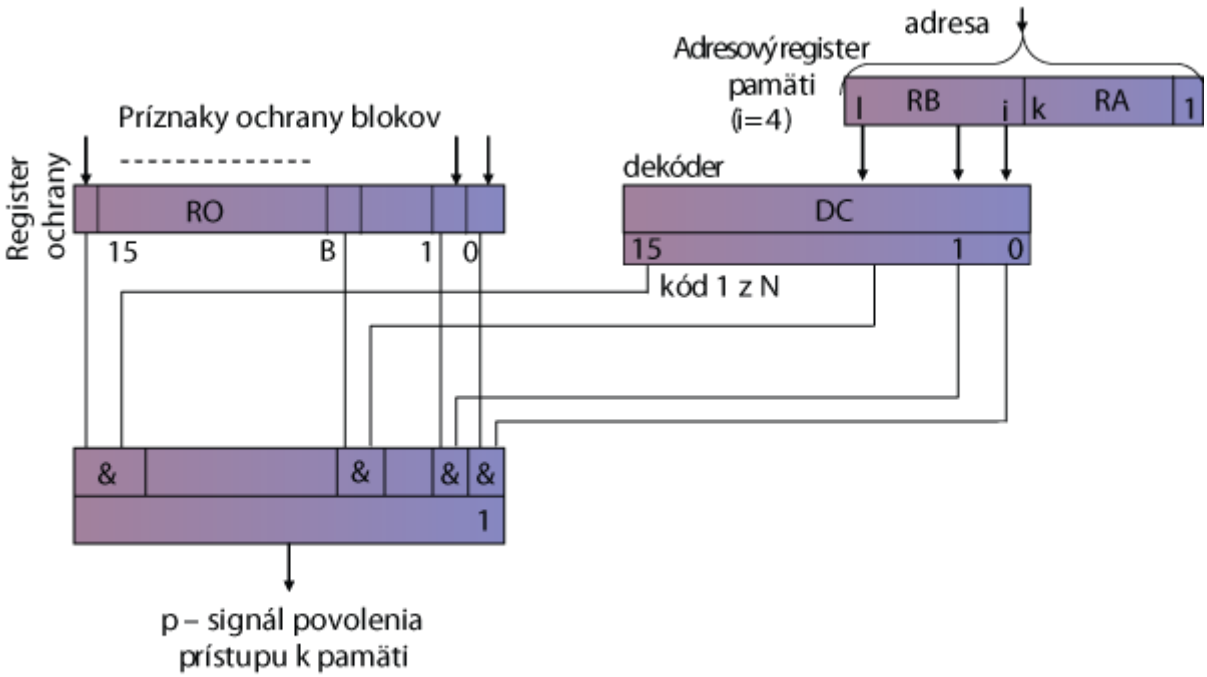
· segmentácia umožňuje, aby bežiacemu procesu bola na žiadosť pridelená ďalšia pamäť.

Ochrana pamäte pri segmentácii môže byť realizovaná:

· medzným registrom na číslo segmentu,

· pre každý segment medzným registrom, obsahujúcim maximálnu povolenú adresu segmentu (limit veľkosti segmentu); pričom segmenty môžu mať rôznu dĺžku,

· príznakovým registrom ochrany (viď nasledujúci obrázok).

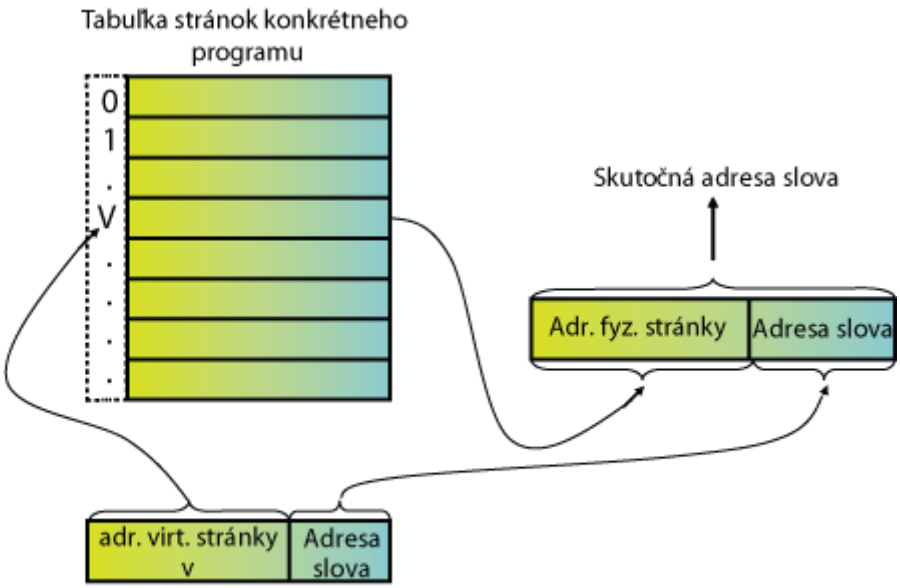


Ochrana pamäti medzným registrom

3.1.5 Stránkovanie pamäti

Procesy pre svoj beh požadujú súvislý úsek pamäte. Nutnosť prideľovať súvislé úseky pamäti a ich uvoľňovanie v ľubovoľnom poradí, podľa toho ako končia jednotlivé procesy, vedie k fragmentácii pamäte. Jednou z metód, ako sa s fragmentáciou vyrovnať, je premiestňovanie segmentov, ktoré však môže byť časovo náročné. Stránkovanie pamäte umožňuje prideliť procesu niekoľko nesúvislých úsekov pamäte a vytvoriť pre proces ilúziu, že táto pamäť je súvislá. Pri stránkovaní pamäte je fyzická pamäť rozdelená na rámce - frames (niekedy sa nerozlišuje rámec a stránka).

Logická adresa (= adresa použitá v programe) je rozdelená na dve zložky, číslo stránky a posunutia v rámci stránky (OFFSET). Veľkosť stránky býva radovo kilobyty . Pri veľkosti stránky 4 KB je pre offset potrebných 12 bitov (2^12 = 4K), alebo spodných 12 bitov logickej adresy je offset, zostávajúce bity sú číslo stránky. Po rozklade adresy (všetko vykonáva procesor bez asistencie programátora) na číslo stránky a offset, sa číslo stránky použije ako index do tabuľky stránok (každý proces má svoju vlastnú). V tabuľke stránok je uvedené číslo rámca vo fyzickej pamäti. K číslu rámca sa pripojí offset a výsledkom je fyzická adresa v pamäti.



Princíp stránkovania pamäti

Výhody stránkovania pamäte:

- odstránenie fragmentácie,
- nie je nutné premiestňovanie blokov pamäte.

Nevýhody stránkovania pamäte:

- posledná stránka procesu nebýva celkom využitá a preto veľkosť stránok nesmie byť príliš veľká (tzv. vnútorné fragmentácie),
- nutná HW podpora,
- súčet pamäťových nárokov procesov v pamäti nemôže prekročiť veľkosť fyzickej pamäte.

Ochrana pamäte, znamená znemožnenie použitia adresy mimo adresný priestor procesu, je realizovaná pomocou medzného registra, ktorý obsahuje maximálne číslo stránky pre príslušný proces. Procesu potom nie je dovolené meniť obsah tabuľky stránok.

3.1.6 Stránkovanie na žiadosť (demand paging)

Z pozorovania princípov prideľovanie pamäti vyplýva, že väčšina procesov sa chová tak, že po určitú dobu používa niekoľko málo oblastí pamäti a s priebehom procesu sa tieto oblasti menia relatívne pomaly (princíp lokality pamäťových odkazov). Vďaka tomu nie je nutné, po celú dobu behu procesu, udržiavať celý jeho adresný priestor v pamäti. Adresovanie funguje podobným spôsobom ako pri obyčajnom stránkovaní. V tabuľke stránok je však pre každú stránku údaj, či sa stránka nachádza v pamäti alebo na disku. V prípade, že je stránka na disku, je uvedené i jej umiestnenie na disku. Pre stránkovanie sa používa spravidla zvláštny súbor, partition (oblasť na disku) alebo dokonca disk. V minulosti sa používali bubnové pamäte s mnohými hlavami, ktoré spracovávali niekoľko bitov súčasne (pre zvýšenie rýchlosti).

V prípade, že proces odkazuje na stránku, ktorá je prítomná vo fyzickej pamäti, všetko prebieha ako pri bežnom stránkovaní. Pokiaľ však stránka vo fyzickej pamäti nie je (je na disku), dôjde k vyvolaniu vnútorného prerušenia - "výpadok stránky". Obslužný program prerušenia musí do vnútornej pamäte zaviesť stránku z disku, opraviť odkaz v tabuľke stránok a zaistiť zopakovanie inštrukcie, ktorá výpadok stránky spôsobila. Pokiaľ je vo vnútornej pamäti voľné miesto, použije sa ľubovoľný z voľných rámcov. Pokiaľ sú všetky rámce plné, je nutné vybrať niektorý z nich a preniesť ich do stránkovacieho súboru na disk. Existuje niekoľko algoritmov nahrádzania stránok alebo "výberu obete".

Výhody stránkovania na žiadosť:

- odstránenie fragmentácie,
- nie je nutné premiestňovanie blokov pamäte,
- súčet pamäťových nárokov procesov v pamäti môže prekročiť veľkosť fyzickej pamäte.

Nevýhody stránkovania na žiadosť:

- posledná stránka procesu nebýva celkom využitá, preto veľkosť stránok nesmie byť príliš veľká (tzv. vnútorná fragmentácia),
- nutná HW podpora.

Ochrana pamäte je rovnaká ako u normálneho stránkovania, naviac tu musí byť ochrana stránkovacieho súboru na disku realizovaná.

3.1.6.1 Algoritmy nahradzovania stránok

Zdanlivo najjednoduchší je algoritmus FIFO. Tento algoritmus vyhodí z pamäti stránku, ktorá je v nej najdlhšie. Bohužiaľ to môže byť stránka, ktorá sa používa trvalo, čo efektivitu algoritmu znižuje. Algoritmus FIFO tiež vykazuje tzv. FIFO anomáliu. Pokiaľ sa vykoná ten istý výpočet dvakrát, s rôzne veľkou vnútornou pamäťou, malo by pri výpočte s väčšou pamäťou dôjsť najviac k rovnakému počtu výpadkov stránok ako pri výpočte s menšou pamäťou. Pri použití stratégie FIFO to nemusí vždy platiť. Naviac, nie je jednoduché implementovať nahradzovanie stránok pomocou stratégie FIFO. V praxi sa preto používajú iné algoritmy.

Optimálny algoritmus nahradzovania stránok by vyhodil z pamäte tú stránku, ktorá v budúcnosti nebude používaná najdlhšiu dobu. Predpovedať budúcnosť však nie je možné, preto sa používajú algoritmy, ktoré pre odhad budúceho chovania, používajú chovanie v minulosti. Algoritmus LRU (least recently used) vyhadzuje z pamäti tú stránku, ktorá nebola najdlhšiu dobu používaná. Implementácie tohto algoritmu môže používať buď register, udávajúci čas posledného odkazu na danú stránku alebo frontu, na jej začiatok sa zaraďuje stránka, na ktorú bol práve prevedený odkaz. Či má byť z pamäti niektorá stránka vyhodená, vyberie sa tá, ktorá nebola používaná najdlhšie (v prípade použitia fronty je to posledná v rade). Algoritmus LRU je síce kvalitný, ale jeho hardvérová implementácia je obtiažna. Preto sa používa zjednodušenie algoritmu LRU, nazývané NUR (not used recently). V tomto prípade je každému rámcu priradený jednobitový príznak, či bola príslušná stránka použitá. Pri hľadaní obete sa vyberie tá stránka, ktorá použitá nebola.

V algoritmoch je vhodné brať do úvahy nahradzovanie stránok, či bol obsah stránky zmenený. V prípade, že nebol, stačí stránku iba zahodiť (jej kópia je na disku). Pokiaľ bola stránka zmenená, je nutné ju nahrať na disk, čo trvá približne rovnako dlho ako načítanie novej stránky z disku. Pre tento účel býva pre každý rámec k dispozícii jednobitový príznak, ktorý sa vynuluje pri zavedení stránky do vnútornej pamäte a nastaví pri zápise do rámca.

3.1.7 Segmentácia so stránkovaním na žiadosť

V prípade stratégie prideľovania pamäti segmentáciou so stránkovaním na žiadosť, sa logická adresa určí z čísla segmentu, čísla stránky a offsetu na stránke. Každý proces má vlastnú tabuľku segmentov a každý segment má vlastnú tabuľku stránok (pre zdieľaný segment je tabuľka len jedna).

Výhody segmentácie so stránkovaním na žiadosť:

- odstránenie fragmentácie,
- je možné používať viac pamäte (virtuálna pamäť), než je veľkosť fyzickej pamäte,
- je možné zdieľať segmenty.

Nevýhody segmentácie so stránkovaním na žiadosť:

- zložitosť,
- nutná podpora hardvéru.