Správa primárnej pamäte

Pamäť je centrum operácií moderného počítačového systému. Pamäť pozostáva z veľkého poľa slov alebo bajtov, každý má svoju vlastnú adresu. CPU vyberie inštrukcie z pamäte podľa hodnoty **čítača inštrukcií** (*Program Counter*). Tieto inštrukcie môžu zapríčiniť ďalšie zavádzanie a ukladanie dát na špecifické adresy pamäte.
Typický **cyklus vykonávania inštrukcií** najskôr vyberie inštrukciu z pamäte. Inštrukcia je potom dekódovaná a môže spôsobiť, že sa z pamäte vyberú operandy. Po tom, ako inštrukcia bola vykonaná na operandoch, výsledok inštrukcie môže byť uložený späť do pamäte.
Pamäťová jednotka vidí len **prúd adries pamäte;** nevie, ako sú generované (čítačom inštrukcií, indexáciou, priamo, nepriamo a tak ďalej) alebo čo reprezentujú (inštrukcie alebo dáta). Preto môžeme ignorovať, ako je adresa pamäte je generovaná programom. Zaujíma nás iba postupnosť pamäťových adries generovaná bežiacim programom.
Hlavná pamäť musí obsluhovať operačný systém aj rôzne užívateľské procesy. Z toho dôvodu potrebujeme prideliť rôzne časti pamäte čo najefektívnejšie.
Pamäť je zvyčajne rozdelená na dve oblasti: jedna pre príslušný operačný systém a jedna pre užívateľské procesy

Prideľovanie adries

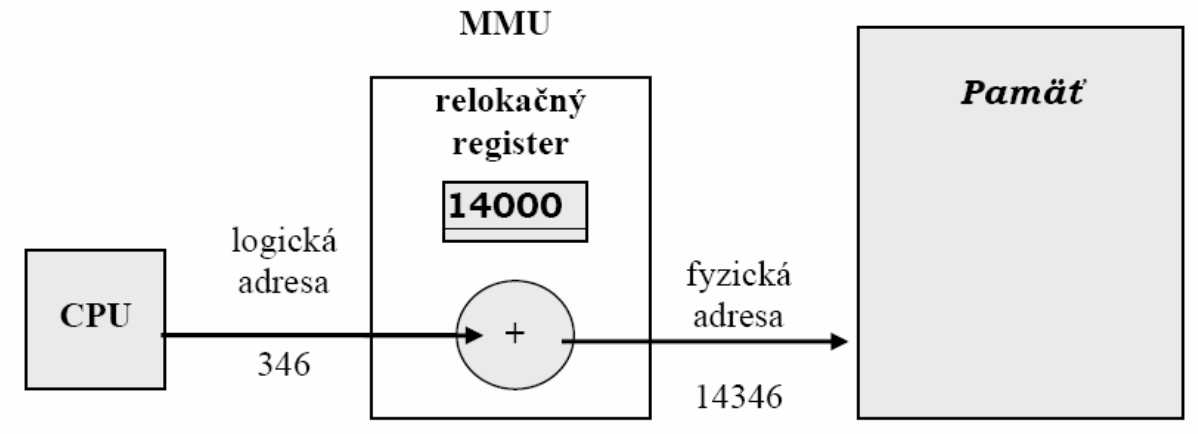
Zvyčajne program je uložený na disku ako **binárny spustiteľný súbor**. Program musí byť prenesený do pamäte a uložený vo vnútri procesu, aby sa mohol vykonať. Kolekcia procesov na disku, ktorá čaká na prenesenie do pamäte, aby bola vykonaná, tvorí **vstupný front**. V závislosti na použitom riadení pamäte, proces môže byť prenášaný medzi diskom a pamäťou počas svojho vykonávania.
Normálna procedúra je vybrať jeden z procesov vo vstupnom fronte a zaviesť tento proces do pamäte. Počas vykonávania procesu, proces pristupuje k inštrukciám a dátam z pamäte. Napokon, proces skončí a jeho pamäťový priestor je prehlásený ako voľný.
Vo väčšine prípadov užívateľský program prejde niekoľko krokov — niektoré z nich môžu byť nepovinné — skôr, ako je vykonaný. Adresy môžu byť reprezentované rôznym spôsobom počas týchto krokov. Adresy v zdrojovom programe sú spravidla **symbolické adresy.** Kompilátor obvykle **priradí** (*Bind*) tieto symbolické adresy k **relokovateľným adresám** (napr. „14 bajtov od začiatku tohto modulu"). Spojovací editor alebo zavádzač priradí tieto relokovateľné adresy k **absolútnym adresám** (napríklad 74014). Každé priradenie je mapovaním (zobrazením) z jedného adresovateľného priestoru do druhého.
Klasicky, priradenie inštrukcií a dát k adresám pamäte sa môže uskutočniť na hocijakom kroku tejto cesty:

• **Čas kompilácie:** Ak počas kompilácie viete, kde proces bude sídliť v pamäti, tak môže byť generovaný **absolútny kód**. Napríklad, ak viete, že užívateľský proces začína na pozícii *R*, potom generovaný kompilačný kód bude začínať na tejto pozícii. Ak v neskoršom čase sa zmení počiatočná adresa, potom bude nevyhnutné prekompilovať tento kód. MS–DOS programy formátu `.COM` sú absolútne kódy určené v čase kompilácie.
• **Čas zavádzania:** Ak v čase kompilácie nie je známe, kde proces bude uložený v pamäti, potom kompilátor musí vygenerovať **relokovateľný kód** (*premiestniteľný kód*). V tomto prípade, finálne prideľovanie adries je zadržané až do času zavádzania. Ak sa zmení počiatočná adresa, potrebujeme len znovu zaviesť užívateľský kód s ohľadom na túto zmenenú hodnotu.
• **Čas vykonávania:** Ak proces môže byť presunutý počas svojho vykonávania z jedného pamäťového segmentu do druhého, tak prideľovanie adries musí byť zadržané až po čas behu.

Aby táto schéma fungovala, k dispozícii musí byť špeciálny hardvér. Väčšina univerzálnych operačných systémov používa túto metódu.

Logický — verzus fyzický adresový priestor

Na adresu, ktorá je generovaná CPU sa zvyčajne odvolávame ako na **logickú adresu**, zatiaľ čo na adresu, ktorú vidí jednotka na riadenie pamäte a register pamäťových adries sa zvyčajne odvolávame ako na **fyzickú adresu**.
Metódy na prideľovanie adresy v čase kompilácie a v čase zavádzania generujú identické logické a fyzické adresy. Avšak, schéma na prideľovanie adresy v čase vykonania má za následek odlíšenie logických a fyzických adries. Množina všetkých logických adries generovaných programom je **logický adresový priestor**; množina všetkých fyzických adries odpovedajúcich týmto logickým adresám je **fyzický adresový priestor**. Preto v schéme na prideľovanie adresy v čase vykonania logický a fyzický adresový priestor sa líšia.
Mapovanie z logických do fyzických adries v čase vykonávania sa vykonáva hardvérovým zariadením nazývaným **jednotka na riadenie pamäte** (*Memory Management Uni*t — *MMU*).
Môžeme si vybrať spomedzi množstva rozdielnych metód na vykonanie takého mapovania,. Zatiaľ si objasníme toto mapovanie na jednoduchej MMU schéme, ktorá je zovšeobecnením schémy bázového registra.



Hodnota v relokačnom registri sa *pripočíta* ku každej adrese generovanej užívateľským procesom v čase, keď je posielaná do pamäte. Napríklad, ak báza je 14000, pokus užívateľa adresovať pozíciu 0 je dynamicky relokovaný na pozíciu 14000; prístup k pozícii 346 je mapovaný na pozíciu 1434. Operačný systém MS–DOS bežiaci na procesoroch rodiny Intel 80x86 používa čtyři relokačné registre pri zavádzaní a behu procesov.
Užívateľský program nikdy nevidí *reálne fyzické* adresy. Užívateľský program sa zaoberá *logickými* adresami. Pamäť mapujúci hardvér konvertuje logické adresy na fyzické adresy. Takto, teraz máme dva rozličné typy adries: logické adresy (v rozsahu od 0 do *max*) a fyzické adresy (v rozpätí od *R* + 0 do *R* + *max* pre bázovú hodnotu *R*). Užívateľ generuje iba logické adresy a myslí si, že proces beží na pozíciách od 0 do *max*. Užívateľský program dodáva logické adresy; tieto logické adresy hardvér mapuje do fyzických adries skôr, ako sú použité. Koncepcia, že *logický adresový priestor* a *fyzický adresový priestor* sú separované, je hlavnou koncepciou pre vlastné riadenie pamäte.

Efektívne využívanie primárnej pamäte

Operačný systém môže podporovať rôzne spôsoby pre zefektívnenie využitia operačnej (fyzickej) pamäte. Napríklad:

- **Dynamické zavedenie (dynamic loading)** programov. Procedúra sa nenačíta z disku do pamäte, kým nie je po prvýkrát volaná z programu. Je to výhodné pri málo využívaných funkciách programu. Ne-výhodou je časové zdržanie na prvé volanie každej procedúry.

- **Dynamické zostavovanie (dynamic linking)**. Tu sa niektoré procedúry nepíšu, ale už ich niekto umiestnil v knižniciach a tie sa k programu pridávajú až počas jeho behu (a prvého zavolania funkcie z knižnice). Ide o dynamické knižnice Unixu či „dll“ súbory Windows. Výhodou je menšia spotreba miesta na disku, keďže napríklad vstupno-výstupné procedúry nemusí obsahovať každý program, stačí odkaz na ne do knižnice. Nevýhodou je, že ak chceme zmeniť knižnice, nastáva problém kompatibili-ty.

- **Prekrývanie (overlay)**. Často sa určité časti programov využívajú len v určitom čase (napríklad pri štartovaní programu), potom môžu byť prepísané zvyšnou časťou programu. Je to skôr vecou prekla-dača než OS, pri programovaní pre MS-DOS to podporuje napríklad Borland Pascal i Borland C/C++.

- **Odkladanie stránok na disk (paging)**. To sme si už spomínali, má ho každý moderný operačný systém a umožňuje využívať viac pamäte, než sa fyzicky nachádza v počítači. Staršie operačné systémy odkladali na pevný disk celé procesy (swapovali), dnes sa odkladajú len časti procesov – najmenej používané stránky (stránkovanie).

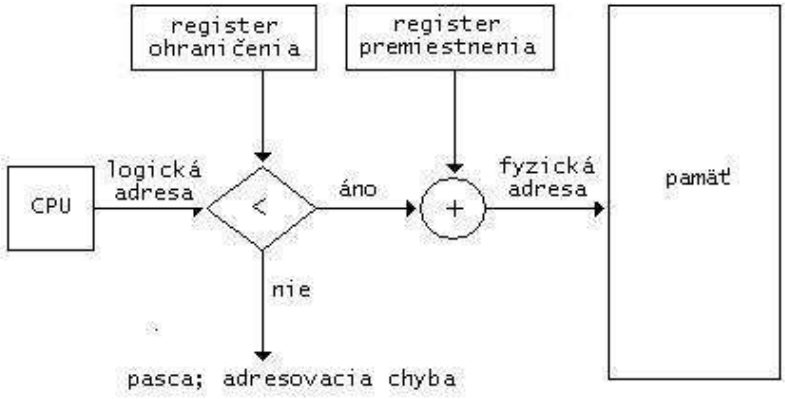
Súvislá alokácia pamäti

Zvyčajne chceme usídliť v pamäti niekoľko užívateľských procesov súčasne. Preto potrebujeme zvážiť, ako prideliť voľnú pamäť procesom, ktoré čakajú vo vstupnej fronte na disku na prenesenie do pamäti. Pri súvislej alokácii pamäti je každý proces obsiahnutý v jednej súvislej časti pamäti — partícii.

Ochrana pamäti

Predtým, ako budeme preberať pridelenie pamäti, musíme prebrať kľúčový bod: ochranu pamäti — ochrániť operačný systém od užívateľských procesov a užívateľské procesy jeden od druhého. Ochranu môžeme poskytnúť použitím relokačného registra a limitného registra.

Relokačný register obsahuje hodnotu najmenšej fyzickej adresy (adresu začiatku partície); **limitný register** obsahuje veľkosť partície. Napríklad relokácia = 100040 a limit = 74600. Každá logická adresa musí byť menšia ako limitný register; MMU mapuje logickú adresu *dynamicky* pripočítaním hodnoty relokačného registra. Táto mapovaná adresa je odoslaná do pamäti.



Pridelenie pamäti

Teraz sme pripravení vrátiť sa k prideleniu (*alokácii*) pamäti. Jednou z najjednoduchších metód pridelenia pamäti je rozdeliť pamäť na niekoľko **partícií** (*oblastí*) pevnej dĺžky. Každá partícia môže obsahovať presne jeden proces. Takže stupeň multiprogramovania je ohraničený počtom partícií. V tejto **metóde viacnásobných pevných partícií**, keď partícia je voľná, proces je vybraný zo vstupnej fronty a zavedený do voľnej partície. Keď proces skončí, partícia sa stane dostupnou pre iný proces. Táto metóda bola originálne použitá u operačného systému IBM OS/360; už sa viac nepoužíva.
Ďalej popísaná metóda sa nazýva **metóda viacnásobných variabilných partícií**; je používaná primárne v dávkovom prostredí. Operačný systém obsahuje tabuľku ukazujúcu, ktoré časti pamäte sú k dispozícii a ktoré sú obsadené. Na začiatku celá pamäť je k dispozícii užívateľským procesom a je považovaná ako jeden veľký pamäťový blok, **diera**. Keď príde proces a potrebuje pamäť, hľadáme dieru dostatočne veľkú pre tento proces. Ak ju nájdeme, pridelíme len toľko pamäti, koľko je treba, ponechávajúc zvyšok k dispozícii na uspokojenie budúcich požiadaviek.

Keď procesy vstupujú do systému, sú vložené do vstupného frontu. Operačný systém berie do úvahy pamäťové požiadavky každého procesu a množstvo dostupného pamäťového priestoru pri rozhodovaní, ktorým procesom bude prideľovať pamäť. Keď procesu je pridelený pamäťový priestor, proces je zavedený do pamäti a môže súperiť o CPU. Keď proces skončí, uvoľní pamäť, ktorú operačný systém môže potom vyplniť iným procesom zo vstupného frontu.
V nejakom danom čase máme k dispozícii zoznam veľkostí dostupných blokov a vstupný front. Operačný systém môže usporiadať vstupný front podľa nejakého algoritmu rozvrhovania. Pamäť je prideľovaná procesom zo vstupného frontu, až kým nakoniec pamäťové požiadavky nasledujúceho procesu nemôžu byť splnené; žiadny z dostupných blokov (alebo dier) nie je dostatočne veľký na uchovanie tohto procesu. Operačný systém môže čakať, až kým nie je k dispozícii dostatočne veľký blok alebo môže preskočiť vo vstupnom fronte a pozrieť sa, či nenájde iný proces s menšími pamäťovými požiadavkami. Vo všeobecnosti *množina dier* rôznych veľkostí je rozptýlená v pamäti v danom čase. Keď príde proces a potrebuje pamäť, systém hľadá v tejto množine dieru dostatočne veľkú pre tento proces. Ak diera je príliš veľká, je rozdelená na dve: jedna časť je pridelená prichádzajúcemu procesu; druhá je vrátená do množiny dier. Keď proces ukončí, uvoľní svoj blok pamäti, ktorý je potom položený späť do množiny dier. Ak nová diera susedí s inými dierami, tieto susedné diery sú zlúčené a vytvoria jednu väčšiu dieru. V tomto okamihu

systém môže skontrolovať, či existujú procesy čakajúce na pamäť a či by táto novo uvoľnená a pre usporiadaná pamäť mohla splniť požiadavky niektorého z čakajúcich procesov.
Táto procedúra je zvláštnym prípadom všeobecného **problému dynamického prideľovania pamäti**, ktorý spočíva v tom, ako splniť požiadavku veľkosti *n* zo zoznamu voľných dier. Existujú viaceré riešenia tohoto problému. Množina dier je prehľadaná kvôli určeniu, ktorú dieru je najlepšie
prideliť. Stratégie **first–fit**, **best–fit** a **worst–fit** sú tými najbežnejšími na vyhľadanie voľnej diery z množiny dostupných dier.
• *First–fit*: Prideliť prvú dieru, ktorá je dostatočne veľká. Hľadanie môže začať buď na začiatku
množiny dier alebo tam, kde skončilo predchádzajúce first–fit prehľadávanie. Hľadanie môžeme zastaviť vtedy, keď nájdeme dostatočne veľkú voľnú dieru.
• *Best–fit*: Prideliť najmenšiu dieru, ktorá je dostatočne veľká. Musíme prehľadávať celý zoznam, pokiaľ nie je zoradený podľa veľkosti.
• *Worst–fit*: Prideliť najväčšiu dieru. Musíme prehľadávať celý zoznam, pokiaľ nie je utriedený podľa veľkosti.
Simulácie ukázali, že tak first–fit ako aj best–fit sú lepšie ako worst–fit, čo sa týka využitia pamäte. Ani first–fit ani best–fit nie je jasne lepší vo využití pamäte, ale first–fit je rýchlejší.

Fragmentácia

Tieto algoritmy ale trpia externou fragmentáciou. Keď sú procesy zavádzané a odstraňované z pamäte, voľný pamäťový priestor je rozbitý na malé kúsky. **Externá fragmentácia** je, keď existuje dostatok celkového pamäťového priestoru na splnenie požiadavky, ale tento priestor nie je súvislý; pamäť je rozdelená na veľké množstvo malých dier, takže tam nie je žiadna dostatočne veľká diera pre splnenie požiadavky. Tento problém fragmentácie môže byť vážny. V najhoršom prípade by sme mohli mať blok voľnej pamäti medzi každými dvoma procesmi.
Výber first–fit alebo best–fit stratégie môže ovplyvniť stupeň fragmentácie. First–fit je lepšia pre niektoré systémy, zatiaľ čo best–fit je lepšia pre iné.
Fragmentácia pamäti môže byť externá ako aj interná. Predpokladajme, že proces požaduje 1002 bajtov a máme dieru 1004 bajtov. Ak pridelíme presne požadovaný blok, ostane nám diera 2 bajty. Náklady na udržiavanie stopy tejto 2 bajtovej diery budú podstatne väčšie ako úžitok z tejto diery. Všeobecný prístup spočíva v rozdelení fyzickej pamäti na bloky pevnej dĺžky a pridelenie pamäti, ktorej jednotka je rovná veľkosti bloku (napríklad 1 kB). Pomocou tohto prístupu môže byť pamäť pridelená procesu len nepatrne väčšia ako požadovaná pamäť. Rozdiel týchto dvoch čísel je **interná fragmentácia** — veľkosť pamäte, ktorá je interná v partícii, ale nie je použitá.

Zdroj:
M. Schmotzer: Operačné systémy
J. Studenovský: Operačné systémy