# Pod pokličkou vašeho počítače: adresování procesorů Intel x86

**Pokud vás láká pohled do "zákulisí" počítačových systémů, možná vás zaujme tento článek: popíšu v něm, jak všelijak se procesor typu x86 "dívá" na operační paměť. Začnu nejjednoduším, reálným režimem, ale většinu času se budu věnovat chráněnému režimu procesoru 386 nebo lepšího.**

15. 2. 2005 0:00 Daniel Novotný

**Nálepky:** Intel

## Reálný režim procesoru 8086 a vyšších

V tomto režimu se procesor nachází po inicializaci (resetu). Reálný režim je zpětně kompatibilní se stařičkým 8086 a současné moderní OS jej pro svou běžnou práci nepoužívají. Používá se v systémech MS-DOS, DR-DOS a podobných. Fyzická adresa je v tomto režimu dvacetibitová. Jednoduchým výpočtem přijdeme na to, že to znamená jeden megabajt paměti.

Procesor však paměť nevidí jako jeden jednomegový blok s dvacetibitovou adresou – logická adresa, tedy adresa pro procesor, se skládá ze dvou částí: segmentu a offsetu.

Jsou zde čtyři segmentové registry definující čtyři segmenty: CS (Code Segment) obsahuje kód programu, DS (Data Segment) data, SS (Stack Segment) zásobník a ES (Extra Segment) je tu ještě navíc. Segmentové registry jsou 16bitové a definují horních 16 bitů dvacetibitové adresy.

Druhá část adresy – offset – je rovněž 16bitová a definuje dolních 16 bitů dvacetibitové adresy. Můžeme ji „vzít“ z jiného registru či přímo z instrukce jako přímý operand…

Jak jste již (asi) pochopili, segment s offsetem se částečně překrývají a pro výpočet výsledné fyzické adresy se používá součet s posunutím: `adresa = (segment<<4)+offset` (nebo ekvivalentní: `adresa=segment*16+offset` …podle toho, co se vám líbí víc).

Možnosti procesoru poté determinují možnosti programovacího systému na něm založené: vzpomínám si, jak jsem se kdysi divil, proč nemůžu mít v TurboPascalu pole delší než 64 KB (šestnáctibitový offset dál „nedosáhne“), proč se ukazatele dělí na blízké a vzdálené (blízké jsou pouze offsety v rámci segmentu, vzdálené segment:offset) …

Kromě paměťových omezení je nevýhodou také to, že program nikdo nehlídá: kdokoli může zapisovat do segmentových registrů, a z toho vyplývá, že libovolný program teoreticky může číst, zapisovat, nebo skákat kamkoli do celého paměťového prostoru – pokud se v DOSu špatně zachoval jeden program, mohl bez problémů „seknout“ celý počítač.

Tímto s reálným režimem skončím. Dovolím si přeskočit šestnáctibitový chráněný režim procesoru 286 (maximální paměť tu byla 16MB, ale velikost segmentu byla stále maximálně 64 KB) a proberu podrobně adresování v chráněném režimu 386.

## Chráněný režim procesoru 386 a vyšších

Existují dva způsoby řízení virtualizace paměti a přístupových práv: segmentace a stránkování. Procesor 386 je podporuje pro jistotu oba. Názvy jednotlivých adres jsou následující:

logická adresa –segmentace → lineární adresa –stránkování → fyzická adresa

Stejně jako v případě reálného režimu se logická adresa skládá ze segmentové a offsetové části. Offset je nyní 32bitový, ale to není to nejdůležitější – struktura paměti je nyní daleko složitější než u reálného režimu:

Offset zůstal offsetem, jen se zvětšil, ale číslo v segmentovém registru nyní nijak přímo nepřičítáme: segmentový registr se v chráněném režimu skládá z dvoubajtové (16bitové) viditelné části zvané *selektor* a osmibajtové neviditelné části zvané *deskriptor*.

Probereme si nejdříve selektor:

Horních třináct bitů selektoru je indexem do tabulky deskriptorů. Následuje bit, který určuje, zda bude selektor indexovat globální tabulku (přístupnou přes registr GDTR), nebo lokální tabulku (přístupnou přes registr LDTR). Nejnižší dva bity určují úroveň oprávnění segmentu – o tom budeme hovořit později.

Jakmile se nějaký „k tomu určený“ (např. segmentový) registr naplní selektorem, „natáhne“ se do jeho neviditelné části příslušný deskriptor, na který selektor v dané tabulce ukazoval. Tam zůstane „cachovaný“ – segmentové registry se nemění tak často, ale používají se stále, a takhle je deskriptor v registru, čili rychle přístupný.

Deskriptor – ať již z tabulky deskriptorů, nebo „natažený“ do neviditelné části nějakého registru – obsahuje 32bitovou *bázi*, což je počáteční lineární adresa daného segmentu. Dále obsahuje *limit* daného segmentu – ten ohraničuje, kde segment končí. Limit se počítá malinko složitěji – je to dvacetibitové číslo plus bit „G“ – „Granularity“. Pokud G=0, dvacetibitový limit se počítá po bajtech a maximální velikost takového segmentu je tedy 1 MB. Pokud naopak G=1, počítá se limit po čtyřech KB. Maximální velikost takového segmentu je 4 GB – a to už „celkem stačí“ :-) . Také zde je jeden bajtík obsahující *přístupová práva* segmentu: zda je možno jej číst/zapisovat/spouštět a na jaké úrovni oprávnění musí proces být, aby mu to bylo povoleno. Zde je již vidět, proč se vlastně chráněný režim jmenuje „chráněný“: procesor hlídá procesy, aby „nešlapaly tam, kam nemají“ – nemohu tedy překročit hranice segmentu, nechtěně zapisovat do kódu nebo se naopak snažit spustit kus paměti, kde jsou data: procesor mi to nedovolí, taková akce se neprovede a místo ní procesor „vyhodí výjimku“ v podobě přerušení. Takováto přerušení ošetřuje OS, a ten už si sám rozhodne, co s „provinilým“ procesem udělá: z těchto „luhů a hájů“ pochází např. signál SIGSEGV a známá UNIXová chybová hláška `Segmentation fault`.

Tabulek deskriptorů může být více; jedna je globální, „kořenová“. Její lineární adresa (tedy „přímá“ 32bitová adresa) je spolu s limitem (velikostí tabulky) uložena v registru GDTR. Toto je speciální případ, kdy je lineární adresa uložena přímo ve viditelné části registru, všude jinde se adresuje přes deskriptory a selektory.

V globální tabulce se kromě deskriptorů „normálních“ datových či kódových segmentů nachází i deskriptor popisující „dceřinou“, lokální tabulku deskriptorů. Lokálních tabulek může být více, třeba pro každý proces jedna, nebo (druhý extrém) může OS používat pouze globální tabulku a na lokální se „vykašlat“. Další dva druhy deskriptorů jsou deskriptor brány a deskriptor TSS. *Brána* se používá v situaci, kdy chce proces na nízké úrovni oprávnění volat podprogram úrovně vyšší – můžete si to představit jako SETUID. Nebylo by žádoucí, aby měl takovýto program možnost skočit na libovolnou instrukci vyšší úrovně oprávnění (nebo dokonce kód z tohoto segmentu číst) – bránu si potom je možno představit opravdu jako „branku“ do jiného světa: do jiného segmentu nevidím a nemohu z něj nic spouštět, kromě zavolání jednoho konkrétního vstupního bodu podprogramu (určeného bránou). O posledním typu deskriptoru – TSS čili Task Switch Segment – se jen letmo zmíním: tam se ukládají obsahy registrů procesoru při přepnutí kontextu časovačem.

Nyní bych se rád zmínil o *úrovních oprávnění*. Ty se kódují dvoubitovým neznaménkovým číslem: 0 znamená nejvyšší privilegia a 3 nejnižší. Úroveň privilegií daného procesu je zapsána v dolních dvou bitech jeho registru CS.

Některé instrukce (zvané „ring 0“) může provádět pouze proces na úrovni 0. Úroveň 3 se považuje za „běžnou uživatelskou“ a u některých operací (např. co se týče stránkování- viz níže) se na ní uplatňují dodatečné kontroly. Ještě bych se zmínil o IOPL, což jsou dva bity registru FLAGS, které definují nejnižší možnou úroveň, při níž jsou povoleny vstupně-výstupní (I/O) instrukce.

Definice jemných odstínů úrovní oprávnění je na OS – většina UNIXů je „dělaná“ na dvouúrovňový model SupervisorMode/UserMode – je tedy možné používat pouze úrovně 0 a 3 a o mezistupně se nezajímat, stejně tak je možné využít těchto rozdílů – například pokud máme mikrojádro (úroveň 0), na něm nějaké servery (úroveň 1), pod tím knihovny (úroveň 2) a pod nimi uživatelské aplikace (úroveň 3) – fantazii se meze nekladou…

Abychom se vrátili k hlavnímu tématu: lineární adresa vzniká jako součet báze daného segmentu (segmenty definuje programátor/překladač/OS) a offsetu (to je ta část ukazatele, se kterou běžně manipulují uživatelské programy). Nyní přichází ke slovu stránkovací jednotka.

Účelem stránkování je hlavně virtualizace paměti. Procesy nám vznikají a zanikají, alokovat jim souvislé bloky paměti (třeba pro velké segmenty) se nedaří, také zjišťujeme, že se nám všechny naráz do paměti nevlezou a potřebovali bychom swapovat na disk…

Lineární adresa se rozdělí na tři části: 10–10–12 bitů.

Prvních deset bitů je indexem do *stránkového adresáře*. Ukazatel na stránkový adresář je uložen v registru CR3, jehož obsah se ukládá spolu s ostatními registry procesu při přepnutí kontextu.

Položka stránkového adresáře ukazuje na *stránkovou tabulku*. Tu indexuje druhých deset bajtů z lineární adresy. Položka ze stránkové tabulky již (konečně!) obsahuje ukazatel do skutečné, fyzické paměti, který na závěr posuneme (přičteme) o posledních dvanáct bitů lineární adresy.

Pointa je v tom, že ne všechny stránky musí být opravdu nahrané v paměti. V adresáři a tabulce je příznak určující, jestli jsou odkazovaná data přítomna v RAM. Přístup na stránku, která v RAM není, má za následek přerušení „výpadek stránky“, na které reaguje OS nahráním příslušné stránky z disku. Také je zde bit určující, zda daná stránka změnila obsah: to je důležité pro rozhodnutí, zda ji po dealokaci zahodit, nebo vypsat zpátky na disk. Také jsou zde přístupová práva, která mohou zakázat procesu s úrovní privilegií 3 přístup či zápis do stránky.

Je toho hodně, že? To musí být složité a pomalé, říkáte si. Naštěstí jednak existuje důmyslný systém vyrovnávacích pamětí (deskriptory segmentů se cachují v neviditelné části registru, pro stránky existuje vyrovnávací paměť zvaná TLB), jednak není nutno využívat všech těchto možností nepřímých odkazů.

Stránkování lze vypnout vynulováním nejvyššího bitu registru CR0. Lineární adresy jsou pak shodné s fyzickými.

Segmentaci takto jednoduše „vypnout“ nejde, ale nic nám nebrání nadefinovat si jeden velký segment s bázovou adresou 0 a limitem rovným velikosti operační paměti – a přiřadit selektor tohoto segmentu všem segmentovým registrům. Tím „zploštíme“ dvourozměrný pohled na paměť na jednorozměrný (nazývaný někdy „flat memory model“), jako ukazatele můžeme používat pouze 32bitové offsety…

Celkem existují tyto možnosti:

1. vypnout stránkování a „zploštit“ segmenty: dostáváme rychlý přístup k celé operační paměti, bez přístupových práv a podobných „zdržování“. Užitečné, pokud neprogramujeme víceuživatelský OS, ale třeba nějaký řadič V/V zařízení, podobného modelu využívaly také některé DOSové hry běžící v chráněném režimu
2. vypnout stránkování a ponechat pouze segmentaci: neznám žádný systém používající tento model, možná se poučím z reakcí čtenářů
3. stránkování zapnout, řešit pomocí něj přístupová práva a segmenty víceméně „zploštit“: takto funguje většina UNIXových systémů pro PC – Linux, FreeBSD…, také 32bitová Windows fungují tímto způsobem. Má to tu výhodu, že adresování v programech se „dělá“ pouze pomocí 32bitových offsetů – na segmenty můžeme zapomenout, ochrana paměti funguje pomocí „stránkovacích“ přístupových práv, programy „vidí“ svoji paměť jako souvislý blok od nuly dál adresovaný jednoduše pouze 32bitovým registrem, ovšem ve fyzickém adresovém prostoru může být paměť „na kusy“. Kam proces nemá „sahat“, to mu vůbec nenamapujeme, takže to „nevidí“.
4. mít zapnuto obojí – stránkování i segmentaci: nejsložitější, nejobecnější model. Využívá jej například systém OS/2.

Tak, to by bylo ode mě všechno. Rozmotejte zamotané hlavy, a až příště uvidíte Segmentation fault či Neplatnou operaci, už víte, odkud vítr vane.