



KIV Operační systémy

Režim jádra a uživatelský režim



Chceme, aby...

- ..mohlo zároveň běžet několik procesů
- ..proces nemusel vědět, že zároveň s ním běží další procesy
- ..bylo jedno, kde v paměti běží který proces
- ..proces nemohl přistupovat do paměti jiného procesu či jádra
- ..výsledkem nebylo zpomalení počítače
- ..výsledkem bylo zefektivnění práce na počítači



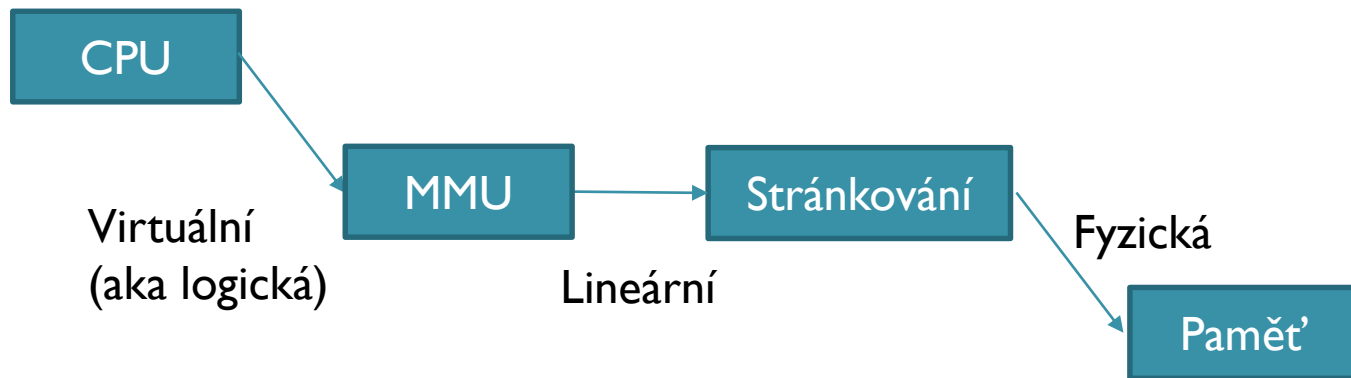
Memory Management Unit

- Výše uvedené cíle implikují, že procesy budou pracovat se svým formátem (tj. virtuální) adresy do paměti, která bude následně převedena na fyzickou adresu do paměti
- Protože sw implementovaný převod by byl pomalý, převod virtuální na fyzickou adresu obstará hw
 - Konkrétně Memory Management Unit (MMU)
 - Detaily převodu zadá MMU přímo OS



segment:offset

- x86 pracuje s virtuální adresou ve formátu segment:offset
- Z ní je třeba získat tzv. lineární adresu, a teprve tu lze převést na fyzickou adresu v režimu, ve kterém dokážeme od sebe izolovat jednotlivé procesy a jádro





Protected mode

- V první řadě musíme zabránit uživatelským procesům, aby nemohly modifikovat kód a data jádra
- Jelikož x86 adresuje pomocí segmentu a offsetu (vůči segmentu), řešením bylo připojit dodatečné informace ke konkrétním segmentům
 - => už nepracujeme se segmentem, ale se segment deskriptorem
 - Procesor běží v tzv. Protected mode



Protected mode - přepnutí

- Poprvé ho implementoval 80286, komplikované přepnutí
- 80386 ho rozšířila a zjednodušila jeho přepnutí

EnablePM:

```
cli                                ;zamaskování přerušení

mov eax, cr0

bts eax, 0                        ;přepneme do

mov cr0, eax                      ;chráněného režimu

lgdt [newGDT]                    ;nastavíme selektory

lidt [newIDT]                    ;přerušení a výjimky

mov eax, selSS

mov ss, eax                      ;zásobník

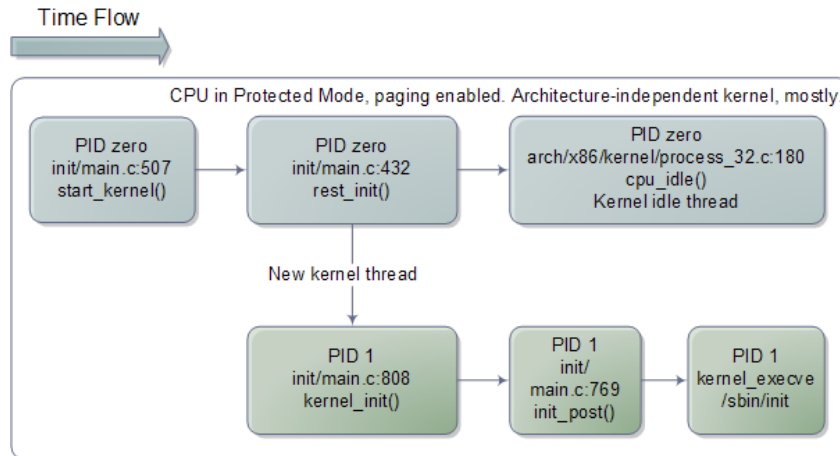
ljmp selCS, @pm                  ;nastav segment selektor kódu:EIP
```

pm:

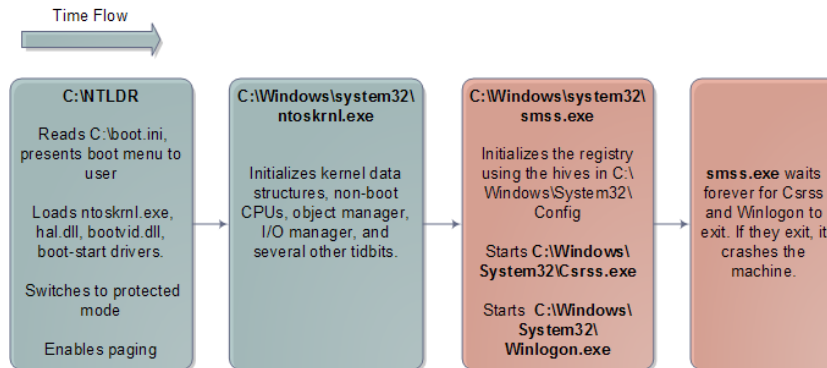
```
ret                                ;a vrátíme se v pm
```

Protected mode – zavedení OS

- Linux



- Windows NT



<http://duartes.org/gustavo/blog/post/kernel-boot-process/>

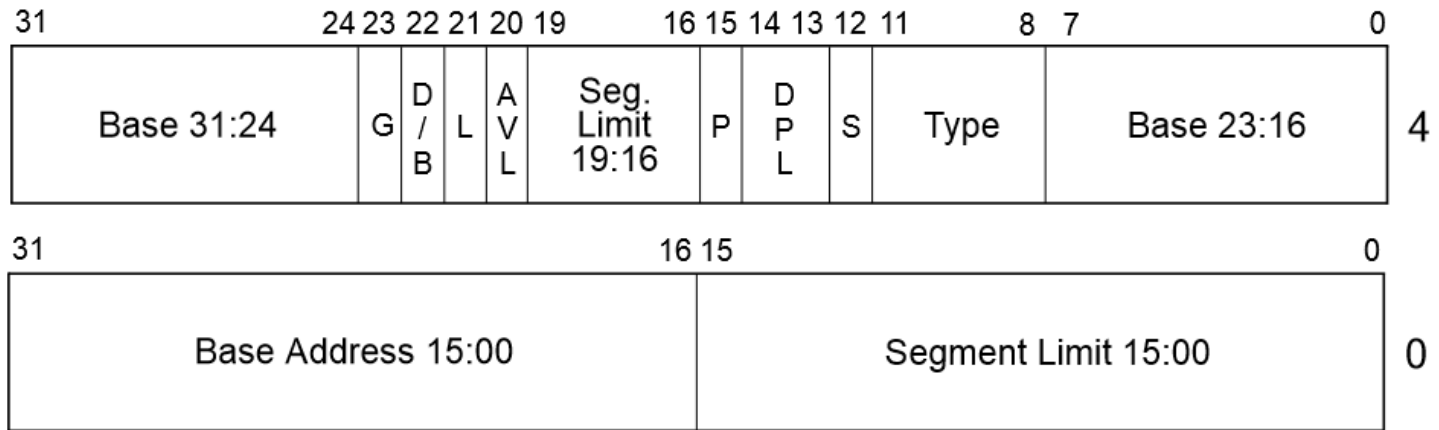


Segment Descriptor

- Segment má:
 - Základní (base) adresu (lineární adresa)
 - Velikost (limit)
 - Typ (kód, data, atd.)
 - Přístupová práva
 - Další vlajky



Segment Descriptor



- L — 64-bit code segment (IA-32e mode only)
- AVL — Available for use by system software
- BASE — Segment base address
- D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
- DPL — Descriptor privilege level
- G — Granularity
- LIMIT — Segment Limit
- P — Segment present
- S — Descriptor type (0 = system; 1 = code or data)
- TYPE — Segment type

Intel® 64 and IA-32 Architectures
Software Developer's Manual
Volume 3A:
System Programming Guide, Part 1



Izolace jádra

- Pro izolaci je důležitá hodnota bitů CPL/DPL
- Číslo oprávnění/privilege level aktuálního kódu
 - 00 - většinou jádro
 - 01 – může být jádro, když na 00 poběží hypervizor virtualizace
 - 10 – může být ovladač, který nesmí do jádra
 - 11 – většinou uživatelský proces
- Kód s nižším číslem oprávnění může přistupovat do segmentu s vyšším číslem oprávnění
- Opačně to nelze => izolace jádra od uživ. procesů



Izolace procesů

- Před vlastní inicializací protected mode musí jádro OS vytvořit tabulku deskriptorů segmentů
- Tabulky existují dvě
 - Globální používaná jádrem
 - uložena v registru gdtr privilegovanou instrukcí lgdt
 - Lokální používaná konkrétním jedním procesem
 - Uložena v registru ldtr privilegovanou instrukcí lldt
 - Tj. při přepnutí kontextu může lldt vykonat pouze jádro s CPL=0 a tudíž si uživatelský proces nemůže měnit jemu přidělenou paměť



Segment deskriptory v kódu

- Uživatelský proces může mít až 6 segmentů s deskriptory uloženými v cs, ds, ss, es, fs a gs
 - V 64-bitovém long-mode jsou nulové až na fs a gs
- Adresa v paměti má prefix požadovaného segmentu
 - `call cs:[adresa_funkce]`
 - `mov eax, ds:[adresa_retezce]`
 - `mov ecx, ss:[ebp-offset_parametru_funkce]`
 - V ebp bývá hodnota esp při volání funkce, tzv. frame pointer



Výhody segmentace

- Izoluje procesy a jádro
- Lze sdílet segmenty – např. read-only programový kód sdílených knihoven
- Lze relokovat i pouze jeden segment
- Není nutné alokovat nevyužitou paměť
- Tabulka deskriptorů se vejde do MMU



Nevýhody segmentace

- Segmenty mohou mít různou délku => jak je poskládat do paměti?
- Segmenty mohou být velké => fragmentace paměti
- Jak efektivně implementovat sdílenou paměť mezi procesy?
- Jak efektivně odkládat paměť z RAM na disk, abychom zvýšili celkovou dostupnou paměť počítače?



Stránkování

- Odstraňuje nevýhody segmentace
 - x86 umožňuje kombinovat segmentaci a stránkování
 - Nelze povolit stránkování bez protected mode
 - x86 ji umožňuje použít v protected (32-bit) a long (64-bit) mode
- Celá paměť se rozdělí do stránek o pevné velikosti
 - 4kB, 2MB, 4MB a 1GB
 - Virtuální stránka: page (o velikosti frame)
 - Fyzická stránka: frame (o velikosti page)



Page Table

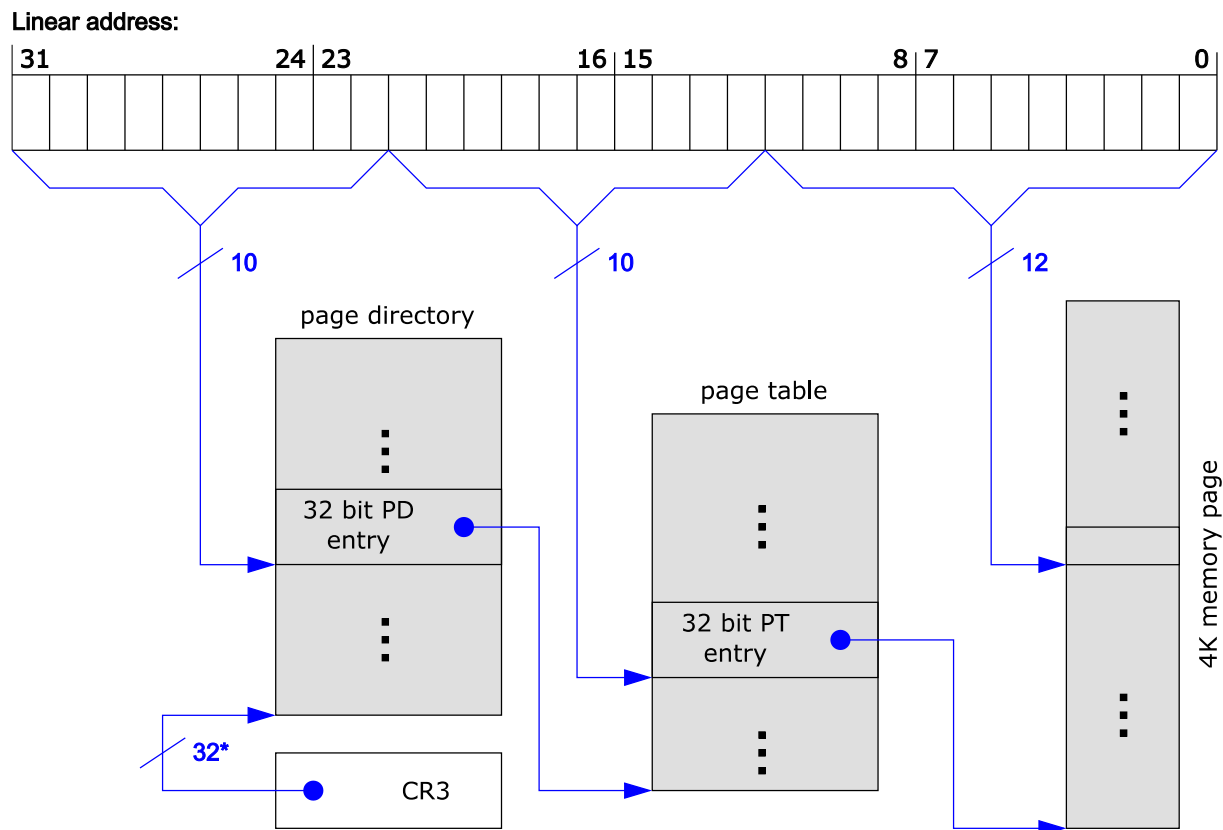
- Jádru OS vytvoří tabulku stránek
 - Řídící registr (x86) cr3 ukazuje na tuto tabulku stránek
 - Pouze jádro s CPL=0 může měnit obsah cr3
 - Tj. dosáhneme stejného efektu jako s GDT a LDT
 - Procesor nastaví cr3 při přepnutí kontextu
- Každá stránka má své vlajky, mj. zahrnující
 - Jádro vs. Uživatelský proces
 - Writeable, NX – do not execute
 - Present (zda je fyzicky v RAM), Dirty a Accessed



Hierarchická Page Table

- Jenom několik málo procesů využije celý dostupný paměťový prostor, např. 2^{32}
- Řešením je vytvořit vnější/nadřízenou/page directory tabulku stránek, která bude odkazovat na další tabulku stránek
 - Paměťový prostor procesu pak nemusí být spojitý, ale mohou v něm být díry na místech, které program nepoužívá

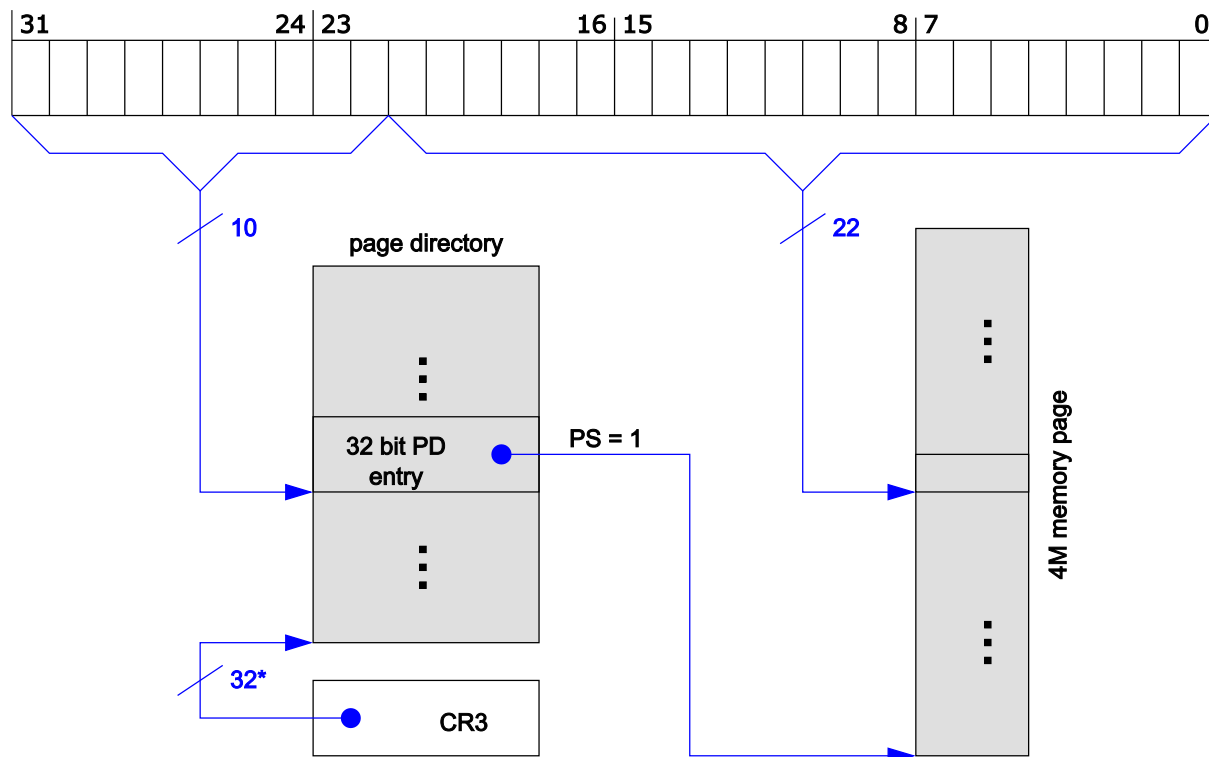
Stránkování 4kB



*) 32 bits aligned to a 4-KByte boundary

Stránkování 4MB

Linear address:



*) 32 bits aligned to a 4-KByte boundy



Translation look-aside buffer

- Běžící proces často potřebuje jenom omezené množství stránek
 - Jak tedy zrychlit převod virtuální adresy na fyzickou?
 - Pomocí asociativní paměti Translation look-aside buffer (TLB)
 - Je rychlá, ale také je malá

Page index	Frame index



Stránkování s TLB

- Běžící proces často potřebuje jenom omezené množství stránek
 - Jak tedy zrychlit převod virtuální adresy na fyzickou?
 - Pomocí asociativní paměti Translation look-aside buffer (TLB)
 - Musí se vyprázdnit při změně kontextu
 - Je rychlá, ale také je malá
 - Větší hit rate když je větší, ale také je pak pomalejší
 - Má zásadní vliv na výkon

Page index	Frame index



Sdílená paměť

- Efektivní cesta jak sdílet data mezi různými procesy
- Procesy jsou zodpovědné za konsistenci dat ve sdílené paměti
- OS pouze zajistí sdílení paměti
- Do tabulky stránek procesu OS přidá page entry, která ukazuje na ten samý rámec (fyzická stránka) jako page entries v tabulkách stránek ostatních procesů, které tímto „trikem“ sdílejí tu samou paměť

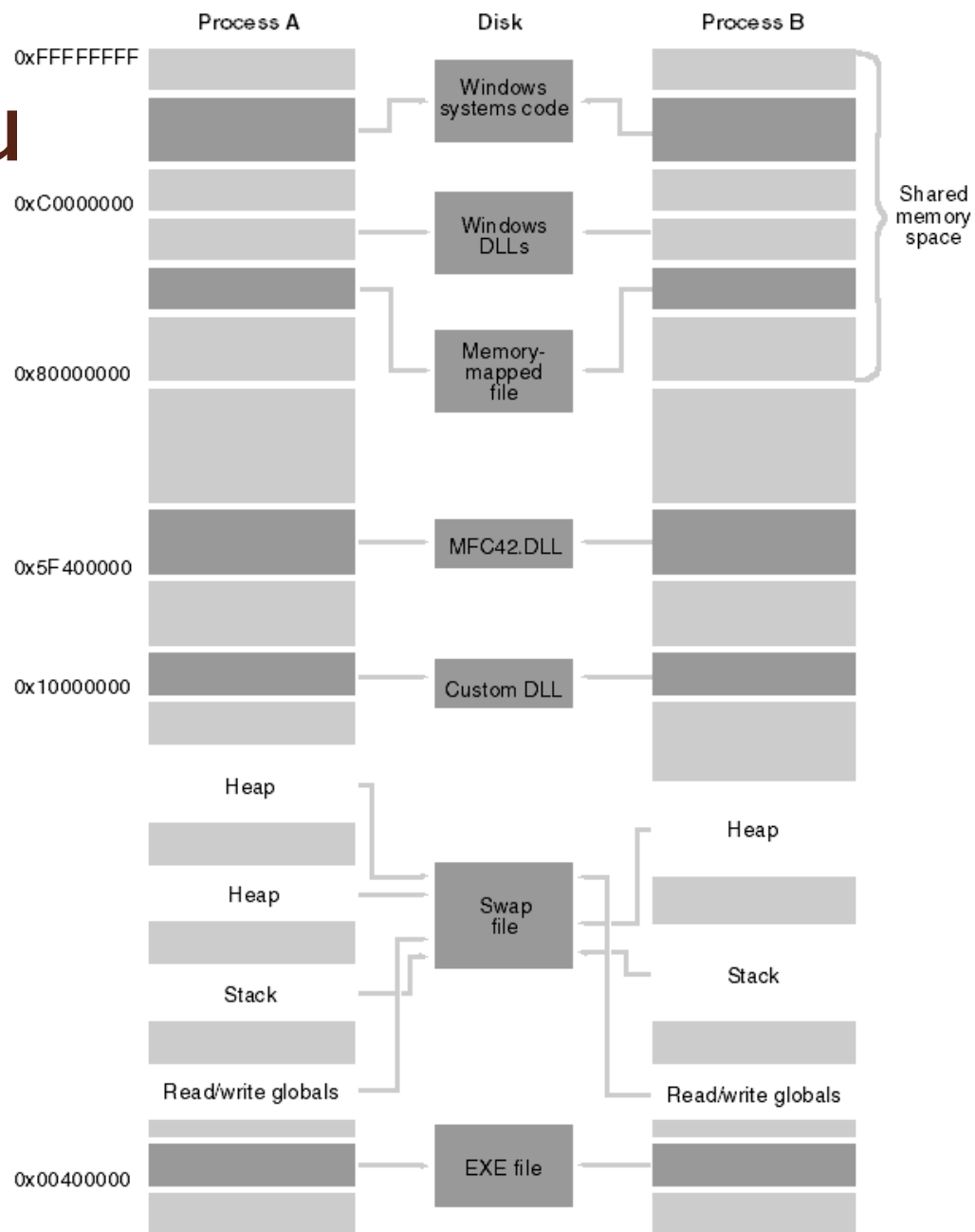


Sdílení kódu

- Pokud několik procesů používá sdílený programový kód, proč ho do paměti nahrávat vícekrát?
- Příslušné stránky s kódem se označí jako read-only a namapují se do paměťových prostorů příslušných procesů
 - Tj. stále jde o sdílenou paměť
- Příkladem jsou dynamické knihovny
 - dll
 - so



Sdílení kódu





Copy on write

- Co když několik procesů na začátku sdílí stejná data, která považují za privátní, ale mění je jenom zřídka?
 1. Příslušná stránka se označí jako read-only
 - Dokud z ní proces jenom čte, nic se neděje
 2. Jakmile se proces pokusí zapsat do read-only stránky, procesor vygeneruje vyjímku, kterou zachytí OS
 3. OS pak alokuje novou stránku, zkopíruje do ní původní read-only stránku, a aktualizuje tabulku stránek procesu
 4. Po návratu z obsluhy proces normálně zapíše data už do nové stránky, aniž by o něčem vůbec věděl



Swapování

- Chceme-li poskytnout procesům více paměti, než kolik máme fyzicky RAM, nezbyvá než paměť z RAM dočasně odložit do jiné paměti – flash nebo pevný disk
- Working Set – dynamická množina stránek, které má proces aktuálně ve fyzické paměti
- Chce-li dát OS více paměti některému procesu a není volný rámec, zmenší Working Set jiného procesu odebráním stránky, přičemž uloží obsah rámce na disk



Page Fault

1. Proces chce číst data ze stránky, která není v RAM
2. MMU nedokáže přeložit virtuální adresu na fyzickou adresu
 - => vygeneruje výjimku PageFault
3. OS uloží rámec některé jiné stránky, třeba i jiného procesu, na disk a načte do něj stránku požadovanou aktuálním procesem
4. OS příslušně upraví tabulky stránek dotčených procesů
5. Po dokončení proces dál normálně pokračuje aniž by něco poznal



Page Fault - následky

- Ve skutečnosti OS nebude měnit working sets při každém Page Fault
 - Namísto toho se požadavky mohou nasbírat a vyřídit později – hromadně
 - Page Fault-ovaný proces se mezitím může pozastavit
- Swapování má extrémně negativní vliv na výkon systému
 - Zaměstnává procesor, disk a příslušnou i/o sběrnici



Mapování souborů

- Chceme-li zrychlit práci se souborem, je možné OS požádat o namapování souboru do paměti
- Proces pak zapisuje a čte přímo z rychlé paměti, namísto práce s pomalejším diskem
- OS namapuje soubor do paměti po stránkách jako u swap souboru
- Při ukončení práce se souborem se pak na disk zapíše pouze ty stránky, které mají nastavený dirty bit
 - Pokud by měl soubor např. několik GB, proč zapisovat vše?



Změna privilege level

- Uživatelský proces běží ve svém paměťovém prostoru
- Pokud volá funkci jádra operačního systému, které vyžaduje vyšší úroveň oprávnění, např. CPL=0, je nutné nějak změnit privilege level
- Dělá to procesor v okamžiku, kdy obsluhuje přerušení at' už int nebo IRQ
 - Úroveň oprávnění určí z adresy/vektoru obsluhy přerušení
 - Obsluha přerušení běží se zvýšeným CPL tak dlouho, dokud neudělá iret



Physical Address Extension (PAE)

- Éra 32-bitových x86 s více než 4GB RAM
- Ačkoliv procesor, tj. i jádro OS, dokázalo adresovat pouze 4GB RAM, tabulky stránek mohly adresovat více než 4GB RAM
- V podstatě šlo první kroky ke 64-bitovým tabulkám stránek – u PAE s 36-bitovou fyzickou adresou
 - Zavedeno s Pentium Pro
 - Nicméně, už 386 by teoreticky zvládla 64TB virtuální paměti – technická realizace v praxi je ovšem něco jiného



80386 virtuální adresový prostor

- Virtuální adresy mají 48-bitů – dáno MMU
 - 16 bitů segment selektor, 32 bitů offset v rámci segmentu
- $16 + 32 = 48$ bitů virtuálního adresy
 - Ale 2 bity selektoru jsou použity na privilege level
 - 1 bit selektoru indikuje globální/lokální tabulku
 - \Rightarrow 46 k adresování použitelných bitů $\Rightarrow 2^{46} = 64\text{TB}$
- Jedná se ale jenom o teoretický limit! V praxi nikdy nebylo použito.



Segmentace a stránkování

- Ačkoliv je segmentace považovaná za historický artefakt, nedá se říci, že by byla nepoužívaná
 - Např. Vx32 user-level sandboxing na x86 ji používá ke spouštění nedůvěryhodných programů na FreeBSD, Linuxu a Mac OS
 - Implementace TLS ve Windows – viz dále



Segmented Paging

- Tabulky stránek jsou segmentovány
- Virtuální adresa je `logical_page:offset`
- `logical_page` je `segment_number:segment_offset`



Paged Segmentation

- Segmenty se skládají ze stránek
- Virtuální adresa je seg:offset
- Offset je page_number:page_offset
- Way to go on x86



HW vs SW

- Proč to zatím vypadá jako specifikace hw?
- Protože at' už je to Linux, UNIX, Mac OS či Windows, všichni nakonec dělají to samé
 - A takhle už víme, co to je, aniž bychom se upnuli na konkrétní implementaci