

# B4B35OSY: Operační systémy

## Lekce 5. Systém NOVA a stránkování

Petr Štěpán

stepan@fel.cvut.cz



19. října, 2022

# Outline

1 Správa paměti

2 Virtualizace paměti

# Obsah

**1** Správa paměti

**2** Virtualizace paměti

# Názvosloví

## ■ FAP

- fyzický adresní prostor
- skutečná paměť počítače – RAM
- velikost závisí na možnostech základní desky a na osazených paměťových modulech

## ■ LAP

- logický adresní prostor
- někdy také virtuální paměť
- velikost záleží na architektuře CPU
  - 16 bitová adresace – 64 KiB
  - 20 bitová adresace – 1 MiB
  - 32 bitová adresace – 4 GiB
  - 64(48) bitová adresace – 16 EiB (256TiB)

# Počítače bez správy paměti

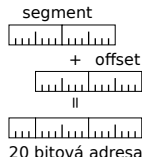
- Výhody systému bez správy paměti:
  - rychlost přístupu do paměti
  - jednoduchost implementace
  - lze používat i bez operačního systému – robustnost
- Nevýhody systému bez správy paměti
  - Nelze kontrolovat přístup do paměti (kdokoli může cokoli v paměti přepsat)
  - Omezení paměti vlastnostmi HW
- Použití
  - Historické počítače
    - Osmibitové počítače (procesory Intel 8080, Z80, apod.)
    - 8bitová datová sběrnice, 16bitová adresová sběrnice, možnost využít maximálně 64 kB paměti
  - Programovatelné mikrokontrolery
  - Řídicí počítače – embedded
  - V současné době již jen ty nejjednodušší řídicí počítače 8/16-bitové (např. Atmel Xomega)

# Jednoduché segmenty

## Jednoduché segmenty – Intel 8086

- Procesor 8086 má 16bitovou datovou sběrnici a registry
- Procesor má 20 bitů adresové sběrnice.
- 20 bitů je ale problém. Co s tím?
- Řešením jsou jednoduché segmenty:

- Procesor 8086 má 4 tzv. segmentové registry
- Adresa je tvořena adresou segmentu 16 bitů a
- adresou uvnitř segmentu (offset) 16 bitů.
- Výsledná FA se tvoří podle pevného pravidla:
  - $adr = (segment \ll 4) + offset$

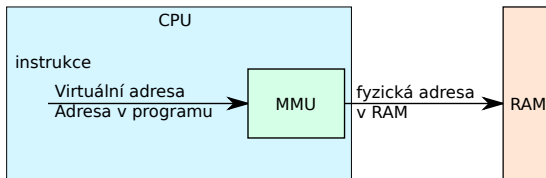


- Prostředek, jak používat větší paměť než dovoluje systém
- Využívá se i v současnosti u jednoduchých 16-bitových procesorů (např. Infineon xc167)
  - někdy se hovoří o mapování
- zavádí dva druhy adres:
  - near – uvnitř segmentu, pouze 16bitový ukazatel
  - far – mezi segmenty, 16bitový ukazatel a 16 bitů číslo segmentu

# Segmentace

## Skutečné segmenty – Intel 80286

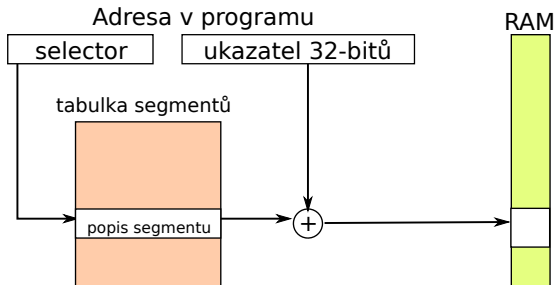
- První procesor s MMU na čipu
  - MMU – memory management unit – převádí adresu (16 bit selektor, 16 bit offset) na 24 bitů adresa ve fyzické paměti



- Program je kolekce segmentů
- Každý segment má svůj logický význam:
  - hlavní program, procedura, funkce,
  - objekt a jeho metoda, proměnné, pole, ...

# Segmenty – Intel 80286

- Základní úkol – převést adresu typu (segment selector, offset) na adresu FAP
  - CS, DS, SS, ES – code, data, stack, extra segment selector 16 bitů
    - bit 0–1 RPL – request privilege level – úroveň ochrany
    - bit 2 TI – 0 – global descriptor (patří všem procesům), 1 – local descriptor (jen pro jeden proces)
    - bit 3–15 – index v tabulce





# Segmenty – Intel 80286

- Tabulka segmentů – Segment table (ST) – Zobrazení 2-D (segment, offset) LAP do 1-D (adresa) FAP
- ST je uložena v normální paměti RAM, informace vybraných segmentech CS, DS, ES, SS je uložena uvnitř CPU
  - Registr L/GDT – Local/global descriptor table 24-bitů – umístění tabulky segmentů v paměti
  - Registr LL/GDT – limit Local/global descriptor table 16 bitů – počet segmentů procesu, velikost tabulky
- Položka ST (64 bitů):
  - base – 24 bitů – počáteční adresa umístění segmentu ve FAP,
  - limit – 16 bitů – délka segmentu (max. 64 KiB)
  - práva – 8 bitů – P – presnet, DPL – descriptor privilege level, S – Segment descriptor (system/user) oprávnění ke zápisu, E- executable, ED Expansion direction (>limit, <limit), w- Writeable, A =accessed
  - rezerva – 8 bitů – pro 386 rozšíření
    - 386 modifikuje segmenty na 20 bitů limit, 32 bitů adresa
    - G – velikost segmentu je v bajtech, nebo v  $2^{12}$  tj. 4KiB – maximální velikost  $2^{32}$
- vzdálené skoky – lcall – long call, int – i změna segmentu, lret, iret – vzdálený návrat

# Segmentace – vlastnosti

## ■ Výhody segmentace

- Segment má délku uzpůsobenou skutečné potřebě
  - minimum vnitřní fragmentace
  - Lze detekovat přístup mimo segment, který způsobí chybu segmentace – výjimku typu „segmentation fault“
- Lze nastavovat práva k přístupu do segmentu
  - Operační systém požívá větší ochrany než aplikační proces
  - Uživatel nemůže ohrozit operační systém
- Lze pohybovat s daty i programem v fyzické paměti
  - posun počátku segmentu je pro aplikační proces neviditelný a nedetekovatelný

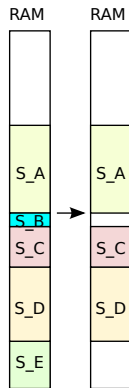
## ■ Nevýhody segmentace

- Alokace segmentů v paměti je netriviální úloha
  - Segmenty mají různé délky. Při běhu více procesů se segmenty ruší a vznikají nové.
  - Problém s externí fragmentací
- Režie při přístupu do paměti
  - Převod na lineární adresu se opírá o tabulku segmentů a ta je také v paměti
  - Při změně segmentového registru – nutné načíst položku z tabulky
  - Častá změna segmentů (po pár instrukcích) – časově náročná

# Alokace segmentů

Obecný problém – alokace bloků paměti – umístění segmentu v RAM

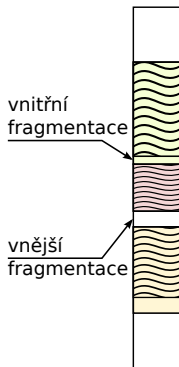
- "Díra-blok neobsazené paměti
- Segmentu se přiděluje díra, která jeho požadavek uspokojí
  - tím může vzniknout další malá díra
- Díry jsou roztroušeny po FAP
- Evidenci o dírách a obsazených místech udržuje jádro OS
- Kde přidělit oblast délky  $n$ , když je volná paměť rozmístěna ve více souvislých nesousedních sekcích?
  - First fit – první volná oblast dostatečné velikosti – rychlé, nejčastější
  - Best fit – nejmenší volná oblast dostatečné velikosti – neplýtvá velkými děrami, mohou vznikat mini-díry
  - Worst fit – největší volná oblast – zanechává velké volné díry vhodné pro další alokaci



# Fragmentace

## Obecný problém nevyužitelného prostoru

- Externí (vnější) fragmentace
  - Celkové množství volné paměti je sice dostatečné, aby uspokojilo požadavek procesu, avšak prostor není souvislý, takže ho nelze přidělit
  - Existence mnoha malých děr
- Interní (vnitřní) fragmentace
  - Přidělená díra v paměti je o málo větší než potřebná, avšak zbytek je tak malý, že ho nelze využít
- Redukce externí fragmentace pomocí setřásání
- Přesouvají se obsahy úseků paměti s cílem vytvořit (jeden) velký souvislý volný blok
- Použitelné pouze při dynamické relokaci
- Při absolutních adresách v paměti by bylo nutno přepočítat a upravit všechny adresy v instrukcích
- Problém s I/O: S vyrovnávacími paměťmi plněnými z periférií (zejména přes DMA) nelze kdykoli hýbat, umísťují se proto do prostoru JOS



# Stránkování

## Stránkování – Intel 80386

- Procesor 386 – 32 bitový procesor – přidal k segmentům stránkování:
  - Souvislý LAP procesu není zobrazován jako jediná souvislá oblast FAP
- FAP se dělí na úseky zvané rámce
  - Pevná délka, zpravidla v celistvých mocninách 2
    - (512 až 8.192 B, nejčastěji 4KiB, ale někdy i 4 MiB)
- LAP se dělí na úseky zvané stránky
  - Pevná délka, shodná s délkou rámců
- Proces o délce  $n$  stránek se umístí do  $n$  rámců
  - rámce ale nemusí v paměti bezprostředně sousedit
- Mechanismus překladu logická adresa → fyzická adresa
  - pomocí tabulky stránek (PT = Page Table)
- Může vznikat vnitřní fragmentace
  - stránky nemusí být zcela zaplněny

## Kvíz – velikost stránek a rámců

Na čem závisí velikost stránek a rámců?

- A - Na velikosti paměti.
- B - Na velikosti disku.
- C - Na architektuře procesoru.
- D - Na velikosti adresové sběrnice paměti.

# Stránkování – překlad adres

- Logická adresa použitá v programu se dělí na:
  - číslo stránky,  $p$  (index do tabulky stránek)
    - tabulka stránek obsahuje počáteční adresy rámců přidělených stránkám
  - posunutí (offset) ve stránce,  $d$ 
    - relativní adresa (posunutí = offset, displacement) ve stránce/v rámci
- Protože velikost stránky je mocnina 2, je rozklad na číslo stránky a posunutí jednoduchý
  - číslo stránky  $p = \frac{addr}{2^k} = addr \gg k$
  - posunutí  $off = addr \% 2^k = addr \& (2^k - 1)$
- V jazyce C je rozklad na číslo stránky pro 32-bitový systém s 4KiB stránkami
  - číslo stránky  $p = addr \gg 12$
  - posunutí  $off = addr \& 0x00000FFF$

# Obsah tabulky stránek

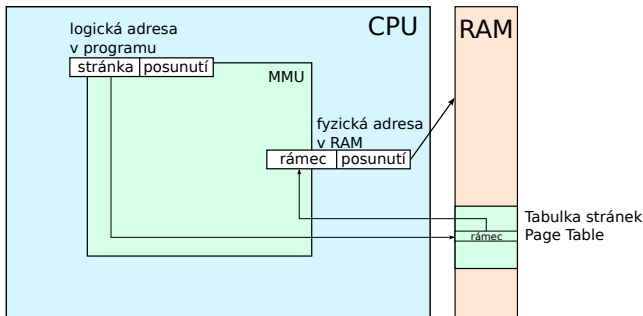
## Položky tabulky stránek:

- Číslo rámce
  - umístění stránky v reálné paměti počítače
- Atributy stránek
  - Základní příznaky
    - p present – stránka je v paměti, číslo rámce je platné
    - ps page size – velikost stránky
    - g global – stránka je globální, nepatří jednomu procesu
  - Řízení přístupu
    - r/w read/write – povolení zápisu do stránky
    - u/s user/supervisor – povolení přístupu pro uživatele
  - Optimalizace
    - pwt page-level write through – nastavení cache
    - pcd page-level cache disable – zákaz použití cache
  - Statistika
    - a accessed – stránka použita pro čtení
    - d dirty – stránka použita pro zápis
  - Virtualizační příznaky
    - v/i = valid/invalid indikuje přítomnost stránky ve FAP
    - a = accessed značí, že stránka byla použita
    - d dirty indikuje, že obsah stránky byl modifikován



# Efektivita tabulky stránek

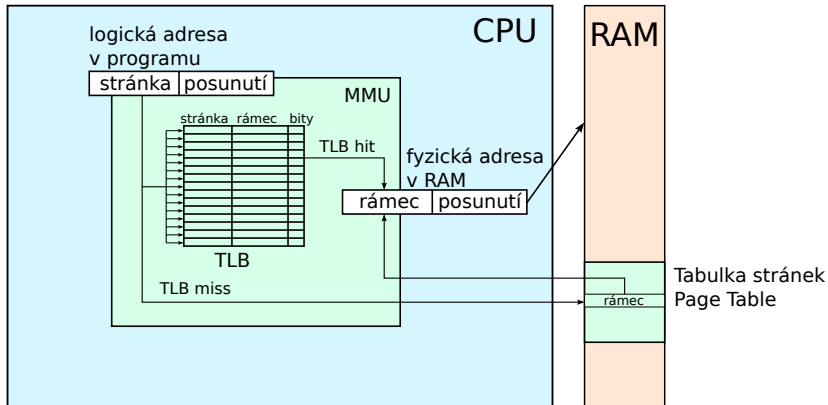
- každý přístup do paměti znamená, že je potřeba převést číslo stránky na číslo rámce, to znamená číst z RAM
- jedna instrukce může číst i dvakrát z paměti z různých stránek
- jedna instrukce tedy potřebuje 4 přístupy do paměti
- přístup do paměti velmi zdržuje



# TLB

- Řešení – speciální rychlá cache paměť pro čísla rámců a čísel stránek – Translation Lookaside Buffer
- asociativní paměť – paměť adresovaná obsahem
  - oproti normální paměti, ptám se, kde v paměti je hodnota 15?
- TLB je asociativní paměť
- relativně malá kapacita, vysoká efektivita a zrychlení přístupu do paměti
- nevýhoda – obvodová složitost implementace TLB
- MMU se zeptá TLB znáš hodnotu rámce pro číslo stránky  $p$ ?  
Odpověď buď ano je to  $r$  (TLB hit), nebo neznám (TLB miss)

# Tabulka stránek



# Význam TLB

- Skutečná přístupová doba – Effective Access Time (EAT) – 10–100 cyklů procesoru
- Přístupová doba TLB – 0.5 – 1 cyklus procesoru
- Neúspěšnost TLB, TLB miss – 0.01 % – 1 %
- Příklad:
  - Přístupová doba do fyzické paměti  $t = 30$ cyklu, do TLB  $t_{TLB} = 1$ cyklus
  - Neúspěšnost TLB  $\alpha = 0.01$  (jedno procento)
  - Stránkování bez TLB  $t_{celkem} = 2 \cdot t = 60$ cyklu
  - Průměrná doba přístupu do paměti s TLB
$$t_{celkem} = \alpha \cdot (t_{TLB} + 2 \cdot t) + (1 - \alpha) \cdot (t_{TLB} + t) = 31.3$$
cyklu

# Vliv TLB

- Velikost TLB 8–4096 položek
- Moderní procesory mají více-úrovň TLB – podobně jako úrovně cache
- Intel Core i7 má 64 TLB položek L1 první úrovně a 1536 TLB položek L2 úrovně
- I pro malé TLB je úspěšnost nalezení položky 99–99.99% (souvisí s principem lokality tj. prostorovou závislostí programů)
- Problém TLB je při změně procesu a tím i změně tabulky stránek
- Intel umožňuje speciálními instrukcemi ponechat stránku v TLB

# Velikost tabulky stránek

## ■ Otázka velikosti PT

- Každý proces má svoji PT
- 32-bitový LAP, 4 KiB stránky – PT má 1 Mi položek, tj. velikost 4 MiB
- 64-bitový LAP, 4 KiB stránky – PT má 4 Pi (peta) položek, tj. velikost 32 PiB
- musí být stále v hlavní paměti

## ■ Hierarchické stránkování

- Zobrazování LAP se dělí mezi více úrovní PT
- Pro 32-bitový LAP typicky dvouúrovňové PT
- PT 0 obsahuje definice (odkazy) vlastních tabulek PT 1
- Tabulky stránek nižších úrovní mohou být odkládány na disk
- v RAM lze zobrazovat jen skutečně využitá stránky s vlastními PT

## ■ Hašovaná PT

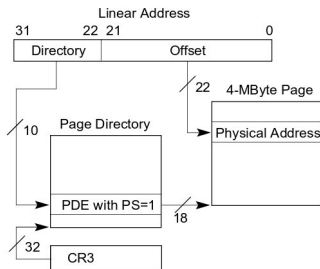
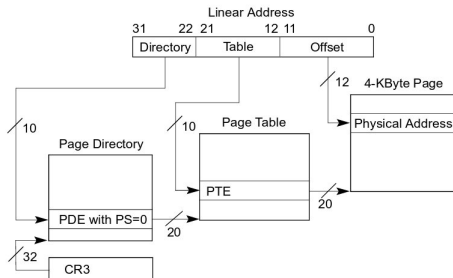
- Náhrada přímého indexování číslem  $p$  v PT hašovací funkcí  $\text{hash}(p)$

## ■ Invertovaná PT

- Jediná PT pro všechny koexistující procesy
- Počet položek je dán počtem fyzických rámců
- Vyhledávání pomocí hašovací funkce  $\text{hash}(pid, p)$

# Dvouúrovňové stránkování 32-bitů

- 32-bitový procesor se stránkou o velikosti 4 KiB – 12 bitů posunutí (offset)
- 10 bitů index v tabulce tabulek (page directory –  $PT_0$ )
- 10 bitů index v tabulce stránek (page table –  $PT_1$ )
  - při nastaveném bitu PS – velikost stránky 4 MiB, nepoužije se tabulka  $PT_0$



## Kvíz

Kdo provádí převod virtuální adresy na fyzickou?

- A - operační systém v rámci přerušení.
- B - operační systém jako svoji službu.
- C - CPU konkrétně MMU.
- D - sběrnice propojující CPU a paměť.



# Více úrovněové stránkování (32bitů) – bitová aritmetika

Tabulka tabulek - vrchních 10 bitů adresy

```
pdir[virt >> 22]
```

Test přítomnosti tabulky stránek v paměti

```
if ((pdir[virt >> 22] & PRESENT) == 0)  
// tabulka stranek není v paměti
```

Pozice tabulky stránek v paměti

```
ptab = pdir[virt >> 22] & ~PAGE_MASK;
```

Tabulka stránek - prostředních 10 bitů adresy

```
ptab[(virt >> PAGE_BITS) & 0x3ff]
```

## NOVA stránkování 32-bitů

```
#define PAGE_BITS      12
#define PAGE_SIZE     (1 << PAGE_BITS)
#define PAGE_MASK     (PAGE_SIZE - 1)
```

```
class Ptab {
public:
enum {
PRESENT = 1<<0,
RW      = 1<<1,
USER    = 1<<2,
ACCESS  = 1<<5,
DIRTY   = 1<<6, };
static void insert_mapping (mword virt, mword phys, mword attr);
static void * remap (mword addr);
};
```

```
void Ptab::insert_mapping (mword virt, mword phys, mword attr) {
mword* pdir = static_cast<mword*>(Kalloc::phys2virt(Cpu::cr3()));
mword* ptab;
if ((pdir[virt >> 22] & PRESENT) == 0) { // add ptab
ptab = static_cast<mword*>(Kalloc::allocator.alloc_page(1, Kalloc::FILL_0));
mword p = Kalloc::virt2phys (ptab);
pdir[virt >> 22] = p | ACCESS | RW | PRESENT | USER;
} else { // find ptab
ptab = static_cast<mword*>(Kalloc::phys2virt (pdir[virt >> 22] & ~PAGE_MASK));
}
ptab[(virt >> PAGE_BITS) & 0x3ff] = (phys & ~PAGE_MASK) | (attr & PAGE_MASK);
}
```

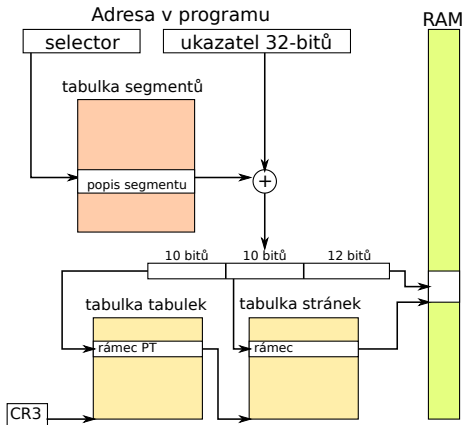
# Kvíz – obsah tabulky stránek

V tabulce stránek je uloženo:

- A - číslo stránky, číslo rámce, příznakové bity.
- B - číslo rámce, příznakové bity.
- C - pouze číslo stránky.
- D - pouze příznakové bity.

# Segmentace se stránkování IA-32

- V 32 bitovém módu nelze zrušit používání segmentů
- LAP: 2x8 Ki segmentů s délkou až 4 GiB každý
- Logická adresa = (popisovač segmentu, offset), offset = 32-bitová adresa v segmentu
- Lineární adresní prostor všech segmentů se stránkuje s použitím dvouúrovňového mechanismu stránkování
- Délka stránky 4 KiB, offset ve stránce 12 bitů, číslo stránky 2x10 bitů
- OS to řeší nafouknutím segmentů na 4GiB – což ve skutečnosti ruší segmenty



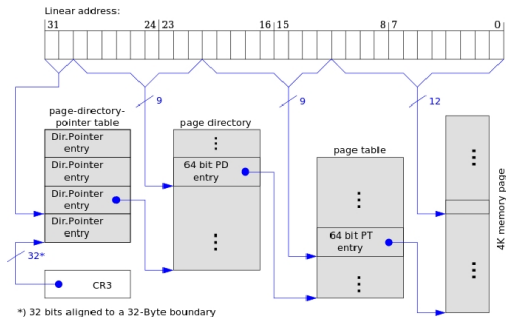
# Kvíz

Více-úrovňové stránkování:

- A - **zrychluje** nalezení odpovídající paměti, ale **zvyšuje** velikost paměti pro uložené tabulky.
- B - **zrcyhluje** nalezení odpovídající paměti a **zmenšuje** velikost paměti pro uložené tabulky.
- C - **zpomaluje** nalezení odpovídající paměti, ale **zmenšuje** velikost paměti pro uložené tabulky.
- D - **zpomaluje** nalezení odpovídající paměti i **zvyšuje** velikost paměti pro uložené tabulky.

# Tříúrovňové stránkování 32-bitů s PAE

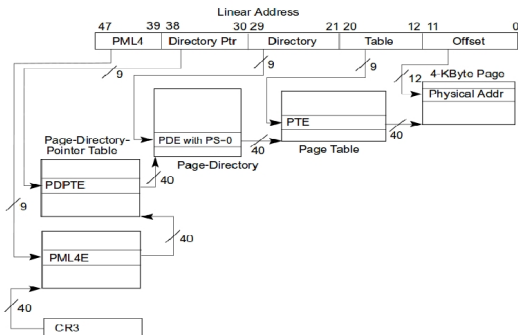
- PAE – fyzická paměť až 4 PiB – 52 bitů pro adresaci, tedy číslo rámce má 40 bitů
- potřebujeme více bitů pro uložení čísla rámce → položka v tabulce stránek bude mít 64 bitů
- do 4KiB se vejde jen 512 položek, tzn. index do této tabulky má 9 bitů
- 32-bitový procesor se stránkou o velikosti 4 KiB – 12 bitů posunutí (offset)
- 9 bitů index v tabulce tabulek
- 9 bitů index v tabulce stránek
- 2 bitů index v tabulce stránek druhé úrovně



# Stránkování IA-32e

4× pomalejší pokud není nalezen překlad v TLB.

- Lineární adresa 48 bitů – Virtuální prostor o velikosti 256 TiB
- Fyzická adresa 52 bitů – což je 4 PiB RAM
- Varianty s 4KiB, 2MiB nebo 1GiB stránkami
- Posunutí 12 bitů, 21 bitů, nebo 30 bitů
- 9 bitů indexy do tabulek tabulek/stránek



# Kvíz – ukazatele

Pokud v programu je ukazatel `int *p = &a;` Pak ukazatel `p` obsahuje:

- A - virtuální adresu.
- B - fyzickou adresu v RAM.
- C - ukazatel do stránkovací tabulky.
- D - záleží na typu překladače.



# Obsah

1 Správa paměti

2 Virtualizace paměti

# Virtualizace paměti

- Sdílený kód
  - Jediná read-only kopie kódu ve FAP sdílená více procesy
    - více virtuálních instancí editoru, shellů, jen jednou ve FAP.
- Privátní data
  - Každý proces si udržuje svoji vlastní virtuální kopii kódu a svoje reálná data
  - Stránky s privátním kódem a daty mohou být kdekoliv v LAP
- Sdílená data
  - Potřebná pro implementaci meziprocesní komunikace (mmap)

# Odkládání na disk

- Úsek FAP přidělený procesu je vyměřován mezi vnitřní a vnější (sekundární) paměťi oběma směry
  - Uložení stránek na disk, načtení z disku
  - Swap out, swap in (roll out, roll in)
  - Trvání výměn je z podstatné části tvořena dobou přenosu mezi paměťi a diskem je úměrná objemu vyměřované paměti
- Při nenalezení stránky ve fyzické paměti, nebo při porušení práv při přístupu do stránky nastane přerušení – chyba stránky (page fault)
- POZOR – některé části systému nelze odložit na disk
  - obsluha přerušení, která spravuje výpadek stránky
  - data potřebná k obsluze tohoto přerušení

# Principy stránkování

## Kdy stránku zavádět do FAP? (Fetch policy)

- Stránkování při spuštění
  - Program je celý vložen do paměti při spuštění
  - velmi nákladné a zbytečné, předem nejsou známy nároky na paměť, dříve se nevyužívalo, dnes je využívána
- Stránkování či segmentace na žádost (Demand Paging/Segmentation)
  - Tzv. „líná metoda“, nedělá nic dopředu
  - Řeší problémy s dynamickou alokací proměnných
- Předstránkování (Prepaging)
  - Nahrává stránku, která bude pravděpodobně brzy použita
- Čištění (Pre-cleaning)
  - změněné rámce jsou ukládány na disk v době, kdy systém není vytížen
- Kopírovat při zápisu (copy-on-write)
  - Při tvorbě nového procesu není nutné kopírovat žádné stránky, ani kódové ani datové. U datových stránek se zruší povolení pro zápis.
  - Při modifikaci datové stránky nastane chyba, která vytvoří kopii stránky a umožní modifikace

# Líná metoda – Demand paging

- Při startu procesu zavede OS do FAP pouze tu část programu (LAP) kam se předává řízení – vstupní bod programu
  - Pak dochází k dynamickému zavádění částí LAP do FAP po stránkách „na žádost“ tj. až když je jejich obsah skutečně referencován
- Pro překlad LA → FA se využívá Tabulka stránek (PT)
  - Sada stránek procesu, které jsou ve FAP – rezidentní množina (resident set)
  - Odkaz mimo rezidentní množinu způsobuje přerušení výpadkem stránky (page fault ) a tím vznikne „žádost“
    - Proces, jemuž chybí stránka, označí OS jako pozastavený
    - OS spustí I/O operace k zavedení chybějící stránky do FAP (možná bude muset napřed uvolnit některý rámeček, viz politika nahrazování dále)
    - Během I/O přenosu běží jiné procesy; po zavedení stránky do paměti se aktualizuje tabulka stránek, „náš“ proces je označen jako připravený a počká si na CPU, aby mohl pokračovat
- Výhoda: Málo I/O operací, minimum fyzické paměti
- Nevýhoda: Na počátku běhu procesu se tak tvoří série výpadků stránek a proces se „pomalu rozbíhá“

# Princip lokality

- Odkazy na instrukce programu a data často tvoří “shluky”
- Vzniká časová lokalita a prostorová lokalita
  - Provádění programu je s výjimkou skoků a volání podprogramů sekvenční
  - Programy mají tendenci zůstat po jistou dobu v rámci nejvýše několika procedur
  - Většina iterativních výpočtů představuje malý počet často opakovaných instrukcí,
  - Často zpracovávanou strukturou je pole dat nebo posloupnost záznamů, které se nacházejí v „sousedních“ paměťových lokacích
- Lze pouze dělat odhady o částech programu/dat, která budou potřebná v nejbližší budoucnosti

# Heuristiky stránkování

## ■ Předstránkování (Pre-paging)

- Sousední stránky LAP obvykle sousedí i na sekundární paměti, a tak je jich zavádění poměrně rychlé
  - bez velkých přejezdů diskových hlaviček
- Platí princip časové lokality – proces bude pravděpodobně brzy odkazovat blízkou stránku v LAP. Zavádí se proto najednou více stránek
- Výhodné zejména při inicializaci procesu – menší počet výpadků stránek
- Nevýhoda: Mnohdy se zavádějí i nepotřebné stránky

## ■ Čištění (Pre-cleaning)

- Pokud má počítač volnou kapacitu na I/O operace, lze spustit proces kopírování změněných stránek na disk
- Výhoda: uvolnění stránky je rychlé, pouze nahrání nové stránky
- Nevýhoda: Může se jednat o zbytečnou práci, stránka se ještě může změnit

# Copy-on-write

## Kopírování až při zápisu

- velmi vhodné při vytvoření procesu – služba fork
- kód je sdílen, ten se nekopíruje ve FAP, pouze se připojí do nového virtuálního prostoru (zkopíruje se pouze odpovídající část stránkovací tabulky)
- data by měla být vlastní, měla by se vytvořit kopie dat ve FAP
  - není nutné to dělat, pokud nikdo (rodič ani potomek) nebude data měnit
  - to se dá pojistit zakázáním zápisu do stránek dat
    - při zápisu do stránky, nastane chyba stránkování (page fault)– OS zjistí, že je potřeba tuto stránku zkopírovat mezi rodičem a potomky (mohlo dojít k více voláním fork)
- složitost této metody je dána možností vytvoření více potomků se stejnými datovými stránkami, z nichž některé jsou již zkopírovány a některé sdíleny



# Stránkování – politika nahrazování

- Co dělat, pokud není volný rámeček ve FAP
  - Např. při startu nového procesu
- Politika nahrazování (Replacement Policy)
  - někdy též politika výběru oběti
- Musí se vyhledat vhodná stránka pro náhradu (tzv. oběť)
  - Kterou stránku „obětovat“ a „vyhodit“ z FAP?
  - Kritérium optimality algoritmu: minimalizace počtu (či frekvence) výpadků stránek

# Co dělá kdo?

## HW – CPU (MMU)

- 1 MMU automaticky převádí logickou adresu programu na fyzickou adresu podle tabulek stránek
- 2 když MMU nemůže převést logickou adresu na fyzickou vyvolá výjimku (přerušeni)

## SW – operační systém

- 1 při svém zavádění nastaví CPU, aby používalo stránkování (tj. zapne MMU)
  - Kvůli zpětné kompatibilitě podporují moderní procesory více typů stránkování (32-bit, 64-bit, PAE, ...). OS si rozhodne, jaký typ se použije (většinou ten nejnovější, podporovaný jak HW, tak OS)
- 2 plní obsahy tabulek stránek, aby logické adresy odpovídaly určeným fyzickým rámcům (každý proces má vlastní tabulku stránek)
- 3 řeší výjimky (výpadky stránek) – přístup k virtuálním adresám, které:
  - nejsou mapovány do fyzické paměti (buď stránka na HDD (swap), nebo typicky chyby v programu, např. dereference NULL ukazatele)
  - jsou mapovány s jinými příznaky (např. zápis do read-only stránky ⇒ buďto chyba nebo aktivace copy-on-write)

# Stránkování – výběr oběti

## ■ Určení oběti:

### ■ Politika nahrazování říká, jak řešit problémy typu:

#### ■ Kolik rámců procesu přidělit?

#### ■ Kde hledat oběti?

#### ■ Jen mezi stránkami procesu, kterému stránka vypadla nebo lze vybrat oběť i mezi stránkami patřícími ostatním procesům?

## ■ Některé stránky nelze obětovat

### ■ Některé stránky jsou dočasně „zamčené“, tj. neodložitelné

### ■ typicky V/V vyrovnávací paměti, řídicí struktury OS, ...

## ■ Je-li to třeba, musí se rámec zapsat na disk („swap out“)

### ■ Nutné to je, pokud byla stránka od svého předchozího „swap in“ modifikována. K tomu účelu je v PT příznak dirty (modified) bit, který je automaticky (hardwarově) nastavován při zápisu do stránky (rámce).

# Algoritmus FIFO

Hledáme algoritmus, který je rychlý a vede na nejmenší počet výpadků stránek

- Obětí je vždy nejstarší stránka
- FIFO – jednoduché, rychlé, ale neefektivní
- Nevýhoda – i staré stránky se používají často

číslo rámce	1	2	3	4	1	2	5	1	2	3	4	5	3
1	1	1	1	1	1	1	5	5	5	5	4	4	4
2		2	2	2	2	2	2	1	1	1	1	5	5
3			3	3	3	3	3	3	2	2	2	2	2
4				4	4	4	4	4	4	3	3	3	3

Celkem 10 výpadků

# Optimální algoritmus

- Oběť – stránka, ke které bude přistupováno (čtení či zápis) ze všech nejpozději
  - tj. po nejdelší dobu se s ní nebude pracovat
- Budoucnost však v reálném případě neznáme
  - lze jen přibližně predikovat
- Lze užít jen jako porovnávací standard pro ostatní algoritmy
  - Zpětně při analýze jiných algoritmů „známe budoucnost“

číslo rámce	1	2	3	4	1	2	5	1	2	3	4	5	3
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2		2	2	2	2	2	2	2	2	2	4	4	4
3			3	3	3	3	3	3	3	3	3	3	3
4				4	4	4	5	5	5	5	5	5	5

Celkem 6 výpadků

# Algoritmus LRU

- Predikce založená na historii
  - Předpoklad: Stránka, ke které nebylo dlouho přistupováno, nebude potřeba ani v blízké budoucnosti
- Oběť – stránka, ke které nejdéle nikdo nepřistoupil
  - LRU se považuje za nejlepší aproximaci optimálního algoritmu
  - bez věštecké křišťálové koule lze těžko udělat něco lepšího

číslo rámce	1	2	3	4	1	2	5	1	2	3	4	5	3
1	1	1	1	1	1	1	1	1	1	1	1	5	5
2		2	2	2	2	2	2	2	2	2	2	2	2
3			3	3	3	3	5	5	5	5	4	4	4
4				4	4	4	4	4	4	3	3	3	3

Celkem 8 výpadků

# LRU – implementace

## ■ Řízení časovými značkami

- Ke každé stránce (rámci) je hardwarově připojen jeden registr, do nějž se při přístupu do stránky hardwarově okopírují systémové hodiny (time stamp)
- Při hledání oběti se použije stránka s nejstarším časovým údajem
- Přesné, ale náročné jak hardwarově tak i softwarově
  - prohledávání časovacích registrů
  - každá instrukce musí modifikovat časovou značku 1–2 stránek

## ■ Zásobníková implementace

- Řešení obousměrně vázaným zásobníkem čísel referencovaných stránek
- Při použití přesune číslo stránky na vrchol zásobníku
- Při určování oběti se nemusí nic prohledávat, oběť je na dně zásobníku
- Problém:
  - Přesun na vrchol zásobníku je velmi náročný, hardwarově složitý a nepružný; softwarové řešení nepřichází v úvahu kvůli rychlosti
  - Nutno dělat při každém přístupu do paměti!

# Aproximace LRU

- Příznak přístupu (Access bit, reference bit) – a-bit
  - Spojen s každou stránkou, po „swap-in“ = 0, při přístupu ke stránce hardwarově nastavován na 1
- Algoritmus druhá šance
  - Používá a-bit, FIFO seznam zavedených stránek – tzv. mechanismus hodinové ručičky
    - Každé použití stránky nastaví a-bit
    - Každé ukázání hodinové ručičky způsobí vynulování a-bitu (stránka dostane druhou šanci)
    - Obětí se stane stránka, na niž ukáže hodinová ručička a a-bit je nulový
  - Akce ručičky závisí na hodnotě a-bitu:
    - a=0: vezmi tuto stránku jako oběť
    - a=1: vynuluj a-bit, ponechej stránku v paměti a posuň ručičku o pozici dále
  - Jednoduché jako FIFO, při výběru oběti se vynechává stránka aspoň jednou referencovaná od posledního výpadku
  - Numerické simulace – dobrá aproximaci čistého LRU



# Algoritmus druhé šance

## ■ Ukázka příkladu s algoritmem druhé šance

číslo rámce	1	2	3	4	1	2	5
1	$1_{a=1}$	$1_{a=1}$	$1_{a=1}$	$1_{a=1}$	$1_{a=1}$	$1_{a=1}$	$5_{a=1}$
2		$2_{a=1}$	$2_{a=1}$	$2_{a=1}$	$2_{a=1}$	$2_{a=1}$	$2_{a=0}$
3			$3_{a=1}$	$3_{a=1}$	$3_{a=1}$	$3_{a=1}$	$3_{a=0}$
4				$4_{a=1}$	$4_{a=1}$	$4_{a=1}$	$4_{a=0}$

2	1	5	4	5	3
$5_{a=1}$	$5_{a=1}$	$5_{a=1}$	$5_{a=1}$	$5_{a=1}$	$5_{a=0}$
$2_{a=1}$	$2_{a=0}$	$2_{a=0}$	$2_{a=0}$	$2_{a=0}$	$3_{a=1}$
$3_{a=0}$	$1_{a=1}$	$1_{a=1}$	$1_{a=1}$	$1_{a=1}$	$1_{a=1}$
$4_{a=0}$	$4_{a=0}$	$4_{a=0}$	$4_{a=1}$	$4_{a=1}$	$4_{a=0}$

Celkem 7 výpadků

# Modifikovaná druhá šance

- Algoritmus označovaný též NRU (not recently used)
  - Vedle a-bitu se používá i bit modifikace obsahu stránky (dirty bit, d-bit)
    - nastavován hardwarem při zápisu do stránky
  - Hodinová ručička maže a-bity
    - proto je možná i stránka s nastaveným d-bitem a nulovým a-bitem

d	a	Význam
0	0	stránka se vůbec nepoužila
0	1	ze stránky se pouze četlo
1	0	stránka má modifikovaný obsah, ale dlouho se k ní nepřistupovalo
1	1	stránka má modifikovaný obsah a byla i nedávno použita

- Pořadí výběru (da): 00, 01, 10, 11
- Využití d-bitu šetří nutnost zápisu modifikované stránky na disk před odstraněním z paměti

# Přidělování rámců procesům

- Pevné přidělování
  - Procesu je přidělen pevný počet rámců
    - buď zcela fixně, nebo úměrně velikosti jeho LAP
  - Podhodnocení potřebného počtu rámců  $\Rightarrow$  velká frekvence výpadků
  - Nadhodnocení  $\Rightarrow$  snížení maximálního počtu spuštěných procesů
- Prioritní přidělování
  - Procesy s vyšší prioritou dostanou větší počet rámců, aby běžely „rychleji“
  - Dojde-li k výpadku, je přidělen rámec patřící procesu s nižší prioritou
- Proměnný počet rámců přidělovaných globálně (tj. z rámců dosud patřících libovolnému procesu)
  - Snadná a klasická implementace, užíváno mnoha OS (UNIXy)
  - Nebezpečí „výprasku“ (thrashing)
    - mnoho procesů s malým počtem přidělených rámců  $\Rightarrow$  mnoho výpadků
- Proměnný počet rámců přidělovaných lokálně (tj. z rámců patřících procesu, který způsobil výpadek)
  - Metoda tzv. pracovní množiny (working sets)

# Thrashing

- Jestliže proces nemá v paměti dost stránek, dochází k výpadkům stránek velmi často
  - nízké využití CPU
  - OS „má dojem“, že může zvýšit počet běžících vláken/procesů, aby se CPU víc využilo, protože se stále se čeká na dokončení I/O operací
    - odkládání a zavádění stránek
    - Tak se dostávají do systému další procesy a situace se zhoršuje
- Thrashing – “Výprask” – počítač nedělá nic jiného než výměny stránek

# Pracovní množiny

## Model pracovní množiny procesu $P_i$ (working set) $WS_i$

- Množina stránek, kterou proces referencoval při posledních  $n$  přístupech do paměti ( $n \approx 10.000$  – tzv. okno pracovní množiny)
- Pracovní množina je aproximace prostorové lokality procesu. Jak ji ale určovat?
  - Při každém přerušení od časovače lze např. sledovat a-bity stránek procesu, nulovat je a pamatovat si jejich předchozí hodnoty. Jestliže a-bit bude nastaven, byla stránka od posledního hodinového „tiku“ referencována a patří do  $WS_i$
  - Časově náročné, může interferovat s algoritmem volby oběti stránky, avšak účelné a často používané
  - Pokud suma všech  $WS_i$  (počítaná přes všechny procesy) převýší kapacitu dostupné fyzické paměti, vzniká „výprask“ (thrashing)
    - Snadná ochrana před „výpraskem“ – např. jeden proces se pozastaví

# Četnost výpadků stránek

- Linux nepočítá pracovní množiny, ale četnost výpadků stránek
- Pro každý proces se udržuje statistika, kolik výpadků stránek nastalo v čase
- Procesy s vyšší četností výpadků dostanou více reálné paměti
- Procesy v nižší četností mohou mít méně reálné paměti
- Thrashing nastane, pokud četnost výpadků všech procesů bude růst