Operační systémy 2

Virtuální paměť

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

Motivace (1/2)

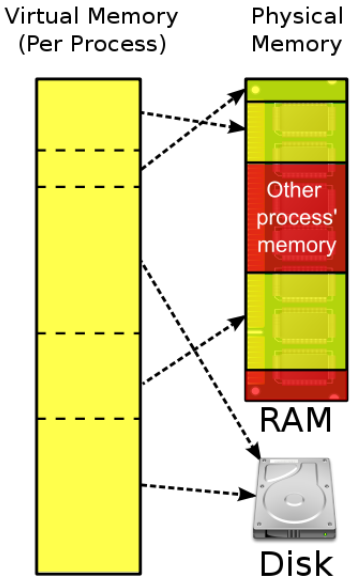
- pameť RAM je/byla relativně drahá $\Longrightarrow$ nemusí vždy dostačovat
- aktuálně používaná data (např. instrukce) musí být v RAM, nepoužívaná data nemusí (velké programy $\Longrightarrow$ nepoužívané funkce a data), princip lokality
- je vhodné rozšířit primární paměť (RAM) o sekundární (např. HDD, SSD)
- zvětšením dostupné paměti je možné zjednodušit vývoj aplikací (není potřeba se omezovat v množství použité paměti)
- sekundární paměť bývá řádově pomalejší

Odbočka: přístupové doby

L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	
Mutex lock/unlock	25 ns	
Main memory reference	100 ns	
Compress 1K bytes with Zippy	3,000 ns	
Send 1K bytes over 1 Gbps network	10,000 ns	(0.01 ms)
Read 4K randomly from SSD*	150,000 ns	(0.15 ms)
Read 1 MB sequentially from memory	250,000 ns	(0.25 ms)
Round trip within same datacenter	500,000 ns	(0.5 ms)
Read 1 MB sequentially from SSD*	1,000,000 ns	(1 ms)
Disk seek	10,000,000 ns	(10 ms)
Read 1 MB sequentially from disk	20,000,000 ns	(20 ms)
Send packet CA-Netherlands-CA	150,000,000 ns	(150 ms)

Motivace (2/2)

- k efektivní implementaci je potřeba spolupráce HW (MMU) a OS
- z pohledu procesu musí být přístup k paměti transparentní
- virtuální paměť (VM) je součástí soudobých OS (swapování)
  - Windows NT – stránkovací soubor (pagefile.sys)
  - Linux – swap partition (ale může být i soubor)
- bezpečnost dat v sekundární paměti? (např. po vypnutí počítače; možnost číst data z paměti)


Virtual Memory
(Per Process)
Physical
Memory
Other
process'
memory
RAM
Disk

Přístup do paměti (1/2)

Předpoklad

- systém si eviduje, které stránky jsou v primární paměti a které ne
- stránkovací tabulka (MMU)

Postup

- přístup na stránku, která **má** přiřazený rámec v primární paměti $\Longrightarrow$ bez změny
- přístup na stránku, která **nemá** přiřazený v primární paměti

1. přerušení – **výpadek stránky** (page fault)
2. obsluha přerušení (operační systém) načte stránku do rámce v primární paměti, aktualizuje stránkovací tabulku (je-li primární paměť plná, je potřeba nějakou jinou stránku přesunout do sekundární paměti, ,,odswapovat")
3. zopakuje se instrukce, která vyvolala výpadek stránky

Přístup do paměti (2/2)

Dílčí komplikace

- pokud je odsouvaná stránka sdílená (např. CoW), je potřeba aktualizovat všechny tabulky, kde se vyskytuje
- potřeba efektivně převádět rámce na stránky $\Longrightarrow$ invertovaná tabulka stránek

Vlastnosti stránek

- příznaky uložené ve stránkovací tabulce (nutná podpora HW)
- present/absent bit – přítomnost stránky v primární paměti (nutné k detekci výpadků stránek)
- reference bit – nastaven na 1, pokud se ze stránky četlo nebo do ní zapisovalo
- dirty bit – nastaven na 0, pokud má stránka přesnou kopii v sekundární paměti; 1 nastaveno při změně
- mohou mít přístupová práva (NX bit)

Rezervovaná stránka

- existují v adresním prostoru, ale nezapisovalo se do ní
- každá stránka je nejdřív rezervovaná
- vhodné pro velká pole, ke kterým se přistupuje postupně
- zásobník

Komitovaná stránka (Commited)

- stránka má rámec v primární nebo sekundární paměti
- musí řešit jádro
- paměť je často současně komitovaná i rezervovaná

Výměna stránek

- není-li volný rámec v primární paměti, je potřeba nějak získat volný rámec

Varianta 1

- najde se stránka, která má přesnou kopii v primární i sekundární paměti (dirty bit = 0),
- nastaví se stránce dirty bit 1 a daný rámec primární paměti se použije

Varianta 2

- vybere se ,,oběť" – stránka v primární paměti, která bude přesunuta do sekundární
- pokud má stránka nastavený dirty bit 1, překopíruje se obsah rámce do sekundární paměti

Poznámka

- některé stránky je možne zamknout, aby nebyly odswapovány (nutné pro jádro, rámce sdílené s HW)

Výběr oběti (1/3)

- hledáme stránku, která nebude v budoucnu použitá (případně bude, ale v co nejvzdálenější budoucnosti)

FIFO

- velice jednoduchý algoritmus
- stačí udržovat frontu stránek
- při načtení nové stránky je stránka zařazena na konec fronty
- pokud je potřeba uvolnit stránku, bere se první z fronty
- nevýhoda – odstraní i často užívané stránky
- Beladyho anomálie – za určitých okolností může zvětšení paměti znamenat více výpadků stránek

Least Frequently Used (LFU)

- málo používané stránky $\Longrightarrow$ nebudou potřeba
- problém se stránkami, které byly nějaký čas intenzivně využívány (např. inicializace)

Most Frequently Used (MFU)

- právě načtené stránky mají malý počet přístupů

Least Recently Used (LRU)

- jako obět je zvolena stránka, která je nejdéle nepoužívaná
- je potřeba evidovat, kdy bylo ke stránce naposledy přistoupeno
- řešení:
  1. počítadlo v procesoru inkrementované při každém přístupu a ukládané do tabulky stránek
  2. ,,zásobník'' stránek – naposledy použitá stránka se přesune na vrchol
- nutná podpora hardwaru (nebývá k dispozici)

LRU (přibližná varianta)

- každá stránka má přístupový bit *(reference bit)* nastavený na 1, pokud se ke stránce přistupovalo
- na počátku se nastaví reference bit na 0
- v případě hledání oběti je možné určit, které stránky se nepoužívaly
- varianta
  - možné mít několik přístupových bitů
  - nastavuje se nejvyšší bit
  - jednou za čas se bity posunou doprava
  - přehled o používání stránky $\Longrightarrow$ bity jako neznaménkové číslo $\Longrightarrow$ nejmenší = oběť

Výběr oběti (3/3)

Algoritmus druhé šance

- založen na FIFO
- pokud má stránka ve frontě nastavený přístupový bit, je nastaven na nula a stránka zařazena na konec fronty
- pokud nemá, je vybrána jako oběť
- lze vylepšit uvážením ještě dirty bitu
- varianta: hodinový algoritmus

Optimalizace

Buffer volných rámců

- OS si pro každý proces udržuje seznam volných rámců
- přesun oběti je možné udělat se zpožděním
- případně, pokud je počítač nevytížený, je možné ukládat stránky s dirty bitem na disk a připravit se na výpadek (nemusí být vždy dobré)

Optimalizace inicializace a práce s daty

- demand paging (stránkování na žádost)
- můžeme načíst celý proces/data do primární paměti (může být neefektivní)
- do paměti se načtou jen data (stránky), která jsou potřeba (případně související $\Longrightarrow$ sekvenční čtení; prefetch)

Problémy virtuální paměti

- globální vs. lokální alokace rámců

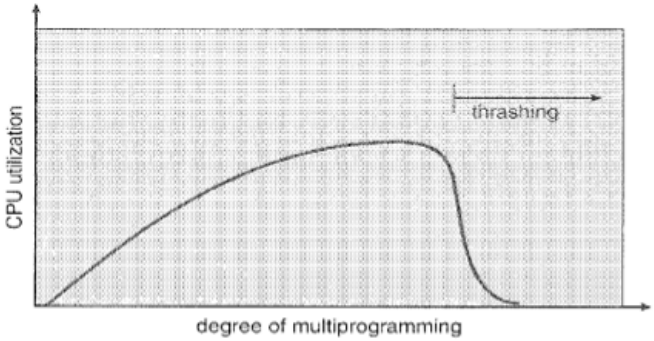
Minimální počet rámců

- každý proces potřebuje určité množství rámců (např. `movsd` potřebuje v extrémním případě 6 rámců)
- stránkovací tabulka(y) musí být opět v rámci
- přidělování rámců procesům
  - rovnoměrně (nespravedlivé vůči velikosti procesu)
  - podle velikosti adresního prostoru
  - podle priority
  - v případě výpadků stránek podle priority (globální alokace rámců)
- pokud počet rámců klesne pod nutnou mez, je potřeba celý proces odsounout z primární paměti
- hrozí thrashing (proces začne odsouvat stránky z paměti, které právě potřebuje)

Thrashing (1/2)

- systém je ve stavu, kdy odvádí spoustu práce ale bez rozumného efektu
- modelová situace:
  - pokud poklesne vytížení procesoru, systém spustí další proces
  - pokud je použitý algoritmus s globální alokací rámců, může odebírat rámce ostatním procesům
  - ostatní procesy můžou tyto rámce požadovat a brát je ostatním procesům
  - čeká se na sekundární paměť $\Longrightarrow$ sníží se využití CPU
  - procesor se pokusí spustit další proces, etc.
- lokální alokace rámců může thrashing omezit
- ideální je, aby měl proces tolik rámců kolik potřebuje

Thrashing (2/2)



Obrázek 16: Thrashing – s rostoucím počtem procesů klesá vytížení procesoru. [SGG05]

Řešení thrashingu

Pracovní množina rámců (working-set)

- vychází z principu lokality
- má-li proces tolik rámců, kolik jich v nedávné době (lokalitě) použil $\Longrightarrow$ OK
- má-li jich více $\Longrightarrow$ neefektivní využití
- má-li jich méně $\Longrightarrow$ hrozí thrashing a je lepší celý proces odsunout z primární paměti
- hrozí hladovění velkých procesů
- náročný výpočet (např. podobný algoritmu druhé šance)

Frekvence výpadků stránek

- sledujeme, jak často dochází u procesu k výpadku stránky
- je potřeba stanovit horní a dolní mez
- pokud proces je mimo tyto meze $\Longrightarrow$ přidat/ubrat rámce

Velikost stránek

- stránky mají velikost 2^n, typicky v intervalu 2^{12} – 2^{22}, i.e., 4 KB – 4 MB
- závisí na HW architektuře (může být i víc nebo míň)
- z pohledu fragmentace je vhodnější mít stránky menší
- více menších stránek zabírá místo v TLB $\Longrightarrow$ časté cache miss
- při přesunu z/do sekundární paměti může být velká stránka výhodnější (přístupová doba)
- některé systémy umožňují používat různé velikosti
- Windows NT ≤ 5.1 & Solaris velké stránky pro jádro malé pro uživatelský prostor
- Windows Vista a novejší – large pages (je potřeba povolit)
- Linux – transparentní velké stránky; hugetblfs

Invertovaná tabulka stránek (Inverted Page Table)

- někdy je potřeba namapovat rámce zpět na stránky
- prohledávat tabulky stránek je neefektivní (miliony záznamů)
- slouží k tomu převrácená tabulka stránek (tabulka pevné velikosti podle počtu rámců)
- mapuje se adresa (případně PID)

Mapování souborů a I/O do paměti

Soubory

- operace `open`, `read`, `write` mohou být pomalé (systémové volání)
- mechanismus, který je použitý pro práci se sekundární paměti, lze použít pro práci se soubory
- soubor se načítá do paměti po blocích velikosti stránky podle jednotlivých přístupů (demand paging)
- k souboru se přistupuje pomocí operací s pamětí (přiřazení, `memcpy`, ...)
- data se nemusí zapisovat okamžitě (ale až s odmapováním stránky/souboru)
- více procesů může sdílet jeden soubor $\Longrightarrow$ sdílená pamět (WinNT)
- možnost použít copy-on-write

I/O

- lze namapovat zařízení do paměti (specifické oblasti) a přistupovat k němu jako k paměti
- pohodlný přístup, rychlý přístup
- např. grafické karty

Poznámky na závěr

- jádro může požadovat souvislý blok rámců
- stránkovací tabulky jsou opět jen stránky $\Longrightarrow$ můžou být odsunuty?
- spolupráce s cache
- prezentovány základní algoritmy
- reálně se implementují složitejší metody (heuristiky)