Vysoké učení technické v Brně
Fakulta informačních technologií

Operační systémy

Neoficiální přepis demonstračního cvičení č. 2

ze dne 3. 5. 2012

Verze: 0.94
Vytvořil: Petr Jirout
Přednášel: prof. Ing. Tomáš Vojnar Ph.D.

1 O tomto dokumentu

Tento dokument vychází z druhého demonstračního cvičení z předmětu Operační systémy, v akademickém roce 2011/2012, vedeného prof. Ing. Tomášem Vojnarem Ph.D. dne 3. 5. 2012. Já sám nejsem autorem těchto příkladů ani jejich řešení a není zaručeno, že se některé z nich (či obdobné) objeví na zkoušce.

Jakékoliv připomínky ke struktuře a vzhledu dokumentu (včetně chyb) směřujte na e-mail `xjirou07@stud.fit.vutbr.cz`. Všechny ostatní připomínky směřujte na příslušné kompetentní osoby zodpovědné za tento předmět. Vysázeno v LaTeXu.

2 Příklady

2.1 TLB miss

a) Jaký je maximální počet TLB miss u instrukce o délce `4 B`, která čte `4 B` z paměti?
b) Jaký je tento počet v případě přesunu `4 B` v paměti?
c) Jak se toto maximum změní, víme-li, že se jedná o instrukci, která v kódu lineárně následuje za aktuálně prováděnou instrukcí?
Uvažujte ideální podmínky – TLB je dostatečně veliká, nedojde k přepnutí kontextu, k příchodu signálů atp.

a) Musíme vzít v úvahu, že jak data, tak instrukce mohou zasahovat do dvou různých tabulek (nemusí být zarovnány na hranice stránek). Ke každé této stránce je poté třeba provést překlad na odpovídající rámec, přičemž žádný z těchto překladů nemusí být v TLB. Tudíž pro instrukci musíme provést 2 překlady, stejně jako pro data. 2 + 2 = **až 4 TLB miss**.

b) Nyní musíme oproti variantě *a)* uvažovat i cílovou adresu, na kterou zapisujeme data a tato adresa opět nemusí být zarovnaná – další možné 2 TLB miss. 2 + 2 + 2 = **až 6 TLB miss**.

c) I když by byla v tomto případě instrukce nezarovnaná, tak jedna její část určitě bude ve stránce, kterou už máme v TLB (přeložená pro předchozí instrukci). Překlad tabulek pro data se však nijak neovlivní, pouze bude potřeba o jeden překlad méně pro instrukci. Pro variantu *a)* tak získáváme výsledek 1 + 2 = **až 3 TLB miss**. Pro variantu *b)* pak platí 1 + 2 + 2 = **až 5 TLB miss**.

2.2 Počet přístupů do paměti

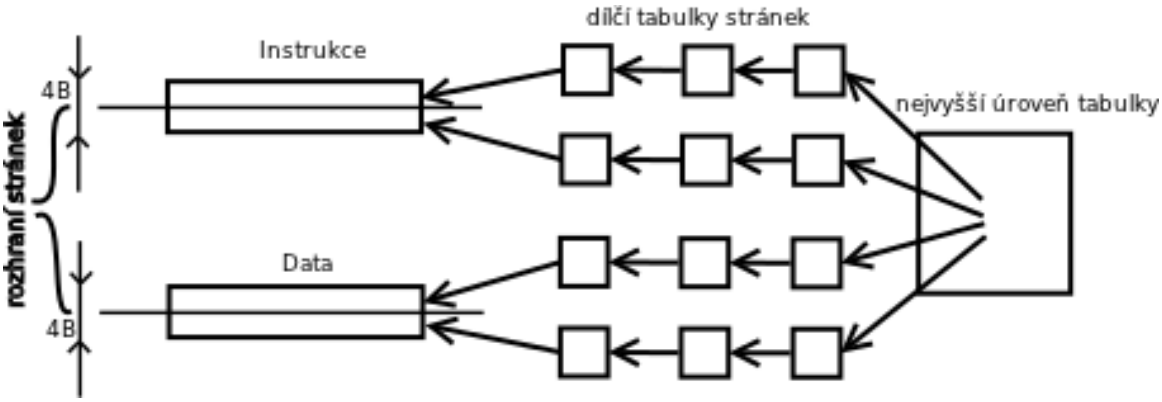
a) Jaký je maximální počet přístupů do paměti při použití čtyřúrovňové tabulky stránek u instrukce o délce `4 B` přistupující pro `4 B` do paměti (čtení nebo zápis)?
b) Jak se tento počet změní v případě přesunu těchto dat v paměti?
c) Jak se toto maximum změní, víme-li, že se jedná o instrukci, která v kódu lineárně následuje za aktuálně prováděnou instrukcí?
Uvažujte ideální podmínky – TLB je dostatečně veliká, nedojde k přepnutí kontextu, k příchodu signálů atp.

a) Pro představu si můžeme průběh překladu schematicky znázornit, viz Obrázek 1. Pro každou stránku tudíž potřebujeme 4 přístupy pro překlad. V našem případě musíme přeložit 4 stránky – 2 pro data, 2 pro instrukci. Pokud máme již přeloženou fyzickou adresu, musíme data/instrukci ještě přečíst ⇒ +1 přístup do paměti pro každou „část“ instrukce nebo dat. Celkově tedy máme $\mathbf{4 \cdot (4 + 1) = 20}$ **přístupů do paměti**.

b) Pokud budeme přesouvat data v paměti, potřebujeme opět 4 + 1 přístupů do paměti pro získání/zapsání dat nebo instrukce z daného místa v paměti (4 pro překlad, 1 pro extrakci dat). Oproti předchozímu případu je tu však odlišnost – nyní nebudeme jenom číst data, ale budeme je i přesouvat na jiné místo v paměti – tudíž musíme přeložit i cílovou adresu, což jsou další dvě „sady“ (překlad + čtení/zápis) přístupů do paměti. Z toho plyne, že celkem budeme mít $\mathbf{6 \cdot (4 + 1) = 30}$ **přístupů do paměti**.

c) Stejně jako v předchozím případě, není zde zapotřebí provádět překlad pro první část instrukce (tzn. 4 přístupy do paměti kvůli překladu), avšak musíme si tato data přečíst (+1 k celkovému počtu přístupů

do paměti). Takže pro variantu *a)* budeme potřebovat $(3 \cdot (4 + 1)) + 1 =$ **16 přístupů do paměti**. Případně můžeme využít výsledku z varianty *a)* (20) a od něj odečíst 4 přístupy do paměti kvůli překladu, které nemusíme provést ⇒ (20 - 4) = 16. Pro variantu *b)* postupujeme ekvivalentně, získávajíce tak výsledek **26 přístupů do paměti**. V tomto příkladu neuvažujeme vliv instrukční cache (pokud bychom uvažovali, ušetříme tím i čtení první části instrukce ⇒ výsledek by byl 15, resp. 25 přístupů do paměti).



Obrázek 1: Schéma překladu logické adresy na fyzickou

2.3 Počet přístupů do primární paměti

Jaký je maximální počet přístupů do primární paměti RAM při použití N-úrovňové hierarchické tabulky stránek v rámci provedení předem nenačtené instrukce o délce `4 B`, která načítá z paměti `4 B` dat, předpokládáme-li práci se stránkami o velikosti `4 KiB`. Pro načtení jedné položky jednotlivých dílčích tabulek stránek postačuje jako obvykle jeden přístup do paměti. Odvozený vztah zdůvodněte. Limity: jeden vztah a cca 3 rozvité věty (vztah bez zdůvodnění je za 0 b). (6 bodů)

- Jak instrukce, tak data mohou být ve dvou různých stránkách ⇒ 2 + 2. Pro překlad každé jednotlivé stránky musíme „projít“ N úrovněmi tabulky stránek + každou část instrukce či dat je třeba přečíst, tzn. budeme ještě *N-krát* číst z paměti. Toto nám dává výsledný vztah $\mathbf{4 \cdot (N + 1)}$ **přístupů do paměti**.

2.4 Převod čísla stránky na číslo rámce

Předpokládejte existenci struktury `struct inv_tb_item` popisující v jazyce C položku invertované tabulky stránek *kombinované s použitím hashování*. Dále nechť číselný typ `pg_num_t` popisuje čísla stránek a rámců a číselný typ `pid_t` čísla procesů. Dále předpokládejme, že je definována funkce `pg_num_t hashPg(pg_num_t pg, pid_t pid)`, která ke stránce `pg` procesu s číslem `pid` vrací odkaz (index) do uvažované tabulky stránek. Implementujte v jazyce C funkci pro převod čísla stránky na číslo rámce s prototypem

```
pg_num_t page_to_frame(struct inv_tb_item *inv_tb, pg_num_t pg, pid_t pid);
```

Předpokládejte přitom, že parametr `inv_tb` ukazuje na první položku uvažované tabulky stránek, `pid` je číslo procesu, který převod provádí, a `pg` je číslo převáděné stránky. Můžete předpokládat, že všechny číslené položky všech nepoužitých řádků tabulky mají hodnotu `NEGATIVE`, která neodpovídá žádné stránce, rámci ani procesu. Funkce v případě, kdy je možno dané číslo stránky převést na odpovídající číslo rámce, vrací příslušné číslo rámce. Pokud převod není možno provést, funkce vrací hodnotu `NEGATIVE`. Pro potřeby implementace funkce `page_to_frame` si doplňte potřebné položky struktury `inv_tb_item`. Limity: cca 6 řádků kódu pro tělo funkce bez deklarací + patřičné položky struktury `inv_tb_item`. (9 bodů)

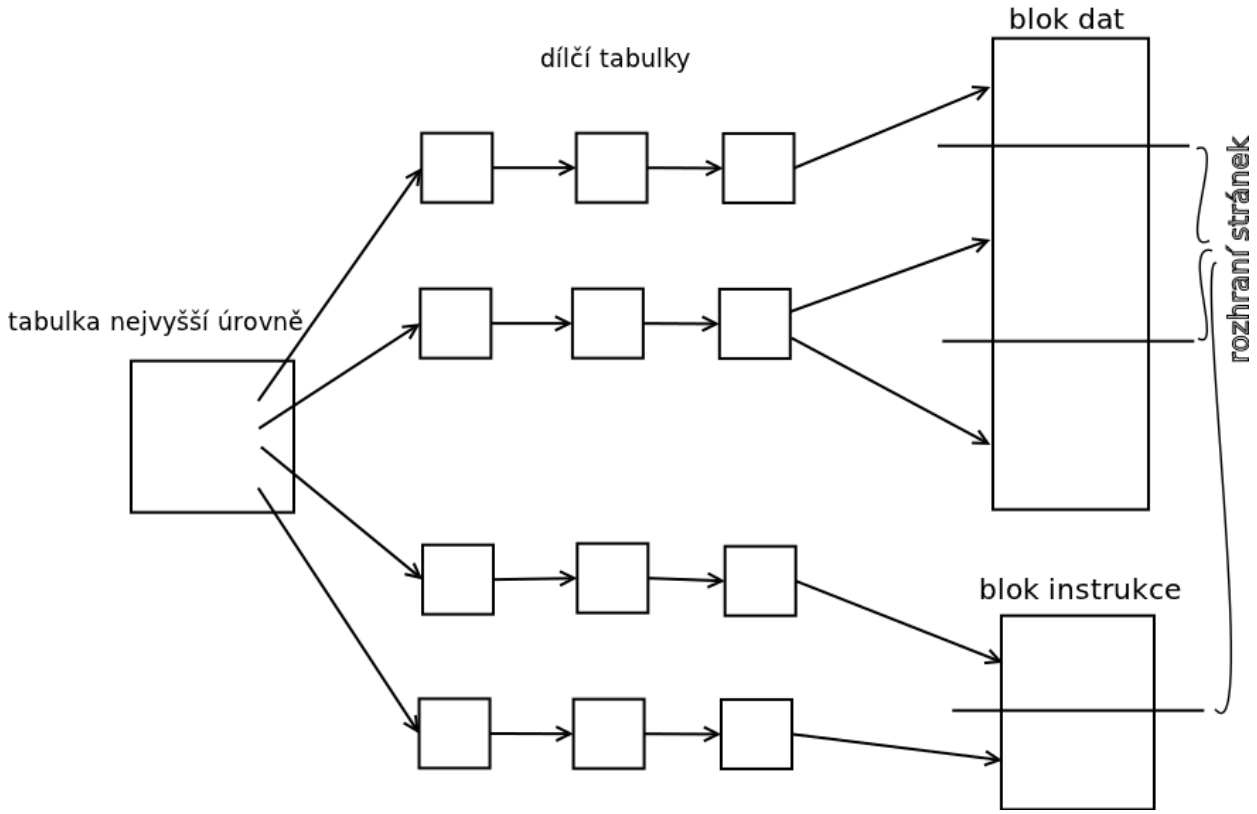
```
struct inv_tb_item {
    pid_t pid;      //cislo procesu
    pg_num_t pg;    //cislo stranky
    pg_num_t next;  //index na dalsi radek v invertovane tabulce
};

pg_num_t page_to_frame(struct inv_tb_item *inv_tb, pg_num_t pg, pid_t pid) {

    pg_num_t fr = hashPg(pg, pid); //ziskame index prvniho kandidata na nami
                                   //hledany ramec

    //dokud neodpovida pid nami hledanemu pid, nebo nesouhlasi cislo stranky
    //pokracujeme ve hledani
    while (inv_tb[fr].pid != pid || inv_tb[fr].pg != pg) {
        if (inv_tb[fr].next != NEGATIVE) //posuneme se na dalsi polozku/index
            fr = inv_tb[fr].next;
        else
            return NEGATIVE; //nemame kam pokracovat, nepodarilo se najit
                             //odpovidajici ramec
    }

    return fr; //nalezli jsme shodnou polozku, vracime tento index
}
```

Převod čísla stránky na číslo rámce

2.5 Výpadky stránek

Jaký je maximální počet výpadků stránek v systému se stránkami o velikosti `4 KiB`, při použití čtyřúrovňové tabulky stránek, u které pouze dílčí tabulka nejvyšší úrovně je chráněna proti výpadku, při provádění předem nenačtené instrukce o délce `4 B`, která přesouvá `8 KiB` z jedné adresy paměti na jinou? (6 bodů)

- Znázorníme si tento případ schematicky (stačí pouze pro zdrojovou adresu dat a instrukci, neboť pro cílovou adresu dat je situace shodná se zdrojovou adresou, tudíž i počet výpadků bude stejný) viz Obrázek 2. Uvažujme nyní pouze zdrojovou adresu dat. `8 KiB` dat může zasahovat do maximálně 3 stránek, z nichž všechny mohou vypadnout ⇒ **3 výpadky**. 3 datové stránky jdoucí za sebou, mohou být odkazovány ze 2 dílčích tabulek stránek nejnižší úrovně (a tyto tabulky opět musí být odkazovány z tabulek vyšších úrovně).
- Vypadnout mohou i všechny dílčí tabulky, kromě tabulky nejvyšší úrovně (ta je chráněna proti výpadku), takže máme dalších $\mathbf{2 \cdot 3}$ **výpadků**.
- Jak již bylo uvedeno výše, tato situace je shodná jak pro zdrojová, tak cílová data ⇒ musíme zdvojnásobit dosavadní počet možných výpadků $(3 + 6)$. Cílová i zdrojová data tak mohou dohromady při své adresaci vypadnout nejvýše $(2 \cdot (3 + 6))$krát ⇒ **18 výpadků stránek**.
- Instrukce mohou být uloženy ve 2 různých stránkách, odkazovaných ze 2 různých dílčích tabulek nejnižší úrovně. Vypadnout tak mohou 3 dílčí tabulky a 1 přístup pro samotná data, což nám dává $\mathbf{2 \cdot (3 + 1)}$ **výpadků**.
- Celkový počet výpadků pak již získáme prostým součtem. $18 + 8 =$ **26 výpadků**.



Obrázek 2: Překlad logické adresy na fyzickou pro zdroj dat a instrukce

2.6 Invertovaná tabulka

Uvažujme základní invertovanou tabulku stránek pro N rámců, předpokládáme, že pro načtení jedné položky tabulky stránek potřebujeme jeden přístup do paměti a chceme načíst data, které jsou uloženy v jedné stránce. Kolik přístupů do paměti může v nejhorším případě nastat?

- Při hledání odpovídající položky v invertované tabulce stránek procházíme postupně všechny položky této tabulky, dokud nenatrefíme na shodu (či chybu). V našem případě tak můžeme projít maximálně N položek, které se všechny nacházejí v paměti, takže potřebujeme **N přístupů do paměti**.
- Pokud již máme adresu fyzického rámce, musíme na tuto adresu ještě přistoupit a přečíst námi požadovaná data. To nám dává **1 přístup**.
- Výsledek je opět součtem všech přístupů, v našem případě tudíž **N + 1 přístupů do paměti**.

2.7 Výpadky stránek s algoritmem LRU

Předpokládejme systém s virtuální pamětí, ve které se používá lokální výměna stránek a ve kterém se při výběru odkládané stránky používá algoritmus LRU. Dále předpokládejme, že v tomto systému je spuštěn proces, který má napevno přiděleny 4 rámce, do nichž na začátku není namapována žádná stránka. Ke kolika výpadkům stránky dojde, pokud daný proces postupně přistoupí k následujícím stránkám: `1 2 3 4 4 1 5 2 2 3 5 1`. Zdůvodněte.

- Proces nejprve postupně přistoupí k prvním čtyřem stránkám, a jelikož 4 rámce přidělené procesu jsou na začátku prázdné a zároveň jsou první čtyři přistupované stránky odlišné, získáváme **4 výpadky stránek**. Stav rámců: |1|2|3|4|.
- Dále se proces pokusí přistoupit ke stránce 4, která však již je namapovaná v rámci – žádný výpadek. Jako další v pořadí proces přistupuje ke stránce 1, která je také již namapovaná – žádný výpadek.

- Nyní proces přistupuje ke stránce 5, která však zatím nikde není namapovaná, musíme jí do paměti namapovat. Nemáme však volné místo (všechny 4, procesu přidělené, rámce již byly vyčerpány), musíme tak vybrat oběť, kterou z rámců odstraníme. Jelikož používáme algoritmus LRU (*Last Recently Used*), jako oběť vybereme ten rámec, který byl použit před nejdelší dobou. To je v našem případě rámec se stránkou č. 2 a na její místo namapujeme stránku č. 5 = **1 výpadek**. Nový stav rámců: |1|5|3|4|.
- Teď chceme přistoupit ke stránce č. 2, kterou jsme však právě vyhodili. Musíme jí tedy znovu namapovat, přičemž oběť vybíráme stejně jako v předchozím případě, **1 výpadek**, Nový stav rámců: |1|5|2|4|.
- Další přistupovaná stránka č. 2 se nachází v našich rámcích – žádný výpadek.
- Na řadě je nyní stránka č. 3, která se v rámcích nenachází. Oběť bude v tomto případě stránka č. 4, **1 výpadek**. Nový stav rámců: |1|5|2|3|.
- Nyní přistupujeme ke stránce č. 5, která již v našich rámcích je – žádný výpadek. Stejné je to i v případě dalšího přístupu ke stránce č. 1 – žádný výpadek.
- Celkově tak získáváme 4 + 1 + 1 + 1 = **7 výpadků stránek**.