Vysoké učení technické v Brně
Fakulta informačních technologií

Operační systémy

Neoficiální přepis demonstračního cvičení č.1

ze dne 19.4.2012

Verze: 0.92
Vytvořil: Petr Jirout
Přednášel: prof. Ing. Tomáš Vojnar Ph.D.

1 O tomto dokumentu

Tento dokument vychází z prvního demonstračního cvičení z předmětu Operační systémy, v akademickém roce 2011/2012, vedeného prof. Ing. Tomášem Vojnarem Ph.D. dne 19.4.2012. 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 Volání funkce `open()`

Jakými typickými kroky prochází systém při volání `open()` na zatím neotevřený soubor, u něhož předpokládáme, že jde otevřít? (5 – 6 bodů)

1. Vyhodnotí zadanou cestu k souboru, ověří oprávnění ke spouštění a přístupová práva, načte *i–uzel* souboru.
2. Vytvoří novou položku v tabulce *v–uzlů*, do níž vloží načtený *i–uzel* spolu s dalšími údaji, zejmá s počtem odkazů na *v–uzel* (hodnota 1).
3. Vytvoří nový záznam v tabulce otevření souborů, naplní ho odkazem na *v–uzel*, nastaví v položce patřičný režim otevření, polohu v souboru (na začátek ⇒ 0) a počet odkazů na položku (= 1).
4. Alokuje první volnou položku v tabulce popisovačů, a naplní jí odkazem na položku v tabulce otevření souborů.
5. Vrátí příslušný index do tabulky popisovačů.

2.2 Počet čtení bloků z disku

Jaký je maximální počet čtení bloků z disku při provedení následujících operací?
`h = open("/dir/symlink", O_RDONLY); read(h, buf, 10); read(h, buf, 10);`
Předpokládejte přitom, že adresáře jsou kratší než 1 blok, `dir` je adresář, `symlink` je symbolický odkaz na soubor `/dir/file`, `file` je klasický soubor delší než `20B`, žádný blok není na počátku ve vyrovnávací paměti, je ale dostatek paměti a používají se všechny běžně používané vyrovnávací paměti, nedochází k interferenci s dalšími procesy, přepnutí kontextu, příchodu signálů apod. Limity: 1 číslo (bez zdůvodnění za 0b) a cca 6 mírně rozvitých vět. (6 bodů)

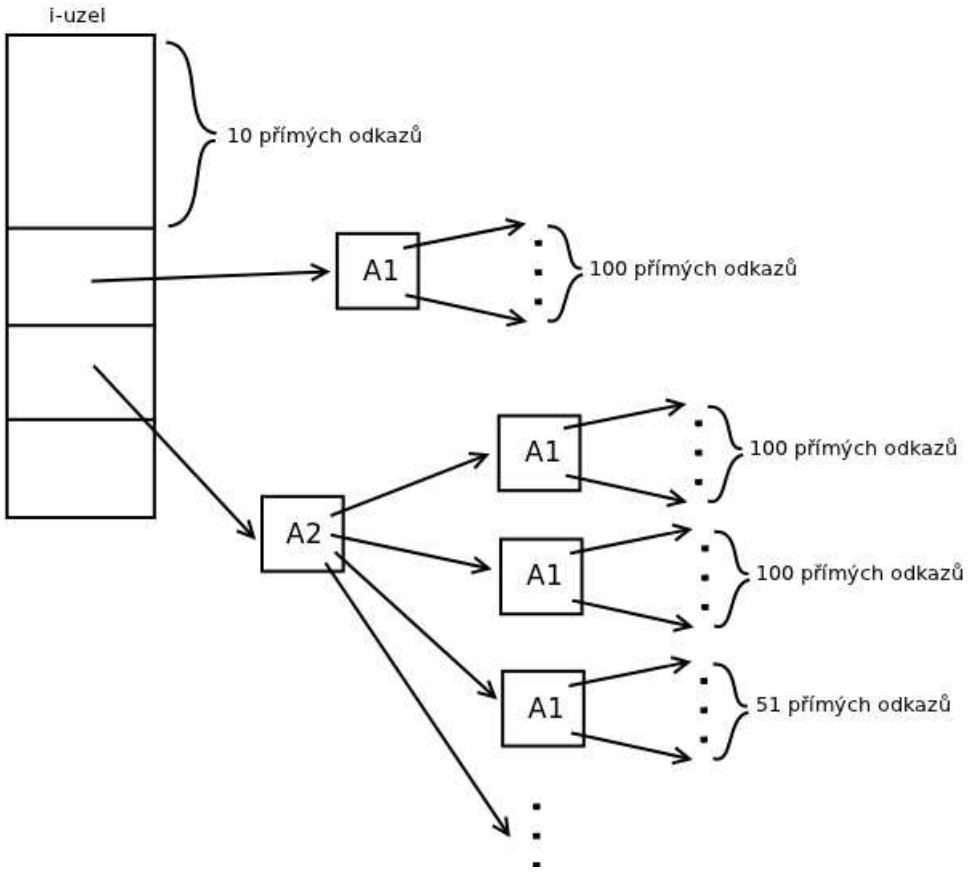
- Načte se blok s *i–uzlem* kořenového adresáře `"/"` a blok s obsahem kořenového adresáře `"/"` – **2 čtení**.
- Načte se blok s *i–uzlem* adresáře `"/dir"` a blok s obsahem adresáře "/dir" – **2 čtení**.
- Načte se blok s *i–uzlem* `"/dir/symlink"` (protože je cesta `"/dir/file"` krátká, tak bude uložena přímo v *i–uzlu*) – **1 čtení**.
- Překlad "/dir" na *i–uzel* je ve vyrovnávací paměti, stejně jako obsah adresáře "/**dir**". – **0 čtení**
- Načte se blok s *i–uzlem* `"/dir/file"` a první blok obsahu `"/dir/file"` (jako důsledek 1. operace `read()`). – **2 čtení**
- Při druhé operaci `read()` se data převezmou z vyrovnávacích pamětí (kde se nacházejí jako důsledek první operace `read()`) – **0 čtení**

Výsledek: 2 + 2 + 1 + 0 + 2 + 0 = **7 čtení**.

2.3 Počet alokačních bloků

Uvažujte velikost alokačního bloku `1000B (1 kB)`, velikost odkazu na blok pak `10B` (uvedené velikosti jsou zvoleny pro snazší výpočet, na principu nic nemění). Kolik alokačních bloků (bez *i–uzlu*) bude na disku s klasickým Unixovým FS zabírat soubor o velikosti `360,5 kB`? Zdůvodněte. Doporučuji se při vysvětlení opřít o schématický obrázek, ale ten musí být doprovázen vysvětlujícím textem. Limity: 1 číslo (bez zdůvodnění za 0b), 1 obrázek a cca 3 rozvité věty. (6 bodů)

- `360,5 kB` dat vyžaduje 361 bloků (nejmenší alokovatelná jednotka je alokační blok ⇒ nelze alokovat `0,5 kB`) – **361 bloků bude třeba pokrýt**.
- Z *i–uzlu* lze přímo odkazovat (blok přímých odkazů) na 10 bloků ⇒ zbývajících 351 bloků bude odkazováno nepřímo – **0 bloků odkazů + 10 datových**, zbývá pokrýt 351 dat. bloků.
- Použijeme 1 blok nepřímých odkazů 1. úrovně, který bude odkazovaný přímo z *i–uzlu* a který je schopný přímo odkazovat na 100 datových bloků (velikost alokačního bloku děleno velikost odkazu na blok) – **1 blok odkazů + 100 datových**, zbývá pokrýt 251 dat. bloků.
- Použijeme 1 blok nepřímých odkazů 2. úrovně, který bude odkazovaný přímo z *i–uzlu*. Z tohoto bloku lze odkazovat až na 100 bloků nepřímých odkazů 1. úrovně, které jsou schopny dohromady přímo odkazovat až na $100 \cdot 100$ datových bloků, nám však budou postačovat 3 bloky nepřímých odkazů 1. úrovně (neboť nám již zbývá pokrýt jen 251 datových bloků) – **1 + 3 bloků odkazů, 251 datových**, zbývá pokrýt 0 dat. bloků.



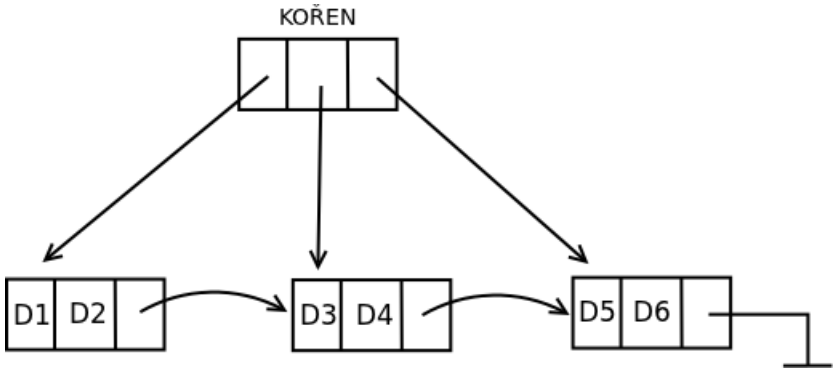
Obrázek 1: Zjednodušené schéma našeho příkladu

Výsledek: 361 datových + 1 + (1 + 3) = **366 bloků**.

2.4 Počet uzlů B+ stromu

Mějme klasický *B+ strom*, v jehož uzlech můžeme mít maximálně 3 odkazy na podřízené (či následné) uzly nebo na datové bloky souboru. Kolik bude mít tento strom uzlů, když popisuje soubor o 6 blocích? Pečlivě zdůvodněte (číslo bez zdůvodnění za 0b). K jakým jiným účelům (uveďte alespoň dva) mimo popisu rozložení dat souborů na disku se v souborových systémech používají B+ stromy, resp. jejich různé varianty? Limity: cca 3 rozvité věty/souvětí. (6 bodů)

- V listech jsou 3 odkazy, ale 1 odkaz je zapotřebí pro provázání listů do seznamu. Zbývají nám tedy 2 odkazy na data ⇒ na data je potřeba 3 listů $(6/2 = 3)$ – **3 listy**
- Na všechny listy klasického *B+ stromu* je však ještě třeba odkazovat z jednoho kořene. My máme nyní 3 listy potřebná na data, která lze pokrýt jedním listem, který se tím pádem stane listem kořenovým – **1 list**.



Obrázek 2: Zjednodušené schéma našeho příkladu

Výsledek: 3 + 1 = **4 listy**.
Další využití B+ stromů (resp. jejich varianty): Správa volného místa, obsahy jednotlivých adresářů.

2.5 Analýza programu

Jaký součet vytiskne níže uvedený program, předpokládáme-li, že je spuštěn standardním způsobem z příkazové řádky a nenastane chyba při vytváření ani použití souboru `/tmp/x`? Výsledek pečlivě zdůvodněte. Limity: 1 číslo (bez vysvětlení 0b) a cca 5 rozvitých vět. (5 bodů)

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

int main(void) {
    int fd,sum,i;
    char a[] = { 0, 1 };
    char b[] = { 2, 3 };
    char c[] = { 28, 28, 28, 28, 28, 28 };

    //Vytvarime soubor s pravy vlastniky pro cteni i zapis a
    //otevirame ho pro zapis. Existuje-li jiz soubor, je zkracen
    //na prazdny.
    fd = open("/tmp/x",O_WRONLY|O_CREAT|O_TRUNC,S_IWUSR|S_IRUSR);
    write(fd,a,2); //do /tmp/x zapise 2B /tmp/x: 0,1
    lseek(fd,1,SEEK_CUR); //posune se o 1B za aktualni pozici (stane se z nej
                          //ridky soubor), /tmp/x: 0,1,_,
    write(fd,b,2);  //zapise 2B, /tmp/x: 0,1,_,2,3
    close(fd);
    sum = fd = open("/tmp/x",O_RDONLY); //do sum se vlozi 3, protoze
                                        //popisovace 0,1,2 jsou jiz obsazeny
                                        //(stdin/stdout/stderr), cteci hlava
                                        //fd bude ukazovat na 1.prvek souboru
    sum += read(fd,c,6);//nacte 6B do pole c, c[]: 0,1,0,2,3,28 (na misto
                        //nezapsane polozky se doplnuje nula) a pocet nactenych
                        //polozek pricte k sum (+5, nebot mame ridky soubor)
    close(fd);
    for (i=0; i<6; i++)  //pricteme obsah pole c[] k sum (kde je 3+5)
        sum += c[i];      //3 + 5 + 0 + 1 + 0 + 2 + 3 + 28 = 42

    printf("The sum is %d\n",sum);

    return 0;
}
```

Program

Výsledek = **The sum is 42**.

2.6 Semafory

Deklarujte strukturu reprezentující monitor (bez syntaktické podpory – tzn. uživatel musí explicitně volat přímo dané funkce) s jednou čekací podmínkou, operací `signal()` a předností procesů, které zavolají `signal()` při zpětném vstupu do monitoru, oproti procesům do monitoru nově vstupujícím.
Pozn.: Tento příklad lehce přesahuje náročnost příkladů u zkoušky – to však neznamená, že jeho zjednodušená varianta se u zkoušky nemůže objevit. Při odpovídání na tuto otázku je žádoucí uvést příklady v pseudokódu, či „C-like“ kódu. Rozhodně nepište eseje!

```
struct sMonitor{
    semaphore mutex; //strazi vstup do monitoru
    semaphore cond;  //cekaci semafor
    semaphore prio;  //semafor priority
    int condCount, prioCount;   //citace, kolik je cekajicich/prioritnich
};

void monitorInit(struct sMonitor * m){
    init(m->mutex, 1); //vstup do monitoru je odemceny, nekdo muze vstoupit
    init(m->cond, 0);  //oba vnitrni semafory jsou zamknute
    init(m->prio, 0);
    m->condCount = m->prioCount = 0;
}

void monitorEnter(struct sMonitor * m){
    lock(m->mutex); //zamyka pristup do monitoru, vsichni ostatni musi cekat
}

void monitorExit(struct sMonitor * m){ //opoustim monitor
    if (m->prioCount > 0) //nekdo ceka na prioritnim semaforu,
        unlock(m->prio);  //uvolnime prioritni proces
    else                  //nikdo prioritni neceka
        unlock(m->mutex); //odemykam hlavni semafor
}

void condWait(struct sMonitor * m){
    m->condCount++; //budeme cekat na podmince
    if (m->prioCount > 0) //nekdo ceka na prioritnim semaforu,
        unlock(m->prio);  //tak mu umoznime pokracovat.
    else                  //nikdo prioritni neceka
        unlock (m->mutex);//otevreme vstup do monitoru
    lock(m->cond);  //cekame, nez nas probudi
    m->condCount--; //byli jsme probuzeni, konec cekani na podmince
}

void condSignal(struct sMonitor * m){ //proces bezi uvnitr monitoru a chce
                                      //nekoho uvolnit
    if (m->condCount > 0){ //na podmince nekdo ceka, mame komu signalizovat
        m->prioCount++;
        unlock(m->cond); //odemkneme mu podminkovy semafor, tim uvolnim
                         //signalizovany proces
        lock(m->prio);   //zamkneme prioritni semafor a cekame na prioritnim
                         //semaforu, dokud nas nekdo nevysbodi
        m->prioCount--;
    }
}
```

Monitor implementovaný pomocí semaforů

2.7 Počet pomocných adresovacích bloků

Předpokládejte velikost datového bloku `1 kB` a velikost odkazu `4B`. Jaký je počet pomocných adresovacích bloků pro soubor maximální velikosti v klasickém Unix FS? (Uvažujte maximální teoretickou velikost v rámci FS). Pro představu nám může posloužit obrázek 1 z příkladu 2.3.

- Maximální velikost souboru = co největší možný počet datových bloků. To znamená, že musíme využít všechny dostupné adresovací bloky.
- Nyní nás nezajímá samotný počet datových bloků, tudíž můžeme ignorovat 10 přímých odkazů na datové bloky dostupné z *i–uzlu*.
- Nejprve využijeme 1 blok nepřímých odkazů první úrovně – **1 blok**.
- Poté postoupíme na blok nepřímých odkazů druhé úrovně – tzn., že jeden blok druhé úrovně může adresovat 250 bloků první úrovně (velikost dat. bloku / velikost odkazu). – **1 + 250 bloků**.
- Zbývá už jenom blok nepřímých odkazů třetí úrovně – obdobně jako předtím, jeden blok třetí úrovně může adresovat až 250 bloků druhé úrovně, z nichž každý může adresovat až 250 bloků první úrovně – **1 + 250 + 250 · 250 bloků**.

Výsledek: $1 + (1 + 250) + (1 + 250 + 250 \cdot 250)$ = **63003 pomocných adresovacích bloků**.

2.8 Definice pojmů

Definujte následující pojmy: *diskový sektor, alokační blok, extent.*

- **Diskový sektor** je nejmenší jednotka, kterou disk umožňuje přečíst/zapsat. Typická velikost je `512B` u novějších disků i `4096B` (tyto disky však umožňují i emulaci `512B` velkých sektorů).
- **Alokační blok** je nejmenší jednotka diskového prostoru, se kterou může OS umožňuje pracovat uživatelům. Jeho velikost je 2^n diskových sektorů, nejčastěji `4096B`.
- **Extent** je způsob indexace alokovaného prostoru, který má proměnnou velikost a nachází se na disku v posloupnostech fyzicky i logicky za sebou.

Definujte pojem *uváznutí* při přístupu ke sdíleným zdrojům s výlučným přístupem.

- Při **uváznutí** máme **skupinu** procesů, z nichž **každý** je **pozastaven** nebo **čeká** na zdroj s **výlučným** přístupem, přičemž tento zdroj je vlastněný někým ze **stejné** skupiny.