ZOS opakování 2014

Verze 2. června 2014

L. Pešička

Poznámky

- Přečíst i ostatní soubory s opakováním v Courseware ZOS-Prednasky

Důraz na porozumění

- Jak jednotlivé moduly OS spoluprací na dosažení celku

- Znát nejen, jak nějaký algoritmus funguje, ale k čemu jej můžeme využít

Přístupová práva

- **Klasická unixová**
  (vlastník, skupina, ostatní) -> (r, w, x, s, t)
  rozdíl mezi právy na soubor, adresář
  chmod 777 vse.txt
  chmod a+w zapis.txt

- **ACL (Access Control List)**
  jsou rozšířením původních klasických práv
  může být uvedeno více lidí, více skupin

Poznámka

Klasická unixová práva (vlastník, skupina, ostatní) se nenazývají ACL, protože mají:

Jen JEDNOHO vlastníka

Jen JEDNU skupinu

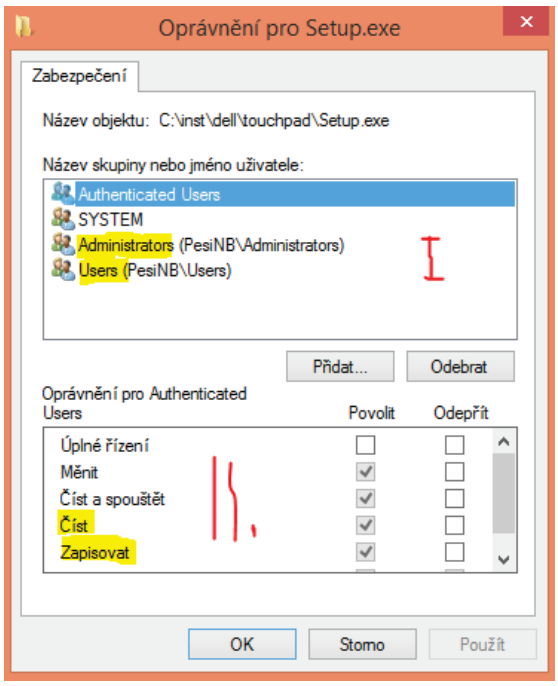
Zatímco **ACL** umožnují práva nastavit VÍCE lidem, VÍCE skupinám

Poznámka 2

- Na linuxu jsme si zkoušeli nastavení klasických unixových práv

- Nastavení ACL si můžete vyzkoušet třeba na Windows na souborovém systému **NTFS**, nebo na distribouvaném fs **AFS**, který se používá ve škole

ACL ve Windows



Průzkumník – vybereme soubor – pravá myš – vlastnosti – zabezpečení

I. Pro koho práva
   více uživatelů, více skupin
   všimněte si ikony u názvů:
   jeden panáček – uživatel
   více lidí - skupina

II. Jaká práva dané zvolené entitě
   číst, zapisovat, spouštět, …

ACL

- Ke každému souboru, adresáři přísluší tabulka popisující ACLka

  soubor **dokument.txt** :

Uživatel (0) nebo skupina (1)	identifikace	Práva
0	pepa	R, W
0	tomas	R
1	students	R
1	projektALFA	R, W

ACL poznámka

- Ve skutečnosti v tabulce identifikace není jméno uživatele či skupiny, ale dlouhý řetězec, který jednoznačně identifikuje uživatele či skupinu
- Liší se v různých systémech
  může to být **UID** (user ID), ale často některé systémy generují další identifikační řetězec spojený s účtem při jeho založení

Otázka k přemýšlení

- Klasický i-uzel obsahuje atributy souboru, ale neobsahuje ACL.
  Jak byste ACL do systému využívajícího i-uzly implementovali?

Další otázka k i-uzlům

- Víme že jsou zde přímé ukazatele, ukazatele nepřímé 1., 2., 3. úrovně
- Jak ale vlastně poznáme, kolik ukazatelů pro konkrétní soubor využijeme?

Odpověď

Jedním z nejdůležitějších atributů souboru je
VELIKOST !!!!!!!!!!!!
Udává přesnout velikost souborů v bytech

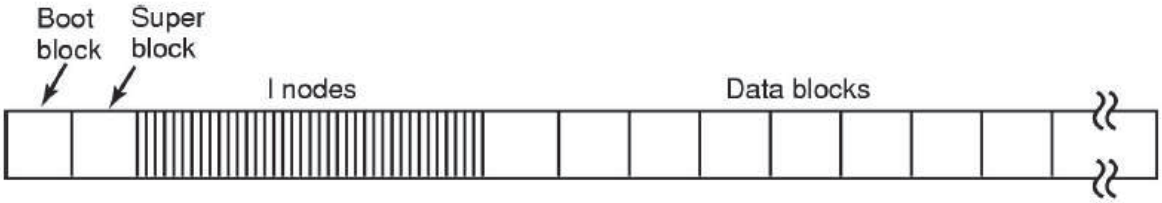
Z něj poznáme, kolik ukazatelů použít
stejně tak kolik dat z daného bloku využijeme

Dalšími atributy jsou třeba
časy (vytvoření, modifikace, …)
klasická unixová práva (vlastník, skupina, ostatní)
počet odkazů na soubor (kolik hardlinků máme)

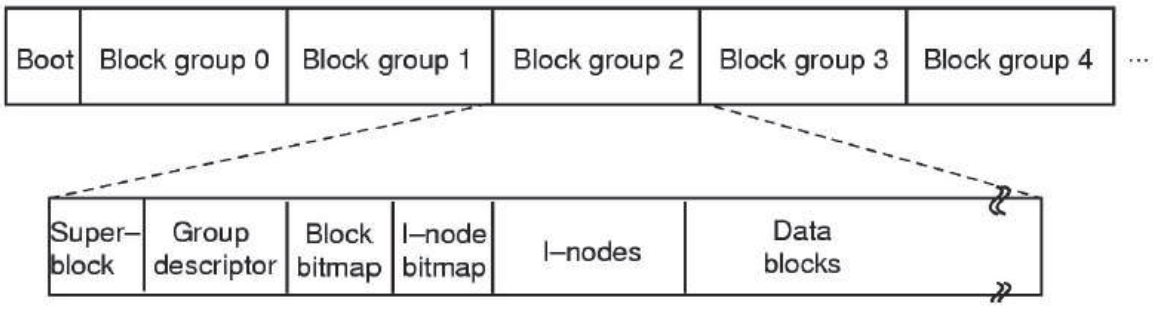
Disk

- Disk obsahuje diskové bloky
- Diskové bloky mohou obsahovat **data** nebo **metadata**
- **Data –** obsah konkrétního souboru, např. text vaší bakalářské práce ☺
- **Metadata –** např. seznam ukazatelů na datové bloky, tedy neobsahují uživatelská data, ale data která pomáhají udržet strukturu souborového systému

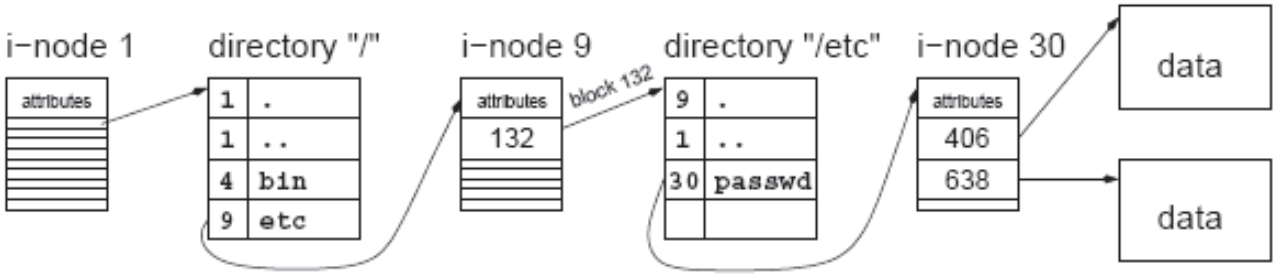
i-uzly: starší, novější



nahoře – starší , dole – novější s využitím block group



adresář, nalezení souboru



Různé souborové systémy

- **NTFS**
  - Používají Microsoft Windows XP, 7, 8, …
  - Obsahuje ACL (access control list)
- **FAT**
  - Jednoduchý fs kompatibilní mezi více systémy
  - Neobsahuje ACL ani základní linuxová práva
  - Různé verze: FAT32,
    exFAT (SDXC karty), který už může ACL podporovat
- ext3, ext4, xfs, …

Žurnál

- Žurnálovat se mohou data, ale často jen metadata – tj. nerozpadne se nám struktura souborů, např. nezmizí adresář, ale o obsah konkrétního otevřeného souboru můžeme přijít

Důvod: náročnost

Žurnál

1. Zapisuji do žurnálu
2. Je-li žurnál kompletní, přidám značku žurnál kompletní
3. Začnu zapisovat do „správných“ datových bloků
4. Pokud jsem zapsal vše, smažeme žurnál

Jak systém pozná, že se něco pokazilo?

- Při připojení filesystému na něj zapíše značku „připojeno“
- Při korektním odpojení zapíše značku „odpojeno“

- Když chce systém připojit fs, podívá se na značku. Pokud byla připojeno – něco není v pořádku, je třeba provést kontrolu

Výhoda žurnálu

- Není-li žurnál, kontrola trvá velmi dlouhou.. Projet datové bloky (v seznamech volný-obsazený), projet položky adresáře..
- V případě žurnálu se podívám:
  - Prázdný žurnál: není co dělat, OK
  - Žurnál se značkou žurnál kompletní: přepíšu do správných datových bloků
  - Žurnál bez značky žurnál kompletní: jen smažu žurnál, do ostrých dat nic nezapisuji

Disk

- **Disk** (/dev/sda)
  - Rotační (plotna, stopa, sektor)
  - SSD disky (chipy, velká rychlost, omezený počet zápisů)
- Schéma rozdělení disku
  - MBR (nejčastější pro klasická PC,NB s Linux-Windows) – 4 primární oblasti, lze logické
  - GPT (Apple OS X počítače) – 16 oblastí

Partitions

- Disk (/dev/sda) je rozdělený na oblasti
  Způsob rozdělení je **MBR**, GPT, …
- Každá oblast může mít jiný filesystém, např. ntf3, ext3, atd..
- Na disku můžeme mít více operačních systémů
- 1. oblast - /dev/sda1
- 2. oblast - /dev/sda2
- 1. oblast druhého pevného disku - /dev/sdb1

Otázka

- Jak Linux ví, které další disky má při spuštění systému připojit? Kde to zjistí?

Odpověď

- Při startu připojí kořen (**/**)
- Podívá se do **/etc/fstab**
  zde zjistí, které další svazky má přimountovat

- Např. uživatelé **/home** můžou ležet na jiné partition stejného disku či na úplně jiném disku

Otázka

- Proč může být výhodou, aby **/home** byl někde jinde?

Odpověď

- Uživatelé jsou nevyzpytatelní a ve své neskonalé moudrosti mohou zaplnit celý /home adresář.

- Vlastního systému se to ale nedotkne, neboť leží na jiné partitione, takže z jeho volného místa se „neukrajuje“

Zdá se nám jeden disk málo spolehlivý?

- Řešením je RAID 1, 5, 6, …
- RAID může být HW (karta v serveru) nebo SW

- Kolik disků potřebuji
- Kolik disků můžu ztratit
- Co je to degradovaný režim
- Co je to hot spare disk
- Co je to hot plug disk
- Co je to RAID 0 – řetězení, prokládání
- Pojem JBOD (just buch of disks)

Jak funguje program

- Programátor Tomáš napíše zdrojový kód programu v céčeku: zdrojak.c
- Programátor přeloží zdroják, vznikne program uložený někde na disku: program1

Co platí?

- Program je soubor na disku, obsahuje strojové instrukce, kterým CPU rozumí (umí interpretovat)
- Program má nastavené právo spustit

Spuštění programu

- Uživateli Pepa běží terminál a v něm zadá příkaz pro spuštění programu ./program1

Co se stane?

Z programu uloženého na disku v souboru se stane proces (bude zaveden do paměti a spuštěn). OS vytvoří záznam v tabulce procesů --- PCB, proces dostane PID a poběží s právy uživatele UID Pepy

Proces v uživatelském režimu

- Dokud proces jen počítá v uživatelském režimu, např. c = faktorial(10)
  OS hlídá jen, zda proces neběží moc dlouho (neuplynulo mu časové kvantum)

- Pokud mu časové kvantum vypršelo, OS uloží jeho kontext do příslušného PCB v tabulce procesů a pustí jiný proces
- Až nadejde čas, z PCB obnoví kontext našeho procesu a ten pokračuje, aniž by si všiml, že byl jeho běh přerušen

Proces může chtít službu po operačním systému

- Buď přímo po OS, nebo po knihovní funkci, která následně zavolá službu OS
- Systémové volání

Příklad:

- Vytvoř nový proces – fork()
- Otevři soubor – open() (fopen – knihovní fce)
- Vytiskni ahoj – printf(„Ahoj“);
  zde vyvolá knihovní funkci, která následně požádá o systémové volání operační systém

K čemu jsou systémová volání

- Dát věcem řád
  - v uživatelském režimu není přímý přístup k HW, tedy nemůžu sám jako proces manipulovat s diskem – mohl bych něco poškodit, nebo se dostat k něčemu, k čemu mi nebyl povolený přístup
- Usnadnění
  - I kdybych měl přímý přístup a mohl rozsvěcovat jednotlivé pixely obrazovky, než bych z bodů sestavil nápis Ahoj, tak bych se nadřel… OS nám výrazně pomůže

Analogie s pizerií

- Mám hlad, chci něco jíst
- Nemám přímý přístup do kuchyně ke sporáku, mouce, zelenině, masu

- Podívám se do jídelníčku, najdu číslo služby kterou chci (jakou pizzu)
- Svůj požadavek řeknu servírce a ta se o jeho splnění postará

Analogie s pizerií

- Není žádný přímý kontakt se sporákem, masem, kuchařem…

- Než mi služba bude poskytnuta, může dojít k ověření, zda mám **oprávnění** pro požadování této služby (servírka se zeptá: Máš peníze?)

Systémové volání

- LD EAX, číslo služby // služba, kterou chci
- Do ostatních registrů další parametry
  (např. kde leží řetězec názvu souboru, který chci otevřít, jestli chci jen pro čtení nebo i zápis atd)
- INT 0x80 // SW přerušení, vyvolání služby OS

Jako uživatel Pepa jsem spustil program1, který chce otevřít soubor ahoj.txt na disku

Příklad reálný – Linux - getpid.c

```
int pid;

int main() {

    __asm__(
          "movl $20, %eax  \n"     /* getpid system call */
          "int  $0x80      \n"        /* syscall */
          "movl %eax, pid  \n"      /* get result */
          );
  printf("Test volani systemove sluzby...\nPID: %d\n", pid);


  return 0;
}
```

- do registru EAX dáme číslo služby 20
- systémové volání přes int 0x80
- v registru EAX máme návratovou hodnotu pro naši službu (getpid)

Co se stane po INT 0x80?

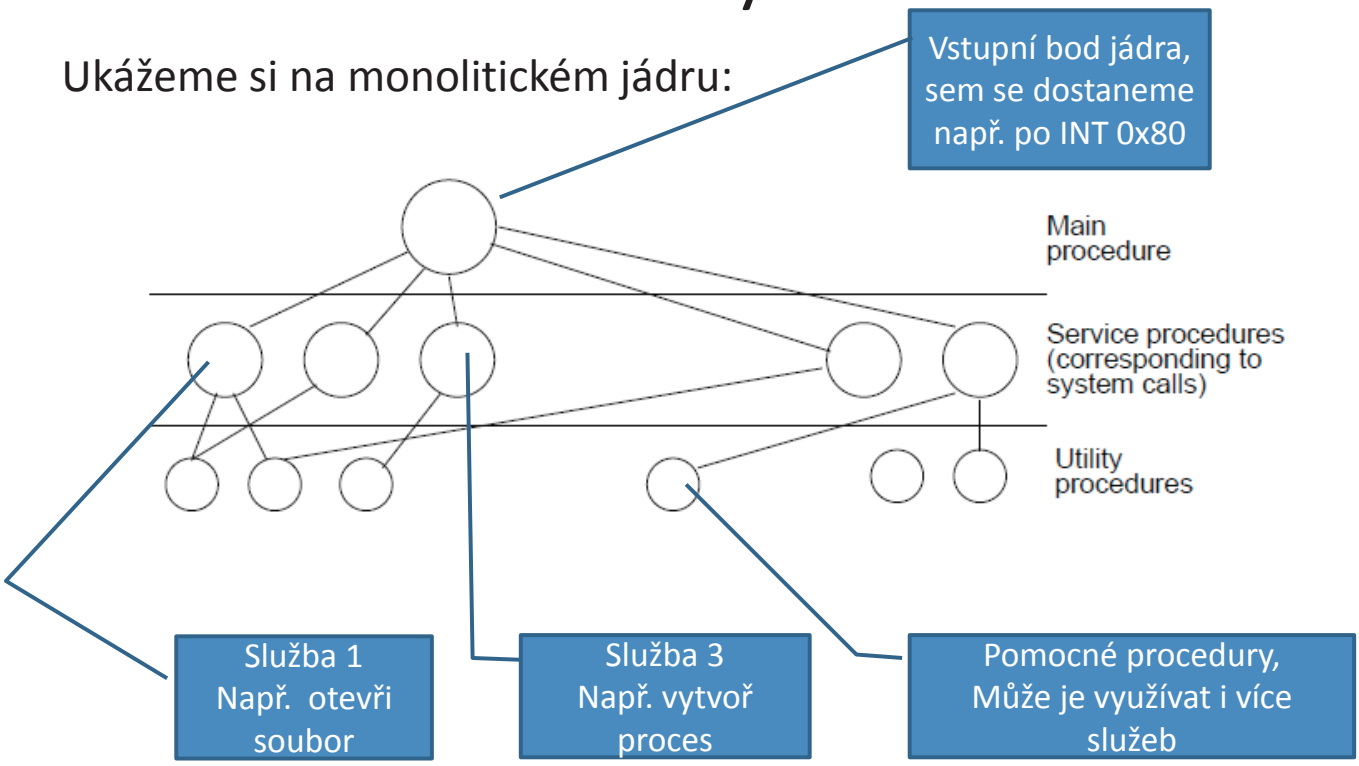
- Podívám se do vektoru přerušení na index 0x80, tj. 128 dekadicky
- Na tomto indexu leží adresa vstupního bodu OS, který následně dle hodnoty v registru EAX rozhodne, jaké volání bude provedeno
- Při vyvolání SW přerušení dojde k přepnutí do privilegovaného režimu (příznak v registru CPU), návratová adresa (kde pokračovat – CS:IP) se uloží na zásobník

Zpracování systémového volání

- Vstupní bod OS (1. vrstva monolitu) dle hodnoty EAX registru najde příslušný podprogram
- Podprogram (2. vrstva monolitu – systémové služby) se vykoná a může k tomu využít další procedury (3. vrstva)

Jak jádro implementuje jednotlivé služby?

Ukážeme si na monolitickém jádru:



Kontrola oprávnění

- Pokud potřebujeme otevřít soubor ahoj.txt pro čtení a zápis, jádro zkontroluje, zda daný soubor existuje a pokud ano, zda k němu máme práva (tj. zda je uživatel, který program spustil – Pepa – uvedený v ACL k danému souboru ahoj.txt)
- Když vše ok, jádro provede vše potřebné a vrátíme se ze systémového volání a dostaneme „ukazovátko na soubor“

Poznámky

- Systém za nás provedl akci, kterou jsme sami udělat nemohli
- Systém přitom ověřil, zda máme právo danou operaci provést – např. otevřít soubor ahoj.txt dle přístupových práv k tomuto souboru

Virtuální paměť

- Co umí dnešní CPU?
  - Segmentace se stránkováním
- Můžeme něco z toho vypnout?
  - Stránkování
- Co když nechci používat segmenty?
  - Roztáhnu je všechny od 0 .. MAX (mohou se překrývat)

K čemu slouží swap?

- Součást virtuální paměti
- Není-li místo ve fyzické paměti (RAM), začneme navíc využívat prostor na disku (SWAP)
- Nevýhodou je ohromné zpomalení systému
- Uživatel velmi rychle pozná, kdy už systému nestačí fyzická paměť RAM, ale začne pravidelně swapovat
- Co s tím? Dokoupit fyzickou paměť…

Jak může být swap realizován?

- Soubor na disku
  - Používají Windows
- Disková partition
  - Nejčastější u Linuxu
  - Výhodou je rychlost, nemusí se zdržovat pravidly konkrétního filesystému na dané partition, řeší si po svém

Co když dojde fyzická paměť i swap?

- Další alokační požadavky na pamět už nelze uspokojit

Přesun dat mezi swapem a fyzickou pamětí

- Kdo zajišťujě přesun ze swapu do fyzické paměti? DMA
- Jaký démon se stará v Linuxu o to, aby stále bylo nějaké množství volné fyzické paměti?
  - Zloděj stránek
- Je-li fyzická paměť plná, jak určím, který rámec z RAM vyhodit na disk?
  - Strategie Second Chance, Clock, LRU, NRU, atd.

Tabulka stránek procesu

- Každý proces má svojí
- Mapování číslo stránky na číslo rámce
- Říká, zda je daná stránka v určitém rámci v RAM, nebo leží ve swapu na disku (a kde)
- Velikost stránky = velikost rámce = 4kB
- Mapování provádí MMU, je součást CPU
- K urychlení se používá TLB cache

Výpadek stránky

- Proces využívá logickou adresu 5000, používá se stránkování
- CPU zjistí z tabulky stránek, že stránka zahrnující adresu 5000 není v RAMce, ale na disku ve swapu
- Nemůže dál pokračovat –vnitřní přerušení – výpadek stránky
- Součástí obsluhy tohoto přerušení je: uvolnit místo v RAM je-li třeba, nahrát stránku z disku do RAM, změnit mapování v tabulce stránek a znovu se pokusit vykonat stejnou instrukci

Semafory

- Mají smysl pouze tehdy (stejně jako semafory na křižovatce), pokud je všechny vlákna přistupující k sdílenému objektu kontrolují

- Najde-li se jeden, který na kontrolu správného semaforu zapomene, hrozí problém současného přístupu

Semafory

- Na semaforu je červená , s = 0
- Na semaforu je zelená, s = 1, 2, …

Pro vzájemné vyloučení hodnoty semaforu 0,1 (obsazeno, volno)

Pro jiné synchronizace, např. vkládání do bufferu i vyšší hodnoty – hodnota semaforu pak znamená, kolik položek se do bufferu ještě vejde

Monitory

- Test podmínky
- if (piv < 1) waitc(c1);

- Obecnější, bez ohledu na typ sémantiky:
- while (piv < 1) waitc(c1);