Co před námi tají /proc I.

Verze článku pro čtení: http://root.cz/clanek.php4?id=737

22.06.2001

Jedna ze základních vlastností operačního systému Unixového typu je, že před vámi nic neskrývá. Druhou by mohlo být, že vše lze přizpůsobit a nastavit k obrazu svému. K čtení i nastavování hodnot jádra systému se dnes nejčastěji používá souborový systém /proc.

Na základě vlastní potřeby a vyburcován dopisy v [mailto:linux@linux.cz] linux@linux.cz jsem se rozhodl vytvořit malý seriál o tomto souborovém systému. Pro začátek bych se rád ponořil do obsahu jeho mnoha podadresářů a souborů a pro sebe i pro vás se pokusil zjistit, co všechna ta čísla vlastně znamenají. Možná to bude trochu připomínat detektivní pátrání, ale věřím, že s pomocí Webu, zdrojového kódu Linuxu a grepu se mi podaří shromáždit spoustu zajímavých informací.

Pokud bude zájem, mohl bych též připojit jeden díl o programování modulů pro jádro, které vytváří položky v adresáři */proc*. Možná by neškodilo i malé zamyšlení nad přenositelností programů využívajících tyto informace a knihovny typu *libproc*.

Takže vzhůru do pátrání. Nejprve malý úvod pro ty, kteří nemají úplně jasno v tom, co to ten */proc* je. Jedná se o virtuální souborový systém připojovaný do adresáře /proc. K připojení se použije klasický příkaz mount, například takto:

```
mount -t proc proc /proc
```

Na první pohled je podivné, co je to za speciální soubor v druhém parametru. Odpověď je (překvapivě) zcela prostá. Žádné zařízení *proc* neexistuje a v zápise je použito pouze pro zachování stejné syntaxe s ostatními druhy souborových systémů. Vše, co je třeba, pozná mount podle typu. Ten je v tomto případě *proc*. Pokud jste se někdy vrtali v jádře, víte, že operace nad souborovými systémy (jakož i nad spoustou jiných věcí) jsou realizovány přes popisnou strukturu daného zařízení. Ta obsahuje ukazatele na funkce pro všechny možné operace. Pokud tedy například provedeme otevření souboru v systému, kde je Ext2, použije se ukazatel na funkci "open", který je pro systém Ext2 nastaven na funkci řekněme "ext2_open". Ta schroustá data z disku (respektive z cache) a vrátí je uživateli. Pokud je ale daným systémem souborů */proc*, má nastavenu funkci, která data vygeneruje z informací v jádře. A uživatel nic nepozná, pracuje se soubory úplně stejně. Zápis probíhá obdobně, až na to, že zapisovaná data se nikam neukládají, ale použijí se k nastavení parametrů jádra.

Teď už máme představu, jak souborový systém proc funguje, takže vzhůru na prozkoumávání jeho obsahu. Ten se samozřejmě bude lišit v závislosti na verzi jádra operačního systému, ale hlavně dle toho, co všechno jsme do jádra zakompilovali a/nebo jaké jsou v daném okamžiku nahrány moduly. Určitý základ ale bude stejný, zbytek bude vycházet z mého aktuálního nastavení. Pokud Vám tedy bude nějaká oblast chybět, řekněte si. Na mém počítači je nyní poslední stabilní jádro 2.4.5 (ano, jsem sebevrah :).

Nejprve se podíváme na výpis adresáře */proc*:

```
924/      devices      interrupts  loadavg  net/        sys/
946/      dma          iomem       locks    partitions  sysvipc/
968/      driver/      ioports     meminfo  pci         tty/
apm       execdomains  irq/        misc     self@       uptime
bus/      filesystems  kcore       modules  slabinfo    version
cmdline   fs/          kmsg        mounts   stat
```

```
cpuinfo  ide/         ksyms       mtrr     swaps
```

Tento výpis je "poněkud" zkrácen, adresářů pojmenovaných číslem je ve skutečnosti mnohem víc. Nyní se pojďme podívat na jednotlivé položky výpisu. V tomto dílu si jako předkrm před dalšími zajímavostmi, na jejichž pečlivější nastudování budu potřebovat trochu času, podíváme na informace o procesech. To jsou právě ony adresáře pojmenované číslem. Jak jistě uhodnete, číslo odpovídá PID procesu.

Příkladem nám bude adresář informací o procesu *bash*. Tento proces má PID podle následujícího výpisu programu *ps -e* rovno 300:

```
  PID TTY          TIME CMD
  ...
  300 tty1     00:00:00 bash
  ...
```

Vybrali jsme proces, který jsme sami vytvořili, abychom měli patřičná přístupová práva ke čtení jeho informací. Obsah adresáře */proc/300* je potom následující:

```
-r--r--r--   1 sioux  users   0 Jun 16 13:07 cmdline
lrwxrwxrwx   1 sioux  users   0 Jun 16 13:07 cwd -> /home/sioux
-r--------   1 sioux  users   0 Jun 16 13:07 environ
lrwxrwxrwx   1 sioux  users   0 Jun 16 13:07 exe -> /bin/bash
dr-x------   2 sioux  users   0 Jun 16 13:07 fd/
-r--r--r--   1 sioux  users   0 Jun 16 13:07 maps
-rw-------   1 sioux  users   0 Jun 16 13:07 mem
lrwxrwxrwx   1 sioux  users   0 Jun 16 13:07 root -> /
-r--r--r--   1 sioux  users   0 Jun 16 13:07 stat
-r--r--r--   1 sioux  users   0 Jun 16 13:07 statm
-r--r--r--   1 sioux  users   0 Jun 16 13:07 status
```

Poprvé si to ještě trochu rozpitváme, příště už se budeme zajímat především o obsah souborů. Nejprve práva. Většina je pouze pro čtení, *mem*, což je obraz paměti, je i zapisovatelný (ale pouze pro majitele). Nic překvapujícího, tedy dále. Vlastník souborů je určen podle aktuálního EUID a EGID procesu. Pokud tedy proces bude po dobu svého běhu nějak měnit efektivní UID nebo GID, změna se zde projeví. Velikost je u většiny položek při výpisu adresářů nulová, neboť skutečný obsah je generován až při samotném přístupu k souboru a může se tedy měnit. Časy vytvoření (ale i přístupu atd.) jsou nastaveny na dobu vygenerování záznamů.

A teď už konečně jednotlivé položky:

cmdline
: v našem případě je obsahem "-bash". Jedná se o stejné řetězce, jaké vidí aplikace v poli *argv[]*. Obsahuje tedy název aplikace a případné parametry.

cwd
: symbolický link na aktuální pracovní adresář.

environ
: obsahuje seznam proměnných prostředí dostupných danému procesu. Jde o nulou ukončené řetězce:

  ```
  HOME=/home/sioux\0SHELL=/bin/bash\0TERM=linux\0....
  ```

  Na konci je pouze nula ukončující poslední řetězec a konec seznamu je nutno rozeznávat podle konce souboru.

exe
: symbolický link na soubor, který obsahuje program pro daný proces. Zde je to: "/bin/bash".

fd

Tento adresář obsahuje odkazy na použité deskriptory souborů. Jde o symbolické linky na soubory (pro příklad: "0 -> /dev/tty1").

maps

V jádrech řady 2.4 jsou diskové soubory jako spustitelné (binární) programy a knihovny připojovány do adresového prostoru procesu pomocí mapování příslušného souboru funkcí *mmap()*. Ta způsobí zobrazení obsahu souboru z disku do LAP (Logický Adresový Prostor) procesu a pokud je potom požadována stránka z této oblasti, natáhne se při ošetřování výjimky požadovaná stránka do fyzické paměti. Při uvolňování se tyto stránky (u souborů s kódem) prostě zahodí, protože je lze načíst znovu z disku. Tato položka v systému */proc* obsahuje informace o takto mapovaných souborech. Každý řádek odpovídá jedné oblasti a obsahuje následující položky: rozsah použitých logických adres, práva pro danou oblast, offset v počtu stránek, hlavní a vedlejší číslo zařízení, ze kterého pochází soubor, inode příslušného souboru a nakonec jméno souboru. Pro nás je začátek souboru následující:

```
08048000-080c0000 r-xp 00000000 03:04 17053  /bin/bash
080c0000-080c6000 rw-p 00077000 03:04 17053  /bin/bash
080c6000-08100000 rwxp 00000000 00:00 0
40000000-40015000 r-xp 00000000 03:04 15669  /lib/ld-2.2.so
40015000-40016000 rw-p 00014000 03:04 15669  /lib/ld-2.2.so
....
```

Je vidět i díra v LAP, dále je patrné, že kód a statická data jsou mapována zvlášť kvůli přístupovým právům. Přístupové právo 'p' značí "private", opakem je 's' pro "shared".

mem

Zpřístupňuje logický adresový prostor procesu. Bohužel se mi tento soubor nepodařilo přečíst. Při čtení to vždy hlásilo chybu:

```
se710.root /proc/277> cat ./mem
cat: ./mem: Tento proces neexistuje
```

Pokud zjistím, v čem byl problém, dodělám errata.

root

symbolický link na kořenový adresář pro daný proces. Většinou je to "/", pouze pro chrootované programy je hodnota jiná.

stat

obsahuje informace o procesu. Stejné informace, ale v čitelnější podobě, přináší soubor *status*

statm

Informace o stavu paměti využívané procesem. Jednotlivé čísla mají následující význam: velikost programu, počet stránek celkem, počet sdílených stránek, počet stránek kódu, počet stránek dat a zásobníku, počet stránek knihoven, počet modifikovaných stránek. Všechny počty stránek se vztahují k aktuálnímu obsahu fyzické paměti pro daný proces. V našem případě je obsah:

```
"89 1 1 0 0 1 0"
```

status

Zde lze najít následující informace o procesu (a opět ukázka našeho procesu 300):

```
Name:   bash
State:  S (sleeping)
Pid:    300
PPid:   294
TracerPid:      0
Uid:    500     500     500     500
Gid:    100     100     100     100
FDSize: 256
Groups: 100
VmSize:     2432 kB
VmLck:         0 kB
```

```
VmRSS:          0 kB
VmData:       260 kB
VmStk:         24 kB
VmExe:        480 kB
VmLib:       1412 kB
SigPnd: 0000000000000000
SigBlk: 0000000000010000
SigIgn: 0000000000384004
SigCgt: 000000004b813efb
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
```

Položky jméno, stav, PID, PPID (Parent PID) jsou snad jasné. Jenom připomenu, že možné hodnoty stavu jsou: R(running), S(sleeping), D(disk sleep), Z(zombie), T(stoped), W(paging). Takže co ten zbytek.

TracerPID: dle zdrojového kódu jádra nastaven napevno na hodnotu '0'.
Uid: pole obsahuje postupně UID, EUID, SUID, FSUID
Gid: pole obsahuje postupně GID, EGID, SGID, FSGID
FDSize: velikost tabulky deskriptorů souborů
Groups: seznam skupin do nichž proces patří
VmSize: Obsazený LAP
VmLck: Velikost uzamčené paměti (LAP i FAP)
VmRSS: Velikost přiděleného FAP (RAM)
VmData: Velikost datových stránek
VmStk: Velikost zásobníku
VmExe: Velikost kódu programu
VmLib: Velikost paměti pro kód knihoven
SigPnd: maska signálů doručených během blokování
SigBlk: aktuální maska blokovaných signálů
SigIgn: aktuální maska ignorovaných signálů
SigCgt: aktuální maska obsluhovaných signálů (aplikací)
CapInh,Prm,Eff: zděděné, nastavené a efektivní vlastnosti procesu. Obsluhuje některá (především bezpečnostní) nastavení procesu dle normy POSIX. Příkladem budiž CAP_CHOWN, což je příznak povolující nebo zakazující změnu vlastníka. (detaily - norma POSIX)

Tak a to je pro dnešek vše. Očekávám vaše připomínky, názory a komentáře v diskusi k článku. Ve výpisu adresáře */proc* v tomto článku vidíte, co bude postupně obsahem dalších dílů.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=737

Co před námi tají /proc II.

Verze článku pro čtení: http://root.cz/clanek.php4?id=746

02.07.2001

Minule jsme si dali lehký úvod, takže co bude následovat dnes? Podíváme se na skupinu souborů v adresáři /proc poskytujících základní a/nebo všeobecné informace o systému.

Nejprve několik drobných změn. Kdo jste četli reakce na minulý díl, víte, že se mi (bohužel) podařilo uzmout tohle téma jinému člověku před nosem. Tak to totiž dopadá, když člověk nečte konferenci pořádně a vrhá se do diskuse od prostředka. Doufám, že se s původním autorem (a s redakci Roota) nějak domluvíme, každopádně o informace určitě nepřijdete.

Další "drobností" je, že mám nový počítač, a tak budou informace od nynějška poněkud aktuálnější co se týče hardwaru. Původním referenčním strojem byla Acer Extensa 710DX, teď mám k dispozici novou Pé-trojku. To jenom kdyby se někomu zdálo, že si některé informace v prvním dílu neodpovídají se zbytkem. Takových údajů by ale mělo být minimum.

Pro oživení opět uvedu základní výpis adresáře proc v nynější aktuální podobě:

```
809  cmdline      fs/          kmsg     mounts      slabinfo  version
840  cpuinfo      ide/         ksyms    mtrr        stat
842  devices      interrupts   loadavg  net/        swaps
847  dma          iomem        locks    nv/         sys/
848  driver/      ioports      meminfo  partitions  sysvipc/
apm  execdomains  irq/         misc     pci         tty/
bus/ filesystems  kcore        modules  self@       uptime
```

Jak vidíte, změn není mnoho, respektive je jich minimum.

Tentokrát se tedy pustíme do obsahu jednotlivých souborů v základním adresáři. Takže vzhůru na ně!

apm

Obsahem souboru je pro nás:

```
1.14 1.2 0x07 0x01 0xff 0x80 -1% -1 ?
```

To vypadá docela vtipně, nemyslíte? Ale zase tak tajuplné to není. První číslo je verze ovladače APM v jádře (modul apm.o), druhé označuje verzi APM BIOSu a závisí tedy na konkrétním počítači. Tyto informace můžete získat pomocí příkazu *apm -v*. Ten vypíše následující informace:

```
APM BIOS 1.2 (kernel driver 1.14)
AC on-line, no system battery
```

Na stroji poháněném bateriemi (Acer Extensa 710) je potom výpis následující:

```
APM BIOS 1.2 (kernel driver 1.14)
AC off-line, battery status critical: 6%
```

Další čtyři čísla v souboru apm jsou příznaky s následujícím významem:

1. Příznaky APM BIOSU
bit 0 nastaven: 16bit APM podporováno

bit 1 nastaven: 32bit APM podporováno
bit 2 nastaven: stav procesoru IDLE způsobí zpomalení hodin
bit 3 nastaven: APM BIOS zakázaný (DISABLED)
bit 4 nastaven: APM BIOS nevyužitý (DISENGAGED)

2. Stav zdroje napájení
0x00: vypnut (off-line)
0x01: zapnut (on-line)
0x02: využívá se záložní zdroj (APM BIOS >=1.1)
0xff: Neznámý (unknown)

3. Stav baterie
0x00: Nabitá (High)
0x01: Částečně vybitá (Low)
0x02: Téměř vybitá (Critical)
0x03: Nabíjí se (Charging)
0x04: Vybraná baterie nepřítomna (APM BIOS >=1.2)
0xff: neznámá (unknown)

4. Příznaky baterie
0x00: Nabitá (High)
0x01: Částečně vybitá (Low)
0x02: Téměř vybitá (Critical)
0x03: Nabíjí se (Charging)
0x04: Není systémová baterie
0xff: neznámá (unknown)

5. Zbývající množství energie: (procento)
0-100 procento zbytku energie
-1 neznámo, zobrazí se i tehdy, pokud baterie chybí

6. Zbývající množství energie (čas)
num min/sec zbývá num minut nebo sekund běhu. Pokud nelze určit, zobrazí se znak '?'.

Na tomto případě bych rád ukázal, jak skvělé je mít k dispozici zdrojový kód systému (nejen operačního), na kterém pracujete. Veškeré výše uvedené informace jsem totiž bez problémů nalezl v kódu a dokumentaci k modulu *apm.o* v souboru *arch/i386/kernel/apm.c*. Mám také zaručeno, že dodané informace jsou absolutně čerstvé a aktuální a vše si mohu vyzkoušet, ověřit a v případě potřeby modifikovat. Proto doporučuji se na tento soubor podívat a "za domácí úkol" si uvedené informace nalézt. Jen podotýkám, že pracuji s jádrem verze 2.4.5.

cmdline

tento soubor obsahuje kompletní seznam parametrů předaných jádru zavaděčem OS (v našem případě LILO) při startu. Parametry jádra jsou v podstatě jedinou cestou, jak může zavaděč s jádrem komunikovat. Jedná se o maximálně 256 znaků dlouhý řetězec ukončený '\0'. Ukázkou je tento řádek:

```
BOOT_IMAGE=l4 ro root=303 BOOT_FILE=/boot/vmlinuz-2.4.5
```

"BOOT_IMAGE" je název sekce v konfiguračním souboru */etc/lilo.conf*, "BOOT_FILE" soubor s jádrem, "ro" způsob připojení kořenového svazku (pouze pro čtení je kvůli kontrole integrity pomocí *fsck*) a nakonec "root" je označení pro diskový oddíl, ze kterého se bude

bootovat. 303 znamená zařízení s čísly 03:03, tedy /dev/hda3. (kanál IDE0, 3 partition)

cpuinfo

Tahle položka je celkem průhledná. Prostě "oťukne" váš procesor a vypíše o něm vše možné i nemožné. Příkladem budiž třeba:

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 8
model name      : Pentium III (Coppermine)
stepping        : 6
cpu MHz         : 998.392
cache size      : 256 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 3
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8
apic sep mtrr pge mca cmov pat pse36 mmx fxsr sse
bogomips        : 1992.29
```

Jenom letecky. První řádek udává identifikaci (číslo) procesoru v jádře a má smysl pouze pro SMP systémy. Další položky potom popisují model procesoru, jeho frekvenci a velikost cache. Přesnější velikost cache se vypisuje při startu systému a lze ji tedy nalézt v souboru */var/log/dmesg*. Zajímavé může být rozepsání obsahu příznaků procesoru při instrukci *cpuid*, což je předposlední řádek. Tam je potom vidět například podpora MMX instrukcí, APIC - lepšího řadiče přerušení, instrukcí SSE a dalších vymožeností moderních procesorů.

To je pro tentokrát poslední položka výpisu. Moc toho dnes pravda nebylo, ale snad už příště bude líp. Doufejme, že už bude vyřešen problém dvou autorů stejného tématu a seriál bude moci pokračovat dál, jen možná s jiným podpisem.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=746

Co před námi tají /proc III.

Verze článku pro čtení: http://root.cz/clanek.php4?id=754

12.07.2001

Tentokrát budeme pokračovat v popisu obsahu souborů základního adresáře proc. Na řadě jsou dnes soubory devices a dma.

Pokud se podíváte na diskusi k minulému dílu, najdete tam několik zajímavých výpisů souborů z */proc*. Určitě stojí za shlédnutí. Stroj, na kterém články dělám, je taky dual (Abit VP6 - Dual PIII), ale mám zatím (z finančních důvodů) osazen jen jeden procesor. Snad časem, až napíšu hodně dílů tohoto seriálu :-). K výpisům informací o obsazení sběrnice PCI se určitě dostaneme později.

Pro lepší orientaci opět výpis základního adresáře */proc*:

```
809  cmdline      fs/          kmsg     mounts      slabinfo  version
840  cpuinfo      ide/         ksyms    mtrr        stat
842  devices      interrupts   loadavg  net/        swaps
847  dma          iomem        locks    nv/         sys/
848  driver/      ioports      meminfo  partitions  sysvipc/
apm  execdomains  irq/         misc     pci         tty/
bus/ filesystems  kcore        modules  self@       uptime
```

Minule jsme skončili u informací o procesoru (pro některé případně procesorech) a dnes pokračujeme dál:

devices

Nejprve výpis z ukázkového stroje:

```
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 vc/0
  5 ptmx
  7 vcs
 10 misc
 14 sound
128 ptm
136 pts
162 raw
180 usb
195 nvidia

Block devices:
  2 fd
  3 ide0
 22 ide1
```

Jak vidno, soubor obsahuje zaregistrovaná znaková (Character) a bloková (Block) zařízení. Rozdíl mezi nimi je ve velikosti najednou přenášených/adresovatelných dat. Zatímco u znakových zařízení se komunikuje po jednotlivých bajtech, znacích, případně slovech, u blokových zařízení se pracuje (kvůli efektivnosti) s větším blokem dat najednou. Jako příklad znakového zařízení lze uvést třeba zvukovou kartu (*14 sound*), USB sběrnici (*180 usb*) nebo grafickou kartu (*195 nvidia*). Typickými blokovými zařízeními jsou naopak disky všeho druhu, v našem případě floppy rozhraní (*2 fd*) a ide rohraní na (nejen) hardisky (*3 ide0*).

Jádro operačního systému komunikuje s ovladači zařízení pomocí jejich *hlavního* (*major*) a

vedlejšího (*minor*) čísla. Hlavní číslo spolu s typem zařízení určuje druh hardwaru. Například char,1 jednoznačně (při statickém přidělování) určuje fyzickou paměť systému. Jednotlivá čísla zařízení jsou patrná při výpisu adresáře */dev*:

```
brw-rw----    1 root     disk       3,   0 srp 24  2000 hda
brw-rw----    1 root     disk       3,   1 srp 24  2000 hda1
brw-rw----    1 root     disk       3,   2 srp 24  2000 hda2
brw-rw----    1 root     disk       3,   3 srp 24  2000 hda3
brw-rw----    1 root     disk       3,   4 srp 24  2000 hda4
brw-rw----    1 root     disk       3,   5 srp 24  2000 hda5
```

Písmeno na začátku každého řádku určuje typ souboru a *b* znamená právě blokové zařízení. Další informace z výpisu jsou identické jako u normálních souborů až na velikost. Na jejím místě jsou uvedena dvě čísla, a to právě hlavní a vedlejší číslo pro zařízení, které reprezentuje daný (speciální) soubor. V našem výpisu je vidět master disk na prvním kanálu ide. Při statickém přidělování čísel jsou v adresáři */dev* uvedeny všechny (většina) potřebné varianty a využívají se samozřejmě pouze ty, které jsou v danou dobu v systému.

Na mém stroji jsou tedy využity (dle výpisu jádra o rozdělení disku/ů při startu) tyto soubory:

```
Partition check:
 hda: hda1 hda3 hda4
```

odpovídá využití /dev/hda, /dev/hda1, /dev/hda3 a /dev/hda4. Soubor /dev/hda, respektive soubor s hlavním a vedlejším číslem 3:0, ať už se jmenuje jakkoliv, odpovídá celému disku. Číslované /dev/hda? jsou potom jednotlivé diskové oblasti (partitions). Opět zde nezáleží (z pohledu jádra) na jménu souboru, ale na jeho hlavním a vedlejším čísle. Obdobný je význam i pro jiná zařízení než disk. Například při použití více grafických karet v jednom stroji budou mít všechny stejné hlavní číslo a jednotlivé karty se budou rozlišovat pomocí vedlejšího čísla.

Hlavní a vedlejší čísla zařízení jsou dnes přiřazována napevno (staticky) na požádání u patřičné autority. Tou je člověk z okruhu vývojářů jádra, ale jeho jméno si opravdu nepamatuji. V současné době se uvažuje o možnosti přidělování hlavních čísel dynamicky na podobném principu. jako je *devfs* - dynamický souborový systém pro zařízení. Celé je to o tom, že čísel (hlavních) je "pouze" 256, a to dnes při rozmanitosti periferií počítačů přestává stačit. V našem případě je vidět, že ovladač grafické karty má číslo 195, a to už k maximu moc daleko nemá. U vedlejších čísel takový problém není, protože těžko budete mít disk s 255 oddíly nebo 255 grafických karet v systému. Zde jde spíše o to, že je adresář */dev* přeplněn soubory, z nichž se využívá nepatrné procento. Proto někteří lidé přišli s tím, že se bude vytváření souborů v adresáři */dev* a v dalším kroku i přidělování hlavních čísel provádět dynamicky, tedy na požádání, při zavádění ovladačů do systému. Souborový systém */devfs* je již dnes součástí standardních jader (minimálně řady 2.4.x), ale přímé praktické využití je zatím minimální, neboť je nutno modifikovat také systémový software pro práci s novými jmény zařízení. Po dobu přechodu je proto k dispozici démon, který zajišťuje programům překlad přísupu do adresáře */dev* statickým způsobem (překládá jména z nového systému pojmenovávání na stará).

dma

Zde by měly být informace o přiřazení jednotlivých kanálů DMA řadičů, nicméně posuďte sami:

```
 4: cascade
```

To opravdu moc informací není, že. V podstatě chybí veškeré DMA kanály kromě jediného, sloužícího pro spojení obou DMA řadičů. Z důvodu složitosti obvodu totiž řadič DMA není jeden čip, ale funkce je rozdělena mezi čipy dva, a ty spolu musí být nějak propojeny. K tomu

> slouží právě zmiňovaný DMA kanál.
>
> Asi byste řekli, že se využívá více DMA kanálů, to já také, ale při požadavku na výpis se provede průchod strukturou *aktivních* DMA kanálů, a pouze tyto se vypíší. Zkoušel jsem pro vás vykouzlit zajímavější výpis pomocí čtení z disku nebo přehrávání hudby, ale bohužel obsah souboru */proc/dma* byl stále stejný. Pokud máte někdo v tomto souboru i něco navíc, rád bych to viděl (a ostatní určitě taky).

Tak, a na další díl si budete muset počkat zase za týden. Já si teď dávám malou dovolenou a jedu si vyšlápnout do hor.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=754

Co před námi tají /proc IV.

Verze článku pro čtení: http://root.cz/clanek.php4?id=762

19.07.2001

I dnešní díl bude o souborech v základním adresáři. Kupodivu se ukazuje, že i na první pohled zřejmé soubory si vyžadují delší popis a komentář. Asi tento seriál nebude tak krátký, jak jsem si myslel.

Kvůli lepší orientaci opět výpis základního adresáře */proc*:

```
809  cmdline      fs/          kmsg     mounts      slabinfo  version
840  cpuinfo      ide/         ksyms    mtrr        stat
842  devices      interrupts   loadavg  net/        swaps
847  dma          iomem        locks    nv/         sys/
848  driver/      ioports      meminfo  partitions  sysvipc/
apm  execdomains  irq/         misc     pci         tty/
bus/ filesystems  kcore        modules  self@       uptime
```

Předchozí díl končil souborem o DMA kanálech, dnes se tedy vrhneme na další v pořadí - *execdomain*.

execdomain

Řekl bych, že tohle je první soubor, který běžnému uživateli neříká asi vůbec nic. Výpis je skutečně zcelanaprostoúplněnicneříkajícísnůškaznaků. Posuďte sami:

```
0-255   Linux                   [kernel]
```

Tato informace je (dle dokumentace jádra v filesystems/proc.txt) nová pro řadu 2.4 a má co dělat s bezpečností. To nám ale nestačí, nakoukneme proto do zrojového kódu jádra. Existuje přímo soubor *kernel/execdomains.c*, ale množství komentářů v něm je opravdu minimální. První informace tedy je, že základní doména se jmenuje "Linux", což je druhá položka výpisu, že je napevno zakompilována do jádra, to je poslední položka - název modulu, a že platí pro "personality" 0 až 255, tedy pro všechny. Nyní se ovšem vynoří otázka, co znamená pojem "personality". V souboru *include/personality.h* se dozvíme, že možné typy jsou například Linux, SVR3, SVR4, BSD, SunOS, Xenix, Irix64 nebo Solaris.

Soubory, ve kterých se vyskytují alespon trochu rozumné informace, jsou následující: kernel/exec_domain.c, fs/binfmt_elf.c, arch/.../kernel/signal.c. Při použití přibydou ještě kernel/exit.c, fork.c, ksyms.c. V posledně jmenovaných je pouze adminitrativa při implementaci této vlastnosti jako modulu.

Na závěr mohu říci, že mám opravdu jen mlhavou představu, k čemu se *execdomains* využívají. Pravděpodobně jde hlavně o emulaci kódu pro cizí/neznámé platformy a jejich binární formáty. Pro lepší pochopení doporučuji nastudovat problematiku modulu *binfmt*.

Nedlouho před odevzdáním tohoto článku jsem konečně našel nějaké rozumnější informace. Můj odhad byl veskrze správný. Tato vlastnost slouží pro emulaci chování jiných operačních systémů při spouštení jejich nativního kódu. Může ovlivňovat vykonávání některých systémových volání, jako například *select* dle zadaného typu operačního systému. V oficiálním jádře (od Linuse) není implementována žádná dodatečná doména, a je tedy nutno poohlédnout se po ní ve formě patchů nebo modulů na Síti. Můžete si ji samozřejmě také napsat sami :), pro více informací doporučuji *Linux Kernel Internals* kapitolu 3.8.

filesystems

Nejprve samozřejmě ukázkový výpis:

```
nodev   proc
nodev   sockfs
nodev   tmpfs
nodev   pipefs
        ext2
nodev   devpts
nodev   usbdevfs
        vfat
```

Jedná se o seznam souborových systému podporovaných jádrem. Jsou zde vidět skutečné souborové systémy pro disk - *ext2* a *vfat*, dále také virtuální souborové systémy, jako například *proc* nebo *devpts*. Tato druhá skupina vytváří informace v souborovém systému proc, případně zástupné soubory pro *devfs* nebo adresář */dev*. Poslední skupinou jsou souborové systémy, které implementují chování prostředků jako roura (*pipefs*) nebo soket (*sockfs*).

Ještě malou zmínku o *usbdevfs*. Tento vitruální souborový systém zpřístupňuje informace o zařízeních USB. Není ovšem připojen automaticky, a tak je potřeba to udělat ručně:

```
mount -t usbdevfs none /proc/bus/usb
```

K témuž účelu lze samozřejmě využít i *fstab* - přidat do něj řádek:

```
none /proc/bus/usb usbdefs defaults 0 0
```

K podrobnému popisu obsahu těchto adresářu se dostaneme někdy příště.

Přítomnost položky *tmpfs* může být trochu zavádějící. Normálně by člověk předpokládal, že bude přítomna pouze při využívání stejnojmenného souborového systému pro ramdisk. Tmpfs umožňuje ve zkratce vyhradit (proměnlivou) část paměti, kterou lze potom používat jako libovolný jiný souborový systém. Až na to, že po vypnutí nebo restaru se jeho obsah nenávratně ztratí. Implementace souborového systému *tmpfs* se ovšem využívá i tehdy, pokud dotyčný systém není v jádře přímo přítomen (a není ani jako modul), a to pro implementaci některých částí sdílené paměti. Pro další informace doporučuji proto především soubor mm/shmem.c a samozřejmě příslušnou dokumentaci v jádře.

Považuji již za zcela zřejmé, že informace se zde mohou měnit v závisloti na aktuálné nahraných modulech. Stále setrvávají jen ty, které jsou v jádře napevno (z diskových často jen ext2).

Na shledanou s vámi se těším opět za týden.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=762

Co před námi tají /proc V.

Verze článku pro čtení: http://root.cz/clanek.php4?id=770

26.07.2001

Dnešní díl bude o trojici souborů kolem periferií, dozvíte se tedy (snad i něco nového) o přerušeních, vstupně/výstupních portech a mapované paměti pro ovládání periferií.

Již tradičně opět výpis základního adresáře */proc*. Tečka před názvem znamená, že jsme daný soubor/adresář již probrali:

```
.840      .devices      kcore    modules     stat     ide/
.842      .dma          kmsg     mounts      swaps    irq/
.847      .execdomains  ksyms    mtrr        uptime   net/
.848      .filesystems  loadavg  partitions  version  nv/
.apm       interrupts   locks    pci         bus/     sys/
.cmdline  iomem         meminfo  self@       driver/  sysvipc/
.cpuinfo  ioports       misc     slabinfo    fs/      tty/
```

Na začátek si musím trochu sypat popel na hlavu. Posledních pár dílů jsem vás trochu zanedbával, času bylo pořád míň a míň, ale pokusím se s tím něco udělat. Putování po horách už jsem si užil dost, tak budu mít nyní na články (doufám) víc času. Tak co nás čeká tentokrát:

interrupts

Obsah souboru na použitém počítači. Pro ty, co nečtou od začátku, nebo již pozapomněli (obojí je škoda), opakuji, že se jedná o PIII na Abitu VP6. Výbavu je možno vyčíst z výpisu:

```
           CPU0
  0:     460943    IO-APIC-edge  timer
  1:      21161    IO-APIC-edge  keyboard
  2:          0          XT-PIC  cascade
  9:     460757   IO-APIC-level  nvidia
 10:          0   IO-APIC-level  ESS Solo1, eth0
 12:      83236    IO-APIC-edge  PS/2 Mouse
 14:       9901    IO-APIC-edge  ide0
 15:          5    IO-APIC-edge  ide1
NMI:          0
ERR:          0
```

Jak je z názvu souboru patrné, jedná se o informace o využití přerušení. V prvním sloupci je číslo přerušení, v Dalším počet obsloužených přerušení pro jednotlivé procesory (zde pouze jeden - zatím :). Následuje typ obsluhy a ovladač, který obsluhu provádí. Poslední dva řádky určují počet nemaskovatelných přerušení a chyb při obsluhách přerušení.

Nejprve trochu teorie. Desky pro více procesorů (a možná i některé novější jednoprocesorové) obsahují vylepšený řadič přerušení - APIC (Advanced Programmable Interrupt Controller), v Linuxu nazývaný IO-APIC. Jedná se o systém doručování přerušení k procesoru/ům a také příslušný hardwarový prvek. Oproti klasickému řadiči přerušení umí obsluhovat více procesorů (každý z procesorů může obsluhovat přerušení) a také zvládá například sdílení jednoho přerušení více zařízeními (v našem případě přerušení 10 se zvukovkou a síťovkou).

Zbývá ještě říci pár slov o tom, co to je nemaskovatelné přerušení. Obecně lze přerušení rozdělit na několik typů. Prvním typem jsou přerušení procesoru. Jedná se o způsob oznámení fatálních chyb za běhu počítače. Příkladem buďtěž dělení nulou, chybná instrukce,

výpadek stránky a další. Většina z nich způsobuje neopravitelnou chybu (někdy v rámci procesu - dělení nulou v uživatelském kódu, někdy celého jádra při chybě v jeho kódu). Jiné (jako výpadek stránky) jsou standardně obslouženy jádrem a nemusí vést k chybě (výpadek stránky vede k chybě SIGSEGV, pokud ho nelze ošetřit). Dalším typem jsou přerušení pro kontakt s periferiemi. Jsou vyvolávána signálem INTR a zpracovávána procesorem za účasti řadiče přerušení. Řadič řeší například priority přerušení, zachytávání do front(y) a další. Tento typ přerušení je popisován v souboru interrupts. Posledním typem jsou takzvaná softwarová přerušení vyvolávaná instrukcí INT. Ta se používají například jako prostředek pro volání služeb jádra. Z důvodů zachování konzistentního stavu operačního systému je možno v kritických situacích pomocí instrukce CLI (x86) zakázat obsluhu přerušení. Existují ale situace, které je třeba řešit okamžitě (chyba paměti atd.), proto exituje přerušení NMI, které nelze zakázat (jinak též maskovat). To slouží pro oznamování právě těchto chybových stavů. V současném jádře (2.4.5) jsou rozpoznávány chyby 0x80 a 0x40, tedy chyba parity paměti a chyba IO_CHECK. Další chyby mohou být případně rozpoznávány ovladači periferních zařízení na sběrnici MCA (MicroChannel).

Posledním využitím přerušení NMI je takzvaný NMI Watchdog. Je to způsob, jak ladit situace úplného zatuhnutí systému. Generováním nemaskovatelných přerušení lze jistým způsobem přimět systém (občas) k nějaké činnosti. Více informací naleznete v dokumentaci k jádru v souboru *mni_watchdog.txt*.

A teď o tom, co konkrétně náš výpis říká. Počítač obsahuje pouze jeden procesor (přesněji přerušení obsluhuje pouze jeden procesor). V počítači máme 16 linek přerušení (hardwarových) a prozatím jsme se obešli bez chyby. Počet nemaskovatelných přerušení (dále NMI) je rovněž nulový. Jde o čítač všech přerušení procesoru číslo 0x02. V předposledním sloupci je možné nalézt tři typy přerušení. *XT-PIC* je standardní řadič přerušení a dnes je používán téměř výhradně pro kaskádové spojení jednotlivých prvků řadiče (podobně jako DMA řadič). Potom tu máme *IO-APIC* přerušení obsluhovaná řadičem APIC. Ta jsou buď *edge*, což znamená samostatné přerušení pro každé zařízení (IDE disky) s poměrně složitým ovládáním handleru, nebo *level*, což je kromě jiného přerušení pro PCI sběrnice s možností sdílení přerušení. Základní rozdíl je v tom, že obsluha přerušení typu *edge* musí dospět do určité své části (edge), než je možno jednoduže zakázat další přerušení téhož typu. Pro podrobnosti odkazuji na soubory *arch/<xxx>/kernel/io_apic.c* a další z něj vyplývající. Názvy ovladačů zařízení jsou snad dostatečně vysvětlující, a tak jen dodám, že počet přerušení u přerušení časovače je počet tiků časovače od spuštění počítače s přetočením na hodnotě max(unsigned int). Také si lze udělat obrázek o využití zbývajících zařízení. Je vidět, že se nepoužily (od spuštění) zvukovka ani síťovka a disky na ide1 nejsou připojeny (případně využívány, ale to by těch přerušení bylo asi přece jen víc).

No trošku jsem se rozšoupl, tak iomem a ioports přesunu do dalšího dílu. A místo závěrečného proslovu se do něj radši rovnou pustím :)

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=770

Co před námi tají /proc VI.

Verze článku pro čtení: http://root.cz/clanek.php4?id=779

02.08.2001

Pro tentokrát máme resty z minula, takže dojedem, co jsme nestihli a pak se uvidí co dál. Opět tedy budeme mluvit o periferiích a to o jejich ovládání přes porty a mapovanou paměť.

Opět dle tradice výpis základního adresáře */proc*. Tečka před názvem znamená, že jsme daný soubor/adresář již probrali:

```
.840      .devices      kcore    modules     stat     ide/
.842      .dma          kmsg     mounts      swaps    irq/
.847      .execdomains  ksyms    mtrr        uptime   net/
.848      .filesystems  loadavg  partitions  version  nv/
.apm      .interrupts   locks    pci         bus/     sys/
.cmdline   iomem        meminfo  self@       driver/  sysvipc/
.cpuinfo   ioports      misc     slabinfo    fs/      tty/
```

iomem

Nejprve nebytný výpis pro ukázku:

```
00000000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000f0000-000fffff : System ROM
00100000-0ffeffff : System RAM
  00100000-001c646f : Kernel code
  001c6470-0020953f : Kernel data
0fff0000-0fff2fff : ACPI Non-volatile Storage
0fff3000-0fffffff : ACPI Tables
d0000000-d7ffffff : PCI Bus #01
  d0000000-d7ffffff : nVidia Corporation NV11
d8000000-dbffffff : VIA Tech., Inc. VT82C693A/694x [Apollo PRO133x]
dc000000-ddffffff : PCI Bus #01
  dc000000-dcffffff : nVidia Corporation NV11
de000000-de0000ff : Realtek Semiconductor Co., Ltd. RTL-8139
  de000000-de0000ff : 8139too
fec00000-fec00fff : reserved
fee00000-fee00fff : reserved
ffff0000-ffffffff : reserved
```

V prvním sloupci je příslušná oblast paměti (adresového prostoru) a v druhém je řečeno, kdo oblast využívá.

V/V paměť je první ze dvou možných způsobů ovládání periferií. Představte si adresový prostor (v našem případě - architektura Intel x86 - jde o 32 bitů), do kterého jsou mapovány paměťové prvky periferních zařízení. Může jít jednak o stavové a příznakové registry a nebo třeba o vyrovnávací paměti. Důležité je, že ačkoliv vše vypadá jako přístupy do fyzické paměti RAM, ve skutečnosti se jedná o sběrnicovou komunikaci s daným periferním zařízením. S (operační) pamětí RAM to nemá nic společného. Před použitím této paměti je třeba ji ovšem namapovat do LAP. To znamená provést nastavení transformace mezi logickým (virtuálním) adresovým prostorem aplikace/jádra a sběrnicovými adresami pro dané periferní zařízení. Potom je možno (na architektuře Intel x86) použít k přístupu obyčejné ukazatele. Pro čtení a zápis do takových oblastí je ovšem stále vhodné použít speciální funkce a ne běžné dereferencování. To zaručí vyšší přenositelnost mezi jednotlivými

architekturami procesorů a sběrnic. Na Intelu jsou odpovídající funkce pro čtení a zápis (*read[b|w|l]* a *write[b|w|l]*) pouze "prázdnými" makry. Detaily viz soubor .../include/arch-i386/io.h.

Následující tabulka ukazuje rozdělení adresového prostoru s velikostmi a hranicemi jednotlivých úseků:

název	**horní hranice**	**velikost**
System RAM	639 KB	639 KB
reserverd	640 KB	1 KB
Video RAM area	768 KB	128 KB
Video ROM	800 KB	32 KB
System ROM	1024 KB	64 KB
System RAM	256 MB	255 MB
Kernel code	-	1817 KB
Kernel data	-	268 KB
ACPI storage	256 MB	12 KB
ACPI tables	256 MB	52 KB
PCI Bus #1 (NV)	3.25 GB	128 MB
VIA Tech.	3.44 GB	64 MB
PCI Bus #1 (NV)	3.44 GB	32 MB
Realtek 8139	3.47 GB	256 B
...		
reserved	4 GB	

V tabulce je vidět nejprve pozůstalost z dob DOSu - rozdělení začátku paměťového prostoru (640 KB). Dále je vidět vyhrazená paměť pro video adaptér - dnes opět historie. Po doplnění na 1 MB je vidět vlastní kód jádra spolu s jeho datovými strukturami. Oblast system RAM je vlastně dostupná (fyzická) operační paměť RAM bez prvního megabajtu (historie). Následuje několik kilo informací pro ACPI (Advanced Configuration and Power Interface) a pak přijde to hlavní. Dvě z instalovaných PCI/AGP zařízení (síťovka a grafika) jsou vidět ve výpise spolu s oblastmi paměti, které využívají. Také prostředky základní desky (VIA chipset) mají vyčleněny své části paměťového prostoru. Celý rozsah paměťového prostoru je potom 4 GB (2^32).

ioports

```
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
0cf8-0cff : PCI conf1
5000-500f : VIA Technologies, Inc. VT82C686 [Apollo Super ACPI]
```

```
6000-607f : VIA Technologies, Inc. VT82C686 [Apollo Super ACPI]
c000-c00f : VIA Technologies, Inc. Bus Master IDE
  c000-c007 : ide0
  c008-c00f : ide1
c400-c41f : VIA Technologies, Inc. UHCI USB
c800-c81f : VIA Technologies, Inc. UHCI USB (#2)
cc00-ccff : Realtek Semiconductor Co., Ltd. RTL-8139
  cc00-ccff : 8139too
d000-d03f : ESS Technology ES1969 Solo-1 Audiodrive
  d000-d00f : ESS Solo1
d400-d40f : ESS Technology ES1969 Solo-1 Audiodrive
  d404-d40f : ESS Solo1
d800-d80f : ESS Technology ES1969 Solo-1 Audiodrive
  d800-d80f : ESS Solo1
dc00-dc03 : ESS Technology ES1969 Solo-1 Audiodrive
  dc00-dc03 : ESS Solo1
e000-e003 : ESS Technology ES1969 Solo-1 Audiodrive
  e000-e003 : ESS Solo1
```

Zde opět první sloupec udává použitý rozsah adres portů a druhý název zařízení, které tento rozsah používá.

V/V (Vstupně/Výstupní) Porty jsou druhý způsob jak ovládat periferie. Z historického hlediska je starší a vznikl na počítačích s malým adresovým prostorem - tedy s úzkou adresovou sběrnicí. Příkladem budiž archeologické vykopávky Intel 8086 a podobné (nejlépe ještě omezené operačními systémy jako MS DOS). Tam byl 16ti bitový adresový prostor zatraceně malý na to aby se jím plýtvalo a tak byla snaha řešit komunikace s periferiemi jinou cestou. Výsledkem byly porty. Jedná se opět o další (tentokrát zcela samostatný) adresový prostor, který se ovládá speciálními instrukcemi procesoru. Těmi jsou instrukce typu *inb*, *outb* a podobné včetně jejich řetězcových verzí. Hlavní nevýhodou (kromě zbytečné složitosti při dvou oddělených přístupech) bylo omezení této množiny instrukcí. Proto pozdější architektury navržené rovnou pro 32 a více bitů (procesory Alpha od Digitalu) tento způsob vůbec nepodporovaly a v případě Linuxu je nabízen pouze jako mezivrstva emulující chování přes V/V paměť.

Při programování není třeba provádět mapování paměti - jedná se o oddělený adresový prostor a opět existují funkce pro přenos dat z a do adaptéru. Porty jsou rozděleny podle velikosti přenášené jednotky a proto existují modifikace základní funkce *in[b|w|l]* a stejně tak i *out[b|w|l]*. Linux poskytuje i rozhraní na řetězcové (a rychlejší) formy pomocí funkcí *insb* a *outsb*.

Na architekturách, které porty nabízejí, se využívají, i když spíše ze setrvačnosti a pro prostředky s jednoduchým nebo omezeným přenosem dat. Můžeme je tak najít ve spojitosti s ovládáním DMA kanálů, ACPI, USB sběrnic a případně zvukové karty. Některé periferie kombinují použití obou metod jak je vidět třeba na síťové kartě Realtek RTL-8139.

Vzhledem k tomu, že se ukazuje, že mám skutečně o něco více volného času, tak po domluvě s redakcí zvyšujeme tempo (alespoň dočasně) na (zhruba) dva články za týden. To by mohlo znamenat, že se dostaneme do půlky výpisu ještě před Vánocemi :).

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=779

Zdroj: Root.cz (www.root.cz)

Co před námi tají /proc VII.

Verze článku pro čtení: http://root.cz/clanek.php4?id=783

07.08.2001

Pokud šlo vše dobře, je toto je první díl, který k vám přišel dříve jak po týdnu. Jen doufám, že mi to pracovní nadšení vydrží dost dlouho. Dnes se podíváme na několik informací o samotném jádře.

Opět dle tradice výpis základního adresáře */proc*. Tečka před názvem znamená, že jsme daný soubor/adresář již probrali:

```
.840      .devices      kcore    modules     stat     ide/
.842      .dma          kmsg     mounts      swaps    irq/
.847      .execdomains  ksyms    mtrr        uptime   net/
.848      .filesystems  loadavg  partitions  version  nv/
.apm      .interrupts   locks    pci         bus/     sys/
.cmdline  .iomem        meminfo  self@       driver/  sysvipc/
.cpuinfo  .ioports      misc     slabinfo    fs/      tty/
```

Dnes poprvé narazíme na informace, které jsou z bezpečnostního hlediska problematické, a proto jsou jako jedny z mála v systému *proc* pouze pro čtení a pouze pro superuživatele *root*. Na tomto serveru ovšem samozřejmě oprávnění ke čtení máme, a tak si můžou počíst všeci kořeni :).

kcore

> V tomto souboru jádro zpřístupňuje celou fyzickou paměť přesně tak, jak ji jádro vidí. Hodí se tedy ideálně k odposlechu uživatelů (ideálně :) a případně (hlavně) k ladění nebo zjišťování detailních informací. Výpis je ve formátu ELF a informace jsou podobné klasickému výpisu *core*. To je však jen hlavička, po ní následují kopie jednotlivých stránek virtuální paměti z jádra. Detailní popis (v jazyce C :) najdete v souboru *.../fs/proc/kcore.c*.
>
> Ještě si neodpustím několik detailnějších informací, pokud po nich netoužíte, prostě následující dva odstavce přeskočte. Jde o to, jak vlastně vypadá rozložení fyzické paměti (fyz. adr. prostor = FAP) a jak vypadá logický adresový prostor (= LAP). Nejprve fyzická paměť. Když se startuje OS (Unixového typu - Linux), předá zavaděč operačního systému (LILO apod.) řízení na startovací kód obrazu jádra. Tento kód se skrývá v bootsektoru diskového oddílu, ze kterého se jádro startuje (to řeší zavaděč - LILO). Program v bootsektoru (stejně jako LILO pracuje - na x86 - v reálném režimu procesoru) provede nahrání obrazu jádra z disku do paměti, což je doprovázeno jistě známou hláškou "Loading ....". V paměti je obraz umisťován na začátek fyzické paměti. Na architektuře Intel x86 je to od jednoho megabajtu (z historických důvodů - viz MS DOS - kompatibilita se staršími jedinci řady x86), na jiných možná rovnou od nuly (to nemám možnost vyzkoušet). Dnes už není obraz jádra pouze kopírován z disku do paměti, nýbrž se provádí jeho rozbalení (dekomprimace) z formátu zip nebo bzip2. To proto, aby se obraz vešel na disketu a podobná zařízení s malou kapacitou. Takto je zaplácáno určité množství fyzické paměti a jede se dál. Konkrétně je paměť zaplácána vlastním kódem jádra (identifikovaným u programů poněkud matoucím označením *text*) a statickými (staticky alokovanými) inicializovanými daty (kupodivu označovanými jako *data*). Zbytek paměti je potom rozdělen na rámce po 4KB (na Intelu a na Aplha procesoru je to 8KB, jinde zase jinak) a jádro začne jako jeden z prvních úkolů připravovat struktury pro popis mapování virtuální paměti. Jakmile je to hotovo, je možno se přepnout do kýženého chráněného režimu a pokračovat v další inicializaci.

A teď co logický AP. Ten je u procesu tvořen logickými adresami v rozsahu 0 - 4GB. Z toho část pod hodnotou definovanou v jádře jako PAGE_OFFSET (implicitně 3 GB) je určena pro vlastní proces a do oblasti 3 - 4 GB se namapuje jádro. To potřebuje mít také svůj adresový prostor pro manipulaci s pamětí, nicméně ani jádro nemůže pracovat přímo s fyzickými stránkami paměti. Proto je část stránek obsazených jádrem a jeho daty na začátku paměti namapována na virtuální paměť ve zmiňovaném rozsahu (nad 3GB). Mapování je uděláno jednoduše tak, že když k fyzické adrese přidáte konstanty PAGE_OFFSET, dostanete virtuální adresu naopak. To samozřejmě platí jen v režimu jádra. Tak je v rozsahu PAGE_OFFSET až PAGE_OFFSET+velikost_RAM namapována skutečná fyzická paměť, což jádro využívá například k nulování nebo kopírování celých stránek. Potom následuje z bezpečnostních důvodů mezera v LAP jádra (aktuálně 8 MB) a začíná část adresového prostoru pro dynamickou alokaci paměti v jádře. Pokud si tedy v ovladači zařízení alokujete svých 12 kilobajtů paměti, dostanete virtuální adresu někde v tomto intervalu. Tato virtuální adresa bude ukazovat na stránku ve fyzické paměti, na kterou už ukazuje virtuální adresa z prvního mapování jádrem (v rozsahu PAGE_OFFSET až PAGE_OFFSET + velikost_RAM). V určitých případech by na ni mohlo ukazovat ještě i několik virtuálních adres jednotlivých procesů. Na konci virtuálního adresového prostoru jádra se nachází oblast vyhrazená pro tzv. "compile-time virtual memory" (soubor .../include/asm-xxx/pixmap.h). Ty jsou určeny pro speciální případy, kdy je potřeba mít v čase kompilace části jádra pevně danou hodnotu adresy, ale její skutečné umístění je možné určit až při startování jádra. Při použití tohoto řešení je virtuální adresa konstantní a mění se fyzická adresa, na kterou se ta virtuální transformuje.

ELF hlavička tohoto souboru obsahuje také několik užitečných informací, nicméně její hlavní účel je pravděpodobně takový, aby výpis měl podobu běžného souboru *core* a mohl být tedy použit pro alespoň částečné ladění prostředky, jako je GDB apod. Pro zjištění obsahu vlastních informací v hlavičce doporučuji opět soubor *kcore.c* a hlavně funkci *elf_kcore_store_hdr*.

Ukázkový výpis se tentokrát nekoná, asi si dokážete představit binární smetí. Na druhou stranu, kdybych se trošku snažil, můžu vám ukázat obsah stránky paměti, kde je uložen text tohoto článku. Alespoň byste viděli, jak Emacs uchovává v paměti své buffery.

kmsg

Ukázkový výpis byl vyroben příkazem *cat /proc/kmsg > /tmp/kmsg*, následným připojováním a odpojováním svazku s FAT32 filesystémem a poté vkládáním a rušením modulu *apm.o*. Veškeré akce musel provádět superuživatel, stejně tak výpis samotného souboru.

```
<4>MSDOS FS: Using codepage 852
<4>MSDOS FS: IO charset iso8859-2
<6>apm: BIOS version 1.2 Flags 0x17 (Driver version 1.14)
```

Tento soubor obsahuje zprávy vypisované jádrem, které se také objeví v syslogu. Opět se jedná o soubor, který může číst pouze superuživatel. Pokud se odsud přečtou nějaké informace, dle dokumentace k *syslog (2)* už nejsou zapsány do syslogu - lze je číst pouze jednou, ale v praxi se při čtení pouze zkopírují a objeví se následně i v syslogu. Veškeré logovací informace jsou ale bufferované, a proto dochází k určitým zpožděním.

K výpisu těchto informací lze stejně tak použít program *dmesg*, nebo přímo *syslog* a jeho soubory ve /var/log/. Syslog má tu výhodu, že přidá ještě další informace, jako čas pořízení výpisu a název počítače (to se hodí pro vzdálené provádění logování). Navíc se v souboru /proc/kmsg nezobrazují všechny hlášky, ale pouze ty, které jdou na konzoli. Nicméně

úpravám a lidové tvořivosti se meze nekladou - vše podle hesla: "Co si pokazíš, to máš!" :)

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=783

Co před námi tají /proc VIII.

Verze článku pro čtení: http://root.cz/clanek.php4?id=786

09.08.2001

Tentokrát se díky velkému vytížení při psaní dílu o přerušeních vracím k několika informacím, které jsem opomněl zmínit. Tímto děkuji nejmenovanému laskavému čtenáři za připomenutí. Pokračování popisu souborů (dnes to měl být ksyms) odložíme na příště, neboť tohle téma mi připadá taky poměrně zajímavé.

Řekneme si něco podrobností o systému přerušení na architektuře Intel x86 a jeho obsluze v systému Linux. Základní otázka zvídavého čtenáře byla, kde jsou přerušení, jejichž čísla v patřičném výpisu souboru *interrupts* chybí. Odpověď je na jednu stranu velmi jednoduchá: Při tisku se prochází pole všech možných přerušení a tisknou se ty, u nichž je platný (nenulový) ukazatel na strukturu *action*, která obsahuje informace o obsluze přerušení. Patřičná funkce se jmenuje *get_irq_list* a nalézá se v souboru arch/xxx/kernel/irq.c. Položka *action* pro nás zatím bude pro zjednodušení představovat pouze akci vykonanou po přijetí příslušného přerušení. Pokud není co vykonat, nikdo asi přerušení neobsluhuje a tudíž nemá cenu něco vypisovat (stejně není co).

Pokud někomu tohle vysvětlení stačí, nechť laskavě přeskočí pár následujících odstavců (až na ten poslední :). Pro ty ostatní pro lepší orientaci zopakuji původní výpis z pátého dílu o souboru *interrupts*:

```
           CPU0
  0:     460943    IO-APIC-edge  timer
  1:      21161    IO-APIC-edge  keyboard
  2:          0          XT-PIC  cascade
  9:     460757   IO-APIC-level  nvidia
 10:          0   IO-APIC-level  ESS Solo1, eth0
 12:      83236    IO-APIC-edge  PS/2 Mouse
 14:       9901    IO-APIC-edge  ide0
 15:          5    IO-APIC-edge  ide1
NMI:          0
```

Na první pohled by tohle základní vysvětlení možná mohlo vypadat použitelně, jenže existuje několik ALE. S tím, co jsme si právě řekli, si vystačíme například při vysvětlování, proč chybí přerušení číslo 7, o kterém (například z bios setupu) vím, že obsluhuje paralelní port. Podporu paralelního portu nemám zakompilovanou napevno do jádra, protože paralelní port používám výjimečně. Proto je tohle přerušení skutečně volné. Pokud budu potřebovat třeba propojit dva počítače pomocí PLIP a zavedu modul pro paralelní port například takto:

```
modprobe parport_pc
```

s následujícím obsahem */etc/modules.conf*

```
...
#Paralell port module options
alias parport_lowlevel parport_pc
options parport_pc io=0x378 irq=7
...
```

Tak dostanu do syslogu hlášení:

```
Aug  2 16:19:24 sirion kernel: 0x378: FIFO is 16 bytes
...  0x378: writeIntrThreshold is 8
...  0x378: readIntrThreshold is 8
...  parport0: PC-style at 0x378 (0x778), irq 7, using FIFO
```

```
                [PCSPP,TRISTATE,COMPAT,ECP]
```

Během inicializace modulu se provede mimo jiné žádost o přidělení potřebného přerušení, přesněji IRQ. Pokud provedu nyní výpis obsahu souboru /proc/interrupts, přibude nový řádek (označený šipkou):

```
   2:          0          XT-PIC  cascade
-> 7:          2    IO-APIC-edge  parport0
   9:     539350   IO-APIC-level  nvidia
```

Pokud moduly odstraním (příkazem rmmod), řádek opět zmizí. Stejná situace bude u ovladačů pro USB rozhraní, které nejsou také momentálně nahrány. Ty ovšem využívají IRQ 12 spolu s PS/2 myší a tak řádek nepřibude, jen se doplní druhý ovladač pro IRQ 12.

Dále se budeme zabývat sériovými porty. Dle BIOSu (Setupu) mají přiřazeny IRQ 3 a 4. Ovladač sériového portu mám opět jako modul, takže ho zavedu pomocí příkazu modprobe:

```
modprobe serial
```

A patřičný výpis ze syslogu:

```
Aug  2 17:27:38 sirion kernel: Serial driver version 5.05a (2001-03-20)
                               with MANY_PORTS SHARE_IRQ SERIAL_PCI enabled
Aug  2 17:27:38 sirion kernel: ttyS00 at 0x03f8 (irq = 4) is a 16550A
Aug  2 17:27:38 sirion kernel: ttyS01 at 0x02f8 (irq = 3) is a 16550A
```

Nyní by se dalo čekat, že bude situace stejná jako u paralelního portu. Kupodivu se však soubor *interrupts* vůbec nezmění. Po delším pátrání jsem přišel na to, že vlastní přerušení se alokují až v okamžiku otevření souborů */dev/ttyS[0|1]*. Pokud tedy zadáme třeba jen *cat /dev/ttyS0 & cat /dev/ttyS1*, objeví se kýžené řádky:

```
   2:          0          XT-PIC  cascade
-> 3:          1    IO-APIC-edge
-> 4:          1    IO-APIC-edge
   7:          2    IO-APIC-edge  parport0
   9:     253542   IO-APIC-level  nvidia
```

K malému úžasu ovšem není přítomen název ovladače (poslední sloupec výpisu) a to i přes to, že funkce pro požadavek IRQ se zdá být volána správně (s hodnotou příslušného parametru "serial"). Přiznám se, že mi není úplně jasné, proč tomu tak je. Předává se (a uchovává se) sice jen hodnota ukazatele na řetězec, ale adresový prostor jádra by měl být jednotný. Možná je to proto, že je ovladač přeložen jako modul.

Jakmile soubory ttySx uzavřeme, uvolní se i příslušné IRQ. Stejně to funguje i u ovladače floppy mechaniky. Stačí dát mount /mnt/floppy máme další řádek:

```
   4:          1    IO-APIC-edge
-> 6:          4    IO-APIC-edge  floppy
   7:          2    IO-APIC-edge  parport0
```

Doufám, že jste získali představu jak to funguje, a tak zbytek jen shrnu. Chybí nám už jen IRQ číslo 5, 8, 11 a 13. Teď už to není složité. IRQ 5 patří ovladači APIC - tedy řadiči přerušení. IRQ 8 je Real Time Clock a jeho absence je opět způsobena chybějícím kódem v jádře nebo modulem. Řádek uvedu v koncovém výpisu. IRQ 11 je přiřazeno pro RAID řadič HPT370, ten je ovšem v současné době nevyužit. Podle literatury patří IRQ 13 jednotce FPU, tohle řešení komunikace s procesorem se (předpokládám) používalo v době samostatného koprocesoru. Dnes je tedy buď nedostupné (rezervované), nebo volné k libovolnému využití.

Na závěr této konkrétní ukázky tedy celý výpis ještě jednou:

```
           CPU0
  0:     842363    IO-APIC-edge  timer
  1:      37642    IO-APIC-edge  keyboard
  2:          0          XT-PIC  cascade
  3:          1    IO-APIC-edge
  4:          1    IO-APIC-edge
  6:          6    IO-APIC-edge  floppy
  7:          2    IO-APIC-edge  parport0
  8:          0    IO-APIC-edge  rtc
  9:     486873   IO-APIC-level  nvidia
 10:          0   IO-APIC-level  ESS Solo1, eth0
 12:      50836    IO-APIC-edge  PS/2 Mouse
 14:      19831    IO-APIC-edge  ide0
 15:          5    IO-APIC-edge  ide1
NMI:          0
ERR:          0
```

Teď bych ještě rád zúročil všechnu námahu vynaloženou při hledání přesné implementace obsluh přerušení v Linuxu na architektuře Intel x86 a přidám ještě odstaveček o konstrukci IDT.

Procesory Intel x86 v chráněném režimu uchovávají informace o obsluze přerušení v segmentu, jehož obsahem je IDT - Interupt Descriptor Table. To je tabulka vstupních bodů přerušení. Pokud se vyskytne libovolné přerušení, vezme se jako index do této tabulky a provede se obsluha dle obsahu položky v tabulce. Tabulka může být v (nejen v chráněném režimu) umístěna kdekoliv v paměti počítače (tedy nejen prvních 1024 bajtů) a odkazuje na ni registr IDTR. Ten je ovládán privilegovanými instrukcemi LIDT, SIDT Load IDT, Store IDT. Prvky tabulky jsou deskriptory - struktura určená pro popis dalších segmentů v paměti v tomto případě segmentů s kódem obsluhy přerušení. Existuje několik variant obsahu deskriptoru pro obsluhu přerušení, ale tím se zde nebudu zabývat. Velikost této tabulky může být až 256 popisovačů (deskriptorů). Připomenu, že existují čtyři druhy přerušení: externí - maskovatelné a nemaskovatelné a interní - výjimky a softwarové přerušení. Ty jsou naskládány do této tabulky takto:

Index IDL	**využití**
0x00-x01F	systémové,rezervované přerušení. 0x00 - 0x13 jsou výjimky procesoru, 0x02 je nemaskovatelné přerušení
0x20-0x2F	hardwarová přerušení - IRQ. Mapování je nastaveno ve funkci *init_8259A()* v souboru arch/xxx/kernel/i8259.c. Master obvod mapuje přerušení IRQ 0-7 do vektoru na pozici 0x20-0x27, slave obvod pak IRQ 8-15 do pozic 0x28-0x2F. Jedná se o maskovatelná přerušení.
0x30-0xEF	Přerušení pro potřeby ovladačů. Taktéž maskovatelná. Přerušení 0x80 má speciální význam, používá se k provádění systémového volání. To nejde provést pouhým voláním funkce, protože jádro musí běžet v režimu maximální priority (na Intelu Privilege Level - Úroveň oprávnění 0). Volání systému tedy probíhá tak, že se na zásobník vloží požadované parametry volání a nakonec index do tabulky všech systémových volání. Potom se provede zmiňované přerušení, jehož obsluha vybere (částečně) obsah zásobníku a zavolá skutečnou funkci jádra.
0xF0-0xFF	Konec tabulky je vyhrazen systémovým přerušením převážně pro SMP architektury. Z toho 0xF0 - 0xFA jsou volné pro budoucí využití. Význam vektorů 0xFB až 0xFF lze nalézt v souboru include/asm-i386/hw_irq.h. Přerušení jsou využity například k invalidaci TLB nebo pro APIC.

Softwarová přerušení jsou vyvolávána instrukcí INT jejímž parametrem je index do tabulky. Tímto způsobem lze tedy vyvolat libovolné z uvedených přerušení. Vyvolání instrukcí INT lze ovšem maskovat. Maximální počet uvažovaných přerušení (např. při výpisu souboru *interrupts*) je

definován jako NR_IRQS a jeho hodnota je v současnosti 224 (16 IRQ + 208 APIC přerušení - adresový prostor 0x20-0xFF).

Příště už budeme pokračovat normálně souborem ksyms.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=786

Co před námi tají /proc IX.

Verze článku pro čtení: http://root.cz/clanek.php4?id=789

14.08.2001

Teď už budeme zase pokračovat v dalších souborech adresáře /proc. Dnes je na řadě soubor ksyms. Příště se už dostaneme k písmenu L.

Opět dle tradice výpis základního adresáře */proc*. Tečka před názvem znamená, že jsme daný soubor/adresář již probrali:

```
.840      .devices      .kcore    modules      stat      ide/
.842      .dma          .kmsg     mounts       swaps     irq/
.847      .execdomains   ksyms     mtrr         uptime    net/
.848      .filesystems   loadavg   partitions   version   nv/
.apm      .interrupts    locks     pci          bus/      sys/
.cmdline  .iomem         meminfo   self@        driver/   sysvipc/
.cpuinfo  .ioports       misc      slabinfo     fs/       tty/
```

Na začátek jen malé upozornění. Před psaním tohoto článku jsem aktualizoval jádro. Původní 2.4.5 bylo nahrazeno jádrem 2.4.7 se "stejnou" konfigurací. Je proto teoreticky možné spatřit jisté nekonzistence ve vzorových příkladech celé série článků. Pokud si toho budu vědom, upozorním vás, z obsahu informací o změnách (Changelog) však zatím není patrný žádný zásadní rozdíl.

ksyms

> Obsahem souboru jsou informace o symbolech exportovaných jádrem. Jedná se o vstupní bod rutiny, dále řetězec vycházející z jejího názvu a případně název modulu, který rutinu poskytuje. Příkladem buďtež třeba následující tři části souboru:
>
> ```
> d08d617c nvHalVideoFree_NV20    [NVdriver]
> d0862874 CliMakeDeviceFifoList  [NVdriver]
> d0878eb0 dacDisableCRTCSlave    [NVdriver]
> d08e00c6 CH_FSCI_720x576NC      [NVdriver]
> d0869258 DldGetImageOffset      [NVdriver]
> ....
> d0825700 vfat_create_Rbe27f635  [vfat]
> d08259a0 vfat_unlink_Rc285bb37  [vfat]
> d0825a30 vfat_mkdir_R0fe130b5   [vfat]
> d08258f0 vfat_rmdir_R12aa0637   [vfat]
> ....
> c01081a0 enable_irq_Rfcec0987
> c0108140 disable_irq_R3ce4ca6f
> c01089d0 disable_irq_nosync_R27bbf221
> c0108570 probe_irq_mask_R360b1afe
> c0105510 kernel_thread_R7e9ebb05
> ```
>
> Teď je vhodná chvíle říci si něco málo o fungování modulů v Linuxu. Budu se snažit být stručný, protože rozsáhlejší popis bude pravděpodobně zabírat celý jeden díl (možná ten o souboru /proc/modules). Takže, jak je asi známo, může být jádro nakonfigurováno tak, že hlavní část kódu je obsažena v základním bloku, který se nahrává a spouští po startu systému. Odpovídající soubor se často jmenuje *vmlinuz* a bývá umístěn v adresáři */boot*, případně v kořenovém adresáři /. Umístění i jméno se ovšem mohou lišit dle distribuce, respektive dle rozvahy uživatele/správce daného systému. Ani jedno není totiž nikde pevně dáno a závisí pouze na odpovídající konfiguraci zavaděče systému (/etc/lilo.conf pro LILO).
>
> Mohou ovšem existovat také části jádra, které potřebujeme pouze občas. Příkladem budiž

ovladač pro nějaký exotičtější souborový systém nebo třeba obyčejný sériový port. V tom případě lze daný ovladač v běžném jádru vynechat a vytvořit jedno "nadupané" jádro, které v případě potřeby nabootujeme (to si lze představit na počítači pro jednoho člověka), nebo budeme používat nadupané jádro rovnou. Problémem ale může být jednak větší spotřeba paměti a jiných systémových zdrojů (teoreticky třeba IRQ nebo DMA kanál, obojí ovšem svědčí spíše o špatně napsaném ovladači), druhý problém může nastat v okamžiku, kdy je třeba přidat nový ovladač nebo aktualizovat stávající na novou verzi. Potom je v případě monolitického jádra třeba vše překompilovat a restartovat.

Proto byly vymyšleny moduly. Jedná se o části jádra, které lze *za běhu* počítače (a operačního systému spolu s jinými programy) vkládat a vyndávat z jádra. Jde o princip zpřístupnění rozhraní modulu pro zbytek jádra a potažmo uživatelské aplikace (rozšiřující systémové volání).

S jejich příchodem bylo nutno doplnit do jádra další administrativu, která jednak zajišťovala vlastní zpřístupnění exportovaných symbolů (názvy funkcí a proměnných), a také bylo potřeba řešit problém s rozličnými verzemi jádra. Představte si totiž situaci, kdy zkompilujete modul pro jádro třeba 2.2.něco, a potom je tato binárka z nějakého důvodu použita k jádru 2.2.něcojiného nebo dokonce 2.4.něco (a nedej bože 1.x.x - tedy starší jádro, než bylo použito pro překlad). Během vývoje jádra se totiž může určitým způsobem měnit jeho rozhraní. Markantní to nebude ani tak u názvů (respektive deklarací) funkcí, jako spíše u změny definice datových typů jádra. Stačí přidat položku do nějaké tabulky informací v jádře (nebo ubrat či změnit typ), a binární podoba modulu se s těmito změnami nedokáže vyrovnat. Uvědomme si, jak totiž vypadá kód pro manipulaci se složitějšími datovými typy nebo pro volání funkcí (s parametry) v assembleru (a potom ve strojovém jazyku). Tam se adresace položky XY z tabulky UV provádí tak, že se sečte adresa začátku UV a offset (posunutí) XY od první položky v bajtech (v některých případech i bitech). Pokud přidáme nebo ubereme položku tabulky, která leží mezi začátkem UV a umístěním XY, offset nebude ukazovat na správná data. Konfliktních situací může vzniknout více, jako ukázka malé noční můry to však doufám stačí.

První řešení je nasnadě. Povolit vložení modulu pouze do jádra stejné verze, pro jakou byl modul zkompilován. To ovšem nemusí řešit všechny problémy (například vlastnoruční úpravy v jádře nebo cizí patche) a navíc je to značně omezující - pro každé nové jádro je třeba přeložit všechny moduly. Lepší bude řešit změny přímo na úrovni jednotlivých symbolů než pro celé jádro najednou. Pokud se tedy změní něco, co využívá daný symbol, bude nutné přeložit znovu příslušný modul. Jde tedy o způsob, jak jednoznačně reprezentovat použité rozhraní jednotlivých symbolů.

Konečným (v dnešní době) řešením je úprava jména symbolu přidáním digitálního podpisu rozhraní (v našem případě "prosté" CRC) k názvu symbolu. Pokud se změní rozhraní, změní se i podpis a chyba se ohlásí jako nenalezený název symbolu. Zbývá zodpovědět otázku, jak se počítá CRC. Provádí to program */sbin/genksyms*, který na vstup dostane zdrojový text (v jazyce C) zpracovaný preprocesorem (jazyka C) a pro každý datový typ provede jeho rozvinutí do základní podoby. To v podstatě znamená patřičná nahrazení pro všechny konstrukce jako *typedef, struct, enum, union* a další. Vznikne definice typu pouze na úrovni jednotek -datových typů - definovaných přímo jazykem. Takový popis se potom vezme jako řetězec a právě z něj se spočítá CRC. To se jako 32bitové číslo spolu se znaky *_R* nebo *_Rsmp_* (smp je pro víceprocesorová jádra) přidá za název vlastního symbolu (což může být, jak jsem zmínil, název funkce nebo externí proměnné). Při překladu jádra je tento program volán v části, kde se tvoří závislosti spuštěním *make dep*. Podobné čachry provádí také překladač jazyka C++ z důvodů možnosti přetěžování funkcí.

V našem ukázkovém výpisu jsou vidět tři typy záznamů. První je pro ovladač ke grafické kartě. Modul se jmenuje NVdriver a při jeho překladu nebyly symboly explicitně exportovány pomocí příslušného makra EXPORT_SYMBOL(), nemají tudíž ochranu pomocí CRC. Proč tomu tak je, se zeptejte pánů od NVidia Corp. Pokud nejsou symboly exportovány, nevadí to samotnému modulu, ten funkce jádra používá ve správném tvaru podle definice (testy na MODVERSIONS na začátku souborů .c). Jde o to, že pokud bude někdo používat rozhraní definované tímto modulem, nemusí zaregistrovat změny v něm. Část tohoto modulu je ovšem distribuována v binární podobě (to by mohlo být hledané vysvětlení neEXPORTovaných symbolů). V této části modulu se rovněž vyskytují symboly se jménem *_nv_rmsym_číslo*, což by mohlo naznačovat další manipulaci ohledně exportu symbolů.

Klasickým příkladem modulu s exportovanými a "ocejchovanými" symboly je ovladač souborového systému FAT - modul *vfat*. Dokonce i názvy funkcí jsou docela jasné. Jsou zde rutiny pro vytvoření a smazání souboru, adresáře. Posledním příkladem jsou symboly exportované samotným jádrem.

Ještě k první položce - adrese. Ta se dá využít třeba pro ladění, kdy získáme (z Oops) obsah registrů. Hodnota EIP (Instruction Pointer) označuje právě prováděné místo. Pokud tedy určíme z nabízených možností nejbližší nižší adresu, známe název funkce, kde došlo k chybě. Tuto práci za nás může také udělat utilitka */usr/bin/ksymoops*. Adresa se vztahuje k adresovemu prostoru jadra a tudiz bude vetsi nebo rovna PAGE_OFFSET.

Výpis obsahu souboru ksyms zajišťuje funkce *get_ksyms_list* ze souboru kernel/module.c. Prochází všechny moduly a z každého vypíše jeho tabulku symbolů, což je seznam řetězců jmen v tabulce informací o modulu. Jako poslední modul je potom uvedeno samotné jádro (hlavní část jádra).

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=789

Co před námi tají /proc X.

Verze článku pro čtení: http://root.cz/clanek.php4?id=792

16.08.2001

Jsme u písmene L. L jako loadavg nebo locks. Taky máme malé výročí - desátý díl seriálu. Když uvážím, že jsem to celé plánoval tak na čtyři až pět dílů, vychází mi z toho dvě možnosti. Buď mám hodně špatnej odhad, nebo jsem moc ukecanej. Nechám na vás, co si vyberete.

Jak velí tradice, nejprve výpis základního adresáře */proc*. Tečka před názvem znamená, že už to máme za sebou. Pravda, nepřibývají zrovna moc rychle.

```
.840      .devices      .kcore    modules      stat      ide/
.842      .dma          .kmsg     mounts       swaps     irq/
.847      .execdomains  .ksyms    mtrr         uptime    net/
.848      .filesystems   loadavg  partitions   version   nv/
.apm      .interrupts    locks    pci          bus/      sys/
.cmdline  .iomem         meminfo  self@        driver/   sysvipc/
.cpuinfo  .ioports       misc     slabinfo     fs/       tty/
```

loadavg

Zase jeden ze souborů, který možná normálnímu uživateli moc neřekne. Těm znalejším jsou asi jasná první tři čísla. Ale co zbytek?

```
0.00 0.00 0.00 1/37 844
```

Jak vidíte, psaní článků, jako je tento, není příliš náročnou úlohou pro počítače dnešních dnů. Ještě že máme počítačové hry a SETI :).

Kolem hodnot loadavg se před časem vedla poměrně rozsáhlá debata na konferenci *linux@linux.cz (cz.comp.linux)*. Číslo by mělo znamenat průměrnou zátěž systému za poslední minutu (první hodnota), pět minut (druhá) a patnáct minut (třetí hodnota). Dle manuálové stránky k programu *top* jde o průměrné množství procesů ve stavu READY za posledních x minut. Podle tohoto jednoduchého výkladu by to znamenalo, že ideální stav je, když jde o čísla kolem (těsně pod) 1.00. Jestliže je hodnota menší, je systém v prostoji, je-li větší, je systém naopak zahlcen a nestíhá vyřizovat požadavky jednotlivých programů na systémové zdroje. Schválně jsem předchozí tvrzení formuloval pokud možno obecně, protože se liší názory na to, co vše by mělo být a co ve skutečnosti je zahrnuto v této hodnotě. Příkladem budiž proces čekající na dokončení diskové operace. Pokud budeme uvažovat pouze zátěž procesoru (CPU), je tento proces neaktivní a čeká. Pokud se budeme na systém dívat jako na celek, proces využívá jeho část a je tedy aktivní. Na druhou stranu, pokud proces čeká na data čtená z disku, lze předpokládat (pokud vyloučíme chyby nebo výpadky hardwaru), že čekání v blízké a předem (shora) ohraničitelné době skončí. Pokud ale čekáme na data třeba ze síťové karty, nelze předem říct, jak dlouho bude trvat, než je dostaneme. Pokud druhá strana ještě nezačala vysílat, jsou celé síťové rozhraní OS i vlastní HW v klidu a nepřispívají tedy k zatížení systému.

Nyní se tedy pokusíme ze zdrojového textu určit přesný (a aktuální) způsob určování těchto hodnot. Po dlouhé době se odpovídající funkce pro čtení souboru z adresáře proc neodvolává na další obsluhu, ale řeší vše briskně sama. Z funkce *loadavg_read_proc()* zjistíme rychle význam posledních třech čísel. Úplně na konci je poslední přidělené PID, číslo posledního vytvořeného procesu. V našem případě jde o proces, který provedl výpis tohoto souboru (*cat /proc/loadavg*). Dvojice čísel oddělených lomítkem udává počet právě běžících procesů

(těch, které jsou ve stavu READY nebo RUN). Jestliže je na ukázkovém počítači na tomto místě číslo 2, znamená to, že dva procesy se střídají na procesoru, a zatímco jeden aktivně běží, druhý čeká ve frontě, až na něj přijde při příštím přeplánování řada. Za lomítkem je celkový počet procesů v systému. Jedná se o aktuální počet prvků tabulky procesů a nemusí jít vždy o uživatelské procesy. Jsou zde zahrnuty i "procesy" jako *bdflush, kapm-idle* nebo *kswapd*, které slouží k řešení systémových záležitostí, běží zcela v režimu jádra a jako procesy jsou implementovány jen kvůli zjednodušení. Je tak totiž jednodušší třeba počítat časy běhu jednotlivých procesů, poměr mezi systémovým a uživatelským časem a určitě by se našla i hromada dalších věci.

Tak a teď vlastní hodnoty loadavg. V tomto zvláštním případě budu co nejpřesněji popisovat aktuální stav ze zdrojových kódu, abych co nejvíce zamezil nepřesnostem.

V souboru *include/linux/sched.h* je popsána definice formátu čísla, který se používá pro uložení hodnot loadavg do 32bitového čísla (v tomto případě int). Jedná se o formát fixed-point. Ten je definován tak, že bity 0 až FSHIFT-1 (současná hodnota 11) jsou určeny pro uložení reálné části (za desetinou tečkou), bity FSHIFT až 31 uchovávají informaci o celé části čísla. Pomyslná desetinná tečka (čárka, jste-li příznivci LOCALES) je na pozici mezi bitem FSHIFT a bitem FSHIFT-1. Jednotlivé řády představuje následující schéma. Bity v čísle typu int označme takto:

```
2^31 2^30 ... 2^FSHIFT , 2^FSHIFT-1 ... 2^1 2^0
```

Jim potom na stejných pozicích odpovídají významově bity formátu fixed-point:

```
2^20 2^19 ... 2^0 , 2^-1 2^-2 .... 2^-11
```

Tímto způsobem jsou tedy uložena čísla pro loadavg. Při tisku jejich hodnot se nejprve ke každému přičte 0.005, čímž se po odřezání dosáhne zaokrouhlení na dvě platné číslice za desetinou tečkou. Před tiskem jsou pak použita makra pro ořezání nepotřebných částí - reálné části pro celé číslo a zbytku reálné části pro dvě cifry za desetinou tečkou. Trošku s podivem je, že pole pro uložení těchto tří hodnot *avenrun[]* obsahuje dle definice hodnoty typu unsigned long. Nicméně pro dnešní 32bitové počítače i při ořezání na int dostaneme maximální load kolem milionu procesů. To asi běžnému uživateli stačí (alespoň mně většinou jo :).

Tím jsme vyřešili, co se vlastně tiskne, teď už "zbývá" jen zjistit, co se do patřičných částí vlastně ukládá...

... (po dvou hodinách procházení zdrojových kódů). Už v tom mám jasno. Držte si kloboučky, jedeme z kopce. Na začátek jenom důležitá poznámka. Uvedený postup a princip je nový pro jádro verze 2.4.7 (možná je to už i v 2.4.6, ale není to v 2.4.5). Tak, celkem jednoduché bylo zjistit, kdo nastavuje prvky pole *avenrun*. Je to funkce *calc_load()* v souboru kernel/timer.c. Ta pracuje s frekvencí LOAD_FREQ (aktuálně je to 5*HZ. HZ je počet tiků časovače za sekundu definovaný v souboru *include/asm/param.h* s hodnotou 100 pro x86, 1024 pro Alphy nebo 1000 pro rtLinux - Real Time Linux). Jednou za pět sekund se spočítají aktivní úlohy. Jednoduše se projde tabulka všech úloh v systému, a pokud je hodnota položky *state* v tabulce procesu TASK_RUNNING nebo TASK_UNINTERRUPTIBLE, zvýší se počet "běžících" úloh o jedničku. Zajímavé je, že dle přiřazení hodnot (v *include/linux/sched.h*) se vylučují pouze stavy TASK_RUNNING a TASK_INTERRUPTIBLE. Jestliže máme spočítáno množství "bežících" (lepší by bylo označení aktivních - v případě, že termín "bežící" včetně uvozovek nepřenesete přes srdce, proveďte si *s/"běžící"/aktivní*) procesů, můžeme spočítat hodnotu zatížení systému pomocí

makra *CALC_LOAD* ze souboru *include/linux/sched.h*, kde se provádí následující výpočet:

```
load = (load * e + N * (FIXED_1 - e)) >> FSHIFT
```

Exponent je číslo ve formátu fixed-point o hodnotě 1/(exp^(5sec/[1min|5min|15min])), N je počet aktivních procesů, funkce exp() představuje mocninu přes Eulerovo číslo.

Tím jsme objasnili, jak číslo vznikne, teď ještě kdy se funkce *calc_load* vlastně volá. Je to jednou za určitý počet tiků (tedy ne nutně po každém). Volání je provedeno z funkce update_times, která je volána z handleru dolní poloviny přerušení pro časovač. Handler se jmenuje *time_bh*. Když už jsme v tom, tak ještě vysvětlení, jak momentálně fungují dolní handlery přerušení a proč může být počet tiků mezi voláními více než jedna.

Vlastní handler přerušení od časovače je funkce *timer_interrupt()*. Ta je nastavena ve funkci *time_init()* volané při startu systému ze *start_kernel()*. Handler přerušení provede funkci *do_timer_interrupt()*, a ta konečně zavolá funkci *do_timer()*. Trošku složité? Bude hůř. Pokud vás to nebaví, raději to přeskočte, než abyste se nechali znechutit. Vypisovat jména funkcí je možná nuda, ale až to někdo z vás bude hledat, mohlo by mu to usnadnit život.

Zatím jsme se dostali do nejnižší funkce obsluhy přerušení, která nás zajímá. Zde je obsluha ukončena, když je předtím provedeno volání *mark_bh(TIMER_BH)*. To zajistí provedení dolní poloviny handleru přerušení, kterou tvoří funkce *timer_bh()* zmiňovaná v předchozím textu.

Pro ty nejnáročnější ještě třešnička na závěr. Jak jsou aktivovány dolní poloviny handlerů přerušení? Existuje pole struktur jménem *bh_task_vec*. Další je pole ukazatelů na funkce *bh_base*. Obě mají momentálně 32 položek. Při inicializaci plánovače ve funkci *sched_init()* je položka pole *base_bh* s indexem TIMER_BH (aktuálně 0) inicializována na funkci *timer_bh()*. Dále existuje funkce *bh_action()*, která po provedení různých kontrol a zajištění jedinečnosti spuštěné instance zavolá funkci napojenou na požadovanou položku v poli *bh_base*. V jiné inicializační funkci jsou položky pole *bh_task_vec* nastaveny tak, aby volaly právě funkci *bh_action* s potřebným parametrem.

Tímto se dostáváme k pojmu *softirq*. Dal by se jednoduše vysvětlit jako softwarové přerušení. Ovšem jen kdyby neexistovala ještě jedna softwarová přerušení. Standardně se pod tímto pojmem rozumí část klasického 256ti prvkového vektoru přerušení, která není alokována ani pro systémové výjimky, ani pro tzv. hardwarové přerušení označované jako IRQ. Malou exkurzi už jsme měli v dílu číslo 8. To, o čem teď budu psát, budu nazývat *softirq*, a jak uvidíte, nemá to s přerušeními v klasickém slova smyslu nic společného. V Linuxu existuje taková šikovná věc, jmenuje se fronta úloh. Jedná se o seznam úloh naplánovaných k provedení, něco jako "Seznam věcí, které mám dnes udělat". S tím rozdílem, že je možno v libovolném okamžiku zavolat funkci, která frontu probere a jednotlivé úkoly provede. Programátor si může definovat vlastní fronty a hospodařit s nimi dle libosti, většinou však vystačí s frontami, které má předdefinované jádro.

Teď si vzpomeňte na naši funkci *mark_bh()*. Ta udělá to, že do fronty úloh nazvané *tasklet_hi_vec* přidá patřičnou úlohu z pole *bh_task_vec*. Jak jsme si řekli, jedna z nich je i obsluha časovače funkcí *timer_bh*, dále se volá funkce pro vyvolání *softirq* jménem *HI_SOFTIRQ*. Vyvolání je překvapivě jednoduché. Již dříve jsme mluvili o statistikách pro jednotlivá přerušení v jádře. Existuje také speciální statistika jménem *softirq_pending* a vyvolání přerušení číslo N se provede nastavením N-tého bitu právě v čítači této statistiky. Ten se jako běžný čítač vůbec nepoužívá. Dobrým příkladem této techniky je implementace signálů.

Dobrá, máme zařazenu naši obsluhu do fronty úloh a "generovali" jsme "přerušení". Teď už zbývá jen někdo, kdo frontu provede a vyprázdní. Ten někdo je proces *ksoftirqd*. Ten existuje pro každý přítomný procesor a vzhledem k existenci stejného počtu front jménem *tasklet_hi_vec* (a jiných) je možno provádět obsluhu paralelně. Zpět k našemu démonu. Ten nedělá nic jiného než opakované testování právě statistiky přerušení *softirq* - a pokud se nějaké takové přerušení vyskytne, zavolá funkci pro jeho obsluhu *do_softirq()*. Jedním z "přerušení" je i *HI_SOFTIRQ* a jeho obsluhou je funkce *tasklet_hi_action()*. Ta už pouze vyprázdní frontu úloh *tasklet_hi_vec* a jednotlivé úlohy provede. Jak si jistě ještě vzpomínáte, mezi úlohami byla i naše dolní polovina obsluhy přerušení časovače.

Na úplný závěr přece jen ještě jedna otázka. I když byla nejkritičtější obsluha časovače provedena v horní polovině, stále zbývá pro dolní polovinu dost akcí, které nejde odkládat moc dlouho. Jak je tedy možné svěřit jejich provedení procesu, který je plánován stejným algoritmem jako všechny ostatní procesy? Odpovědí může být třeba to, že po vyvolání přerušení a jeho obsluze horní polovinou je možné procesy přeplánovat a v takovém případě velmi pravděpodobně dostane šanci proces *ksoftirqd*, který běží s prioritou (nice) nastavenou na 19 (což je v našich krajích poměrně dost :).

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=792

Co před námi tají /proc XI.

Verze článku pro čtení: http://root.cz/clanek.php4?id=797

21.08.2001

Znáte to, někde něco chvilku necháte bez dozoru a už je to pryč. Ano, uhodli jste správně, dnes bude řeč o zámcích. Konkrétně o zámcích zmiňovaných v /proc/locks. Jedná se o prostředky k ochraně obsahu souborů - souborové zámky.

Jak velí tradice, nejprve výpis základního adresáře */proc*. Tečka před názvem znamená, že už to máme za sebou:

```
.840      .devices      .kcore    modules      stat      ide/
.842      .dma          .kmsg     mounts       swaps     irq/
.847      .execdomains  .ksyms    mtrr         uptime    net/
.848      .filesystems  .loadavg  partitions   version   nv/
.apm      .interrupts    locks    pci          bus/      sys/
.cmdline  .iomem         meminfo  self@        driver/   sysvipc/
.cpuinfo  .ioports       misc     slabinfo     fs/       tty/
```

locks

Občas se stane, že s jedním souborem potřebuje pracovat více procesů najednou. Vlastně stačí, když nemůžeme zaručit, že nějaký soubor máme sami pro sebe. Potom může dojít k situaci, že na stejném místě v souboru jeden proces data čte a druhý je zapisuje. Pokud mají takové procesy spolupracovat, je jistě třeba zajistit alespoň minimální synchronizaci. Obdobná situace ale může nastat i pro procesy, které spolu nemají nic společného (kromě zmíněného souboru). Tam je to o to horší, že nemáme vůbec žádnou představu, které procesy, kdy, na jakém místě a jak s daným souborem pracují.

Řešením, které pro takové situace poskytuje operační systém, jsou souborové zámky. Informace o nich poskytuje právě popisovaný soubor:

```
1: POSIX  ADVISORY  WRITE 657 03:03:32413 0 EOF \
   c15c9178 c01fdb70 c15c90c4 00000000 c15c93ac
1: -> POSIX  ADVISORY  WRITE 479 03:03:32413 0 EOF \
   c15c93a0 c15c9400 c01fdb78 c15c9178 c15c9408
1: -> POSIX  ADVISORY  WRITE 659 03:03:32413 0 EOF \
   c15c93fc c15c9514 c15c93a4 c15c9178 c15c951c
1: -> POSIX  ADVISORY  WRITE 661 03:03:32413 0 EOF \
   c15c9510 c15c9120 c15c9400 c15c9178 c15c9128
1: -> POSIX  ADVISORY  WRITE 660 03:03:32413 0 EOF \
   c15c911c c01fdb78 c15c9514 c15c9178 c15c9184
2: FLOCK  ADVISORY  WRITE 490 03:03:115476 0 EOF \
   c15c90c0 c15c917c c01fdb70 00000000 c15c90cc
```

Znakem "\" je rozdělen původně jeden řádek na dva. Celkem je tedy v souboru šest záznamů, každý o 13(+1) sloupcích. Je toho poměrně dost, tak si to projdeme pěkně postupně. První sloupec je číslo zámku. To se může opakovat, protože stejný zámek může mít více procesů. První řádek udává aktuálního majitele zámku, proces, který má nyní k souboru přístup. Další řádky se stejným číslem udávají ostatní procesy pracující s daným zámkem. Ty jsou však v současné době (alespoň co se týče zámků) blokovány. Blokované procesy jsou v řádcích označených na začátku šipkou ("->"). Poté následuje několik příznakových slov. Ty udávají typ a parametry zámku. Následuje číslo procesu, který vlastní daný zámek, a potom popis zařízení, kde se zámek nachází. Umístění je dáno hlavním a vedlejším číslem zařízení (oddělena dvojtečkou) a číslem i-uzlu (inode) souboru, který je uzamčen. Následuje rozsah

zámku. Zde je to ve všech případech celý soubor - začátek je 0 a konec je EOF. Posledními informacemi jsou adresy navazujících zámků, které ale bez další pomoci asi nemají valný význam. První je adresa struktury, která se na daném řádku vypisuje, pokračuje to odkazem na předchozí a následující zámek v systému. Předposlední číslo je další zámek pro tento i-uzel a nakonec je zde odkaz na další položku v seznamu zámků blokovaných procesů. Bylo to trochu rychle, ale dál si vše postupně rozepíšeme podrobněji spolu s popisem implementace a chování.

Podrobnější rozbor začněme z pohledu programátora. Bohužel zde není situace úplně jednoduchá, vlivem historického vývoje a soupeření vzniklo několik rozhraní a způsobů implementace zámků. Dvěmi základními větvemi Unixů, které spolu soupeří od "počátku světa", jsou originální AT&T Unixy a jejich klony (označované jako System V.) = SVR4 větev a verze z Berkeley v Kalifornii = BSD větev. Časem přibyla ještě norma POSIX, a hned máme tři odlišné přístupy.

Nejprve pár slov o zámcích všeobecně pro ty, kteří nemají moc dobrou představu, o co vůbec jde. Používají se především dva druhy zámků. Prvním je zámek *sdílený* (*shared*), nazývaný též *čtecí* (*read*). Ten je možno aplikovat na oblast, která není uzamčena vůbec nebo je uzamčena právě sdíleným zámkem. Slouží k deklaraci faktu, že proces bude dané místo souboru číst (odtud druhý název) a nepřeje si, aby se mu v ní někdo vrtal. Druhým typem je zámek *exkluzivní* (*exclusive*) neboli *zápisový* (*write*). Ten umožňuje zamknout pouze oblast, na kterou nebyl aplikován žádný (ani sdílený) zámek. Slouží k získání jedinečného přístupu k části souboru, kterou je pak možno beztrestně modifikovat. Pokud proces požaduje uzamčení nějaké oblasti, které je v dané době neproveditelné (zámek *write* na oblast zamčenou *read*), zůstane zablokován do doby, než je možno požadavek na uzamčení uspokojit. Variantou je neblokující volání, kdy je funkce v této situaci ukončena chybou (EWOULDBLOCK). Proces potom může odložit uzamčení oblasti na později a dělat zatím něco jiného.

BSD Unixy obsahují podporu zamykání souborů od verze 4.2 (dle man flock), jejich sémantika je však dost odlišná od ostatních. Hlavní rozdíl je v tom, že se zamykají celé soubory najednou. Další věcí je, že tyto Unixy ve svém základu nepodporovaly zámky povinné, ale pouze doporučující. Tady si nejsem úplně jist českou terminologií, budu proto používat tyto výrazy: *mandatory* pro povinné zámky a *advisory* pro doporučující (nepovinné).

Pro neznalé vysvětlím rozdíl. Klasické souborové zámky, jak byly vymyšleny (v obou vývojových větvích), jsou doporučující - advisory. To znamená, že pokud uzamkneme soubor tímto zámkem, není pro jiný proces vynucen zákaz přístupu. Místo toho má tento proces možnost zjistit si (explicitně), zda je jím požadovaná část souboru volná (tím, že také použije odpovídající zámek). Pokud ale natvrdo, bez dalších testů něco zapíše na zamčenou pozici v souboru, nikdo mu v tom nezabrání. Použití advisory zámků se potom velmi blíží jiným (explicitním) synchronizačním nástrojům, jako jsou například semafory. Po čase se zjistilo, že výše popsané chování není vyhovující a že když si proces soubor zamkne, měl by mít právo očekávat, že bude mít exkluzivní přístup. Proto lidé ze skupiny kolem systému System V (dle Documentation/mandatory.txt) přišli s povinným zamykáním - mandatory locking. Pokud je implementován a použit povinný zámek, musí operační systém zajistit exkluzivitu přístupu i tehdy, pokud některá ze soupeřících stran nepoužije explicitní zamykání. Při rozhodování, jak navrhnout systém povinných zámků, se vycházelo z toho, že by se nemělo měnit existující rozhraní. Řešením je použití úplně stejných funkcí s tím, že požadavek na povinný zámek se vyjádří nastavením práv příslušného souboru (mimo tyto funkce) na takovou hodnotu, kdy bit X (provedení) pro skupiny není nastaven (0), zatímco

SGID bit nastaven je (1). To je kombinace, která jinak valný smysl nedává.

BSD verze rozhraní je implementována funkcí a poté systémovým voláním *flock()*. Ta zamyká celý soubor na úrovni tabulek operačního systému (inode). Není tedy uzamčen deskriptor souboru a zámek se duplikuje pro volání jako *dup()*, respektive při duplikaci ukazují oba deskriptory na jeden soubor, a tudíž také na ten samý zámek. Možné operace se zámky v této implementaci jsou *LOCK_SH* pro získání sdíleného zámku, *LOCK_EX* pro exkluzivní zámek a *LOCK_UN* pro zrušení zámku. Je možno (*LOCK_NB*) povolit neblokující režim operací. Systémové volání *flock()* nepodporuje zamykání souborů přes NFS (a pravděpodobně ani jiné síťové souborové systémy), alespoň dle informací z manuálové stránky. Dokument *Documentation/mandatory.txt* tvrdí, že BSD verze nepodporuje povinné zamykání, nicméně dle zdrojového textu v *fs/locks.c* je jako další parametr možno použít i *LOCK_MAND* pro povinné zamykání, a to v kombinaci s *LOCK_READ* pro čtecí nebo *LOCK_WRITE* pro zápisové zámky. To vypadá jako dobrá zpráva. Ta špatná je, že při testování ve funkcích jako *open*, *read* nebo *write* se používá funkce (*locks_verify_area()*), která pracuje jen se zámky typu POSIX. Ve skutečnosti jsou tedy (zatím?) povinné zámky pro BSD rozhraní ignorovány.

Druhým typem rozhraní je to ve stylu SVR4. Jedná se o funkci *fcntl()* s argumenty *F_GETLK*, *F_SETLK* a *F_SETLKW*. První příkaz slouží ke zjištění aktuálního stavu požadované oblasti, druhý nastaví požadovaný zámek a třetí je variantou nastavení zámku s blokováním. Informace o zámku jsou předávány ve struktuře typu *struct flock*, která obsahuje položky pro typ zámku (*F_RDLCK, F_WRLCK, F_UNCLK*), určení oblasti v souboru, s kterou se bude manipulovat (počátek a délka). Při zjišťování informací je také vrácena hodnota PID procesu, který právě vlastní zámek. Rozhraní normy POSIX je pouze zapouzdřením a vnitřně volá funkci *fcntl()* s potřebnými parametry. Jedná se o funkci *lockf()* s parametry *F_LOCK, F_UNLOCK, F_TEST, F_TLOCK*. Toto rozhraní pracuje pouze s explicitními (zápisovými) zámky. Velkou výhodou použití *fcntl()* je možnost práce s částí souboru. To ve spoustě případů velmi zvyšuje výkon konečné aplikace (jako třeba logovací démon). Zajímavá je také chyba, vrácená tehdy, nepodaří-li se alokovat další zámek - ENOLCK ("no luck!":). Na rozdíl od BSD rozhraní (kde to nedává smysl) se zde provádí slučování a modifikace zámků opět pro zvýšení efektivity.

Poté, co jsme si vysvětlili, jak zámky fungují ze strany programátora, se opět vraťme k našemu výpisu. Typ zámku (druhý sloupec) tedy může být *POSIX* pro zámky vyvolané přes *fcntl()*, *FLOCK* pro zámky přes stejnojmennou funkci a případě *ACCESS* pro zámky blokovaných procesů při užití povinného zamykání. V případě POSIXu může být typ *MANDATORY* nebo *ADVISORY*, u BSD rozhraní je prozatím jen druhá možnost. Poslední vlastností je typ. Zde jsou možnosti *READ, WRITE, RW* a *NONE*. Snadno přijdete na to, které mají smysluplný význam. K adresám na konci každého řádku zbývá jen dodat, že existují tři seznamy zámků: Seznam všech zámků v systému (bez ohledu na typ), dále seznam všech zámků pro daný i-uzel a nakonec seznam všech procesů, které se na daném zámku mohou blokovat.

Tentokrát jsem se nedostal tolik do způsobu implementace v aktuálním jádře, ale o mnoho jste nepřišli, není na tom nic převratného. Zájemce odkazuji na soubor *fs/locks.c* a příslušné hlavičkové soubory. Tohle téma je dle mého názoru z důvodu několika rozhraní trochu složitější, a tak prosím nad případnými nepřesnostmi přivřete jedno (až dvě - dle potřeby) oko a pokud objevíte (zcela náhodou :) větší chybu, dejte mi vědět.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>

Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=797

Zdroj: Root.cz (www.root.cz)

Co před námi tají /proc XII.

Verze článku pro čtení: http://root.cz/clanek.php4?id=802

23.08.2001

Dnes jsme se konečně dostali na další písmeno. Je to M jako meminfo. Zbývá mi poslední týden volna, tak bych si ještě rád vyjel trochu do hor. Proto to dneska nebudeme moc přehánět. Na celý článek mám totiž "jenom" pár hodin. Tak ať se to raději projeví na kvantitě než na kvalitě.

V tomto dílu budete svědky malého zázraku. Po dlouhé době snad zvládneme více než jeden soubor. Na druhou stranu dva není zase až takový rozdíl :). Pravda, je to hlavně tím, že druhý soubor odbudeme pár větami.

Jak velí tradice, nejprve výpis základního adresáře */proc*. Tečka před názvem znamená, že už to máme za sebou:

```
.840      .devices      .kcore    modules      stat      ide/
.842      .dma          .kmsg     mounts       swaps     irq/
.847      .execdomains  .ksyms    mtrr         uptime    net/
.848      .filesystems  .loadavg  partitions   version   nv/
.apm      .interrupts   .locks    pci          bus/      sys/
.cmdline  .iomem         meminfo  self@        driver/   sysvipc/
.cpuinfo  .ioports       misc     slabinfo     fs/       tty/
```

meminfo

Přehledný způsob, jak získat (nejen úplně základní) informace o stavu paměti. První dva řádky jsou stejné (až na jednotky) jako výpis programu *free*:

```
        total:    used:    free:    shared:  buffers: cached:
Mem:  262213632 115609600 146604032        0  3411968 58961920
Swap: 133885952        0 133885952
MemTotal:       256068 kB
MemFree:        143168 kB
MemShared:           0 kB
Buffers:          3332 kB
Cached:          57580 kB
SwapCached:          0 kB
Active:          18352 kB
Inact_dirty:     42560 kB
Inact_clean:         0 kB
Inact_target:        0 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:       256068 kB
LowFree:        143168 kB
SwapTotal:      130748 kB
SwapFree:       130748 kB
```

Pro jednotlivé velikosti platí většinou celkem intuitivní pravidla. Nejprve první dva řádky. Total je celkové množství dostupné fyzické paměti/swapu v bajtech. U fyzické paměti je od velikosti skutečné fyzické paměti odečteno množství alokované pro jádro při startu systému. To lze nalézt ve výpise při bootování jádra. Used je množství nějakým způsobem použité paměti/swapu. Free je množství paměti, která je nevyužita (ani pro cache nebo jiné účely). Celkem dle očekávání platí Total = Used + Free. Dále, jak bylo řečeno, Fyz.RAM = Total + KernelMem.

Poněkud pofidérní položka je Shared. Intuitivně bych očekával, že jde o počet stránek (krát

jejich velikost), které jsou sdíleny více procesy (například kód programu mingetty nebo Eterm). Bohužel v architekturách ia32, ia64, m68k, arm, alpha, parisc, ppc, sparc i s390 je na tohle místo natvrdo dosazena nula. To může znamenat dvě věci. Buď je význam této položky jiný než můj odhad, nebo všechny tyhle architektury nemají potřebný HW k tomu, aby se tato informace dala (rozumně) spočítat. Jediný, kdo něco počítá, je mips a cris.

Dále se dostáváme k paměti použité pro vyrovnávací účely. Jde o "Buffers" a "Cached". Paměť označená jako "Buffers" je použita jako vyrovnávací paměť pro vstupně/výstupní operace nad blokovými zařízeními. Jsou zde tedy především bloky dat z disků. V Linuxu (narozdíl třeba od mnoha komerčních Unixů) není její velikost (maximum) pevně dána, ale využívá se pro tento účel i dostupná volná paměť. Začne-li paměť docházet, část bufferů se aktualizuje na cílovém médiu (pokud byly modifikovány) a uvolní se pro potřeby uživatelských programů. "Cached" je obdoba pro stránky fyzické paměti. Časem se přišlo na to, že pokud uživatel čte/zapisuje data z/na externí zařízení (jako disk), je druhým koncem komunikace paměť. Proto má smysl provádět uchovávání obsahu ne na úrovni bloků, ale na úrovni paměťových stránek. Existuje proto systém, jak efektivně vyhledávat a manipulovat s patřičnými stránkami. Opět, pokud potřebujeme volnou paměť, najde se stránka, která již není používána, aktualizuje se na médiu a uvolní. Prvním systémem vyrovnávacích pamětí byly buffery na úrovni bloků, pak ale přišla stránková cache a zjistilo se, že je obecnější a výhodnější. Proto dnes existují dvě úrovně vyrovnávacích pamětí.

V dalším výpise se ještě opakují stejné položky, a to jmenovitě: MemTotal, MemFree, MemShared, Buffers, Cached, SwapTotal, SwapFree. Jejich význam je zcela identický s odpovídajícími složkami popsanými výše.

Aktivní stránky paměti jsou ty, které jsou v daném okamžiku skutečně využívány běžícími procesy. Neaktivní se potom dělí na modifikované - před uvolněním je třeba zapsat je někam na disk (do swapu), čisté - nemodifikované, ty lze uvolnit okamžitě, a tzv. target. To je číslo (velikost paměti), které udává míru aktivity subsystému pro správu paměti. Zjednodušeně by se to dalo označit jako průměrné množství paměti požadované za poslední měřený interval (řádově sekundy). Tak jako loadavg určuje zatížení procesoru, tohle číslo určuje zatížení VM (Virtual Memory) subsystému.

V Linuxu je paměť rozdělena na tzv. Low a High. High paměť byla (pro pamětníky) i v DOSu, dokonce s podobným významem jako tady. Jen velikosti a hranice se "trošku" posunuly. V Linuxu je jako High paměť označována fyzická paměť za hranicí normálního adresování v 32bitovém režimu (přes 4GB). Tato paměť není (jako v DOSu) přímo dostupná a musí se pro použití namapovat (viz třeba EMS) do adresovatelné oblasti. Linux tak dokáže pracovat až s 64GB fyzické paměti. Tento systém rozdělení paměti je charakteristický pro 32bitové architektury (konkrétně ia32). Alphy a ia64 už takové zbytečnosti díky svému rozsahu adres nepotřebují.

misc

Soubor misc obsahuje výpis obsazených vedlejších čísel pro zařízení s hlavním číslem 10:

```
175 agpgart
135 rtc
134 apm_bios
  1 psaux
```

Při výpisu adresáře /dev vidíme následující (pro klasický /dev bez devfs).
ls -l /dev | grep " 10," (zkráceno):

```
crw-rw-r--    1 root     root      10, 175 srp 24  2000 agpgart
crw-rw----    1 root     root      10,   4 srp 24  2000 amigamouse
crw-------    1 root     root      10, 134 čen 11 22:21 apm_bios
crw-rw----    1 root     root      10,   1 srp 20 19:17 psaux
crw-rw----    1 root     root      10,   1 srp 24  2000 psmouse
crw-rw-r--    1 root     root      10, 135 srp 24  2000 rtc
crw-rw----    1 root     root      10,   6 srp 24  2000 sunmouse
crw-rw----    1 root     root      10, 165 srp 24  2000 vmmon
```

O co jde? V počítači existuje poměrně dost zařízení, která se (téměř) vždy vyskytují pouze jednou. Klasickým příkladem je *apm_bios* nebo *RTC - Real Time Clock*. Pro ty by bylo zbytečné vyhradit jedno celé hlavní číslo a mít pouze jedno vedlejší (0). Proto je pro tato speciální zařízení vyhrazeno jedno hlavní číslo a jednotlivá zařízení se pak rozlišují číslem vedlejším. Pro jednoduší přehled právě využívaných vedlejších čísel (ke kterým je natažen odpovídající ovladač) existuje právě tento soubor.

Přiznávám, že část pojednávající o vyrovnávajících pamětech je poněkud strohá, a tudíž i možná trochu nepřesná. Bohužel nemám momentálně dostatek potřebných materiálů, abych si to mohl detailněji nastudovat. Jakmile tak učiním, ještě se k tomuto tématu krátce vrátím.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=802

Co před námi tají /proc XIII.

Verze článku pro čtení: http://root.cz/clanek.php4?id=808

28.08.2001

Jak víte z minula, užíval jsem si ještě pár dní volna běháním po horách. Vzhledem k tomu, že jsem se vrátil až v sobotu a v neděli jsem se stěhoval do Brna na privát, dostal jsem se k psaní článku až včera, to znamená jeden den po řádném termínu odevzdání. Ale věřím, že na kvalitě se to neprojevilo. Takže hurá na modules a mtrr.

Jak velí tradice, nejprve výpis základního adresáře */proc*. Tečka před názvem znamená, že už to máme za sebou:

```
.840      .devices      .kcore    modules      stat      ide/
.842      .dma          .kmsg     mounts       swaps     irq/
.847      .execdomains  .ksyms    mtrr         uptime    net/
.848      .filesystems  .loadavg  partitions   version   nv/
.apm      .interrupts   .locks    pci          bus/      sys/
.cmdline  .iomem        .meminfo  self@        driver/   sysvipc/
.cpuinfo  .ioports      .misc     slabinfo     fs/       tty/
```

modules

Protože jsem v dílu číslo devět u popisu souboru *ksyms* sliboval, že v tomto dílu se podrobněji rozpovídám o modulech, bylo by asi vhodné zmínit, že tak zatím neučiním. Důvodem je to, že se ozývají čtenáři, kterým se moje články zdají až příliš rozvláčné. Pokud bych měl ještě o modulech psát, byly by to většinou věci, které drtivá většina lidí bude považovat za zcela zbytečné. Proto, pokud přece jen máte zájem vědět, jak systém modulů, jejich zavádění a případně i programování vypadá, napište mi (na email nebo do diskuse) a dle ohlasů uvidím, co s tím.

Zde je možné nalézt přesně to samé, co říká příkaz *lsmod*. Seznam zavedených modulů, jejich velikosti a návaznosti na ostatní moduly:

```
NVdriver              659104  15 (autoclean)
nls_cp437               4384   1 (autoclean)
via686a                 8496   0 (unused)
i2c-proc                6144   0 [via686a]
i2c-isa                 1200   0 (unused)
i2c-core               13168   0 [via686a i2c-proc i2c-isa]
apm                     9072   2
nls_iso8859-2           3376   2 (autoclean)
nls_cp852               3616   1 (autoclean)
vfat                    9424   2 (autoclean)
fat                    32288   0 (autoclean) [vfat]
esssolo1               24672   0
gameport                1904   0 [esssolo1]
```

Myslím, že význam je docela jasný, ale přece jen. První sloupec je název modulu. Odpovídá názvu souboru v adresáři */lib/modules/verze_jadra/..../jmeno_modulu.o*. Název souboru může být také bez přípony *.o*, jak je tomu u ovladače grafické karty, kde se soubor jmenuje pouze *NVdriver*.

V dalším sloupci je velikost modulu. Tohle číslo není stejné jako velikost souboru s modulem, stejně jako obraz spuštěného procesu v paměti není nikdy kopií binárního souboru, ze kterého je proces spuštěn (ani ihned po startu). Velikost modulu je dána součtem velikostí vlastního kódu, konstantních a staticky inicializovaných dat a případných administrativních

položek.

Následující číslo udává počet použití modulu. Na první pohled tedy vypadá zajímavě, že některé moduly mají na tomto místě nulu. Například modul *fat* je na první pohled aktivní (připojený oddíl se souborovým systémem FAT), ale počet použití je nula. Tohle číslo spravuje interně sám modul, a to pomocí dvou funkcí (maker), jedné pro inkrementaci (*MOD_INC_USE_COUNT*) a druhé pro dekrementaci (*MOD_DEC_USE_COUNT*). Existuje ještě třetí makro pro zjištění aktuálního stavu tohoto počitadla (*MOD_IN_USE*). Tato makra manipulují s položkou *usecount* v tabulce informací o modulech v jádře. Přímý přístup se neprovádí proto, že kromě kontrol zajistí tato makra také atomičnost těchto operací, což může být (na SMP strojích, při práci s přerušeními) velmi důležité. Jejich definici naleznete v *include/linux/module.h*. Makra se používají v praxi tak, že pokud někdo provede operaci nad zařízením (často otevření odpovídajícího souboru v adresáři */dev*), zvýší odpovídající funkce v ovladači počet použití, protože zařízení někdo používá. Naopak při uvolňování zařízení (uzavření souboru) je počet snížen.

Poslední sloupec udává příznaky modulu a jeho závislosti. Závislosti jsou použity utilitami *depmod* a *modprobe* pro (polo) automatické nahrávání sady modulů. Pokud chci například ovladač tiskárny (modul *lp*), stačí zadat *modprobe lp*, a pomocí definovaných závislostí jsou nataženy i moduly pro paralelní port (*parport, parport_pc*). Dále jsou závislosti použity při odstraňování modulů. Nelze odstranit modul, na němž je jiný modul závislý. Závislosti jsou zobrazeny jako seznam (v hranatých závorkách) modulů, které daný modul využívají. Příznaky jsou uvedeny v kulatých závorkách. Asi nejčastějším příznakem je *autoclean*. Ten je nastaven, pokud modul zavede samo jádro "on demand", případně při použití parametru *-k (--autoclean)* pro utility *modprobe (insmod)*. Takto označené moduly mohou být potom automaticky odstraněny, nejsou-li delší čas používány (zpravidla ~ 1 minuta). K odstranění může být použit třeba *cron*, volající opakovaně příkaz k odstranění modulu *rmmod -a*. Parametr *-a* provádí právě odstranění všech nepoužívaných modulů.

Soubor *mounts* přeskočím a nechám na příště, rád bych mu věnoval více času, než mám teď k dispozici (asi hodina na odevzdání :).

mtrr

MTRR znamená Memory Type Range Register. Jedná se o věc specifickou pro architekturu Intel x86. (Při hledání v adresářích se zdrojáky jádra jsem našel zmínku ještě u *sparc64*, ale SPARC bohužel neznám, takže by případné informace byly jen dohady.) Tento rys je k dispozici na procesorech rodiny P6 - tedy PPro, PII, PIII, PIV :). Obdobné mechanismy je možné nalézt také na procesorech Cyrix, AMD, WinChip nebo Cyrix. A o čem to vlastně je?

```
reg00: base=0x00000000 (   0MB), size= 256MB: write-back, count=1
reg01: base=0xd8000000 (3456MB), size=  32MB: write-combining, count=1
reg02: base=0xd0000000 (3328MB), size=  32MB: write-combining, count=1
```

Jak bystrý čtenář jistě instinktivně vycítí, jde o sady registrů, které umožňují kontrolovat přístupy k oblastem paměti. Oblast je dána offsetem (adresou počátku) v LAP (logickém adres. prostoru) a velikostí oblasti. Dále je evidován typ oblasti, tedy typ prováděné kontroly, a počet použití oblasti.

Pro nás bude zajímavý typ *write-combining*. Ten se používá pro grafické karty na sběrnicích PCI nebo AGP. Tento mód umožňuje kombinovat zápisové operace přes tyto sběrnice do větších celků a posílat je najednou. To zvyšuje výkon některých grafických operací (dle *Documentation/mtrr.txt* až 2.5 krát). V uvedeném výpise používá XServer dvě oblasti o velikosti 32 MB ke kontrole

zacházení se zápisy do grafické paměti.

Soubor */proc/mtrr* je jedním z rozhraní určených k ovládání těchto registrů. Zápisem nového řádku je možno kontrolovat chování dalších registrů. Druhým způsobem (používaným třeba v ovladačích *framebufferu*) je rozhraní *ioctl*. Způsob manipulace je pro obě rozhraní detailně (i s příklady) popsán v dokumentaci v *Documentation/mtrr.txt*.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=808

Co před námi tají /proc XIV.

Verze článku pro čtení: http://root.cz/clanek.php4?id=813

30.08.2001

Dneska se vrátíme k souboru s informacemi o připojených discích, který jsme minule přeskočili. Rád bych totiž napsal něco víc, než na co zbýval čas u minulého dílu. Pro detailisty: přešel jsem na jádro 2.4.9!

Jak velí tradice, nejprve výpis základního adresáře */proc*. Tečka před názvem znamená, že už to máme za sebou:

```
.840      .devices      .kcore    .modules      stat      ide/
.842      .dma          .kmsg      mounts       swaps     irq/
.847      .execdomains  .ksyms    .mtrr         uptime    net/
.848      .filesystems  .loadavg   partitions   version   nv/
.apm      .interrupts   .locks     pci          bus/      sys/
.cmdline  .iomem        .meminfo   self@        driver/   sysvipc/
.cpuinfo  .ioports      .misc      slabinfo     fs/       tty/
```

mounts

Základní informací je, že soubor *mounts* obsahuje informace o připojených adresářích. Stejné, jako vypisuje příkaz *mount*:

```
/dev/root / ext2 rw 0 0
/proc /proc proc rw 0 0
none /dev/pts devpts rw 0 0
/dev/hda2 /home ext2 rw 0 0
/dev/hda1 /mnt/user vfat rw,noexec 0 0
/dev/md0 /mnt/raid ext2 rw 0 0
/dev/fd0 /mnt/floppy vfat rw,noexec,nosuid,nodev 0 0
```

Vlastní informace zde obsažené nejsou asi příliš záhadné. Každý řádek obsahuje informace o jedné diskové oblasti, respektive o jednom souborovém systému připojeném do (hlavního - kořenového) adresářového stromu. Nejprve je uveden název speciálního souboru, který odpovídá zařízení, na němž jsou umístěna data. Výjimkou jsou speciální (virtuální) souborové systémy. V našem případě se jedná o souborové systémy */proc* a */dev/pts*. Ty nejsou fyzicky (a perzistentně) umístěny na žádném médiu a mají vyhrazeny speciální hodnoty pro tyto položku (obecně "none"). U ostatních médií se jedná o zařízení, na něž jsou aplikovány vlastní datové operace (jako čtení a podobně). Ještě jednou zvláštností je název speciálního souboru pro kořenový adresář. Položku */dev/root* v adresáři /dev nenajdete (alespoň u mne), tato hodnota je sem zapsána natvrdo v případě, že nepoužíváte *devfs* (*fs/super.c:1672*).

Krátce vysvětlím jména ostatních zařízení. *hdXY* je pevný disk (E)IDE. "X" je písmeno (malé - od "a" dále) a určuje diskovou jednotku. "a" je master disk prvního kanálu, "b" slave, "c" master druhého kanálu a tak dále. "Y" je číslo. 0 uznačuje zařízení odkazující na disk jako celek, od jedničky jsou potom jednotlivé diskové oddíly (partitions). Tohle číslo odpovídá vedlejšímu (minor) číslu daného souboru.

fd0 je disketová mechanika. Pokud jich máte více, bude další *fd1* a dál.

mdX je speciální soubor pro ovladač RAID zařízení. Ovladač pro softwarový RAID zkombinuje definovaným způsobem více diskových oblastí (obecně na různých discích) do jednoho logického celku. Ten je potom přístupný přes tyto speciální soubory. "X" je číslo od

0 výše.

Další položkou na každém řádku je adresář, pod kterým je zpřístupněn obsah daného zařízení. To je snad dostatečně jednoznačné, aby byly další komentáře zbytečné. Jen malá poznámka. Pokud vybraný adresář před připojením obsahuje nějaké soubory (podadresáře), jsou tyto po připojení skryty a nepřístupné. Po odpojení se znovu objeví - zpřístupní.

Na řadě je typ souborového systému. Ten určuje ovladač, který spravuje požadavky na operace s příslušnou částí adresářového stromu. Může se opět jednat jak o virtuální souborové systémy (devpts, proc), tak o skutečné systémy pro správu souborů na fyzických médiích. V současnosti podporuje Linux (verze 2.4.9) například tyto souborové systémy: BFS, CODA, EXT2, HPFS, NTFS, SMBFS, NFS, UFS nebo VFAT. Význam zkratek lze (spolu s dalším popisem) nalézt třeba v *Documentation/filesystems/00-INDEX*.

Předposlední jsou uvedeny parametry zadané při připojování souborového systému. Základním je typ přístupu: pouze pro čtení - "ro", nebo čtení i zápis "rw". Z ostatních parametrů jsou přístupny pouze některé parametry společné pro "všechny" souborové systémy, a to noexec, nosuid, nodev, mand, sync, nosub, noatime, nodiratime. Zvláštní parametry mohou být ještě pro NFS systémy (hard|soft, posix, tcp|udp, ...).

Na konci každé řádky jsou dvě čísla. Jejich hodnota může být 0 (případně nevyplněno) pro vypnutí vlastnosti, nebo 1 (případně vyšší - man fstab) pro zapnutí. První pole udává, zda se bude daná oblast zálohovat programem *dump*. To druhé určuje chování kontroly programem *fsck* při startu systému.

To by byl obsah souboru, teď ještě alespoň pár slov k tomu, jak v Linuxu fungují souborové systémy. Aby bylo možné používat různé souborové systémy (na jednom počítači i obecně), je pro zjednodušení psaní nových systémů a přidávání vlastností jako komprese nebo právě RAID definováno v Linuxu jednoznačné rozhraní. To je určeno pro přístup ze strany uživatelských procesů a jmenuje se VFS - Virtual File System. Jedná se o sady funkcí a definicí, které jsou společné pro všechny (nyní opravdu VŠECHNY) souborové systémy. Definuje takové věci, jako základní rozhraní pro operace nad soubory a adresáři nebo způsob udržování informací o těchto objektech - virtuální uzly (v-nodes).

VFS také udržuje tzv. *dcache* - Directory entry cache. Představte si situaci, kdy chcete otevřít třeba soubor /usr/share/man/cs/man1/echo.1.gz. K tomu je třeba udělat následující. Nejprve musí jádro ze svých tabulek zjistit, na kterém zařízení se nachází oddíl připojený pod kořenový adresář. Potom je nutno z disku načíst superblok tohoto oddílu a z něj určit, kde je i-node popisující kořenový adresář. Potom se načte tolik datových bloků tohoto adresáře, dokud jádro nenajde číslo i-uzlu pro první položku v cestě - tedy adresář "usr". Tak se to opakuje, dokud se nenajde hodnota i-uzlu pro soubor *echo.1.gz*. V ukázkovém případě to může být okolo 10 - 12 čtení z disku. To není moc rychlé. Proto jsou klíčové části udržovány v paměti přímo (třeba superblok). Pro jednotlivé adresáře je následně udržována zvláštní vyrovnávací paměť, která mnohonásobně urychlí opakované vyhledávání souborů.

Pro jednotlivé akce je v v-uzlu patřičný odkaz na skutečnou operaci dle typu souborového systému. Implementovat souborový systém vlastně znamená (kromě jiného) definovat a implementovat funkce jako třeba *do_open()*, *do_read()*. Jejich adresy se potom uloží do odkazů v odpovídajících v-uzlech. Položka v-uzlu *open* tak ukazuje na skutečnou operaci *do_ext2_open()*. Pro jiný soubor může být skutečná operace třeba *do_nfs_open()*.

Není to zase tak složité, jak to na první pohled vypadá. Pokud vás zajímají podrobnosti,

doporučuji především *Documentation/filesystems/vfs.txt*. Zde najdete poměrně vyčerpávající množství detailů. Pokud nestačí, vždycky máte po ruce přímo zdrojáky :).

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=813

Co před námi tají /proc XV.

Verze článku pro čtení: http://root.cz/clanek.php4?id=819

04.09.2001

Dneska zvládneme písmeno P. Probereme informace o diskových oddílech a o obsazení sběrnic PCI (respektive AGP).

Jak asi víte, mám již nějakou dobu novou mašinku. Na asi každém novém počítači je třeba vyladit řadu detailů. Dost dlouhou dobu jsem bojoval s APM biosem, abych ho přemluvil, že chci počítač po shutdownu vypnout (power off). A moc se mi to nedařilo. Proto jsem se nakonec odhodlal vyzkoušet místo APM raději novější ACPI (Advanced Configuration and Power Interface). Kupodivu to funguje, a tak se dnes výpis trochu mění. Přibyl nový adresář *acpi* a ubyl soubor *apm*.

Pokud jste někdo náhodou majitelem Abitu VP6 jako já, ozvěte se mi, rád bych si o pár věcech popovídal.

Jak velí tradice, nejprve (dnes lehce modifikovaný) výpis základního adresáře */proc*. Tečka před názvem znamená, že už to máme za sebou:

```
.840      .devices      .kcore    .modules      stat      ide/      acpi/
.842      .dma          .kmsg     .mounts       swaps     irq/
.847      .execdomains  .ksyms    .mtrr         uptime    net/
.848      .filesystems  .loadavg   partitions   version   nv/
.849      .interrupts   .locks     pci          bus/      sys/
.cmdline  .iomem        .meminfo   self@        driver/   sysvipc/
.cpuinfo  .ioports      .misc      slabinfo     fs/       tty/
```

partitions

> Diskové oddíly se používají v DOSu, ve Windousech, v Unixech a asi i ve většině ostatních operačních systémů - nevíte někdo, jak je to u Jablek (Apple)? Jedná se o způsob, jak rozdělit jeden fyzický disk na několik menších, nazývaných logiké disky nebo též diskové oddíly. Teď zbývá otázka, jaký to má smysl.
>
> Odpověď lze nalézt v historii osobních počítačů. Na počátku měly osobní počítače (nebo též minipočítače) místo pevných disků pouze disketové mechaniky. Často to fungovalo tak, že na jedné byl operační systém a druhá byla pro uživatelovy programy a data. Teď se musím pochlubit, sám jsem i přes svůj nevelký věk takový počítač viděl :) . No a když pak přišly první winchestery, jak se říkalo pevným diskům, uvědomily si počítačové firmy, že uživatelé by byli příliš zmateni, kdyby měli mít najednou místo diskety o kapacitě 360KB disk o velikosti třeba 10MB. Řešením bylo rozdělit těch 10MB na části o velikosti diskety a uživateli podsunout myšlenku, že se vlatně nic nezměnilo, jen má holt těch disketových mechanik o "něco" více. Za čas si uživatelé na novou skutečnost zvykli a už je tolik nerozptylovalo mít jeden (třeba i 40MB) disk a na něm vše dohromady. No to byly časy.
>
> Doufám, že se vám tahle pohádka líbila (měla přece šťastný konec, ne?), a teď už se vrhněme na to, co všichni čekáte:

```
major minor  #blocks  name
   3     0    9770544 hda
   3     1    3076416 hda1
   3     2    3478072 hda2
   3     3    3076447 hda3
   3     4     136521 hda4
  22     0    1251936 hdc
  22     1     625936 hdc1
```

```
  22     2     625968 hdc2
```

Tak, copak to tam máme. Začínám mít počit, že se lehce opakuji. Přiřazování hlavních (major) a vedlejších (minor) čísel, jejich význam a odpovídají názvy souborů v adresáři */dev* jsme si už probrali, tak jen lehce připomenu, že se jedná o master disky na prvním a druhém kanálu IDE. Vedlejší čísla odpovídají jednotlivým diskovým oddílům. Vedlejší číslo 0 a název bez čísla je pro celý disk (fyzický). Jediný údaj, který zůstává, je velikost oddílu, a to je sloupec nadepsaný jako *#blocks*, tedy počet bloků. Pro zjištění velikosti oddílu už stačí jen vědět, že velikost bloku je 1024 bajtů, neboli 1KB.

Dále zkusíme porovnat získané vlastnosti s výpisem příkazu *df*. Ten zjišťuje informace o prostoru pro uživatelská data. Z rozdílu lze tedy vyčíst, kolik místa na disku zabere administrativa operačního systému. Tak se na to mrkneme:

```
Filesystem  1k-blocks  Used     Available Use% Mounted on
/dev/hda3   3028108    1051276  1915304   36% /
/dev/hda2   3362046    1691800  1496343   54% /home
/dev/hda1   3070400    2156448   913952   71% /mnt/user
```

Chybějící oddíl 4 je využit pro swap. Jak lze briskně spočítat, pro souborový systém je vyčleněno na zhruba 3GB oddílu kolem 53MB místa. To na první pohlednení není vůbec málo, ale jestli se někdy dostaneme k povídání o souborových systémech, uvidíte, že to na druhou stranu není zase tak moc.

Nakonec ještě něco o detailech rozdělení disku. Na začátku disku je umístěna tzv. *partition table* (tabulka rozdělení disku). V té je zaznamenán v každé položce typ oddílu, jeho začátek a konec v rozumné adrese, případně další příznaky a informace. Rozumná adresa je CHS (Cylindr, Hlava, Sektor), nebo jednoduše číslo sektoru pro LBA (Lineární adresování bloků). Fuj, to zní divně. Mimochodem, napište mi, jestli vám moc vadí anglické názvy, pokud ne, asi je začnu ve větší míře používat). No tak zpátky. Prozatím byly všechny informace obecné. Ale svět není jen DOS, a tak podobný systém rozdělení disků používají třeba počítače BSD/SUN nebo IRIX/SGI. Rozdíl je v počtu položek v tabulce diskových oddílů. SUN může mít až osm diskových oddílů, SGI šestnást. Starý dobrý DOS nemá počet omezen. V základní tabulce je místo pro čtyři záznamy, ale jedním z typů oddílu může být tzv. rozšířený oddíl (extended). Ten může opět obsahovat další čtyři oddíly, a tak dále. Jediná drobnost je, že z rozšířených oddílů někdy nejde startovat systém. (Alespoň jsem to slyšel říkat o Windousech.)

pci

Tohle je asi jeden z nejobsáhlejších souborů v adresáři */proc* vůbec. Najdete zde spoustu informací o zařízeních připojených na sběrnici PCI respektive AGP. Jako příklad jen část souboru:

```
...
Bus  0, device   7, function  0:
  ISA bridge: VIA Technologies, Inc. VT82C686 [Apollo Super South] (rev 64).
 Bus  0, device   7, function  1:
  IDE interface: VIA Technologies, Inc. Bus Master IDE (rev 6).
    Master Capable.  Latency=32.
    I/O at 0xc000 [0xc00f].
 Bus  0, device   7, function  2:
  USB Controller: VIA Technologies, Inc. UHCI USB (rev 22).
    IRQ 19.
    Master Capable.  Latency=32.
    I/O at 0xc400 [0xc41f].
...
  Bus  0, device  10, function  0:
   Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8139 (rev 16).
    IRQ 17.
```

```
        Master Capable.  Latency=32.  Min Gnt=32.Max Lat=64.
        I/O at 0xcc00 [0xccff].
        Non-prefetchable 32 bit memory at 0xdc000000 [0xdc0000ff].
...
```

Každé zařízení má v tomto výpisu vyhrazen malý odstavec. Ten začíná adresou zařízení, která se skládá z identifikace sběrnice. V běžných počítačích je jen jedna sběrnice (pozor, ne slot!) PCI, takže je zde *Bus 0*. *Bus 1* potom odpovídá AGP sběrnici a zařízení k ní připojené (často grafická karta :). Číslem u *device* jsou odlišena jednotlivá zařízení. Pokud některé z nich plní více funkcí, je číslo zařízení stejné a rozlišení se provádí pomocí *function*. Dále jsou uvedeny třída, výrobce a název zařízení. To jsou číselné hodnoty, jejichž odpovídající textové popisky lze nalézt v *drivers/pci/classlist.h* pro třídy a *drivers/pci/devlist.h* pro výrobce a jednotlivá zařízení. Následují konfigurační údaje daného zařízení, jako adresa IO paměti, číslo IRQ, obsazení mapované paměti a další.

No a teď bych se měl rozepsat o tom, jak PCI sběrnice a její programování funguje, jak se nastavují jednotlivá zařízení a proč je pro USB přiřazeno přerušení 19. Je mi to líto, ale nerozepíšu. Zatím. Ale protože i mne to opravdu zajímá, tak o to nepřijdete a můžete se těšit, že se k tomuto problému ještě vrátím (a se mnou přijde i zákon! :)

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=819

Co před námi tají /proc XVI.

Verze článku pro čtení: http://root.cz/clanek.php4?id=823

06.09.2001

Dneska začneme písmeno S. Podíváme se hlavně na obsah poměrně obsáhlého souboru slabinfo. No a zbytek si necháme na příště.

Jak velí tradice, nejprve výpis základního adresáře */proc*. Tečka před názvem znamená, že už to máme za sebou:

```
.840      .devices      .kcore    .modules      stat      ide/      acpi/
.842      .dma          .kmsg     .mounts       swaps     irq/
.847      .execdomains  .ksyms    .mtrr         uptime    net/
.848      .filesystems  .loadavg  .partitions   version   nv/
.849      .interrupts   .locks    .pci          bus/      sys/
.cmdline  .iomem        .meminfo   self@        driver/   sysvipc/
.cpuinfo  .ioports      .misc      slabinfo     fs/       tty/
```

self

Tohle bude asi nejjednodušší popis z celého seriálu. *self* je symbolický link na adresář s informacemi pro proces, který vyvolal čtení obsahu adresáře */proc*.

```
sioux@sirion /proc $ls -l /proc/self &
[1] 1063
lrwxrwxrwx  1 root root    64 zář  3 16:53 /proc/self -> 1063
[1]+  Done  ls -l /proc/self
```

Na tomto výpisu příkazu *ls* je to dobře vidět. Na druhém řádku vypíše proces PID procesu, který je spouštěn na pozadí. Potom příkaz *ls* vypíše aktuální hodnotu linku *self*. Provedeme-li to samé ještě jednou, bude hodnota PID na druhém řádku jiná:

```
sioux@sirion /proc $ls -l /proc/self &
[1] 1064
lrwxrwxrwx  1 root root    64 zář  3 16:53 /proc/self -> 1064
[1]+  Done  ls -l /proc/self
```

Vše o tom, co je vlastně obsahem adresáře číslo 1063 (resp. 1064), tedy informace o procesu s daným PID, naleznete hned v prvním dílu našeho seriálu.

slabinfo

Jak již bylo řečeno výše, je tohle jeden ze souborů v adresáři */proc*, kde toho lze nalézt opravdu hodně. Proto bude i ukázkový výpis opravdu hodně dlouhý. Abychom se v něm hned neztratili (ale raději až za chvilku), rozdělím ho na několik dílů, vždy po skupině informací logicky patřících k sobě, a budu je uvádět vždy u vysvětlení patřičné části.

Základem našeho bádání bude, co to vlastně je ten SLAB. Slovo slab v angličtině znamená v tomto případě asi nejlépe *tabulka*. Jedná se o speciální druh alokátoru. Alokátor je mechanismus k získávání bloku adres z celého adresového prostoru. Alokátor může alokovat fyzickou paměť, virtuální paměť, datové bloky na disku nebo třeba místo ve swapu. Prostě zavoláte rutiny alokátoru *alloc_something(velikost_jednotky, počet_jednotek)* a on vám vrátí adresu, od které máte od toho okamžiku dovoleno používat požadovaný rozsah adres a objektů na ně mapovaných. Po použití tuto oblast zase uvolníte použitím jiné rutiny. To je všechno asi hodně obecné, tak nějaký konkrétní příklad. Rutinami alokátoru jsou třeba

funkce z *libc* - *malloc()* a odpovídající *free()*. Případně v C++ jsou to operátory *new* a *delete*. Vždy se tedy jedná o to, spravovat určitý rozsah prostředků a na požádání je co nejlépe rozdělovat mezi zájemce.

Postupně se přišlo na to, že operace alokování jsou potřeba v operačních systémech na různých místech (jak bylo nastíněno výše). Proto se návrháři pro zjednodušení rozhodli vytvořit obecný mechanismus alokátoru použitelný potom pro jekýkoliv konkrétní úkol. Testováním se ukázalo, že tato společná komponenta má v celém operačním systému nezanedbatelný vliv na souhrnný výkon. Proto bylo vhodné zjistit, jak by šlo alokátor optimalizovat a tím zlepšit výkon celého operačního systému. Práci alokátoru (pro konkrétní objekt) lze rozdělit na dvě části. Prvním úkolem je získat potřebnou oblast (předpokládejme jako model paměťový alokátor - tedy oblast paměti). Jakmile je jasné, který úsek bude použit, začíná druhá fáze. Zde je nutno objekt inicializovat. V případě paměti to nemusí být na první pohled zřejmé. Prostě uživateli předám adresu paměti, kterou chtěl, a hotovo. Tak jednoduché to ale zase není. Jádro si musí udržovat přehled o tom, které části paměti jsou obsazeny, kým a za jakým účelem. To v tomto případě platí jak pro skutečnou fyzickou paměť, tak pro oblast ve virtuální paměti, kam bude nový úsek zamapován. Administrativa kolem údržby těchto informací může být dosti složitá. U specializovaných objektů jako objekty souborů, adresářových cest a dalších je čas strávený přípravou objektu zase mnohem delší než samotná alokace prostoru. Naopak při dealokaci by měl slušný systém (takový, který aspiruje na vyšší bezpečnostní třídy - jako C2 dle Orange Book) splňovat vlastnost znovupoužitelnosti, a tedy objekty před vrácením nulovat. I pokud tomu tak není, bez dalších opatření musíme stejně při alokaci celý objekt přepsat, nebo se minimálně nespoléhat na to, že má nějaký konkrétní obsah.

A tady se dostáváme k jádru věci. Je možné udržovat speciální (tzn. oddělený) alokační mechanismus (respektive jeho jednu část) a objekty kešovat. Pokud bude někdo chtít stránku paměti a pak ji zase vrátí, je dobré vědět, že to byla stránka paměti pro uživatele a, až zase bude nějakou chtít, znovupoužít tuhle stránku. Tím se dají využít části (minimálně) informací zpracovávaných při administrativě a alokace může být rychlejší. Samozřejmě, že i zde může být požadavek na znovupoužitelnost, ale pokud například generuji síťové pakety, je zde řada informací, která je pro všechny (nebo skupinu) stejná a je bezpečné ji ponechat nesmazanou. Ostatní (důvěrné) informace se samozřejmě promáznou.

Mechanismus alokace a dealokace objektů se potom změní na následující:

Alokace:

```
if (požadovaný objekt je v cache) {
  vyber objekt z cache
} else {
  alokuj paměť
  vytvoř (inicializuj) objekt
}
```

Dealokace:

```
vrať objekt do cache (nic se neuvolňuje)
```

V Linuxu existují tyto odělené cache: vm_area, mm, names, filp, dquot, bh, sigact a dentry. Postupně se dostanemek tomu, pro co se používají. Nejprve, jak jsou tyto cache v Linuxu organizovány. Každá cache se skládá z několika "slabů" (tedy tabulek). Ty bývají zarovnány nejlépe na velikost stránky nebo podobnou vhodnou konstantu a obsahují pevně definovaný počet objektů daného typu. Každý slab může být v jednom ze tří stavů. Zcela prázdný, kdy z

něj není alokován žádný objekt, částečně využit, kdy je alokováno méně objektů, než je celková kapacita, a nakonec samozřejmě plný, když jsou alokovány všechny jeho objekty. Pokud je potřeba alokovat nový objet, použije se nejprve slab s částečným zaplněním, pokud takový není, tak zcela prázdný. Naopak při uvolňování se uvolní objekt ze slabu, který je poloprázný (nejlépe s nejmenším počtem objektů), a pokud není, tak ze zcela zaplněného slabu. To má jediný důvod. Jakýkoli obsazený slab je vlastně aktivní stránka paměti. A čím méně je takovýchto stánek použito, tím je systém alokátoru efektivnější (a pravděpodobně také rychlejší).

Teď si tedy vysvětlíme použití jednotlivých typů cache. *vm_area* slouží pro oblasti viruální paměti, *mm* pro fyzické stránky, *names* pro absolutní jména souborů kopírovaná do adresového prostoru jádra, *filp* pro informace o (otevřených) souborech, *dquot* pro (diskové) quoty, *bh* pro buffery k diskovým operacím, *sigact* je pro administrativu kolem signálů (rozhraní sigaction) a *dentry* je pro adresářovou cache (vyhledávání cest k souborům).

Tak to bylo letecky pro odstrašení nepřátel :) a teď k jednotlivým položkám výpisu našeho souboru. Nejprve nejjednodušší první řádek. Verze je zakompilovaná napevno. Za ní se mohou objevit ještě atributy, a to *(SMP)* pro víceprocesorové systémy, kde je o něco více informací, a potom *(statistics)*, pokud si explicitně zapnete generování podrobných informací. To lze provést v souboru *mm/slab.c*, kde na řádku číslo 89 (v 2.4.9) je definice makra preprocesoru STATS. Je-li hodnota 0, nic se negeneruje, je-li 1, jsou generovány další podrobnější informace. V souboru je v tomto místě i další popis ohledně nastavení ladění a podrobnosti výpisů.

```
slabinfo - version: 1.1
...
```

Od dalších řádek jsou informace pro jednotlivé druhy cache. Je zde vždy název a potom následující parametry: počet aktivních objektů, počet celkových objektů, velikost objektu, počet aktivních slabů, celkový počet slabů a počet stránek (4KB) na jeden slab u tohoto typu:

```
...
kmem_cache             54     78    100    2    2    1
...
```

Myslím, že pro dnešek toho bylo dost, takže jednotlivé typy slabů a jejich význam si necháme na příště. Pokud by měl někdo zájem o podrobné informace o teorii a implementaci slabu v SunOS 5.4, může si z mé stránky stáhnout [http://www.stud.fee.vutbr.cz/~xkaspa06/root/slab.ps] tento dokument.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=823

Co před námi tají /proc XVII.

Verze článku pro čtení: http://root.cz/clanek.php4?id=829

11.09.2001

Minule jsme začali rozebírat soubor informací o SLABu - kešovacím alokátoru objektů v Linuxu. Dneska se dostaneme k významu jednotlivých položek, tedy typů objektů a jim odpovídajících keší.

Tak a tradice už bylo dost. Protože už jsme téměř u konce výpisu základních souborů, zde jsou soubory, které nám ještě zbývají:

```
slabinfo   acpi/     irq/      tty/
stat       bus/      net/
swaps      driver/   nv/
uptime     fs/       sys/
version    ide/      sysvipc/
```

slabinfo - pokračování

Minule jsme probrali úvod a hlavičku, teď jsou na řadě jednotlivé řádky souboru a jejich význam. Pokud to bude možné, budu používat termín *cache*, pokud bude nutné skloňovat nebo něco podobného, uchýlím se k české alternativě *VP = vyrovnávací paměť* stejného významu. Pokud existuje nějaký český překlad (jiný než rychlá vyrovnávací pamět), dejte mi vědět, rád se poučím.

```
...
kmem_cache            54     78    100    2    2    1
...
```

První cache je určena pro ostatní cache. Jednoduše řešeno je to seznam ostatních VP. Jak vidno, vše sedí, neboť ve výpise je právě 54 řádků a tedy 54 aktivních objektů dle prvního sloupce. Tato cache zabírá dva slaby (jeden slab na stránku), tedy 8KB.

```
...
tcp_tw_bucket          0      0     96    0    0    1
tcp_bind_bucket        3    113     32    1    1    1
tcp_open_request       0      0     64    0    0    1
inet_peer_cache        0      0     64    0    0    1
ip_fib_hash            5    113     32    1    1    1
ip_dst_cache           0      0    160    0    0    1
arp_cache              0      0    128    0    0    1
...
```

Tento blok VP je určen pro objekty síťového rozhraní, jedná se o VP pro síťové spojení v různých stavech protokolu. *tcp_tw_bucket* je pro sokety ve stavu TIME_WAIT, nastane po uzavření soketu a je určen pro ošetření paketů, které jsou v tu chvíli ještě na síti. *tcp_bind_bucket* slouží pro znovupoužití portů, *tcp_open_request* je určen pro informace o otevřeném TCP spojení, *inet_peer_cache* pro směrovací informace, *if_fib_hash* pro forwarding (FIB - Forwarding Information Base), *ip_dst_cache* pro cache směrovacích cest a *arp_cache* pro překlad adres pomocí ARP.

```
...
blkdev_requests      256    280     96    7    7    1
dnotify cache          0      0     20    0    0    1
file lock cache        5     42     92    1    1    1
fasync cache           1    202     16    1    1    1
...
```

blkdev_requests je VP pro požadavky na přístup k blokovým zařízením, *dnotify cache* je VP pro požadavky na oznamování změn v souborech. Existuje nový, ješte nedokumentovaný (v man na RH 7.0 :) typ volání *fcntl()*, a to *F_NOTIFY*. Ten umožňuje reagovat na změny vytvoření, modifikace, přístupu nebo smazání souborů. *file lock cache* je určena pro struktury s informacemi k souborovým zámkům, *fasync cache* je pro asynchronní požadavky na bloková zařízení.

```
...
uid_cache               4    113     32    1    1    1
skbuff_head_cache      40     48    160    2    2    1
sock                   43     45    832    5    5    2
sigqueue                1     29    132    1    1    1
cdev_cache            533   1298     64   13   22    1
bdev_cache           6044   8142     64  105  138    1
mnt_cache              10     59     64    1    1    1
inode_cache         87793  88048    480 11006 11006    1
dentry_cache        65537  68940    128 2298 2298    1
filp                  966   1000     96   25   25    1
names_cache             0      2   4096    0    2    1
buffer_head         41001  47280     96 1172 1182    1
mm_struct              43     48    160    2    2    1
vm_area_struct       1776   1888     64   31   32    1
fs_cache               42     59     64    1    1    1
files_cache            42     45    416    5    5    1
signal_act             44     48   1312   15   16    1
...
```

uid_cache obsahuje informace o uživateli (počet procesů, otevřených souborů), *skbuff_head_cache* je VP pro síťové pakety, *sock* pro datové struktury soketů, *sigqueue* je určena pro fronty signálů, *cdev_cache* a *bdev_cache* jsou pro informace o znakových respektive blokových zařízeních, *mnt_cache* pro informace o existujících připojeních pro viruální soub. systém (VFS)., *inode_cache* pro struktury k i-uzlům, *dentry_cache* pro informace k vyhledávání adresářových cest, *filp* pro informace o otevřených souborech, *names_cache* pro buffery pro zpracování cest a názvů souborů, *buffer_head* pro struktury blokové cache, *mm_struct* pro informace o fyzických stránkách, *vm_area_astruct* pro informace o alokovaných oblastech v LAP (logickém adresovém prostoru), *fs_cache* je pro informace o umístění souboru v adresářové struktuře, *files_cache* pro tabulku otevřených souborů v jádře, *signal_act* pro informace o signálech v jádře.

```
...
size-131072(DMA)        0      0 131072    0    0   32
size-131072             0      0 131072    0    0   32
size-65536(DMA)         0      0  65536    0    0   16
size-65536              1      1  65536    1    1   16
...
size-64(DMA)            0      0     64    0    0    1
size-64              1083   1711     64   27   29    1
size-32(DMA)            0      0     32    0    0    1
size-32              9614  11639     32   99  103    1
```

A tohle už je vážně poslední skupina. Jedná se obecnou cache pro paměťové objekty o patřičných velikostech. Položky s *(DMA)* jsou navíc určeny pro DMA přenosy. To znamená na některých platformách jistá omezení. Například na x86 musí být tato paměť fyzicky umístěna v rozsahu 0-16MB.

Teď zase něco málo přeskočíme a vyřídíme nejprve poslední dvě položky, protože jsou jednodušší. Na *stat* a *swaps* se dostane příště.

uptime

Tohle je také velmi jednoduché. Jsou zde dvě hodnoty. První udává vlastní *uptime*, tedy jak dlouho je již systém spuštěn. Druhá udává čas strávený ve stavu *idle* - tedy celkovou dobu, kdy nebylo nic na práci. Následuje příklad:

1712.17 1689.83

Jednotkou jsou v obou případech sekundy. Příkaz *uptime* ve stejnou dobu tedy vypíše:

```
 8:41am  up 28 min,  3 users,  load average: 0.15, 0.09, 0.03
```

A vskutku, 1712 sekund je 28 minut. Vzhledem k tomu, že psaní textu není moc náročná činnost (alespoň pro můj počítač :), je hodnota *idle* také 28 minut. Nicméně je vždy menší než celkový čas.

Teď ještě jak jsou hodnoty získány. To je prosté. V Linuxu existuje konstatnta HZ (na Intel x86 #definovaná jako 100, u Aplhy je to 1024 nebo 1200). Ta je použita pro nastavení časovače při startu systému a udává počet přerušení časovače za sekundu. Hlavním úkolem obsluhy přerušení časovače je potom při každém tiku inkrementovat proměnnou jménem *jiffies*, která potom udává počet tiků časovače od startu systému. Jde o *unsigned long* a je to veřejně přístupná proměnná (pro jádro). Hodnota *uptime* v souboru je tedy jednoduše *jiffies/HZ*. (Obecně to sice tak jednoduché není, protože je třeba ošetřit různá přetečení rozsahů a podobně, ale smysl zůstává stejný.) Druhé číslo vznikne podobně, ale čítač je inkrementován pouze při nečinnosti.

version

Opět jedna lahůdka. Informace zde skoro nepotřebují komentář, uznejte sami:

```
Linux version 2.4.9 (root@sirion) (gcc version 2.96 20000731 \
(Red Hat Linux 7.0)) #1 Út zář 4 19:28:22 CEST 2001
```

Jedná se o informace o verzi jádra Linuxu. Kromě vlastní verze (2.4.9) je zde zaznamenán i čas překladu, kdo ho prováděl a jaký byl použit překladač. Informace jsou natvrdo vloženy do jádra při překladu (viz. soubor init/version.c)

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=829

Co před námi tají /proc XVIII.

Verze článku pro čtení: http://root.cz/clanek.php4?id=843

18.09.2001

Dneska dokončíme poslední ze souborů v základním adresáři. Konkrétně se jedná o soubory "stat" a "swaps". Potom se zaměřím na úplné vymýcení všech bacilů, které se na mě poslední dobou nějak domluvily, a po malé přestávce na zotavenou budeme pokračovat v jednotlivých adresářích.

Dnes naposled se můžete podívat na výpis celého adresáře */proc*. Po dnešku už budou všechny soubory otečkovány, a tak nebude mít smysl je vypisovat stále dokola:

```
.840      .devices      .kcore    .modules      stat      ide/      acpi/
.842      .dma          .kmsg     .mounts       swaps     irq/
.847      .execdomains  .ksyms    .mtrr        .uptime    net/
.848      .filesystems  .loadavg  .partitions  .version   nv/
.849      .interrupts   .locks    .pci          bus/      sys/
.cmdline  .iomem        .meminfo  .self@        driver/   sysvipc/
.cpuinfo  .ioports      .misc     .slabinfo     fs/       tty/
```

stat

Soubor *stat* je jednou ze statistik, které jádro nabízí. Naleznete zde informace o celém systému, určené hlavně pro zběžnou kontrolu jeho stavu:

```
cpu  937 0 617 51832
cpu0 937 0 617 51832
page 56065 6655
swap 1 517
intr 118148 53386 3092 0 0 0 0 2 0 0 50074 0 0 6460 0 5134 0 ....
disk_io: (3,0):(5136,3530,112131,1606,13310)
ctxt 104742
btime 1000281282
processes 735
```

První dvě položky informují o stavu procesoru. Na jednoprocesorové mašině jsou identické, jinak první udává součet pro celý systém a následují jednotlivé procesory. Jednotlivé položky u procesorů znamenají (v tomto pořadí) časy procesoru strávené v režimu: user, nice, system, idle. To znamená, že můj procesor v době výpisu použil 937 jiffies pro běh uživatelského režimu, 617 pro běh v režimu jádra (systémovém) a 51832 jiffies promrhal nicneděláním. 0 na druhém místě říká, že zatím neběžel žádný program, který má prioritu sníženou pomocí volání *nice()*. Na druhou stranu, pokud si vzpomínáte na povídání o procesu jádra pro obsluhu "softwarových" přerušení (ksoftirqd_CPU00) v dílu číslo 10 (o výpočtu loadavg), víte, že právě tento proces má nice rovnu 19. (Ano, skutečně nejmenší možná priorita).

Nejprve slíbené zdůvodnění, jak je možné, že tak důležitý proces má tak malou prioritu. Respektive neřeknu vám, proč má tak malou prioritu, když obsluhuje (kromě jiného) časovač, a ten potřebuje stejně pravidelně vykonávat, co je třeba. Možná odpověď je, že když je tento proces potřeba, zajistí se jeho probuzení jinými prostředky než standardním plánovačem, a bylo by tedy zbytečné, aby ho plánovač sám od sebe spouštěl. A jak to tedy funguje a jaké jsou ty jiné prostředky? Pokud je vyvoláno softwarové přerušení, existují dvě možnosti. Pokud jsme zrovna v té době v obsluze přerušení nebo ve funkci dolní poloviny handleru přerušení, je náš požadavek proveden po návratu z nich. Pokud tomu tak není, je vyvolána funkce *wakeup_softirqd*, která se pokusí probudit démona *ksoftirqd*. To provede funkce *wake_up_proces()*. Ta nakonec probublá do funkce, kde se požadovaná úloha zařadí

na začátek fronty úloh připravených ke spuštění. Je možné požadovat okamžité přeplánování (parametr synchronous - jako v tomto případě), kdy se prostě zavolá plánovač ihned, nebo nechat dokončit aktuálně běžící úlohu a naši vybrat až při regulérním přeplánování (z důvodů vypršení časového kvanta).

Tak a teď budeme pokračovat naším výpisem. Další položkou je *page*. Dvě čísla znamenají počet stránek načtených a zapsaných na disk. Ne pro swap, ale například pro soubory mapované do paměti a vůbec všechny datové stránky. Podobně *swap* udává nejprve počet paměťových stránek přečtených ze swapu a další položka je počet stránek zapsaných na swap. Opakovaná operace se stejnou stránkou (čtení) nebo opakovaný zápis jsou počítány zvlášť.

Následuje blok informací o přerušeních. První číslo je počet všech obsloužených přerušení, za ním následuje statistika pro každé přerušení zvlášť (lze srovnat s informacemi v souboru *interrupts*).

Další na řadě je položka s informacemi o diskových operacích. Pro každé diskové zařízení (ne oddíl, ale celý disk) je zde informace v tomto formátu: (MAJOR, MINOR):(IO_OPERACE, READ_OPERACE, READ_BLOCK, WRITE_OPERACE, WRITE_BLOCK). MAJOR a MINOR jsou hlavní a vedlejší číslo disku. Následuje celkový počet požadavků na zařízení, počet požadavků na čtení, počet přečtených bloků, počet požadavků na zápis a počet zapsaných bloků (v jednom požadavku se často zapisuje několik bloků).

Poslední tři položky jsou již jednoduché. *ctxt* udává počet přepnutí kontextu od startu systému. Přepnutí kontextu je (zjednodušeně) změna prováděného procesu. Jeden proces se z procesoru odstraní a nahradí se jiným, který může procesor po jistou dobu používat (tato doba je právě ono časové kvantum). *btime* je čas nastartování systému a je uveden v počtu vteřin od 1.ledna 1970. Poslední položka udává počet provedení operace *fork()*, tedy počet vytvořených procesů. Vzhledem k tomu, že po operaci fork (respektive funkci *do_fork()* v jádře) nemusí vzniknout proces s novým PID, neodpovídá toto číslo poslednímu použitému PID. (Pomocí příznaku CLONE_PID lze vytvořit dva procesy se stejným PID - překvápko :-)

swaps

Tak a poslední soubor. Tento nám říká, jak jsou v systému nakonfigurovány odkládací prostory:

```
Filename                Type            Size    Used    Priority
/dev/hda4               partition       130748  26320   -1
/home/swap.64MB         file            65528   0       -2
```

Nejprve je uvedeno jméno odkládacího prostoru. Tím může být buď disková oblast, nebo normální soubor na libovolném souborovém systému. Typ udává právě druhá položka. Pak tu máme celkovou velikost a využité místo. Priorita je určena k tomu, aby bylo možno odlišit jednotlivé oblasti (pokud jich je víc) dle jejich výkonu. Například odkládací prostor na samostatném (a rychlém) disku bude mít nejlepší prioritu, oddíl na stejném disku jako systém bude mít menší prioritu a swap v souboru bude mít nejmenší prioritu. Potom bude swapovací algoritmus používat nejprve swap na samostatném disku, pak na diskovém oddílu a teprve nakonec, když už jinde nebude místo, začne používat soubor. POZOR: Platí, že lepší je ten, kdo má vyšší číslo. Pro nastavení se používá příkaz *swapon -p priorita swap* :

```
swapoff /home/swap.64MB
```

```
swapon -p 1 /home/swap.64MB

/dev/hda4               partition       130748  33564   -1
/home/swap.64MB         file            65528   168     1
```

Pro zvýšení výkonu nemá význam mít více odkládacích oblastí na jednom zařízení, to lze doporučit jen pro případ, že krátkodobě potřebujete více dostupné paměti (nebo chcete někomu ukázat, jak vypadá výpis souboru /proc/swaps :). Na druhou stranu umístění (primárního) swapovacího oddílu na jiný disk (v lepším případě rychlejší) je vždy přínosem (lepší paralelismus).

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=843

Co před námi tají /proc XIX.

Verze článku pro čtení: http://root.cz/clanek.php4?id=860

27.09.2001

Po menší odmlce jsme se dneska doškobrtali přes všechny soubory až k adresářům. Takže hrr na ně a začneme pěkně podle abecedy adresářem pro ACPI.

Opět pro představu o tom, co nás čeká a co už máme za sebou, uvedu výpis adresářů, které lze nalézt na mém počítači v */proc*:

```
acpi/      ide/     sys/
bus/       irq/     sysvipc/
driver/    net/     tty/
fs/        nv/
```

Připomínám, že přítomnost i obsah jednotlivých adresářů, stejně jako souborů, je silně závislá na verzi jádra a také na HW i SW konfiguraci počítače, na němž je výpis proveden. Často jde "pouze" o to, zda jsou, či nejsou zavedeny potřebné moduly. Příkladem adresářů, které nemá (nemusí mít) každý, jsou *nv/* nebo *acpi/*.

acpi

> Tak. První adresář, do kterého se ponoříme, je */proc/acpi/*. Jak z názvu patrno, shromažďuje informace a konfiguraci implementace ACPI v linuxovém jádře. A co že to ACPI je? Jedná se o konkurenta (nebo pokračovatele - to záleží na názoru) standardu APM. Podobně jako APM (Advanced Power Management) je určen pro ovládání počítače při nečinnosti za účelem úspory energie (elektrické). ACPI (Advanced Configuration and Power Interface) jde ovšem dále než APM a kromě správy napájení (power management) definuje navíc způsob konfigurace počítače a jeho součástí (jako například PnP). Dalším výrazným rozdílem je, že zatímco APM je implementován v (APM) BIOSu počítače a veškeré činnosti se implementují jako služby (APM) BIOSu, ACPI je navržen jako standard co nejvíce nezávislý na konkrétním HW či operačním systému. Toho bylo dosaženo převedením co nejvíce činností (jak z oblasti správy napájení, tak konfigurace) z BIOSU (nebo obecně firmware jednotlivých částí HW - mainboard a periferie) pod kontrolu operačního systému, aby bylo možné lépe (relativně jednodušeji a hlavně pružněji) manipulovat s jednotlivými požadavky. Je podstatně jednodušší modifikovat nebo vyměnit ovladač v jádře (samozřejmě nejlépe modul) nebo dokonce uživatelský proces, než upgradovat (APM) BIOS.
>
> Osobně nyní ACPI používám v podstatě pouze kvůli tomu, že se mi nepodařilo přemluvit jádro, aby mi vypínalo počítač za použití APM. Když už jsem se tedy rozhodl pro ACPI (jako jedinou variantu kromě ručního vypínání tlačítkem), vyzkoušel jsem i ovladač pro idle režim procesoru. Dále je v jádře obsažena podpora ovládání počítače přes tlačítka na předním panelu (mně nefunguje), což znamená, že po stisku tlačítka RESET se Linux přepne do úrovně 6, což způsobí klasický restart (ale bez jakéhokoliv příkazu). Obdobně potom prý funguje i POWER, který počítač přes runlevel 0 vypne. To zní opravdu zajímavě, ne? Možná to ale bude fungovat spíše na noteboocích, kde jsou takové věci přece jen propracovanější. Podobně to je i s podporou baterií (zjišťování stavu a tak) nebo teplotních čidel.
>
> Linux používá implementaci tohoto standardu od firmy Intel, která se jmenuje ACPI CA. Tedy ACPI Component Architecture. Celý systém je podobně jako u APM složen z ovladače v jádře a uživatelského procesu - démona (acpid). Pro zájemce o další teorii je na obvyklých

místech (součást distribuce, www.linuxdoc.org) k dispozici ACPI-HOWTO, případně stránky projektu [http://phobos.fs.tum.de/acpi] ACPI4Linux nebo odpovídající stránky u [http://developer.intel.com/technology/iapc/acpi] Intelu. V plánu je, že by ACPI mělo časem (nejen v Linuxu) plně nahradit APM, ale jak to dopadne, zatím nikdo neví. Tak to byla teorie a teď už k tomu, co vlastně adresář *acpi/* obsahuje:

```
dsdt  event  info  processor/
```

Obsah je samozřejmě závislý na konfiguraci, podotýkám, že mám nakonfigurovanou pouze obecnou podporu ACPI, vypínání počítače a správu procesoru. Soubor *dsdt* obsahuje výpis stejnojmenné tabulky. Tady bych nerad zabředával do podrobností, protože specifikace ACPI (v současnosti verze 2) je dosti složitá a obsáhlá a já jsem zatím neuznal za vhodné se tím pročítat. Takže jen stručně. ACPI definuje vlastní jazyk pro popis systému (AML - ACPI Machine Language). Ten je použit pro popis konkrétního systému, jeho služeb a vlastností. DSDT tabulka (Differentiated System Description Table) obsahuje právě AML (byte)kód popisu systému. Využívá se normálně přes utility dodávané spolu s ACPI démonem, které jsou schopny jeho obsah převést do něčeho (lehce) srozumitelného. Malý odstrašující příklad dekódovaného obsahu této tabulky (pomocí utility acpidisasm) jsem přece jen přiložil:

```
00000000: Scope _PR_ (\_PR_)
00000007:   Processor CPU_ (\_PR_.CPU_)
00000014:     0x00
00000015:     0x00000000
00000019:     0x00
0000001a:   Processor CPU1 (\_PR_.CPU1)
00000027:     0x01
00000028:     0x00000000
0000002c:     0x00
0000002d: Name _S0_ (\_S0_)
00000033:   Package
00000035:     0x04
00000036:     ZeroOp
00000037:     ZeroOp
```

Dalším souborem je *event*. Ten slouží pro komunikaci ovladače s ACPI démonem. Pokud například zmáčknete tlačítko RESET (jak bylo vysvětleno výše), vygeneruje ovladač událost a zapíše ji do tohoto souboru. Démon potom zjistí, co se stalo, a provede přepnutí runlevelu. Stejně jsou obsloužena teplotní čidla a nebo informace o docházející baterii.

Posledním souborem zde je *info*. Jediný rozumě čitelný soubor:

```
ACPI-CA Version:          20010615
Sx States Supported:      S0 S1 S4 S5
```

Moc informací tady ovšem nenajdeme. Pouze informace o verzi (ovladače v jádře) a podporovaných *S stavech*. Těch je celkem šest a mají následující význam. Stav *S0* je stav, kdy systém pracuje naplno. Stavy *S1* až *S4* jsou určeny pro různé režimy "spánku". *S1* je definován jako stav s nízkou dobou probuzení. Jsou v něm zachovány hodnoty celého systému kromě procesorových (L1 a L2) cache pamětí, které jsou vyprázdněny (invalidovány) pomocí speciální instrukce (WBINVD). Ve stavu *S2* mohou být ztraceny všechny informace kromě stavu hlavní paměti (RAM). *S3* je obdobný jako *S2*, pouze některá zařízení mohou být přepnuta do stavu s ještě nižší spotřebou energie. *S4* je stav s nejmenší spotřebou energie, kdy je systém ještě možno vrátit do stavu před usnutím. Předpokládá vypnutí všech možných zařízení včetně uchování jejich kontextu pro pozdější obnovení. Poslední stav *S5* je určen k softwarovému vypnutí systému. Neprovádí žádné uchování kontextu. No, jak vidíte, podporovány jsou pouze stavy 0, 1, 4 a 5. To v praxi znamená normální provoz, provoz s omezenou spotřebou energie, hibernaci a vypnutí systému. Na

druhou stranu nečekejte, že to takhle bude fungovat. ACPI je ve vývoji a nemůžete tudíž chtít zázraky.

Adresář *processor* obsahuje další podadresáře pro jednotlivé dostupné procesory (respektive spíše patice). Jsou číslovány od 0 a my si probereme obsah právě adresáře 0, kde je osazen procesor. V druhé patici není zatím nic, a tak jsou všechny hodnoty nulové. Pro každý procesor jsou k dispozici dva soubory: *info* a *status*. První z nich zatím neobsahuje žádné zajímavé informace, ten druhý je na tom o něco lépe. Zde je obsah souboru *status*:

```
Bus Mastering Activity:  50703b9d
C-State Utilization:     C1[2095875] C2[2004540] C3[0]
```

Nejprve několik slov ke stavům *Cx*. V tomto případě se jedná o stavy procesoru. Ve stavu *C0* je plně funkční, i když specifikace ACPI umožňuje kontrolovat jeho pracovní frekvenci (snížit v případě, že není potřeba velký výkon), stav *C1* je pro procesory řady Intel x86 navozen instrukcí *HLT*. Pro tento stav je charakteristické, že nepotřebuje žádnou podporu základní desky nebo jiných zařízení. Probuzení se děje při příchodu přerušení. V tomto stavu je na procesoru, aby uchoval svůj vnitřní stav (obsahy registrů atd.). Stav *C2* je rozšířením předchozího pro ještě větší úsporu energie. Zde již může být nutná podpora v čipsetu základní desky. Poslední je stav *C3*. Neočekávaně umožňuje opět větší úsporu energie. Tento stav (stejně jako *C2*) již není dle ACPI specifikace povinný.

První řádek v souboru *status* obsahuje příznak aktivity na sběrnici. Ten je použit pro přechod mezi stavy procesoru. Konkrétně procesor musí opustit stav *C3*, pokud dojde k aktivitě na sběrnici. U druhého řádku udávají jednotlivá čísla počet přepnutí do daného stavu.

Na závěr ještě jeden adresář (i když mimo pořadí). Řeknu k němu jen pár slov, ať se nezdržujeme někdy příště.

driver

Tento adresář se objevil teprve nedávno (respektive "nedávno" berte trochu s rezervou). Byl v jádře 2.4.5 (zopakuji, že nyní používám 2.4.9), ale nějak si na něj nepamatuji o moc dříve. A teď už k tomu, co tento adresář obsahuje. Odpověď je prostá: NIC. Přesvědčte se sami:

```
sioux@sirion ~ $ls -la /proc/driver/
celkem 0
dr-xr-xr-x    2 root     root            0 zář 20 18:24 .
dr-xr-xr-x   51 root     root            0 zář 20 18:18 ..
```

Ano, takhle vypadá NIC, tedy konkrétně prázdný adresář. Kdyby měl být tenhle adresář prázdný, tak by tu ale nebyl. Takže je asi určen pro nějakou konfiguraci, která na mém počítači není. Při prohledání zdrojových textů je možno zjistit, že je určen pro informace některých ovladačů. Konkrétně se jedná o následující hardware:

- diskové pole s řadičem Compaq SMART2 Controller (nebo že by HP :-)
- bezdrátové síťové karty Cisco/Aironet řady 4500 a 4800
- PCMCIA bezdrátové síťové karty Raylink/WebGear

No a vzhledem k tomu, že uvedené kousky jsou spíše vzácnější, můžeme na tento adresář s klidem zapomenout (alespoň většina). Pokud někdo uvedený HW vlastní a vzpomene si na nás ostatní, může do diskuze zaslat obsah odpovídajícího adresáře (za což mu budeme velice vděční), a to nejlépe i s dalšími vhodnými informacemi (verze jádra nebo něco o HW).

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=860

Co před námi tají /proc XX.

Verze článku pro čtení: http://root.cz/clanek.php4?id=879

11.10.2001

Opět malé výročí. Dneska jsme se dopracovali až k dvacátému dílu, což už je myslím slušné. A co nás čeká dnes? Nejprve ještě několik slov k adresáři "drivers", který jsem minule nedopatřením dost odbyl, pak budeme pokračovat adresářem "bus", a aby toho po malém zpoždění nebylo málo, profrčíme ještě i "fs".

Opět pro představu o tom, co nás čeká a co už máme za sebou, uvedu výpis adresářů, které lze nalézt na mém počítači v */proc*. Otečkované jsou ty adresáře, které už jsme prošli.:

```
.acpi/      ide/     sys/
 bus/       irq/     sysvipc/
.driver/    net/     tty/
 fs/        nv/
```

driver

Minule jsem tvrdil, že ovladače, které používají tento adresář, jsou jen tři. Jak se ukázalo, není to pravda. Existuje několik cest, jak provést registraci souboru (podadresáře) v adresáři */proc* (resp. */proc/drivers*). Abych alespoň částečně osvětlil, jak se mi tohle nedopatření stalo, popíšeme si stručně jednotlivé způsoby.

Pro informační soubory určené pouze pro čtení je možno použít funkci *create_proc_read_entry()*. Dále existuje obecná funkce pro vytvoření souboru *create_proc_entry()*. Aby to nebylo tak jednoduché, v jádrech řady 2.0 a 2.2 se používala funkce s obdobným významem jménem *proc_register()* (respektive *proc_register_dynamic()* v řadě 2.0). Pro odstranění jsou potom určeny funkce *remove_proc_entry()* a *proc_unregister()*. Pro speciální soubory lze ještě využít zjednodušené varianty předchozích, a to *proc_symlink(), proc_mknod(), proc_mkdir()*.

Všechny tyto funkce operují nad strukturou *proc_dir_entry*, která obsahuje všechny potřebné informace o libovolném prvku v adresáři */proc*. Funkce se liší tím, že některé pouze vytvoří tuto strukturu a je na vás, ji vyplnit (*proc_create_entry()*), jiné si potřebné informace vezmou jako parametry a vše vyřeší za programátora *proc_create_read_entry()*. A dalším je třeba předat odkaz na už vytvořenou a vyplněnou strukturu a ony ji pouze zaregistrují v souborovém systému */proc* (funkce pro jádra <2.4.x).

U ovladačů pro některý z předdefinovaných adresářů se předpokládá tento postup:

```
my_proc_entry = create_proc_entry("muj_skvelej_ovladac",
                                  S_IFDIR|my_perm, proc_root_driver);
```

Tohle volání vytvoří adresář s názvem *"muj_skvelej_ovladac"* a s právy dle proměnné *my_perm*, umístěný v adresáři *proc_root_driver*. Že se vytvoří adresář, udává příznak *S_IFDIR*. Důležité je právě použití exportované proměnné pro adresář */proc/driver*, tedy *proc_root_driver*. Stejně tak existuje například proměnná *proc_root_bus* nebo *proc_root_net*.

Protože jsem předpokládal (uznávám dosti naivně), že se takto budou chovat všechny ovladače, prováděl jsem jejich vyhledání právě grepem na *proc_root_driver*. Výsledek jste

viděli minule. Když se ozvaly hlasy (tímto děkuji jejich majitelům za upozornění), že něco chybí, podíval jsem se hlouběji, jak to dělají například ovladače pro *RTC*, inzerovaný v diskusi. A co jsem zjistil?

```
create_proc_read_entry ("driver/rtc", 0, 0, rtc_read_proc, NULL);
```

U ostatních ovladačů je použita právě druhá zmíněná metoda. Při použití této funkce se zadá přímo název souboru a funkce, která vygeneruje data při požadavku na čtení patřičného souboru. Asi je jasné, že můj grep neměl šanci. Poněkud pozoruhodné je použití složitější cesty (přes několik adresářů. Tato funkce bere jako svůj kořenový adresář to, co je zadáno jako poslední parametr. Pokud je to NULL, je kořenem */proc*. V názvu je možno jako v tomto případě zadat relativní cestu k souboru a funkce sama vytvoří chybějící adresáře (se standardními právy a nastaveními).

Po tomto malém průzkumu se tedy podívejme, které další ovladače nyní nalezneme pomocí grepu nad *"driver/"*:

```
char/rtc.c                  RTC - Real Time Clock
char/h8.c                   Hitachi H8/337 microcontroller
char/nvram.c                NVRAM (neboli CMOS)

sound/via82cxxx_audio.c     AC97 sound codec
sound/emu10k1/main.c        Creative EMU10K1 sound card
sound/ite8172               ITE IT8172G sound card
sound/nec_vrc5477.c         AC97 codec NEC Vcr5477 na čipech MIPS

usb/uhci.c                  Universal Host Controller Interface pro USB

net/pcmcia/ray_cs.c         Raylink Wireless LAN

media/radio/radio-typhoon.c Typhoon Radio Card
```

Osobně můžu sloužit pouze RTC a CMOS pamětí. Ostatní HW nemám k dispozici. Protože u RTC a CMOSky nenacházím nic extrémně zajímavého, uvedu jen výpisy pro orientaci, a pokud budete mít někdo zájem o detaily, ozvěte se.

```
rtc_time        : 18:46:15
rtc_date        : 2001-10-03
rtc_epoch       : 1900
alarm           : **:00:00
DST_enable      : no
BCD             : yes
24hr            : yes
square_wave     : no
alarm_IRQ       : no
update_IRQ      : no
periodic_IRQ    : no
periodic_freq   : 1024
batt_status     : okay

Checksum status: valid
# floppies     : 1
Floppy 0 type  : 3.5'' 1.44M
Floppy 1 type  : none
HD 0 type      : 2f
HD 1 type      : none
HD type 48 data: 0/0/75 C/H/S, precomp 0, lz 47616
HD type 49 data: 45679/234/148 C/H/S, precomp 2439, lz 3030
DOS base memory: 640 kB
Extended memory: 65472 kB (configured), 65472 kB (tested)
Gfx adapter    : EGA, VGA, ... (with BIOS)
FPU            : installed
```

První je výpis souboru */proc/driver/rtc* a potom následuje soubor */proc/driver/nvram*.Jenom mne trochu zaráží, že CMOSka hlásí pouze 64MB RAM když osazeno je 256MB. Když budu mít čas, tak se na to mrknu, ale nevím, ovladač NVRAM prostě přečte obsah CMOS paměti a zobrazí ho, takže nečekejte zázraky. Teď u sebe nemám popis obsahu CMOS paměti, ale pokud někdo víte, jak jsou tam informace uloženy, zde je kousek kódu, který vypisuje info o paměti:

```
PRINT_PROC( "DOS base memory: %d kB\n", nvram[7] | (nvram[8] << 8) );
PRINT_PROC( "Extended memory: %d kB (configured), %d kB (tested)\n",
            nvram[9] | (nvram[10] << 8),
            nvram[34] | (nvram[35] << 8) );
```

Pole *nvram[]* obsahuje "raw" obsah paměti CMOS. Jedná se o bajty 14 až 128. Jeho velikost je tedy 128-14. A prvek *nvram[0]* odkazuje na bajt 14 v paměti CMOS. Pro více informací je možno nahlédnout do souborů: *drivers/char/nvram.c*, *include/linux/nvram.h* a *include/asm/mc146818rtc.h*.

bus

Dostáváme se k informacím o sběrnicích a jejich obsazení periferiemi. Tady vás asi nepotěší, že na mém počítači už není jediná ISA sběrnice (alespoň klasická se slotem). Na druhou stranu deska Abit VP6, jejímž jsem šťastným vlastníkem, obsahuje čidla pro měření teplot a napětí. Ta jsou připojena přes sběrnici *i2c* a v konkrétním případě je pro připojení zřejmě použita interní ISA. Obsah adresáře */proc/bus* je tedy následující:

```
i2c  i2c-0  i2c-1  pci/
```

Sběrnice *i2c* je representována stejnojmenným souborem, *i2c-[01]* jsou u mne prázdné. Z prvního souboru je možné zjistit právě napojení měřících čidel, v tomto případě přes ISA sběrnici. Další je připojení informačních čipů v pamětech typu DIMM:

```
i2c-0  dummy  ISA main adapter                ISA bus algorithm
i2c-1  smbus  SMBus VT82C686 adapter at 5000  Non-I2C SMBus adapter
```

Tento výpis byl proveden při zavedení těchto modulů: i2c-core, i2c-proc, i2c-isa, i2c-viapro, eeprom a via866a. Funkční moduly jsou poslední dva, *eeprom* pro čtení informací z paměťových modulů DIMM a *via686a* pro informace ze senzorů tepla a napětí. Jedno malé, leč důležité upozornění! Moduly pro čidla nejsou obsaženy v jádře (i když i zde je možno nalézt podporu sběrnice *i2c*), ale najdete je na adrese [http://www.lm-sensors.nu] www.lm-sensors.nu. Jedná se o dva samostatné balíčky *lm_sensors* a *i2c* (u mne obojí verze 2.6.0).

Nebudu zabředávat do informací o připojení senzorů a sběrnicích jako *i2c* nebo *SMBus*. Hlavně proto, že o nich nic moc nevím. Užitečné informace o senzorech a paměti najdeme někde jinde (*/proc/sys/dev/sensors/* a tam si také řekneme, co znamenají.

Teď už raději přejděme na sběrnici PCI, o té Vám toho můžu říci přece jenom něco víc. V adresáři */proc/bus/pci* jsou informace o konfiguračním prostoru jednotlivých karet pro tuto sběrnici. Všechny informace jsou zde uvedeny dvakrát, jednou v souboru *pci/devices* a podruhé v jednotlivých adresářích. Tyto informace využívá například utilita *lspci*, kterou je vhodné použít namísto civění do nul a jedniček. Ale i to civění může být občas zajímavé. Tak nejprve soubor *devices*:

```
0000    11060691    0    d8000008        00000000    00000000
0008    11068598    0    00000000        00000000    00000000
```

```
0039    11060571   0     00000000       00000000   00000000
003a    11063038   13    00000000       00000000   00000000
003b    11063038   13    00000000       00000000   00000000
003c    11063057   5     00000000       00000000   00000000
0050    10ec8139   11    0000cc01       de000000   00000000
0058    125d1969   11    0000d001       0000d401   0000d801
0100    10de0110   10    dc000000       d0000008   00000000
```

Údaje zde můžeme rozdělit po sloupcích. První sloupec udává jednoznačnou adresu zařízení dle sběrnice, jeho první dvě cifry udávají číslo sběrnice. Jak je vidět, všechny řádky kromě posledního zde mají "00" a jsou tedy připojeny na společnou sběrnici PCI. Poslední řádek je zařízení připojené na sběrnici "01". Jak zjistíme později, je sběrnice číslo 0 skutečná PCI a sběrnice číslo 1 je AGP sběrnice. AGP je totiž odvozena od PCI, a tak jsou informace o nich zobrazovány společně.

Druhý sloupec jednoznačně identifikuje zařízení, která se v dané sběrnici nachází. První čtyři cifry udávají identifikaci výrobce, zbývající čtyři identifikaci výrobku. Pro převod na textové popisky je potom možno použít několik zdrojů. Jedním z nich je soubor */usr/share/pci.ids*, další možností je patřičný soubor v jádře: *drivers/pci/pci.ids*. Ten bude pravděpodobně novější (zvlášť pokud používáte vlastní novější jádro :).

Třetí sloupec udává IRQ, které dané zařízení používá. Je-li umožněno sdílení přerušovacích linek, mohou mít dvě různá zařízení stejné IRQ (viz IRQ 11).

Další sloupce obsahují bázové adresy a velikosti paměťových oblastí pro daný adaptér. Například pro grafickou kartu je jednou oblastí její lokální grafická paměť mapovaná do adresového prostoru procesoru (poslední řádek). Nejprve je uvedeno sedm bázových adres, z nichž ne všechny musí být využity, následuje sedm velikostí oblastí s příslušnou bázovou adresou. Například u posledního řádku najdeme v prvním sloupci adresu *DC000000* (MMIO registry) a v druhém adresu *D0000000* (framebuffer). Potom ve sloupcích 8 a 9 jsou vidět velikosti těchto oblastí *01000000* (16MB) respektive *08000000* (128MB). Stejné informace dostaneme jednak pomocí utility *lspci*, a nebo pro XFree86 >4.0 také ze souboru XFree86.0.log pomocí grepu:

```
sioux@sirion /var/log $grep -E "Linear|MMIO" XFree86.0.log
(--) NVIDIA(0): Linear framebuffer at 0xD0000000
(--) NVIDIA(0): MMIO registers at 0xDC000000
```

Zvídavé jistě napadne, proč je ve výpisu adresa D0000008. To vypadá jako nějaký blud. Ovšem pokud si uvědomíme, že velikost oblasti a její bázová adresa budou zřejmě zarovnány na nějakou jednotku větší než bajt (třeba stránka - 4/8 KB), zbývá nám posledních pár bitů adresy k jinému využití. Konkrétně jsou zde uvedeny příznaky pro danou oblast. Jednotlivé příznaky udávají typ oblasti - zda se jedná o 32bitovou, nebo 64bitovou adresu, případně zda je paměť tzv. *prefetchable* - kešovatelná. V této situaci se poslední případ rozlišuje podle toho, zda čtení v dané oblasti má, nebo nemá vedlejší účinky. Čtení framebufferu nám prostě zobrazí jeho hodnotu, ale čtení konfiguračních registrů může měnit jejich nastavení, a proto je nutné ho číst znovu z paměti a ne z cache.

Teď se přeneseme do jednotlivých adresářů. Nejprve jejich název. Ten kóduje identifikaci konkrétního zařízení na sběrnici - jeho umístění. Nejdříve adresář s číslem sběrnice - o tom jsme už mluvili. V nich se nacházejí podadresáře pro jednotlivá zařízení s dalšími podivnými názvy. Je na čase shrnout adresování PCI zařízení. Každé PCI zařízení v systému je jednoznačně adresováno pomocí 16bitového čísla skládajícího se ze tří částí. Prvních osm bitů udává číslo sběrnice, z toho dostaneme až 256 možných sběrnic v jednom počítači. Následuje pět bitů pro zařízení (32 zařízení) a tři bity pro tzv. funkci. Tohle schéma počítá s

multifunkčními zařízeními jako například zvuková karta, na níž je napojena ještě CD-ROMka. V takovém případě bude mít tento komplet přiděleno jedno číslo zařízení, ale lišit se budou čísla funkcí (zvukovka funkce 0, CD-ROM funkce 1). Dalším dobrým příkladem jsou prostředky základní desky (řádky 3-6 ve výpisu souboru *drivers*). Zde je vše jedno zařízení a jednotlivé funkce jsou ISA bridge, IDE cntrl, USB, USB, ACPI.

Tohle identifikační číslo se používá buď jako jedno 16bitové číslo - jako v prvním sloupci v souboru *devices*, nebo jako dvě 8bitová čísla oddělující sběrnici a zařízení, nebo jako trojice bus, dev, fun. Tato trojice se často zapisuje jako

```
bus:dev.fun
```

No a to nám už vysvětluje způsob pojmenování souborů v */proc/bus/pci/[bus_id]/*. Obsahem těchto souborů je kopie tzv. konfiguračního prostoru PCI zařízení. Jeho velikost je vždy 256 bajtů, z toho obsah prvních 64 bajtů je dán standardem, je vždy přítomen a obsahuje nejdůležitější informace o zařízení. Pro popis jednotlivých položek doporučuji například kapitolu 15. z knihy *Linux Device Drivers*, kterou lze on-line najít na internetu u nakladatelství [http://www.oreilly.com/catalog/linuxdrive2/chapter/book] O'Reilly, případně její PDF verzi [http://www.oreilly.com/catalog/linuxdrive2/chapter/bookindexpdf.html] tamtéž.

Jak jsem již řekl, informace uvedené v adresáři */proc/bus/pci/* lze jednoduše získat pomocí utility *lspci* (často */sbin/lspci*). Ta má spoustu voleb, jak informace zobrazit, a bývá příjemnější než přímý výstup konfiguračního prostoru.

fs

Opět se jedná o podobný adresář jako */pro/driversc*. Po menším hledání (obdobným způsobem) jsem našel dva souborové systémy, které jej využívají. Prvním z nich je NFS. Když spustíte NFS démona, objeví se zde adresář */proc/fs/nfs*, který obsahuje soubor *exports*. Ten obsahově odpovídá stejnojmennému souboru v adresáři */etc*, rozdíl je v tom, že zde nalezneme navíc číslo verze (je nastavena napevno v *fs/nfsd/export.c*) a standardní příznaky, které nemusí být v souboru s definicí exportů uvedeny (async, wdelay), a také IP adresu klientského počítače. Je doufám jasné, že tento soubor zobrazuje aktuální informace, a tak je možné ho použít v případě potíží, když je potřeba vědět, jak je export nastaven ve skutečnosti.

```
# Version 1.1
# Path Client(Flags) # IPs
/home   se710(rw,no_root_squash,async,wdelay) # 192.0.0.2
```

Druhým kandidátem na využití je souborový system CODA. Ten si v adresáři */proc/fs/coda* vytváří tyto soubory: *vfs_stats, upcall_stats, permission_stats, cache_inv_stats*. Bohužel CODu neprovozuji, a tak vám nemohu poskytnout ani výpis, ani podrobnosti o obsahu jednotlivých souborů. Pokud máte s CODou někdo nějaké zkušenosti, určitě se o ně rádi podělíte s námi ostatními :)

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=879

Zdroj: Root.cz (www.root.cz)

Co před námi tají /proc XXI.

Verze článku pro čtení: http://root.cz/clanek.php4?id=896

23.10.2001

Dneska se vrhneme na další tři adresáře. Budou to informace o IDE zařízeních, přehled přerušení a údaje o grafické kartě, poskytované ovladačem od firmy nVidia.

Pro představu o tom, co nás čeká a co už máme za sebou, uvedu výpis adresářů, které lze nalézt na mém počítači v */proc*. Otečkované jsou adresáře, které jsme už prošli.:

```
.acpi/      ide/     sys/
.bus/       irq/     sysvipc/
.driver/    net/     tty/
.fs/        nv/
```

ide

Tento adresář, jak je patrné z názvu, sdružuje informace o zařízeních připojených přes rozhraní IDE (respektive EIDE). Nejčastěji se jedná o pevné disky a mechaniky CD ROM, případně IDE verzi ZIPu a podobná zařízení na ukládání dat. Obsah je následující:

```
drivers  hda@  ide0/  via
```

Soubor drivers obsahuje informace o použitých ovladačích nejvyšší úrovně. Najdete zde údaje o použitých ovladačích a jejich verzích, jako příklad třeba:

```
ide-cdrom version 4.59
ide-disk version 1.10
```

Jak vidíte, nic extra zajímavého. Pokud máte někdo IDE ZIP, možná je zde ještě něco jako ide-zip, i když si spíše myslím, že ZIP na této úrovni spadá pod *ide-disk*. Jako obvykle oceníme, pokud nám nějaký vlastník ZIPu nebo i exotičtějších zařízení pošle své poznatky a výpisy do diskuse. Jenom aby vás to později nemátlo, momentálně žádnou CD-ROMku nemám (odešla do věčných lovišť), a tak položka pro *ide-cdrom* vznikla umělým natažením příslušných modulů. Pokud máte podporu CD-ROM napevno v jádře, bude tam tento řádek samozřejmě bez ohledu na přítomnost jakéhokoliv zařízení. To jen abyste se pak neptali, jak to, že je dále ve výpisu jen *hda*.

Link *hda* odkazuje jen zkratkou přímo na adresář *ide0/hda*. I když je to možná pro většinu z vás zbytečné, přece jen se zastavím u způsobů připojení a pojmenování IDE zařízení. Když se podíváte na vaši základní desku, najdete tam určité množství konektorů pro připojení IDE zařízení (jen připomínám, že sem nepatří floppy disk - ten není IDE). Tyto konektory odpovídají jednotlivým *kanálům IDE*, které jsou označeny jako *ideN*, kde na běžných deskách má *N* hodnotu buď *0*, nebo *1*, což odpovídá dvěma konektorům. Na některých deskách (třeba od Abitu VP6 nebo KT7 - doufám, že jsem to označení trefil - kdyžtak mě prosím někdo opravte) je buď integrován řadič RAID, nebo jiným způsobem zvýšen počet kanálů. Potom se jedná o kanály *2* a *3* (případně další - nevím, jestli je nějaké omezení na počet kanálů IDE - obvykle se však používají pouze dva, v případě integrovaného RAIDu celkem čtyři, a tím to končí). Pokud potřebujete připojit další disky, používá se přídavná deska do PCI, a to buď RAID, nebo třeba SCSI adaptér. Dříve byly běžné i karty s dalšími kanály IDE, ale nevím, jestli se dnes ještě dostanou a používají.

Tím máme snad jasno v tom, co znamená adresář *ide0*. V něm potom najdeme soubory společné pro celý kanál. Dále pak dva adresáře se jmény *hdX*, kde *X* je písmeno *a* až (běžně) *h*. Na každý *IDE kanál* je totiž možné připojit dvě IDE zařízení, jedno se nazývá *master* a druhé *slave*. Pro první kanál je master označen *hda* a slave *hdb*. Pro druhý kanál pak *hdc* a *hdd*, a tak dále. První z dvojice písmen je vždy master, druhý slave. Ještě poznamenám, že obecně je vhodné pokud možno každé zařízení připojovat na samostatný kanál. Pokud jsou obsazeny obě pozice na kanálu, musí o něj totiž zařízení soupeřit. Pokud pro jedno zařízení použijete kanál *ide0* a pro druhé kanál *ide1*, je možné, že lepší řadiče IDE budou schopny paralelně obsluhovat oba kanály a zařízení si nebudou konkurovat. Opět, pokud někdo víte, jaká je dnes situace, jestli je tohle chování běžné, nebo ne, a případně názvy řadičů, které ho zvládají, určitě se s námi o své zkušenosti podělíte.

Tak, trochu teorie na úvod máme za sebou a teď už se vrhneme na další soubory. Druhým souborem v základním adresáři *ide/* je v mém případě *via*. Jak někteří již briskně uhodli, soubor se jmenuje po řadiči IDE na základní desce. Pokud je řadičů více, může být i souborů několik - to platí opět například pro integrované RAID řadiče. Jenom aby nevznikly nějaké nejasnosti, když tu v této části mluvím o RAID zařízeních, mám na mysli RAID složený z IDE disků. Nic z toho se netýká SCSI disků a RAIDů z nich. Momentálně jádro rozeznává tyto řadiče IDE (2.4.12): *AEC62XX, ALI15X3, AMD74XX, CMD64X, CS5530, HPT34X, HPT366, SWWKS, PDC202XX, PIIX, SIS5513, SLC90E66 a VIA82CXXX*. Můj počítač je osazen řadičem IDE od firmy VIA (z čipsetu VIA Apollo PRO133A Dual), a tak se soubor jmenuje *via*. Určitě nesmím zapomenout poznamenat, že správcem ovladače pro IDE řadiče VIA je Čech pan Vojtěch Pavlík ze SuSE. Tímto ho zdravím a děkuji za dobře odvedenou práci (pozn. red.: taky zdravim :) - johanka). A teď už vlastní obsah souboru. Ten se pro jednotlivé řadiče může radikálně lišit, základní struktura informací však bude stejná nebo obdobná:

```
----------VIA BusMastering IDE Configuration----------------
Driver Version:                     3.29
South Bridge:                       VIA vt82c686b
Revision:                           ISA 0x40 IDE 0x6
Highest DMA rate:                   UDMA100
BM-DMA base:                        0xc000
PCI clock:                          33MHz
Master Read  Cycle IRDY:            0ws
Master Write Cycle IRDY:            0ws
BM IDE Status Register Read Retry:  yes
Max DRDY Pulse Width:               No limit
----------------------Primary IDE-------Secondary IDE------
Read DMA FIFO flush:          yes                 yes
End Sector FIFO flush:         no                  no
Prefetch Buffer:              yes                 yes
Post Write Buffer:            yes                  no
Enabled:                      yes                 yes
Simplex only:                  no                  no
Cable Type:                   80w                 40w
-------------------drive0----drive1----drive2----drive3-----
Transfer Mode:       UDMA       PIO       PIO       PIO
Address Setup:       30ns      120ns     120ns     120ns
Cmd Active:          90ns       90ns     480ns     480ns
Cmd Recovery:        30ns       30ns     480ns     480ns
Data Active:         90ns      330ns     330ns     330ns
Data Recovery:       30ns      270ns     270ns     270ns
Cycle Time:          20ns      600ns     600ns     600ns
Transfer Rate:   99.0MB/s   3.3MB/s   3.3MB/s   3.3MB/s
```

V první části jsou informace o konkrétním řadiči. Čísla verzí, nejvyšší dostupný režim přenosu dat (zde UDMA100), rychlost PCI sběrnice a další. V druhé sekci se nacházejí informace o jednotlivých kanálech. Primární IDE je kanál *ide0*, sekundární IDE odpovídá

ide1. Nebudu rozebírat jednotlivé řádky, přiznám se, že všem stejně zcela nerozumím, za povšimnutí jen stojí ten poslední - je na něm možno zjistit použitý typ kabelu, a tedy použitelnost vyšších UDMA režimů. Všechny tyto informace jsou čteny z konfiguračního prostoru PCI nebo z I/O portů řadiče. Poslední sekce je určena pro jednotlivá zařízení. Hodnoty v ní však nejsou závislé na konkrétním připojeném disku, ale na nastavení komunikačního protokolu - tedy PIO, DMA nebo UDMA režimu. To je vidět nejvíce na poslední položce. Opravdu neznám disk, který by dnes dokázal přenášet 100MB/s. Na tomto místě bych rád apeloval na pana Pavlíka. Myslím, že i když je zdrojový text napsán dobře a přehledně, dle mého názoru by mu neuškodil i nějaký ten komentář. Také používání "magických konstant" je na většině míst značně matoucí. Proto pokud budete pane Pavlík číst tento článek a tyto věty, zamyslete se nad přidáním několika vysvětlivek do zdrojového textu souboru *drivers/ide/via82cxxx.c*. Myslím, že by to některé lidi učinilo o něco šťastnějšími (mne určitě :). Děkuji.

Nyní se přesuneme do adresáře *ide0*. Zde se nachází několik souborů pro daný kanál, řekl bych nic speciálního, tak jen rychle. *channel* obsahuje číslo kanálu - tedy *0*. *config* je kopií konfiguračního prostoru PCI řadiče. *mate* u mne nyní neobsahuje nic, ale vzhledem k názvu to bude nejspíše identifikace, nějaký odkaz na druhý kanál či něco podobného, *model* pak určuje druh řadiče tohoto kanálu. Dnes asi nejrozšířenější hodnota je "pci". Ostatní možnosti lze nalézt ve funkci *drivers/ide/ide-proc.c:proc_ide_read_imodel()*.

Další na řadě je adresář *hda*. Ten již obsahuje informace o konkrétním zařízení, připojeném na místo patrné z názvu adresáře. Tento disk je tedy připojen na první kanál jako master. Většina souborů je velmi jednoduchých, tak se na ně vrhneme:*cache* udává velikost cache na disku (2048), *capacity* udává celkový počet sektorů. Většina lidí má tak nějak zažito, že sektor je 512 bajtů. Pokud mezi sebou vynásobí tyto dvě hodnoty, mohou se dočkat nepříjemných překvapení. Například můj 10GB disk má najednou jen 9.3GB a podobně. Jak to? Vycházíme ze špatného předpokladu. Správnou velikost sektoru lze zjistit například pomocí utility *hdparm -i /dev/hda*.

```
...
RawCHS=16383/16/63, TrkSize=57600, SectSize=600, ECCbytes=40
...
CurCHS=16383/16/63, CurSects=-66060037, LBA=yes, LBAsects=19541088
...
```

Zde se můžete dočíst, že disk má opravdu takový počet sektorů, který udává soubor *capacity* (položka LBAsects). Na druhou stranu je velikost sektoru překvapivě 600 bajtů (položka SectSize). Pro vlastní data je nutno odečíst ještě oblast pro informace kontroly integrity (ECCbytes). Nakonec se dostaneme k velikosti sektoru 600-40 = 560 bajtů. Velikost disku je potom 560*19541088 = 10.2 * 1024 * 1024 * 1024 bajtů, tedy inzerovaných 10.2 GB. No, není nad to, když se člověk dozví něco nového. A až vám někdo bude zase tvrdit něco o sektoru 512 bajtů velkém, pošlete ho přečíst si tento článek a podívat se do výpisu programu *hdparm*.

Soubor *driver* obsahuje opět název a verzi generického ovladače IDE jako v hlavním adresáři. *geometry* udává skutečnou a logickou (pro BIOS a spol) geometrii disku ve formátu C/H/S - cylindr, hlava, sektor:

```
physical     19386/16/63
logical      1216/255/63
```

media udává třídu zařízení. Dozvíte se zde třeba to, že disk je ve třídě disk :). *model* obsahuje identifikační řetězec jako první řádka výpisu programu *hdparm*. Poslední

smysluplný soubor je *settings*, kde se dá najít aktuální a povolené nastavení všemožných parametrů zařízení. Pak zbývají ještě soubory *identify, smart_values* a *smart_threshold*. Ty při čtení pošlou na zařízení povely protokolu IDE *IDENTIFY, SMART_READ_VALUES, SMART_READ_THRESHOLDS* a zobrazí vrácená data jako 256 16bitových slov. Přesný význam pak záleží na konkrétním zařízení.

irq

Tento adresář je určen pro nastavování obsluhy přerušení. Základem pro všechny systémy je soubor *prof_cpu_mask*, ten obsahuje masku procesorů zpracovávajících přerušení. Pro jiné uživatele než *root* jsou však všechny informace nepřístupné. Superuživatel je schopen jednak zjistit aktuální stav nastavení pomocí čtení obsahu souborů, a dále může zápisem nových hodnot nastavení měnit. Dále jsou zde adresáře s čísly 0 až MAX_IRQ, pro každé IRQ jeden. Pro jednoprocesorový systém jsou tyto adresáře prázdné, pro víceprocesorový systém obsahují soubor *smp_affinity* (každý). A k čemu jsou masky dobré. Maska určuje, které procesory mohou zpracovávat přerušení. Globální maska v souboru *prof_cpu_mask* se vztahuje na všechna přerušení, masky v *smp_affinity* pak k jednotlivým IRQ. Pokud tedy nastavíme masku na *5*, tedy binárně *...01001*, znamená to, že přerušení mohou obsluhovat procesor číslo 1 a procesor číslo 4 (respektive cpu0 a cpu3 podle počítání Linuxu). Pokud nastavíme globální masku, vztahuje se na všechna přerušení, lokálně lze nastavit chování pro jednotlivá přerušení v patřičných adresářích. Pokud může přerušení obsluhovat více procesorů, je plánování obsluhy řízeno algoritmem *Round Robin*. To zjednodušeně řečeno znamená, že se všichni spravedlivě střídají dokola.

nv

V tomto adresáři najdeme jediný soubor. Jmenuje se *card0* a obsahuje několik informací o grafickém adaptéru firmy nVidia. Teoreticky, pokud by adaptérů bylo více (např. v PCI sběrnici), přibudou další soubory (*card1*, ...). Tento adresář a v něm obsažené informace jsou specifické pro ovladač firmy nVidia. Pokud je použit open-source ovladač, žádné adresář *nv* se nevytvoří. Ovladače jiných grafických karet pravděpodobně také poskytují možnost získání informací přes rozhraní */proc*, ale mám jen jednu kartu, tak nevím, kde je hledat. Další věc, s kterou se s námi můžete podělit v diskusi k článku. Obsah souboru *card0* je následující:

```
----- Driver Info -----
NVRM Version: 1.0-1541
------ Card Info ------
Model:        GeForce2 MX
IRQ:          16
Video BIOS:   03.11.00.18
------ AGP Info -------
AGP status:   Enabled
AGP Driver:   AGPGART
Bridge:       Via Apollo Pro
SBA:          Supported [disabled]
FW:           Supported [disabled]
Rates:        2x 1x  [2x]
Registers:    0x1f000213:0x00000102
```

Dozvíme se zde verze ovladače (1.0-1541) a verzi BIOSu grafické karty (03.11.00.18), použité přerušení (16) a model grafické karty. Model se určí pomocí tabulky PCI DevID v souboru *nv-ids.h*. Dále je vidět, zda je podpora AGP zapnuta a který ovladač AGP řídí. V tomto případě je na výběr ze dvou možností: buď ovladač *AGPGART* z jádra, nebo ovladač AGP v modulu *NVdriver*. Dále je uveden název AGP chipsetu. Jeho detekce je však provedena přímo v modulu NVdriver a nemusí být tedy aktuální. Detekují se chipsety od

výrobců VIA, Intel, AMD a Ali, a to jen některé. Neděste se proto, jestliže je zde napsáno něco, s čím vůbec nesouhlasíte. V takovém případě nahlédněte do souboru *nv.c* do funkce *nv_kern_proc_read()*, co se vlastně detekuje. Dále se vypíše nastavení AGP sběrnice včetně jejích možných rychlostí. U této položky trochu pochybuji o její správnosti, protože deska (Abit VP6) by měla umět AGP 4x, ale zde nic není. Nakonec se dozvíte obsah stavového a příkazového registru ovladače AGP, kde se dá zjistit zbytek.

A to je pro dnešek všechno. Příště budeme pokračovat zbytkem menších adresářů a na poslední díl si necháme adresář *sys*, kde je informací opravdu požehnaně (hlavně o vlastním jádře).

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Co před námi tají /proc XXII.

Verze článku pro čtení: http://root.cz/clanek.php4?id=907

30.10.2001

Dneska probereme informace o prostředcích pro mezi-procesorovou komunikaci ze Systemu V, pak si povíme pár věcí o terminálech a hlavně se zaměříme na informace o stavu a nastavení sítě.

Pro představu o tom, co nás čeká a co už máme za sebou, uvedu výpis adresářů, které lze nalézt na mém počítači v */proc*. Otečkovány jsou ty adresáře, které jsme už prošli.:

```
.acpi/     .ide/     sys/
.bus/      .irq/     sysvipc/
.driver/    net/     tty/
.fs/       .nv/
```

Pozor! Opět jsem přešel na novější jádro - tentokrát 2.4.13-pre3. Samozřejmě hlavním důvodem jsou nedočkaví čtenáři *Roota*, kteří vyžadují vždy aktuální informace :)

net

V tomto adresáři najdeme informace o síťovém rozhraní jádra, a to především základní a obecnější informace o využití jednotlivých vrstev a rozhraní. Jsou zde přítomny tyto soubory:

```
arp, dev, dev_mcast, igmp, netlink, netstat, raw, route, rt_cache,
rt_cache_stat, snmp, sockstat, softnet_stat, tcp, udp, unix
```

Na první pohled je jich docela dost, ale hodně jich je podobných a pouze některé obsahují opravdu důležité informace. Nebudeme se v tom příliš šťourat, jednak sítě nejsou moje nejsilnější stránka, a taky po minulém fiasku s velikostí sektoru disku utrpělo moje sebevědomí značnou ránu. Navíc doma mám pouze jeden počítač, a tak ukazovat příklady sítě není zrovna nejjednodušší (kromě loopback). Tak opravdu letecky nejprve to, co je jednodušší: Soubor *arp* zpřístupňuje obsah ARP keše. V té se uchovávají vazby mezi IP adresami a HW adresami jednotlivých síťových karet. To je potřeba pro směrování paketů v rámci jedné sítě. ARP je potom zkratka pro *Adress Resolution Protokol*. *dev* obsahuje statistiky provozu jednotlivých sítových rozhraní:

```
Inter-|   Receive                                                |
 face |bytes    packets errs drop fifo frame compressed multicast|...
    lo:    2453       9    0    0    0     0          0         0
  eth0:       0       0    0    0    0     0          0         0
 plip0:       0       0    0    0    0     0          0         0
```

V ukázce je zobrazena první polovina pro přijaté pakety a po ní následuje v souboru již nezobrazené tabulka pro provoz na zařízení. Je zde vidět jednak množství zpracovaných dat (*bytes* nebo *packets*), počet chybných (*errs*) a zahozených *drop*) paketů. Pro jiná zařízení než lo je ještě k dispozici informace o rámcích (*frames*), kompresi (*compress*) a multicastingovém provozu na tomto zařízení (*multicast*).

Dalším souborem je *dev_mcast*, kde jsou informace o zařízeních schopných multicastového provozu (tedy skupinové komunikace na více adres současně). Je zde vidět počet uživatelů a skupin (třetí a čtvrtý sloupec) a multicastová adresa (poslední položka).

igmp obsahuje další informace k multicastingu pro IP, *netstat* některé statistiky síťového

provozu, *netlink* obsahuje informace o speciálním síťovém rozhraní, které se používá pro komunikaci mezi jádrem a uživatelskými aplikacemi (na jednom stroji).

Soubory *route*, *rc_cache* a *rt_cache_stat* uvádějí důležité informace o směrování IP paketů. Soubor *route* udává použitá směrovací pravidla v jádře a zbylé dva poskytují informace o obsahu a využití keše směrování IP paketů.

snmp udává některé další detailní informace o jednotlivých protokolech, *sockstat* potom přehledně vypisuje využití soketů jednotlivých typů. Jsou zde informace o využití TCP, UDP, RAW a dalších soketů. Nejsou zde však obsaženy sokety rodiny AF_UNIX. Podrobné informace (jako adresy odesilatele a příjemce, identifikace vlastníka, stav front atd.) jsou pro jednotlivé typy soketu v souborech *tcp*, *udp* respektive *unix*.

Posledním souborem je *softnet_stat*. Ten obsahuje statistiky pro všechna zařízení a protokoly (nad IP) dohromady. První dva sloupce udávají počet celkem přenesených paketů a počet celkem zahozených paketů. Význam dalších polí je možno nalézt v *net/core/dev.c:dev_proc_stats()*.

I když tohle byl spíše průzkum bojem, než nějaké pořádné vysvětlení, bude to muset stačit. U části souborů jsou celkem vysvětlující hlavičky tabulek, u zbytku jsem se snažil (tam, kde jsem věděl) napsat, co se dalo. Třeba se k této problematice někdy vrátí nějaký povolanější síťař, než jsem já.

sysvipc

V tomhle adresáři nalezneme informace o aktuálně využívaných prostředcích pro komunikaci mezi jednotlivými procesy, které jsou obvykle označovány jako IPC System V. IPC jako Inter Process Communication a System V (V jako římské pět) je verze Unixu (od AT&T), ve které se tyto prostředky poprvé objevily. Jedná se o tři částečně vzájemně nahraditelné způsoby komunikace mezi jednotlivými procesy. První způsob jsou *zprávy (messages)*, mechanismus zasílání strukturovaných dat do schránek, ke kterým mají přístup všechny procesy (obecně). Je to jako papírová pošta. Napíšete dopis, adresu a někdo ho pak doručí až na místo. Můžete také dopis nechat ležet na ulici a tím ho adresovat libovolnému kolemjdoucímu.

Další metodou komunikace jsou *semafory (semaphore)*. Velmi zjednodušeně je lze popsat jako nastavitelné příznaky. Klasickým použitím je přístup k objektu, který je schopen obsloužit jen jeden požadavek najednou. Jako příklad třeba tiskárna. Pokud budu chtít tisknout, zkontroluji příznak PRINTER_BUSY. Není-li nastaven, nastavím ho - tím si tiskárnu zaberu pro sebe - a tisknu. Až skončím, nastavím příznak zpět na volno. Pokud je ale při testování obsazen, znamená to, že již někdo jiný tiskne. Potom musím počkat, až někdo jiný nastaví příznak do pozice volno, a můžu zkusit celý postup znovu. No snažil jsem se o počítačový příklad, ale pro ty, co o počítačových semaforech nikdy neslyšeli, bude asi lepší si představit železniční přejezd se semaforem. Ten tam plní stejnou funkci. Zajišťuje, aby byl přejezd využíván vždy jen v jednom z kolmých směrů (vlak nebo auta, ale nikdy dohromady).

Posledním způsobem komunikace dvou procesů je sdílená paměť. S tímto pojmem se někdy trochu zahrává, tak jen vysvětlím, jaké jsou možnosti. Pokud se mluví o správě paměti, je sdílenou pamětí označována část fyzického adresového prostoru například s kódem (dynamické) knihovny, který je namapován do více logických adresových prostorů (tedy procesů). Tyto procesy však o tom nemusí mít (a zpravidla nemají) nejmenší tušení. Celý

mechanismus slouží jen pro úsporu paměti, případně času a dalších prostředků. Naproti tomu sdílená paměť v kontextu meziprocesorové komunikace označuje takovou fyzickou paměť mapovanou na více procesů, o které tyto procesy vědí a používají ji k vzájemné komunikaci (synchronizaci, předávání dat atd.).

Probíraný adresář *sysvipc* obsahuje odpovídající tři soubory. V nich je potom možno nalézt aktuálně používané objekty patřičného druhu, na každém řádku jeden. Všechny soubory mají několik společných sloupců, a ty si popíšeme dohromady. První položkou je klíč *key*. Pokud se zamyslíte nad implementací spolupráce více procesů, zjistíte, že musí mít nějakou společnou informaci, kterou se vymezí do skupiny, ve které potom operují. Pokud mají přistupovat ke sdílenému objektu, musí mít pro něj všichni jednoznačnou identifikaci. Tento problém je vyřešen tak, že při vytváření objektů IPC se vždy zadává tzv. klíč, což je hodnota (číslo), na které se dohodnou spolupracující procesy. Pro každý proces potom funkce pro vytvoření IPC objektu vrátí jeho identifikaci, která se používá při další manipulaci. Ta už ale může být pro každý proces jiná (i když to není nutné). Další "společnou" položkou je ID. To je hodnota v druhém sloupci nadepsaném vracená funkcemi pro vytváření IPC objektů . Následuje položka *shmid, semid* nebo *msgid*, to je právě identifikace *perms*, obsahuje standardní unixová přístupová práva pro daný objekt. Ta se kontrolují vůči identifikaci procesu, který s objekty manipuluje. Zbývají identifikace majitele objektu v položkách *uid* a *gid* a identifikace toho, kdo objekt vytvořil, v *cuid* a *cgid*. U každého objektu je ještě několik časů. Jsou to některé z *stime* pro čas posledního odeslání (zápisu), *rtime* pro poslední příjem (čtení zprávy), *ctime* pro čas poslední modifikace (change), *atime* - připojení a *dtime* - odpojení sdílené paměti a nakonec *otime* pro čas poslední operace nad semaforem.

Než se pustíme do položek charakteristických pro jednotlivé typy objektů IPC, uvedu jako ukázku výpis souboru pro sdílenou paměť.

```
key    shmid  perms     size  cpid  lpid nattch   uid   gid  ...
  0        0   1600    46084   473   473       3    48    48  ...
  0 12713985   1777  1051392 28446 28448       1     0   100  ...
  0 24477698   1777   196608 28476 28481       2   500   100  ...
```

Konce řádků s identifikací toho, kdo objekty vytvořil a jednotlivé časy jsou bohužel z důvodů místa ořezány. To důležité ale zůstalo. Když už máme výpis, začneme tedy sdílenou pamětí. Sloupec *size* udává velikost oblasti. Ta se vždy alokuje po celých stránkách kvůli mechanismu, kterým je zajištěno vlastní sdílení (mapování FAP -&gt LAP). Obsluhovaná a udávaná je však velikost požadovaná při vytvoření. Dále jsou zde *cpid* a *lpid*, což jsou čísla procesů (PID), které objekt vytvořil (Creator PID), respektive naposledy použil (Last PID). *nattch* je počet procesů, které sdílenou paměť používají. Přesněji se jedná o počet vyřízených požadavků na připojení funkcí *shmat()*, ale běžně se tato operace provádí v každém procesu právě jednou. *uid* a *gid* už jsme probrali.

U semaforů je navíc jen položka *nsems*. Ta udává počet elementů v jednom objektu semaforu. Pro snazší práci se totiž jako objekt IPC nebere jeden semafor, ale rovnou jejich pole. K tomu se pak přistupuje při jednotlivých operacích najednou (a atomicky).

Zbývají ještě zprávy. U všech objektů IPC je omezen jejich maximální současný počet na jednom systému. U zpráv je navíc omezena velikost jedné zprávy. Právě velikost paměti vyhrazené pro doručované zprávy udává sloupec *cbytes*. Aktuální počet zpráv ve frontě pak sloupec *qnum*. *lspid* je PID procesu, který zprávu naposledy poslal (*msgsnd()*) a *lrpid* toho, kdo naposledy zprávu přijal (*msgrcv()*).

Pokud se budete zajímat o teorii, funkčnost nebo programování IPC, určitě vám bude pro

začátek stačit libovolná knížka o programování a/nebo architektuře Unixu. Pro pokročilé doporučuji samozřejmě zdrojáky jádra (případně ještě manuálové stránky pro všechny - *man 5 ipc*).

tty

Zde najdeme něco o ovladačích jednotlivých terminálových zařízení. Vzhledem k tomu, že v jejich názvech (a i všem ostatním) je teď trochu zmatek (třeba kvůli *devfs*), je celá tahle oblast trochu nepřehledná, ale pokusíme se podívat, co se s tím dá dělat.

Zajímavý pro nás bude pouze jeden soubor z celého adresáře, a to *drivers*. Druhý soubor *ldiscs* a oba adresáře *driver* a *ldisc* neobsahují nic až tak zajímavého.

```
pty_slave            /dev/pts      136   0-255 pty:slave
pty_master           /dev/ptm      128   0-255 pty:master
pty_slave            /dev/ttyp       3   0-255 pty:slave
pty_master           /dev/pty        2   0-255 pty:master
/dev/vc/0            /dev/vc/0       4       0 system:vtmaster
/dev/ptmx            /dev/ptmx       5       2 system
/dev/console         /dev/console    5       1 system:console
/dev/tty             /dev/tty        5       0 system:/dev/tty
unknown              /dev/vc/%d      4    1-63 console
```

Tak, tohle je tedy soubor *drivers*. První sloupec obsahuje jméno driveru pro daný terminál, za ním je jméno terminálu jako položka v souborovém systému */dev*. Následuje hlavní číslo daného zařízení, počet zařízení (rozsah) a nakonec typ. Typy jsou základní čtyři. Systémový terminál, konzole, sériový terminál a nakonec pseudo terminál. Pseudoterminály jsou zde první čtyři, zbytek jsou různé systémové terminály. To zatím není nic moc, takže co dál? No zklamu vás, nic. Sice mám slušnou představu, jak terminály pracují (i když dost obecně), ale ta spousta typů a jejich zařízení mi zatím příliš jasná není. Tak do toho nebudu zabředávat (alespoň nyní) a jdu od toho.

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=907

Co před námi tají /proc XXIII.

Verze článku pro čtení: http://root.cz/clanek.php4?id=923

08.11.2001

Tak a je to tady. Poslední díl našeho putování po tajemných zákoutích adresářového systému /proc. (Bohužel) budeme dnes pokračovat v nasazeném tempu a poslední adresář projdeme také poměrně rychle. Na druhou stranu - základní principy už jsme prošli dříve, tak už to teď půjde samo o sobě rychleji.

Nutno přiznat, že mi poslední dobou nějak dochází dech, respektive čas. Za začátku jsem se snažil pro vás snést modré z nebe, projít zdrojáky do posledního středníku a s každým článkem si hrál i několik dní. Pak začala škola a ostatní povinnosti a s mým volným časem to šlo z kopce. Poslední články už píši během několika hodin (obvykle tak 6-10ti) a na spoustu drobností, kterým bych se rád věnoval, již prostě nemám čas. No, ale teď už si dám pauzu, doženu ostatní resty a časem určitě zase něco napíšu. (Co bych pro vás neudělal, že :)

Tentokrát už by bylo otečkováno opět téměř všechno. Proto raději na tomto místě uvedu výpis obsahu posledního adresáře, který nám zbývá, a to je */proc/sys*. Znovu zopakuji, že (od minulého dílu) jsem aktualizoval na jádro 2.4.13!

```
abi/  debug/  dev/  fs/  kernel/  net/  proc/  vm/
```

abi

> Další z věcí, které se v jádře objevily relativně nedávno. Jako obvykle první pokusy byly v Alanových jádrech (větvi jádra od *Alana Coxe* - patche s postfixem *-ac* oproti oficiálnímu Linusovu jádru). Jedná se o způsob, jak spouštět binárky (pro stejný typ procesoru) přeložené pro jiný systém. Příkladem může být požadavek spouštět program přeložený pro Free BSD (x86) pod Linuxem (také na x86). Celý problém je o tom, poskytnout vrstvu, která odstíní části, které se na obou systémech liší. Těmito částmi může být třeba obsluha signálů, různá definice nebo i chování systémových volání a další.
>
> A teď jak to funguje. Pro každý systém musí v Linuxu existovat vrstva, která odstíní rozdíl právě tohoto systému a Linuxu. Této vrstvě se říká *ABI (Abstract? Binary Interface)*. Jedná se o popis rozhraní systému na binární úrovni, tedy tak, jak jej vidí přeložený program. (V komerčních systémech se ABI využívá například k tvorbě ovladačů distribuovaných bez zdrojového kódu - kompatibilita je zajištěna na úrovni přeložené binárky). Pokud jste pravidelnými čtenáři, možná si vzpomenete na díl, kde jsme probírali soubor [http://www.root.cz/clanek.php4?id=762] */proc/execdomains*. Ten obsahuje právě podporovaná ABI. Jelikož jednotlivé ABI mohou být velmi podobné, jsou v Linuxu sdruženy do něčeho, co se nazývá *execution domain* (raději bez překladu). V již zmiňovaném díle jsme se dozvěděli, že Linux (na úrovni definice konstant pro hodnotu *personality*) podporuje například SVR3, SVR4, BSD, SunOS, Xenix, Irix64 nebo Solaris.
>
> Pokud se podíváme trochu pod pokličku, zjistíme, jak se celý tento systém používá. Každý proces má vlastnost jménem *personality*, a ta určuje *execution domain*, ve které je prováděn (byl spuštěn). Až donedávna bylo pro binárky přeložené přímo pod Linu jedinou možností *personality = Linux*, nyní se začíná postupně pracovat na modulech pro ostatní hodnoty *personality*. Vlastní moduly jsou zatím pouze v Alanově řadě, případně v úpravách jader jednotlivých distribucí, ale postupně se vše probojovává i do oficiálního jádra. První

vlaštovkou je právě probíraný adresář.

Soubory *defhandler_xxx* udávají *personality* pro daný formát. Ten je nastaven napevno v *kernel/execdomain.c* na Linux pro *_elf*, SCO SVR3 pro *_coff* a SVR4 pro zbytek. Zbývají soubory *trace* a *fake_utsname*, zobrazující obsah "stejnojmenných" proměnných, které jsou zatím ovšem nevyužity.

debug

Adresář */proc/debug* zvládneme rychle. Obsahuje hromadu debugovacího smetí pro architekturu *s390* (ehmm to má přece doma každý, že :) a jednu položku k správě paměti pro *ARM*. Ani jedno z toho momentálně nemám k dispozici, takže smůla :(.

Jenom mimochodem, při hledání informací k tomuto adresáři jsem narazil na něco, o co byste neměli přijít. Při konfiguraci jádra je možné na konci konfiguračních voleb (menu *kernel hacking*) zapnout různé dodatečné kontroly a výpisy jádra. Jednou z možností je i *Debug memory allocations*. Její povolení způsobí přidání dalších informací do souboru [http://www.root.cz/clanek.php4?id=779] /proc/slabinfo a některé další kontroly při správě paměti. Na druhou stranu zvětší jádro a zpomalí jeho běh. Dříve bylo nutno ručně ve zdrojích jádra přepsat hodnotu konstanty *SLAB*.

A když už jsme u toho, *Debug High memory support* přidá dvě další kontroly při používání tzv. HIGH paměti (pozor, neplést s DOSem!), *Spinlock debugging* pak kontroly pro krátkodobé zámky jádra (i když takové podivné - viz *include/asm-i386/spinlock.h*), *Verbose BUG reporting* způsobí ukecanější výpisy chyb pro správu paměti (vypíše název souboru a číslo řádky, kde k chybě došlo), *Memory mapped I/O debugging* nahradí funkce pro přístup k periferiím verzemi s dalším testem na požadovanou adresu, *Magic SysRq*, aktivuje funkce klávesy *SysRq* - ta toho umí opravdu hodně, doporučuji dokumentaci v *Documentation/sysrq.txt*.

dev

Dle souboru *include/linux/sysctl.h* využívají tento adresář moduly pro CD-ROM, paralelní port, RAID, monitory HW a MAC HID. Co znamená to poslední, nevím, předpokládám, že něco pro Macintoshe. Já vám neposkytnu popis ani MACu ani RAIDu - nic z toho nemám. První modul využívající tento adresář, který se mi podařilo najít (ono "dev" se hledá špatně) je balík i2c, respektive sensors. Adresář */dev* tak u mne obsahuje podadresáře *sensors*, *cdrom* a *parport*. Pokud máte i jiný obsah, budeme všichni opět rádi, když se s ním pochlubíte v diskusi. Při použití separátního balíku *i2c* se kromě vlastních senzorů dal po vložení modulu *eeprom* zobrazit obsah informačních registrů v paměťových čipech DIMM. Nyní s využitím jaderného *i2c* to nějak nefunguje (zatím :). V adresáři */proc/sys/dev/sensors* najdeme jednak soubor *chips*, který informuje o zavedených ovladačích pro *i2c*, a potom vlastní adresář pro každý z ovladačů. Soubor *chips* u mne vypadá takto:

```
256     via686a-isa-6000
```

Jednoduché, že? Název ovladače je zároveň i názvem odpovídajícího adresáře a to číslo v předu je jméno položky. Bohužel v tomto případě to znamená binární ID, takže to příliš užitečné není. Název a hlavně obsah adresáře se už pro jednotlivé komponenty liší, pro příklad uvedu obsah modulu pro HW monitory na desce Abit VP6. Jedná se o komponentu *via686a* připojenou přes sběrnici *ISA*. Nevím, zda přímo fyzicky (jinak na desce žádná ISA není), nebo jen logicky s využitím algoritmu pro ISA sběrnici. Obsah adresáře *via686a-isa-6000* udává stav HW monitorů na desce:

```
alarms  fan1  fan2  fan_div  in0  in1  in2  in3  in4
temp1  temp2  temp3
```

Jednotlivé soubory obsahují většinou tři hodnoty - aktuální, minimální povolenou a maximální povolenou. Jak je patrné z názvů, jedná se o stavy otáček větráčků (fanX), napájecích napětí (inX) a teploměrů (tempX).

Dalším adresářem je *cdrom*. Ten obsahuje soubory s informacemi a nastavením chování vaší CD-ROMky. Soubory *autoclose, autoeject, check_media, lock* a *debug* povolují (obsah 1) nebo zakazují (obsah 0) po řadě automatické zavření resp. otevření dvířek při při- resp odpojení média, testování typu média, zamykání šuplíku s médiem a výpis ladících hlášek. V souboru *info* najdete několik informací o jednotce a jejích schopnostech (myslím, že tento soubor nepotřebuje zvláštní komentář):

```
CD-ROM information, Id: cdrom.c 3.12 2000/10/18
drive name:             hdc
drive speed:            48
drive # of slots:       1
Can close tray:         1
Can open tray:          1
Can lock tray:          1
Can change speed:       1
Can select disk:        0
Can read multisession:  1
Can read MCN:           1
Reports media changed:  1
Can play audio:         1
Can write CD-R:         0
Can write CD-RW:        0
Can read DVD:           1
Can write DVD-R:        0
Can write DVD-RAM:      0
```

Posledním adresářem, který vám můžu nabídnout, je *parport*. Zde najdete jednak adresář *default* s implicitním nastavením, a potom jeden adresář pro každý paralelní port (většinou jen *parport0*). V celé struktuře se porůznu objevují dva základní soubory. To, kde se nacházejí, určuje rozsah jejich platnosti. Vždy jsou v adresáři *default*, pro celý paralelní port mohou být v *parportX* a pro jednotlivá zařízení potom v *parportX/devices/XXX*. Těmito soubory jsou *spintime* a *timeslice*. První z nich určuje počet mikrosekund aktivního čekání na odpověď periferního zařízení po vyslání příkazu. Vztahuje se vždy na celý paralelní port (najdete ji v *default* a *parportX*). Jejím zmenšením je možno zvýšit rychlost komunikace, pokud zůstane stabilní. Druhým souborem je *timeslice*, udává počet milisekund, které jsou **doporučením** pro ovladač zařízení, jak dlouho může používat port pro sebe, než vrátí řízení (to má smysl při více zařízeních na jednom portu, což se příliš nenosí a ovladač tuto hodnotu může ignorovat, pokud to považuje za vhodné/nutné).

V adresáři *parport/parportX* najdete soubory s nastavením daného portu. Předpokládám, že stačí pouze výpisem:*base-addr, dma, irq, modes*. Poslední zmiňovaný udává použitelné způsoby komunikace:

```
PCSPP,TRISTATE,COMPAT,ECP,DMA
```

Také se zde opakuje soubor *spintime*. Ten je při registraci portu zkopírován z adresáře *default* a ovladač jej potom může změnit. Vyskytují se zde ještě soubory *autoprobeX* pro automatické zjištění nastavení portu, ale pokud použijete nastavení přes parametry modulu parport_lowlevel (parport_pc), není zde nic. Pro každý port je také přítomen adresář *parport/parportX/devices*. Ten obsahuje informace o konkrétním zařízení připojeném na port. Zařízení může být více a každé zde má svůj vlastní podadresář. V souboru *active* je

řečeno, které zařízení je právě aktivní. U mne je to nyní *plip0* a stejnojmenný adresář obsahuje soubor *timeslice*. Ten je opět při registraci zařízení zkopírován z *default* a ovladač zařízení jej má možnost změnit (a nebo ignorovat).

fs

Tímto se dostáváme k informacím o souborových systémech. Obsah je následující:

```
binfmt_misc/        file-max      lease-break-time
dentry-state        file-nr       leases-enable
dir-notify-enable   inode-nr      overflowgid
dquot-nr            inode-state   overflowuid
```

Prvním souborem je *dentry-state*. V něm jsou ukryté informace o stavu adresářové cache. První je počet alokovaných, následuje počet volných položek, dále pak počet vteřin, po kterých může být položka uvolněna v případě nedostatku paměti. Zbytek jsou napevno nuly. Příkladem:

```
5636    5020    45      0       0       0
```

Další dvojicí souborů jsou informace o kvótách. Vždy je přítomen *dquot-nr*, v něm jsou po řadě počet alokovaných a počet volných záznamů kvót. V mém případě obě nulové. Pokud jsou kvóty k dispozici, je zde ještě *dquot-max* s maximálním počtem položek této cache. Opět lze nastavit zápisem do souboru. Obdobně funguje i dvojice *file-nr* a *file-max*. V souboru *file-nr* je opět počet alokovaných, počet volných a maximální počet popisovačů souborů. Maximum lze opět číst a nastavit přes *file-max*. Opět obdobnou funkci má soubor *inode-nr* (alokované prvky cache i-uzlů a volné). V souboru *inode-state* mají první dvě položky stejný význam jako v *inode-nr*, třetí je rozdíl aktuálního počtu položek v cache a maximálního počtu. Toto číslo je větší než nula v případě, že je cache přeplněná. Protože Linux alokuje objekty do cache po stránkách (díl o [http://www.root.cz/clanek.php4?id=779] /proc/slabinfo), může být alokováno více než povolené maximum. Zbytek jsou opět nuly.

Dvojice souborů *overflow[gid|uid]* udává maximální přípustné GID respektive UID pro vlastnictví souborů.

dir-notify-enable umožňuje povolit (1) nebo zakázat (0) oznamování změn v adresářích (jako vytvoření, smazání souboru, zápis nebo přístup k souboru). Programově se to provádí pomocí (v man zatím nedokumentovaného - RH7.0CZ) volání fcntl(F_NOTIFY). Stejně tak *leases-enable* ovládá povolení nebo zakázání souborových zámků typu *lease*.

Adresář *binfmt_misc* obsahuje rozhraní pro podporu cizích binárních formátů. Toho se využívá například při emulaci pomocí *wine*. Pokud chceme, aby se windowsovská binárka spouštěla stejně jako Unixová (zadáním jména z příkazové řádky atd.), zaregistrujeme binární formát Windows a jako interpretr mu přiřadíme *Wine*. Stejně to může fungovat s dosovými aplikacemi a *Dosemu* nebo třeba i se skripty libovolného skriptovacího jazyka (zde je ovšem asi lepší konstrukce #!/bin/interpreter). Soubory se dají rozpoznávat pomocí hodnoty prvních několika bajtů nebo třeba přípony. Po zaregistrování pro *Wine* vypadá obsah následovně:

```
register  status  windows  windowsPE
```

soubor *register* slouží k registraci nových formátů, zapsáním řádku (s požadovaným formátováním) se přidá další formát. *status* obsahuje "enabled". Zbytek jsou informační soubory pro jednotlivé formáty. Příkladem třeba *windowsPE*:

```
enabled
```

```
interpreter /usr/bin/wine
offset 0
magic 5045
```

Zde je vidět interpreter (v tomto případě spíše řekněme zavaděč) a způsob identifikace. V našem případě se kontroluje hodnota 5045 na začátku souboru (offset 0).

Ještě mi nedá, abych neupozornil na malou "zradu" v posledních jádrech. Nyní je třeba souborový systém *binfmt_misc* ručně připojit! Například touto řádkou do *fstabu*:

```
none    /proc/sys/fs/binfmt_misc    binfmt_misc    gid=5,mode=620   0 0
```

kernel

Na tomto místě je možné najít základní konfiguraci jádra Linuxu. Je toho tu proto docela dost:

```
cad_pid          msgmax          printk        tainted
cap-bound        msgmnb          random/       threads-max
core_uses_pid    msgmni          rtsig-max     version
ctrl-alt-del     osrelease       rtsig-nr
domainname       ostype          sem
hostname         overflowgid     shmall
hotplug          overflowuid     shmmax
modprobe         panic           shmmni
```

Začneme pěkně podle abecedy. První dva soubory jsou přístupné pouze pro superuživatele. *cad_pid* udává číslo procesu, kterému je v případě zakázání restartu pomocí kombinace CTRL-ALT-DEL poslán signál SIGINT. Standardně je to proces s PID rovno 1, tedy *init*. Zda je povolen restart, nebo ne, udává hodnota (1=povolen, 0=zakázán) v souboru *ctrl-alt-del*. Ovšem pozor, standardně je tento restart zakázán, a tak je informován *init*, který potom podle */etc/inittab* spustí klasickou ukončovací sekvenci s vypínáním procesů atd. Pokud je naopak tento restart povolen, provedou se pouze speciálně zaregistrované operace v jádře (*funkce register_reboot_notifier()*). Tuto registraci provádějí především různé ovladače, které potřebují před restartem nastavit své zařízení do takového režimu, aby se příště opět úspěšně zavedlo a inicializovalo. Příkladem mohou být zvukové karty maestro, některé ovladače k SCSI zařízením nebo některá RAID zařízení.

Další soubor se jmenuje *cap_bound*. Ten udává masku dostupných oprávnění (capabilities). Takovým oprávněním může být například CAP_SYS_RAWIO pro přímý přístup k diskovým zařízením, CAP_SYS_MODULE pro manipulaci s moduly jádra nebo CAP_SYS_CHROOT pro provedení systémového volání *chroot()*.

Soubor *core_uses_pid* je další přepínač. Pokud je zapnut (1), jádro při vytváření core souboru připojí i číslo PID rušeného procesu. To, že víte, jaké měl proces PID, třeba až tak nepomůže, ale hlavně pokud spadne více procesů zároveň, core se nepřepíše a budou k dispozici všechny.

domainname a *hostname* udávají jednoduše příslušná jména počítače.

hotplug obsahuje název procesu (standardně */sbin/hotplug*), který je spuštěn poté, co byla detekována změna konfigurace na takových rozhraních jako USB. Ten má potom za úkol provést de/inicializaci daného (USB) zařízení. Stejně tak *modprobe* obsahuje umístění příkazu *modprobe*. Ten je potřeba při automatickém nahrávání modulů on-demand přes *kmod*.

Skupina souborů *msgmax*, *msgmnb* a *msgmni* nastavuje chování vrstvy jádra pro zasílání zpráv. První udává maximální velikost jedné zprávy, druhý maximální počet zpráv ve frontě. Poslední udává maximální počet identifikátorů front zpráv = tedy počet front. Podobné je to pro segmenty sdílené paměti. *shmmax* obsahuje maximální velikost sdíleného segmentu, *shmmni* maximální počet segmentů v systému a *shmall* počet stránek, které je systém celkově schopen poskytnout jako sdílenou paměť. U semaforů je vše jednodušší a vše je v jednom souboru - *sem*, který obsahuje čtyři čísla. První udává maximální počet semaforů na proces, druhé maximální počet semaforů v systému, třetí maximální počet operací pro jedno volání *semop()* a poslední maximální počet identifikátorů semaforů.

No už to není podle abecedy, ale nevadí. Další skupinou jsou informace o verzích systému. Sem patří *osrelease*, kde je verze jádra (v mém případě *2.4.13*), *ostype*, kde je *Linux*, a *version*, kde je datum kompilace (*#2 Po říj 22 19:57:20 CEST 2001*)

♀ Soubory *overflowgid* a *overflowuid* už jsme tu jednou měli, teď se jen vztahují místo souborů na identifikace procesů.

Soubor *panic* obsahuje počet vteřin, které systém čeká v případě stejnojmenného stavu, než se restartuje. Pokud je zde nula, zůstane systém viset.

printk nastavuje priority pro tisk zpráv z jádra. Obsahuje čtyři hodnoty. Zprávy s vyšší prioritou než první budou tištěny na konzoli, zprávy bez explicitní priority budou tištěny s touto (druhou v pořadí) prioritou, třetí je nejmenší možná hodnota pro úroveň logování a nakonec poslední číslo je implicitní hodnota pro logování na konzoli (tedy pro první číslo).

Soubory *rtsig-max*|*-nr* určují chování tzv. realtime signálů (dle normy POSIX). Tyto signály se liší tím, že pokud jich přijde více, než stihneme první z nich zpracovat, budou ostatní uloženy do fronty. Aktuální (-nr) a maximální (-max) velikost této fronty (pro každý proces a signál) určují právě tyto soubory.

Zbývají poslední dva soubory. *thread-max* udává maximální počet najednou spuštěných procesů (respektive vláken). Pokud je tento počet překročen, vrátí *fork()* chybu. Soubor *tainted* určuje "nakažení" jádra "nezdravými" moduly. Stav je reprezentován dvoubitovým číslem. Spodní bit nastaven znamená, že byl použit modul s jinou (i žádnou) licencí než GPL. Vrchní bit říká, že byl při vkládání použit parametr *--force*. No o tomhle si můžete myslet, co chcete, ale je to tam. K činnosti je potřeba *modutils >= 2.4.9*.

Ještě zbývá jeden adresář. Jmenuje se *random* a obsahuje nastavení pro generátor náhodných čísel. Jedná se o jaderný generátor, nikoliv ten z *[g]libc* (ale ten ho může teoreticky využívat, i když to myslím momentálně nedělá). Generátor je přístupný přes zařízení */dev/random* a */dev/urandom*. První z nich je kvalitnější, hlídá vydávanou posloupnost (množství generovaných bajtů) v závislosti na stavu kvality aktuální entropie v jádře. Zjednodušeně řečeno první soubor vám dá výstup jen je-li přesvědčen o jeho kvalitě, druhý soubor vám dá tolik čísel, kolik chcete, ale jejich kvalitu si musí ohlídat aplikace. Nastavení chování generátoru je možné přes tyto soubory v adresáři */proc/sys/kernel/random*:

```
boot_id  entropy_avail  poolsize  read_wakeup_threshold
uuid     write_wakeup_threshold
```

boot_id a *uuid* jsou dva soubory, které poskytují náhodná čísla. První z nich tzv. Boot ID UUID a druhé při každém čtení nové UUID. Soubor *poolsize* udává velikost bufferu, přes který jsou vytvářena náhodná čísla. Čím větší, tím kvalitnější je generovaná posloupnost. Počet čísel (resp. bitů), které je možno přes */dev/random* aktuálně získat, lze nalézt v

souboru *entropy_avail*. Pokud budete číst /dev/random, bude se číslo zde uvedené zmenšovat a v klidu pak zase narůstat na maximum (u mne 4096). Posledními soubory jsou *read|write_wakeup_threshold*. Ty udávají minimální počet bytů ve výše zmiňovaném bufferu, při kterém je možno odblokovat procesy čekající načtení respektive zápis do */dev/random*.

net

Tak dle ohlasů v diskusích na tohle hodně z vás netrpělivě čekalo. A je to konečně tady. Teda po pravdě spíše není. A bohužel (zatím) taky nebude. Než začnete litovat toho, že jste tyto články vůbec četli, když to nejdůležitější tu chybí, že jste mi poslali nějaké Qčko, když jsem vás teď tak zklamal, dočtěte alespoň prosím tento odstavec, ať k tomu můžu ještě něco říct. Za prvé, jak jsem již řekl v úvodu - nestíhám. Pokud by měl být i tento adresář popsán ve stejném rozsahu, bylo by to na minimálně dalších 6 až 8 dílů. To ale není to nejpodstatnější. Hlavní je, že o sítích (obecně i pod Linuxem) mám (dle svého názoru) **značně** omezené znalosti. A především věcí, kterých se týká tento adresář. Další věc je, že jestliže znalostí mám pramálo, zkušenosti nemám téměř žádné. S těmito vyhlídkami prostě nevidím cestu, jak pokračovat v tomto adresáři v použitelné kvantitě a hlavně kvalitě. Proto raději na rovinu přiznám, že na to nemám a budu doufat, že se najde někdo sítí znalý, kdo se toho třeba ujme a napíše vlastní články o této problematice. Doufám, že mne alespoň většina pochopí a můj mail nebude zaplaven výhružnými dopisy :(

proc

Zde zatím není vůbec nic. (Teď přesněji, dynamicky přidávaného jsem nic nenašel a staticky se tam nedává určitě nic).

vm

Dostáváme se k nastavení a informacím o správě paměti. Zde je vhodné upozornit, že správa paměti je jedním z hlavních rozdílů mezi Linusovou a Alanovou větví. Nevím, nakolik se to dotkne rozhraní v /proc, ale každopádně implementace je rozdílná. Adresář *vm* na mém počítači (2.4.13) má tento obsah:

```
bdflush  kswapd  overcommit_memory  page-cluster  pagetable_cache
```

Soubor *bdflush* kontroluje stejnojmenného démona. Obsahuje devět hodnot s následujícím významem. První je procento zašpinění blokové cache, které démona aktivuje. "Špinavý" je takový blok, který má jiný obsah než jeho obraz na disku - tedy blok, do kterého se zapisovalo. Jestliže je takto modifikovaných bloků více než (v našem případě) 40%, spustí se *bdflush* démon a provede úklid - zápis na disk. Větší hodnota znamená efektivnější práci s diskem, menší zase lepší dobu odezvy systému. Pak následují tři nuly. Zbývají tři čísla s tímto významem: první udává (v jiffies), jak dlouho může maximálně Linux čekat, než zapíše data z bufferu na disk. Zde se jedná o metadata, zatímco další číslo je to samé pro vlastní data souborového systému (je menší). Hodnota v jiffies se na PC na sekundy převede prostým vydělením 100. Obecně počet jiffies za jednu sekundu udává konstanta *HZ* (jinak má třeba Alpha nebo RealTime Linux). Poslední z trojice udává procento zašpinění, při kterém je *bdflush*spuštěn synchronně. Rozdíl oproti první hodnotě je ten, že *bdflush* je spouštěn pravidelně v jistých intervalech. Při těchto kontrolách bere první hodnotu. Pokud ale při práci s cache vyleze procento nad druhou z hodnot, je bdflush spuštěn ihned. Pak ještě následuje několik nul.

Dalším souborem je *kswapd*. To je další jaderný démon, který slouží k odswapování paměti, když je příliš fragmentovaná nebo dochází. Soubor obsahuje tři hodnoty. První udává

maximální množství stránek, které má démon povoleno odswapovat při jednom vyvolání. Dle dokumentace to není přímo počet stránek, ale spíše velikost paměti v KB. Druhé číslo je minimum stránek, které by měl démon odswapovat během jednoho běhu. Poslední číslo udává, kolik stránek démon zapíše při jednom přístupu na disk. Správa paměti je poslední dobou jedna z nejvíce se měnících oblastí. Když jsem se díval do zdrojáků na aktuální stav, jediný parametr, který jsem našel, že má na něco vliv, je poslední číslo. První dvě se nikde nevyužívají (dle mého nálezu).

overcommit_memory je přepínač (0=Off, 1=On), který udává, zda se bude před voláním *malloc()* (respektive obecně před uživatelskou alokací paměti) provádět kontrola na dostatek volné paměti (volná paměť je zde free_mem + free_swaps). Pokud je 0, kontrola se neprovádí. Toto chování má smysl, pokud se používají programy, které si alokují paměť dopředu a potom ji třeba vůbec nepoužívají (nebo alespoň ne všechnu).

page-cluster obsahuje hodnotu, která udává počet stránek čtených najednou při výpadku paměti. Počet stránek je dán jako 2^page-cluster. Maximální hodnota, která má smysl, je pět, neboť veškeré operace se provádějí nad skupinami 32 stránek.

pagetable_cache udává minimální a maximální velikost cache (v položkách cache - tedy počet adresářových stránek) pro adresářové stránky. Jedná se o stránky paměti, kde jsou uloženy informace o překladu logických adres na adresy fyzické. Na jednoprocesorových počítačích toto nastavení nemá až tak velký význam, ale na SMP strojích přístup do této cache nevyžaduje zámek na paměťový subsystém jádra. Proto větší cache zvyšuje výkon takovéto konfigurace.

Tak a to je opravdu všechno. Vím, že v průběhu jsem sliboval se vrátit ještě podrobně k některým problémům. Budu na to pamatovat, ale spíše jako další články po malé pauze. Doufám, že jste se v tomto seriálu dozvěděli něco nového a někdy příště se budu těšit na shledanou...

Autor: **Tomáš Kašpárek** <kasparek (zavináč) fit (tečka) vutbr (tečka) cz>
Oblast: **Různé** (http://root.cz/index.php4?oblast=5)

Verze článku pro čtení: http://root.cz/clanek.php4?id=923