evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
Slezská univerzita v Opavě
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Šárka Vavrečková

Operační systémy

cvičení

Část II: Linux

Slezská univerzita v Opavě
Filozoficko-přírodovědecká fakulta
Ústav informatiky

Opava, poslední aktualizace 14. května 2013

Anotace: Tento dokument je určen pro studenty druhého ročníku IVT na Ústavu informatiky Slezské univerzity v Opavě. Obsahuje látku probíranou ve cvičeních předmětu *Operační systémy* v části semestru věnované Linuxu.

Probíraná látka navazuje na předmět *Praktikum z operačních systémů*. Předpokládá se základní orientace v běžných nástrojích s grafickým rozhraním v Linuxu, adresářové struktuře, zařízeních, přístupových oprávněních a principu běhu procesů.

Operační systémy cvičení
Část II: Linux

RNDr. Šárka Vavrečková, Ph.D.

Dostupné na: http://fpf.slu.cz/~vav10ui/opsys.html

Ústav informatiky
Filozoficko-přírodovědecká fakulta
Slezská univerzita v Opavě
Bezručovo nám. 13, 746 01 Opava

Sázeno v systému LaTeX

Tato inovace předmětu *Operační systémy* je spolufinancována Evropským sociálním fondem a Státním rozpočtem ČR, projekt č. CZ.1.07/2.3.00/0 9.0197, „Posílení konkurenceschopnosti výzkumu a vývoje informačních technologií v Moravskoslezském kraji“.

Předmluva

Co najdeme v těchto skriptech

Tato skripta jsou určena pro studenty informatických oborů na Ústavu informatiky Slezské univerzity v Opavě. Ve cvičeních předmětu Operační systémy se v první části semestru probírají operační systémy z rodiny Windows a v druhé části semestru unixové systémy se zaměřením na Linux, tato skripta se využívají právě v druhé části semestru.

Navazujeme na obdobná skripta z předmětu Praktikum z operačních systémů, tedy předpokládají se již základní znalosti práce v Linuxu (alespoň v grafickém prostředí), orientace v adresářové struktuře a přístupových oprávněních.

Na to, že jde o látku pouze pro část semestru, se skripta mohou zdát značně rozsáhlá. Důvodem je zařazení mnoha ukázkových (řešených) příkladů a motivačních (neřešených) úkolů, které mají pomoci při pochopení a osvojení si učiva. U některých příkazů jsou uvedeny také jejich obvyklé výstupy, také z důvodu často nedostatečných přístupových oprávnění studentů v učebnách.

Některé oblasti jsou také „navíc“ (jsou označeny ikonami fialové barvy), ty nejsou probírány a ani se neobjeví na testech – jejich úkolem je motivovat k dalšímu samostatnému studiu nebo pomáhat v budoucnu při získávání dalších informací dle potřeby v zaměstnání.

Značení

Ve skriptech se používají následující barevné ikony:

- Nové *pojmy*, názvy souborů, obecné postupy, značení v příkazech, používané symboly, apod. jsou značeny modrým symbolem v poznámce na okraj, který vidíme také zde vpravo.

- Konkrétní *postupy* a nástroje (příkazy, programy, soubory, skripty), způsoby řešení různých situací, do kterých se může administrátor dostat, syntaxe příkazů atd. jsou značeny také modrou ikonou.

- Zelená je také ikona vyznačující sekce, které jsou *opakováním* učiva z přednášek, jiné části těchto skript nebo z předchozího semestru (předmětu Praktikum z operačních systémů).

- Zvlášť je také vyznačen text, který je obvykle *výstupem* probíraných příkazů (včetně těch, jejichž spuštění vyžaduje vyšší přístupová oprávnění) nebo může jít o obsah některých souborů. Barva ikony je oranžová.

- Některé části textu jsou označeny fialovou ikonou, což znamená, že jde o *nepovinné úseky*, které nejsou probírány (většinou; studenti si je mohou podle zájmu vyžádat nebo sami prostudovat). Jejich účelem je dobrovolné rozšíření znalostí studentů o pokročilá témata, na která obvykle při výuce nezbývá moc času.

- Fialová ikona se objevuje také u některých neřešených úkolů. Tyto úkoly *nejsou povinné* a studenti se jimi mohou zabývat podle svých zájmů (jsou obvykle náročnější než jiné úkoly).

- Žlutou ikonou jsou označeny odkazy, na kterých lze získat *další informace* o tématu. Může jít o způsoby získání nápovědy, nejčastěji však u této ikony najdeme odkazy na internet.

- Červená je ikona pro *upozornění* a poznámky.

Opticky (ale už ne barevně) jsou odlišeny také řešené příklady a neřešené úlohy. Příklady jsou číslovány, čísla slouží k jednoduchému odkazování na tyto příklady.

Příklad 0.1

Takto vypadá prostředí s příkladem. Číslo příkladu je 0.1.

Úkoly

Otázky a úkoly, náměty na vyzkoušení, které se doporučuje při procvičování učiva provádět, jsou uzavřeny v tomto prostředí. Pokud je v prostředí více úkolů, jsou číslovány.

Jak probíhají testy

Na zápočtových testech z předmětů týkajících se operačních systémů lze používat počítač, a to nápovědu a nástroje běžně dostupné ve standardní instalaci daného operačního systému, ale bez přístupu na Internet. Nejsou dovoleny dokumenty vlastní ani cizí výroby, které nejsou součástí standardní instalace, nelze používat internetový prohlížeč ani jiný způsob přístupu na externí zdroje informací.

Na stránkách předmětu je k dispozici orientační seznam otázek a úkolů, které se mohou objevit na testu, ovšem v testu se mohou objevit mírné odlišnosti (například v názvech zpracovávaných souborů či adresářů, jiné přepínače příkazů, apod.).

Tato skripta plně pokrývají odpovědi na otázky, které se mohou objevit v zápočtovém testu č. 2 (Linux) v předmětu Operační systémy.

Obsah

Kapitola 1

Textový režim v Linuxu

Tato kapitola je úvodem do správy operačních systémů s jádrem GNU/Linux (dále Linux), většina postupů s mírnou modifikací platí i pro jiné unixové (unix-like) systémy. Podíváme se především na práci se shellem, budeme navazovat na předchozí semestr.

V unixových systémech včetně Linuxu je možné používat několik různých shellů. My se zde budeme věnovat pouze shellu `bash` *– Bourne Again Shell, se kterým jsme se už v předchozím semestru setkali.*

1.1 Textové shelly a příkazy

Navazujeme na učivo z předchozího semestru. Kdo měl problémy se základy textového režimu v shellu `bash`, ten si urychleně zopakuje příslušnou kapitolu ze skript předmětu *Praktikum z operačních systémů* (http://fpf.slu.cz/~vav10ui/obsahy/pos/skripta/pos_linux.pdf).

1.1.1 Textové shelly v Unix-like systémech

Původní v Unixu byl *Bourne Shell* (vznikl roku 1978 a označoval se jednoduše `sh`), jehož autor je Stephen Bourne. O něco později (v době přepisu Unixu do jazyka C) vznikl jako alternativa *C Shell* (označoval se `csh`). Zatímco Bourne Shell byl určen spíše pro psaní skriptů a umožňoval lepší spolupráci programů (včetně směrování), C Shell hodně inspirovaný jazykem C se orientoval hlavně na interaktivní práci přímo v příkazovém řádku (například používání historie pomocí speciálních příkazů) a také přinesl pokročilejší správu úloh.

V současné době se už nesetkáme přímo s Bourne Shellem a C Shellem, ale s jejich potomky:[1]

- *Toronto C Shell* (`tcsh`) je rozšíření `csh` o další možnosti, například používání šipek při práci s historií příkazů, `tcsh` také odstranil některé chyby související s činností ve skriptu, které se vyskytovaly v `csh`, je obvykle nainstalován v adresáři `/bin/tcsh` nebo `/usr/bin/tcsh` nebo `/usr/local/bin/tcsh` (záleží na distribuci),

[1] Pěkné porovnání shellů (dokonce včetně Příkazového řádku ve Windows) najdeme například na stránce http://en.wikipedia.org/wiki/Comparison_of_command_shells.

- *Korn Shell* (`ksh`) – syntaxe příkazů odpovídá Bourne Shellu, ale jinak většinu vlastností přejal z `tcsh`, jde vlastně o hybrid shellů `sh` a `tcsh`, je nainstalován v adresáři `/bin/ksh` nebo `/usr/bin/ksh`,
- *Bourne Again Shell* (`bash`) je nejobvyklejším shellem v Linuxu a je velmi podobný `ksh` (je jednodušší, například oproti `ksh` neobsahuje podporu racionálních čísel a vícedimenzionálních polí), je nainstalován v `/bin/bash` nebo `/usr/bin/bash` nebo `/usr/local/bin/bash`,
- *Debian Almquist Shell* (`dash`) je potomkem shellu `ash` (Almquist Shell) ze systému FreeBSD, jak název napovídá, můžeme se s ním setkat u Debianu a jeho potomků (včetně Ubuntu); je podobný shellu `bash`, ale mírně osekaný (některé vlastnosti shellu `bash` nepodporuje) a rychlejší,
- *Z Shell* (`zsh`) je naopak vybavenější než `bash`, a to směrem k vědeckým výpočtům (je srovnatelný spíše s shellem `ksh`).

V Linuxu je vždy (nebo téměř vždy) nainstalován shell `bash`, a kromě něho i několik dalších. Seznam použitelných shellů (těch, které máme k dispozici) je v souboru `/etc/shells`. Mezi shelly se přepínáme příkazem `chsh` (je to zkratka z CHange SHell).

Poznámka: Linuxový shell je možné ve skutečnosti používat i ve Windows. Výborným řešením je instalace prostředí *Cygwin*.[2] Jedná se o podsystém (podobně jako v Linuxu máme Wine, Cygwin má přesně opačnou funkci – ve Windows představuje rozhraní pro běh unixových aplikací). Instalace Cygwinu není moc náročná. Stáhneme si instalační program, spustíme (budou se stahovat další soubory z Internetu, tedy musíme být i nadále připojeni) a v grafickém rozhraní zvolíme balíčky, které chceme nainstalovat (této fázi věnujte hodně pozornosti, je třeba toho zatrhnout poměrně hodně). Na ploše se objeví nová ikona sloužící ke spuštění konzole s bash.

Příklad 1.1

Ukážeme si použití příkazu `chsh`.

`man chsh` spustíme prohlížeč nápovědy pro příkaz `chsh`, zjistíme si syntaxi příkazu, tedy jaké parametry podporuje, program `man` ukončíme stiskem klávesy Q,

`chsh -l` („l“ jako „list“) vypíše seznam shellů, které je možné si nastavit, tedy vypíše obsah souboru `/etc/shells`,

`chsh -s /bin/zsh` právě jsme svůj *login shell* (tj. shell spouštěný při našem přihlašování) nastavili na Z Shell, změna se projeví buď po restartu konzoly nebo po odhlášení/přihlášení.

Pokud je shell, který chceme takto zapnout, nainstalován, ale přitom není uveden v souboru `/etc/shells`, nebude to fungovat a zobrazí se chybové hlášení. Úpravou tohoto souboru může totiž administrátor zakázat používání těch shellů, které nepovažuje za bezpečné (prostě jejich řádky vymaže), resp. povolit pouze konkrétní shelly, které sám určí.

Další možnou příčinou selhání příkazu je zadání nesprávné cesty k souboru shellu (jde přece o program), zde stačí zjistit, kde konkrétně je shell nainstalován.

[2] http://www.cygwin.com/

Z grafického režimu (obvykle to bývá některé z desktopových prostředí KDE nebo GNOME) spustíme některý terminál nebo konzolu (například `konsole` nebo `xterm` – program představující rozhraní pro zadávání příkazů, obvykle můžeme mít najednou spuštěno více shellů – relací a přepínat se mezi nimi pomocí záložek). V KDE bývají terminály a konzole dosažitelné přes nabídku K, menu Terminály (případně K ⇨ Spustit ⇨ napíšeme `konsole` nebo jiný název terminálu či konzole).

V moderních unixových systémech včetně Linuxu máme souborové manažery, kteří nám usnadňují práci především při operacích s adresáři a soubory. Jsou to například *Midnight Commander* (spouští se příkazem `mc`) pracující v konzoli (a tedy ve kterékoliv distribuci s jakýmkoliv grafickým prostředím), *Konqueror* v desktopovém prostředí KDE, *Nautilus* v desktopovém prostředí GNOME, atd.

1.1.2 Skripty

Skripty (skriptové soubory) jsou textové spustitelné soubory, které pro své spuštění potřebují interpret (třeba některý shell). Soubory se skripty obvykle mívají buď příponu SH, a nebo jsou bez přípony (ale můžeme se setkat i s příponou TXT, je to celkem jedno).

Každý skript je psán vždy pro určitý shell nebo programovací jazyk (například `bash` nebo `perl`) a v jiném nemusí fungovat. Proto často bývá na prvním řádku skriptu *identifikace shellu*. Tento řádek vždy začíná dvojicí znaků `#!`, pokud ovšem se ve skriptu nachází.

Příklad 1.2

První řádek skriptu může vypadat třeba takto:

- `#!/usr/bin/tcsh` znamená, že skript má být interpretován shellem Toronto C Shell,
- `#!/usr/bin/bash` určuje skript s příkazy shellu Bourne Again Shell,
- `#!/usr/bin/perl` (adresa může být jiná, záleží na umístění spustitelného programu `perl`) je skript psaný v programovacím jazyce Perl.

Aby bylo možné skript spouštět v textovém režimu napsáním jeho názvu, musí být označen jako spustitelný, tj. je třeba nastavit příznak `x` v přístupových oprávněních. V opačném případě musíme skript spouštět jako parametr interpretačního programu (například `bash soubor.sh`), ale i tak je vhodné tento soubor označit jako spustitelný nastavením příslušného příznaku.

1.2 Nápověda

Nápovědu k příkazům můžeme získat více způsoby:

- zobrazením manuálové stránky příkazu, a to příkazem `man`,
- někdy je implementován příkaz `apropos`, který použijeme, když nevíme, jak se příkaz nazývá,
- příkaz `whatis` použijeme, když jsme narazili na příkaz (spustitelný soubor), ale nevíme, co provádí (vypíše se krátká informace o příkazu),

- příkaz `info` vypíše krátkou informaci o příkazu,
- v grafickém režimu,
- v Linuxu existují *dokumenty HOWTO* („jak na to“), a to buď přímo v jednotlivých distribucích nebo na internetu, obsahují přímo rady, jak postupovat v určitých situacích,
- obdobně jsou k nalezení *dokumenty FAQ* (Frequently Asked Questions) pro často pokládané otázky,
- v internetovém prohlížeči se také dostaneme na manuálové stránky, například při zadání `man chsh` do adresního řádku (tam, kde bychom jinak zadali `http://...`) se nám automaticky zobrazí manuálová stránka příkazu `chsh` na některém serveru přímo určeném pro manuálové stránky.

Na internetu je hodně stránek věnovaných shellu `bash`, například:

- http://www.gnu.org/software/bash
- http://tldp.rog/LDP/abs/html (skripty)
- http://www.abclinuxu.cz/clanky/show/46130

Získat nápovědu bychom měli umět už z předmětu Praktikum z operačních systémů.

1.3 Práce s adresáři a soubory

1.3.1 Prohledávání

K prohledávání adresářové struktury (hledání souborů) používáme příkaz `find`, k prohledávání obsahu souborů zase `grep`. Program `grep` lze použít také k nalezení názvu souboru v zadaném adresáři – stačí na vstup programu poslat výpis adresáře. Pro nalezení cesty ke spustitelnému souboru (příkazu) používáme příkaz `whereis`.

`find`
: prohledávání adresářové struktury, hledáme soubory (a adresáře) se zadaným názvem nebo jinými parametry, také poskytuje možnost omezeného zpracování souborů nebo ovlivňování formátu vlastního výstupu

  `find *.txt -print` vypíše všechny soubory se zadanou příponou v pracovním adresáři

  `find -name '*.txt' -print` vypíše všechny soubory se zadanou příponou rekurzívně (apostrofy jsou nutné!)

  `find /usr -name soubor -print` hledá *soubor* v adresáři (lze použít i jednoduché regulární výrazy), přepínač `-print` způsobí výpis úplného jména (prohledáváme adresář `/usr`), pokud místo `-name` napíšeme `-iname`, nerozlišují se malá a velká písmena

  `find -user uzivatel -print` hledá soubor vlastníka *uzivatel* (prohledáváme pracovní adresář)

  `find -uid 1520 -name prac* -print` hledáme soubor (rekurzívně), jehož název začíná zadaným řetězcem a jeho vlastník je uživatel se zadaným UID (lze také zadat GID, parametr `-gid`)

`find ~ -iname desk* -print` vypíše vše v pracovním adresáři a jeho podadresářích, co začíná řetězcem „desk“ bez rozlišování malých a velkých písmen

`find -mtime 2 -print` hledáme soubor, který byl modifikován (modified) naposledy před právě 2 dny (přesněji před 2*24 hodinami)

`find -mtime +2 -print` hledáme soubor, který byl modifikován (modified) naposledy před více než 2 dny

`find -mtime -2 -print` hledáme soubor, který byl modifikován (modified) naposledy před méně než 2 dny

`find -atime 3 -print` k souboru bylo přistupováno (accessed) naposledy před 3 dny; obecně: můžeme vyhledávat vše, co bylo modifikováno (m), přistupováno (a) nebo měněno (c, change) v zadané době, zde jde o dny, ale existují volby i pro minuty

`find -type d -size +10k -print` vypíšou se soubory typu adresář (d), jejichž velikost je více než 10 kB

`find -type f -size -2M -print` vypíšou se běžné soubory (f), jejichž velikost je méně než 2 MB (jako typ souboru je možné použít `d` pro adresář, `f` pro běžný soubor, `c` pro znakové zařízení, `b` pro blokové zařízení, `p` pro pojmenovanou rouru, `l` pro symbolický odkaz a `s` pro socket)

`find -perm /u+w,g+w -print` chceme vypsat celé názvy souborů, v jejichž přístupových oprávněních je právo zápisu pro uživatele A/NEBO právo zápisu pro skupinu

`find -perm /u+w,g+w -print` chceme vypsat celé názvy souborů, v jejichž přístupových oprávněních je právo zápisu pro uživatele A/NEBO právo zápisu pro skupinu

`find -perm -u+w,g+w -print` chceme vypsat celé názvy souborů, v jejichž přístupových oprávněních je právo zápisu pro uživatele A ZÁROVEŇ právo zápisu pro skupinu

`find / -perm -4000 -print` vypíše celé názvy souborů (rekurzívně v kořenovém adresáři), které mají nastaven SUID bit (vzpomeňte si z předchozího semestru, co to znamená), SUID, SGID a Sticky bity se dají zadat pouze zadáním číselného módu souboru

`find / \( -perm -4000 -o -perm -2000 \) -type f -exec ls -la{} \;`
teď je příkaz o něco složitější; všímáme si všech souborů s nastaveným SUID *nebo* SGID bitem (takto zapisujeme disjunkci, závorky jsou nutné), má jít o běžné soubory (včetně spustitelných, ne adresáře), na konci je místo obyčejného výpisu volba na spuštění příkazu, v jehož výstupu zjistíme informace o souboru

`grep [přepínače] reg_výraz [soubor ...]`

pro změnu prohledává textové soubory; vypíše řádky textového souboru, které obsahují zadaný řetězec, řetězec je zadáván jako regulární výraz, někt. přepínače:

- ***-i*** nerozlišuje malá a velká písmena
- ***-l*** pouze vypíše názvy souborů, ve kterých nalezl shodu
- ***-n*** vypíše také číslo řádku
- ***-r*** rekurzívně zpracovává i podadresáře
- ***-o*** vypíše jen nalezený řetězec, ne celý řádek
- ***-c*** v souborech pouze spočítá výskyty nalezeného řetězce

V regulárním výrazu můžeme používat symboly shrnuté v tabulce 1.1.

Prvek	Význam
`.`	Libovolný znak
`?`	Nula nebo jeden výskyt předcházejícího řetězce
`*`	Nula nebo více výskytů předcházejícího řetězce
`+`	Jeden nebo více výskytů předcházejícího řetězce
`{m}`	`m` opakování předcházejícího řetězce
`{m,n}`	`m` až `n` opakování předcházejícího řetězce
`{m,}`	`m` nebo více opakování předcházejícího řetězce
`^`	Začátek řádku
`$`	Konec řádku
`[třída]`	Jakýkoli (jeden) znak z množiny v závorkách
`[^třída]`	Jakýkoli znak mimo prvky množiny
`[x-y]`	Jakékoli znaky v daném rozsahu (jeden)
`r1 | r2`	logické nebo – buď řetězec `r1`, nebo řetězec `r2`

Tabulka 1.1: Regulární výrazy v příkazu `grep`

Posix sekvence jsou jakési „zkratky“ pro konkrétní typy znaků. Například místo `0-9`, můžeme napsat posix sekvenci `[:digit:]`. Další možnosti najdeme v tabulce 1.2. Protože jsou posix sekvence uzavřeny do hranatých závorek, pak při použití v regulárním výrazu jsou tyto závorky „zdvojeny“ – vnitřní patří posix sekvenci, vnější uzavírají množinu, ze které se vybírá prvek.

Sekvence	Význam
`[:digit:]`	číslice
`[:xdigit:]`	hexadecimální číslice
`[:alpha:]`	písmena
`[:alnum:]`	písmena a číslice
`[:lower:]`	malá písmena
`[:upper:]`	velká písmena
`[:blank:]`	mezera a tabulátor
`[:space:]`	prázdné znaky (mezera, tabulátor, konec řádku, . . .)
`[:graph:]`	viditelné znaky
`[:print:]`	viditelné znaky a mezera

Tabulka 1.2: Posix sekvence při vyhledávání

Příklad 1.3

Ukážeme si použití příkazu `grep`.

`grep -i "vypis" *.txt` hledáme v souborech s příponou `.txt` slovo `vypis` bez rozlišování malých a velkých písmen

`grep -il "vypis" *.txt` hledáme v souborech s příponou `.txt` slovo `vypis` bez rozlišování malých a velkých písmen, pouze vypíše názvy souborů, ve kterých se toto slovo nachází

`grep -cr "#include.*\.[hc]" *.c` u každého souboru s příponou `.c` vypíše počet vkládaných souborů s příponou `.h` nebo `.c`, a protože tečka je významový znak (určuje jeden jakýkoliv symbol), musíme před ni dát zpětné lomítko, aby byla brána jako součást řetězce

`grep -cr "Dear \(Mr.|Ms.|Miss\)" *.txt` chceme počet anglických oslovení přes všechny soubory se zadanou příponou rekurzívně v podadresářích pracovního adresáře

`grep '[0-9]\{6\}/[0-9]\{3,4\}' soubor.txt` v zadaném souboru hledáme rodná čísla ve tvaru 123456/1234 – nejdřív 6 číslic, pak lomítko a tři nebo čtyři číslice; jinak:
`grep '[[:digit:]]\{6\}/[[:digit:]]\{3,4\}' soubor.txt`

`grep '\([[:digit:]]\{1,3\}\)\{4\}' soubor.log` v zadaném souboru hledáme IP adresy ve tvaru 123.123.123.123 – čtyři skupiny číslic, v každé skupině 1 až 3 číslice; jinak: `grep '\([0-9]\{1,3\}\)\{4\}' soubor.log`

V příkazech si můžeme povšimnout, že je nutno rozlišit vyhledávané znaky a „metaznaky“, které v regulárním výrazu slouží k upřesnění vyhledávacího řetězce (například v posledním příkazu kulaté závorky).

Program `grep` je pružnější než `find` (už pro rozsáhlé možnosti regulárních výrazů), proto je často používán nejen k prohledávání textových souborů, ale také k prohledávání struktury adresářů v kombinaci s příkazem `ls`.

Příklad 1.4

Ukážeme si možnosti prohledávání adresářové struktury příkazem `grep`:

`ls -la | grep -ic "^-"` vypíše počet běžných souborů v pracovním adresáři

`ls -la | grep "^-..x"` vypíše všechny běžné soubory (řádek začíná pomlčkou), které jsou spustitelné (vlastník má právo spouštění)

Příklad 1.5

Ukážeme si, jakým způsobem vypíšeme login shell některého uživatele. Předpokládejme, že chceme znát shell uživatele novak. Pak zadáme:

```
cat /etc/passwd | grep "^novak:" | cut -d : -f 7
```

Vypadá to trochu zvláštně. V koloně (rouře) je první příkaz `cat`, který na svůj výstup pošle obsah souboru `/etc/passwd`, což je soubor se seznamem uživatelů a jejich parametrů. Další na řadě je příkaz `grep`, který vybere pouze řádek obsahující na svém začátku jméno hledaného uživatele.

Třetím příkazem v koloně je `cut` (pozor, neplést si s `cat`), který „usekne“ řádek a zobrazí pouze jeho konec. Jeho první volba, `-d`, stanoví dvojtečku na oddělovač „sloupců“. Druhý parametr stanoví, že chceme pouze sedmé pole v pořadí (pole jsou oddělena oddělovačem zadaným v předchozí volbě).

Může se stát, že sice víme, jak se příkaz nazývá a jaké má parametry, a taky je jasné, že je nainstalován, ale přesto ho nelze spustit. Může to být tím, že cesta k tomuto příkazu není zahrnuta v proměnné obsahující cesty ke spustitelným souborům. Pak je možné spustit příkaz s absolutní cestou, ale tu musíme předem zjistit. K tomu slouží příkaz **whereis**.

Příklad 1.6

Zjistíme cesty k některým příkazům.

`whereis ls` zobrazí cestu k příkazu `ls`

`whereis passwd` zobrazí celkem hodně cest k různým souborům, které se takto nazývají; hned první vypsaný řetězec je správný, tedy zjistíme, že příkaz lze spustit s absolutní cestou `/usr/bin/passwd` (nebo jinak, může se lišit v různých unixových systémech), vypsal se také stejnojmenný konfigurační soubor v adresáři `/etc`

`whereis shutdown` příkaz na vypnutí (restart apod.) systému najdeme zjevně v adresáři `/sbin`, protože na serverech ho typicky může používat pouze root

Úkoly

1. Ve výpisu obsahu svého domovského adresáře najděte všechny položky, jejichž název obsahuje řetězec `rc`.
2. Najděte ve svém domovském adresáři všechny soubory, které obsahují řetězec (dvojznak) `#!`.
3. Vypište počet podadresářů ve svém domovském adresáři.
4. Vypište počet všech běžných souborů ze svého domovského adresáře, které jsou skryté (začínají tečkou).
5. Vypište seznam všech symbolických odkazů ve svém domovském adresáři.
6. Sestavte příkaz, který prohledá systém a najde všechny soubory větší než 1024 MB, nechte zobrazit vlastnosti souboru (abyste zjistili vlastníky těchto souborů), případně můžete z řádků vyfiltrovat pouze údaje o vlastníkovi a názvu souboru.

1.3.2 Automatické zpracování

Hromadné změny v obsahu souboru lze provádět více různými programy. Z nejznámějších jsou **sed** (Stream Editor) a **awk**, kde `sed` má jednodušší syntaxi, která dovoluje například vymazávání vzorů odpovídajících regulárnímu výrazu nebo jejich nahrazování jinými vzory, program `awk` je již mnohem komplexnější a jde vlastně o programovací jazyk hodně podobný jazyku C. Zde si ukážeme pouze program `sed`. Základní syntaxe:

sed [přepínače] [skript] [vstupní_soubor]

zpracování výrazů v textovém souboru, některé přepínače:

-e řetězec skript k provedení (ne soubor, ale řetězec!)

-f soubor soubor se skriptem k provedení

Skript má formu `ap`, kde `a` je *adresa* ve zpracovávaném souboru a `p` je *příkaz*, který se na daném místě má provést. Pro zadání adresy a příkazu platí dále nastíněná pravidla.

Adresy mohou být:

- (bez adresy) příkaz se použije na všechny řádky
- `číslo` číslo řádku, který se má zpracovat
- `číslo~krok` všechny řádky od řádku s daným číslem s násobkem daným krokem (tj. `číslo + i*krok`), například `1~2` budou všechny liché řádky
- `$` poslední řádek vstupu
- `/regulární_výraz/` řádky odpovídající regulárnímu výrazu, lomítka jsou nutná, uvnitř používáme vše, co u grep, ale před některé významové znaky dáváme `\` (`\+`, `\?`, `\{...}`, `\|`) Pokud následuje `I`, nerozlišují se malá a velká písmena
- `adresa1,adresa2` všechny řádky v rozmezí adres
- `adresa,+n` všech n řádků od zadané adresy

Příkazy mají tuto formu:

- `d` vymaž nalezený vzor
- `p` vypiš (vytiskni) nalezený vzor na výstup
- `s/co/čím/přepínače` (lomítka jsou součástí výrazu) nahraď `co` `čím`, a to způsobem určeným přepínači:
  - `g` nahraď všechny výskyty
  - `číslo` nahraď jen výskyt s pořadím `číslo`
  - `i` nerozlišuj malá a velká písmena

Příklad 1.7

Na několika příkladech si ukážeme použití programu `sed`:

`sed -e '1,5d' soubor.txt`
odstraní ze zadaného souboru první až pátý řádek

`sed -e 's/<[^>]*>//g' *.html`
ze všech html souborů v daném adresáři odstraní všechny tagy <...> (vnitřní část zajišťuje, že pokud je na řádku více tagů, bude text mezi nimi zachován), dvě lomítka za sebou jsou nutná, protože mezi nimi se vlastně nachází prázdný řetězec `čím` – `g` znamená „nahradit všechny výskyty“

`cat soubor.txt | sed -e 's/\&/\&/g' | sed -e 's/</\</g'`
`| sed -e 's/>/\>/g' > soubor.html` usnadní převod textového souboru do html formy tím, že všechny symboly `&` nahradí patřičným ekvivalentem v HTML kódu, totéž udělá také se znaménky < a >

Příklad 1.8

Nalezené části výrazu jsou zapamatovány a přístupny pod `\1`, `\2`, atd., ukážeme si na příkladu dávkové změny přípony HTM na HTML:

```
ls *.htm | sed -e 's/\(.*\)\.htm/mv \1.htm \1.html/'
```

- příkaz `ls` vypíše obsah adresáře (jako `dir` ve Windows) podle zadaného vzoru – všechny soubory s příponou `htm`,
- ve skriptu použijeme příkaz `s/co/čím/přepínače`,
- reg. výraz pro „co“ znamená nějaké znaky (tj. `\(.*\)`) následované tečkou (`\.`) a původní příponou,
- provede se volání příkazu `mv`, který přejmenuje soubory s příponou HTM na tytéž soubory, ale s příponou HTML (část před příponou je zapamatována v `\1`).

Úkoly

1. Odhadněte, co bude výstupem těchto příkazů (domovský adresář je ~):
   - `ls -la | sed -e 's/r--/---/g'`
   - `ls -la | sed -e '/^d/d'`
   - `ls -la | sed -e 's/^[rwxdl-]*/t prava/g'`
2. Napište příkaz podobný příkazům v předchozím úkolu, který by v uvedeném výpisu smazal vše, co se nachází za poslední tečkou na řádku (v sekci o regulárních výrazech pro vyhledávání si najděte způsob reprezentace konce řádku).

1.3.3 Možnosti vytvoření nového souboru

V předchozím textu jsme se setkali s několika možnostmi vytvoření nového souboru, některé z nich dovolují zároveň do souboru něco uložit. Takže postupně:

- kopírováním a směrováním do souboru
- `touch soubor`
- `cat /dev/null > soubor`
- `cat > soubor`, napíšeme pár řádků, pak stiskneme Ctrl+D

S posledním uvedeným způsobem jsme se setkali i u Windows (použití příkazu `type` v kombinaci se vstupem `CON`). V Linuxu můžeme vytvořit zcela ekvivalentní příkaz:

`cat /dev/tty > soubor` nebo

`cat < /dev/tty > soubor`

ale proč bychom to dělali, když příkaz `cat` dokáže pracovat i interaktivně (proto můžeme vynechat zadání vstupu jako konzoly). Nicméně, je to možné.

Pokud si „nemůžeme zapamatovat“ ukončující sekvenci Ctrl+D, která představuje konec souboru, máme možnost použít tzv. *vložené soubory* (here documents). Je to (téměř) totéž, jen navíc stanovíme ukončující sekvenci.

Příklad 1.9

Vytvoříme nový soubor pomocí vloženého souboru, tedy stanovíme ukončující sekvenci, po jejímž zadání bude soubor uzavřen, příkaz ukončen, a tato sekvence se nestane součástí vytvořeného souboru. Zadáme:

```
cat > novysoubor.txt << KONEC
```

Dále píšeme na klávesnici to, co chceme, aby bylo v souboru:

```
Toto je první řádek,
a teď píšeme další řádek.
Tento řádek bude poslední.
KONEC
```

Po napsání ukončující sekvence („`KONEC`“) se příkaz ukončí a soubor je vytvořen.

1.4 Další příkazy

Podíváme se na některé další jednoduché příkazy:

`date`
: výpis data a času, parametry můžeme stanovit, co a v jaké formě bude vypsáno

`date +%d.%B` vypíše datum ve tvaru den.měsíc (měsíc bude vypsán slovně, podle jazykového nastavení systému)

`date +%x` vypíše datum ve tvaru měsíc/den/rok

`date +%T` vypíše čas ve tvaru hodiny:minuty:sekundy

`date >> log.dat` tento příkaz umístěný do `.login`, případně `.profile`, způsobí, že při každém přihlášení se do souboru `log.dat` přidá datum tohoto přihlášení, máme tedy přehled o tom, kdy se uživatel tohoto jména přihlašoval

`clear`
: smazání obrazovky

`exec příkaz`
: provede příkaz, ale těsně před tím ukončí rodičovský proces (pokud tento příkaz napíšeme v shellu, pak je ukončen shell a pak proveden příkaz); hlavním důvodem používání tohoto příkazu bylo dříve omezené množství operační paměti, ukončení rodičovského procesu způsobilo uvolnění paměti jím používané, dnes se používá spíše jako poslední spouštěcí příkaz ve skriptech

`echo [-e] [-n] řetězec`
: s tímto příkazem jsme se seznámili už v předchozím semestru, zde je pro připomenutí.

Příklad 1.10

Tento příkaz vytvoří následující tabulku:

```
echo -e "Nadpis1\tNadpis2\nObsah1\tObsah2"
```

```
Nadpis1    Nadpis2
Obsah1     Obsah2
```

Úkoly

1. Vyzkoušejte různé možnosti zobrazení data a času.
2. Zobrazte manuálovou stránku příkazu `echo` (v sekci 1). Zjistěte, jaké přepínače má tento příkaz a jaké formátovací řetězce lze používat mimo výše uvedených.

1.5 Speciální znaky pro uvozování

Různé typy uvozovek a dalších symbolů mohou měnit způsob zacházení s řetězcem.

Apostrofy určují, že vše, co je mezi nimi, tvoří jednolitý celek, i kdybychom mezitím klepli na klávesu Enter, znaky mezi apostrofy jsou obvykle brány jako obyčejný text. Jsou běžně používány například v těchto případech:

- v příkazu `echo` (nebo jiném) chceme text umístit do více řádků, pak na prvním řádku (který začíná příslušným příkazem) je jen levý apostrof, pravý je na posledním řádku, který chceme do vstupu zahrnout,
- názvy souborů obsahující mezery uzavíráme do apostrofů,
- `find / -name '*' -print` (zde zabraňujeme interpretaci hvězdičky; vyzkoušejte i variantu bez apostrofů, obdržíte chybové hlášení)
- atd., setkali jsme se s nimi u regulárních výrazů, kde určovaly hranice výrazu a zamezovaly předčasné interpretaci, setkáme se s nimi také například u proměnných.

Obrácené apostrofy naopak způsobují okamžitou interpretaci svého obsahu. Tedy obsah mezi obrácenými apostrofy je nahrazen svým výstupem. Takto lze vložit výstup jednoho příkazu do parametru jiného příkazu.

Příklad 1.11

Srovnejme (pozor, mezery kolem operátoru jsou povinné!!!):

- `expr 1 + 1`
- `echo Výpočet: 'expr 1 + 1'`
- `echo Výpočet: ` `expr 1 + 1` ` `

Výstup prvního příkazu je číslo 2 (tento příkaz vyhodnotí svůj argument jako aritmetický výraz). V druhém případě se pouze vypíše celý řetězec, nic nebude vypočteno. V třetím případě jsme do parametru příkazu `echo` vložili příkaz pro výpočet, a to v obrácených apostrofech, tedy nejdřív je proveden vnitřní příkaz, pak je jeho výsledek (číslo 2) vložen na místo tohoto příkazu a teprve potom je vypsán celý řetězec.

Zpětné lomítko se narozdíl od ostatních uvozovacích symbolů píše pouze před symbol, nikoliv za něj. Zpětná lomítka používáme k přepnutí významu následujícího prvku (tj. buď vypnutí nebo zapnutí, opačně vzhledem k původnímu nastavení).

S některými způsoby použití jsme se už setkali, například u regulárních výrazů nebo u escape sekvencí v příkazu `echo`. Zde je to například `\n`, tedy výsledek není chápán jako písmeno „n“, ale escape sekvence přechodu na nový řádek.

Zpětné lomítko také umístíme na konec řádku, pokud má příkaz (jeho parametry) pokračovat na následujícím řádku.

Příklad 1.12

```
echo první řádek\
>  druhý řádek \
>  třetí řádek
```

vypíše

```
první řádekdruhý řádek třetí řádek
```

(všimněte si chybějící mezery; chybí, protože jsme ji do původního příkazu před konec řádku nenapsali).

Uvozovky vypínají nahrazování jmen, ale přitom zachovávají význam obrácených apostrofů, proměnných a znaků uvozených zpětným lomítkem. Uvozovky využijeme také jako náhradu běžných apostrofů, pokud je chceme vnořit (běžné apostrofy nelze vnořovat, proto místo „vnějších apostrofů“ použijeme uvozovky).

Úkoly

Využijte obrácené apostrofy k výpisu informací o povolených shellech, vyzkoušejte:

```
cat /etc/shells
ls -la `cat /etc/shells`
```

Ve výpisu také zjistíte případné symbolické odkazy, které jedním názvem shellu odkazují na jiný shell.

1.6 Proměnné

1.6.1 Základy práce s proměnnými

Proměnné se definují v souboru `.bashrc` nebo jsou předdefinovány systémem či v jiném konfiguračním souboru. Nejdůležitější jsou:

`HOME` domovský adresář uživatele

`TERM` typ terminálu

`SHELL` cesta k používanému shellu

`USER` přihlašovací jméno uživatele

PATH seznam adresářů, ve kterých se hledá spouštěný soubor, jednotlivé cesty jsou odděleny dvojtečkou

PS1 prompt, výzva příkazového řádku konzoly nebo terminálu

PWD pracovní adresář

V proměnné `PATH` není ve výchozím nastavení cesta k pracovnímu adresáři a při spuštění programu také narozdíl od Windows není spouštěný program hledán v pracovním adresáři! Proto pokud chceme spustit například program `mujprogram` umístěný v adresáři, který je právě naším pracovním adresářem, provedeme to takto:

```
./mujprogram
```

Teoreticky by se tento „problém“ dal vyřešit přidáním adresáře „`.`“ do proměnné `PATH`, ale toto řešení se z bezpečnostních důvodů nedoporučuje.[3]

Existuje také obdoba rozdělení proměnných na běžné a dynamické, s čímž jsme se setkali už ve Windows. Například proměnná `PS1` je dynamická.

Proměnné jsou buď *lokální* (platné pouze v rámci skriptu, příkazového shellu nebo bloku uvnitř skriptu), a nebo *proměnné prostředí*, které jsou viditelné i mimo oblast, ve které byly deklarovány (pozor, pojem „prostředí“ má v Linuxu opačný význam než ve Windows).

Příkazy pro práci s proměnnými:

`echo $proměnná`
: vypíše obsah proměnné (vyhodnotí ji), pracuje s běžnými i dynamickými proměnnými

`proměnná=výraz`
: změní obsah proměnné

`export proměnná`
: exportuje proměnnou do prostředí (aby ji mohly využívat všechny skripty i příkazový režim), dá se spojit s přiřazením hodnoty do proměnné

`env`
: vypíše proměnné s jejich obsahem (proměnné prostředí)

`set`
: vypíše veškeré proměnné a funkce, které jsou definovány v dané oblasti, ve které pracujeme (výstup je poměrně rozsáhlý)

`unset`
: odstraní proměnnou

`read proměnná`
: načte ze standardního vstupu řetězec do proměnné

[3] Představte si, že ve vašem domovském adresáři (který velmi často bývá pracovním adresářem) bude podstrčen škodlivý software pojmenovaný stejně jako některý z běžně používaných příkazů, nejlépe příkazu obvykle se nacházejícího v adresáři `/sbin`, který často nebývá zahrnut v proměnné `PATH`. Pokud zařadíme do proměnné `PATH` tečku, můžeme tento škodlivý software omylem spustit (a pokud je jeho programátor dost chytrý na to, aby po dokončení své vlastní činnosti spustil původní program na správném umístění, tak to ani nezjistíme). Mohlo by se zdát, že stačí tečku přidat až na konec proměnné `PATH`, aby všechny dříve uvedené adresáře měly přednost, ale toto nefunguje, pokud některé jinak důležité cesty nejsou v této proměnné zahrnuty.

Znak $ vyhodnotí vše, co se za ním nachází (vše až po první mezeru nebo konec řádku považuje za název zpracovávané proměnné), proto když zadáváme za názvem proměnné ještě něco dalšího, uzavřeme tuto proměnnou do lomených závorek:

```
export PATH=${PATH}:/usr/TeX/bin
```

Když chceme, aby se takováto „vnitřní proměnná“ vyhodnocovala při každém volání „vnější proměnné“, uzavřeme výraz do jednoduchých uvozovek (apostrofů).

Příklad 1.13

Předpokládejme, že někde třeba ve skriptu používáme proměnnou. Nejdřív ji vytvoříme a inicializujeme, pak vypíšeme její obsah.

```
prom="A B   C"
echo $prom
echo "$prom"
echo \$prom
```

Druhý, třetí a čtvrtý příkaz nám postupně dají tyto výstupy (všimněte si důsledku použití uvozovek):

```
A B C
A B   C
$prom
```

Příklad 1.14

Do proměnné nemusíme ukládat jen řetězec nebo číslo. Jak víme, obrácené apostrofy jsou prostředkem, jak vynutit vyhodnocení výrazu. Například:

```
prom="Výstup: ; `ls -la`"
echo $prom
echo "$prom"
```

V prvním příkazu jsme do proměnné uložili příkaz (vlastně dva příkazy oddělené středníkem, aby mohly být na jednom řádku) k interpretaci. Obrácené apostrofy uvnitř uvozovek jsou zde nutné.

V dalších příkazech vypisujeme obsah proměnné, čímž vyvoláme interpretaci příkazů v ní uložených. Vyzkoušením si můžeme ověřit význam uvozovek u posledního příkazu.

Příklad 1.15

Další ukázky s proměnnými:

`export PS1='$PWD'` do proměnné `PS1`, tj. do momentálního promptu, si uložíme pracovní adresář

`PS1="\W"` pracovní adresář bez cesty, s cestou je `\w` (a také zobrazuje domovský adresář jako vlnovku)

`PS1="\d"` promptem bude aktuální datum

`PS1="\A"` promptem je aktuální čas – jen hodiny a minuty, jiné možnosti pro čas jsou `\t` nebo `\T`

`PS1="\u\h\$ "` prompt je ve formě uživatel počítač`$` (plus mezera)

`PS1="\u\h:\w> "` prompt je ve formě uživatel počítač: pracovní adresář> (plus mezera)

`PS1='pracuji tak dlouho: $SECONDS'` opět změníme prompt, proměnná `SECONDS` je dynamická

`mkdir $HOME/novy` využili jsme proměnnou uvnitř parametru příkazu; všimněte si složených závorek kolem názvu proměnné, zde jsou nutné

Kromě proměnné `PS1` existují obvykle ještě další proměnné určující prompt. Většinou se setkáme alespoň se sekundárním promptem `PS2`, který je zobrazován například při psaní víceřádkových příkazů, případně při vytváření here documents.

Pokud chceme, aby námi vytvořená proměnná byla viditelná i mimo pracovní prostředí, ve kterém je vytvořena (například i mimo skript, nejrůznějším dalším programům apod., musíme ji *exportovat*:

```
export prom
```

Příkaz `set` je ve skutečnosti mnohem silnější, nejde jen o vypisování seznamu proměnných a funkcí. Můžeme například do proměnné přiřadit výsledek funkce (podrobnosti najdeme v manuálové stránce příkazu `set`).

Úkoly

1. Vyzkoušejte příkazy vypisující proměnné. Vyberte si kteroukoliv proměnnou a vypište její obsah.
2. Vyzkoušejte příkazy z příkladu 1.13. Dále podle příkladu 1.14 vyzkoušejte přiřazení příkazu do proměnné. Také se pokuste změnit si prompt podle dalšího příkladu.

1.6.2 Výpočty

Používání proměnných úzce souvisí s prováděním výpočtů. Pro tyto účely existují především dva základní příkazy:

`let "výraz"`
změna hodnot proměnných s vyhodnocením výrazu, v některých případech jsou uvozovky nutné

`let 3 + 4` vypočte a vypíše výsledek

`let "prom=3+4"` do proměnné je uložen výsledek výrazu

`let "prom+=4"` je podporována „céčkovská“ notace přiřazování

`let "prom+=$prom"` na pravé straně může být také proměnná, ale samozřejmě příslušně označená symbolem dolaru, aby byla interpretována

`expr výraz`

další způsob, jak vyhodnotit výraz a případně v něm použít proměnnou, či do proměnné uložit výsledek, kolem operátorů jsou povinně mezery

`prom=‘expr 3 + 4‘` nejdřív vypočte výraz, pak výsledek přiřadí do proměnné (kolem rovnítka nesmí být mezery)

`expr $prom + 1` vypíše číslo o 1 větší než je hodnota proměnné `$prom` (tj. dosadí za proměnnou, vypočte výraz a vypíše)

Výsledek si můžeme vždy ověřit vypsáním `echo $prom`.

Příklad 1.16

Hodnotu proměnné lze načíst i od uživatele (proměnná, do které načteme vstup od uživatele, nemusí předem existovat, lze ji vytvořit i voláním funkce `read`):

```
echo -n "Zadej číslo: "
read prom1
echo -n "Byla načtena hodnota ${prom1}"
prom2=$prom1
```

Všimněte si závorek kolem názvu proměnné. Zde nejsou přímo nutné, ale je dobré si na ně zvyknout, protože v určitých případech jsou naopak nevyhnutelné. Celý vypisovaný řetězec také nemusí být v uvozovkách, v tomto případě.

```
prom1+=80
let "prom2+=80"
echo prom1=${prom1}, prom2=${prom2}.
```

Úkoly

1. Proveďte příkazy z příkladu 1.16. Zjistěte, jaký je rozdíl ve výstupech příkazů, kde se k proměnné přičítá číslo 80. Ve kterém případě se s proměnnou zachází vždy jako s řetězcovou?
2. Zobrazte manuálovou stránku příkazu `expr` a zjistěte, jaké výrazy jím lze interpretovat.
3. Vyzkoušejte výpočty, kde je na pravé straně výrazu proměnná. Načtěte od uživatele hodnotu proměnné a vypište její dvojnásobek.

Kapitola 2

Skripty a programování

V této kapitole jsou zahrnuta pokročilejší témata, je vlastně rozšířením předchozí kapitoly. Podíváme se na konfigurační soubory, potom na další úlohy týkající se práce v shellu `bash` *(skripty, podmínky, cykly, pole, apod.).*

2.1 Konfigurační a další systémové soubory

Většina konfiguračních souborů není psána pro určitý shell, ale má speciální formát (samozřejmě textový) srozumitelný především tomu programu, který má konfigurovat. Některé z nich jsou skryté (začínají tečkou).

Příklady konfiguračních souborů:

`/etc/fstab`
: seznam souborových systémů, které se připojují při startu systému nebo mohou být připojeny

`/etc/mtab`
: seznam momentálně připojených (mounted) souborových systémů, je dynamicky generován systémem podle obsahu jednoho ze souborů v adresáři `/proc`

`/etc/inittab`
: konfigurace programu *init*

`/etc/securetty`
: seznam terminálů, ze kterých je dovoleno přihlásit se na účet *root*

`/etc/passwd`
: seznam uživatelů

`/etc/group`
: seznam skupin

Do souboru `/var/log/messages` se (ve výchozím nastavení, které může být změněno) ukládají zprávy o běhu systému (co se děje, co se spouští apod.), včetně samotného startu systému. Pokud chceme zjistit, zda došlo k nějaké chybě (a případně i nějaké další informace), můžeme nahlédnout

do tohoto souboru. Přímý přístup k němu má však pouze root, a navíc tento soubor je hodně rozsáhlý (dokonce už jen při startu se toho děje strašně hodně), takže musíme zvolit vhodný způsob přístupu k tomuto souboru a navíc použít vhodný filtr.

Root použije následující příkaz:
`cat /var/log/messages | grep klíčové_slovo`,
běžný uživatel napíše příkaz
`dmesg | grep klíčové_slovo` (program `dmesg` slouží k získání informací o startu systému, také čte ze souboru `/var/log/messages`). Běžný uživatel může takto získat pouze informace uložené do `/var/log/messages` během startu systému, root kterékoliv informace přidané kdykoliv při běhu systému.

Když se přihlašujeme, postupně se načítají tyto konfigurační soubory:

- `/etc/profile` – toto je obecný skript platný pro všechny profily
- `~/.bash_profile`, `~/.bash_login`, `~/.profile` – bere v úvahu vždy jen jeden z těchto tří souborů (vždy ten první podle tohoto pořadí, na který narazí), obsahuje konkrétní profil platný pro daného uživatele
- `~/.bashrc` – tento skript se spustí, pokud jsme se nepřihlašovali v textovém režimu, ale chceme pracovat třeba v konzole

Při odhlašování se pak provádí kód v souboru `~/.bash_logout`.

2.2 Skripty

2.2.1 Co je to skript

Skripty (skriptové soubory) jsou textové spustitelné soubory, které pro své spuštění potřebují interpret (to je některý shell). Soubory se skripty obvykle mívají buď příponu `.sh`, a nebo jsou bez přípony.

Každý skript je psán vždy pro určitý shell nebo programovací jazyk (například bash nebo perl) a v jiném nemusí fungovat. Proto často bývá na prvním řádku skriptu identifikace shellu. Tento řádek vždy začíná dvojicí znaků `#!`, pokud ovšem se ve skriptu nachází. Například:

- `#!/usr/bin/tcsh` znamená, že skript má být interpretován shellem Toronto C Shell,
- `#!/usr/bin/bash` určuje skript s příkazy shellu Bourne Again Shell,
- `#!/usr/bin/perl` (adresa může být jiná, záleží na umístění spustitelného programu `perl`) je skript psaný v programovacím jazyce Perl.

Aby bylo možné skript spouštět v textovém režimu napsáním jeho názvu, musí být označen jako spustitelný, tj. je třeba nastavit příznak `x` v přístupových oprávněních. V opačném případě musíme skript spouštět jako parametr interpretačního programu (například `bash soubor.sh`).

Skript pro shell `bash` vypadá například takto:

```
#!/bin/bash
echo Hello World
echo znamena Ahoj svete
```

Skripty můžeme použít pro automatické zálohování, shromažďování informací, přidávání nových uživatelů a obecně jakoukoliv automatizaci posloupnosti operací.

2.2.2 Parametry a návratové hodnoty

Parametry skriptu jsou přístupné přes proměnné `$0` (název skriptu), `$1` (první parametr), `$2` (druhý parametr), . . . , `$9` (i vyšší), jejich obsahy lze posouvat a tak se dostat i k dalším parametrům pomocí příkazu `shift` (jako nepovinný parametr můžeme použít číslo označující počet přesunů, implicitně je to 1).

Skript může také vracet návratový kód, a to příkazem `exit`. Můžeme jako nepovinný argument zadat číslo návratového kódu, výchozí je číslo 0. Návratová hodnota nemusí být konstantní číslo, můžeme takto předat třeba i obsah proměnné.

`exit` konec skriptu, vracíme hodnotu 0

`exit 8` konec skriptu, vracíme hodnotu 8

`exit $prom` konec skriptu, vracíme hodnotu obsaženou v zadané proměnné

Obdobou proměnné `errorlevel` z Windows je v Unixu proměnná `$?`. Obsahuje status (stav) posledního spouštěného příkazu, což samozřejmě může být i skript.

Běžně používané návratové hodnoty (chybové kódy) jsou 0 (proces proběhl úspěšně), 1 (proces se spustil, ale proběhl neúspěšně – nastaly chyby při běhu), 2 (proces se nespustil), 127 (proces nebylo možno spustit, protože program nebyl nalezen).

Příklad 2.1

Trochu předběhneme a použijeme cyklus until pro procházení všemi parametry, které uživatel při spuštění našeho skriptu použil.

Soubor se skriptem obsahuje tento kód:

```
until [ -z "$1" ]; do  # "z" jako "zero" - provádí se tak dlouho, dokud
                       # nebude splněna podmínka "první parametr je prázdný"
  echo -n "$1 "        # vypíšeme první parametr
  shift                # posun, do prvního parametru se dostane obsah druhého apod.
done                   # konec cyklu
echo                   # zařádkování na konec (prázdný řádek)
```

Předpokládejme, že tento náš skript je uložen v souboru s názvem `pokus.sh` a uživatel ho spustil s těmito parametry:

```
./pokus.sh kalendar auto 54 posledni
```

Pak výstupem bude následující:

```
kalendar auto 54 posledni
```

V cyklu sice vypisujeme pořád první parametr (`$1`), ale díky příkazu posunu se do tohoto parametru postupně přesouvají všechny následující. Cyklus končí tehdy, když je v prvním parametru prázdný řetězec, tj. další parametr již není zadán.

Mohli bychom použít také cyklus `for` (viz dále) – počet použitých (plných) parametrů je totiž uložen v proměnné `$#`.

2.2.3 Další možnosti použití skriptů

Na prvním řádku skriptu bývá uveden příkaz spouštějící shell, který má provést interpretaci následujících řádků. Ve skutečnosti tam může být téměř kterýkoliv příkaz, kterému je pak přesměrován zbývající obsah souboru jako vstup.

Nahrazení shellu následujícím skriptem způsobí, že pokud se uživatel, pro kterého byla změna provedena, pokusí přihlásit do systému, vypíše se mu uvedená hláška a uživateli bude odmítnut přístup.

```
#!/usr/bin/tail +2
Pozor, tvůj účet byl zablokován z důvodu ...
Pokud chceš vše napravit, ...
```

Samotné nahrazení shellu se provede příkazem

```
chsh -s /.../soubor_skriptu uživatel
```

2.3 Podmínky, větvení a cykly

2.3.1 Jednoduché propojení příkazů

V Unixu se setkáváme s podobnými možnostmi propojení příkazů, jaké známe z Windows (ostatně, odkud se to všechno ve Windows asi vzalo). Možnosti:

`příkaz1 ; příkaz2 ; příkaz3` sekvenční provádění příkazů, totéž, jako kdyby byly zvlášť na samostatných řádcích

`příkaz1 | příkaz2 | příkaz3` řetězení vstupů/výstupů, tedy roury (kolony) (pozor, to není plně sekvenční provádění, následující příkaz může začít zpracovávat vstup ještě před ukončením předchozího příkazu, typicky stránkovací filtry)

`příkaz1 || příkaz2 || příkaz3` logické OR, následující příkaz se provede pouze tehdy, když předchozí skončil neúspěchem (tj. vrátil nenulový návratový kód)

`příkaz1 && příkaz2 && příkaz3` logické AND, následující příkaz se provede pouze tehdy, když předchozí skončil úspěchem

Výše uvedené možnosti lze také kombinovat a pracuje se s nimi stejně jako ve Windows, proto je dále nebudeme rozvádět.

Příklad 2.2

```
pgrep nejaky_proces >/dev/null && echo proces běží || echo proces neběží
```

Pro sdružování několika příkazů spojených těmito symboly můžeme použít složené závorky. Navíc se takto dá řešit problém poslání na pozadí celé sekvence příkazů.

Příklad 2.3

```
sleep 30; echo "Konec prace!"
```
30 sekund se nic neděje (ani prompt), pak se vypíše hláška

```
sleep 30; echo "Konec prace!" &
```
30 sekund se nic neděje, pak se na pozadí provede příkaz

```
{ sleep 30; echo "Konec prace!" }&
```
30 sekund můžeme pracovat (je zobrazen prompt), pak se objeví hláška

Symbol `&` samotný na konci názvu příkazu je něco trochu jiného. Znamená asynchronní provádění příkazu, tedy příkaz následovaný tímto symbolem je spuštěn na pozadí a můžeme v shellu dále běžně pracovat. S touto možností se blíže seznámíme ve správě procesů a úloh.

2.3.2 Příkazy pro podmínky a cykly

Předně je třeba si uvědomit jednu důležitou věc – složené příkazy jsou doopravdy složené z více příkazů, a tedy buď jejich části (ty, které se vyhodnocují sekvenčně) oddělujeme středníkem, a nebo pokračujeme na dalším řádku (víceřádkový příkaz).

Podmínka if je další běžnou programátorskou konstrukcí v shellu `bash`. Příkaz větvení `if` má tuto syntaxi:

```
if [ podmínka ]; then
  příkazy oddělené středníkem
  nebo na nových řádcích
  jakýkoliv počet příkazů
fi
```

nebo

```
if [ podmínka ]; then
  příkazy
else
  příkazy
fi
```

```
if [ podmínka ]
  then
  příkazy
  příkaz ; příkaz
fi
```

nebo

```
if [ podmínka ]; then příkaz ; fi
```

Poznámka: Mezery uvnitř hranatých závorek jsou *naprosto nutné*, protože ve skutečnosti je závorka `[` příkazem a vše za ní až po středník za symbolem `]` jsou parametry tohoto příkazu. Místo tohoto příkazu můžeme ve skutečnosti použít jakýkoliv jiný příkaz nebo kolonu (rouru) příkazů, vyhodnocována bude návratová hodnota.

Formu podmínky probereme po ukázce syntaxe cyklů.

Příkazy cyklů: Cykly můžeme provádět pomocí tří druhů příkazů: for, while a until. Opět je třeba dbát na správné oddělování částí složeného příkazu.

Příkaz `for` má syntaxi:

```
for i in $( seznam řetězců ); do
  příkazy
  echo přístup k proměnné i je $i
  echo stejně jako u systémových proměnných
done
```

Můžeme také používat syntaxi převzatou z `csh`:

```
for i in 'seznam řetězců';
do
  příkazy
done
```

Příkazy `while` a `until` mají tuto syntaxi:

```
while [ podmínka ]; do
  příkazy
done
```

```
until [ podmínka ]; do
  příkazy
done
```

Opět je nutné přidávat mezery u hranatých závorek.

Podmínky v příkazech `if`, `while` a `until`: V podmínce se používají řetězce obvykle představující proměnné a dále operátory reprezentované těmito přepínači:

Pro řetězce:

`řetězec`	= true, pokud je řetězec neprázdný
`-z řetězec`	= true, pokud je řetězec prázdný
`řetězec1 = řetězec2`	= true, pokud jsou řetězce stejné
`řetězec1 != řetězec2`	= true, pokud jsou řetězce různé

Pro názvy souborů:

`-f soubor`	= true, pokud soubor existuje a je běžný soubor
`-d soubor`	= true, pokud soubor existuje a je to adresář
`-r soubor`	= true, pokud soubor existuje a je nastaveno právo čtení
`-w soubor`	= true, pokud soubor existuje a je nastaveno právo zápisu
`-x soubor`	= true, pokud soubor existuje a je nastaveno právo spouštění
`-s soubor`	= true, pokud soubor existuje a má nenulovou délku

Pro čísla a proměnné s čísly:

`číslo1 -eq číslo2`	= true, pokud číslo1 = číslo2
`číslo1 -ne číslo2`	= true, pokud číslo1 <> číslo2
`číslo1 -gt číslo2`	= true, pokud číslo1 > číslo2
`číslo1 -lt číslo2`	= true, pokud číslo1 < číslo2
`číslo1 -ge číslo2`	= true, pokud číslo1 ≥ číslo2
`číslo1 -le číslo2`	= true, pokud číslo1 ≤ číslo2

Skládání výrazů:

`! výraz`	negace výrazu
`výraz1 -a výraz2`	logické AND
`výraz1 -o výraz2`	logické OR
`\( výraz \).`	uzávorkování výrazu

Pokud existuje třeba jen malá pravděpodobnost, že v proměnné použité ve výrazu by mohl být prázdný řetězec, musí být název proměnné (i se znakem $) uzavřen do uvozovek.

Pokud v příkazu `if` chceme nechat první větev prázdnou a psát příkazy až do větve `else`, do první větve dáme alespoň prázdný příkaz (to je „:“, dvojtečka).

V sekvencích příkazů (včetně podmínek příkazů) můžeme používat také symboly pro spojování příkazů, tj. `;`, `&&`, `||`. V příkazech cyklů můžeme pro přerušení cyklu použít příkaz `break` nebo `continue`.

Příklad 2.4

Podíváme se na několik jednoduchých skriptů využívajících buď rozhodování (příkaz `if`) nebo některou formu cyklu.

1.
```
#!/bin/bash
if [ "$PWD" = "/" ]; then
  echo Pracovní adresář je root, tam mě nikdo nedostane!
  exit 1
fi
```

2.
```
#!/bin/bash
#přepneme se do adresáře docasny v domovském adresáři
cd ~/docasny

#vytvoříme proměnnou (neexportujeme, použijeme ji jen zde)
souhlas="N"

pwd
for i in $( ls -a ); do
  echo Smazat $i .... (A/N)?
  read souhlas
  if [ "$souhlas" = "A" || "$souhlas" = "a" ]; then
    #rekurzívní mazání souborů a adresářů:
    rm -fr $i
    if [ $? -eq 0 ]; then
      echo OK, soubor smazán
    else
      echo CHYBA při mazání souboru!
    fi
  else
    echo OK, soubor zůstane
  fi
done
```

3.
```
#!/bin/bash
for c in $( 1 2 3 ); do
  echo c = $c
```

```
done
for i in $("$HOME" "$SHELL" "$PS1" ); do
  echo $i je `$i`
done
```

4.
```
#!/bin/bash
klavesa="N"
until [ $klavesa != "N" ]; do
  echo Tak už stiskni nějakou klávesu!
  echo (krome klavesy N)
done
```

5.
```
#!/bin/bash
CITAC=1
POSLEDNI=5
vysledek=1
while [ $CITAC -lt $POSLEDNI ]; do
  let vysledek*=2
  citac+=1
done
echo 2^${POSLEDNI} = $vysledek
```

Příklad 2.5

Množina, přes kterou jde proměnná v cyklu `for`, může být vygenerována příkazem. Nesmíme však zapomenout na to, že dotyčný příkaz generující množinu musíme uzavřít do zpětných apostrofů, aby byl vyhodnocen ještě před vyhodnocením samotného cyklu `for`. Přímo v textovém shellu zadáme:

```
for i in `find . -name '*.txt'` ; do cp $i ${i%txt}bak ; done
```

Do proměnné `$i` postupně přiřazujeme všechny názvy souborů s příponou TXT, a to rekurzívně, kopírujeme je na soubory do stejného adresáře, ale s jinou příponou (BAK). Všimněte si, jakým způsobem bylo provedeno „odstřihnutí" původní přípony a zřetězení s novou příponou. U příkazu `for` jsme použili jednodušší syntaxi ze shellu `csh`, ta samozřejmě funguje i v `bash`.

Častou chybou, kterou zejména začátečníci těžko odhalují, je nesprávné umístění středníku – musí být před klíčovým slovem `do`.

Příklad 2.6

Ve skriptu chceme do proměnné uložit abecedně seřazený seznam uživatelů oddělených mezerou (abychom si procvičili řetězení obsahu proměnných). Pro změnu použijeme novější `bash` syntaxi.

```
SOUBOR=/etc/passwd
uzivatele=""
for i in $( cut -d ":" -f 1 $SOUBOR | sort ) ; do
  uzivatele="$uzivatele $i"
done
echo $uzivatele
```

Jiný je zápis podmínky (nepoužili jsme zpětné apostrofy, ale dolarovku se závorkami, pozor na mezery). „Rozřezali“ jsme řádek souboru podle oddělovačů „dvojtečka“ a vybrali jsme první sloupec. Vzpomeňte si, ve Windows jsme pro tento účel využili příkaz `for` s přepínačem `/F`.

2.3.3 Jednoduché testování

Příkaz `test` provádí vyhodnocení svého argumentu, obvykle vrací hodnotu true nebo false. Umožňuje zjistit, zda jsou dva řetězce (třeba obsahy proměnných) stejné nebo jeden menší (větší apod.) než druhý (především pro proměnné obsahující čísla), zda zadaný soubor existuje, zda je proveditelný (tj. program), atd. Názvy proměnných se dávají do uvozovek.

Syntaxe používaná v podmínkách podmíněného příkazu a smyček je vlastně alias pro příkaz `test`, tedy `[ řetězce ];` je totéž jako `test řetězce`. Příkaz `test` samozřejmě nemusíme používat jen ve skriptech, můžeme využívat možnost provádět jednoduché výpočty a testování přímo v příkazovém režimu.

Příklad 2.7

Ukážeme si použití jednoduchého testování na zjištění existence adresáře. Před přesunem do zadaného adresáře si můžeme takto otestovat, zda existuje:

```
if test -d adresar
> then
>   cd adresar
> else
>   ztratil(-a) jsem se!!!
> fi
```

Parametr `-d` slouží k testování existence adresáře, jiné parametry zase slouží k testování existence jakéhokoliv souboru, blokového či znakového speciálního souboru, souboru s nastaveným SUID bitem, je možné testovat také přístupová oprávnění, zda je adresář prázdný, atd.

2.3.4 Pole

Proměnné nemusí být jen řetězce nebo celá čísla, mohou to být také pole položek. S poli pracujeme podobně jako s jinými proměnnými a syntaxe částečně vychází z jazyka C, jen narozdíl od C nemusíme brát ohled na omezení daná datovými typy a pevnou či dynamickou alokací. Indexace je od 0.

Příklad 2.8

Několik ukázek práce s poli:

1. 

```
#!/bin/bash
#deklarace pole:
nazvyznamek=( Jednicky Dvojky Trojky Ctyrky Petky )
znamky=( 8 5 1 2 0 )

#pozor na slozene zavorky:
echo Pocet jednicek: ${znamky[0]}
echo Vsechny znamky: ${znamky[*]}

echo Tabulka znamek:
echo Znamka /tPocet

#parametr prikazu for je vlastne taky pole:
for z in $( 0 1 2 3 4 ); do
  echo ${nazvyznamek[$z]}: /t${znamky[$z]}
done

#predefinovani pole:
znamky=( 3 4 5 1 )
#paty prvek (znamky[4]) se prepsal na prazdny retezec!
```

2. 

```
#!/bin/bash
kapacitaletadel=( 21 3 15 )

#prikoupime letadlo, jehoz kapacita je 52:
kapacitaletadel=( ${kapacitaletadel[*]} 52 )

#do prvniho letadla jsme prikoupili jedno sedadlo:
kapacitaletadel[0]=”22”
```

Některé systémové proměnné jsou také typu pole, například `BASH_VERSINFO` je pole údajů o používané verzi shellu `bash`.

2.4 Překlad programů

V shellu můžeme spouštět i aplikace grafického režimu, existují zde i textové programy pro úpravu textů (např. nano, pico, vi, vim, emacs, kwrite, kedit), kompilátory jazyka C (gcc, c++, cc, . . .) apod.

Pro překlad programů v jazyce C (nebo také C++) se obvykle používá překladač `gcc` (GNU C Compiler). Jedná se opravdu o překladač, nikoliv editor (jako editor můžeme použít některý z běžných textových editorů, ale existují i speciálně určené pro programování, například Kate nebo prostředí KDevelop).

Program `gcc` se v jednodušším případě používá takto:

`gcc -o vystupni_soubor zdrojovy.c` přeloží zadaný zdrojový soubor, také je stanoven výstupní soubor

Program má velmi mnoho nejrůznějších voleb, často určených pro překlad konkrétního jazyka, některé volby jsou hardwarově závislé (pro konkrétní hardwarovou platformu). Existují také například

volby pro překlad vícevláknových aplikací.

Úkoly

Vytvořte textový soubor `zdroj.c` s následujícím obsahem (například v editoru kwrite nebo gedit, nebo přímo v shellu příkazem `cat > soubor.c << KONEC`):

```
#include<stdio.h>
int main(void){
 printf("Hello World\n");
}
```

Uložte do vašeho domovského adresáře a pak přeložte:

```
gcc -o vystupnisoubor zdroj.c
```

Program spusťte. Nezapomeňte, že váš domovský adresář zřejmě není v proměnné obsahující cesty ke spustitelným souborům, tedy je nutné zadat cestu do pracovního adresáře:

```
./vystupnisoubor
```

2.5 Další nástroje

2.5.1 Aliasy

Alias je zástupný název pro nějaký řetězec, obvykle příkaz nebo část příkazu. Pro práci s aliasy existuje příkaz **`alias [zastupny='retezec']`**, při použití bez parametrů vypíše všechny existující aliasy.

`alias` vypíše nadefinované aliasy

`alias md='mkdir'` vytvoří zkratku pro příkaz `mkdir`

`alias ls='ls -a'` od této chvíle se při zadání příkazu `ls` (případně i s dalšími parametry) budou vypisovat všechny položky včetně skrytých

`alias seznam='cat -n'` vytvoříme vlastní příkaz (vlastně ani ne tak příkaz jako obdobu makra), který vypíše zadaný soubor a zároveň očísluje řádky. Používá se takto:

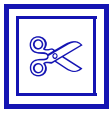
```
seznam soubor.txt
```

`alias odstrel='kill -9'` vytvoříme příkaz, který ukončí zadaný proces, i kdyby byl například zamrzlý, použití pro proces s PID 2510:

```
odstrel 2510
```

`alias textove="ls -la | grep '.txt'"` tento příkaz vypisuje všechny textové soubory v pracovním adresáři, včetně skrytých

`alias c+='mount /mnt/sda1'` vytvořený příkaz `c+` připojí první oddíl prvního pevného disku



Aliasy platí vždy jen do ukončení systému, takže pokud chceme některý zástupný řetězec používat trvale, musíme ho zapsat do některého skriptu, který se spouští hned po startu počítače, přihlášení nebo startu shellu (např. `~/.bashrc`).

2.5.2 Konverze textových souborů

`iconv` je program pro konverzi textových souborů mezi různými kódováními (filtr), například

```
iconv -f cp1250 -t iso8859-2 < souborwin.txt > souboriso.txt
```

zkonvertuje soubor s Windows kódováním cp1250 na ISO8859-2,

```
iconv -f iso8859-2 -t utf-8 < souboriso.txt > souborutf.txt
```

zkonvertuje soubor v kódování ISO8859-2 na unicode UTF-8.
Seznam všech podporovaných kódování se zobrazí `iconv -list`.

Programem s rozsáhlejšími funkcemi je například `recode` (dokáže kromě jiného také odstranit diakritiku ze souboru), dále `enca` (dokáže také detekovat znakovou sadu). Program `cstocs` je zase konvertor specializovaný na češtinu a slovenštinu:

```
cstocs cp1250 utf-8 < soubor.txt > vysledek.txt
```

Program `convmv` dokáže opravit názvy souborů, které nezodpovědný uživatel Windows pojmenoval s použitím háčků a čárek a odeslal jinému uživateli:

```
convmv -f cp1250 -t utf8 soubor
```

Kapitola 3

Úlohy při správě

V této kapitole se setkáme s úlohami při správě operačního systému, především ve správě uživatelů, procesů, zařízení, sítě a dalších. Podíváme se také na práci s moduly jádra.

3.1 Uživatelé a skupiny

3.1.1 Informace o uživatelích

Uživatelé mohou v souvislosti se svými účty používat tyto příkazy:

users
: vypíše seznam přihlášených uživatelů (v seznamu bude alespoň jeden název – účet, pod kterým jsme přihlášeni)

who
: výpis aktivních uživatelů (právě přihlášených podobně jako u předchozího příkazu), a to včetně různých dalších informací

  `who` (bez parametrů) vypíše momentálně přihlášené uživatele a základní informace o nich (například kdy se uživatel přihlašoval)

  `who -aH` zobrazí se seznam všech logování od startu systému (když použijeme parametr `-a` nebo `--all`), nad sloupci se zobrazí záhlaví

w
: takto zjistíme, kdo je přihlášen a co dělá, pracuje se souborem `/var/run/utmp` (dočasný soubor s přihlášenými uživateli), jako parametr můžeme (nemusíme) zadat přihlašovací jméno uživatele, jehož činnost nás zajímá

whoami
: informace o přihlášeném uživateli (přihlašovací jméno)

id
: vypíše identitu uživatele

`id` z výpisu zjistíme své UID, GID, název uživatele a skupiny, a dále seznam skupin, do kterých patříme; výstup může vypadat takto:

```
uid=1002(uzivatel) gid=100(users)
    groups=16(dialout),100(users),182(pskupina)
```

`id novak` vypíšeme podobné informace o zadaném uživateli

`id -u novak` vypíše se pouze UID uživatele *novak* (pomocí parametrů omezujeme množství vypsaných informací)

`finger`

výpis aktivních uživatelů včetně plného jména, kde a kdy se přihlásil apod.

`finger` vypíše požadované informace ve sloupcích

`finger -l` vypíše informace v „long“ – dlouhém formátu, jako seznam

`finger novak` vypíše informace o zadaném uživateli

`finger novak@uctarna` příkaz lze používat i vzdáleně (zadáváme uživatele a počítač, na kterém má být přihlášen)

V souvislosti s uživateli a skupinami nás zajímají některé soubory. O domovských adresářích víme (domovské adresáře běžných uživatelů jsou v adresáři `/home`, domovský adresář roota je `/root`, systémoví uživatelé (většinou procesy jako je například *ftp* nebo *apache*) žádný domovský adresář nemají.

Dále jsou důležité soubory:

- `/etc/passwd` – seznam uživatelů, kteří se mohou do systému přihlásit, formát řádku (záznamu) je

  `login:heslo:UID:GID:plné jméno:homedir:shell`

  pokud na místě hesla (druhá položka na řádku) je symbol „`x`“, znamená to, že heslo je v souboru `/etc/shadow`, pokud je zde uveden řetězec na místě hesla, je šifrovaný
- `/etc/group` – seznam definovaných skupin, formát řádku (záznamu) je

  `login:heslo:GID:vlastník`

  pokud je místo hesla symbol „`!`“, znamená to, že heslo není definováno (což je u většiny skupin), pokud je tam „`x`“, je heslo v souboru `/etc/shadow`
- `/etc/shadow` – šifrovaný soubor obsahující hesla uživatelů a skupin
- `/etc/skel` – adresář používaný jako vzor („kostra“) při vytváření domovských adresářů pro nové uživatele
- `/etc/login.defs` – soubor s výchozími nastaveními bezpečnostního charakteru, například je zde možné nastavit metodu šifrování hesel (DES, MD5, SHA256, SHA5120), jak často si uživatel musí měnit heslo, logování souvisejících informací, minimální hodnota UID pro nově vytvářené běžné uživatele, chování systému při neúspěšném pokusu o přihlášení, hodnota proměnné `PATH` pro běžné uživatele a pro roota, atd.

V příkazu pro vytvoření nového uživatelského účtu můžeme specifikovat vlastní náhradu souboru `/etc/shadow` nebo adresáře `/etc/skel`, to se může hodit z důvodu větší bezpečnosti (například útočník neví předem, kde šifrovaná hesla hledat).

Příklad 3.1

Podíváme se na výpisy různých příkazů na počítači s OpenSUSE (poněkud starší verze), je přihlášen jediný uživatel pracující na dvou terminálech. Nejdřív zjistíme vlastní identifikační čísla:

```
sarka@notebook$ id
uid=1000(sarka) gid=100(users) skupiny=16(dialout),33(video),100(users)
```

Takže mám UID (své identifikační číslo uživatele) 1000, patřím do skupiny `users`, jejíž GID je 100, a je to moje hlavní skupina. Pak patřím ještě do dvou dalších skupin. Kdybychom chtěli získat identifikační čísla jiného uživatele, zadali bychom jako parametr název tohoto uživatele.

Dále chceme vědět, kteří uživatelé jsou právě přihlášeni:

```
sarka@notebook$ who
sarka    :0           2012-04-03 10:28 (console)
sarka    pts/1        2012-04-03 10:30 (:0.0)
sarka    pts/2        2012-04-03 10:30 (:0.0)
```

Nikdo jiný než já není přihlášen. Z výpisu je zřejmé, že k přihlášení byla použita jen jedna konzola (ta, na které běží grafické rozhraní, relace 0), a na ní běží dva terminály. Výpis lze upřesnit zadáním příslušných parametrů:

```
sarka@notebook$ who -aH
JMENO      TERMINAL     CAS               ZAHALI        PID KOMENTAR UKONCENI
           system boot  2012-04-03 10:28
                        2012-04-03 10:28               635 id=si    signál=0 návkód=0
           úroveň běhu 5 2012-04-03 10:28                    minulá=S
                        2012-04-03 10:28              2136 id=l5    signál=0 návkód=0
sarka    ? :0           2012-04-03 10:28   ?          3288 (console)
LOGIN      tty1         2012-04-03 10:28              3893 id=1
LOGIN      tty2         2012-04-03 10:28              3894 id=2
LOGIN      tty3         2012-04-03 10:28              3896 id=3
LOGIN      tty4         2012-04-03 10:28              3899 id=4
LOGIN      tty5         2012-04-03 10:28              3901 id=5
LOGIN      tty6         2012-04-03 10:28              3902 id=6
sarka    + pts/1        2012-04-03 10:30 00:12        4281 (:0.0)
sarka    + pts/2        2012-04-03 10:30   .          4281 (:0.0)
```

Z výpisu je zřejmé, že byla nastavena úroveň běhu 5 (o úrovních běhu se budeme učit v kapitole 4.2.2 na straně 77), to znamená víceuživatelský režim pro běžný provoz. Na konzolách 1–6 není nikdo přihlášen (je tam pouze přihlašovací výzva), na konzole s grafickým režimem ale ano, a taktéž na konci seznamu vidíme dva spuštěné terminály (coby aplikace). Ve výpisu vidíme také časové údaje a PID příslušného běžícího procesu.

Dalším příkazem zjistíme, „kdo jsem“:

```
sarka@notebook$ whoami
sarka
```

Tento příkaz může být užitečný, pokud používáme mechanismy oprávnění a nejsme si jisti, jako který uživatel právě pracujeme.

V některých distribucích existuje také následující příkaz:

```
sarka@notebook$ who am i
sarka    pts/2        2012-04-03 10:30 (:0.0)
```

Tento příkaz není všude podporován, oproti předchozímu výpisu jsme se dozvěděli některé informace navíc (například na kterém terminálu a konzoli dotyčný uživatel právě pracuje).

Poslední, ale zato nejkratší příkaz:

```
sarka@notebook$ w
 10:42:21 up 14 min,  3 users,  load average: 0,35, 0,44, 0,30
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
sarka    :0        10:28   ?xdm?   1:38   0.12s /bin/sh /usr/bin/startkde
sarka    pts/1     10:30   11:18   2.02s  1.98s top
sarka    pts/2     10:30    0.00s  0.32s  0.00s w
```

Teď je jasné, co na jednotlivých spuštěných terminálech vlastně běží.

Příklad 3.2

Předpokládejme, že pracujeme v grafickém prostředí (KDE) se spuštěnou konzolou, což celkem ani nemá vliv, dále na prvním terminálu (`tty1`, Ctrl+Alt+F1) jsme přihlášeni, ale právě nemáme spuštěn žádný interaktivní příkaz, na druhém terminálu jsme přihlášeni a máme spuštěn přikaz `cat` a na třetím terminálu jsme také přihlášeni a máme spuštěn příkaz `top` (interaktivní příkaz vypisující stav spuštěných procesů). Po zadání příkazu `w` získáme tento výstup:

```
 21:24:57 up 1:11,  4 users,  load average: 0.14, 0.09, 0.19
USER   TTY    LOGIN@   IDLE    JCPU    PCPU    WHAT
sarka  :0     20:14    ?xdm?   40.17s  0.12s   /bin/sh /usr/bin/startkde
sarka  tty1   21:20    4:35    0.08s   0.08s   -bash
sarka  tty2   21:21    3:06    0.06s   0.00s   cat
sarka  tty3   21:24   18.00s   0.22s   0.14s   top
```

V záhlaví najdeme momentální čas, dále jak dlouho běží systém, kolik uživatelů je právě přihlášeno a průměrné doby načítání programů v posledních 1, 5 a 15 minutách (tyto údaje jsou pro každé spuštění tohoto příkazu mírně odlišné).

Další záznamy se vztahují k přihlášeným uživatelům. Protože k tomuto systému se přihlašuji pouze já (síťová karta je zneaktivněna), jiné přihlašovací jméno v seznamu není. Ale zato pro jedno jsou zde čtyři řádky, tedy uživatel je přihlášen 4×.

Na řádku najdeme přihlašovací jméno, na kterém terminálu je uživatel přihlášen, kdy se přihlásil, celkovou dobu nečinnosti uživatele na terminálu, hodnota JCPU představuje souhrn času stráveného procesy z tohoto terminálu na procesoru (nezapočítávají se procesy běžící na pozadí), hodnota PCPU představuje dobu činnosti procesu, který je uveden v následujícím sloupci WHAT. Podle sloupce WHAT poznáme, který proces je hlavním běžícím procesem na daném terminálu (při práci v prostředí KDE je to vždy `startkde` se základním shellem, ať máme spuštěn jakýkoliv další program).

Úkoly

1. Srovnejte výpis příkazů `w`, `who`, `whoami`, `id` a `finger`, také s některými uvedenými parametry. Ve výstupu kterého příkazu zjistíte, kdy se uživatel přihlásil, jaké má UID a GID, jaký je jeho shell, jaký má domovský adresář?
2. Projděte si obsah všech konfiguračních souborů a adresářů zde zmíněných, pokud to dovolují vaše přístupová oprávnění.
3. Procvičte si práci s terminály (konzolami) – zkratkou Ctrl+Alt+F1 se přesuňte na `tty1`, přihlaste se (pokud to jde) a příkazem `tty` zjistěte název terminálu (resp. speciálního souboru pro tento terminál), dále zkratkou Alt+F2 na druhý terminál (proč zde nepoužíváme klávesu Ctrl ?), přihlaste se a podobně i na třetím terminálu.

   Na každém z terminálů spusťte některý interaktivní příkaz (například `man` pro některou manuálovou stránku, samotný `cat`, `yes`, `top` a nebo stránkovací příkaz na některý dlouhý soubor). Pak vyzkoušejte příkaz `w`. Odhlásíte se příkazem `logout`.

3.1.2 Vlastnosti účtů a skupin

Program **`passwd`** je jedním z nejdůležitějších příkazů pro nastavení vlastností uživatelských účtů a skupin. Nejčastěji je používán ke změně hesla, ale také dalších informací souvisejících s účtem (například lze zadat dobu platnosti hesla, plné jméno uživatele apod.), příkaz může být používán buď majitelem účtu (ne vždy) a rootem. Možnosti:

`passwd novak` chceme změnit heslo zadaného uživatele, budeme dotázáni na původní a nové heslo (pak pro kontrolu ještě jednou)

`passwd -g skupina` tímto příkazem lze měnit heslo skupiny (ano, i skupiny mohou mít heslo), změnu může provést buď root nebo hlavní člen skupiny

`passwd -f` takto změníme „veřejné údaje“, které jsou volně k zobrazení v souboru `/etc/passwd` nebo například programem `finger` (například plné jméno, telefonní číslo, apod.)

`passwd -s` změna login shellu (podobně jako `chsh`)

`passwd -S uzivatel` vypíše nastavení účtu uživatele (zda je účet zamknutý, je bez hesla nebo s heslem, datum poslední změny hesla, jak často musí být heslo měněno apod.)

`passwd -x 90 uzivatel` nastavíme maximální stáří hesla na 90 dnů (po uplynutí této doby musí uživatel změnit heslo během prvního přihlašování po uplynutí)

`passwd -w 85 uzivatel` po uplynutí této doby se uživateli začne objevovat upozornění, že by měl změnit heslo

`passwd -l uzivatel` zamknutí (lock) účtu uživatele

`passwd -u uzivatel` odemknutí (unlock) účtu uživatele

`passwd -e uzivatel` uživatel je přinucen změnit heslo při příštím spuštění (používá se u nově vytvořených účtů, aby si uživatel vytvořil heslo)

Pro práci s hesly skupin existuje také příkaz **`gpasswd`**.

Uživatele a skupiny lze přidávat nebo naopak odstraňovat. Při vytváření nového uživatelského účtu lze buď zadat všechny parametry přímo do příkazu, a nebo můžeme pro implicitní parametry použít soubor `/etc/default/useradd`.

Tento soubor může vypadat třeba takto:

```
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
GROUPS=video,novaskup
CREATE_MAIL_SPOOL=no
```

`SKEL` (zkratka ze „skeleton“, kostra) je adresář s implicitním profilem. V tomto adresáři můžeme vytvořit vše, co chceme, by se zkopírovalo do profilu přiřazeného k nově vytvořenému účtu.

Protože čísla UID bývají do 500 vyhrazená systémovým uživatelům (root, ale také některé speciální procesy, démony, mají své UID z důvodu vyšších přístupových práv), volíme UID vždy větší než 500 (a samozřejmě takové číslo, které ještě není vyhrazeno pro jiného uživatele), obvykle dokonce nad 1000.

Každý uživatel má svou hlavní skupinu, její GID je uvedeno na řádku uživatele v souboru `/etc/passwd`. Může však patřit do více skupin (to jsme viděli ve výstupech příkazů z předchozího textu) a příkaz `newgrp` umožňuje uživateli pracovat pod jinou skupinou než je jeho hlavní.

Práce s uživatelskými účty:

useradd

přidání nového uživatele

`useradd -m novak` vytvoříme nový účet s využitím implicitních parametrů, volba `-m` znamená, že se má automaticky vytvořit domovský adresář uživatele

`useradd -m -d /home/novak -gid 180 -s /bin/bash -c "komentář" novak` některé volby zadáváme ručně (domovský adresář, GID, shell a komentář), zbylé budou brány ze souboru `SKEL`

`useradd -m -f novak` účet je hned po vytvoření neaktivní

userdel

odstranění uživatelského účtu

`userdel novak` odstraní zadaný uživatelský účet

`userdel -rf novak` navíc odstraní také domovský adresář uživatele (volba `-f` znamená force, tedy nedbá se na varování v případě souborů nevlastněných daným uživatelem)

usermod

změna vlastností uživatelského účtu (obvykle už existujícího)

`usermod --inactive 3 novak` tento příkaz zneaktivní zadaný účet na 3 dny

`usermod -L novak` uzamkne účet (nebo `--lock`)

`usermod -U novak` odemkne účet (nebo `--unlock`)

`usermod -c "Nové Jméno" novakova` změna komentářového řetězce účtu

`newusers` *`soubor`*
: v souboru jsou sepsáni uživatelé, které je třeba přidat do systému, v podobném formátu jako je v souboru `/etc/passwd`, příkaz přidá všechny uživatele ze souboru do systému

`groupadd`
: vytvoření skupiny

`groupadd -gid 814 skupina` vytvoření skupiny, můžeme (nemusíme) zadat GID

`groupdel skupina`
: odstranění skupiny

`newgrp skupina`
: změna skupiny během práce uživatele (uživatel patří do více skupin)

Při vytváření uživatelských účtů a skupin můžeme také použít volby pro LDAP (tj. koexistenci s Active Directory v heterogenní síti).

Příklad 3.3

Hromadné přidání nových uživatelů: Vytvoříme textový soubor, kde každý řádek bude patřit jednomu uživateli. Formát je stejný jako u řádků v souboru `/etc/passwd`. Na řádku napíšeme postupně

- přihlašovací jméno,
- heslo v nezakódovaném tvaru (k tomuto souboru se proto nikdo nepovolaný nesmí dostat!!!),
- UID nového uživatele (toto pole se doporučuje nechat prázdné, pak bude vhodné UID vybráno automaticky),
- číslo výchozí skupiny, do které je uživatel zařazen, pokud je zde číslo GID, které ještě není přiřazeno žádné skupině, bude vytvořena nová skupina s tímto GID a s názvem stejným jako je login uživatele, jehož záznam je na tomto řádku obsažen,
- poznámka (obvykle se zde napíše skutečné jméno uživatele),
- domovský adresář (`/home/...`),
- výchozí shell uživatele.

Pokud uvedená skupina nebo adresář ještě neexistuje, automaticky jsou vytvořeny. Jednotlivé položky na řádku jsou odděleny dvojtečkou (pro zapamatování: v proměnné `PATH` a v souborech `/etc/passwd`, `/etc/group` a dalších jsou položky také odděleny dvojtečkou), například

```
komarek:px214ej:1206:506:Jan Komarek:/home/komarek:/bin/bash
```

Tento soubor použijeme jako atribut příkazu `newusers`, např: pokud je pojmenovan uzivatele.t, příkaz vypadá

```
newusers uzivatele.t
```

Úkoly

1. Zjistěte údaje o svém účtu, které se vztahují k heslu. V manuálové stránce zjistěte, co který vypsaný údaj znamená.

2. Zjistěte, jaké jsou výchozí údaje pro vytvoření nového účtu (vypište obsah příslušného souboru).

3.1.3 Přístupová oprávnění

Pro změnu parametrů souvisejících s oprávněním (vlastník, skupina, řetězec oprávnění) používáme tyto příkazy:

`chown uživatel soubor`
: změna majitele souboru, může použít pouze původní majitel nebo správce

`chgrp skupina soubor`
: totéž pro skupinu

`chmod práva soubor`
: změna přístupových práv souboru, obecně módu souboru, používá se ve dvou tvarech.

1. tvar (absolutní):
práva jsou tři číslice (vlastník, skupina, ostatní), každá z intervalu 0..7, každá z těchto číslic je součtem čísel
`4` – čtení
`2` – zápis
`1` – spouštění
`0` – nic

Například
`rw-r-xr--` $\Rightarrow 4+2+0, 4+0+1, 4+0+0 \Rightarrow 654$
$\Rightarrow$ použijeme `chmod 654 soubor`
`rwxr-x---` $\Rightarrow 4+2+1, 4+0+1, 0+0+0 \Rightarrow 750$
$\Rightarrow$ použijeme `chmod 750 soubor`

2. tvar:
použijeme, když chceme pozměnit jen některá práva,

Například
`chmod u+rw,g-w,o-rwx soubor`
takto jsme nastavili vlastníkovi souboru čtení a zápis, skupině jsme odebrali právo zápisu a ostatním jsme odebrali všechna práva,

`chmod g+r-w soubor`
skupině jsme nastavili právo čtení a odebrali právo zápisu.

`umask xyz`
: vypíše masku práv, která se implicitně strhávají při vytvoření nového souboru (standardně 022, tedy se použijí práva 755).

Nastavení příznaků módu souboru: V předchozím semestru jsme si vysvětlili, co znamenají příznaky `setuid`, `setgid` a `sticky`. Nyní se podíváme na možnost nastavení těchto příznaků.

`chmod u+s soubor`
: nastavení příznaku `setuid` pro soubor

`chmod g+s soubor`
: nastavení příznaku `setgid` pro soubor

`chmod +t soubor`
: nastavení příznaku `sticky` pro soubor

Příznaky rušíme podobně, jen místo znaku „+“ dáme „−“.

Příklad 3.4

Skupina uživatelů pracujících na projektu: Postupujeme takto:

1. Přihlásíme se jako root
2. `groupadd nova_skupina`
3. uživatelé u1, u2, u3, . . . jsou členy skupiny users a nové skupiny (nastavíme členství ve skupinách pro dané účty):
   ```
   usermod -G users, nova_skupina u1
   usermod -G users, nova_skupina u2
   ...
   ```
4. `mkdir /vol/novy_adresar`
5. nastavíme pro tento adresář skupinu a vlastníka (vedoucího skupiny – uživatele u1), dále přístupová práva:
   ```
   chgrp nova_skupina /vol/novy_adresar
   chown u1 /vol/novy_adresar
   chmod o-rwx /vol/novy_adresar
   ```

3.1.4 Navyšování přístupových oprávnění

Když potřebujeme navýšit přístupová oprávnění v textovém režimu, je k dispozici alespoň jeden z těchto příkazů:

`su [uživatel]`
: získání práv jiného uživatele (SubstituteUser), když není uveden žádný uživatel, doplní se automaticky root, další zadávané příkazy již spouštíme jako nový uživatel

`sudo [příkaz]`
: „SUperuser DO“, proveď `příkaz` (obvykle) jako superuživatel, tedy root, pokud neuvedeme příkaz, ve výchozím nastavení platí zvýšená oprávnění po určitou omezenou dobu (5 minut)

U obou příkazů existují také upřesňující volby, které zde nebudeme uvádět.

Příkaz `su` je starší a prakticky nekonfigurovatelný, v mnoha linuxových distribucích není používán. Naproti tomu chování příkazu `sudo` lze poměrně dobře konfigurovat a související mechanismus se také nazývá sudo mechanismus.

Sudo mechanismus tedy znamená (výhradní) využívání příkazu `sudo` k získávání vyšších přístupových oprávnění, uživatel *root* se obvykle ani nesmí přihlašovat klasickým způsobem. Konfigurace mechanismu je uložena v souboru `/etc/sudoers`, který je sice textový, ale lze jej editovat pouze speciálním programem `/usr/sbin/visudo`.

Pro roota je v `/etc/sudoers` obvykle záznam

```
root ALL=(ALL) ALL
```

Tuto konfiguraci lze použít i pro jiné uživatele, ale jen pokud jim plně důvěřujeme a sdělíme jim heslo roota. Typicky se dá využít pro alternativní administrátory.

Obecná syntaxe je

```
uživatel počítač=(efektivní uživatel) příkaz
```

První položka znamená uživatele, pro kterého řádek platí. Pokud chceme místo uživatele zadat celou skupinu, pak před její název napíšeme symbol `%`, abychom odlišili uživatele od skupiny, před název síťové skupiny oproti tomu dáváme symbol `+`.

Následuje počítač, na kterém je toto nastavení platné (tj. pokud je záznam v souboru jiného počítače, neplatí). Může to být přímo název počítače, IP adresa, adresa (pod)sítě apod.

V závorkách je název uživatelského účtu, ke kterému se takto dočasně přihlašujeme, a posledním parametrem je seznam příkazů, které takto uživatel může spouštět.

Příklad 3.5

Ukážeme si několik řádků, které mohou být v souboru `/etc/sudoers`.

`novak novakuvstroj.firma.cz = (ALL) /bin/kill,/bin/killall`
: zadaný uživatel může na svém počítači (i vzdáleně) spouštět příkazy pro ukončování jiných příkazů

`novak novakuvpc.firma.cz = (ALL) NOPASSWD: /bin/kill,/usr/bin/killall`
: totéž, navíc nebude vyžadováno heslo roota

`%wwwadmin www.webserver.cz = (www) /usr/local/apache/bin/apachectl`
: členové zadané skupiny mohou na zadaném web serveru spouštět zadaný příkaz jako uživatel *wwww*

`pepa mailserver = /bin, !/bin/su`
: zadanému uživateli na mailserveru dovolíme spouštět cokoliv z adresáře `/bin` kromě příkazu `su` (zde vidíme zápis negace), musejí autentizovat sami sebe (pracují pod svým účtem, není zde nic v závorkách)

`fileadmin ALL, !mailserver = ALL`
: zadaný uživatel může spouštět vše na všech strojích kromě mailserveru

`ALL = (ALL) NOPASSWD: /sbin/shutdown`
: všichni mohou všude spustit příkaz, který vypíná počítač, nebude vyžadováno heslo

Abychom nemuseli pokaždé zadávat dlouhé sekvence příkazů, je možné definovat tzv. aliasy sekvencí (to není totéž co alias příkazu). Rozlišují se aliasy pro uživatele (`User_Alias`), pro počítač (`Host_Alias`), pro efektivního uživatele (`Runas_Alias`) a pro spouštěný příkaz (`Cmnd_Alias`). Vestavěným aliasem je například `ALL` (je vytvořen pro všechny typy aliasů), znamená „vše“.

Příklad 3.6

Ukážeme si práci s aliasy sekvencí pro sudo mechanismus.

`User_Alias ADMINS=webadmin,mailadmin,honza`
: nadefinovali jsme alias pro uživatele (zde administrátory různých serverů)

`User_Alias SEKRETARKY=hanka,jana,pepa`
: podobně, pro sekretářky (pokud nastoupí nová sekretářka, přidáme ji do tohoto seznamu)

`Host_Alias SERVERS=master,mailserver,wwwserver`
: nadefinovali jsme alias pro umístění (zde různé servery)

`Runas_Alias OPERATORS=root,operator,zalohovac`
: tento alias je použitelný v „závorce“

`Cmnd_Alias KILL=/bin/kill,/usr/bin/killall`
: alias pro programy, jejichž spouštění chceme řídit, zde programy pro ukončování procesů

`Cmnd_Alias KONEC=/sbin/shutdown,/sbin/reboot`
: alias pro programy ukončující systém (vypnutí a restart)

`SEKRETARKY LOCAL = NOPASSWD: KONEC`
: sekretářky mají právo bez zadávání hesla ukončit nebo restartovat lokální počítač (LOCAL je implicitní alias pro místní počítač, na kterém je uživatel přihlášen)

`novak ALL, !SERVERS = (ALL) KONEC, KILL`
: uživatel může na všech počítačích kromě serverů v roli všech ukončovat procesy a běh systému

`ADMINS SERVERS = ALL`
: administrátoři mohou na serverech všechno (to není zrovna bezpečné)

3.2 Procesy a úlohy

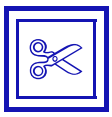
3.2.1 Příkazy pro práci s procesy

S procesy můžeme pracovat například pomocí těchto příkazů:

`top`
: Příkaz interaktivně vypisuje seznam procesů, které momentálně běží, a to v pravidelných intervalech. Ukončí se stiskem klávesy q (nebo jiným druhem ukončení).

  Příkaz `top` obvykle zobrazuje pouze ty procesy, které zabírají větší než zanedbatelné množství systémových zdrojů (především CPU).

Pro řízení výpisu můžeme používat některé klávesové zkratky, například Shift+P setřídí procesy podle zatížení CPU (Processor, může odhalit procesy, které se zasekly), Shift+M podle množství zabrané paměti (Memory), M zapneme nebo vypneme zobrazování informací o paměti zabrané procesy, K umožní bez nutnosti opuštění programu poslat některý signál vybranému procesu (zadáme PID procesu a signál), H nápověda.



ps

Vypíše všechny momentálně běžící (running) procesy na určitém terminálu (PID, speciální soubor terminálu, čas CPU, který proces již využil, název procesu). Můžeme použít například tyto přepínače:

`ps -f` přidá ještě některé informace, například UID, PPID procesu, čas spuštění

`ps -ef` totéž co předchozí, ale pro *všechny* procesy v systému, nejen z mého terminálu (tato forma je zřejmě nejpoužívanější)

`ps -u uzivatel` vypíše procesy určitého uživatele a další informace (zátěž CPU a RAM).

`ps -eH` takto zobrazíme také „stromovou strukturu“ procesů

`ps -eo uid,gid,sid,nice,nlwp,cmd` parametr `-o` umožní vybrat sloupce, které chceme vypsat

`ps -ef --sort uid,pid` můžeme zadat kritéria třídění (ne všechny sloupce lze takto zadat)

`ps aux` tato syntaxe (přepínače bez pomlček, trochu jiná písmena pro přepínače) se používá na systémech z řady BSD (například FreeBSD), ale je použitelná i v Linuxu (někteří lidé si „`aux`“ lépe pamatují)

pstree

vykreslí hierarchii procesů (strom vztahů rodič – potomek). Můžeme také používat některé přepínače, například `-c` zobrazí všechny procesy bez shlukování stejně nazvaných, `-p` zobrazí u každého procesu PID, `-a` zobrazí také parametry, se kterými byly procesy spuštěny. Program pracuje v pseudografickém prostředí. Podobně fungující program, ale v grafickém prostředí (KDE), jde `kpm`

pgrep proces

tento příkaz je vlastně kombinace příkazů `ps` a `grep`, vrátí PID zadaného procesu

pmap PID

podrobný rozpis paměti využívané procesem (nejen vlastněné), také zjistíme, jestli proces využívá sdílenou paměť’některé knihovny nebo jiného procesu. PID získáme například pomocí `pgrep`

whereis *příkaz*

takto zjistíme, kde konkrétně se příkaz nachází

`whereis top` zjistíme, kde konkrétně se nachází příkaz `top`, vypíše se

```
top: /usr/bin/top /usr/bin/X11/top /usr/share/man/man1/top.1.gz
```

To, co hledáme, je obvykle hned první cesta, tj. s celou cestou bychom příkaz spouštěli takto:

```
/usr/bin/top
```

`whereis sudo` vypíše, kde konkrétně se nachází příkaz `sudo` (a taky jestli je nainstalován, takto rychle zjistíme, jestli je podporován sudo mechanismus:

```
sudo: /usr/bin/sudo /usr/lib/sudo /usr/bin/X11/sudo /usr/share/man/man8/sudo.8.gz
```

(všimněte si rozdílu v cestě u poslední položky – takto poznáme také číslo sekce, ve které je manuálová stránka příkazu, zde 8)

Úkoly

1. Spusťte interaktivní příkaz, který v pravidelných intervalech zobrazuje seznam běžících procesů. Projděte si manuálovou stránku tohoto příkazu a zjistěte, jak lze ovlivnit délku intervalu, ve kterém jsou informace obnovovány.
2. Zjistěte údaje o procesu s PID 20 – kdo je vlastníkem procesu, kdo je jeho rodičovským procesem, jak se proces nazývá.
3. Vypište údaje o procesu s názvem *acpid* (zajímá nás především jeho PID, PPID, cesta, se kterou byl spuštěn, vlastník, případně další údaje). Pro zkrácení výpisu můžete použít vyhledávací filtr.
4. Zjistěte, jak využívá paměť proces *sshd* (tj. je třeba nejdřív zjistit jeho PID).
5. Zjistěte, kde jsou nainstalovány programy `man`, `mount`, `gcc` a `ls`.

3.2.2 Skupiny a relace procesů

Jak víme, v unixových systémech jsou procesy uspořádány ve stromové struktuře na základě vztahu rodič–potomek, potomci byli spuštěni rodičovským procesem.

Skupina procesů je seskupení takových procesů, mezi kterými existují blízké „rodinné“ vztahy. Je to vlastně podstrom ve stromové struktuře procesů. Proces, který je v kořeni tohoto podstromu, je hlavním procesem skupiny.

Skupina procesů má také své identifikační číslo – *PGID* (Process Group ID). Číslem PGID je PID hlavního procesu skupiny. Typicky tvoří skupinu procesy spuštěné v rámci jediného terminálu.

Příklad 3.7

Máme spuštěn program `konsole` a v něm (v jedné ze záložek) manuálovou stránku některého příkazu. V tom případě je v rámci `konsole` spuštěna instance shellu, v ní program `man`, který využívá stránkovač `less`.

Ve výpisu příkazu `ps -eHo pid,pgid,comm` budou kromě jiného i tyto řádky:

```
 PID  PGID   COMMAND
...
4245  3980   konsole
4247  4247     bash
6466  6466       man
6478  6466         less
4774  4774     bash
6569  6569       ps
```

To znamená, že program `less` patří do skupiny, jejímž hlavním procesem je proces `man`.

Většina systémových procesů má PGID nastaveno na 0. Proces s PID 0 neexistuje, tedy nejsou zařazeny do žádné skupiny. To však neplatí pro všechny systémové procesy. Například proces `hald` (démon zajišťující nízkoúrovňový přístup k hardwaru, vrstva HAL) je hlavním procesem své vlastní skupiny procesů (svých potomků). Další skupiny procesů (ale už spíše v uživatelském prostoru) jsou tvořeny inicializačními procesy grafických prostředí (například v KDE jde o `kdeinit`).

Relace (session, sezení) je obdoba skupiny procesů na trochu vyšší úrovni. Relace je také podstrom stromu procesů, může souviset například s procesy jednoho přihlášeného uživatele (typicky souvisí s terminálem, ze kterého jsou procesy spouštěny).

Podobně jako skupina procesů je jednoznačně určena svým PGID (což je PID hlavního procesu skupiny), relace je jednoznačně určena svým SID (Session ID), což je PID hlavního procesu relace.

V mnoha případech se číslo SID shoduje s číslem PGID, u systémových procesů je to prakticky pravidlo. Odlišnosti najdeme u uživatelských procesů spouštěných ze společného (virtuálního) terminálu.

Příklad 3.8

Podíváme se na podobný výpis jako v předchozím příkladu. Ve výpisu příkazu `ps -eHo pid,sid,pgid,comm` budou kromě jiného i tyto řádky:

```
 PID   SID  PGID   COMMAND
...
4245  3980  3980   konsole
4247  4247  4247     bash
6466  6447  6466       man
6478  6447  6466         less
4774  4774  4774     bash
6569  6574  6569       ps
```

Vidíme, že na některých řádcích se SID a PGID liší. Jednotlivé záložky konzoly (zde jsou zobrazeny dvě, jde o procesy `bash`) tvoří samostatné virtuální konzoly, v každé konzole je jedna relace sdílející společné SID (což je PID procesu `bash`, který je v kořeni relace).

Skupiny procesů se využívají především z důvodu usnadnění komunikace mezi procesy. Procesy v rámci jedné skupiny jednodušeji komunikují, mohou si posílat tzv. *signály*.

Účelem relací je především zjednodušená *správa úloh*. S úlohami pracujeme vždy v rámci jedné relace, a uvnitř relace jsou jednotlivé procesy identifikovány kromě svého PID také tzv. číslem úlohy. Kromě toho se také využívají výhody snadnější komunikace mezi procesy v rámci relace, například pokud je ukončován hlavní proces relace, odešle všem svým potomkům signál k ukončení, a tedy se rekurzívně ukončí celý podstrom procesů.

Úkoly

Spusťte příkaz, který slouží k vypsání identifikačních čísel procesu (včetně PGID a SID), jak jste viděli na příkladech. V seznamu najděte skupiny procesů a relace.

Do výpisu můžete přidat také další sloupce, například `user` pro zobrazení vlastníka procesu.

3.2.3 Úlohy a multitasking

V shellu `bash` můžeme s procesy pracovat na vyšší úrovni – pracujeme *s úlohami*. Tento styl práce souvisí s multitaskingem a práce s úlohami vlastně znamená práci v multitaskovém režimu i v textovém shellu.

Úloha může

- běžet na popředí, pak pracujeme přímo s ní a nezobrazuje se prompt, taková je nejvýše jedna,
- běžet na pozadí (třeba nějaký delší výpočet), pak „nezabírá prompt“ a umožňuje nám spustit na popředí jinou úlohu,
- být pozastavená na pozadí, tedy na popředí může běžet jiná úloha, ale tato úloha zrovna neběží (pozastavené jsou také ty úlohy, které by jinak běžely na pozadí, ale nemají zrovna co dělat).

Klávesové zkratky pro práci s úlohami:

Ctrl+C ukončení úlohy běžící na popředí (některé programy tuto zkratku nereagují)

Ctrl+Z pozastavení úlohy na popředí (platí totéž)

Ctrl+L překreslení obrazovky terminálu nebo konzoly (hodí se například tehdy, když úloha na pozadí nečekaně pošle něco na výstup)

Když chceme úlohu hned *spustit* na pozadí, na konec připíšeme jako argument symbol & (za mezeru), například

```
dlouhý_výpočet &
```

Symbol & musí být posledním symbolem na řádku.

Některé úlohy mohou běžet na pozadí a nezdržují na terminálu uživatele, který může spouštět jakýkoliv počet jiných úloh. Každá úloha má přiřazeno *identifikační číslo úlohy*, které není totéž jako PID procesu, ale číslování je jen v rámci terminálu (konzoly) – reálně v rámci relace, kde uživatel pracuje (tedy mnohem nižší čísla). Pokud chceme toto identifikační číslo použít jako parametr některého z následujících příkazů, musíme před ním napsat znak %, abychom odlišili číslo úlohy od čísla procesu.

Každá úloha, která běží na našem terminálu, má tedy své číslo úlohy, číslo procesu a pak ještě další čísla (PID, SID, atd.). V příkazech je nutné rozlišit číslo úlohy od čísla procesu, tedy, jak bylo výše uvedeno, před číslo úlohy dáváme %.

Dále můžeme používat tyto příkazy:

`jobs`

Vypíše všechny úlohy na terminálu a jejich stav (běžící, pozastavena apod.), poslední z úloh, které jsme poslali na pozadí, je označena symbolem „+“, předposlední symbolem „−“

`fg [%uloha]`

Příkazem `fg` pošleme úlohu, která je na pozadí, zpět na popředí a začnou se vypisovat také její výstupy. Pokud je takových úloh více a příkaz je použit bez parametru, je poslána na popředí ta, která byla jako poslední „odsunuta“ (tj. ve výpisu úloh je označena symbolem „+“). Jako parametr můžeme zadat číslo úlohy, která má být přenesena na popředí (nesmíme zapomenout znak %).

`bg [%uloha]`
Spustí pozastavenou úlohu na pozadí. Opět můžeme v parametru zadat číslo úlohy. Pokud úloha právě nemá co provádět, zůstane pozastavená.

Příklad 3.9

Ukážeme si práci s úlohami. Nejdřív spustíme několik interaktivních úloh.

`yes > /dev/null &` spustíme program `yes`, který nedělá nic jiného, než že vypisuje řádky s písmenem „y“, tento výstup přesměrujeme do souboru `/dev/null`, aby se nezobrazoval, symbol na konci zajistí, že proces bude přímo směrován na pozadí; objeví se řádek (číslo může být jiné, zobrazuje se PID procesu):

```
[1] 5783
```

`cat &` tento příkaz známe, teď jsme ho spustili na pozadí (nemá co dělat, tak byl pozastaven), vypsalo se

```
[2] 5787
[2]+ Stopped      cat
```

`rev &` tento příkaz narozdíl od předchozího vše, co dostane na svůj vstup, obrací (reverze), vypsalo se

```
[3] 5793
[3]+ Stopped      rev
```

`jobs` chceme seznam úloh na tomto terminálu, výpis:

```
[1]   Running     yes > /dev/null &
[2]-  Stopped     cat
[3]+  Stopped     rev
```

Symboly + a – nám říkají, která úloha byla na pozadí poslána jako poslední, resp. předposlední.

`fg` teď jsme na popředí poslali tu úlohu, která byla na pozadí poslána jako poslední

Ctrl+Z tuto úlohu jsme poslali zpět na pozadí, zobrazí se

```
[3]+ Stopped      rev
```

`fg %1` na popředí pošleme úlohu s číslem 1, tj. `yes`

Ctrl+Z úlohu jsme poslali zpět na pozadí

`kill %1 %2 %3` ukončíme všechny tři úlohy, zobrazí se

```
[1]+  Terminated  yes > /dev/null &
[2]   Terminated  cat
[3]-  Terminated  rev
```

Úkoly

Vyzkoušejte postup ukázaný v příkladu 3.9. Můžete pracovat s interaktivními úlohami v příkladu uvedenými, případně i s dalšími, například `ispell` (kontrola anglického pravopisu), `nslookup` (interaktivní překlad DNS názvů na IP adresy), apod.

Procvičte si spouštění úloh na pozadí, vypisování úloh, přenášení na popředí, pozastavování a ukončování.

3.2.4 Komunikace procesů

Procesy mohou vzájemně komunikovat několika způsoby:

- posíláním signálů,
- posíláním zpráv (IPC, InterProcess Communication), zpráva narozdíl od signálu může obsahovat i data, používá se například při synchronizaci procesů,
- přes sdílenou paměť, do které má přístup více procesů,
- přes soubory, případně dočasné soubory, používá se i v mechanismu zřetězování příkazů pomocí pipes (rour) nebo socketů, s rourami lze zacházet jako se soubory.

Nás zajímají především signály. Běžícím procesům a úlohám tedy můžeme posílat signály (čísla znamenající jednoduché příkazy), například:

1 (SIGHUP) – tento signál posílá jádro procesům ve skupině, jejíž hlavní (rodičovský) proces byl „zavěšen“ – například terminál či komunikační program ukončen; typickou reakcí procesu na tento signál (především u démonů) je znovunačtení konfiguračních souborů, což je vlastně restart procesu, některé procesy se při obdržení tohoto signálu ukončují.

Pokud chceme, aby proces spouštěný určitým příkazem v takovém případě nereagoval na signál SIGHUP (tj. aby démon nenačítal znovu konfiguraci, resp. běžný proces se neukončoval při ukončení rodičovského terminálu či nepřesouval se místo ukončení pod proces `init`), použijeme pro jeho spuštění příkaz `nohup příkaz &`

2 (SIGINT) – přerušení procesu nebo úlohy *z klávesnice* (ekvivalent Ctrl+C)

9 (SIGKILL) – bezpodmínečné ukončení (proces po sobě neuklidí prostředky (např. paměť), zůstanou zabrány, dokud není ukončen rodičovský proces

15 (SIGTERM) – implicitní volba (když není uveden žádný signál, vybere se tento), zdvořile požádá proces, aby se ukončil a dá mu možnost po sobě uklidit prostředky, proces může tento signál ignorovat, pokud chceme ukončit strom procesů (v rámci jedné skupiny nebo alespoň se stejným – naším – UID), použijeme právě tento signál,

19 (SIGSTOP) – pozastavení úlohy, ekvivalentní použití klávesové zkratky Ctrl+Z

18 (SIGCONT) – pozastavená úloha může pokračovat v činnosti (continue)

Signály můžeme posílat pouze svým vlastním úlohám a procesům, kromě roota – ten může signál odeslat komukoliv.

K posílání signálů slouží příkazy **`kill`**, **`pkill`** a **`killall`**, některé terminálové (konzolové) programy mají tuto volbu také v menu.

`kill -9 6223` zaslání signálu č. 9 (SIGKILL) *procesu* s PID 6223

`kill -KILL 6223` totéž

`kill -9 %2` zaslání téhož signálu *úloze* č. 2

`kill %2` pokud nezadáme signál, je poslán signál 15 (SIGTERM) pro standardní ukončení procesu

`kill -l` výpis seznamu signálů (list)

`pkill -P 5421` pošle signál SIGTERM všem procesům, jejichž PPID (rodičovský proces) odpovídá zadanému

`pkill -u 1002` ukončí procesy zadaného uživatele (zadáváme UID)

`killall -g 3981` ukončí procesy ve skupině procesů se zadaným PGID (ukončí celý podstrom, není nutné zvlášť ukončovat jednotlivé procesy ve skupině)

Příklad 3.10

Chceme pozastavit úlohu, jejíž PID je 3189 a číslo úlohy je 3. Máme tyto možnosti:

- `kill -19 3189`
- `kill -19 %3`
- `kill -STOP 3189`
- `kill -STOP %3`

Je jedno, kterou možnost zvolíme, fungují všechny.

Signály se mohou lišit na různých distribucích Linuxu a samozřejmě na různých dalších unixových systémech. Seznam nadefinovaných signálů zjistíme na manuálové stránce příkazu `kill` nebo vypíšeme příkazem `kill -l`.

Příklad 3.11

Došlo k nějaké závadě, která znemožní odhlásit se, potom:

- přihlásíme se znovu
- zjistíme číslo starého (běžícího) shellu, který se nám předtím nepodařilo ukončit odhlášením, a to pomocí příkazu `ps`
- ukončíme tento shell příkazem `kill -9 PID`, kde za PID dosadíme zjištěné číslo procesu.

Úkoly

1. Spusťte příkaz `cat` bez parametrů a pozastavte ho (Ctrl+Z).
2. Spusťte na pozadí příkaz `yes`, jeho výstup přesměrujte tak, aby se nevypisoval:
   `yes > /dev/null &`
3. Stejně (na pozadí) spusťte příkaz `rev` bez parametrů.
4. Příkazem `jobs` zobrazte seznam úloh, zjistěte čísla těchto úloh.

5. Obnovte proces `yes` na pozadí, proces `cat` zastavte příkazem `kill`. Vypište seznam úloh a porovnejte s předchozím.
6. Příkaz `cat` spusťte na pozadí, přesuňte ho na popředí a pomocí klávesové kombinace ho ukončete. Ukončete také proces `rev` a `yes`.

3.2.5 Plánování spouštění procesů

Plánovat spouštění procesů lze dvěma různými mechanismy – *at* a *cron*. První z nich je jednodušší a slouží spíše k jednoduchému jednorázovému naplánování spuštění úlohy (například za hodinu nebo stanovený den v určitou hodinu), druhý má mnohem širší možnosti. Nejdřív se podíváme na jednodušší mechanismus:

`at`

naplánování spouštění programů nebo skriptů v zadaném čase, aby plánování fungovalo, musí být spuštěn démon `atd`

`at -l` vypíše seznam naplánovaných úloh

`at now + 2 minutes` takto naplánujeme úlohu (jako posloupnost spuštění příkazů a programů), po zadání se objeví prompt PS2 (obvykle symbol >) a můžeme zadávat jednotlivé příkazy, které se v zadané době mají spustit, konec zadávání: klávesová zkratka Ctrl+D stejně jako při vytváření souboru (ale můžeme využít také podobný postup, jaký známe z „here documents“ – strana 10)

`at Mon 18:00` v pondělí 18 hodin

`at May 15 07:15 2010` další způsob zadání času (15. května, atd.)

`at -r 7` ze seznamu naplánovaných úloh odebereme úlohu číslo 7 (číslo úlohy zjistíme s výpisu seznamu úloh)

Příklad 3.12

Ukážeme si práci s programem `at`. Nejdřív ověříme, zda je spuštěn démon `atd`, když zjistíme, že ne, tak ho spustíme (musíme mít vyšší přístupová oprávnění) a následně naplánujeme jednoduchou úlohu.

`ps -e | grep atd` výstup je prázdný (není vybrán žádný řádek)

`sudo /usr/sbin/atd` jsme požádáni o heslo a pak je démon `atd` spuštěn (toto funguje v takové linuxové distribuci, která používá sudo mechanismus; jinak bychom museli zadat příkaz `su` a pak spustit plánovacího démona)

`ps -e | grep atd` na výstupu bude řádek

```
5140 ?  00:00:00 atd
```

To znamená, že tento démon už běží (zřejmě bude zobrazeno jiné PID)

`at now + 2 minutes` začneme plánovat, zobrazí se prompt >

`> touch ~/novysoubor.txt` chceme, aby se v zadané době vytvořil nový soubor v domovském adresáři, klepneme na Enter a pak stiskneme Ctrl+D, prompt PS2 zmizí, jsme informováni o ukončení plánování objevením řádku

```
job 1 at 2010-05-15 04:47
```

a objeví se běžný prompt PS1, můžeme zadávat další příkazy

`at -l` vypíše seznam naplánovaných úloh, ta naše by tam měla být (pokud ještě neuplynuly ty dvě minuty)

`ls ~` po dvou minutách si vypíšeme obsah pracovního adresáře, měl by tam být nově vytvořený soubor

`at 05:01 << KONEC` zase budeme plánovat, ale využijeme mechanismus „here document“, abychom nemuseli lovit v paměti zkratku Ctrl+D, zadáme:

```
> echo Naplánováno
> KONEC
```

hotovo, naplánovali jsme, že v zadané době dostaneme zprávu, v této době se zobrazí tato informace:

```
You have new mail in /var/spool/mail/sarka
```

`cat /var/spool/mail/sarka` zobrazíme obsah zprávy:

```
From sarka@linux-2vp6.site Sat May 15 04:39:02 2010
Return-Path: <sarka@linux-2vp6.site>
X-Original-To: sarka
Delivered-To: sarka@linux-2vp6.site
Received: by linux-2vp6.site (Postfix, from userid 1002)
        id 3F471485BA; Sat, 15 May 2010 04:39:02 +0200 (CEST)
Subject: Output from your job        2
To: sarka@linux-2vp6.site
Message-Id: <20100642023781.3F471485BA@linux-2vp6.site>
Date: Sat, 15 May 2010 04:39:02 +0200 (CEST)
From: sarka@linux-2vp.site (Sarka Vavreckova)

Naplánováno
```

Tímto způsobem je tedy uživatel informován o události v systému, která se ho týká, a neplatí to pouze o plánování.

Nástroj `cron` je mnohem sofistikovanější. Démon, který sleduje čas a spouští naplánované úlohy, se nazývá `crond` (aby mechanismus byl použitelný, tento démon musí být spuštěn). V KDE je standardně přítomno grafické rozhraní pro používání tohoto nástroje, `kcron`. Podíváme se na základní syntaxi:

`cron`

nástroj (je spouštěn jako démon), který dokáže plánovat také pravidelné akce, například pravidelné zálohování.

`man crontab, man 5 crontab, man cron` získání nápovědy pro cron

`crontab -l` zobrazí se aktuální seznam naplánovaných úloh (to, co daný uživatel má právo vidět)

`crontab -e` editace záznamů cronu, soubor se otevře v editoru specifikovaném v proměnné `VISUAL` nebo `EDITOR` (pokud je to editor `vi`, měli bychom vědět alespoň to, jak ho ukončit – příkazem `:q` (nebo `:wq`, pokud chceme před ukončením uložit svou práci), význam je především v tom, že nemusíme zjišťovat, do kterého souboru vlastně máme záznam ukládat

Naplánované úlohy jsou uloženy v souborech `/etc/crontab`, nebo v některém souboru adresáře `/etc/cron.d`, a dále především v souborech

- `/etc/crontab` – základní soubor s naplánovanými úlohami
- `/etc/cron.d/` – adresář se soubory v podobném formátu, pro různé uživatele
- `/etc/cron.daily` – zde jsou úlohy, které mají být spouštěny denně pod účtem root
- `/etc/cron.weekly` – úlohy spouštěné rootem každý týden
- `/etc/cron.monthly` – úlohy spouštěné rootem každý měsíc
- `/var/spool/cron/` – adresář, ve kterém si cron udržuje informace o konkrétních plánováních, do tohoto adresáře má přístup pouze root, ale ani ten do souborů nezasahuje

Každá úloha je na samostatném řádku a má tuto formu:

```
minut hodin den měsíc rok příkaz
```

Pokud údaj nemá být zadán, napíšeme znak „`*`“.

Příklad 3.13

Pro naplánování spuštění příkazu pro smazání dočasných souborů v zadaném adresáři, a to denně v 14:05, bude v souboru `/etc/crontab` nebo `/etc/cron.daily` řádek

```
5 14 * * * rm -rf /home/uzivatel/*.tmp
```

3.3 Správa zařízení

3.3.1 Paměťová média

Protože jsou všechny adresáře v jediném stromě, odpadá problém s přepínáním mezi disky (oddíly na disku). Přesto však existují příkazy související s disky, které nás mohou zajímat.

`df`

vypíše volné místo na jednotlivých discích (disk free)

`df` ke každému oddílu na disku se vypíšou údaje o velikosti oddílu, kolik je využito (i procenta) a do kterého adresáře je připojen (přípojný bod)

`df -T` ke každému oddílu se vypíše také typ souborového systému (pokud ovladač souborového systému využívá modul FUSE, vypíše se, například místo ntfs, údaj „fuseblk“)

`df -Th` kromě výše uvedených údajů se údaje o množství místa zobrazí v „lidském“ formátu včetně jednotek (místo počtu bloků údaje v KB, MB, GB)

`df -i` údaje o dostupném, volném apod. místě se zapisují v počtech i-uzlů (inodes) místo v počtu bloků (tj. máme představu o počtu souborů a adresářů na oddílu)

`df -a` vypíšou se informace o všech připojených souborových systémech včetně mnoha virtuálních (ne všech), tento výpis má smysl například tehdy, když chceme přehled připojených souborových systémů

du

vypíše obsazené místo na disku (disk usage) zabrané konkrétním souborem nebo adresářem (včetně jeho podadresářů, rekurzívně)

`du` vypíše rekurzívně všechny adresáře v pracovním adresáři a ke každému množství místa, které zabírá (v blocích)

`du -a` podobně, ale kromě adresářů vypisuje i ostatní soubory a místo jimi zabrané

`du -h soubor.pdf` pro zadaný soubor vypíše množství místa, které zabírá, použitý přepínač způsobí výpis v „lidských" jednotkách

`du -h 'ls *.txt'` vypíše místo zabrané soubory s příponou TXT nacházejícími se v pracovním adresáři; příkaz nepracuje jako filtr za jiným příkazem, proto mu dáme výstup příkazu `ls` jako parametr (obrácené apostrofy jsou nutné, aby se vnořený příkaz vyhodnotil dříve než vnější)

lsof

vypíše seznam otevřených souborů (vychází ze seznamu procesů, ke každému zjišťuje, které soubory má tento proces otevřené) – list open files; zjistíme především název procesu, PID, uživatele, FD (buď číselný deskriptor běžného souboru nebo identifikace typu – pracovní adresář, mem – soubor mapovaný do paměti, apod., k deskriptoru bývá připojen symbol pro typ otevření – r, w, apod.), typ (např. DIR pro adresář, LINK pro symbolický odkaz, IPv4 pro IPv4 socket, atd.), čísla zařízení, offset souboru, číslo i-uzlu a samozřejmě název souboru

`lsof` vypíše všechny otevřené soubory, jde o velmi dlouhý výpis (na desktopu obvykle několik tisíc položek)

`lsof > ~/seznamlsof.txt` to už je lepší, výstup si můžeme prohlédnout v souboru

`lsof | grep "\.so[0-9\.]$" > ~/pouziteknihovny.txt` podobně jako v předchozím příkazu, ale na výstup se dostanou pouze používané soubory, které jsou knihovnami (mají příponu .SO, za příponou může být ještě číslo verze knihovny)

`lsof ~` získáme seznam procesů, které pracují s naším domovským adresářem

`lsof -u uživatel` vypíše pouze seznam souborů používaných procesy se zadaným uživatelem (můžeme zadat UID nebo název uživatele, když je jich více, tak je oddělíme čárkami, kolem čárek nesmí být mezery)

`lsof | grep pipe` zjistíme, které procesy právě komunikují přes nepojmenovanou (anonymní) rouru

`lsof | grep socket` zjistíme, které procesy právě komunikují přes socket

sync

synchronizace diskových oddílů a cache pamětí

fsck

kontrola vadných sektorů na disku (parametry najdeme v nápovědě), ve skutečnosti se používají varianty pro konkrétní souborový systém – `fsck.ext2`, `fsck.ext3`, `raiserfsck`, `fsck.vfat`, `fsck.ntfs`, apod.

`apropos fsck` takto zjistíme konkrétní příkazy pro různé souborové systémy

`fsck.ext3 /dev/sd9` provede kontrolu zadaného oddílu se souborovým systémem ext3 (oddíl by měl být odpojen)

`mkfs`
vytvoření souborového systému (v terminologii Windows zformátování, parametry najdeme v nápovědě), opět existují varianty pro různé souborové systémy – `mkfs.ext2fs`, `mkfs.ext3fs`, `mkfs.vfat`, apod.

Příklad 3.14

Po zadání příkazu `df -ah` získáme tento výstup (notebook dualboot Windows 7 a SUSE Linux):

```
Filesystem      Size  Used  Avail   Use%   Mounded on
/dev/sda7        20G  3.8G    16G    20%   /
/proc             0     0      0     -     /proc
sysfs             0     0      0     -     /sys
debugfs           0     0      0     -     /sys/kernel/debug
udev           2.0G  192K   2.0G     1%   /dev
devpts            0     0      0     -     /dev/pts
/dev/sda8        40G  188M    38G     1%   /home
/dev/sda9        60G  5.1G    51G    10%   /usr
/dev/sda2       117G   30G    87G    26%   /mnt/winC
/dev/sda5       168G  256M   168G     1%   /mnt/winD
fusectl           0     0      0     -     /sys/fs/fuse/connections
securityfs        0     0      0     -     /sys/kernel/security
none              0     0      0     -     /proc/sys/fs/binfmt_misc
```

Úkoly

1. Zjistěte, kolik volného místa je na jednotlivých oddílech pevného disku (vyzkoušejte, jestli funguje i na připojená výměnná média, jako například USB flash disky).
2. Zjistěte, kolik místa na disku zabírá adresář s dokumenty ve vašem domovském adresáři. Dále zjistěte, kolik místa na disku zabírá adresář `/var/log`.
3. Zjistěte, které procesy pracují s vaším domovským adresářem.
4. Zjistěte, které soubory jsou právě používány procesy, jejichž vlastníkem jste vy.
5. Zjistěte, kdo právě používá soubor `/dev/null`.

3.3.2 Připojování a odpojování pamětí

Pro připojení a odpojení diskového oddílu nebo výměnného paměťového média (na kterém je obvykle jediný oddíl) používáme tyto příkazy:

`mount`
připojí disk do systému (resp. jeho souborový systém)

`umount`
odpojí disk

Soubor `fstab`. Při správě disků pracujeme opět buď s nástroji dostupnými v grafickém prostředí, nebo používáme příkazy shellu a soubor `fstab` a příkaz `mount`.

Soubor `/etc/fstab` slouží k *evidenci připojitelných logických disků*. Každý řádek odpovídá jednomu disku nebo oddílu se souborovým systémem. Řádek se skládá z těchto částí:

1. Určení zařízení, které se bude připojovat (speciální soubor), pro různé disky například `/dev/sda1`, `/dev/cdrom`, `/dev/fd0`.

   V některých distribucích (jmenovitě novější verze SUSE) se můžeme setkat s poněkud chaotickým označením – místo `/dev/sdax` můžeme najít například `/dev/disk/by-id/ata-`*`výrobní-kód-disku`*`-partx`.
2. Název přípojného bodu (mounting point), přes který bude zařízení přístupné (`/mnt/c`, `/mnt/cdrom`, případně `/windows/C`, `/media/cdrecorder`). Zde uvedený adresář můžeme určit také sami, jenom ho musíme předem vytvořit.
3. Typ souborového systému na zařízení. Může zde být `ext2`, `ext3`, `reiserfs` nebo jiný linuxový souborový systém, `vfat`, `ntfs`, `msdos` pro disky, se kterými pracujeme ve Windows, `nfs` je pro disky vzdálených počítačů, pro CD je `iso9660`, pro DVD je to `udf` (Universal Disk Format), pro swap partition je to hodnota `swap`.

   Můžeme se setkat s s volbou `auto`, to znamená automatické rozpoznání souborového systému při připojování. Je to vhodné například pro diskety, USB disky nebo CD/DVD.
4. Seznam parametrů pro `mount`. Nejdůležitější jsou
   - `ro` (pouze pro čtení), `rw` (i pro zápis),
   - `user` (připojit může i běžný uživatel, odpojit tentýž uživatel), `users` (připojit běžný uživatel, odpojit kdokoliv, i někdo jiný),
   - `noauto` (po startu systému není připojen automaticky, musí se ručně, typické pro výměnná média),
   - `iocharset=iso8859-2` a `code=852` (slouží ke správnému nastavení češtiny, může být i jiná znaková stránka),
   - `locale=cs_CZ.UTF-8` – podobně, ale pro (Windowsovské) disky s názvy souborů v Unicode, dnes běžné na NTFS,
   - `exec` (povolení spouštět na tomto oddílu programy), `noexec` (zakázáno spouštět na tomto oddílu programy, velmi doporučováno pro datové disky),
   - `noatime` (při přístupu k souboru – access – nebude aktualizován čas posledního přístupu k souboru, může zrychlit práci s oddílem),
   - `nodev` (i kdyby byl některý soubor z tohoto oddílu označen jako speciální soubor zařízení, nebude s ním takto zacházeno, bude brán jako obyčejný soubor; z hlediska bezpečnosti se doporučuje pro všechny oddíly, kde se nepočítá s existencí speciálních souborů, například `/home`, `/usr`, datové disky),
   - `nosuid` (nastavené SUID a SGID bity budou ignorovány),
   - `sync` (zařízení bude při každém zápisu okamžitě synchronizováno, cache paměť se bude využívat jen minimálně; pro SSD disky a USB flash disky je tato volba nevhodná, zkracuje životnost disku), atd.

   Některé volby jsou použitelné pouze pro některé souborové systémy. Podrobnosti o volbách najdeme v manuálech – `man fstab`, `man mount`).

Pokud jsou uplatňovány ACL a rozšířené atributy přístupu (o nich v následující kapitole), setkáme se u některých oddílů s údajem `acl,user_xattr`.

5. Pokud je zde znak `1`, bude program `dump` tento oddíl zálohovat.
6. Pokud je zde znak `1` nebo `2`, bude program `fsck` tento oddíl kontrolovat při startu. Hodnota `1` určuje větší prioritu (zkontroluje se nejdřív), používá se pro hlavní oddíl, `2` je pro ostatní. Hodnota `0` se zadává pro ty oddíly, které se nemají kontrolovat (například windows oddíly nebo výměnná média).

Kořenový oddíl by měl obsahovat v posledních dvou částech znak `1`. S jeho řádkem bychom neměli nic dělat, hrozí riziko kolapsu (resp. nespuštění) Linuxu.

Příklad 3.15

Soubor `/etc/fstab` může vypadat například takto (zarovnání do sloupců je provedeno pomocí tabulátorů):

```
/dev/sda6     swap              swap     defaults               0 0
/dev/sda7     /                 ext3     acl,user_xattr         1 1
/dev/sda8     /home             ext3     acl,user_xattr         1 2
/dev/sda9     /usr              ext3     acl,user_xattr         1 2
/dev/sda2     /mnt/winC         ntfs-3g  users,gid=users,fmask=133,dmask=022,
locale=cs_CZ.UTF-8  0 0
/dev/sda5     /mnt/winD         ntfs-3g  users,gid=users,fmask=133,dmask=022,
locale=cs_CZ.UTF-8  0 0
proc          /proc             proc     defaults               0 0
sysfs         /sys              sysfs    noauto                 0 0
debugfs       /sys/kernel/debug degubfs  noauto                 0 0
usbfs         /proc/bus/usb     usbfs    noauto                 0 0
devpts        /dev/pts          devpts   mode=0620,gid=5        0 0
```

Jak víme, v `/etc/fstab` jsou připojitelné souborové systémy, kdežto v souboru `/etc/mtab` najdeme seznam souborových systémů, které jsou právě připojeny. Při uvedeném obsahu souboru `/etc/fstab` najdeme v souboru `/etc/mtab` tyto řádky:

```
/dev/sda7 / ext3 rw,acl,user_xattr 0 0
/proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
debugfs /sys/kernel/debug debugfs rw 0 0
udev /dev tmpfs rw 0 0
devpts /dev/pts devpts rw,mode=0620,gid=5 0 0
/dev/sda8 /home ext3 rw,acl,user_xattr 0 0
/dev/sda9 /usr ext3 rw,acl,user_xattr 0 0
/dev/sda2 /mnt/winC fuseblk rw,noexec,nosuid,nodev,allow_other,
default_permissions,blksize=4096 0 0
/dev/sda5 /mnt/winD fuseblk rw,noexec,nosuid,nodev,allow_other,
default_permissions,blksize=4096 0 0
fusectl /sys/fs/fuse/connections fusectl rw 0 0
securityfs /sys/kernel/security securityfs rw 0 0
none /proc/sys/fs/binfmt_misc binfmt_misc rw 0 0
/dev/sdb1 /media/disk vfat rw,nosuid,nodev,noatime,flush,uid=1000,
utf8,shortname=lower 0 0
```

Každý záznam je samozřejmě na jediném řádku, zde některé dlouhé záznamy „přetekly“ na následující řádek (řádky končící čárkou). Druhý výpis se může zdát méně přehledný, protože jednotlivá

pole na řádku jsou oddělována mezerou místo tabulátoru. Poslední vypsaný řádek patří právě připojenému výměnnému zařízení (USB flash disku).

Všimněte si vztahu mezi údaji v těchto dvou souborech a mezi výpisem, který jsme viděli v příkladu 3.14 na straně 52.

Připojování disků. Pokud je v `fstab` na určitém řádku volba `noauto`, musíme tento disk sami připojit, abychom ho mohli používat. Týká se to výměnných zařízení, jako jsou CD, DVD, USB flash disky nebo diskety.

Tento úkol je snadno řešitelný v grafickém prostředí – v kontextovém menu ikony představující paměťové médium (disketu, flash disk apod.) zvolíme Připojit (Mount), pro odpojení Odpojit (UMount), v různých distribucích mohou být tyto položky různě nazvány. Pokud na ploše (nebo jinde) tyto ikony nemáme, můžeme je tam přidat (v nastavení plochy určíme zobrazovaná zařízení).

V shellu použijeme například tyto příkazy:

`mount` (bez parametrů) vypíše připojené disky, podobně jako `cat /etc/mtab`

`mount /mnt/winD` pokud je tento přípojný bod uvedený v souboru `/etc/fstab`, pro připojení zadaného oddílu stačí příkazu `mount` tato informace, tedy připojíme zadaný oddíl

`mount -o r /mnt/winD` připojí systém pouze pro čtení

`mount -o r /mnt/winD` připojí systém pouze pro čtení

`mount -o rw /mnt/winD` připojí systém pro čtení i zápis, implicitní volba

`mount -l -t` *`fstype`* zobrazí seznam připojených oddílů se zadaným souborovým systémem

`mount -t vfat /dev/sdb2 /mnt/flash` připojení USB flash disku se souborovým systémem FAT32, zadáváme speciální soubor a přípojný bod

Připojování a odpojování vyžaduje práva správce systému (pokud nepoužíváme kontextové menu v grafickém režimu), proto před použitím příkazu `mount` nebo `umount` použijeme příkaz `su` nebo `sudo` pro získání práv superuživatele.

Příklad 3.16

Ukážeme si připojování a odpojování disků:

`mount /dev/sdb1 /home`
připojí partition 1 druhého disku, na něm jsou domovské adresáře uživatelů (to se obvykle provádí automaticky při startu systému)

`mount /dev/sda1 /mnt/C -t ntfs`
připojí partition 1 prvního disku se souborovým systémem NTFS (tady pozor, záleží, jaký ovladač NTFS používáme, většinou by to asi bylo trochu jinak)

`mount /dev/sda3 /usr -t ext3fs`
připojí partition 3 prvního disku, na ní jsou adresáře instalovaných programů (`/usr`), souborový systém ext3

`mount /dev/fd0 /mnt/floppy -t vfat`
připojí disketu se souborovým systémem FAT

`mount /mnt/floppy`
: využití pouze druhého údaje na řádku v `fstab`

`umount /mnt/cdrom`
: odpojení CD

Pokud přípojný bod není uveden v `fstab`, musíme příkazu `mount` dodat i parametry, které by jinak byly v `fstab` uvedeny (forma je uvedena v `man mount`).

Připojení USB flash disku. Následující postup je v novějších distribucích plně řešen automatickým připojováním (`automount`, `supermount`, `gnome-mount`, apod.), ale hodí se u hodně starých distribucí, které chceme zprovoznit na „letitém" hardwaru.

Když chceme připojit USB flashdisk, nejdřív vytvoříme odpovídající adresář v `/mnt`, pokud tam ještě není:
`mkdir -m 666 /mnt/flash` (parametrem příkazu mohou být i přiřazená přístupová práva, at'tento krok nemusíme dělat dalším příkazem).

Pak pro připojení stačí zadat `mount -t vfat /dev/sdb1 /mnt/flash` (použijeme samozřejmě správný speciální soubor, může být jiný než zde uvedený).

Jestliže si chceme trochu zjednodušit připojování, můžeme do `fstab` přidat řádek

```
/dev/sdb1 /mnt/flash vfat noauto,users,sync 0 0
```

Pak stačí pro připojení zadat jen `mount /mnt/flash` nebo flashdisk připojíme v grafickém prostředí.

3.3.3 Program `fdisk`

Jedním z nejdůležitějších programů pro správu disků je program `fdisk` (podoba s Windowsovským fdiskem je pouze v názvu a v základních funkcích). Příkazy používané v programu jsou v tabulce 3.1.

Příkaz	*Význam*
`fdisk -l /dev/hda`	vypíše informace o udaném disku
`fdisk /dev/hda`	zobrazí se okno pro práci s diskem
`man fdisk`	nápověda k fdisku,
`klávesa M`	nápověda při práci s programem
`klávesa P`	zobrazí oddíly disku
`klávesa N`	vytvoří novou partition (program nás vede, občas stiskneme nějakou klávesu, udáme začátek a velikost partition)
`klávesa D`	smaže partition

Tabulka 3.1: Příkazy `fdisk`u

Programem `fdisk` je možné vytvářet i jiné než linuxové partitions, včetně i msdos a vfat, ale nedoporučuje se to, DOS a Windows s takto vytvořenými partitions nemusejí pracovat správně. Narozdíl od DOSovského `fdisk`u, který umí vytvořit pouze jeden primární oddíl a ostantní musí být zahrnuty v extended partition, linuxový `fdisk` dokáže vytvořit běžné 4 primární odíly.

3.3.4 Hlavní a vedlejší číslo zařízení, udev

Jádro nepracuje s „řetězcovými“ názvy zařízení, ale pouze s jejich číselným označením. Každé zařízení má přiřazena dvě čísla:

- Hlavní číslo je jedinečné pro každý druh zařízení,
- Vedlejší číslo je jedinečné pro každou instanci druhu zařízení.

Například když máme dvě síťové karty, budou mít stejné hlavní číslo, ale jiná vedlejší čísla.

Ve výstupu příkazu `ls -las /dev` jsou tato čísla uvedena tam, kde u běžných souborů je jejich délka. Čísla zařízení se také obvykle dají zjistit v některém z nástrojů v grafickém rozhraní.

Pokud je pro správu zařízení místo *devfs* použit souborový systém *udev* (což je dnes obvyklé), pak jsou čísla zařízení dynamicky generovaná.

Souborový systém `udev`, jak již víme, slouží ke správě přístupu k zařízením. Jedná se o virtuální souborový systém, který poskytuje rozhraní mezi procesy a ovladači zařízení běžícími převážně v jádře. Jeho činnost zajišťuje démon `udevd`, který periodicky zjišťuje informace ze souborového systému `sysfs`. Díky tomu má neustále přehled o změnách v seznamu připojených zařízení a může včas na tyto změny reagovat.

Úkoly

Vypište obsah adresáře `/dev` tak, aby se u každého speciálního souboru zobrazilo hlavní a vedlejší číslo zařízení. Zjistěte hlavní a vedlejší číslo zařízení odpovídajících oddílům na vašem pevném disku.

3.4 Moduly jádra

Jádro Linuxu (technicky vztato, Linux je právě jádro) je monolitické a rozšiřitelné pomocí modulů. Úkolem modulů je tedy rozšiřovat funkčnost jádra.

Existují různé druhy modulů, například ovladače zařízení, ovladače souborových systémů, moduly zajišťující funkčnost sítě (například ovladače síťových protokolů), firewall, moduly pro sledování hardwaru včetně stavu baterie nebo teploty procesoru.

Moduly jsou uloženy v souborech s příponou KO (kernel object). Obvykle jsou uloženy v adresáři `/lib/modules/...kernel/`, kde najdeme podadresáře nazvané podle typů modulů (`fs`, `net`, `crypto`, apod.), ve kterých již jsou soubory s příponou KO.

Moduly lze do jádra zavádět za běhu, obvykle není třeba restartovat systém. Pokud je však modul špatně napsaný, může se stát, že při výskytu chybného či podezřelého chování bude nastaven některý z *tained příznaků*, což právě znamená problém v jádře, a jediný způsob, jak se těchto příznaků zbavit, je restart. Některé z tained příznaků jsou vážné (například „hardwarová chyba paměti“), jiné se moc neřeší (například „načten modul s nevyhovující licencí“, jako je proprietální nebo neuvedená).

Pro práci s moduly jádra můžeme využít například tyto příkazy:

`/sbin/lsmod`
: (list modules) vypíše seznam modulů, které jsou právě zavedeny v jádře, ke každému také jeho

velikost a kým je používán (zde zjistíme také závislosti modulů), tento program jednoduše načte a zformátuje údaje uvedené v souboru `/proc/modules`, tedy alternativně můžeme použít příkaz `cat /proc/modules`

`/sbin/modprobe`
: slouží k zavádění modulů do jádra nebo jejich odstraňování, také zjišťuje závislosti mezi moduly a při zavedení každého modulu zavede i moduly, na kterých je závislý

`/sbin/modprobe eth0` zavede do jádra modul `eth0`

`/sbin/modprobe -r fuse` odstraní z jádra zadaný modul a všechny moduly, které ho používají (pokud to jde; kdyby některý z modulů byl používán jakýmkoliv procesem, odstranění by nemohlo proběhnout)

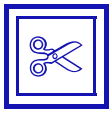
`/sbin/modprobe -l nfs*` vypíše všechny existující moduly (přesněji jejich soubory) vyhovující zadané masce

`/sbin/modprobe -l | grep bluetooth` pokud se hledaný řetězec nenachází přímo v názvu modulu, ale například v názvu adresáře, ve kterém je soubor modulu uložen, použijeme pro vyhledávání raději `grep`

Úkoly

1. Zjistěte, které moduly jsou právě načteny v jádře.
2. Vypište seznam souborů modulů jádra, které ve svém názvu obsahují řetězec „ipv4“ (tento řetězec může být z obou stran obklopen jakýmikoliv znaky).
3. Vypište seznam všech souborů modulů obsahujících řetězec `irda`.
4. Zjistěte, které moduly se vztahují k firewallu *netfilter* (tento řetězec nemusí být nutně v názvu souboru, pravděpodobněji jej najdete v názvu adresáře).

3.5 Operační paměť



V případě operační paměti můžeme používat tyto příkazy:

`free`
: zobrazí informaci o využívání operační paměti (především kolik operační paměti je volné, jak je používán odkládací prostor apod.)

`pmap PID`
: vypíše seznam všech souborů, které má zadaný proces namapovány v paměti, také údaje o využívání paměti procesem (kolik má soukromé paměti pro zápis a pro čtení, a kolik sdílené), mezi položkami je také paměť zásobníku (stack) a haldy (heap)

Veškeré údaje o využívání operační paměti (více než příkazem `free`) zjistíme ze souboru `/proc/meminfo`. Některé údaje o využívání paměti procesem získáme také příkazem `top`.

Zatímco v tradičních unixových systémech se virtuální paměť spravuje pomocí odkládání (swapování, tedy odkládání celého nedělitelného adresového prostoru procesu), v Linuxu je používáno

stránkování (paging, odkládání jednotlivých stránek procesu), třebaže je pro tuto činnost z historických důvodů také používán pojem *swap*. Délka stránek, na které je rozdělena virtuální paměť, je obvykle 1 kB nebo 4 kB, podle velikosti bloků na disku, kam se paměť odkládá.

V Linuxu je swapování optimalizováno tak, aby v případě nutné potřeby dalšího volného místa ve fyzické paměti byla menší pravděpodobnost, že bude potřeba odkládat některé stránky do swapu – stránky, které se nepoužívají, i když jsou rezervovány, jsou odloženy, aby zbytečně nezabíraly místo ve fyzické paměti.

Používá se buď swap soubor nebo swap partition (doporučuje se spíše partition), swap partition nemá žádný souborový systém (nelze ji připojit), v souboru `fstab` ale musí být uvedena (pokud je používána, samozřejmě) a místo označení souborového systému je zde řetězec `swap`. Swapovacích souborů i partitions může být používáno i více (obvykle až 8).

Příklad 3.17

Nové swap soubory můžeme vytvářet podle potřeby za běhu systému, vytvoříme soubor s určitými vlastnostmi (musí to být soubor bez prázdných míst s vhodnou délkou).

`dd if=/dev/zero of=/novy_swap bs=4096 count=65536`
: jeden ze způsobů vytvoření souboru vhodného pro použití jako swap

`mkswap /novy_swap`
: převedení existujícího souboru na swap soubor (na jeho začátek se zapíšou potřebné řídicí struktury, provádí se obvykle jen při instalaci)

`swapon /novy_swap`
: zapnutí používání souboru pro swapování, může to být kterýkoliv soubor, který byl označen pomocí `mkswap`

Pokud chceme používat swap partition, stačí, když je uvedena v souboru `fstab` s označením `swap` (může jich opět být více než jedna), zapnutí používání těchto oblastí pro swap se provádí také příkazem `swapon`.

Pokud máme instalováno více operačních systémů, za jistých okolností a po provedení některých nastavení mohou sdílet stejný odkládací prostor.

3.6 Síť

3.6.1 Soubory

Při správě malé sítě se může používat soubor `/etc/hosts`, který obsahuje doménové adresy a aliasy pro určité IP adresy v malé síti. Tento soubor už známe, dostal se i do systému Windows. Je zde také například řádek

```
127.0.0.1     localhost
```

Víme, že primárním cílem tohoto souboru je urychlit překlad těch doménových adres uzlů v síti (na IP adresy), které se typicky hodně používají. Tento soubor je používán protokolem IPv4 iIPv6, takže na systému podporujícím oba protokoly zde budou i řádky pro protokol IPv6, například

```
::1          localhost ipv6-localhost ipv6-loopback
fe00::0      ipv6-localnet
ff00::0      ipv6-mcastprefix
ff02::1      ipv6-allnodes
ff02::2      ipv6-allrouters
ff02::3      ipv6-allhosts
```

V adresáři `/etc` najdeme i další konfigurační soubory související se sítí, například `networks` (seznam známých sítí) nebo `ethers`.

Dynamické (momentální) nastavení sítě obvykle najdeme v podadresářích a souborech v systému `/proc`, například v souboru `/proc/net/arp` najdeme informaci k překladu IP a MAC adres.

Většina konfiguračních skriptů obvykle bývá v adresáři `/etc/sysconfig/network` a jeho podadresářích (příp. `/etc/sysconfig/network-scripts`), v některých linuxových distribucích to je adresář `/etc/rc.d/rc.inet1` (Slackware), `/etc/conf.d/net` (Gentoo) nebo `/etc/network` (Debian).

Příklad 3.18

Předpokládejme, že konfigurace sítě je v adresáři `/etc/sysconfig/network-scripts`. Přesuneme se do tohoto adresáře. Měl by tam být soubor `ifcfg-eth0`, ve kterém je uložena konfigurace prvního síťového rozhraní (karty) – IP adresa, maska, způsob získání IP adresy (pokud dynamicky, tak zde IP adresu nenajdeme) apod. Pokud máme více síťových rozhraní, pro každé z nich zde bude jeden takový soubor.

V souboru `/etc/resolv.conf` nastavujeme kromě jiného adresu DNS serveru (tj. když nevíme, kde tuto adresu zadat, podíváme se do tohoto souboru). Jde o řádek (nebo o více řádků, když chceme zadat i záložní DNS servery) ve formátu
`nameserver` *`adresa`*, například

```
nameserver 193.84.192.10
```

Zadaný DNS server bude využíván okamžitě po uložení souboru, není třeba restartovat systém ani žádný proces.

Příklad 3.19

Předpokládejme, že z počítače chceme vytvořit router. To není až tak těžké, stačí mít zařízení s více síťovými rozhraními a provést několik nastavení. Jedno z nich je povolení forwardingu (přeposílání). To se provede menší změnou v souboru `/etc/sysctl.conf`:

- najdeme řádek `net.ipv4.ip_forward=0` (je možné, že místo teček budou lomítka, pravidla syntaxe umožňují obojí)
- změníme na `net.ipv4.ip_forward=1`

- aby změna začala platit, musíme buď restartovat systém a nebo jednoduše přimět démona sysctl, aby znovu načetl své konfigurační soubory (včetně toho, který jsme pozměnili a samozřejmě uložili):
  `sysctl -p`

Ovšem zapnutí forwardingu nestačí, je třeba na počítačích v dotyčných sítích (které router propojuje) nastavit toto zařízení jako bránu (například pokud má karta eth0 IP adresu 193.84.200.1, u počítačů v síti, do které je připojena, nastavíme tuto adresu jako adresu brány, IP adresu karty eth1 zase použijeme v druhé síti, samozřejmě vždy tu adresu, která je v dané síti viditelná).

A pak musíme nakonfigurovat firewall a další bezpečnostní mechanismy.

Všimněte si, že není třeba restartovat počítač, třebaže jsme provedli podstatnou změnu v nastavení systému.

Úkoly

Projděte si všechny konfigurační soubory související se sítěmi, které byly zmíněny v této sekci a které jsou dostupné ve vaší distribuci.

3.6.2 Základní příkazy pro práci se sítí

Poznámka: V unixových systémech máme v oblasti správy sítí mnohem více možností, než kolik je popsáno v této podkapitole. O něco více informací je uvedeno například ve skriptech k předmětu Počítačové sítě a decentralizované systémy, v příloze C (tento text je zjednodušeným výřezem z uvedeného dokumentu). Skripta jsou dostupná na http://axpsu.fpf.slu.cz/~vav10ui/sitedec.html.

Při práci se sítí především komunikujeme přes protokoly ftp, telnet nebo ssh (ostatní jsou používány spíše v grafickém prostředí, např. http). Ke komunikaci přes tyto protokoly obvykle slouží příkazy stejně pojmenované.

`ftp` ⟨**`počítač`**⟩ přenos souborů; je nutné přihlásit se k počítači, který zadáváme (tj. přihlásit se). Po přihlášení obvykle můžeme zobrazit seznam příkazů, které můžeme používat, zadáním `?`.

`telnet` ⟨**`počítač`**⟩ příkaz pro vzdálené připojení k UNIXovému serveru, na kterém máme konto, je vyvolána přihlašovací procedura. Používáme všechny příkazy výše (i níže) uvedené, jako bychom seděli přímo u počítače, na který se přihlašujeme, včetně vyvolání nápovědy. Práci s telnetem ukončíme klávesou q.

`ssh` <**`počítač`**> podobně jako telnet, ale spojení je šifrované, proto bezpečnější, doporučuje se používat místo telnetu.

Dále v UNIXu samozřejmě najdeme prostředky pro komunikaci mezi uživateli přihlášenými ke stejnému UNIXovému systému, elektronickou poštu, atd.

Z nejdůležitějších příkazů pro práci se sítí:

ifconfig

zjištění MAC adresy (to je hardwarová adresa síťové karty – adresa na spojové vrstvě v OSI modelu), a také IP adresy pro určité rozhraní, masky sítě, atd., přidělení IP adresy určitému rozhraní (například ethernetové kartě – jako rozhraní použijeme eth0), je v adresáři `/sbin`

`ifconfig -a` zobrazí informaci o všech síťových rozhraních

`ifconfig eth0` zobrazí informaci o síťovém rozhraní eth0 (to obvykle bývá ethernetová karta), výstup může vypadat takto:

```
eth0  Link encap:Ethernet  HWaddr 00:0a:31:8a:75:72
      inet addr:172.28.124.128
      Bcast:172.28.124.255
      Mask:255.255.255.0

      inet6 addr: fe52::126a:24af:ff8b:818b/64 Scope:Link
      UP BROADCAST NOTRAILERS RUNNING MULTICAST
      MTU:1500  Metric:1

      RX packets:3612 errors:3558 dropped:0 overruns:0 frame:0
      TX packets:2344 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:384534 (375.5 KiB)
      TX bytes:798863 (780.1 KiB)
      Interrupt:19
      Base address:0x2024
```

`ifconfig eth0 down` „shodí“ zařízení eth0 (zneaktivní tuto kartu)

`ifconfig eth0 up` aktivuje zařízení eth0

`ifconfig eth0 down hw ether 00:00:00:00:00:02` kromě toho, že zneaktivní zařízení eth0, také mu přiřadí MAC adresu; podpříkaz `hw ether` *`adresa`* znamená, že jde o ethernetovou kartu, které přiřazujeme danou adresu (před podobnými změnami je vždy nutné kartu zneaktivnit)

`ifconfig eth0 172.119.124.104 netmask 255.255.255.0 up` nastaví IP adresu a masku podsítě pro eth0, pak je zaktivní (předpokládáme, že předtím byla tato karta neaktivní)

Tento příkaz má další volby, kterými lze například určovat multicast a broadcast adresy, přidělovat zdroje (I/O paměť, IRQ apod.), nastavovat metriky, atd.

arp

práce s tabulkami ARP (tyto tabulky slouží k rychlejšímu překladu mezi IP adresou a hardwarovou – MAC – adresou síťové karty)

`arp -a -n` zobrazí ARP tabulku, druhý přepínač znamená, že všechny adresy se zobrazí v číselném tvaru (místo doménových názvů)

`arp -n -i eth1` zobrazí ARP tabulku zadaného síťového rozhraní (to následuje za přepínačem `-i`)

`arp -s 123.123.123.123 -i eth1` přidání záznamu do ARP tabulky pro kartu eth1

`arp -d 123.123.123.123 -i eth1` odebrání záznamu z ARP tabulky pro kartu eth1

route

práce se směrovací tabulkou (spuštěný bez parametrů vypíše hlavní směrovací tabulku)

host
tento jednoduchý program převádí IP adresu na DNS a naopak, podle toho, jaký parametr zadáme. Různými parametry můžeme řídit další zobrazované informace.

nslookup
kontrola činnosti DNS serveru, případně můžeme program spustit v interaktivním režimu (spuštění bez parametrů), ve kterém zadáváme příkazy interpretované programem `nslookup`, nápověda se zobrazí příkazem `help`

dig
Tento příkaz (zkratka z Domain Information Gropher) slouží k podobným účelům jako `nslookup`, ale nabízí více možností. Je u administrátorů oblíbenější než `nslookup`, zvláště pro účely testování nastavení DNS serverů.

`dig www.root.cz` získáme vyčerpávající odpověď se všemi potřebnými informacemi, kromě žádané IP adresy (resp. více adres) například adresu DNS serveru, který odpověděl, jak dlouho trvala komunikace, ke každé adrese informaci o třídě záznamu (kde je platný), typ záznamu, atd.

`dig -x 91.213.160.118` reverzní dotaz, chceme doménový název k zadané IP adrese

`dig www.root.cz +short` krátká odpověď ve stylu nástroje `nslookup` (tj. vypíše se pouze IP adresa)

`dig google.com ANY` chceme všechny typy záznamů týkající se daného serveru (výchozí jsou záznamy typu A, tj. IPv4 adresy)

`dig seznam.cz NS +short` vypíšou se doménová jména DNS serverů v zadané doméně, jejich IP adresy můžeme zjistit následným dotazem podle vypsaných doménových jmen

Další možnosti příkazu `dig` lze získat v jeho manuálové stránce.

Další skupinou užitečných příkazů jsou příkazy zjišťující dostupnost uzlu v síti, vypisující cetu k danému uzlu nebo související statistiky:

ping [-c n] [-R] počítač
zjištění dostupnosti a odezvy určité IP nebo doménové adresy. Funguje prakticky stejně jako tento příkaz ve Windows. Odesílá cílovému počítači testovací zprávy ICMP request (každou sekundu) a čeká na odezvu. Příkaz pracuje interaktivně, jeho činnost přerušíme například stiskem kláves Ctrl+C, pak vypíše souhrnnou statistiku (kolik paketů bylo odesláno, přijato, podíl ztracených, dále odezvu cílového počítače – nejrychlejší, průměr a nejpomalejší odpověď, střední odchylka). Cílový počítač zadáváme buď jeho IP adresou, nebo jeho URL (tj. www.xxx.zz).

Přepínač `-c` určuje, kolik paketů má být odesláno, přepínač `-R` zobrazí všechny uzly sítě, přes které vedla komunikace, další volby zjistíme z manuálových stránek.

traceroute počítač
tento nástroj slouží k trasování (vypsání cesty k zadanému počítači). Je obvykle v adresáři `/sbin` nebo `/usr/sbin`. Funguje podobně jako `ping -R`, ale výpis je podrobnější a možnosti příkazu rozšiřují další přepínače (v manuálových stránkách).

arping [přepínače] počítač
: příkaz s podobnou funkcí jako `ping`, ale používá jiný typ paketů (pakety posílané příkazem `ping` mohou být některými servery a firewally ignorovány či zahazovány).

  Můžeme používat přepínač `-c` a také některé další přepínače se stejným významem jako u `ping`, dále např. přepínač `-b` způsobí, že všechny pakety budou zasílány formou broadcast (bez tohoto přepínače pouze první dotaz, ten slouží ke zjištění, kde hledat uzel sítě s danou IP adresou).

netstat
: kontroluje konfiguraci a provoz na síti, velmi komplexní nástroj s různými přepínači (pro zobrazení směrovací tabulky, zobrazení spojení, atd.) – pokud nepracujeme s přístupovými oprávněními roota, vypisují se jen informace o našich vlastních procesech a jejich spojeních

  `netstat -a` vypíše všechny nejdůležitější informace (otevřené síťové porty, vzdálený systém, se kterým se komunikuje, stav spojení

  `netstat -natp | grep LISTEN` zobrazí všechny otevřené porty protokolu TCP (přepínač `-t`, pro UDP by byl přepínač `-u`), součástí výpisu budou také procesy, které porty otevřely (přepínač `-p`), filtrujeme pouze naslouchající procesy (čekají na příchozí spojení)

  Implementace příkazu `netstat` se liší na různých unixových systémech, především na BSD systémech se setkáme s odlišnými přepínači nebo mírnými odlišnostmi v množství vypsaných informací (sloupců) u tohoto příkazu.

  Některé podobné informace získáme také z výstupu příkazu `lsof` (strana 51).

Příkaz **iptables** slouží ke konfiguraci firewallu *netfilter* (což je standardní firewall v Linuxu, podrobněji v následující kapitole) – jde vlastně o obslužný program k modulu jádra *netfilter*, který je samotným firewallem. Existují další nástroje včetně těch s grafickým rozhraním, které zjednodušují práci s firewallem.

Vybavenost unixových systémů nástroji pro práci se sítěmi je obecně vysoká, ovšem v každém unixovém systému svým způsobem specifická. Některé firmy distribuující unixové systémy přidávají své vlastní typické nástroje.

Úkoly

1. Zjistěte dostupnost serveru `www.google.com`. Zjistěte cestu přes směrovače k tomuto serveru.
2. Zjistěte (vypište) informace o síťových rozhraních na svém počítači.
3. Projděte si manuálovou stránku příkazu `netstat` a zjistěte nejdůležitější přepínače. Porovnejte s přepínači, které se používají u stejnojmenného příkazu ve Windows.

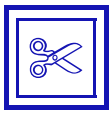
3.7 Mechanismus iproute2 (příkaz ip) – adresy, síť, směrování

Ve všech novějších distribucích se setkáme s ještě komplexním příkazem **ip**, který je součástí balíčku `iproute2`. Tento příkaz byl zařazen jako náhrada příkazů `ifconfig`, `route` a `arp` (pro úplnost jsme

se seznámili i s nimi), v současné době se místo těchto tří doporučuje používat spíše příkaz `ip`. Jeho konfigurační soubory najdeme v `/etc/iproute2`. Má specifické vnitřní příkazy, postupně probereme nejdůležitější varianty.

3.7.1 Konfigurace síťového rozhraní a adres

Nejdřív se podíváme na formu příkazu `ip` pro práci se síťovými rozhraními a IP adresami.



`ip link ...` pracujeme na vrstvě L2 (linkové) ISO/OSI modelu, vpodstatě i L1, tedy pracujeme s MAC adresami a síťovým rozhraním (například můžeme kartu odpojit či připojit)

- `ip link show` zobrazí stav spojení (tj. stav síťového rozhraní), místo `show` je možné všude používat `list` nebo `ls`, na multihomed zařízeních může být výpis velmi dlouhý; narozdíl od `ifconfig` se výpíšou i *ta síťová rozhraní, která* z nějakého důvodu *nelze řádně aktivovat*
- `ip -s link show eth0` zadali jsme název rozhraní (tj. nebudou se vypisovat informace pro všechna, pokud jich je více), a navíc parametr `-s` znamená podrobnější výpis
- `ip link set dev eth0 up` aktivuje rozhraní eth0
- `ip link set dev eth0 down` deaktivuje rozhraní eth0
- `ip link set dev eth0 address 02:00:00:00:11:22` změníme MAC adresu síťové karty eth0, všimněte si prvního oktetu; pozor, změna MAC nemusí někdy dopadnout dobře, jde o potenciálně nebezpečnou operaci

`ip addr ...` pracujeme na vrstvě L3 (síťové), tedy s IP adresami (místo `addr` lze použít `address` nebo `a`)

- `ip addr show` takto zjistíme IP adresu a související informace, případně můžeme přidat i název karty, která nás zajímá, jedna karta může mít i více než jednu IPv6 adresu (jednu primární a ostatní sekundární)
- `ip address show` totéž, také funguje `ip a show` nebo `ip a ls`
- `ip addr show dev eth0 primary` zjistíme primární adresu zařízení eth0
- `ip addr del 193.90.220.42/25 brd + dev eth0` odebereme kartě eth0 zadanou adresu; pozor – pokud se jedná o primární adresu, odeberou se i všechny sekundární

Příklad 3.20

Podíváme se na výstupy několika příkazů:

```
sarka@notebook:~$ ip link show
```

```
1: lo: <LOOPBACK,UP,10000> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,10000> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:1d:72:31:0a:a0 brd ff:ff:ff:ff:ff:ff
3: sit0: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
```

```
sarka@notebook:~$ ip -s link show
```

```
1: lo: <LOOPBACK,UP,10000> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    RX: bytes  packets  errors  dropped overrun mcast
```

```
    24914      270      0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    24914      270      0       0       0       0
2: eth0: <BROADCAST,MULTICAST,UP,10000> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:1d:72:31:0a:a0 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    379366     562      0       0       0       9
    TX: bytes  packets  errors  dropped carrier collsns
    100799     557      0       0       0       0
3: sit0: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
    RX: bytes  packets  errors  dropped overrun mcast
    0          0        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    0          0        0       0       0       0
```

Všimněte si, že v ostrých závorkách za názvem rozhraní jsou příznaky rozhraní (například zda jde o loopback, jestli je aktivní, povolení broadcastu, apod.).

```
sarka@notebook:~$ ip addr show
1: lo: <LOOPBACK,UP,10000> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,10000> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:1d:72:31:0a:a0 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.2/24 brd 10.0.0.255 scope global eth0
    inet6 fe80::21d:72ff:fe31:aa0/64 scope link
       valid_lft forever preferred_lft forever
3: sit0: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
```

Kdyby nás zajímalo pouze zařízení eth0, napsali bychom

```
ip addr show dev eth0
```

Úkoly

1. Vypište údaje o svém síťovém rozhraní, a pak je vypište včetně statistiky (podrobnější informace). Srovnejte oba výpisy. Určete, kolik máte síťových rozhraní (na koncové stanici to odpovídá počtu síťových karet plus případných virtuálních rozhraní), jakou máte velikost MTU, jaké máte MAC adresy, příp. jaká je broadcast MAC adresa, jaký je provoz na vstupu/výstupu rozhraní.
2. Zjistěte IP adresy (IPv4 i IPv6) na svých síťových rozhraních.

3.7.2 Směrování a filtrování

Příkaz `ip` se používá při práci se směrovacími tabulkami a při definování pravidel pro filtrování obdobně jako u firewallu.

V Linuxu neexistuje pouze jedna směrovací tabulka. Ve výchozím nastavení máme vždy alespoň hlavní (main) směrovací tabulku, která je zároveň výchozí (default), a lokální tabulku (v lokální směrovací tabulce najdeme především loopback (místní cesty) a směrování broadcastů a multicastů), další tabulky si můžeme dle potřeb vytvořit. Každá tabulka je identifikována svým jménem a číslem, v příkazech můžeme používat cokoliv z toho. Co se týče čísel směrovacích tabulek, tak u nově vytvořených můžeme použít čísla z intervalu 1 až 252, čísla předdefinovaných tabulek vidíme ve výpisu níže.

Zatímco příkaz `route`, se kterým jsme se již dříve seznámili, dovoluje přistupovat pouze k hlavní tabulce, příkaz `ip` umožňuje pracovat s jakoukoliv tabulkou.

Seznam směrovacích tabulek je uložen v souboru `/etc/iproute2/rt_tables`. Výchozím obsahem je

```
255     local
254     main
253     default
0       unspec
```

(plus řádky s komentáři, komentářový symbol je `#`).

Novou směrovací tabulku vytvoříme jednoduše přidáním nového řádku do tohoto souboru. Obvykle soubor needitujeme přímo, ale použijeme přesměrování s přidáním na konec:

```
echo číslo název >> /etc/iproute2/rt_tables
```

například

```
echo 15 novatab >> /etc/iproute2/rt_tables
```

Všimněte si, že při přesměrování jsme použili dvojitou „šipku“, protože nechceme přepsat původní obsah (na to pozor!), ale přidat nový záznam na konec souboru.

Příkazy:

`ip route ...` pracujeme se směrovací tabulkou, taktéž na vrstvě L3 (ale nad protokolem IP)

- `ip route show` zobrazí hlavní směrovací tabulku (pozor, ne všechny)
- `ip route show table local` zobrazí lokální směrovací tabulku
- `ip route show table 12` zobrazí směrovací tabulku číslo 12 (na tabulku se odkazujeme jménem nebo číslem)
- `ip route add default via 193.90.220.1` nastavení výchozí brány pro všechny cíle, které ve směrovací tabulce nejsou přímo uvedeny
- `ip route add 193.90.100.0/25 via 193.90.220.3` přidání nového (statického) řádku do směrovací tabulky (zadáváme adresu sítě s prefixem a dále adresu zařízení, přes které se k první zadané adrese dá dostat)
- `ip route add 193.90.100.0/25 via 193.90.220.3 table administrativa` záznam jsme přidali do zadané směrovací tabulky `administrativa`, nikoliv do hlavní směrovací tabulky
- `ip route delete 193.90.100.0/25` odstranění řádku tabulky (příp. lze opět zadat tabulku)

`ip route add prohibit 193.221.88.0/28` takto zakážeme směrování na zadanou adresu, tj. defacto tuto adresu znepřístupníme ze zařízení s touto tabulkou (používají například zaměstnavatelé k zamezení přístupu z daného počítače/směrovače k určitým oblastem), zadaná podsít'či uzel je ohlášen(a) jako „no route to host“ (ICMP zpráva)

`ip route add blackhole 69.63.189.11` pokud někdo z lokální sítě odešle paket na tuto adresu (mimochodem, je to jedna z IP adres serverů Facebooku), paket bude zahozen a dotyčný nebude informován

Příklad 3.21

Ukážeme si rozdíl mezi hlavní a lokální směrovací tabulkou na klientské stanici:

```
sarka@notebook:~$ ip route show

193.84.195.0/25 dev eth0  proto kernel  scope link  src 193.84.195.30
default via 193.84.195.1 dev eth0

sarka@notebook:~$ ip route show table local

broadcast 127.255.255.255 dev lo  proto kernel  scope link  src 127.0.0.1
broadcast 193.84.195.0 dev eth0  proto kernel  scope link  src 193.84.195.30
local 193.84.195.30 dev eth0  proto kernel  scope host  src 193.84.195.30
broadcast 193.84.195.127 dev eth0  proto kernel  scope link  src 193.84.195.30
broadcast 127.0.0.0 dev lo  proto kernel  scope link  src 127.0.0.1
local 127.0.0.1 dev lo  proto kernel  scope host  src 127.0.0.1
local 127.0.0.0/8 dev lo  proto kernel  scope host  src 127.0.0.1
```

V hlavní tabulce bychom si měli především všimnout řádku začínajícího `default`, to je směrování na bránu lokální sítě.

Směrování lze spojit s filtrováním či podrobnějším řízením směrování podle zadaných *pravidel* (politik), čemuž říkáme *Advanced Routing* (pokročilé směrování). Zatímco příkaz `ip route` pracuje pouze s adresami, pomocí **`ip rule`** lze směrování ovlivnit také obsahem jiných polí IP datagramu (směrování podle zásad), takto lze implementovat i základní mechanismus NAT.

Význam více než jedné směrovací tabulky doceníme především ve spojení právě s příkazem `ip rule`, který nám umožňuje například pakety odcházející z jedné konkrétní adresy v místní síti směrovat podle jedné tabulky, a pakety odcházející z jiné adresy směrovat podle další tabulky, případně zajistit, že uzly v určité podsíti nebudou míst přístup k serveru s citlivými údaji, kdežto uzly z jiné (prověřené či administrativní) sítě k němu mít přístup budou.

Úkoly

Vypište hlavní směrovací tabulku a porovnejte s výpisem pomocí příkazu `route` (je v předchozí sekci). Najděte adresu výchozí brány.

3.7.3 Objevování sousedů

Protokol IPv4 používá pro objevování sítě protokol ARP, u IPv6 to je NDP (Neighbour Discovery Protocol), respektive mechanismus NDis (Network Discovery). Pro práci s „okolím“ máme k dispozici vnitřní příkaz příkazu `ip neighbour`.

`ip neighbour ...` pracujeme s vazbami mezi adresami L2 a L3 (tj. MAC a IP) ve stejné síti (tj. se „sousedy“, odtud klíčové slovo), v případě IPv4 jde vlastně o práci s ARP tabulkami, u IPv6 jsou to *neighbour tabulky* (klíčové slovo `neighbour` můžeme zkracovat na `neighbor`, `neigh` nebo `n`)

- `ip neigh show` zobrazí momentální stav ARP nebo NDP cache (tabulky)
- `ip neigh show dev eth0` zobrazí seznam sousedů připojených k rozhraní eth0
- `ip neigh show to 193.168.200/24` zobrazí se info o sousedech patřících do zadané podsítě
- `ip -s neigh show to 10.0.0.1` podrobnější statistika souseda se zadanou IP adresou (zjistíme navíc počet uživatelů záznamu a dále před kolika sekundami byl záznam použit/potvrzen/aktualizován)
- `ip neigh show nud permanent` zobrazí všechny sousedy, jejichž stav je „permanent“, obvykle jde o ručně přidané sousedy
- `ip neigh add 193.168.200.1 lladdr 00:0a:1b:2c:3d:4e dev eth2 nud permanent` přidali jsme do ARP tabulky statický záznam o sousedovi (zadáváme IP adresu, MAC adresu – Link Layer Addr, rozhraní, přes které je soused dosažitelný, a pak stav)

  pokud přidáváme nový záznam, můžeme hodnotu `nud` nastavit na jednu z hodnot `noarp`, `reachable`, `permanent` nebo `stale`, čímž také určíme, zda záznam bude mít omezenou platnost a jestli má být ověřován
- `ip neigh del 193.168.200.1 dev eth2` odstranili jsme záznam z tabulky

Příklad 3.22

Zatímco mezilehlé síťové zařízení (router, switch) má sousedů poměrně hodně, koncové zařízení připojené přes ethernet (konektor RJ-45) má obvykle jediného souseda – router nebo switch. Na koncovém zařízení vypadají výpisy následovně:

```
sarka@notebook:/home/sarka# ip neigh show
10.0.0.138 dev eth0 lladdr 00:21:63:e2:0f:98 STALE

sarka@notebook:/home/sarka# ip -s neigh show
10.0.0.138 dev eth0 lladdr 00:21:63:e2:0f:98 ref 12 used 172/172/152 STALE
```

Druhý výpis obsahuje podrobnější údaje o sousedovi (což je router, jehož uvnitř viditelná IP adresa je 10.0.0.138). Na řádku postupně vidíme IP adresu, rozhraní, přes které jsme k sousedovi připojeni, dále počet odkazů na toto připojení a tři časové údaje (počet sekund od chvíle, kdy bylo připojení naposledy použito/potvrzeno/aktualizováno).

Úkoly

Zobrazte seznam svých sousedů (ARP/NDP cache). Pokud máte více aktivních síťových rozhraní, pak zvlášť pro několik rozhraní. Vyzkoušejte zobrazení se statistikou (podrobnějšími informacemi) o daném propojení.

3.7.4 Tunely

Mechanismus `iproute2` slouží také k vytváření IP/IP tunelů. Lze použít pro vytvoření VPN tunelu, zapouzdření IPv6 do IPv4 na cestě bez podpory IPv6, vytvoření mostu mezi více síťovými rozhraními na jednom zařízení patřícími do různých sítí, vytvoření vzdáleného mostu, apod.

Je možné pracovat s několika různými typy tunelů, ale předně musíme mít v jádře načten příslušný modul pro daný typ tunelu. Nejběžnější jsou

- *IP-IP tunely* (modul `ipip`) – dokážou zapouzdřit jen unicast IP datagramy (tj. tunely typu point-to-point),
- *GRE tunely* (modul `ip_gre`) – zapouzdřují také jiné typy paketů, a to spojení unicast i multicast, jsou poměrně hodně používány,
- *SIT tunely* (Simple Internet Transition, modul `ipv6`) – slouží k propojení IPv6 sítí přes IPv4 sítě.

V příkazech pro práci s tunely se typ tunelu projevuje v povinném parametru `mode` (tedy mód tunelu), určuje způsob, jakým se má s transportovaným paketem zacházet (především jak a do čeho se má zapouzdřit). Používáme následující příkaz:

`ip tunnel ...` zabezpečený přenos, tunelování

`ip tunnel show mujtunel` zobrazí informaci o vytvořeném tunelu s názvem `mujtunel`, zobrazí se obvykle typ (mód) tunelu, vzdálená a místní IP adresa (konce tunelu), pak další zadané (nepovinné) vlastnosti jako název síťového rozhraní, ze kterého tunel vede, hodnota TTL pro pakety jdoucí do tunelu, hodnota TOS, apod. (vpodstatě se zobrazuje to, co se zadalo při vytvoření tunelu)

`ip -s tunnel show mujtunel` kromě výše uvedených informací se zobrazí také statistika tunelu podobná výstupu příkazu `ip -s link show` (jde vlastně o stejný typ informace, jen se místo síťového rozhraní týká tunelu), viz výstup na straně 65

`ip tunnel add mujtunel mode ipip local 194.50.20.42 remote 195.84.152.140 ttl 32` vytvořili jsme nový tunel typu IP-IP se zadanou místní a vzdálenou adresou a hodnotou TTL

`ip tunnel del mujtunel` zrušili jsme tunel

`ip tunnel change mujtunel remote 195.35.84.15` změna nastavení tunelu (změnili jsme vzdálenou adresu)

Abychom mohli tyto příkazy používat, musíme mít předně v jádře načten příslušný modul.

Kapitola 4

Nasazení systému

V této kapitole se budeme zabývat pokročilými mechanismy řízení přístupu, popisem běhu systému včetně jeho startu, nástroji na logování provozu, firewallem, projdeme si instalaci systému a aplikací včetně příkazů pro balíčkovací systémy, a v neposlední řadě se podíváme na možnosti zabezpečení Linuxu.

4.1 Pokročilé mechanismy řízení přístupu

4.1.1 POSIX ACL

V normě POSIX, kterou již (alespoň podle jména a určení) známe z předchozího semestru, jsou definovány také bezpečnostní *seznamy řízení přístupu*, ACL (Access Control List).

Samotné ACL se ukládají jako metadata souborového systému ve formě rozšířených atributů. Podobně jako u ACL ve Windows nebo v síťových zařízeních, i zde jde o to pro vyjmenované uživatele nebo členy skupin definovat přístupová oprávnění (jde o jemnější rozdělení než vlastník–skupina–zbytek). Princip přístupových oprávnění rwx je vpodstatě zachován, jen máme více možností pro jejich přiřazování. Rozlišujeme tři *typy ACL*:

- ACL pro uživatele,
- ACL pro skupiny,
- ACL pro ostatní

Pro práci s ACL slouží více různých příkazů. Jejich seznam (resp. seznam souvisejících manuálových stránek) můžeme získat například takto:

```
apropos -s 1 acl ; apropos -s 8 acl
```

(vypíšeme příkazy, které v popisu obsahují tuto zkratku, a to pouze z manuálových stránek v sekcích 1 a 8). Výstup může vypadat takto:

```
aclocal (1)     - manual page for aclocal 1.10.1
chacl (1)       - change the access control list of a file or a directory
getfacl (1)     - get file access control lists
setfacl (1)     - set file access control lists
```

```
smbcacls (1)    - Set or get ACLs on an NT file or directory names
slapacl (8)     - Check access to a list of attributes
tcpdmatch (8)   - tcp wrapper oracle
vfs_gpfs (8)    - gpfs specific samba extensions like acls and prealloc
```

Nás zajímají především příkazy `getfacl` a `setfacl`, které slouží k zobrazení a změnám ACL seznamů. Existující nastavení můžeme také exportovat (zálohovat) do souboru.

Příklad 4.1

Mechanismus ACL je transparentní, přístupové funkce fungují i v případě, že není zapnutý. Například na systému s nainstalovanými (to je obvyklé) a vypnutými ACL (nepoužívanými):

```
sarka@notebook:~$ getfacl ~/.bashrc
# file: .bashrc
# owner: sarka
# group: users
user::rw-
group::r--
other::r--
```

Pokud je na zadaný soubor uplatněn ACL, například uživateli `nadrizeny` mají být explicitně nastavena práva `rw-`, pak by zde byl jeden řádek navíc, a to

```
user:nadrizeny:rw-
```

Tedy formát výpisu je celkem jednoduchý. Nejdřív (zakomentovaně) je uveden název souboru či adresáře, vlastník a skupina, následují řádky s nastavenými ACL ve formě

```
typACL:jméno:oprávnění
```

s tím, že pokud jméno není uvedeno, jedná se o výchozí nastavení (u typu pro uživatele nastavení pro vlastníka, u typu pro skupinu nastavení pro výchozí skupinu souboru, u typu „ostatní“ pro nevyjmenovaný zbytek světa).

Příklad 4.2

Změnu ACL provedeme takto:

```
setfacl -m u:nadrizeny:rw-,g::rw-,g:apache:rwx,o:--- soubor
```

Jednomu z uživatelů jsme nastavili právo čtení a zápisu, skupině souboru taktéž, skupině vytvořené pro webový server apache jsme povolili vše (pozor, aby to bylo opravdu nutné) a zbytku světa jsme vše zakázali.

Všimněte si, že v poslední části nastavení ACL (pro zbytek světa) nejsou dvě dvojtečky, ale jen jedna. To je zcela vpořádku, u „ostatních“ se ani nepočítá s možností zadávat něčí jméno.

Lze definovat také masku, která například sníží všechna oprávnění, která byla explicitně nadefinována.

Úkoly

V manuálových stránkách zjistěte, jak lze při nastavování ACL některého adresáře použít rekurzi.

4.1.2 Atributy

Atributy v souborových systémech. Některé souborové systémy v Linuxu dnes podporují atributy. Jedná se především o souborové systémy ext2 a ext3, ale podporu atributů (trochu jinak pojatých) najdeme také například u XFS. Atributů je celkem dost (a rozhodně nemají nic společného s tím, co známe jako atributy u souborových systémů pro Windows). Kromě jiného jsou užitečné například

- `i` soubor nelze modifikovat, smazat ani přejmenovat, ani na něj nelze vytvořit odkaz
- `a` soubor lze otevřít pouze v režimu *append* (tj. můžeme jen přidávat data na konec, ale ne modifikovat původní obsah), hodí se například u důležitých LOG souborů
- `S` (velké S) po provedení jakýchkoliv změn dojde k okamžité synchronizaci, tj. data jsou okamžitě zapisována na disk
- `I` (velké I) je nastavován automaticky u všech adresářů, které byly indexovány pro vyhledávání
- `A` znamená, že u souboru nebude při přístupu k němu aktualizována hodnota *atime* (access time), což u často otevíraných souborů může urychlit práci s nimi
- `s` (malé s) je nastaven u souborů, jejichž smazání má být „důkladné“, bohužel implementace tohoto atributu je odlišná v různých distribucích a nemusí být zcela bezpečná (například obsah souboru může být při mazání přepsán nulami, náhodnými hodnotami, a nebo v nejhorším případě nemusí být na něj vůbec brán zřetel)

S atributy pracujeme pomocí těchto příkazů:

`lsattr`
: vypíše nastavené atributy pro zadaný soubor nebo více souborů (názvy souborů použijeme jako parametry příkazu), seznam souborů (či adresářů) můžeme tomuto příkazu předat i přes rouru (například `ls -a | lsattr`)

`chattr`
: slouží ke změně atributů souboru (některé atributy nelze měnit, například atribut o indexaci), jako parametr píšeme název souboru, jehož atributy chceme nastavovat, dále atribut a případně další volby (například pro rekurzivní nastavení atributů)

  `chattr +s soubor.txt` nastavení atributu „s“ pro zadaný soubor, po zadání příkazu `lsattr soubor.txt` získáme tento výstup:

  ```
  s-------------- soubor.txt
  ```

  `chattr -d souborbezzalohy.sh` takto odebíráme atribut (nastavený atribut „d“ znamená, že soubor nebude automaticky zálohován, tedy odstraněním atributu jsme zajistili, že soubor bude zálohován, pokud je to nastaveno pro daný oddíl)

Rozšířené (extended) atributy. Mechanismus rozšířených atributů umožňuje definovat vlastní názvy atributů (a samozřejmě i jejich hodnoty). Nejsme tedy odkázáni jen na omezenou množinu atributů souvisejících se souborovým systémem. Rozšířené atributy jsou používány především v souvislosti se zvýšením zabezpečení, setkáme se s nimi například v modulu SELinux a na nich je také založen mechanismus POSIX ACL.

Pro práci s rozšířenými atributy slouží příkazy `getfattr` a `setfattr`.

4.1.3 PAM

Systém PAM (Pluggable Authentication Modules) je mechanismus rozšiřujících bezpečnostních modulů používaný na Linuxu, ve FreeBSD a v Solarisu. Pomocí PAM lze například povolovat přihlášení jen v určitý čas a pouze z určitého místa (terminálu), stanovit limity na prostředky (počty spuštěných procesů na relaci, spotřebu paměti, apod., takto můžeme do určité míry zabránit útokům typu DoS), volit různé metody šifrování hesel, atd.

Je to systém založený na zásobníku s moduly (`pam_stack`) s tím, že při autentizaci uživatele je zásobník procházen shora dolů a každým modulem musí „projít“, tedy každý modul v zásobníku musí dát souhlas s autentizací.

PAM využívají různé služby, ke kterým se autentizují uživatelé (kromě jiného také služba přihlašování uživatele do systému nebo některé démony pro přístup ke své konfiguraci). Pokud služba chce využívat PAM, vytvoří si vlastní zásobník modulů, který specifikuje v souboru v adresáři `/etc/pam.d`. Tedy celkově je třeba vytvořit v tomto adresáři vlastní soubor a v něm specifikovat svůj zásobník. Pokud služba nechce vytvářet vlastní soubor, může použít výchozí `/etc/pam.d/other`.

Svůj zásobník si služba může vytvářet z modulů uložených v souborech, které se obvykle jmenují stejně a mají příponu .SO). Modulů je celkem hodně, například:

- `pam_access` – lze omezit místa, odkud se uživatel (nebo skupina uživatelů) může přihlašovat (platí i pro vzdálená přihlášení, lze zadat také IP adresy)
- `pam_time` – lze omezit čas přihlašování uživatelů
- `pam_securetty` – omezení možnosti přihlášení uživatele root ze specifikovaných zařízení (definujeme „bezpečné“ terminály pro přihlášení roota)
- `pam_cracklib` – kontrola hesel, zda nejsou snadno prolomitelná tzv. slovníkovým útokem
- `pam_pwhistory` – ukládá se historie hesel, čímž se dá zabránit, aby si uživatelé (nucení si občas měnit heslo) například nevolili cyklicky střídavě pár hesel
- `pam_tally` – tento modul sleduje množství neúspěšných přihlášení, která následují přímo za sebou; stanovíme hranici (například 3) a po překročení této hranice lze usoudit, že jde o slovníkový útok na heslo, tedy zablokujeme účet
- `pam_userdb` – při autentizaci se používá databáze Berkeley DB database (je indexována podle přihlašovacího jména, ke každému je uloženo heslo)

Seznam modulů získáme například z manuálových stránek, můžeme zadat

```
apropos -s 8 pam
```

Každý PAM modul musí být nějak nakonfigurován. Konfigurace je obvykle uložena v souborech `/etc/security/název.conf`, například konfigurace modulu `pam_access` je v souboru `/etc/security/access.conf`. Tyto soubory jsou obvykle dobře okomentovány, tedy pokud chceme konfiguraci změnit, obvykle nebývá problém.

Příklad 4.3

Konkrétní konfigurace ale může být uvedena také přímo u názvu modulu. Například do souboru `/etc/pam.d/login` přidáme řádek

```
account    required   /lib/security/pam_tally.so deny=3 no_magic_root
```

To znamená, že po třech selháních přihlášení (`deny=3`) bude účet zablokován, a platí to i pro přihlašování roota. Existují také další parametry, které bychom zjistili v manuálové stránce (`man pam_tally`, například doba, po kterou má být účet zamknut).

Úkoly

1. Podle návodu v textu vypište seznam PAM modulů.
2. Dále se podívejte do adresáře `/etc/pam.d` a vypište si obsah některých souborů, které v něm najdete.

4.1.4 Capabilities (kvalifikace)

Capabilities (kvalifikace, také schopnosti) představují nízkoúrovňový mechanismus (na úrovni jádra) řízení přístupu. Můžeme je chápat jako nízkoúrovňové ACL stanovené pro určité typy operací (ne pro objekty). Konkrétnímu uživateli lze přiřadit nebo odebrat určité oprávnění k provádění stanovené činnosti. Některé z nejdůležitějších kvalifikací:

`CAP_CHOWN` právo změnit vlastníka souboru

`CAP_KILL` právo zasílat signály cizím procesům

`CAP_SETPCAP` právo přenášet kvalifikace na jiný proces (standardně mají pouze vlákna jádra)

`CAP_LINUX_IMMUTABLE` právo nastavovat některé vlastnosti, například nastavovat či rušit atribut „`a`“ (soubor lze otevřít pouze v režimu append), je to jedna z mála možností jak ochránit nastavení atributů

`CAP_SYS_RAWIO` tato kvalifikace umožňuje provádět nízkoúrovňové I/O operace se zařízeními

Pro práci s capabilities existují funkce, ale lze také nainstalovat balíček s programem `lcap`.

Capabilities jsou sice funkční, ale problematický mechanismus (především proto, že procesy v uživatelském prostoru nemohou své kvalifikace volně předávat a dědit).

4.1.5 Chráněné prostředí pro běh procesu

Někdy je vhodné z bezpečnostních důvodů omezit prostředí, ve kterém daný proces běží. Pod pojmem prostředí se zde především myslí dostupnost některých „citlivějších“ souborů a adresářů. Toto upravené prostředí se nazývá *chroot* (changed root).

Jak víme, kořenovým adresářem je *root*, označovaný symbolem „/“. Tento adresář vidí jako kořenový většina procesů. Je však možné vytvořit nový adresář někde ve struktuře adresářů, nakopírovat do něj potřebné soubory a adresáře (jen v nejnutnější míře) a sdělit spouštěnému procesu, že tento vytvořený adresář je root a tam má také hledat vše, co bude potřebovat. Skutečný root je pro tento proces neviditelný, vidí pouze ten svůj, virtuální.

Při vytvoření a zprovoznění chroot prostředí potřebujeme tyto příkazy (některé z nich známe):

`mkdir`
: je třeba vytvořit adresář, který bude pro náš proces viditelný jako root

`cp`
: do tohoto adresáře musíme zkopírovat soubory a adresáře, které proces potřebuje ke svému běhu, protože jinak by se k nim nedostal

`chroot`
: tímto příkazem spustíme proces uvnitř našeho chráněného prostředí

`chroot /chranenyadresar program` spuštění programu v chráněném prostředí, adresář musí být předem vytvořen a do něj také zkopírován binární soubor s programem, jsou nutná vysoká přístupová oprávnění (tj. například předřadíme příkaz `sudo`)

`ldd`
: tento příkaz nám pomůže zjistit knihovny, které zadaný proces potřebuje ke svému běhu

`ldd /bin/bash` vypíše se seznam knihoven, které se dynamicky linkují při spuštění programu `/bin/bash` (dosadíme svůj program, který chceme spouštět v chráněném prostředí), vypsané knihovny musíme zkopírovat do adresáře pro chráněné prostředí se zachováním adresářové struktury (například zřejmě bude nutné vytvořit podadresář `lib` pro knihovny)

Chráněné prostředí chroot není samospasitelné. Existují způsoby, jak se proces uzavřený v chroot prostředí může dostat ven, například zneužitím příkazů `mount` nebo `mknod` (proces by mohl odpojit a znovu připojit souborový systém, čímž by zrušil izolaci, druhým příkazem by mohl vytvořit vlastní speciální soubor zařízení s přímým přístupem do paměti, čímž by zase získal příležitost zrušit svou izolaci). Tyto problémy se dají částečně řešit co největším omezením oprávnění a případně vytvořením chráněného prostředí na oddílu připojeném s takovými volbami jako je například nemožnost interpretovat speciální soubory zařízení a připojení pouze pro čtení.

4.2 Běh systému

4.2.1 Inicializace systému a proces `init`

Víme, že před startem systému je spuštěn zavadač (u Linuxu obvykle Grub nebo LILO). Tento zavaděč nejdřív nabídne možnost spuštění nainstalovaných operačních systémů nebo program na kontrolu paměťových modulů.

Předpokládejme, že jsme vybrali některý Linux. Zavádění systému probíhá takto:

1. Jádro je obvykle komprimováno, tedy je nutné ho rozbalit, zavést do paměti a zkontrolovat jeho integritu. Také se na začátku zavádí počáteční RAMdisk (*initrd*).
2. Vytvoří se tabulka stránek paměti (protože také jádro bude potřebovat paměť) a detekuje se nejdůležitější hardware (především procesor).
3. Inicializace jádra znamená reálné spuštění celého jádra: vytvoření potřebných datových struktur pro správu procesů, pamětí apod., nastavení obsluhy přerušení, inicializaci ovladačů, které

jsou zakompilovány do jádra (některé musejí být zakompilovány, aby vůbec jádro mohlo fungovat, například ovladače disků), detekci zařízení. Pokud je nastaveno automatické zavádění modulů jádra, provede se také nyní.

4. Vytvoří se první „opravdové“ výpočetní vlákno, což je proces `/sbin/init`. Proces `init` řídí veškeré další operace v systému včetně zbytku startu systému, a je také rodičem všech ostatních procesů. Úkolem tohoto procesu je také korektní ukončení systému.
5. Proces `init` řídí zbytek zavádění systému, a to pomocí úrovní běhu (viz dále). Provádí inicializaci uživatelské části systému. Jsou připojeny souborové systémy, aktivovány bezpečnostní technologie, síťová rozhraní, spuštěny démony, nastaveny a nastartovány konzoly, a nakonec také na jedné z konzol (obvykle sedmé) grafické prostředí.

4.2.2 Úrovně běhu

Unixové systémy ve standardu System V přejímají *úrovně běhu* (v BSD se nepoužívají). Úroveň běhu je stav procesu `init`. Podle toho, která úroveň běhu je použita, se určuje, které části systému mají být spuštěny.

0	Zastavení systému (systém neběží)
1	Jednouživatelský režim
2–5	Uživatelské režimy (běžný provoz)
6	Znovuzavedení systému (reboot)

Tabulka 4.1: Obvyklé úrovně běhu systému

Jednouživatelský režim je používán pro administraci systému, když je potřeba, aby se *root*ovi v systému „nemotali“ další uživatelé, zvláště tehdy, když potřebuje, aby v některém souborovém systému nebyly žádné soubory otevřené (například při zálohování nebo kontrole konzistence souborového systému). Pro běžný provoz se používají uživatelské režimy, každý unixový systém má pro tento účel stanovenu výchozí volbu (bývá to většinou 3, 4 nebo 5).

Výchozí úroveň běhu, která je použita při zavedení systému, je určena v souboru `/etc/inittab`. Jinou úroveň můžeme nastavit například při startu systému v zavaděči, a to spuštěním jádra s parametrem `single` (pro jednouživatelský režim) nebo `emergency` (pro záchranný režim), nebo za běhu systému příkazem `telinit`.

Každý systém si konkrétní význam úrovní 2–5 určuje sám, často bývá odlišena práce bez grafického rozhraní a s grafikou, případně se sítí nebo bez ní. Například úroveň 2 bývá víceuživatelský režim bez sítě a bez grafiky, úroveň 5 je víceuživatelský systém se sítí a grafikou.

Při spouštění systému je nastavena určitá úroveň běhu a proces init podle čísla úrovně pozná, co konkrétně má provést. V souboru `/etc/inittab` je pro každou úroveň běhu stanoven skript určující, co přesně se má provést, a také jsou stanoveny operace, které se mají provést při stisknutí klávesové zkratky Ctrl+Alt+Del. Například tam můžeme najít tyto řádky:

```
#  výchozí úroveň běhu:
id:5:initdefault:
# pokud nestartujeme do záchranného režimu, provede se následující:
```

```
si::bootwait:/etc/init.d/boot
#  pro jednotlivé úrovně jsou uvedeny skripty, které se mají spustit:
10:0:wait:/etc/init.d/rc 0
11:1:wait:/etc/init.d/rc 1
12:2:wait:/etc/init.d/rc 2
13:3:wait:/etc/init.d/rc 3
14:4:wait:/etc/init.d/rc 4
15:5:wait:/etc/init.d/rc 5
16:6:wait:/etc/init.d/rc 6
#  co se má provést v jednouživatelském režimu:
ls:S:wait:/etc/init.d/rc S
~~:S:respawn:/sbin/sulogin
#  co dělat při stisknutí Ctrl+Alt+Del:
ca::ctrlaltdel:/sbin/shutdown -r -t 4 now
...
```

(celý soubor je celkem dlouhý, plný komentářů). Tento soubor needitujeme, obvykle nebývá důvod. Změny se většinou provádějí spíše ve skriptech, na které je zde odkazováno, jako je soubor `/etc/rc.d/rc` (jeho první parametr bývá číslo úrovně běhu) a dále skripty, které jsou z něj případně volány.

Momentální úroveň běhu zjistíme příkazem

```
who -r
```

Poznámka: Pokud chceme restartovat systém například během instalace (tím není myšlen zoufalý čin při zamrznutí systému, to se v Linuxu obvykle nepřihodí, ale restart z důvodů nového načtení všech konfiguračních souborů), použijeme klávesovou zkratku Ctrl+Alt+Del. Jedná se o „regulérní restart“, na kterém se podílí operační systém, a nebo jinou operaci, podle nastavení procesu `init`. Technicky jde o poslání signálu SIGINT procesu `init`.

Úkoly

1. Zjistěte, která je momentální úroveň běhu.
2. Prohlédněte si obsah konfiguračních souborů zde uvedených (`/etc/inittab`, `/etc/rc.d/...`).

4.3 Logování provozu

Syslog je mechanismus logování a souvisejících úloh dostupný na každém zařízení, na kterém běží (téměř) jakýkoliv unixový systém včetně Linuxu. Jádrem mechanismu je démon *syslogd*.

4.3.1 Vstup syslogu

Démon *syslogd* čte svůj vstup ze zařízení (socketu) `/dev/log`. Tento vstup filtruje (vybírá to, co ho zajímá) podle konfigurace, která se nachází v souboru `/etc/syslog.conf` (to je tedy jeho konfigurační soubor) a pak podle nastavení ukládá hlášení o takto zjištěných událostech do jednoho nebo více LOG souborů.

Data, která syslogd přijímá, se skládají ze tří částí:

- *kategorie* určuje typ nebo odesílatele události, existují tyto kategorie:
  - auth – autentizace uživatelů,
  - authpriv – informace o autentizaci určená pro administrátora,
  - cron – zprávy od démona cron (plánování procesů),
  - daemon – zprávy od různých démonů,
  - kern – zprávy od jádra (kernel),
  - lpr – zprávy od tiskového subsystému,
  - mail – zprávy týkající se elektronické pošty,
  - mark – časová razítka (timestamps), které se pravidelně zapisují do logu,
  - news – diskusní skupiny,
  - security – totéž co auth,
  - syslog – vlastní zprávy syslogu (i z jiného uzlu v síti),
  - user – obvykle zprávy od aplikací v uživatelském režimu,
  - local0–local7 – pro tyto kategorie lze definovat vlastní význam,
- *priorita* určuje důležitost události:
  - emerg – systém je nepoužitelný nebo vážně ohrožen (emergency),
  - alert – je nutný okamžitý zásah,
  - crit – kritická situace,
  - err – chyba,
  - warning – varování,
  - notice – normální, avšak významná, zpráva,
  - info – informativní zpráva,
  - debug – ladicí zpráva (debugger),
- vlastní *text* zprávy.

Tento typ informací tedy *syslog* získá.

4.3.2 Filtrování vstupu

Jak bylo výše uvedeno, ve svém konfiguračním souboru má určeno, co s takovým záznamem provést. V tomto souboru jsou záznamy ve formě

```
kategorie.priorita  TAB  cíl
kategorie.=priorita  TAB  cíl
kategorie.!priorita  TAB  cíl
kategorie.!=priorita  TAB  cíl
```

(priorit může být i více, oddělují se středníkem, totéž platí i o dvojicích kategorie.priorita). Pokud je před prioritou jen tečka, nastavení platí pro uvedenou a všechny vyšší priority. Pokud chceme nastavení omezit jen na zadanou prioritu, přidáme před ni „=“. Symbol „!“ znamená negaci, tedy pokud je uveden, daná priorita (a všechny vyšší) nebude zahrnuta. Lze kombinovat: „!=“, nebude zahrnuta pouze uvedená priorita.

Mezi prioritami a cílem musí být vždy alespoň jednou stisknutý tabulátor, mezera nestačí.

Kategorii můžeme zadat symbolem `*`. To znamená, že se konfigurace týká jakékoliv kategorie.

Cíl určuje, kam konkrétně má být událost oznámena, logována. Může to být buď konkrétní log soubor (výchozí nebo jakýkoliv jiný), nebo výpis na konzolu či přeposlání na jiný počítač v síti. Pokud chceme, aby byla událost oznámena více cílům (například zobrazena určitému uživateli a zároveň uložena do souboru), oddělíme cíle čárkou. Možnosti:

- soubor – výchozí je `/var/log/messages`, ale můžeme si určit názvy souborů například pro různé kategorie nebo priority,
- `@server.firma.cz` nebo `@IPadresa` – událost bude přeposlána,
- `user1`, `user2`, `...` – událost bude oznámena zadanému uživateli (to může být root, admin, operator, apod., podle toho, jaké máme vytvořené uživatele), ale jen tehdy, když je uživatel právě přihlášen,
- `*` – událost bude oznámena všem přihlášeným uživatelům,
- `@loghost` – můžeme použít, pokud v souboru `/etc/hosts` je definován cíl `loghost`.

Lze použít také přesměrování do pojmenované roury, ze které pak může číst jakýkoliv proces, který určíme (a průběžně zpracovávat oznámení o událostech), stačí uvést název souboru a před něj napsat symbol roury:

```
kern.=err       |/var/log/jadro
```

Příklad 4.4

Takto nějak mohou vypadat záznamy v souboru `/etc/syslog.conf`:

```
mail.*                      /var/log/maillog
```

⇒ Všechny události z kategorie *mail* jsou uloženy do souboru `/var/log/maillog`

```
security.*;security.!=debug /var/log/secure
```

⇒ Všechny události z kategorie *security* kromě událostí s prioritou *debug* jsou uloženy do souboru `/var/log/secure`

```
cron.*                      /var/log/cron
```

⇒ Všechny události z kategorie *cron* jsou uloženy do souboru `/var/log/cron` (to znamená události související s plánovaným spouštěním procesů)

```
kern.debug;auth.notice      /dev/console
```

⇒ Události týkající se ladění jádra a běžné a přesto významné události autentizace jsou vypsány na systémové konzoli

```
authpriv.*                  /var/log/secure
```

⇒ Všechny „citlivější“ události autentizace jsou uloženy do souboru `/var/log/secure`

```
lpr.info                    /var/log/lpd-info
```

⇒ Informační zprávy a všechny s vyšší prioritou o událostech z tiskového subsystému se uloží do `/var/log/lpd-info`

```
*.err                       admin,/var/log/errors
```

⇒ Všechny chybové a vážnější zprávy jsou okamžitě oznámeny uživateli *admin* (pokud je přihlášen) a zároveň uloženy do souboru `/var/log/errors`

```
*.=crit                    /var/log/messages,@loghost,root
```

⇒ Všechny kritické události jsou uloženy do souboru `/var/log/messages`, dále poslány na adresu definovanou pod aliasem *loghost* a zároveň je okamžitě informován *root*, pokud je přihlášen

```
*.emerg                    *,/var/log/emergency
```

⇒ O mimořádně nebezpečných událostech jsou informováni všichni přihlášení uživatelé a zároveň je přidán záznam do souboru `/var/log/emergency`

Pokud chceme, aby *syslog* přijímal zprávy z jiných systémů, musíme spustit démona *syslogd* s parametrem `-d` (platí v Linuxu):

```
/usr/sbin/syslogd -m 0 -r
```

Přepínač `-m` určuje délku intervalu, v jakém se *syslogd* ozývá, tj. do logu zapisuje, že „žije“. Nastavením na 0 toto chování zrušíme. Parametr `-r` znamená „remote“, *syslogd* pak naslouchá na portu 514 a přijímá všechny UDP pakety.

Na FreeBSD je nutné použít jinou syntaxi:

```
/usr/sbin/syslogd
```

(bez parametrů, protože naslouchání na portu 514 je zde výchozí chování). Na FreeBSD je také možné určit uzly v síti, jejichž UDP pakety budou přijímány. Odlišnou syntaxi (i od této) mají systémy OpenBSD, Solaris a jiné, je tedy třeba vždy prostudovat manuálovou stránku:

```
man syslogd
```

Také je možné, že budeme muset povolit naslouchání služby *syslogd* na UDP portu. To se provede v `/etc/services` řádkem `syslog  512/UDP`, ale je pravděpodobné, že tam takový záznam už je. Na zařízeních, ze kterých jsou UDP pakety přijímány, je pak nastaveno výše popsaným způsobem zasílání na tento „sběrný“ počítač.

Pokud *syslogd* právě běží a my jsme provedli změnu jeho konfigurace (v souboru `/etc/syslog.conf`), musíme *syslogd* restartovat, aby zaregistroval změny ve své konfiguraci.

4.3.3 Výstup syslogu

Zatím jsme si ukázali, jaké údaje dostává *syslogd* na svůj vstup, podle čeho je filtruje a kam je ukládá. Zbývá podívat se, v jakém formátu. Na výstupu je záznam obsahující

- čas, kdy k události došlo,
- kategorii (kernel, mail apod.),
- text zprávy.

Součástí není priorita, protože ta už byla uplatněna při filtrování. Může zde být například záznam:

```
Feb 21 10:00:28 IOL kernel: device eth0 left promiscuous mode
```

Když syslog nastavujeme, hodí se možnost vyzkoušení jeho funkčnosti. K tomu může sloužit nástroj *logger*. Například událost kategorie *daemon* a priority *info* se zadanou zprávou vygenerujeme takto:

```
logger -p daemon.info "testujeme syslog"
```

Ruční správa logů může být celkem náročná. Je třeba hlídat obsah souborů a navíc sledovat jejich délku (včas umazat starší záznamy), stroj naslouchající na UDP portu by měl být dostatečně chráněn (je zde vysoké riziko DoS útoků, tedy firewall je rozhodně na místě). S některými úlohami mohou pomoci přídavné nástroje. Například rotaci logů (včetně označování či odstraňování starších záznamů) zvládne *logrotate*.

Existují také propracovanější verze *syslog*u, například *syslog-ng* (umí komunikovat i přes TCP a zvládne také regulární výrazy, nejen hvězdičku, v současných distribucích je velmi oblíbený), *nsyslogd* (komunikuje přes TCP/SSL), *Secure Syslog*, *Modular Syslog* a další. Volně šiřitelný program *NTSyslog*[1] (vlastně služba, kterou můžeme instalovat), umožňuje používat *syslog* také ve Windows (logy z Windows lze posílat na vzdálený systém a tam například vyhodnocovat). Nástroj *logwatch*[2] můžeme použít k sumarizaci logů, provádí analýzu logů za stanovené období a vytváří souhrnnou zprávu. Program swatch[3] je užitečný, když naopak potřebujeme být o určité události informováni pokud možno okamžitě – detekuje námi definované situace a stanoveným způsobem informuje, a protože je psán v perlu, je to velmi pružný nástroj využívající regulární výrazy.

4.4 Firewall

V linuxových jádrech (od verze 2.4) najdeme firewall *NetFilter*, což je obousměrný stavový firewall (provádí funkce SPI).[4] *NetFilter* dokáže filtrovat pakety podle údajů v hlavičkách protokolů TCP, UDP, IP, ICMP a také dalších uvedených v souboru `/etc/protocols`, je použitelný pro mnoho činností souvisejících se zpracováním paketů, včetně mechanismu NAT.

Jedná se o modul jádra, ke kterému se nepřistupuje přímo, ale přes obslužný program `iptables`.[5] Takže pozor – firewall je NetFilter, obslužný program běžící v uživatelském režimu je `iptables`.

4.4.1 Princip firewallu

Základem je *tabulka* (table), v každé tabulce je několik řetězců *chain* (čti: [čejn]), které již obsahují pravidla uplatňovaná na pakety (tato pravidla jsou zřetězená a na paket se uplatňují postupně, dokud se nenajde „pasující“ pravidlo, proto se tomu říká chain). Tabulka obvykle určuje, co konkrétně se má dít (například jen filtrování, nebo překlad adres či změna některých dalších údajů v záhlaví

[1] *NTSyslog* najdeme na http://ntsyslog.sourceforge.net.

[2] *logwatch* získáme na http://www.logwatch.org.

[3] Program swatch (Simple Watch) je dostupný na http://swatch.sourceforge.net.

[4] Stavové firewally dokážou pracovat samozřejmě nejen se stavovými, ale i s nestavovými protokoly (jako je například UDP), přestože se v některých publikacích můžeme dočíst, že ne.

[5] Ve starších jádrech, se kterými se (doufejme) už nesetkáme, se používal firewall *IPChains* se stejnojmenným obslužným programem (`ipchains`).

procházejícího paketu), chain obvykle určuje, kdy se to má dít (například na vstupu do sítě, výstupu, při přeposílání mezi jinými dvěma uzly, před směrováním, po směrování apod.).

Standardně jsou definovány tyto tabulky (v příkazu `iptables` se před název tabulky dává přepínač `-t`):

- *filter* (výchozí)
- *mangle* (pro transformace – upravujeme záhlaví, hodnoty TTL, pole TOS/QoS, apod.)
- *nat* (pro mechanismus NAT)
- *raw* a *conntrack* jsou pomocné mechanismy označování (marking) paketů pro pozdější zpracování a stavového filtru (SPI), obvykle je třeba jejich funkcionalitu do jádra doplnit (načíst moduly)

V každé tabulce je tedy několik *chainů* (řetězců), a to:

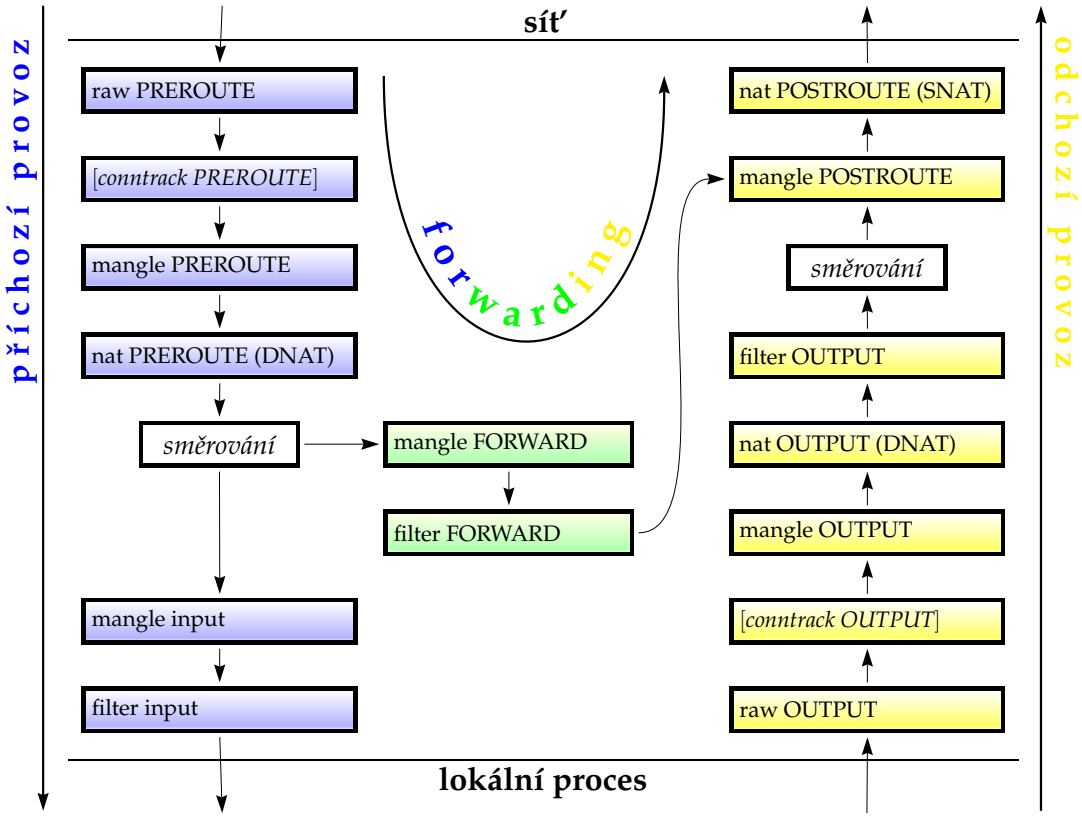
- Tabulka *filter* obsahuje standardně tyto chainy:
  - chain *INPUT* se uplatní na pakety, které do systému přicházejí,
  - chain *OUTPUT* se uplatní na pakety, které ze systému odcházejí,
  - chain *FORWARD* je určen pro pakety, které nejsou vytvořeny uvnitř systému a ani pro něj nejsou určeny, předávají se jen mezi rozhraními systému (průchozí pakety), typicky v zařízení, které slouží jako směrovač,

  pokud jde paket do chainu FORWARD, nepadne už do žádného jiného chainu.
- Tabulka *nat* obsahuje standardně tyto chainy:
  - chain *PREROUTING* pro příchozí pakety, na které se má použít DNAT (tj. má se přeložit cílová adresa, a to ještě před vlastním směrováním),
  - chain *POSTROUTING* pro odchozí pakety, na které se má použít SNAT (překládá se zdrojová adresa, až po směrování),
  - chain *OUTPUT* pro odchozí pakety, kdy se má modifikace provést ještě před dalším zpracováním.
- Tabulka *mangle* obsahuje standardně chainy pojmenované jako všechny, které jsou uvedeny u předchozích tabulek (INPUT, OUTPUT, FORWARD, PREROUTING, POSTROUTING) s tím rozdílem, že v nich najdeme pravidla modifikující i jiná pole záhlaví než jen ta adresová.

V tabulkách si můžeme vytvořit další chainy (uživatelsky definované), nemusíme zůstat jen u standardních, případně nové tabulky. Nové tabulky se však často přidávají jako další moduly jádra.

Na obrázku 4.1 je znázorněna dráha paketu mezi jednotlivými tabulkami a chainy. Všimněte si, že přeposílané (forwarded) pakety nedorazí do žádného chainu typu INPUT ani OUTPUT.

Program `iptables` umožňuje pracovat s chainy (vytvořit nebo zrušit chain, nastavit výchozí politiku) a s jejich obsahem (přidávat a odstraňovat pravidla v chainu, vypsat seznam pravidel). U pravidel v chainu záleží na pořadí. Každé pravidlo má své pořadové číslo. Nová pravidla můžeme do chainu vkládat na začátek či na konec chainu, a nebo na konkrétní pozici zadanou pořadovým číslem.



Obrázek 4.1: Dráha paketu mezi výchozími tabulkami a chainy v Netfilteru

4.4.2 Základní vnitřní příkazy a parametry pro filtrování

Ve vnitřních příkazech se používají tzv. *targety* určující akci, která se má s určeným paketem provést.

Můžeme použít například tyto targety (ve skutečnosti je jich více):

- `ACCEPT` akceptuj, paket má povoleno projít, obvykle v tabulce *filter*,
- `DROP` zahoď bez odezvy (tiché zahození), taktéž v tabulce *filter*,
- `REJECT` odmítni (s odezvou), odešle ICMP zprávu o odmítnutí (přesněji – o nedostupnosti adresy) uzlu, od kterého paket přišel, pro tabulku *filter*,
- `DNAT` a `SNAT` patří k tabulce *nat*, účel je zřejmý – akcí je překlad cílové nebo zdrojové adresy na jinou IP adresu, používáme při překladu statických adres,
- `MASQUERADE` také patří k tabulce *nat* chainu POSTROUTING, u odchozích paketů překládá celý rozsah vnitřních (zdrojových) adres na jednu vnější (slouží ke skrývání lokálních adres), narozdíl od SNAT bývá adres více a častěji se používá pro PAT (překlad portů), používáme pouze tehdy, když potřebujeme překládat dynamické soukromé adresy,

a další. Jednotlivé targety obvykle patří ke konkrétní tabulce, tedy když přidáme do jádra další modul s tabulkou, přidají se i další použitelné targety. Za targety se také považují uživatelsky definované chainy.

Příkaz `iptables` se používá s nejméně jedním parametrem (obvykle více), což bývá určení tabulky, vnitřní příkaz nebo parametr. Obvykle (ne však vždy) zadáváme tabulku, se kterou chceme v příkazu pracovat, k tomu použijeme přepínač `-t`), například `-t nat`. Zde se podíváme na vnitřní příkazy a parametry, které se používají při základním filtrování, v dalších podsekcích projdeme další (překlady adres, stavový firewall, značení paketů).

Obvykle tedy používáme některý vnitřní příkaz (pozor, velká písmena) představující:

`-P` nebo `--policy` určuje výchozí zásadu (politiku) pro chain, ve kterém chceme pracovat (pokud na paket nebude pasovat žádné pravidlo chainu, použije se tato politika), za přepínačem následuje název chainu a dále *target* určující akci, která se má provést, výchozí politiku lze určit pouze pro předdefinované chainy, nikoliv pro uživatelsky definované; například
`iptables -P INPUT -j DROP`,

`-L` nebo `--list` vypisuje seznam všech pravidel v zadaném chainu, například
`iptables -L` (výpis bývá dlouhý dokonce i na desktopu, všechny tabulky)
`iptables -t filter -L -n`
`iptables -t filter -L -nv`
`iptables -t filter -L INPUT -n`
první příkaz vypíše všechna pravidla tabulky *filter*, nechává IP adresy (nepřekládá na doménové, díky poslednímu parametru), druhý příkaz udělá totéž, ale ve verbose („upovídaném“) módu, tudíž se dozvíme více, třetí příkaz vypíše pouze pravidla v chainu INPUT,

`-A` nebo `--append` připojí nové pravidlo na konec chainu, následuje název chainu a specifikace pravidla

`-I` nebo `--insert` připojí nové pravidlo na začátek chainu (pokud nezadáme žádný index) nebo na zadaný index (číslo pravidla můžeme napsat za název chainu), dto., první index je 1,

`-R` nebo `--replace` přepíše existující pravidlo na daném indexu,

`-D` nebo `--delete` odstraní pravidlo z chainu, následuje název chainu a buď číslo pravidla nebo jeho specifikace,

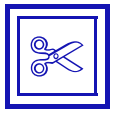
`-F` nebo `--flush` následované názvem chainu tento chain vyprázdní (vymaže všechna jeho pravidla)

`-Z` nebo `--zero` vynuluje všechny čítače v tabulce, obvykle se používá zároveň s příkazem `-L`, abychom viděli stav čítačů před jejich vynulováním, tedy například
`iptables -t filter -LZ`

`-N` nebo `--new-chain`, `-X` nebo `--delete-chain` pro vytvoření nebo odstranění chainu, následuje vždy název chainu,

`-E` nebo `--rename-chain` pro přejmenování chainu, následuje původní a nový název chainu.

Parametry slouží k upřesnění příkazu (určují, podle čeho se pakety mají filtrovat), narozdíl od příkazů jsou pro ně určena malá písmena, používáme především:



`-p` nebo `--protocol` určuje protokol, např. `-p tcp`, negace s vykřičníkem, například `-p ! udp`,

`--dport` nebo `--sport` dovoluje k protokolu podle předchozí odrážky přidat konkrétní číslo portu (zdrojového nebo cílového, také se dá označit názvem příslušného aplikačního protokolu), například

```
iptables -t filter -A OUTPUT -p tcp --dport 80 -j DROP
iptables -t filter -A OUTPUT -p tcp --dport http -j DROP
```
(obojí: zablokovali jsme odchozí tcp spojení přes http port)
```
iptables -t filter -A FORWARD -p tcp --dport imap -j ACCEPT
iptables -t filter -A FORWARD -p tcp --dport pop3 -j ACCEPT
```
(povolili jsme přeposílání paketů se stahovanou poštou, aplikační protokol IMAP a POP3)

`--icmp-type` můžeme použít filtrování pro určitý typ ICMP zprávy, například
```
iptables -t filter -A FORWARD -p icmp --icmp-type 8 -j DROP
```
zahazujeme všechny zprávy ICMP Echo Request (není to zrovna podle RFC, protože tím zablokujeme odpovědi na `ping`)

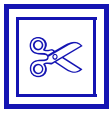
`--syn` v TCP záhlaví je nastaven příznak SYN (samotný bez příznaku ACK), což znamená, že zřejmě jde o paket, který navazuje nové spojení (ne nutně, také je důležité otestovat stav spojení, viz dále o SPI)

`-s` nebo `--source` nebo `--src` a `-d` nebo `--destination` nebo `--dst` je zdrojová a cílová adresa, u sítě píšeme i délku prefixu, například `-s 192.89.120.0/24`, opět můžeme použít vykřičník pro negování (tj. všechny adresy kromě této),

`-i` nebo `--in-interface` zadává vstupní rozhraní, používá se v chainech INPUT, FORWARD a PREROUTING, například `-i eth0`, můžeme také zadávat negaci, například `-i ! eth4` znamená pakety ze všech síťových rozhraní kromě eth4,

`-o` nebo `--out-interface` podobně pro výstupní rozhraní, na které je paket směrován (v chainech OUTPUT, FORWARD a POSTROUTING), například
```
iptables -t filter -A OUTPUT -o eth1 -j ACCEPT
```
veškerý ochozí provoz (neodpovídající jiným pravidlům) odcházející přes port eth1 bude povolen

Další parametry slouží k upřesnění činnosti, která se má s paketem provést:



`-j` nebo `--jump` slouží k zadání provedení targetu (viz výše), který za tímto přepínačem následuje, znamená tedy odskok (jump) k provedení targetu, kterým může být některá akce (ACCEPT, DROP, SNAT apod.), a *nebo* název uživatelsky definovaného chainu, do kterého takto „odskočíme“ s tím, že se po procházení uživatelského chainu můžeme vrátit zpět (tj. odskok se zapamatováním adresy), například
```
iptables -t filter -A INPUT -p tcp -j tcppakety
```
(pokud jde o TCP paket, přesuneme se do chainu *tcppakety*; pokud v tom chainu nenajdeme žádné vyhovující pravidlo, vrátíme se zpět do chainu INPUT a pokračujeme následujícími pravidly, uživatelský chain *tcppakety* byl předem vytvořen příkazem `-N`)

`-g` nebo `--goto` je pro uživatelsky definované chainy podobné jako `-j`, s tím rozdílem, že se před odskokem jinam nezapamatuje adresa momentálního chainu (přesněji – pokud už byla předtím některá adresa zapamatována pomocí `-j`, nepřepíše se jinou adresou, tudíž při případném návratu během vyhodnocování jednoho paketu se nevrátíme do chainu, ze kterého odcházíme, ale bude přeskočen), například
```
iptables -t filter -A INPUT -p tcp -g tcppakety
```

podobně jako předchozí, ale pokud v uživatelském chainu nenalezne vhodné pravidlo, už se do INPUT nevrací (ovšem v *tcppakety* může být také parametr `-j` nebo `-g` směrující vyhodnocování paketu jinam).

Zbývá ještě několik pomocných přepínačů používaných u příkazu `-L`:

`-v` nebo `--verbose` zapíná verbose („ukecaný“) mód, chceme delší výpis s více informacemi,

`-n` nebo `--numeric` použijeme, když chceme ve výpisu IP adresy místo doménových, výpis je pak celkově rychlejší a snadněji se hromadně zpracovává,

další bychom našli v manuálové stránce příkazu: `man 8 iptables`

Výše uvedený výčet příkazů a parametrů není zdaleka úplný, podrobnější informaci bychom našli opět v manuálové stránce příkazu, případně na odkazech na konci této sekce.

Poznámka: Měli bychom si uvědomit, že zpracovávání paketu v tabulce končí v okamžiku, kdy NetFilter najde první vyhovující pravidlo s cílovou akcí akceptující nebo zahazující (například ACCEPT nebo DROP). Pokud je v tomto pravidle akce DROP nebo jiná „zahazovací“, paket bude okamžitě zahozen, i kdyby třeba hned následující pravidlo tento paket povolovalo (k dalšímu pravidlu se nedostaneme).

Příklad 4.5

Pár ukázek práce s `iptables`:

`/sbin/iptables -t filter -L` výpis pravidel v tabulce `filter`

`/sbin/iptables -L -v -n --line-numbers` podrobný výpis (statistika) všech tabulek, s IP adresami (n), v chainech jsou řádky číslovány

`/sbin/iptables -P INPUT -j DROP` stanovíme výchozí politiku pro chain INPUT, ve které řekneme, že vše příchozí se má zahodit (samozřejmě musíme kromě jiného přidat před toto pravidlo také pravidla, která upřesní, co se zahazovat nemá)

`/sbin/iptables -P OUTPUT -j ACCEPT` všechno odchozí povolíme, propustíme dál (opět bychom měli uvažovat, jestli nepřidáme „výjimky“)

`/sbin/iptables -A INPUT -i lo -j ACCEPT` povolíme to, co probíhá mezi lokálními procesy (komunikují přes sockety na loopbacku)

`/sbin/iptables -A INPUT -p tcp --syn -j DROP` budou zahozeny všechny příchozí pakety, které mají v TCP záhlaví nastaven pouze SYN bit (to je vždy první paket spojení, tedy znemožníme navazování spojení zvnějšku)

`/sbin/iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT` povolili jsme pakety s ICMP zprávou č. 8 (Echo Request) z vnějšku, můžeme použít číselné i slovní označení typu zprávy

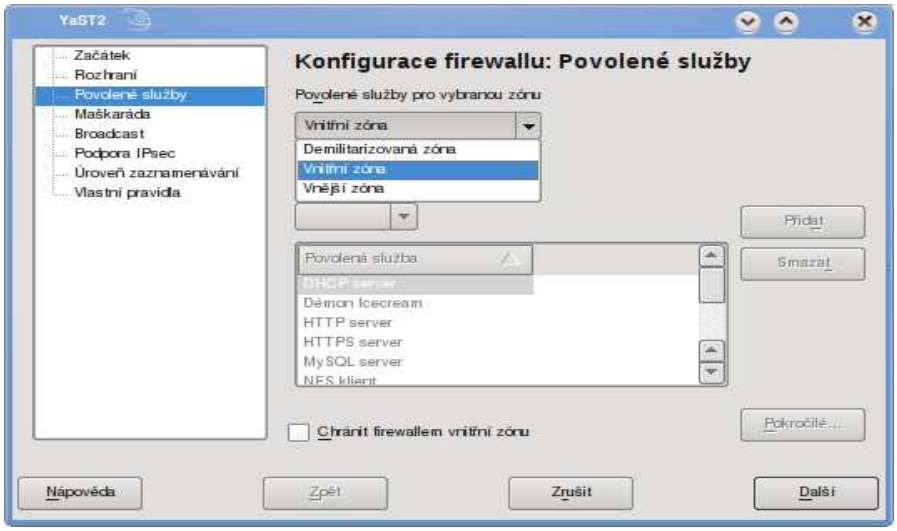
`/sbin/iptables -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT` povolili jsme příchozí ICMP pakety Time Exceeded (vypršení času při navazování spojení)

`/sbin/iptables -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT` povolili jsme příchozí ICMP pakety hlásící nedostupnost cíle

Firewall v Linuxu toho ve skutečnosti umí mnohem více, ale to by bylo nad rámec těchto skript (vpodstatě by se o něm dala napsat celá skripta). Z dalších funkcí můžeme jmenovat třeba podporu SPI (tj. filtrování podle stavu spojení, kdy vidíme na vyšší vrstvy ISO/OSI modelu), mechanismus NAT a MASQUERADE (překlad adres), označování paketů (značky mohou být zpracovány buď v jiných tabulkách firewallu, a nebo vně firewallu příkazem `ip rule` pro řízení směrování), a další.

Není vždy nutné provádět konfiguraci a její uložení v textovém režimu. Existuje dokonce několik různých grafických rozhraní k programu `iptables`. V desktopových distribucích je najdeme celkem běžně, případně je možné si některý z nich stáhnout z repozitáře a nainstalovat.

V unixových systémech založených na BSD (například OpenBSD) se často setkáme s firewallem *PacketFilter* (starší byl IPFilter), obslužný program je `pfctl`.



Obrázek 4.2: Firewall Netfilter – grafická nástavba v SUSE

V Linuxu se obvykle setkáme také s grafickým rozhraním firewallu, například na obrázku 4.2 je rozhraní pro konfiguraci firewallu v SUSE Linuxu.

Úkoly

1. Nastavte výchozí politiku pro odchozí provoz na akceptování.
2. Sestavte sadu pravidel, která z příchozího provozu zakážou všechno kromě provozu protokolu HTTP, POP a IMAP.
3. Na koncové stanici chceme povolit u příchozích paketů pouze ty, které jdou na TCP port 80 nebo TCP port 8080, všechny ostatní příchozí zakážeme. Jak budou vypadat pravidla pro příslušný chain?
4. Sestavte pravidla, která povolí ICMP pakety (jakékoliv, všechny typy ICMP zpráv) v příchozím, odchozím i průchozím provozu.

4.5 Počítání adres

Ukážeme si práci se zajímavým příkazem `ipcalc`, který podle zadané IP adresy podsítě vypíše masku, inverzní masku, rozsah adres uzlů v zadané podsíti, broadcast adresu a počet možných uzlů v síti.

Příklad 4.6

Příkaz chceme `ipcalc` spustit na stanici s nainstalovaným Ubuntu. Tato distribuce sice dotyčný nástroj ve výchozí instalaci neobsahuje, ale dokáže poradit, jakým způsobem máme nástroj nainstalovat:

```
sarka@notebook:~$ ipcalc 10.1.0.0/16

The program 'ipcalc' is currently not installed.  You can install it by typing:
sudo apt-get install ipcalc
```

Takže program není nainstalován. Ale byli jsme upozorněni, že se nachází v repozitáři (jsme připojeni k Internetu, máme přístup do repozitářů Ubuntu), máme před očima příkaz, kterým program nainstalujeme. Protože známe administrátorské heslo, není to problém (všimněte si příkazu `sudo`, který v Ubuntu a některých dalších distribucích slouží k získání vyšších přístupových oprávnění).

```
sarka@notebook:~$ sudo apt-get install ipcalc

[sudo] password for sarka: ***************
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  python-pylibacl python-brasero rdiff-backup python-pyxattr
Use 'apt-get autoremove' to remove them.
The following NEW packages will be installed:
  ipcalc
0 upgraded, 1 newly installed, 0 to remove and 378 not upgraded.
Need to get 26.6kB of archives.
After this operation, 131kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/ maverick/universe ipcalc all 0.41-2 [26.6kB]
Fetched 26.6kB in 0s (72.0kB/s)
Selecting previously deselected package ipcalc.
(Reading database ... 162465 files and directories currently installed.)
Unpacking ipcalc (from .../archives/ipcalc_0.41-2_all.deb) ...
Processing triggers for man-db ...
Setting up ipcalc (0.41-2) ...
```

Instalace je hotova, restartovat netřeba (nejsme přece ve Windows), tedy konečně můžeme spustit příkaz, jako parametr zadáme IP adresu (pod)sítě včetně délky prefixu:

```
sarka@notebook:~$ ipcalc 10.1.0.0/16

Address:   10.1.0.0             00001010.00000001. 00000000.00000000
Netmask:   255.255.0.0 = 16     11111111.11111111. 00000000.00000000
Wildcard:  0.0.255.255          00000000.00000000. 11111111.11111111
=>
Network:   10.1.0.0/16          00001010.00000001. 00000000.00000000
HostMin:   10.1.0.1             00001010.00000001. 00000000.00000001
```

```
HostMax:   10.1.255.254         00001010.00000001. 11111111.11111110
Broadcast: 10.1.255.255         00001010.00000001. 11111111.11111111
Hosts/Net: 65534                 Class A, Private Internet
```

Existují různé nástroje pro testování zabezpečení sítě, které nejsou součástí běžných distribucí, ale rozhodně stojí za to si je stáhnout a používat. Například *SATAN*[6] (System Administrators Tool for Analyzing Networks) prochází celou síť a provádí bezpečnostní testy. Je oblíbený jak u administrátorů, tak i u hackerů, i když každý z nich má samozřejmě jinou motivaci.

4.6 Instalace aplikací

Aplikace pro Linux jsou obvykle dostupné v *repozitářích*. Repozitář je (obvykle hierarchicky) uspořádaná databáze balíčků se softwarem. Může se nacházet na internetu (typicky oficiální repozitáře zvolené distribuce Linuxu) nebo například na výměnném paměťovém médiu (nebo kdekoliv jinde, kam máme přístup). Výhodou Linuxu oproti Windows je tedy existence jednoho nebo více centralizovaných úložišť, která jsou prověřená a především snadno a rychle dostupná.

Pro všechny nástroje pro instalaci balíčků (package) platí, že je třeba mít předem určeny repozitáře. Ty jsou automaticky během instalace nastaveny na instalační médium (obvykle DVD, ze kterého instalujeme Linux) a dále na jeden nebo více oficiálních repozitářů na webu naší linuxové distribuce. Z těchto repozitářů můžeme instalovat nový software, ale především jsou zde zajišťovány automatické aktualizace.

4.6.1 Binární a zdrojové balíčky

Postup instalace se liší podle toho, v jaké formě jsou instalační soubory. Pokud to jsou binární balíčky (RPM, DEB, apod.), použijeme příslušný nástroj (pro RPM je to program `rpm` nebo `urpmi`, pro DEB je to `dpkg` nebo `apt`, atd.), ve většině distribucí existují grafická rozhraní k těmto programům nebo přímo programy provádějící totéž, ale s grafickou nástavbou (*rpmDrake* v Mandrake, *xrpm* v RedHat, *YaST2* v SUSE, . . .). S některými jsme se seznámili v minulém semestru.

Příklad 4.7

V distribuci s RPM balíčky (tj. v distribucích odvozených z RedHat):

`rpm -i balicek.rpm` (případně další parametry, jsou v `man rpm`, nejčastěji se používá `-ivh`) – instalace zadaného (běžného binárního) rpm balíčku (můžeme používat i jednoduché regulární výrazy, především „`*`“)

`rpm -e balicek.rpm` odinstalace zadaného balíčku

`rpm -rebuild balicek.rpm` (případně další parametry) – instalace zdrojového rpm balíčku

`rpm -U balicek.rpm` aktualizace programu, pro který je balíček vytvořen

`rpm -i balicek.rpm` zobrazí informaci o balíčku

[6] Ke stažení na http://www.porcupine.org/satan/.

`rpm -qa` vypíše se seznam všech nainstalovaných balíčků (hodně dlouhý, doporučuje se ho nějak filtrovat, například zadáním dalších nepovinných parametrů nebo přes `grep`)

`rpm -l balicek.rpm` vypíše seznam souborů, které jsou v balíčku zahrnuty

`rpm -f soubor` zjistíme, ve kterém balíčku se nachází zadaný soubor

Z bezpečnostních důvodů je vhodné před samotnou instalací alespoň zkontrolovat digitální podpis a kontrolní součet balíčku (použije se pouze přepínač `--checksig`). Jde o to, aby v balíčku bylo právě to, co očekáváme a co tam dal autor, a ne třeba trojský kůň.[7]

Další nástroj v distribucích odvozených z RedHatu je `urpmi`:

`urpmi balicek.rpm` nainstaluje balíček

`urpme balicek.rpm` odstraní balíček (pozor, změna v názvu příkazu)

`urpmi.update -a` aktualizuje zdroje softwaru (repozitáře)

`urpmi --update --auto -auto-select` stáhne a nainstaluje aktualizace softwaru

Můžeme se také setkat s nástrojem `yum` (Fedora Core) nebo `yast` (SUSE).

Příklad 4.8

V distribucích s DEB balíčky (odvozených od distribuce Debian) se setkáme s nástrojem `dpkg` a především velmi oblíbeným `apt`. Podíváme se především na druhý z těchto nástrojů.

V distribucích odvozených z Debianu jsou adresy zdrojů (tedy repozitářů) uloženy v textovém souboru `/etc/apt/sources.list`. Pokud chceme přidat nový repozitář, můžeme to provést buď pomocí příslušných nástrojů pro práci s balíčky (včetně grafických) nebo přidat na konec tohoto souboru nový záznam.

Příkazy pro práci s balíčky a repozitáři:

`apt-get install balicek.deb` instalace balíčku

`apt-get -s install balicek.deb` pouze simuluje instalaci, takto zjistíme, jestli by během instalace nemohl nastat nějaký problém

`apt-get remove balicek.deb` odstranění balíčku

`apt-get update` update zdrojů softwaru (aktualizace souboru `/etc/apt/sources.list`)

`apt-get -u upgrade` aktualizace nainstalovaných balíčků na nejnovější verze (předem je třeba provést update předchozím uvedeným příkazem), přepínač `-u` sice není nutný, ale doporučuje se (budeme informováni podrobně o všech balíčcích, kterých se aktualizace týká, jinak by nebyly vypsány)

`apt-get -u dist-upgrade` aktualizace celé distribuce (význam přepínače je stejný jako u předchozího příkazu)

[7] Trojský kůň je program fungující podobně jako virus – program budící dojem užitečnosti, ale jeho kód byl zaměněn za jiný, poněkud méně „užitečný“.

`apt-get check` zkontroluje databázi nainstalovaných balíčků a zjistí případné nesrovnalosti (poškozenou instalaci, problémy v závislostech mezi balíčky apod.), nic neinstaluje, pouze zkontroluje

`apt-cdrom add` pokud máme v mechanice instalační DVD (nebo CD), tímto příkazem bude automaticky přidáno ke zdrojům do souboru `/etc/apt/sources.list`

`apt-cache search gcompris` hledáme názvy balíčků, které souvisejí s programem `gcompris`

`apt-cache show gcompris` chceme podrobnější informace o balíčku s názvem `gcompris` (zde zadáváme přímo název balíčku, v předchozím příkazu šlo o výraz k vyhledávání)

`apt-cache depends gcompris` zjistíme závislosti zadaného balíčku

Příklad 4.9

V souboru `/etc/apt/sources.list` najdeme (kromě komentářů) řádky ve formě

`deb` *`uri distribuce komponenty`*
`deb-src` *`uri distribuce komponenty`*

Řádek obvykle začíná slovem `deb` nebo `deb-src` (podle toho, zda jde o binární nebo zdrojové balíčky), následuje adresa repozitáře (může to být server na internetu nebo třeba paměťové médium, ze kterého jsme instalovali systém), název distribuce (nemusí být konkrétní název) a pak bližší určení větve v repozitáři (alespoň „main“ pro hlavní větev, často se setkáme se „stable“ nebo „stable/updates“).

Například takto může vypadat tento soubor v distribuci Debian, verze Etch:

```
# Debian GNU/Linux Stable - security updates
deb     http://security.debian.org/ stable/updates main contrib non-free
deb-src http://security.debian.org/ stable/updates main contrib non-free

# Debian GNU/Linux Stable - packages
deb     ftp://ftp.linux.cz/pub/linux/debian stable main contrib non-free
deb-src ftp://ftp.linux.cz/pub/linux/debian stable main contrib non-free
deb     http://ftp.cz.debian.org/debian/ stable main contrib non-free
deb-src http://ftp.cz.debian.org/debian/ stable main contrib non-free
#deb     http://http.us.debian.org/debian stable main contrib non-free
#deb-src http://http.us.debian.org/debian stable main contrib non-free

deb ftp://ftp.de.debian.org/debian etch main
deb http://http.us.debian.org/debian/ sarge main

# Debian GNU/Linux - TeX Live
deb     http://www.tug.org/texlive/Debian/ pool/
deb-src http://www.tug.org/texlive/Debian/ pool/
deb     http://www.tug.org/texlive/Debian/ updpkg/
deb-src http://www.tug.org/texlive/Debian/ updpkg/

# Instalační média (DVD 1-3), vše je zakomentováno:
# deb  cdrom:[Debian GNU/Linux 4.0 r3 _Etch_ - Official i386 DVD Binary-3
       20080217-11:31]/ etch main
# deb  cdrom:[Debian GNU/Linux 4.0 r3 _Etch_ - Official i386 DVD Binary-1
```

```
       20080217-11:31]/ etch contrib main
# deb  cdrom:[Debian GNU/Linux 4.0 r3 _Etch_ - Official i386 DVD Binary-2
       20080217-11:31]/ etch main
```

Tento soubor lze přímo editovat (s právy roota), ale jen tehdy, pokud právě neběží žádný program pracující s balíčky (například `apt-get` nebo některý z nástrojů s grafickým rozhraním, třeba *Adept*).

Po každé změně je třeba aktualizovat vnitřní databázi mechanismu *apt*, například příkazem `apt-get update`.

Další informace najdeme například na

- http://www.debian.org/doc/manuals/apt-howto/index.cs.html
- http://www.tfug.org/helpdesk/linux/rpm.html

Úkoly

Ujasněte si, který typ balíčků používá vaše distribuce. Podle typu balíčkovacího systému vyzkoušejte alespoň příkazy pro zjišťování informací o balíčcích a repozitářích.

4.6.2 Instalace ze zdrojových kódů

Protože jsou tyto instalované programy dostupné v archivech (gzip, tar, tar.gz, .tgz, .Z apod.), prvním krokem je samozřejmě rozbalení archivu do nějakého vhodného adresáře. Dalším krokem je hledání souboru *README* mezi rozbalenými soubory a případně skriptu *Install*, v těchto souborech bývá postup instalace podrobně popsán. Je zřejmé, že když budeme překládat ze zdrojových kódů, musíme mít nainstalován překladač, což je obvykle `gcc`.

Dále je potřeba upravit konfiguraci instalace (tj. podle našeho systému a případně hardwarové architektury nastavit parametry překladu), obvykle pomocí skriptu *Install* nebo *configure* (to je běžnější). Tento skript pak může mít parametry (ty zjistíme výše popsaným způsobem, ze souborů dodávaných se zdrojovým kódem, jako parametry se používají často volby, které mají řídit postup instalace – co se má instalovat, co ne, příp. konfigurace po nainstalování).

Abychom zajistili, že se spustí skript v pracovním adresáři a ne jiný stejně pojmenovaný (předpokládejme, že pracovním adresářem je adresář se zdrojovými soubory), použijeme syntaxi
`./configure`

Pokud se jedná o aplikaci využívající grafické rozhraní, bude konfigurace instalace trochu jiná. Mezi zdrojovými soubory může být například skript `Imake`, pak se pro překlad místo `make` použije příkaz `xmkmf`.

Instalace se provádí obvykle pomocí programu `make`. Tento program používá skript `Makefile`, který spustí příslušný překladač a ze zdrojových souborů vytvoří binární. Někdy je potřeba provést také příkaz `make` s dalšími parametry, například

- `make install` zkopíruje vytvořené binární soubory na správná místa v adresářové struktuře,

- `make clean` odstraní nepotřebné objektové soubory,
- `make -p` (před vlastní instalací) spustí pouze simulaci překladu, na výpisech programu můžeme sledovat, co vlastně bude provedeno.

Příklad 4.10

Ve většině případů stačí posloupnost akcí (nacházíme se v adresáři se zdrojovými kódy):

```
./configure
make
sudo make install
```

Příkaz `make install` je nutné spustit s vyššími přístupovými oprávněními. Zde jsme použili příkaz `sudo`, ale ve skutečnosti se samozřejmě budeme řídit tím, co používáme v našem systému (třeba `su`).

Instalovat můžeme buď do sdílených částí (adresář `/usr` nebo lépe `/opt`), ale pokud k instalované aplikaci má mít přístup pouze jeden konkrétní uživatel, je vhodné instalovat do jeho domovského adresáře (v jeho domovském adresáři vytvoříme adresář `~/bin` nebo podobně, a tam instalujeme).

4.7 Bezpečnost systému

Linux je obecně považován za stabilní a bezpečný systém. Někdy se objeví bezpečnostní mezera, ale narozdíl od Windows se reakce v podobě záplaty objevuje velmi rychle (často během 24 hodin). Pro bezpečnost má velký význam i to, že uživatelé (a především správci) Linuxu jsou většinou „znalí problematiky“ a nezanedbávají bezpečnostní hlediska – používají různé bezpečnostní mechanismy, pravidelně zjišťují, zda je třeba systém aktualizovat, udržují spuštěné nástroje pro zajištění bezpečnosti na síti (firewall, apod.).

Zatímco dříve se hesla uživatelů ukládala spolu s ostatními informacemi o uživatelích v zašifrované podobě (šifrovaná pouze hesla) v souboru `/etc/passwd`, dnes se prosazuje ukládání hesel do zvláštního souboru, `/etc/shadow`, taková „odstínovaná“ hesla se nazývají *stínová hesla* (původní soubor `/etc/passwd` už není zašifrovaný, ale běžný uživatel ho neotevře).

Příručka správce zabezpečení Linuxu je na http://www.securityportal.com/lasg.

SELinux je projekt NSA (Národní agentura pro bezpečnost USA). Pokud linuxová distribuce obsahuje podporu SELinuxu (což je dnes celkem běžné), v jádře existuje modul, který kontroluje přístupová práva ke všem operacím v systému u uživatelů, procesů a dalších objektů. Omezuje přístupová práva především procesům spuštěným ze sítě.

Viry nejsou na Linuxech příliš rozšířené. Je to z mnoha důvodů – „pravé“ viry jsou vlastně úseky programu, musí tedy být většinou (ne vždy!) psány pro určitý operační systém, a tím je většinou systém Windows (unixové systémy jsou pořád ještě méně rozšířené než Windows, takže efekt by nebyl zase až tak veliký).

Ostatní druhy škodlivých kódů fungující na trochu jiném principu (skripty, případně viry přímo psané pro unixové systémy) sice teoreticky mají šanci, ale hradbou je zase obvykle vyšší zabezpečenost unixových systémů, tedy systém přístupových práv (běžný uživatel nemůže zasahovat do

jádra ani instalovat a spouštět programy, které do dokážou), firewall s routerem a další mechanismy, takže škodlivý kód může do systému proniknout většinou jen vinou administrátora, který zanedbal bezpečnostní opatření.

Obdobné možnosti mají vlastně i systémy Windows s jádrem NT, i když možná ne tak propracovaná (unixové systémy jsou mnohem starší, programátoři měli víc času a zabezpečení bylo primární potřebou), zde se také projevuje především lidský faktor. Unixové systémy včetně Linuxu si navíc instalují především uživatelé „znalejší “, kteří vědí, že podceňovat bezpečnost se nevyplácí.

Důležitá je také otázka bezpečnostních chyb. V tisku (a hlavně na Internetu) se objevují často hlášky typu „byla objevena bezpečnostní díra X v operačním systému Y, která může způsobit Z“. Bezpečnostní díry se objevují v každém operačním systému (a vlastně v jakémkoliv rozsáhlejším a složitějším projektu, tomu se nedá vyhnout). Proto je důležité „záplatovat, záplatovat a záplatovat“, at' se jedná o jakýkoliv systém. Bezpečnostní záplaty jsou dostupné na Internetu také pro Linuxové distribuce (je možné mít zapnuté automatické aktualizace, a to *pro všechny aplikace*, nejen systém).

I v Linuxu můžeme používat antivirové programy. Některé firmy známé svými produkty pro Windows dnes vydávají také verze pro Linux, například AVG, Kaspersky Anti-Virus for Samba nebo Nod32, to jsou však komerční produkty. Jejich výhodou je propracovaný servis ze strany firmy. Existuje však antivir, který je už od svých začátků volně šiřitelný a přímo určený pro unixové systémy – ClamAV (jeho nástavba s grafickým rozhraním pro KDE je KlamAV), nabízí jak antivirový scaner, tak i „rezidentní“ ochranu (je spuštěn démon) a ověřování pošty, můžeme aktivovat také automatické aktualizace produktu. Zajímavý je také projekt SpamAssassin, který také má verzi pro Linux.

Objevují se také nástroje, které mají zvýšit bezpečnost existujících distribucí. Zde můžeme zařadit nejen SELinux, ale třeba také *Bastille Linux*. Bastille Linux je nástroj, který uživateli klade otázky a na základě jeho odpovědí mění konfiguraci systému ke kompromisu mezi bezpečností systému a uživatelskou přívětivostí.

Nastavení se týkají povolování a zakazování některých funkcí, které by mohly být případně zneužity k průniku do systému, zakazování běhu některých funkcí jako démonů, ukončování některých nepotřebných serverů, zabezpečování heslem, konfigurace firewallu (tady pozor při konfiguraci přes síť, Bastille by takto mohl náhle přerušit svou činnost a počítač by zůstal nedostupný). Musíme být samozřejmě přihlášeni s dostatečnými právy. Projekt je dostupný zatím pro některé nejznámější distribuce a jejich klony na stránkách http://freshmeat.net.

Záchranný systém má podobný význam jako nouzový režim nebo záchranná konzola ve Windows 2000/XP. V různých distribucích ho lze spustit různě, je to jedna z věcí, kterou by si měl každý předem nastudovat v dokumentaci. Jednou z možností je použití parametru *emergency* při startu jádra.

Záchranný režim znamená úroveň běhu pouze jednouživatelskou, a dále většinou nejsou připojeny žádné oddíly kromě kořenového, tedy v případě potřeby je musíme připojit ručně. Taktéž se obvykle musíme obejít bez grafického rozhraní (ale lze je spustit).

4.8 Další nástroje pro správu

Výborný nástroj pro správu systému je *Webmin*. Je ke stažení na http://www.webmin.com a pro jeho provoz stačí prakticky jen internetový prohlížeč (třeba Firefox nebo Mozilla) a samozřejmě nainstalovaného a spuštěného démona webmin. Pracujeme v grafickém rozhraní (co jiného v internetovém prohlížeči), ke konfiguraci lokálního počítače (toho, na kterém pracujeme) se dostaneme přes `https://localhost:10000`, k jiným počítačům v síti přes jejich adresy. Můžeme konfigurovat i to, co není v Ovládacím centru včetně různých serverů (Samba, Apache, . . .).

Na adrese http://www.linuxprinting.org je možné najít řešení různých problémů při tisku v Linuxu. Položka Printer Listing pro námi zadanou tiskárnu zobrazí úroveň podpory v Linuxu (do jaké míry funguje), dále zde najdeme i ovladače pro různé druhy tiskáren.

Pro bezpečnost sítě jsou užitečné nástroje Nessus (http://www.nessus.org, vyhledává bezpečnostní díry v počítačové síti), Nmap (http://www.nmap.org, kontroluje síťová zařízení přes IP protokol), Snort (http://www.snort.org, monitoruje podezřelou činnost na síti, detekuje možné průniky do sítě), atd. Takových nástrojů je hodně a s některými se seznámíme ve čtvrtém ročníku v předmětu Počítačové sítě a decentralizované systémy.

Literatura

Existuje mnoho zdrojů, které může správce systému nebo sítě použít pro další sebevzdělávání (a sebevzdělávání je v této době a tomto oboru nezbytně nutné). Následují některé ze zdrojů, které mohou být pro tento účel použitelné.

Informační portály o Linuxu a internetové časopisy:

[1] http://www.tldp.org

[2] http://www.linux.cz

[3] http://docs.linux.cz

[4] http://www.penguin.cz

[5] http://www.linuxworld.cz

[6] http://www.linuxzone.cz

[7] http://www.linuxsoft.cz

[8] http://www.abclinuxu.cz

[9] http://www.root.cz

[10] http://www.linuxexpress.cz

[11] http://www.lpmagazine.org

Články a učebnice:

[12] *Učebnice GNU/Linuxu* [online]. ABCLinuxu.cz.
URL: http://www.abclinuxu.cz/ucebnice/obsah

[13] COOPER, M. *Advanced Bash-Scripting Guide* [online].
URL: http://www.tldp.org/LDP/abs/

[14] PATOČKA, M. *Porovnání systémů Linux a FreeBSD* (seriál) [online].
URL: http://www.root.cz/clanky/porovnani-linux-freebsd/

[15] STUTZ, M. *The Linux Cookbook: Tips and Techniques for Everyday Use* [online].
URL: http://www.dsl.org/cookbook/cookbook_toc.html

[16] BROWN, M. A. *Guide to IP Layer Network Administration with Linux* [online].
URL: http://linux-ip.net/html/

[17] BRANDEJS, M. *Výukové materiály k předmětu Unix* [online].
URL: http://www.fi.muni.cz/usr/brandejs/unix

[18] BRANDEJS, M. *Starší výukové materiály k předmětu Unix* [online].
URL: http://www.fi.muni.cz/usr/brandejs/unix_old/pomucky/pomucky-unix.html

[19] KERŠLÁGER, M. *Linux* [online].
URL: http://www.pslib.cz/ke/Linux

[20] ŽEMBER, M. *Exploity* [online].
URL: http://www.ms.mff.cuni.cz/~zembm2am/exploity/exploity.html

[21] HÄRING, D. *Nebojte se Linuxu: plánování, hierarchie a řízení procesů I* [online].
URL: http://www.linuxzone.cz/index.phtml?idc=252&ids=29

[22] KOVÁŘ, D. *Proč máme `/proc`?* [online]
URL: http://www.linuxexpres.cz/praxe/proc-mame-proc

[23] KAŠPÁREK, T. *Co před námi tají `/proc`* [online].
URL: http://www.root.cz/clanky/co-pred-nami-taji-proc/

[24] GAGYI, M. *`Sysfs` – brána do jádra* [online].
URL: http://www.abclinuxu.cz/clanky/system/sysfs-brana-do-jadra

[25] BRADY, P. *Linux Commands – a Practical Reference* [online].
URL: http://www.pixelbeat.org/cmdline.html

[26] MILAR, B. *Seriál o BASHi* [online].
URL: http://www.linuxexpres.cz/praxe/serial-o-bashi

[27] *Introduction to BASH* [online]. GNU.org.
URL: http://www.gnu.org/software/bash

[28] FUCHS, J. BASH (seriál) [online].
URL: http://www.abclinuxu.cz/clanky/show/46130

[29] MATYS, J. *Programování pod Linuxem pro všechny* [online].
URL: http://www.root.cz/serialy/programovani-pod-linuxem-pro-vsechny/

Pro hlubší studium:

[30] LASSER, J. *Rozumíme Unixu*. Praha, Computer Press, 2002.

[31] JELÍNEK, L. *Jádro systému Linux: kompletní průvodce programátora*. Brno, Computer Press, 2008.

[32] LOCKHART, E. *Bezpečnost sítí na maximum*. Brno, Computer Press, 2005.

[33] HATCH, B. – LEE, J. – KURTZ, G. *Hacking bez tajemství: Linux*. Brno, Computer Press, 2003.

[34] RAYMOND, E. S. *Umění programování v Unixu*. Brno, Computer Press, 2004.

[35] SMITH, R. W. *Linux ve světě Windows*. Praha, Grada Publishing, 2006. Některé strany dostupné na:
http://books.google.cz/books?id=uDO_I1kIRBYC&printsec=frontcover

[36] VYCHODIL, V. *Operační systém Linux: Příručka českého uživatele*. Brno, Computer Press, 2003. Ukázková kapitola dostupná na:
http://vychodil.inf.upol.cz/errata/download/linux-pcu-kap5.pdf

[37] *The Linux Foundation – publications* [online]. Seznam odkazů na publikace o Linuxu.
URL: http://www.linuxfoundation.org/collaborate/publications

[38] PRODĚLAL, J. *Jádro BSD* [online].
URL: http://www.fi.muni.cz/~kas/p090/referaty/2003-podzim/skupina16/kernel_xprodel.html

[39] *FreeBSD Handbook* [online].
URL: http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/

[40] PECHANEC, J. *Programování v Unixu* (v jazyce C) [online].
URL: http://www.devnull.cz/mff/pvu/