



**SLEZSKÁ
UNIVERZITA**

FILOZOFICKO-
PŘÍRODOVĚDECKÁ
FAKULTA V OPAVĚ

Šárka Vavrečková

Operační systémy

II

Linux

Slezská univerzita v Opavě
Filozoficko-přírodovědecká fakulta
Ústav informatiky

Opava, poslední aktualizace 17. května 2024

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 *Operační systémy I*. Předpokládá se základní orientace v adresářové struktuře a textovém režimu UNIXových systémů, nejdůležitějších konfiguračních souborech, základy přístupových oprávnění v UNIXových systémech.

Operační systémy II

Linux

RNDr. Šárka Vavrečková, Ph.D.

Dostupné na: <http://vavreckova.zam.slu.cz/opsys.html>

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

Sázeno v systému L^AT_EX

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 II 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 v druhé části semestru.

Navazujeme na obdobná skripta z předmětu Operační systémy I, tedy předpokládají se již základní znalosti práce v UNIXových systémech, v přístupových oprávněních a základ práce v textovém režimu.

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*, značení apod. jsou označeny modrým symbolem, který vidíme zde vlevo. Tuto ikonu (stejně jako následující) najdeme na začátku odstavce, ve kterém je nový pojem zaváděn.
-  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.
-  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.
-  Žlutou ikonou jsou označeny odkazy, na kterých lze získat *další informace* o tématu. Nejčastěji

u této ikony najdeme webové odkazy na stránky, kde se dané tématice jejich autoři věnují podrobněji.

-  Červená je ikona pro *upozornění* a poznámky.

Pokud je množství textu patřícího k určité ikoně větší, je celý blok ohraničen prostředím s ikonami na začátku i konci, například pro definování nového pojmu:



Definice

V takovém prostředí definujeme pojem či vysvětlujeme sice relativně známý, ale komplexní pojem s více významy či vlastnostmi.



Podobně může vypadat prostředí pro delší postup nebo delší poznámku či více odkazů na další informace. Mohou být použita také jiná prostředí:



Příklad

Takto vypadá prostředí s příkladem, obvykle nějakého postupu. Příklady jsou obvykle komentovány, aby byl jasný postup jejich řešení.



Úkol

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.



Pro příkazy a jejich výstupy je používáno neproporcionální písmo, výstupy jsou sázeny o něco menším fontem, aby tolik nesplývaly se zadaným příkazem.

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

Předmluva	iii
1 Úvod do správy v textovém režimu	1
1.1 Textové shelly v UNIX-like systémech	1
1.2 Nápověda	2
1.3 Adresáře a soubory	3
1.3.1 Možnosti vytvoření nového souboru	3
1.3.2 Filtrování souborů	4
1.3.3 Prohledávání adresářové struktury	5
1.3.4 Prohledávání textů	9
1.4 Speciální znaky pro uvozování	12
1.5 Proměnné	14
1.5.1 Opakování práce s proměnnými	14
1.5.2 Výpočty	17
1.6 Skripty	19
1.6.1 Co je to skript	19
1.6.2 Parametry a návratové hodnoty	20
1.6.3 Další možnosti použití skriptů	22
1.7 Podmínky, větvení a cykly	22
1.7.1 Jednoduché propojení příkazů	22
1.7.2 Příkazy větvení	24
1.7.3 Cyklus přes množinu	28
1.7.4 Cykly pro rozsah hodnot	31
1.8 Překlad programů	32
1.9 Konverze textových souborů	33
2 Správa procesů	34
2.1 Příkazy pro práci s procesy	34
2.2 Skupiny a relace procesů	37

2.3	Úlohy a multitasking	39
2.3.1	Úlohy na popředí a pozadí	40
2.3.2	Úlohy a procesy	42
2.4	Komunikace procesů	44
2.5	Plánování spouštění procesů	46
3	Řízení přístupu a správa uživatelů	50
3.1	Uživatelé a skupiny	50
3.1.1	Příkazy pro zjištění informací o uživateli	50
3.1.2	Soubory a adresáře související s uživateli	55
3.1.3	Práce s uživatelskými skupinami	57
3.1.4	Práce s uživatelskými účty	60
3.1.5	Vlastnosti hesla	63
3.2	Přístupová oprávnění k souborům	65
3.2.1	Vlastník a přidružená skupina	65
3.2.2	Změna přístupových oprávnění souboru	66
3.2.3	Maska oprávnění	69
3.3	Práce pod jiným uživatelským účtem	70
3.3.1	Příkaz <code>su</code>	70
3.3.2	Příkaz <code>sudo</code>	72
3.4	Pokročilé mechanismy řízení přístupu	76
3.4.1	Atributy	76
3.4.2	POSIX ACL	77
3.4.3	PAM	79
3.4.4	Capabilities	80
3.4.5	Chráněné prostředí pro běh procesu	80
3.5	Příkaz <code>stat</code>	82
4	Správa zařízení	83
4.1	Paměťová média	83
4.1.1	Příkazy pro paměťová média	83
4.1.2	Soubor <code>fstab</code>	85
4.1.3	Připojování a odpojování pamětí	88
4.1.4	Program <code>fdisk</code>	90
4.1.5	Hlavní a vedlejší číslo zařízení, <code>udev</code>	91
4.2	Operační paměť	92
4.3	Procesor	92
4.4	Informace o architektuře a základní desce	93
4.4.1	Základní deska a přehled komponent	93
4.4.2	Sběrnice PCI, PCIe, USB	94
4.5	Moduly jádra	96

5	Správa sítě	98
5.1	Soubory	98
5.2	Několik základních příkazů	99
5.3	Starší příkazy pro práci se síťovým rozhraním	100
5.4	Překlady a testování	101
5.4.1	Mechanismus DNS	101
5.4.2	Testování dostupnosti a výpis statistik	102
5.5	Mechanismus <code>iproute2</code> pro práci se sítí	104
5.5.1	Konfigurace síťového rozhraní a adres	104
5.5.2	Směrování a filtrování	105
5.5.3	Objevování sousedů	107
5.5.4	Tunely	108
5.6	Ovlivňování runtime informací o síti	109
5.7	Počítání adres	112
6	Nasazení systému	114
6.1	Běh systému	114
6.1.1	Start systému a proces <code>init</code>	114
6.1.2	Informace o startu a běhu systému	116
6.1.3	Úrovně běhu	117
6.1.4	Ukončení běhu systému	118
6.2	Inicializační systémy a služby	118
6.2.1	Služby podle <code>sysvinit</code>	119
6.2.2	Služby podle <code>upstart</code>	122
6.2.3	Služby podle <code>systemd</code>	124
6.3	Instalace aplikací	126
6.3.1	Binární a zdrojové balíčky	126
6.3.2	Instalace ze zdrojových kódů	130
6.3.3	Aktualizace balíčků	130
6.3.4	Přechod na novou verzi distribuce	131
6.4	Log soubory	132
6.5	Logování provozu	134
6.5.1	Vstup syslogu	134
6.5.2	Filtrování vstupu	135
6.5.3	Výstup syslogu	136
6.6	Firewall	137
6.6.1	Ovládání firewallu	137
6.6.2	Struktura firewallu	138
6.6.3	Základní vnitřní příkazy a parametry pro filtrování	140
6.7	Bezpečnost systému a další nástroje	144

Úvod do správy v textovém režimu

Navazujeme na učivo z předchozího semestru. Kdo měl problémy se základy textového režimu v shellu `bash`, ten si urychleně zopakujte příslušnou kapitolu ze skript předmětu *Operační systémy I*.

1.1 Textové shelly v UNIX-like systémech

 Původním shellem 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`,
- *Korn Shell* (`ksh`) – syntaxe příkazů odpovídá Bourne Shellu, ale jinak většinu vlastností přejal z `tcsh`, jde vlastně o hybrid shellů `sh` a `tcsh`,
- *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),
- *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 osekáný (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).

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



Příklad

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 ,

`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,

`chsh -s /bin/bash` nastavili jsme zpět shell *bash*.



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.

 Také v UNIXových shellech existují buď vnitřní (vestavěné) příkazy, které jsou součástí příslušného shellu, nebo vnější (externí) příkazy reprezentované spustitelným souborem, jako další typ specifický pro UNIXové systémy můžeme uvést aliasy.

 Příkaz `type` nám řekne, o jaký typ příkazu se jedná – vnitřní nebo vnější. Jako parametr jednoduše zadáme příslušný příkaz.



Příklad

Zadáme tyto příkazy:

<code>type cd</code>	<code>type echo</code>	<code>type passwd</code>
<code>type find</code>	<code>type ls</code>	<code>type ping</code>

Některé jsou vnitřní příkazy shellu `bash`, další jsou buď vnější příkazy nebo aliasy (jak víme, aliasy se někdy používají pro nastavení výchozích parametrů příkazů, často se s tím setkáváme právě u `ls`).



Úkoly

1. Zobrazte si obsah souboru `/etc/shells`.
2. Vyzkoušejte příkaz `type` podle předchozího příkladu, pak dosazujte jiné příkazy, na které si vzpomene.



1.2 Nápověda

Zopakujeme si, jak se dostat k nápovědě. Nápovědu k příkazům můžeme získat více způsoby:

- zobrazením manuálové stránky příkazu, a to příkazem `man`, mohou být k dispozici i `info` stránky,
- někdy je implementován příkaz `apropos`, který použijeme, když nevíme, jak se příkaz nazývá,

- příkaz `what is` použijeme, když jsme narazili na příkaz (spustitelný soubor), ale nevíme, co provádí,
- 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,
- jsou k nalezení *dokumenty FAQ* (Frequently Asked Questions) pro často pokládané otázky,
- v internetovém prohlížeči (přesněji – zeptáme se Googlu) se také dostaneme na manuálové stránky, například zadáme `man chsh` do adresního řádku.



Další informace:

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

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

Získat nápovědu bychom měli umět už z předmětu Operační systémy I.



1.3 Adresáře a soubory

1.3.1 Možnosti vytvoření nového souboru

 V předchozím semestru jsme se setkali s několika možnostmi vytvoření nového souboru, některé z nich dovolují zároveň do souboru něco uložit. Máme tyto možnosti:

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

Poslední uvedený způsob funguje i u Windows (použití příkazu `type`, kterému na vstup směřujeme vstup z konzoly `CON`). V Linuxu (na rozdíl od windowsovského `type`) příkaz `cat` dokáže pracovat i interaktivně, proto nemusíme explicitně zadávat vstup (konzolu, zde třeba `/dev/tty`).



Příklad

V pracovním adresáři vytvoříme nový soubor s názvem `soubor.txt`:

```
cat > soubor.txt Enter

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

Ctrl+D, pak Enter
```

Klávesová kombinace `Ctrl+D` je důležitá a v tomto případě bychom si ji měli pamatovat, vytvoří textový znak znamenající konec souboru.



 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. Použití si ukážeme na příkladu:



Příklad

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

```
cat > soubor.txt << KONEC   
  
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. Řetězec KONEC nebude součástí výsledného souboru.



Úkol

Vyzkoušejte si vytvoření nového souboru v textovém režimu pomocí příkazu `cat`, včetně využití mechanismu *here document*.



1.3.2 Filtrování souborů

 Pro jednoduché filtrování seznamu souborů se dá použít příkaz `ls` nebo `echo` (obojí jsme probírali v předchozím semestru), přičemž používáme zástupné symboly (globing characters) – je to zjednodušená obdoba regulárních výrazů:

- hvězdičku (jakýkoliv řetězec jakýchkoliv symbolů), například `a*` znamená, že hledáme řetězec, kde první písmeno je `a` a následuje jakýkoliv počet čehokoliv dalšího,
- otazník (jeden jakýkoliv symbol), například `m???` znamená, že hledáme čtyřpísmenná slova, která začínají písmenem `m`,
- hranaté závorky (zadáváme množinu povolených symbolů, přičemž jeden z nich je na daném místě použit), např. `[a-z]` určí, že na daném místě může být jakékoliv malé písmeno, kdežto schéma `[a-z]*` znamená řetězec začínající (jakýmkoliv) malým písmenem,
- hranaté závorky s vykřičníkem určují negaci obsahu množiny – např. `[!a-z]` znamená, že na tomto místě může být cokoliv kromě malého písmene, nebo `ab[!0-9]*` určuje, že hledáme řetězec se třemi znaky, z nichž první dva jsou písmena `ab`, následuje jeden znak, který není číslicí a za ním je cokoliv (takže schématu by vyhovoval například řetězec `abcxyz21`).

 Tyto zástupné symboly jsou vyhodnocovány textovým shellem (např. `bash`), nikoliv příkazem, v jehož parametru se použijí. Příkazu bude naservírován až zpracovaný výsledek.



Příklad

Předpokládejme, že máme v pracovním adresáři soubory

`abc.txt`, `amx.jpg`, `bcd.gif`, `abdca.txt` a adresáře `alibaba`, `Mercedes`

a zadáme `ls a*`, ve skutečnosti se postupně provede tento příkaz:

```
ls abc.txt amx.jpg abdca.txt alibaba
```

přičemž u souborů se prostě vypíše jejich název a u adresáře se příkaz pokusí vypsat jeho obsah.

K samotnému příkazu `ls` se hvězdička vůbec nedostane.

Pokud bychom v tomto případě zadali příkaz `echo a*`, shell by ho „přeložil“ na

```
echo abc.txt amx.jpg abdca.txt alibaba
```

takže důsledkem by prostě bylo vypsaní těchto řetězců hned za sebou. Ovšem pro příkaz `echo` nemáme k dispozici další možnosti formátování výstupu ani rekurze.

Kdybychom chtěli v našem případě vypsat všechny položky, které začínají písmenem `b` nebo `M`, napsali bychom:

```
echo [bM]*
```

Bash tento příkaz „přeloží“ na

```
echo bcd.gif Mercedes
```

tedy budou na výstupu právě tyto dva řetězce.



Úkol

Zopakujte si toto učivo z předchozího semestru a procvičte si filtrování na názvech souborů a adresářů ve vašem domovském adresáři. Například vypište seznam všech položek, které začínají písmenem `P`, začínají tečkou, za kterou následuje písmeno `g`. Dále vypište všechny položky s názvem končícím písmenem `p`.



1.3.3 Prohledávání adresářové struktury

K prohledávání adresářové struktury (hledání souborů) používáme příkazy `find` a `locate`, k prohledávání obsahu souborů zase `grep` (viz dále). 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 příkazy `whereis` a `which`.



Zde se podíváme na použití příkazu `find`. Hledáme soubory (a adresáře) se zadaným názvem nebo jinými parametry, příkaz také poskytuje možnost omezeného zpracování souborů nebo ovlivňování formátu vlastního výstupu. Jako první parametr se obvykle zadává místo, od kterého má začít vyhledávání. Pokud tento parametr nezadáme, dosadí se pracovní adresář. Můžeme také používat zástupné symboly (globes) – hvězdičku apod., a dokonce máme dvě možnosti:

- buď necháme zástupné symboly interpretovat shellem (podobně jako v předchozí sekci),
- nebo jejich interpretaci svěříme samotnému příkazu `find` (protože ten to na rozdíl od mnoha jiných příkazů umí).

Jestliže chceme použít druhou možnost, uzavřeme vyhledávací výraz do jednoduchých apostrofů (pozor – ne opačných), čímž zamezíme shellu v interpretaci zástupných symbolů.



Příklad

Projdeme si možnosti příkazu `find` postupně podle jednotlivých kritérií. Nejdřív se zaměříme na samotný hledaný řetězec, jeho formát a vypnutí rozlišování malých a velkých písmen.

`find *.txt -print` vypíše všechny soubory se zadanou příponou v pracovním adresáři, přičemž interpretaci zástupných symbolů necháváme na shellu, přepínač `-print` se dává celkem běžně, ale ve skutečnosti není až tak nutný (způsobí výpis celého názvu nalezeného souboru, což je ale výchozí chování, pokud nepoužijeme modifikátory výstupu)

`find -name '*.txt'` vypíše všechny soubory se zadanou příponou rekurzivně; jak vidíme, hvězdička je přenechána příkazu `find`, nikoliv shellu (apostrofy jsou nutné!)

`find /usr/xyz -name abcd` hledá soubor, který má v názvu řetězec `abcd`, v adresáři `/usr/xyz`, přepínač `-print` způsobí výpis úplného jména; rozlišují se malá a velká písmena (takže třeba pokud by v tom adresáři byl soubor, v jehož názvu je řetězec „AbCD“, nebyl by vypsán v nalezených)

`find ~ -iname 'desk*'` vypíše s rekurzí vše, co začíná řetězcem „desk“, bez rozlišování malých a velkých písmen (protože jsme použili parametr `-iname` místo `-name`)



Příkaz `find` umí výborně filtrovat podle různých kritérií, některá z nich si zde ukážeme (další najdete v manuálové stránce tohoto příkazu: `man find`).



Příklad

Můžeme vyhledávat například podle vlastníka či přidružené skupiny souboru (jak víme, každý soubor má svého vlastníka a přidruženou skupinu):

`find -user uživatel` hledá soubor vlastníka `uživatel` (prohledáváme pracovní adresář)

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

Také se dá vyhledávat podle typu nebo velikosti souboru:

`find -type d -name abcd` hledáme rekurzivně z pracovního adresáře všechny soubory, které jsou adresáři (parametr `-type d`) a v názvu mají řetězec `abcd`

`find -type f -name abcd` podobně, ale chceme běžné soubory (file), nikoliv adresáře (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 -type f,d,l -name abcd` zajímá nás víc různých druhů souborů

`find \( -type f -o -type d -o -type l \) -name abcd` totéž, jen trochu víc rozepsané

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

`find -type f -size -2M` vypíšou se běžné soubory (`f`), jejichž velikost je méně než 2 MB

`find /usr/bin /usr/sbin -executable` vypíšeme všechny soubory ze zadaných adresářů, které jsou spustitelné (týká se i skriptů coby textových spustitelných souborů)

Další kritérium je časové – vyhledáváme podle data/času změny (modifikace), posledního přístupu (včetně přístupu „pro čtení“), atd.:

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

`find -mtime +2` hledáme soubor, který byl modifikován (modified) naposledy před více než 2 dny
`find -mtime -2` hledáme soubor, který byl modifikován (modified) naposledy před méně než 2 dny
`find -atime 3` k souboru bylo přistupováno (accessed) naposledy před 3 dny; obecně: můžeme vyhledávat vše, co bylo modifikováno (m) či přistupováno (a) v zadané době, zde jde o dny, ale existují volby i pro minuty
`find ~ -mtime 0` najde vše v pracovním adresáři, co bylo modifikováno před méně než 24 hodinami (tj. uplynul méně než jeden den)



Každý soubor má přiřazena přístupová oprávnění pro vlastníka, přidruženou skupinu a „zbytek světa“. Také toto může být vyhledávací kritérium pro příkaz `find`.



Příklad

Můžeme vyhledávat jak podle „písmenného“ zadání oprávnění, tak i číselného módu:

`find -perm -u+w,g+w` chceme vypsat celé názvy souborů, v jejichž přístupových oprávněních je právo zápisu pro uživatele A ZÁROVEŇ právo zápisu pro skupinu (ostatní oprávnění nejsou testována)
`find -perm /u+w,g+w` chceme vypsat celé názvy souborů, v jejichž přístupových oprávněních je právo zápisu pro uživatele A/NEBO právo zápisu pro skupinu (ostatní oprávnění nejsou testována)
`find -perm /u=w,g=w` totéž
`find -perm -664` hledáme soubory, které mají nastavena oprávnění `rw-rw-r--`, ale pokud je nastaven i jiný bit (například některé `x` nebo `w` ve třetí části), taky bude takový soubor zahrnut
`find -perm /222` hledáme soubory, kde alespoň někdo má právo zápisu – vlastník, skupina nebo ostatní, nebo třeba všichni (ostatní oprávnění nejsou testována, klidně může být i právo čtení nebo spouštění)
`find -perm 664` hledáme soubory, které mají *přesně* nastaven řetězec oprávnění na `rw-rw-r--`, bude ignorovat soubory, které mají v řetězci oprávnění jakoukoliv odlišnost
`find / -perm -4000` vypíše celé názvy souborů (rekurzivně 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



Pokud před řetězec nebo číselnou reprezentaci oprávnění u `-perm` dáme

- pomlčku: stanovíme minimální požadovaná oprávnění (u souboru mohou být i další oprávnění, nebudou vadit), například `-444` by znamenalo, že vlastník, skupina i ostatní musí mít právo čtení,
- lomítko: ještě liberálnější, stanovíme, že alespoň některé z uvedených oprávnění je požadováno, nemusejí být všechna (a mohou být u souboru i další), například `/444` by znamenalo, že alespoň někdo musí mít právo čtení,

- nic: striktní, stanovíme naprosto přesná oprávnění, která musí hledaný soubor mít, například 644 by znamenalo, že všichni musí mít právo čtení a vlastník i právo zápisu, nikdo nesmí mít žádná další oprávnění.



Poznámka:

Zejména způsob vyhledání všech souborů s nastaveným SUID bitem bychom si měli zapamatovat; jak víme, SUID bit je sice někdy nevyhnutelný, ale taky může být nebezpečný. Takže zjednodušeně (chceme jen názvy):

```
find / -perm -4000
```



Úkoly

1. Pomocí příkazu `find` vyhledejte všechny soubory, jichž jste vlastníky a jejichž název začíná tečkou. Dále vyhledejte všechny soubory ve svém domovském adresáři, ke kterým bylo (jakkoliv) přistupováno před méně než dvěma dny.
2. Pokud máte administrátorská přístupová oprávnění (třeba přes mechanismus `sudo`), použijte postup nalezení všech programů s nastaveným SUID bitem (je naznačen v poznámce).



Další příkaz sloužící k prohledávání adresářové struktury je `locate`. Pracuje na trochu jiném principu než `find` – obsah připojených zařízení se pravidelně indexuje (jednou denně) a vytváří se databáze položek v adresářové struktuře, a příkaz `locate` vyhledává právě v této databázi. Důsledkem je výrazně rychlejší vyhledávání. Nevýhodou je, že databáze nemusí být plně aktuální (pak ji ale můžeme aktualizovat ručně spuštěním příkazu `updatedb`, k tomu potřebujeme administrátorská oprávnění).

Vzhledem k vlastnostem tohoto příkazu je výhodné používat `locate` v případech, kdy hledáme soubor, o kterém sice víme, jak se jmenuje, ale nemůžeme si vzpomenout, kde se nachází, přičemž tento soubor existuje víc než jeden den. Příkaz pracuje pouze s těmi položkami, ke kterým máme přístupová oprávnění, takže nejsme obtěžováni upozorněními typu „Přístup zamítnut“.



Příklad

Pokud hledáme soubor `faktury` s nějakou příponou (nevíme jistě, jakou jsme souboru dali) a nevíme, kde hledat, napíšeme:

```
locate faktury.*
```

Vypíše se seznam souborů s plnou cestou, jejichž název odpovídá zadanému schématu. Prohledává se opravdu celá indexovaná adresářová struktura, nikoliv jen z pracovního adresáře.

Hledaný řetězec může být i v cestě k hledanému objektu, takže pokud například chceme seznam dynamických knihoven s moduly programu Gimp, napíšeme

```
locate *gimp*modules*
```

vypíšou se nám řádky s moduly (knihovnamy s příponou `.so`) a pár dalších řádků.



Konfigurace mechanismu `locate` je v souboru `/etc/updatedb.conf`. Také tento soubor je přístupný pouze s administrátorskými oprávněními.



Úkoly

1. Vyzkoušejte si mechanismus `locate`. Pokud máte nainstalován Gimp, vyzkoušejte postup naznačený v předchozím příkladu (ten s moduly). Dále vyhledejte všechny soubory s názvem `passwd`.
2. Porovnejte chování těchto dvou příkazů (doporučuji co nejvíc rozšířit okno terminálu):

```
locate '*.jpg'
find / -name '*.jpg'
```

Všimněte si také, jak příkaz (ne)reaguje v případě nedostupných položek.



Pokud chceme vyhledat spustitelný soubor konkrétního příkazu, použijeme jeden z těchto příkazů: `whereis` nebo `which`. První z nich vyhledá opravdu všechny výskyty souboru s tímto názvem (tj. nejen spustitelné soubory, ale i třeba stejně pojmenované konfigurační soubory, manuálové stránky, atd.) – takže se ptáme „Kde se nachází soubor s tímto názvem?“, druhý se zaměřuje opravdu jen na spustitelné soubory.



Úkol

Vyzkoušejte příkazy `whereis` a `which` s parametrem `passwd`, srovnajte výstup. Dále tyto příkazy vyzkoušejte na jiných příkazech – například `locate`, `ls`, `cut`, `shutdown`, `man`.



1.3.4 Prohledávání textů



Když chceme prohledávat textový soubor nebo třeba textový výstup jiného programu, použijeme příkaz `grep`, případně s přepínačem `-E`. Od předchozích příkazů se odlišuje nejen tím, že prohledává texty (nikoliv strukturu souborů), ale i tím, že dokáže používat plnohodnotné regulární výrazy.

Přepínač `-E` zapíná rozšířené reg. výrazy, často bývá defaultní. Další důležité přepínače:

- `-i` nerozlišuje malá a velká písmena
- `-l` pouze vypíše názvy souborů, ve kterých našel shodu
- `-n` vypíše také číslo řádku
- `-r` rekurzivně 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.



Rozšířené (extended) regulární výrazy (tj. pokud použijeme přepínač `-E` nebo rovnou příkaz `egrep`) berou všechny symboly uvedené v tabulce 1.1 jako speciální symboly, a pokud je chceme speciálního významu zbavit, použijeme zpětné lomítko. Například `*` má speciální význam: určuje, že to, co je před ním uvedeno, se může opakovat jakýkolivpočetkrát, ale pokud chceme v textu prostě vyhledat hvězdičku, zadáme do výrazu `\*`.

Základní regulární výrazy příkazu `grep` to mají naopak – téměř všechny symboly z tabulky 1.1 (kromě tečky, hranatých závorek `[]`, stříšky `^` a dolarovky `$`) ve výchozím stavu nemají speciální význam, ale pokud jim ho chceme dát, musíme před ně napsat zpětné lomítko.

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; <code>a{0,}</code> je totéž jako <code>a*</code>
^	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
(výraz)	uzávorkování výrazu; metaznaky *, ? apod. za závorkou platí pro celou závorku

Tabulka 1.1: Možnosti pro regulární výrazy v příkazu `grep`

Symbolsy tečka hranaté závorky, stříška, dolarovka mají ve výchozím stavu speciální význam u základních i rozšířených výrazů; v obou případech například `[a-z]` je množina malých písmen.



Poznámka:

Jaký je rozdíl mezi jednoduchými výrazy se zástupnými symboly, kterým rozumí programy typu `bash`, `find` apod., a regulárními výrazy v programu `grep` (ať už základními nebo rozšířenými)? U plnohodnotného regulárního výrazu:

- symboly hvězdička, plus a otazník se vztahují k předchozímu symbolu/množině/podvýrazu, nikoliv k „čemukoliv“, navíc můžeme určit číslem/intervalem počet opakování předchozího prvku,
- můžeme vyjádřit fakt, že hledáme řetězec na začátku či konci řádku, začátku či konci slova,
- dá se vyjádřit disjunkce či konjunkce podvýrazů,
- pro jeden výskyt jakéhokoliv znaku se používá symbol tečka, nikoliv otazník, kdežto otazník znamená jeden nebo žádný výskyt předchozího uvedeného podvýrazu,
- negace uvnitř množiny se nevyjadřuje vykřičníkem, ale stříškou, přičemž stříška se používá také k indikaci začátku řádku.



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.



Příklad

Ukážeme si použití příkazu `grep`.

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

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

Tabulka 1.2: Posix sekvence při vyhledávání

```
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 sou-
                          borů 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 -c "Dear \(Mr.\|Ms.\|Miss\) " *.txt    chceme počet anglických oslovení v zadaných souborech
grep -Ec "Dear (Mr.|Ms.|Miss)" *.txt    totéž, ale pomocí rozšířených výrazů
egrep -c "Dear (Mr.|Ms.|Miss)" *.txt    totéž
grep '[0-9]\{6\}/[0-9]\{3,4\}' soubor.txt    v 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\}\(\.[[:digit:]]\{1,3\}\)\{3\}' soubor.log    v zadaném souboru hledáme
všechny IP adresy ve tvaru 123.123.123.123 – čtyři skupiny číslic, v každé skupině 1 až 3 číslice;
jinak:
grep '[0-9]\{1,3\}\(\.[0-9]\{1,3\}\)\{3\}' soubor.log
egrep '[0-9]{1,3}(\.[0-9]{1,3}){3}' soubor.log
```

**Poznámka:**

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). Kdybychom před symboly kulatých a složených závorek nepsali zpětné lomítko, shell by je „sebral“ a pokusil je interpretovat.



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

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 spusti-  
telné (vlastník má právo spouštění)
```



Příklad

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).



Úkoly

1. Zaměřte se na soubor `.profile` ve vašem domovském adresáři (nebo některý jiný konfigurační soubor). Nejdříve vypíšte všechny řádky tohoto souboru, které začínají symbolem `#` (tj. potřebujete vědět, jak určit, že hledaný řetězec je na začátku řádku). Pak naopak vypíšte všechny řádky souboru, které nezačínají tímto symbolem (tj. opět potřebujete vědět, jak určit, že řetězec je na začátku řádku, a navíc potřebujete negovat – bude to negace jednoprvkové množiny).
2. Najděte ve svém domovském adresáři všechny soubory, které obsahují řetězec (dvojnáček) `#!`.
3. Ve výpisu obsahu svého domovského adresáře najděte všechny položky, jejichž název obsahuje řetězec `rc`.
4. V tabulce 1.1 je naznačeno, že `x*` se dá zapsat jako `x{0,}`. Jak podobným způsobem zapíšete `x+` a jak `x??`?
5. Vypíšte počet běžných souborů ze svého domovského adresáře, které jsou skryté (začínají tečkou).
6. Vypíšte seznam všech symbolických odkazů ve svém domovském adresáři. Využijte vhodný regulární výraz na výstup příkazu `ls -l`.



1.4 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 jednotlivý celek, i kdybychom mezitím klepli na klávesu `Enter`. Znaky mezi apostrofy jsou brány jako obyčejný text, zamezuje se interpretaci čehokoliv (zástupných symbolů, proměnných, aliasů apod.). Jsou 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 zabráňujeme interpretaci hvězdičky shellem; 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

Jaký bude výstup těchto příkazů?

- `echo Datum je anglicky date`
- `echo Datum je anglicky `date``

V prvním případě se na obrazovku vypíše prostě to, co je v paramtru příkazu `echo`, v druhém případě se však řetězec `date` interpretuje (bude považován za příkaz, nikoliv za obyčejný řetězec) a místo něj se na výstupu dosadí momentální datum a čas.



Příklad

Příkaz `expr` slouží k výpočtu matematických výrazů. Srovnejme (pozor, mezery kolem operátoru jsou povinné!!!):

Příkaz	Výstup příkazu
<code>expr 1 + 1</code>	2
<code>echo Výpočet: `expr 1 + 1`</code>	Výpočet: <code>expr 1 + 1</code>
<code>echo Výpočet: `expr 1 + 1`</code>	Výpočet: 2

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 na rozdíl od ostatních uvozovacích symbolů píše pouze před symbol, ne 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 (opravdu na konec řádku, nesmí za ním být třeba mezera), pokud má příkaz pokračovat na následujícím řádku – pak je jeho účelem přepínat speciální význam symbolu pro konec řádku (už to nebude speciální symbol vynucující zařádkování).



Příklad

```
echo první řádek\
```

```
> druhý řádek \
```

```
> třetí řádek
```

vypíše

```
první řádekdruhý řádek třetí řádek
```

(všimněte si chybějící mezery; tato mezera chybí, protože jsme ji do původního příkazu před zpětné lomítko nenapsali).



Uvozovky vypínají speciální význam zástupných symbolů (takže například hvězdička uzavřená do uvozovek ztrácí speciální význam a stává se pouhým znakem), 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).

Pozor, uvozovky a apostrofy mají svůj význam (vypnutí speciálního významu) pro daný shell – často je používáme tehdy, když chceme řetězec „uchránit“ před zásahem shellu, aby byly v původním tvaru bez interpretace předány jako parametr příkazu. Samotný příkaz už si s řetězcem samozřejmě může dělat co chce.



Úkol

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.



1.5 Proměnné

1.5.1 Opakování práce s proměnnými

Proměnné se definují v souboru `.bashrc` nebo `.profile` nebo jsou či v jiném konfiguračním souboru (včetně těch systémových). Proměnnou vytváříme standardně přiřazením, například napíšeme `prom=abc` a tak vytvoříme proměnnou `prom` včetně inicializace. Pak ale vzniká *lokální proměnná* platná jen v nejbližším okolí (například shellu, funkci).



Pokud chceme proměnnou *exportovat do prostředí* (tedy zveřejnit do všech procesů, které byly z daného shellu vytvořeny), použijeme příkaz `export` (parametrem bude příslušná proměnná), případně můžeme spojit vytvoření proměnné a `export` do jediného příkazu:

```
export PROM=abc
```

čímž rovnou vytvoříme proměnnou, která bude přístupná i mimo lokální určení ve všech procesech, jejichž (pra)rodičem je shell, v němž pracujeme.

Je zvykem, že proměnné, které chceme exportovat do prostředí, se zapisují velkými písmeny a proměnné, které mají zůstat lokální, malými písmeny.

 Nejdůležitější proměnné 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



Poznámka:

V proměnné PATH není ve výchozím nastavení cesta k pracovnímu adresáři a při spuštění programu také na rozdíl 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.²



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

 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 – podřízených procesů (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)

²Představte 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.

set

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

unset proměnná

odstraní proměnnou

read proměnná

načte ze standardního vstupu řetězec do proměnné

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

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

Příkaz	Výstup příkazu	Komentář
prom="A B C"		jen jsme vytvořili proměnnou
echo \$prom	A B C	vypsali jsme obsah proměnné, ale pozor – mezery
echo "\$prom"	A B C	mezery zachovány
echo '\$prom'	\$prom	apostrofy zamezují interpretaci čehokoliv, proto berou speciální význam symbolu \$
echo ` \$prom `	A: příkaz nebyl nalezen	zpětné apostrofy naopak vynucují interpretaci u čehokoliv, tedy po dosazení obsahu proměnné se shell marně pokouší interpretovat řetězec A B C jako příkaz A s parametry B a C
echo \ \$prom	\$prom	vzali jsme speciální význam symbolu \$

Všimněte si důsledku použití uvozovek v parametru příkazu echo. Výstup je v podstatě podobný jako u předchozího příkazu (tj. bez uvozovek), ale zabránili jsme shellu v interpretaci mezer coby oddělovačů podřetězců, tedy aby sekvenci oddělovačů nezjednodušil na jediný. Nicméně uvozovky nezabránilly interpretaci proměnné.



Příklad

Do proměnné nemusíme ukládat jen řetězec nebo číslo. Jak víme, obrácené apostrofy jsou prostředkem, jak vynutit vyhodnocení výrazu, přičemž nám nic nebrání vnořit je do uvozovek. 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 u posledního příkazu opět můžeme ověřit význam uvozovek.



Příklad

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.

 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 postupy ukázané v příkladech v této sekci (pokud ovšem nejsou destruktivní).



1.5.2 Výpočty

 Používání proměnných úzce souvisí s prováděním výpočtů. Pro tyto účely máme sice rovnou dva příkazy: `let` a `expr`, ale ne vždy jsou nainstalovány (nicméně je lze doinstalovat), tak existuje i „závorkový“ (obecnější) postup.



Příklad

Příkaz `let` slouží ke *změně hodnot proměnných* s vyhodnocením výrazu, v některých případech jsou uvozovky nutné. Ukážeme si, jak se používá.

```
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

`let "prom--"` snížení hodnoty proměnné o 1



Příklad

Příkaz `expr` slouží k podobnému účelu (ovšem nemusíme výsledek ukládat do proměnné), jen si musíme dávat pozor, abychom kolem operátorů měli mezery. Opět pár ukázek použití:

`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`.



Pro označení řetězce za matematický výraz a vynucení jeho interpretace se používají dvojité kulaté závorky. Pokud se tato konstrukce objeví za rovnítkem nebo kdekoli jinde, kde může být i něco jiného než výpočet, předsadíme před levou dvojzávorku symbol dolarovky.



Příklad

Ukážeme si počítání s využitím dvojitých závorek. Syntaxe samotného výrazu odpovídá céčkovské syntaxi.

```
prom=2
(( prom+=8 ))
echo $prom
```

Pozor – kdybychom v druhém příkazu nepoužili dvojité závorky, bylo by to v podstatě správně, ale dostali bychom špatný výsledek – pracovalo by se totiž s řetězcem (zřetězili bychom „2“ a „8“, výsledek by byl „28“ místo čísla 10). Mezery uvnitř dvojzávorek jsou nepovinné (ale nejsou chybou), zde je jejich úlohou jen zvýšení přehlednosti.

```
x=$(( prom*2 ))
echo Výsledek výpočtu 259-prom/4 je $(( 259-prom/4 ))
```



Takže dvojité závorky značí, že s tím, co je uvnitř, se má zacházet jako s matematickým výrazem. Pokud se tam objeví řetězec něčeho jiného než číslic, bude s ním automaticky zacházeno jako s proměnnou, nemusíme před název proměnné dávat dolarovku. Vlastně mají podobnou roli jako zpětné apostrofy.



Příklad

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.

Na následujících výrazech si ve výsledku všimněte rozdílu, když použijeme dvojité závorky:

```
prom1+=80
(( prom2+=80 ))
echo prom1=${prom1}, prom2=${prom2}.
```



Úkoly

1. Proveďte příkazy ze dvou posledních příkladů. 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. Vyzkoušejte výpočty, kde je na pravé straně výrazu proměnná. Načtěte od uživatele hodnotu proměnné a vypište její dvojnásobek.



1.6 Skripty

1.6.1 Co je to skript

 Skript je textový spustitelný soubor, který pro své spuštění potřebuje interpret (třeba některý shell). Soubory se skripty obvykle mívají buď příponu `sh`, anebo 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.



Příklad

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.



Příklad

Vytvoříme jednoduchý skript, ve kterém bude jediný příkaz. Nejdřív vytvoříme nový soubor, který pojmenujeme třeba `testik.sh`. Do tohoto souboru uložíme následující řádky:

```
#!/usr/bin/bash
echo Toto je muj prvni skript
```

Dále předpokládejme, že tento soubor se nachází v našem pracovním adresáři. Skript už nyní můžeme spustit tak, že jeho název předáme jako parametr interpretačnímu programu:

```
bash testik.sh
```

Jenže my chceme, abychom mohli skript spouštět i jako samostatný spustitelný soubor. Pak musíme nastavit oprávnění „x“, třeba v grafickém režimu. V textovém režimu by se to provedlo takto (ted' trochu předbíháme) – platilo by pro všechny uživatele:

```
chmod +x ./testik.sh
```

Soubor bychom pak spustili takto:

```
./testik.sh
```

Tečka a lomítko jsou nutné, jak už jsme si vysvětlili v sekci o proměnných, konkrétně se to týká vlastností proměnné `PATH`.



Pokud nastavíme přístupové oprávnění „x“, můžeme skript spouštět i poklepáním myši v grafickém režimu (ovšem musíme počítat s tím, že po ukončení se okno se skriptem zavře a nebude možné si přečíst případné výstupy).

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



Úkol

Vytvořte jednoduchý skript vypisující řetězec, datum apod. (případně podle své fantazie využijte některé nedestruktivní příkazy z těch, které jsme už probírali). Zajistěte, aby bylo možné tento skript spustit tak, jak je ukázáno v příkladu.



1.6.2 Parametry a návratové hodnoty



Parametry skriptu jsou přístupné přes proměnné `$0` (název skriptu), `$1` (první parametr), `$2` (druhý parametr), ..., `$9`. S těmito parametry zacházíme naprosto stejně jako s běžnými proměnnými, tedy pokud například chceme vypsát na obrazovku obsah druhého parametru, zadáme

```
echo Druhý zadaný parametr skriptu je: $2
```

Pokud bychom chtěli „ještě něco dodat“, uzavřeme číslo parametru do složených závorek:

```
echo Ve čtvrtém parametru je číslo znamenající násobnost: ${4}krát větší náklady...
```

Pokud potřebujeme přistupovat k více než devíti parametrům, pak použijeme příkaz `shift` posouvající obsah parametrů o jeden krok doleva (takže pod `$9` pak bude dostupné to, co bylo původně v desátém parametru, apod., posouvá se obsah všech parametrů). Jako nepovinný parametr můžeme použít číslo označující počet kroků posunutí, 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

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 pořadí prvního parametru (`$1`), ale díky příkazu `shift` 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 `$#`.



Úkol

Napište jednoduchý skript, ve kterém budete počítat se třemi parametry. Tyto tři parametry postupně vypíšete na výstup.



1.6.3 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
```

1.7 Podmínky, větvení a cykly

1.7.1 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). Nejdřív jednoduché možnosti propojení příkazů:

`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 – výstupy/vstupy se předávají postupně po blocích, typicky stránkovací filtry)

 V UNIXových systémech máme možnost využít speciální symboly pro podmíněné vyhodnocování. Pokud vezmeme symboly `&` a `|` a zdvojíme je (bez mezery mezi nimi), budou se chovat následovně:

- dvojsymbol `&&` znamená konjunkci:
 - pokud předchozí příkaz skončil s úspěchem (true), bude vyhodnocen i následující příkaz,
 - pokud předchozí příkaz skončil s neúspěchem (false), nebude následující příkaz vyhodnocován (protože false v konjunkci s čímkoliv znamená vždy false v celém výrazu),
- dvojsymbol `||` znamená disjunkci:
 - pokud předchozí příkaz skončil s úspěchem (true), další příkaz již nebude vyhodnocován (protože jedno true v disjunkci znamená true v celém výrazu),
 - pokud předchozí příkaz skončil s neúspěchem (false), bude vyhodnocen i další příkaz.

Shrňme si syntaxi:

`příkaz1 && příkaz2 && příkaz3` logické AND, následující příkaz se provede pouze tehdy, když předchozí skončil úspěchem (jeho návratový kód byl 0, to zde znamená „true“).

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



Příklad

Chceme ukončit skript při výskytu chyby a zároveň do proměnné `$?` uložit číslo, ze kterého by bylo možné poznat, jaká chyba nastala.

```
...  
cd xyz 2>/dev/null || exit 1  
cp *.txt /media/usb/zaloha || exit 2  
...
```

Pokud došlo k chybě u příkazu `cd` (přesun do neexistujícího adresáře), chybové hlášení bude zahozeno, ale skript se ukončí s chybovým kódem 1, následující příkaz už nebude proveden. Pokud ale `cd` skončí úspěchem (přesuneme se do `xyz`), spustí se příkaz `cp` (kopírování). U něj opět platí, že když dojde k chybě, skript bude ukončen (ale chybové hlášení se objeví).

Uživatel našeho skriptu může zjistit důvod problému tak, že po ukončení našeho skriptu otestuje obsah proměnné `$?` . Když v ní bude číslo 0, je vše v pořádku; číslo 1 znamená, že chyba nastala při přesunu do neexistujícího adresáře; číslo 2 indikuje problém s kopírováním.



 Oba operátory můžeme také kombinovat. Typické použití je vypisování hlášení při správném nebo nesprávném provedení prvního příkazu v pořadí.



Příklad

Podívejme se na tento příkaz:

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

Příkaz `pgrep` v sobě kombinuje funkce příkazů `ps` (výpis běžících procesů) a `grep` (filtrování). Vypíše efektivního vlastníka zadaného procesu (zjistíme, jaká oprávnění daný proces má).

Výsledek je směřován do odpadkového koše, tj. vlastně nechceme nic vypsat. Pouze v případě, že zadaný proces existuje, vypíše se hláška „proces běží“ (protože při vyhodnocení `true AND něco` potřebujeme znát výsledek toho `něco`, aby byl výpočet kompletní), a v případě, že zadaný proces neexistuje (`pgrep` nám „vynadá“ hodnotou `false`, že po něm chceme nemožné), se vypíše druhá hláška (při vyhodnocení `false OR něco` potřebujeme znát výsledek toho `něco`, aby byl výpočet kompletní).



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.

 Pro sdružování několika příkazů spojených těmito symboly můžeme použít závorky (ale jen v případě, že za nimi ještě má něco následovat, třeba symbol `&`). Pozor, za levou a před pravou závorkou musí být mezera. Takto se dá řešit problém poslání na pozadí celé sekvence příkazů.



Příklad

```
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





Úkol

Sestavte příkaz, který provede toto:

- z pracovního adresáře rekurzivně dále vyhledá všechny soubory, jejichž název obsahuje řetězec „satelit“ (použijte příkaz `find`), výstup nasměrujte tak, aby se nezobrazil ani neuložil,
- pokud hledání skončí úspěchem, nechte vypsát hlášku „soubor existuje“,
- pokud hledání skončí neúspěchem, nechte vypsát hlášku „soubor neexistuje“.

Předpokládejme, že takový soubor nikde není (jestli ano, pak zvolte jiný řetězec).

Otestujte. Nejdřív by příkaz měl vypisovat hlášku o neexistenci souboru. Pak vytvořte soubor s takovým názvem (třeba příkazem `touch`) a otestujte znovu. Teď by měl příkaz vypisovat pozitivní hlášku.



1.7.2 Příkazy větvení

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, anebo pokračujeme na dalším řádku (víceřádkový příkaz, klepneme na ).

 **Příkaz `if`** je základním příkazem pro větvení kódu v shellu `bash`. Blok začíná klíčovým slovem `if` a končí sklíčovým slovem `fi` (tj. zrcadlovým obrazem prvního klíčového slova). Jako podmínku můžeme použít cokoliv, co má nějakou návratovou hodnotu (třeba příkaz, který při svém úspěšném provedení vrátí 0 jako `true` nebo nenulovou hodnotu jako `false`).

Podmínku vždy ukončujeme buď symbolem středník (jak v ukázce níže vidíme v prvním sloupci), nebo musíme přejít na další řádek (druhý sloupec), taky je možné využít i větev `else` (třetí sloupec):

<code>if podmínka ; then</code>	<code>if podmínka</code>	<code>if podmínka ; then</code>
<code> příkazy</code>	<code> then</code>	<code> podmínka splněna</code>
<code>fi</code>	<code> příkazy</code>	<code>else</code>
	<code>fi</code>	<code> podmínka nesplněna</code>
		<code>fi</code>

 V případě, že při nesplnění podmínky chceme testovat ještě další podmínku, máme dvě možnosti: buď vnoříme jiný blok s druhou podmínkou, nebo použijeme klíčové slovo `elif`:

<code>if podmínka1 ; then</code>	<code>if podmínka1 ; then</code>
<code> příkazy pro splněnou podmínku1</code>	<code> příkazy pro splněnou podmínku1</code>
<code>else</code>	<code> elif podmínka2 ; then</code>
<code> if podmínka2 ; then</code>	<code> příkazy nesplněnou podmínku1,</code>
<code> příkazy nesplněnou podmínku1,</code>	<code> ale splněnou podmínku2</code>
<code> ale splněnou podmínku2</code>	<code> else</code>
<code> else</code>	<code> žádná podmínka není splněna</code>
<code> žádná podmínka není splněna</code>	<code>fi</code>
<code> fi</code>	
<code>fi</code>	

 Negaci vyjadřujeme vykřičníkem (ostatně, to je v `bash` i v množinách zadaných hranatými závorkami). Za vykřičníkem vždy píšeme mezeru (on je to ve skutečnosti vlastně taky příkaz).

**Příklad**

Píšeme skript, jehož první parametr je název souboru, který má být vytvořen v daném adresáři. Pokud ten adresář neexistuje, bude předem vytvořen. První způsob je tento:

```
if cd ~/dokumenty/faktury ; then
    touch ~/dokumenty/faktury/$1
    echo Soubor $1 vytvoren
else
    mkdir ~/dokumenty/faktury
    cd ~/dokumenty/faktury
    touch ~/dokumenty/faktury/$1
    echo Adresar s fakturami neexistoval => vytvoren, soubor $1 vytvoren
fi
```

Jde to i jednodušeji, třeba s využitím negace (vykřičníku):

```
if ! cd ~/dokumenty/faktury ; then
    mkdir ~/dokumenty/faktury
    echo Adresar s fakturami neexistoval => vytvoren
fi
touch ~/dokumenty/faktury/$1
echo Soubor $1 vytvoren
```

Nebo jinak (s využitím podmíněného vyhodnocování, které jsme probírali v předchozím textu), všimněte si způsobu přechodu na další řádek uvnitř složeného příkazu:

```
cd ~/dokumenty/faktury 2>/dev/null || mkdir ~/dokumenty/faktury ; \
    cd ~/dokumenty/faktury ; echo Adresar s fakturami neexistoval => vytvoren
touch ~/dokumenty/faktury/$1
echo Soubor $1 vytvoren
```

**Příklad**

Pokud přidáváme nové uživatele do systému, můžeme si předem ověřit, zda už náhodou takový uživatel v systému není. K vytváření nového uživatele se dostaneme později, teď si ukážeme jen to otestování. Použijeme příkaz `grep` a necháme ho najít daného uživatele v souboru `/etc/passwd`, kde, jak víme, je seznam registrových uživatelů.

Předpokládejme, že název uživatele máme v lokální proměnné `uzivatel`.

```
if ! grep ${uzivatel} /etc/passwd >/dev/null 2>&1 ; then
    ...
    ted' toho uzivatele vytvorim, protoze podminka je splnena, pokud zadaný uživatel neexistuje
    (vykřičník neguje podmínku)
fi
```

Abychom nemuseli složitě řešit směrování výstupu i chybového výstupu do `/dev/null`, můžeme u příkazu `grep` použít prepínač `-q` pro tichý režim (quiet), jen vrátí hodnotu `true` (`=0`) nebo `false` (`>0`):

```
if ! grep -q ${uzivatel} /etc/passwd ; then
    ...
    ted' toho uzivatele vytvorim, protoze podminka je splnena, pokud zadaný uživatel neexistuje
    (vykřičník neguje podmínku)
fi
```



**Poznámka:**

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).



Testování parametrů jako je srovnávání hodnot čísel/řetězců a proměnných, existence souboru či adresáře apod., provádíme pomocí příkazu `test`. Tento příkaz má za parametr testovací logický výraz, vrací buď hodnotu `true` (jako každý proces – návratová hodnota =0) nebo `false` (>0).

Buď použijeme přímo klíčové slovo `test`, nebo použijeme jeho „závorkovou“ podobu. Níže vidíme vlevo syntaxi tohoto příkazu v obou podobách, vpravo pak jeho začlenění jako podmínku příkazu `if`.

```
test výraz                if test výraz; then
[ výraz ]                 if [ výraz ] ; then
```



Ted' se podíváme na podobu testovacího výrazu. Používají se řetězce obvykle představující proměnné a dále operátory reprezentované těmito přepínači:

Pro řetězce:

<code>řetězec</code>	= true, pokud je řetězec neprázdný
<code>-z řetězec</code>	= true, pokud je řetězec prázdný
<code>řetězec1 = řetězec2</code>	= true, pokud jsou řetězce stejné
<code>řetězec1 != řetězec2</code>	= true, pokud jsou řetězce různé

Pro názvy souborů:

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

Pro čísla a proměnné s čísly:

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

Skládání výrazů:

<code>! výraz</code>	negace výrazu
<code>výraz1 -a výraz2</code>	logické AND
<code>výraz1 -o výraz2</code>	logické OR
<code>\(výraz \)</code>	uzávorkování výrazu

**Poznámka:**

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. Totéž platí v případě, že v proměnné je uložen řetězec s mezerami nebo obecně jinými než základními znaky.





Příklad

Ukážeme si použití jednoduchého testování na zjištění existence adresáře. Před přesunem do zadaného adresáře si můžeme takto otestovat, zda existuje (vlevo je zápis s klíčovým slovem, vpravo závorková forma):

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

if [ -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.



Příklad

Názvy adresářů jsou vlastně řetězce, tedy se s nimi dá zacházet jako s jinými řetězci. Následující příkaz zjišťuje, zda jsme v kořenovém adresáři (uvedeme si už jen závorkovou formu):

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



Příklad

Víme, že pro spuštění souboru se skriptem potřebujeme právo „x“. Můžeme otestovat, zda ho máme:

```
if [ ! -x ./testik.sh ] ; then
    echo Nemám dostačující přístupová oprávnění. Končím.
    exit 1
fi
./testik.sh
```

Vykřičník nám neguje výraz v podmínce. Kolem hranatých závorek (obou) a za vykřičníkem bychom měli udělat mezeru.



Úkoly

1. Vyzkoušejte si ověřování existence uživatele pomocí příkazu `grep` podobně, jak je to v příkladu na straně 25 (místo proměnné použijte své vlastní přihlašovací jméno nebo třeba `root`, případně i řetězec, který není žádným přihlašovacím jménem). Ve větvích `then` a `else` jen vypíšte informaci o existenci nebo neexistenci uživatele. Vyzkoušejte obě větve.

Nemějte obavy v shellu na těch správných místech klepnout na  – na dalších řádcích složeného příkazu se jednoduše dostanete do promptu nižší úrovně a budete pokračovat ve psaní. Pokud budete chtít v shellu psát celý příkaz na jeden řádek, musíte na místech, kde byste jinak dali konec řádku, umístit středník (třeba za `then` nebo před `fi`).

2. Vyzkoušejte i to, co je v následujících příkladech (práce s adresáři, varianty příkazu `test` nebo hranaté závorky). Nejdřív v pracovním adresáři vytvořte skript, ve kterém zjistíte, jestli pracovním adresářem je či není kořenový adresář (`root`), a pak buď v jiném skriptu nebo přímo v shellu otestujte, zda k původnímu skriptu máte oprávnění spuštění.

To otestování se dá provést i bez příkazu `if`:

```
[ -x skript.sh ] || echo nemam opravneni
```



 **Větvení pomocí příkazu `case`** je praktičtější, pokud chceme otestovat více alternativ pro danou hodnotu či výraz. Princip je podobný jako v jiných programovacích jazycích – stanovíme výraz, jehož výsledek chceme otestovat (typicky proměnnou, podle které větvíme) a určíme jednotlivé větve.

V syntaxi je za hodnotou větve pravá závorka, každá větev se ukončuje dvojicí středníků, celý složený příkaz ukončujeme zrcadlovým obrazem prvního klíčového slova – `esac`. Můžeme (nemusíme) použít větve typu `default` (místo konkrétní hodnoty dáme hvězdičku).

```
case $prom in
hodnota1)
    příkazy pro případ, že $prom=hodnota1
    ;;
hodnota2)
    příkazy pro případ, že $prom=hodnota2
    ;;
hodnota3 | hodnota4)
    příkazy pro případ, že $prom=hodnota3 nebo $prom=hodnota4
    ;;
*)
    toto je větev default, resp. else; provede se, když žádná z předchozích větví není použita
esac
```

1.7.3 Cyklus přes množinu

Je dána množina prvků (řetězců, čísel apod., cokoliv, co můžeme dosadit do proměnné). Touto množinou může být například množina všech názvů souborů, které odpovídají zadané masce (pomocí hvězdičky apod.), nebo prvky množiny můžeme rovnou vyjmenovat. Přes všechny prvky této množiny potřebujeme provést určitý příkaz nebo sekvenci příkazů.

 Příkaz `for` slouží právě k tomuto účelu. Stanovíme množinu prvků a určíme proměnnou, do které jsou postupně prvky dané množiny dosazovány, a pak samozřejmě ty příkazy. Základní syntaxe je tato (jsou tu ukázány dva způsoby reprezentace množiny řetězců):

```
for prom in řetězec1 řetězec2 řetězec3 ; do    mnozina="řetězec1 řetězec2 řetězec3"
    příkazy používající proměnnou $prom        for prom in $mnozina ; do
done                                             příkazy používající proměnnou $prom
done
```

Pokud bychom chtěli klíčové slovo `do` napsat až na další řádek, středník si můžeme odpustit. Všimněte si rozdílu oproti předchozím složeným příkazům: klíčové slovo pro ukončení složeného příkazu tentokrát není zrcadlovým obrazem prvního klíčového slova.

 Klíčové slovo `in` je ve skutečnosti příkazem, jehož parametry jsou prvky zadané množiny řetězců, středník za množinou tento příkaz odděluje od příkazu `do`, jehož úkolem je přijmout výsledek příkazu `in` a předat ho vnořeným příkazům.

Příklad

Protože víme, že zástupné symboly, jejichž interpretaci jsme nezakázali (zpětným lomítkem, uvozovkami, apostrofy), interpretuje shell ještě před předáním řetězce zadanému příkazu, je jasné, že v řetězcích klidně můžeme tyto zástupné symboly používat.

```
for i in *.txt ; do
    echo Kopíruji $i
    cp ${i} /media/sitovy/zalohatxt
done
```

Příklad

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`.

```
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 rekurzivně, kopírujeme je na soubory do stejného adresáře, ale s jinou příponou (`BAK`). Všimněte si, jakým způsobem bylo provedeno „odstříhnutí“ původní přípony a zřetězení s novou příponou.

Kdybychom chtěli celý příkaz umístit na jeden řádek:

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

Příklad

Následující skript (zapsaný v souboru) v cyklu prochází všechny soubory z pracovního adresáře včetně skrytých, u každého se zeptá, jestli ho má smazat. Pokud ano, použije příkaz pro rekurzivní mazání. Všimněte si, jakým způsobem je ošetřena chyba při mazání souboru.

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

#vytvoříme proměnnou
souhlas="N"
```

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

Řetězec ``ls -a`` bude interpretován shellem, výsledek zahrnuje všechny skryté soubory a adresáře z pracovního adresáře. Tento řetězec můžeme zapsat i jinak:

```
$( ls -a )
```



Příklad

V parametru určujícím množinu mohou být jakékoliv řetězce. Také tam mohou být proměnné, jejichž interpretaci si chceme vynutit. Dokonce můžeme naopak vynucení pozdržet.

```
#!/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
```



Příklad

Ve skriptu chceme do proměnné uložit abecedně seřazený seznam uživatelů oddělených mezerou (abychom si procvičili řetězení obsahu proměnných).

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

Místo zápisu podmínky pomocí dolarovky se závorkami jsme alternativně mohli použít zápis pomocí zpětných apostrofů (je to jedno, obojí způsobí interpretaci obsahu). „Rozřezali“ jsme řádek souboru podle oddělovačů „dvojtečka“ a vybrali jsme první sloupec.





Úkoly

1. Zapište do shellu následující složený příkaz (tak jak je, na víc řádků):

```
for i in `ls -Ra` ; do
    if [ -x $i -a ! -d $i ] ; then
        echo spustim $i
    fi
done
```

Výstupem příkazu `ls -Ra` je seznam všech názvů souborů z pracovního adresáře, a to rekurzivně. Každý z těchto názvů je profiltrován příkazem `if` – projdou jen ty, ke kterým máme nastaveno oprávnění „x“ a zároveň to nejsou adresáře (vykřičník). Co tedy bude výstupem příkazu?

2. Pokud jste správně zapsali složený příkaz z předchozího úkolu (a na více řádků), zkuste teď udělat krok do historie – stiskněte jednou kurzorovou klávesu pro pohyb zpět v historii. Jak se vám zobrazil předchozí zadaný příkaz?

Zde si všimněte, kam konkrétně patří středníky, pokud budeme chtít složený příkaz zapsat na jediném řádku.



1.7.4 Cykly pro rozsah hodnot



V shellu `bash` dále máme cykly `while` a `until`. Jejich syntaxe je následující:

<pre>while [podmínka] ; do příkazy done</pre>	<pre>until [podmínka] ; do příkazy done</pre>
---	---

Cyklus `while` provádí příkazy ve svém těle tak dlouho, *dokud platí podmínka*, cyklus `until` naopak provádí příkazy ve svém těle, *dokud nezačne platit podmínka*. Opět je třeba dávat pozor na umístění mezer, formát podmínky je stejný jako u příkazu `if`, resp. `test`.



Příklad

Následující dvě varianty kódu jednoduše čekají na stisknutí jakékoliv klávesy kromě klávesy „N“.

<pre>#!/bin/bash klavesa="N" until [\$klavesa != "N"] ; do echo Stiskni něco jiného než N... a Enter read klavesa done</pre>	<pre>#!/bin/bash klavesa="N" while [\$klavesa = "N"] ; do echo Stiskni něco jiného než N... a Enter read klavesa done</pre>
--	---



Příklad

Proměnná `CITAC` postupně roste o 1, přičemž určuje, kolikrát bude proměnná `vysledek` násobena číslem 2. Výsledek celého výpočtu bude mocnina čísla 2. Skript můžeme vytvořit v textovém editoru, například `xed` nebo `nano`.

```
#!/bin/bash
CITAC=1
POSLEDNI=5
vysledek=1
while [ $CITAC -le $POSLEDNI ] ; do
    (( vysledek*=2 ))
    (( CITAC+=1 ))
done
echo 2^{POSLEDNI} = $vysledek
```



Úkol

Projděte si příklady v této sekci. U posledního příkladu pozměňte kód tak, aby obsah proměnné POSLEDNI bral z parametru skriptu.



1.8 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, prostředí Eclipse a další. 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.



Úkol

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

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

Uložte do vašeho domovského adresáře a pak přeložte:

```
gcc -o world world.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:

```
./world
```



1.9 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` nebo `iconv -l`.

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
```

Správa procesů

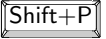
2.1 Příkazy pro práci s procesy

S procesy můžeme pracovat například pomocí těchto příkazů:

 **Příkaz top** interaktivně vypisuje seznam procesů, které momentálně běží, a to v pravidelných intervalech. Ukončí se stiskem klávesy  (nebo jiným druhem ukončení) – tak jako každý interaktivní proces.

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

Pro řízení výpisu můžeme používat některé klávesové zkratky, jejich seznam získáme stisknutím klávesy  nebo .

 Z klávesových zkratk je užitečná například , která setřídí procesy podle zatížení CPU (Processor, může odhalit procesy, které se zasekly),  podle množství zabrané paměti (Memory), stisknutím  zapneme nebo vypneme zobrazování informací o paměti zabrané procesy,  umožní bez nutnosti opuštění programu poslat některý signál vybranému procesu (zadáme PID procesu a signál).

 **Příkaz ps** vypisuje všechny momentálně běžící (running) procesy na určitém terminálu (PID, speciální soubor terminálu, čas CPU, který proces již využil, název procesu). Na rozdíl od předchozího příkazu tento příkaz není interaktivní (tj. vypíše výsledek a ukončí se). To, které procesy a jaké jejich vlastnosti chceme vypsat, určujeme pomocí přepínačů.



Příklad

Můžeme použít například tyto přepínače:

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

`ps -fp 1583` vypíše podrobnější informace o procesu se zadaným PID (číslo je za „p“)

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

`ps -e | grep firefox` chceme údaje o procesu s názvem *firefox*, ostatní řádky se nezobrazí

`ps -u uživatel` 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)



Nejběžnější je používání `ps -ef` nebo `ps aux`.



Úkol

Srovnajte výstup příkazů `ps -e` a `ps -ef`, zaměřte se na proces s PID 1. Všimněte si, že je zde určitá nekonzistence v názvu souboru, ze kterého proces vznikl. K tomuto procesu se vrátíme v části skript věnované inicializačním systémům a službám v poslední kapitole.



Poznámka:

Jak vyplynulo z výše uvedeného příkladu, u příkazů (nejen `ps`, i dalších) se používá několik druhů syntaxe, přičemž v Linuxu jsou vcelku bez problémů podporovány zároveň. Setkáváme se především s těmito druhy:

- tradiční UNIXová syntaxe: přepínače jsou často jednoznakové, před ně se dává pomlčka, mohou se sdružovat
například `ps -ef`
- BSD syntaxe: přepínače jsou taky často jednoznakové, ale bez jakékoliv pomlčky, taky se mohou sdružovat
například `ps aux`
- FSF (Free Software Foundation) syntaxe: používají se spíše slovní (víceznakové) parametry, před nimi jsou dvě pomlčky, takové parametry nemůžeme sdružovat
například `ls --all` (místo UNIXového `ls -a`)



Příkaz `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.

Strom procesů se dá vypsat i jinak: `ps --forest` nebo `ps -H`
(plus další přepínače podle potřeby)

Příkaz `pgrep` je vlastně kombinace příkazů `ps` a `grep`, vrátí PID zadaného procesu.



Příklad

Pomocí parametrů můžeme přidávat další kritéria.

`pgrep bash` vrátí PID procesu `bash`, pokud je ovšem nějaký spuštěn (nebo několik, když je jich spuštěno víc)

`pgrep -u novak bash` vrátí seznam procesů (včetně PID), které se nazývají *bash* a zároveň je jejich efektivním uživatelem *novak*

`pgrep -x systemd` vypíše pouze ty procesy, jejichž název je přesně zadaný řetězec (jinak by byly ve výstupu i procesy, u nichž je řetězec součástí názvu, například *systemd-journal*), můžeme si ověřit příkazem

```
ps -e | grep systemd
```

(tento proces je základním systémovým procesem ve většině současných linuxových distribucí, dříve tuto roli plnil proces *init*)

`pgrep -d, systemd` tímto určíme oddělovač, který se má použít v případě, že je výstupů víc než jeden, výchozím oddělovačem je konec řádku (v tomto případě získáme sekvenci PID procesů s názvem *systemd*, budou odděleny čárkou)

`ps -fp `pgrep -d, -x systemd`` vytvoří čárkami oddělený seznam PID procesů s názvem *xterm*, pak ho předá procesu *ps*, který o zadaných procesech vypíše podrobné informace (vzpomeňte si, k čemu slouží zpětné apostrofy)



 **Příkaz `mpmap`** vytvoří podrobný rozpis paměti využívané procesem (nejen vlastněné), jako parametr se používá PID procesu. Také zjistíme, jestli proces využívá sdílenou paměť některé knihovny nebo jiného procesu. PID získáme například pomocí `pgrep`.

 **Příkaz `whereis`** vypisuje umístění spustitelného souboru příkazu a dalších souvisejících souborů (včetně manuálové stránky).



Příklad

Podíváme se na výstupy příkazu `whereis`.

`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 1 – kdo je vlastníkem procesu, kdo je jeho rodičovským procesem, jak se proces nazývá. Pak si vyberte jiný proces (s vysokým PID) a opět o něm zjistěte informace.

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 (popř. zda vůbec) jsou nainstalovány programy *man*, *mount*, *gcc* a *ls*.



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. Bývá to část podstromu ve stromové struktuře procesů (ne celý podstrom, často jenom nejbližší potomci, a ani ti tam nemusejí patřit všichni). Proces, který je v kořeni tohoto podstromu, je (většinou, až na výjimky) hlavním procesem skupiny. Typickým důvodem zařazení procesů do téže skupiny je možnost snadnější komunikace mezi nimi, ať už pomocí roury nebo signálů. Skupina procesů má své identifikační číslo – *PGID* (Process Group ID).

Další situací, kdy vzniká skupina, je použití roury při zadávání uživatelem v shellu. Jestliže spustíme řadu příkazů propojených symbolem roury „|“, pak takové procesy patří do stejné skupiny.

Systémové procesy mají obvykle své PGID nastaveno na 0. Proces s PID 0 neexistuje, tedy nejsou zařazeny do žádné skupiny. Démony, tj. procesy služeb, už obvykle do některé skupiny patří, mají nenulové PGID. Démon *smbd* (tj. Samba pro sdílení souborů v rámci lokální sítě) mívá potomky vzniklé ze stejného souboru (tj. taky *smbd*), přičemž všechny patří do téže skupiny, totéž platí i o některých jiných démonech (například *avahi-daemon*).

Další skupiny procesů jsou tvořeny inicializačními procesy grafických prostředí.

 *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. Typickým důvodem zařazení procesů do jedné relace je možnost kaskádového posílání signálů (procesy distribuují signál dál svým potomkům, ti zase svým potomkům atd. rekurzivně až ke konci relačního stromu). Procesy patřící do jedné skupiny (mající stejné PGID) taky patří do jedné relace, ale naopak to neplatí (v relaci obvykle míváme více skupin procesů). Procesy dokonce mohou přecházet mezi skupinami, ale vždy v rámci téže relace.

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 většinou PID hlavního procesu relace. V mnoha případech se číslo SID shoduje s číslem PGID.

Systémové procesy mají taktéž většinou SID rovno 0, stejně jako PGID. U démonů se obvykle SID kryje s PGID.



Příklad

Předpokládejme, že jsme v systému, který používá grafické prostředí MATE. Máme spuštěno několik instancí (oken) programu *Terminal*. V jednom z nich je spuštěn níže uvedený příkaz *ps* a v druhém je zobrazena manuálová stránka některého příkazu, přičemž pro zobrazení manuálové stránky je použit stránkovač (ten zajišťuje interaktivitu celé úlohy).

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

```
PID  SID  PGID  COMMAND
...
488  488  488  avahi-daemon
543  488  488  avahi-daemon
...
612  612  612  smbd
638  612  612  smbd
639  612  612  smbd
...
788  788  788  mate-session
...
1524 788  788  mate-terminal
1532 1532 1532  bash
4725 1532 4725  ps
4618 4618 4618  bash
4708 4618 4708  man
4715 4618 4708  pager
...
```

První dvojice záznamů se týká dvou procesů démona `avahi`. První spuštěný si spustil potomka ze stejného spustitelného souboru, potomek patří do skupiny i relace vytvořené svým předkem (tj. PGID i SID jsou odvozeny od PID rodiče).

Další je trojice záznamů procesů s názvem `smbd`. První z nich spustil dva potomky se stejným kódem, přičemž tito potomci jsou zařazeni do stejné skupiny i relace.

Proces `mate-session` je hlavním procesem v další skupině i relaci, jeho PID 788 je použito jako PGID a SID u procesů tvořících prvky grafického rozhraní (například proces provozující hlavní panel na obrazovce, a také proces terminálu, který vznikl klepnutím na tlačítko pro spuštění terminálu na hlavním panelu, proto rodičovský vztah).

Máme spuštěna dvě okna terminálu, přičemž samotný terminál je pořád ten jeden proces, ale v jednotlivých oknech máme dvě různé instance procesu `bash`. Každá z nich si vytváří vlastní skupinu i relaci. Ovšem procesy z příkazů, které jsou v oknech spouštěny, sice budou mít vlastní skupinu (do ní si mohou řadit vlastní potomky), ale relaci dědí od instance `bash`, ve které existují.

Příslušnost ke skupině: procesy `man` a `pager` jsou ve stejné skupině (vytvořené nadřízeným příkazem), a tedy si mohou snadno předávat informace o tom, co má být zobrazeno (`pager` dostává instrukce o obsahu stránek od `man`). Příslušnost k relaci: pokud zavřeme toto okno terminálu, signál pro ukončení procesu bude vyslán všem v dané relaci (od `bash` přes `man` až k procesu `pager`).



Příklad

Předpokládejme, že na dvou různých terminálech (`tty2` a `tty3`) jsme se přihlásili, na terminálu `tty2` jsme spustili dva příkazy:

- `cat &`
- `ping www.google.com`

Na terminálu `tty3` byl spuštěn příkaz `rev | grep "abc"`. Vzpomeňte si na konzoly – na druhou konzolu (tedy do terminálu `tty2`) se z grafického rozhraní dostaneme klávesovou zkratkou `Ctrl` + `Alt` + `F2`, z druhé na třetí zkratkou `Alt` + `F3` (bez `Ctrl`), protože jdeme z textové do textové), zpět do

grafického režimu se dostaneme přesunem na konzolu, na které běží grafické prostředí, což může být tty7 (v tom případě  + .

Podívejme se na výpis následujícího příkazu:

```
sarka@sarka-VB-pc:~$ ps -eHo tty,pid,pgid,sid,comm | grep "tty[23]"
```

tty2	2166	2166	2166	login
tty2	2197	2197	2166	bash
tty2	2220	2220	2166	cat
tty2	2223	2223	2166	ping
tty3	2247	2247	2247	login
tty3	2269	2269	2247	bash
tty3	2364	2364	2247	rev
tty3	2370	2364	2247	grep

Díky filtru `grep` se vypsaly pouze řádky procesů z tty2 a tty3. Srovnáme řádky příkazů spuštěných uvnitř shellů.

Nejdřív se zaměříme na PGID. V případě terminálu tty2 to jsou `cat` a `ping`, na terminálu tty3 jde o `rev` a `grep`. Zatímco první dva příkazy jsou spuštěny na sobě nezávisle, druhé dva příkazy jsou propojeny rourou a tvoří jednu úlohu. Všimněte si, že první dva příkazy mají různé PGID (2220 a 2223, vlastně to jsou PID těchto procesů, tedy každý má svou jednočlennou skupinu), druhé dva příkazy mají společné PGID (2364, což je PID procesu `rev`, prvního v rourě). Z toho vyplývá, že procesy spuštěné v rourě tvoří společnou skupinu.

A teď SID. Všechny procesy spuštěné na terminálu tty2 mají SID 2166, všechny procesy spuštěné na terminálu tty3 mají SID 2247, což je PID příslušného procesu `login`. Na každém terminálu je totiž nutné se přihlásit „zvlášť“. Z toho vyplývá, že relace opravdu souvisí s přihlášením, všechny procesy vzniklé při přihlášení na konkrétním terminálu patří do téže relace. I když se tentýž uživatel přihlásí na různých terminálech (zde konzolách), jde o různé relace.



Úkol

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

Do výpisu můžete přidat také další sloupce, například `user` pro zobrazení vlastníka procesu, případně se podívejte do manuálové stránky příkazu `ps` a hledejte část „Standard format specifiers“ (napište lomítko, za ně tento řetězec, , klávesou  se pohybujete na další výskyty řetězce).



2.3 Úlohy a multitasking

V shellu `bash` můžeme s procesy pracovat na vyšší úrovni – pracujeme s *úlohami*. V jedné úloze může být i více procesů určitým způsobem propojených (zařazených čistě sekvenčně pomocí středníku, komunikujících přes rouru, atd.). Tento styl práce souvisí s multitaskingem a práce s úlohami vlastně znamená práci v multitaskovém režimu i v textovém shellu.

2.3.1 Úlohy na popředí a pozadí

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 (zobrazí se prompt a my můžeme zadat další příkaz),
- 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:

- | | |
|---|--|
|  | ukončení úlohy běžící na popředí (některé programy tuto zkratku nereagují) |
|  | pozastavení úlohy na popředí (platí totéž) |
|  | 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 připseme symbol & (za mezeru):

`dlouhý_výpočet &`

 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 (ano, relace: o relacích jsme se bavili v předchozí sekci), kde uživatel pracuje (tedy mnohem nižší čísla). V jedné úloze může být víc procesů, které pak mají různá čísla PID, ale totéž číslo úlohy.

Pokud chceme toto identifikační číslo použít jako parametr některého z příkazů pro práci s procesy, musíme před ním napsat znak %, abychom odlišili číslo úlohy od čísla procesu. Výjimkou jsou příkazy určené pouze pro úlohy, kdy příkaz „ví“, že jde o číslo úlohy a ne o PID.

 Příkazy pro práci s úlohami:

`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]`

pošle úlohu, která je na pozadí, zpět na popředí. Jako parametr můžeme zadat číslo úlohy, která má být přenesena na popředí. Pokud je příkaz použit bez parametru, je poslána na popředí ta, která byla jako poslední „odsunuta“ (ve výpisu úloh je označena „+“) – princip „zásobník“.

`bg [uloha]`

spustí pozastavenou úlohu na pozadí. Opět můžeme v parametru zadat číslo úlohy. Úloha se nepřesune na popředí, jenom začne dostávat procesor.

 **Poznámka:**

Pokud chceme úlohu běžící na popředí poslat na pozadí, nemůžeme to logicky udělat příkazem, protože ta úloha „zabírá prompt“ (nemůžeme zadat žádný příkaz). Postup:

- Použijeme klávesovou zkratku , čímž úlohu pozastavíme (je na pozadí, nemá přidělován procesor – je pozastavená, ovšem máme k dispozici prompt a můžeme zadávat příkazy).
- Pokud neznáme číslo této úlohy, teď máme možnost ho zjistit: zadáme `jobs`, poslední odsunutá úloha je ta, kterou potřebujeme, poznáme ji podle symbolu „+“.

- Pokud chceme, aby na pozadí i běžela (zatím tam jen stojí), použijeme příkaz `bg`, jako parametr zadáme číslo úlohy.



Příklad

Pohrajeme si se stavem jedné úlohy.

`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 směřován na pozadí; objeví se řádek (číslo může být jiné, zobrazuje se PID procesu):

```
[1] 5783
```

`jobs` vypíšeme seznam úloh spuštěných na tomto terminálu, výstup bude jednořádkový (pokud jsme předtím nespustili ještě nějakou jinou úlohu):

```
[1]+ Running      yes > /dev/null &      (případně se vypíše české Běží)
```

`fg 1` úlohu s číslem 1 chceme poslat na pozadí, takže nám zmizí prompt

 tuto úlohu jsme poslali zpět na pozadí, zobrazí se

```
[1]+ Stopped      yes > /dev/null      (nebo Pozastavena)
```

`jobs` ověříme si, jak je to se stavem naší úlohy:

```
[1]+ Stopped      yes > /dev/null      (nebo Pozastavena)
```

`bg 1` úlohu necháme na pozadí, ale spustíme ji tam, bude dostávat čas procesoru, vypíše se:

```
[1]+ yes > /dev/null
```

`jobs` ověříme stav:

```
[1]+ Running      yes > /dev/null &      (případně Běží)
```

`kill %1` už nás to přestalo bavit, tak tento proces ukončíme; příkazu `kill`, který k tomu slouží, buď zadáme PID procesu (v našem případě by bylo `kill 5783`), nebo, jako zde, číslo úlohy se symbolem procenta



Všimněte si, že v parametrech příkazů `fg` a `bg` píšeme číslo úlohy bez symbolu procenta (protože tyto příkazy primárně pracují s číslem úlohy), kdežto u příkazu `kill` zadáváme číslo úlohy se symbolem procenta (protože tento příkaz umí pracovat s obojím a musíme to nějak rozlišit).



Příklad

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

`yes > /dev/null &` spustíme na pozadí program `yes`, výstup opět přesměrujeme do souboru `/dev/null`, zobrazí se:

```
[1] 5783
```

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

```
[2] 5787
```

`jobs` ověříme si stav našich dvou spuštěných úloh, výstup je tento:

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

Jak vidíme, v hranatých závorkách je číslo úlohy.

`rev &` tento příkaz na rozdíl od předchozího vše, co dostane na svůj vstup, obrací (reverze), výpis:

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

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

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

Symbols + 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, tj. `rev`

 tuto úlohu jsme poslali zpět na pozadí, zobrazí se

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

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

 úlohu jsme poslali zpět na pozadí

`kill %1 %2 %3` ukončíme všechny tři úlohy, případně můžeme pomocí `jobs` zkontrolovat, v jakém jsou stavu



Poznámka:

Určitě jste si všimli, že ve výstupu příkazu `jobs` se na konci řádku někdy objevuje symbol `&` a jindy ne, ale přitom vlastně všechny vypisované procesy byly s tímto symbolem spuštěny. Když si pozorně prohlédnete jednotlivé případy, zjistíte, že `&` je jen na řádcích, kde vidíme `Running` (nebo `Běží`).



Úkol

Vyzkoušejte postupy ukázaný v příkladech, klidně i na jiných interaktivních či déle běžících procesech. Můžete pracovat s interaktivními úlohami v příkladu uvedenými, případně i s dalšími, například `sleep 5000` (zvolte hodně velký počet sekund), `nslookup` (interaktivní překlad DNS názvů na IP adresy), `ping` s nějakou existující adresou (v UNIXových systémech je výchozí chování takové, že posílá pakety ICMP Echo Request, dokud není přerušeno ,  (výstup pošlete do `/dev/null`), užitečná je i kalkulačka nebo nějaký jiný program s GUI, apod.



2.3.2 Úlohy a procesy

Zatímco pro výpis procesů používáme příkaz `ps`, pro výpis úloh používáme příkaz `jobs`. Pokud u příkazu `ps` nepoužijeme parametr `-e`, vypíšu se pouze procesy související s daným terminálem, přesněji s danou instancí shellu `bash`.

**Příklad**

Vyzkoušíme následující posloupnost příkazů:

```
cat &
rev &
```

Ted' vypíšeme seznam procesů/úloh pomocí příkazů `ps` a `jobs`:

`ps`

Výstup:

PID	TTY	TIME	CMD
4023	pts/2	00:00:00	bash
4039	pts/2	00:00:00	cat
4040	pts/2	00:00:00	rev
4043	pts/2	00:00:00	ps

`jobs`

Výstup:

```
[1]- Stopped  cat
[2]+ Stopped  rev
```

Příkaz `ps` vypisuje i proces, který je na vyšší úrovni ve stromu procesů, a tedy se ve výstupu příkazu `jobs` nemůže objevit – `bash`. Taky vypisuje sám sebe (ve výstupu příkazu `jobs` se neobjeví jednoduše proto, že v té chvíli už neexistuje).



Jedna úloha se může skládat z více procesů. Pokud chceme na pozadí poslat složitější úlohu, uzavřeme celou úlohu, tedy sekvenci procesů, do závorek a za ně napíšeme symbol `&`:

```
( proces1 ; proces2 ; proces3 )&
```

Pozor na to, kde všude jsou středníky a mezery. Kolem středníků mezery být nemusejí, ale na vnitřní straně závorek ano.

**Příklad**

Vyzkoušejte tyto příkazy:

```
sleep 5; echo hotovo
```

```
( sleep 5; echo hotovo )
```

V prvním i druhém případě na 5 sekund (trochu víc) zmizí prompt a po uplynutí této doby se vypíše zadaný řetězec, pak teprve se prompt objeví. Rozdíl je pouze v tom, že oba procesy jsou zařazeny do stejné úlohy (což od pohledu nepoznáme).

```
sleep 5; echo hotovo &
```

V třetím případě je reakce zdánlivě stejná, s tím rozdílem, že se ještě před výpisem příkazu `echo` objeví řádek s oznámením o spuštění úlohy na pozadí (na první pohled nepoznáme, o jakou úlohu jde, to bychom si museli ověřit PID). Každý z těchto procesů je v samostatné úloze, ta druhá úloha probíhá na pozadí (první ne, takže těch 5 sekund nevidíme prompt).

```
( sleep 5; echo hotovo )&
```

Ve čtvrtém případě jsou oba procesy opět zařazeny v jedné úloze, a ta je jako celek poslána na pozadí (zobrazí se informace o poslání úlohy na pozadí, hned se objeví prompt). Po zhruba 5 sekundách se na výstup (za prompt) vypíše zadaný řetězec a shell čeká na potvrzení klávesou Enter.

```
sleep 5 & echo hotovo &
```

V pátém případě obě úlohy posíláme na pozadí, takže se hned zobrazí prompt. Obě se spustí paralelně, takže čekáme 5 sekund a zároveň (tj. hned) se vypíše daný řetězec, systém čeká na Enter.





Úkol

Vyzkoušejte postupy uvedené v obou příkladech.



2.4 Komunikace procesů

Procesy mohou vzájemně komunikovat několika způsoby:

- posíláním signálů,
- posíláním zpráv (IPC, InterProcess Communication), zpráva na rozdíl 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 něco na způsob restartu procesu (ale v realu ten proces běží v kuse), 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), zůstane zabráný, 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. Abychom vůbec byli schopni tento mechanismus používat, musíme umět identifikovat příjemce, tedy znát jeho PID nebo číslo úlohy.

 K posílání signálů slouží příkazy `kill` (posíláme signál jedinému procesu, proces zadáváme číslem úlohy nebo PID), `pkill` (podobně, jen procesy zadáváme jiným způsobem) a `killall` (hromadné posílání signálu), některé terminálové (konzolové) programy mají tuto volbu také v menu.



Příklad

Ukážeme si, jak se používají příkazy `kill`, `pkill` a `killall`. Jako parametr zadáváme identifikaci příjemce (jeho PID nebo číslo úlohy, před číslo úlohy napíšeme symbol procenta), volitelně určení signálu.

`kill -l` výpis seznamu signálů (list), což se hodí, když si nevzpomeneme na slovní ani číselné vyjádření žádaného signálu (když si nepamatujeme ani přepínač `-l`, měli bychom alespoň vědět, jak zobrazit manuálovou stránku příkazu)

`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

`pkill -HUP syslogd` pošle signál SIGHUP všem procesům s názvem `syslogd`, což znamená, že tento démon načte znovu své konfigurační soubory

`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

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

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

- přihlásíme se znovu, třeba v jiné konzoli,
- 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.

Jinou možností by bylo prostě systém vypnout, ale to někdy není možné (například servery nevypínáme).





Úkol

Zadejte postupně tyto příkazy:

- `cat &`
- `rev &`
- `yes > /dev/null &`

Příkazem `ps` (bez parametrů) si nejdřív zjistíte, jaká jsou PID spuštěných procesů. Pak si příkazem `jobs` ověřte, jaká jsou čísla těchto úloh.

Ted' začněte úlohám/procesům posílat signály (používejte čísla úloh nebo PID, obojí už znáte, pozor na zápis):

- Úloze spuštěné příkazem `yes` pošlete signál na pozastavení (SIGSTOP). Pozor na reprezentaci signálu (rozlišují se velká/malá písmena, parametr je `-STOP`). Pak si příkazem `jobs` ověřte, jestli je opravdu pozastavená.
- Následně tutéž úlohu signálem SIGCONT uveďte do provozu a opět zkontrolujte, jestli se to povedlo. Pokud si nejste jisti názvy nebo čísla signálů, použijte variantu příkazu `kill -l` pro výpis (list) seznamu signálů.
- Příkazem `kill` ukončete úlohu z příkazu `cat` bez udání signálu, tentýž příkaz použijte na úlohu `rev` se signálem SIGTERM.
- Ukončete také zbývající úlohu.



2.5 Plánování spuštění procesů

Plánovat spuštění procesů lze dvěma různými mechanismy – *at* a *cron*. První z nich je jednodušší a slouží spíše k jednoduchému jednorázovému plánování spuštění úlohy (například za hodinu nebo stanovený den v určitou hodinu), druhý má mnohem širší možnosti. Můžeme se setkat s tím, že *at* není nainstalován, což lze celkem jednoduše zajistit, systém nám dokonce při pokusu o spuštění poradí jak – na obrázku 2.1 vidíme ukázkou. Mechanismus *cron* naopak nainstalován bývá, jen pro daného uživatele nemusí být zrovna používán, reakci systému taktéž vidíme na obrázku 2.1.

```
student@student-VirtualBox:~$ at -l
Příkaz „at“ nebyl nalezen, ale je možné ho nainstalovat pomocí:
sudo apt install at

student@student-VirtualBox:~$ crontab -l
no crontab for student
student@student-VirtualBox:~$ crontab -e
no crontab for student - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano          <---- easiest
 2. /usr/bin/vim.tiny
 3. /bin/ed
Choose 1-3 [1]:
```

Obrázek 2.1: Pokud o spuštění nenainstalovaného příkazu `at`

Nejdřív se podíváme na jednodušší mechanismus. Příkaz `at` se používá pro naplánování spuštění programů nebo skriptů v zadaném čase, aby plánování fungovalo, musí být spuštěn démon `atd`.



Příklad

Používání příkazu `at` si ukážeme na příkladech.

`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 4)

`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

Ukážeme si „souvislou“ 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 li-
                    nuxové 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 ad-
    resáři, 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. Dokáže plánovat nejen jednorázově, ale také pro pravidelné spouštění podle zadaného časového plánu.

Démon, který sleduje čas a spouští naplánované úlohy, se nazývá `crond` (aby mechanismus byl použitelný, tento démon musí být spuštěn). Podíváme se na základní syntaxi:

`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 `/etc/crontab` nebo v některém souboru adresáře `/etc/cron.d` nebo některém z dalších souborů určujících frekvenci provádění, celkový přehled:

- `/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

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
```





Úkol

Ověřte si, zda je spuštěn některý plánovací démon (`atd` nebo `crond`). Pro ten mechanismus, který máte spuštěn, zkontrolujte naplánované úlohy.



Řízení přístupu a správa uživatelů

3.1 Uživatelé a skupiny

Ze zimního semestru víme, že v UNIXových systémech existují uživatelé a skupiny, jak fungují přístupová oprávnění, jak se reprezentují (buď ve formě řetězce nebo jako číselný mód), a že procesy mají přiřazeného tzv. efektivního uživatele, jehož přístupová oprávnění používají.

3.1.1 Příkazy pro zjištění informací o uživateli

Informace o uživateli se dají zjistit více různými příkazy.

 **Příkaz users** vypíše seznam *přihlášených* uživatelů, přičemž v tom seznamu bychom měli být i my. Nic jiného se nevypisuje, ale je možné, že tentýž uživatel bude vypsán vícekrát (je přihlášen na různých konzolách, případně má otevřeno víc oken terminálu).

 **Příkaz who** vyrobí výpis aktivních uživatelů (právě přihlášených podobně jako u předchozího příkazu), a to včetně různých dalších informací. Má různé parametry, kterými určujeme, co konkrétně a v jaké formě má být vypsáno.

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

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

Výpisem programu **who** je tabulka přihlášených uživatelů s některými údaji, přičemž jeden ze sloupců je terminál, ze kterého se uživatel přihlásil:

- Pokud je v tomto sloupci řetězec **tty** následovaný nějakým číslem (například **tty2**), znamená to, že uživatel je přihlášen lokálně v konzoli, v textovém režimu (buď není spuštěno GUI nebo použil klávesovou zkratku pro přesun do textového režimu, v případě **tty2** například **Ctrl+Alt+F2**).
- Řetězec **pts** následovaný lomítkem a číslem (například **pts/2**) znamená, že uživatel je přihlášen lokálně v grafickém režimu a pracuje v některé aplikaci emulující terminál (XTerm, Terminal, gnome-terminal, Konsole, atd.), nebo je přihlášen vzdáleně a v systému funguje přes nějakého démona využívajícího protokol pro vzdálený přístup (například OpenSSH pro protokol **ssh**, nebo v horším případě program **telnet** využívající stejnojmenný protokol).

Pokud je uživatel připojen vzdáleně, ve sloupci určujícím datum přihlášení bude „přilepen“ i údaj o tom, odkud (z jaké adresy) se připojil; podle toho na první pohled poznáme vzdálené přihlášení.

 Předchozí příkazy „odněkud tahají“ informace o přihlášených uživateli. Tyto informace jsou ve dvou souborech:

- `/var/log/wtmp`
- `/var/run/utmp`

Do prvního z nich se zaznamenává opravdu každé přihlášení, a to i v nedávné historii, ať už proběhne jakoukoliv metodou, lokálně či vzdáleně, a taky jakoukoliv další aktivitu v daném terminálu/konzoli. Druhý pouze sleduje, kde je kdo právě přihlášen, a ne každá přihlašovací metoda ho používá. Programy většinou používají soubor `utmp`, ale můžeme si vynutit podrobnější výstup ze souboru `wtmp`, například u příkazu `who`:

```
who /var/log/wtmp
```

 Přepínač `-b` slouží ke zjištění, kdy systém startoval (bootoval).

 **Příkazem `w`** zjistíme, kdo je přihlášen a co konkrétně dělá, jako parametr můžeme (nemusíme) zadat přihlašovací jméno uživatele, jehož činnost nás zajímá. Je zvláštní, že sice jde o zkratku z příkazu `who`, ale výstup může být podrobnější (více různých sloupců) a parametry jsou jiné.

 **Příkaz `whoami`** vypisuje informace o přihlášeném uživateli (přihlašovací jméno); v případě, že „měníme identity“ (například používáme příkaz pro získání oprávnění uživatele `root`), se takto dozvíme, pod jakým přihlašovacím jménem právě pracujeme.

 **Příkaz `id`** vypíše identitu uživatele, přičemž opět máme k dispozici různé přepínače.

`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 vy-psaných informací)

Příklad

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:

```
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 než sebe samotného, zadali bychom jako parametr název tohoto uživatele.

Dále chceme vědět, kteří uživatelé jsou právě přihlášení:

```
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:

```
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 náv kód=0
	úroveň běhu 5	2012-04-03 10:28			minulá=S	
		2012-04-03 10:28		2136	id=15	signál=0 náv kód=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 6.1.3 na straně 117), to znamená víceuživatelský režim pro běžný provoz. Na konzolách 1–6 není nikdo přihlášen (je tam pouze přihlašovací výzva), na konzoli 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 zjistím, „kdo jsem“:

```
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:

```
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:

```
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**

Podíváme se na výstupy příkazů v novějším systému (Linux Mint, rok 2016):

```
users
```

```
sarka sarka sarka sarka
```

Na výstupu se opakuje tentýž uživatel vícekrát, protože je přihlášen na více terminálech, což vidíme i na výstupu následujícího příkazu:

```
who
```

```
sarka    tty8      2017-04-18 15:07 (:0)
sarka    pts/1     2017-04-18 15:08 (:0.0)
sarka    pts/2     2017-04-18 15:11 (:0.0)
sarka    pts/3     2017-04-18 15:24 (:0.0)
```

Pokud použijeme parametr `-a`, je výstup podrobnější, najdeme tam i řádky pro terminály, na kterých není nikdo přihlášen, a taky další sloupce včetně PID hlavního procesu na daném terminálu:

```
who -aH
```

JMENO	TERMINAL	CAS	ZAHALI	PID	KOMENTAR	UKONCENI
	start systému	2017-04-18 15:06				
	úroveň běhu 2	2017-04-18 15:06				
LOGIN	tty4	2017-04-18 15:06		900	id=4	
LOGIN	tty5	2017-04-18 15:06		902	id=5	
LOGIN	tty2	2017-04-18 15:06		909	id=2	
LOGIN	tty3	2017-04-18 15:06		910	id=3	
LOGIN	tty6	2017-04-18 15:06		913	id=6	
LOGIN	tty1	2017-04-18 15:07		1809	id=1	
sarka	+ tty8	2017-04-18 15:07	start	2031	(:0)	
sarka	+ pts/1	2017-04-18 15:08	00:09	2517	(:0.0)	
sarka	+ pts/2	2017-04-18 15:11	00:23	2517	(:0.0)	
sarka	+ pts/3	2017-04-18 15:24	.	2517	(:0.0)	

A teď jiný program, jehož výstup zahrnuje kromě jiného také proces, který z daného shellu právě dostává procesor:

```
w
```

```
15:33:23 up 26 min,  4 users,  load average: 0,00, 0,02, 0,13
USER    TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
sarka   tty8      :0               15:07   26:38  14.72s  0.15s  mate-session
sarka   pts/1     :0.0            15:08   42.00s  0.08s  0.00s  pager -s
sarka   pts/2     :0.0            15:11   14:57   0.10s  0.03s  pager -s
sarka   pts/3     :0.0            15:24    1.00s  0.06s  0.01s  w
```

Vidíme, že na terminálu `tty8` (což je osmá konzola, na které v tomto systému běží GUI) je aktivním procesem `mate-session` (správce relací v prostředí MATE), na dalších dvou terminálech (virtuálních, v okně uvnitř GUI) je stránkovací program, zřejmě spolupracující s procesem `man` nebo jiným pracujícím s textovými výstupy (ale to zde nepoznáme, vypisuje se jen proces, který právě využívá procesor).

Zbývá výpis identifikátorů souvisejících s přihlášeným uživatelem:

```
id
```

```
uid=1000(sarka) gid=1000(sarka) skupiny=1000(sarka),4(adm),24(cdrom),...
```

(výstup byl zkrácen)



**Příklad**

Předpokládejme, že pracujeme v grafickém prostředí (KDE) se spuštěnou konzolou, což celkem ani nemá vliv, dále na prvním terminálu (`tty1`, `Ctrl+Alt+F1`) jsme přihlášení, ale právě nemáme spuštěn žádný interaktivní příkaz, na druhém terminálu jsme přihlášení a máme spuštěn příkaz `cat` a na třetím terminálu jsme také přihlášení 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).



Příkaz `last` sice taky vypisuje údaje týkající se přihlašování uživatelů, ale nezajímá se jen o momentálně přihlášené uživatele. Používá informace ze souboru `/var/log/wtmp`. Vypisuje seznam za určitý počet uplynulých dnů, i když v té době došlo k mnoha restartům. Užitečné je, že u jednotlivých uživatelů vypisuje i veškeré časové údaje – čas přihlášení, odhlášení, dobu trvání.

Je možné se také napojit na soubor `/var/log/btmp` s chybnými (bad) pokusy o přihlášení, případně můžeme přímo použít příkaz `lastb`.



Příkaz `getent` má poněkud širší možnosti použití než jen zjišťování informací o uživateli a skupinách. Je to nástroj pro přístup do jednoduchých lokálních nebo síťových databází (třeba reprezentovaných textovým souborem s řádky odpovídajícími jednotlivým záznamům). Základní syntaxe: `getent databáze hledaný záznam`

**Příklad**

Například pokud chceme zjistit informace o uživateli `novak` v lokálním systému, budeme přistupovat k databázi `/etc/passwd` takto:

```
getent passwd novak
```

Dotaz na skupinu `mail` bude vypadat takto:

```
getent group mail
```

Výsledek bude v případě lokálního dotazu podobný, jako když použijeme příkaz `grep` na příslušný soubor, v našich případech:

```
grep novak /etc/passwd
grep mail /etc/group
```



Úkoly

1. Srovnajte výpis příkazů `users`, `who`, `w`, `whoami` a `id`, 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 manuálovou stránku příkazu `who` a podívejte se, jaké přepínače se dají použít. Některé vyzkoušejte.
3. Procvičte si práci s terminály v konzolách – zkratkou `Ctrl+Alt+F1` se přesuňte na `tty1`, přihlaste se (pokud to jde) a příkazem `tty` zjistíte 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 několika z terminálů v konzolách (tj. v textovém režimu) 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`.

Odhláste se příkazem `logout` (pokud jste se na terminál přihlásili, je dobré se i odhlásit, pokud hned systém nevypínáte).



3.1.2 Soubory a adresáře související s uživateli

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 žádný domovský adresář nemají.

Podíváme se na některé důležité soubory.



Soubor `/etc/passwd` obsahuje seznam uživatelů, kteří se mohou do systému přihlásit. Formát řádku je

```
login:heslo:UID:GID:řetězec GECOS:homedir:shell
```

přičemž jednotlivé položky na řádku znamenají:

- přihlašovací jméno,
- buď heslo, nebo jeho hash, nebo (pravděpodobněji) jenom písmeno `x`, což znamená, že heslo (přesněji jeho hash) je v souboru `/etc/shadow`,
- číslo UID uživatele a číslo GID jeho hlavní skupiny,
- v poli GECOS (General Electric Comprehensive Operating System) obvykle najdeme plné jméno uživatele, ale mohou tam být i další osobní údaje oddělené čárkami – pracoviště, telefon, atd.,
- domovský adresář uživatele, pokud existuje,
- hlavní shell uživatele, u běžných uživatelů to je většinou `/bin/bash`.

 **Soubor** `/etc/group` obsahuje seznam definovaných skupin. Formát řádku (záznamu) je `login:heslo:GID:uživatelé`

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/gshadow`. GID je číslo skupiny, a jako poslední údaj na řádku je seznam uživatelů do skupiny zařazených, přičemž primární uživatel skupiny (jehož uživatelské jméno je shodné s názvem skupiny) obvykle není uveden.

Proč není u skupiny uveden primární (nej důležitější) uživatel/člen? Protože vztah mezi skupinou a jejím primárním členem najdeme v souboru `/etc/passwd` – každý uživatel, který může být členem nějaké skupiny, má číslo GID své primární skupiny ve čtvrtém poli svého řádku.

Poznámka:

Původně byly hashe hesel ukládány přímo do souborů `/etc/passwd` a `/etc/group`. To se změnilo nejen z bezpečnostních důvodů, ale i proto, že ve „stínových“ souborech mohou být u hesla i další užitečné informace jako je třeba stáří hesla (kdy naposledy bylo změněno).

 **Soubory** `/etc/shadow` a `/etc/gshadow` obsahují šifrovaná hesla (hashe) uživatelů a skupin, a pak další související informace. Běžný uživatel tento soubor neotevře ani pro čtení, tj. pro prohlížení potřebujeme práva roota.

Příklad

Pokud máme přístup k vyšším oprávněním (typicky na vlastním zařízení), bude fungovat buď

```
sudo cat /etc/shadow
```

nebo

```
su
```

```
cat /etc/shadow
```

 Formát řádku v souboru `/etc/shadow` je následující:

`login:hash hesla:čas poslední změny:min:max:varování:inactive:expire:rezervováno`

přičemž nás zajímají především první dvě položky. U systémových účtů používaných hlavně démony bývá místo hashe hesla symbol *. Další pole mají tento význam:

- **třetí pole čas poslední změny** určuje, kdy naposledy bylo heslo změněno,
- **min** vynucuje minimální počet dnů, kdy se po změně heslo nesmí znovu měnit,
- **max** stanovuje, jak maximálně dlouho může heslo být používáno (do té doby si je uživatel musí změnit, jinak mu bude deaktivován účet),
- **varování** – pokud je toto pole použito, při překročení **max** není účet deaktivován, ale uživatel je varován, že musí heslo změnit (je zde počet dnů, které zbývají do deaktivace, a tedy po které se uživateli bude zobrazovat varování),
- **inactive** – když to uživatel ignorováním času na změnu hesla „dotáhl“ k deaktivaci účtu, po určitý počet dnů (který je zde stanoven) má možnost to napravit: bude mu dovoleno se přihlásit, pokud si během přihlašování změní heslo (jinak neprojde přes přihlašovací proces), jeho účet je zatím „jen“ neaktivní,
- **expire** – zde je stanovena doba, do které má být účet platný (například u dočasných zaměstnanců).

**Poznámka:**

V některých polích (v třetím a možná v předposledním) je uložen časový údaj. Tento údaj však není v člověkem čitelné podobě, ale jde o číslo znamenající počet dnů uplynulých od data 1. ledna 1970 (to je oficiální datum vzniku první verze UNIXu), tedy od začátku hlavní „epochy“. Je to číslo, které může být buď menší (počet dnů od daného data) nebo větší (počet sekund od daného data).

Převod na skutečný počet let, měsíců a dnů (případně až sekund) bychom teoreticky mohli zvládnout z paměti, ale naštěstí k tomu máme nástroje, jeden z nich je na webu

<https://www.epochconverter.com/>

V horní části stránky jsou převody až k sekundám (epocha ve formě většího čísla), o něco dále převody do řádu dnů (epocha ve formě menšího čísla, které bychom využili i ve „stínových“ souborech).



Adresář `/etc/skel` je vzorovým adresářem („kostrou“) při vytváření domovských adresářů pro nové uživatele, takže když chceme, aby něco konkrétního bylo v domovských adresářích a konfiguračních souborech nově vytvářených uživatelů, změnu provedeme zde.



Soubor `/etc/default/useradd` obsahuje výchozí parametry, které se použijí při vytváření nového uživatele, pokud ručně v parametrech příkazu neurčíme jinak. Jedním z těchto parametrů může být i název adresáře, který se má použít jako „kostra“ pro domovský adresář uživatele (obvykle to je `/etc/skel`, ale můžeme ho nazvat i jinak). Na soubor `/etc/default/useradd` se podíváme v sekci, ve které probíráme vytváření nového uživatele.



Soubor `/etc/login.defs` obsahuje výchozí nastavení bezpečnostního charakteru, například je zde možné nastavit metodu šifrování hesel (DES, MD5, SHA256, SHA512 apod.), 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).

**Úkol**

Projděte si obsah všech konfiguračních souborů a adresářů zde zmíněných, pokud to dovolují vaše přístupová oprávnění. Zaměřte se nejen na údaje o svém účtu, skupině a hesle, ale také na údaje o rootovi a systémových účtech.



3.1.3 Práce s uživatelskými skupinami



V UNIXových systémech existují dva přístupy pro práci s uživatelskými skupinami:

- Klasický systém práce se skupinami znamená, že pro běžné uživatele máme jednu společnou skupinu, obvykle pojmenovanou `users`, tato skupina je primární skupinou pro běžné uživatele.
- Systém `UPG` (User Private Group) je dnes běžný ve většině distribucí Linuxu. Při vytváření uživatele je vytvořena stejně pojmenovaná skupina, tento uživatel je primárním (obvykle také

jediným) členem této skupiny a tato skupina je jeho primární skupinou. Nicméně jeden uživatel může patřit do více dalších (již specializovaných) skupin.

 Uživatel obvykle patří do více než jedné skupiny, ale jedna z nich je vždy *aktuální*. Víme, že u souborů se přístupová oprávnění určují ve třech oblastech: pro vlastníka/členy přidružené skupiny/zbytek světa, a změnou aktuální skupiny můžeme ovlivnit udělení oprávnění v druhé z těchto oblastí.

Příklad

V příkladech na stranách 51 a 53 je ve výstupu příkazu `id` vidět určitá odlišnost mezi starším a novějším systémem. U toho prvního vidíme ve výstupu

```
uid=1000(sarka) gid=100(users) skupiny=...
```

u druhého systému jsou tyto údaje:

```
uid=1000(sarka) gid=1000(sarka) skupiny=...
```

Takže první systém používá klasické skupiny, kdy běžní uživatelé jsou shrnuti do jedné skupiny (pojmenované `users`), kdežto druhý systém používá skupiny podle UPG – každý uživatel má vlastní stejně pojmenovanou skupinu, také číslo GID jeho skupiny obvykle souhlasí s číslem UID.



 Pro název existují určité konvence – první znak může být buď malé písmeno nebo podtržítka, pak mohou následovat malá i velká písmena, číslice, podtržítka, pomlčky. Pomlčka nesmí být posledním znakem. Nicméně se doporučuje zůstat jen u malých písmen. Existuje také limit pro délku názvu skupiny: ve většině linuxových distribucí to je 32, některé UNIXové systémy a některé linuxové distribuce mají limit 16.

 **Příkaz `groupadd`** slouží k vytvoření nové skupiny. Povinným parametrem je název nové skupiny, volitelně můžeme určit její GID (pokud neurčíme, vybere se první „volné“). Plný tvar příkazu je `groupadd -gid číslo skupina`

Poznámka:

V systému existuje hranice pro čísla UID a GID taková, že čísla menší než tato hranice jsou určena pouze pro systémové uživatele, kdežto čísla nad touto hranicí (včetně) pro běžné uživatele. Záleží na konkrétním systému, obvykle je touto hranicí 500 nebo 1000.



 **Příkaz `groupdel`** naopak slouží k likvidaci existující skupiny. Jediným parametrem je název skupiny, kterou chceme smazat:

```
groupdel skupina
```

Nelze smazat skupinu, která je primární skupinou některého uživatele, tj. předem bychom takové uživatele museli přeradit do jiných skupin. Pozor – pokud existují soubory, kde je daná skupina přidruženou skupinou, stanou se z takových souborů „skupinová sirotci“.

 **Příkaz `newgrp`** rozhodně není určen k vytváření skupin, jeho účelem je změnit aktivní skupinu uživatele. Pokud z nějakého důvodu potřebujeme získat oprávnění příslušející jiné než naší primární skupině (tj. potřebujeme změnit aktivní skupinu), použijeme tento příkaz:

```
newgrp skupina
```

(samozřejmě musíme být členem i této skupiny, jinak by příkaz vypsal chybové hlášení).

Zpět ke své primární skupině se dostaneme příkazem `exit`, protože ve skutečnosti se změnou aktivní skupiny vlastně spouští nový podržený shell, příkazem `exit` tento podržený shell ukončíme a vrátíme se do původního.

 **Příkaz** `groupmod` slouží k modifikaci vlastností (názvu nebo čísla) skupiny – „mod“ znamená modify. Povinným parametrem je původní název skupiny. Také musíme zadat nový název nebo nové GID dané skupiny.

Pokud chceme nastavit nový název, napíšeme před ten nový název přepínač `-n`, když GID, použijeme přepínač `-g`:

```
groupmod -n nový_název starý_název
groupmod -g nové_gid název
```



Příklad

Předpokládejme, že máme skupinu `udrzba` a je to přidružená skupina souboru `test.sh`. Pak výpis vlastností tohoto souboru pomocí příkazu `ls -l test.sh` vypadá následovně:

```
rwxr-xr-x 1 novak udrzba 37 dub 18 16:25 test.sh
```

(tj. v poli pro přidruženou skupinu je napsáno `udrzba`). Pokud změním název skupiny `udrzba` na `testeri`, zadáme:

```
groupmod -n testeri udrzba
```

Výpis vlastností souboru se nám změní takto:

```
rwxr-xr-x 1 novak testeri 37 dub 18 16:25 test.sh
```

Skupina se z hlediska souboru `test.sh` nezměnila (protože má pořád stejné GID, a to je evidováno), jen se změnil její název, ale to se projevuje pouze ve výpisu, nikoliv v evidenci vlastností souboru.



Poznámka:

Pokud změním název skupiny, nemá to obvykle žádné negativní následky na soubory, které mají tuto skupinu přiřazenu co se týče přístupových oprávnění, protože v souborovém systému jsou u souborů přidružené skupiny evidovány pomocí čísel GID.

Naopak – pokud změním GID skupiny, nastává problém u všech souborů, které mají tuto skupinu přidruženou. Takové soubory se stanou „skupinovými sirotky“, stejně jako v případě, kdy by ta skupina byla smazána. Proto bychom si měli podobné akce předem důkladně rozmyslet.



Pokud máme podezření, že v souborovém systému existují soubory, které nemají přidruženou žádnou skupinu (ať už po smazání skupiny nebo změně GID přidružené skupiny), najdeme je tímto příkazem:

```
find / -nogroup
```



Úkoly

1. Zjistěte ze souborů `/etc/passwd` a `/etc/group`, zda vaše distribuce používá systém správy skupin UPG. Dále zjistěte, jaká je číselná hranice mezi identifikátory systémových a běžných účtů a skupin.

2. Pokud máte možnost, vyzkoušejte si vytváření, rušení a přejmenování skupin.
3. Zjistěte, do kterých skupin patříte (třeba příkazem `id`). Pak si vyzkoušejte změnu aktivní skupiny.



3.1.4 Práce s uživatelskými účty

 Při vytváření nového uživatelského účtu lze buď zadat všechny parametry přímo do příkazu, anebo 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. Obvykle je to adresář `/etc/skel`, o kterém byla řeč v předchozím textu.

Některé položky se vztahují k obsahu souboru `/etc/passwd`, například GID hlavní skupiny uživatele (ovšem většina linuxových distribucí používá systém UPG, proto se tato položka nepoužívá), další skupiny, umístění domovského adresáře (pozor – ne název domovského adresáře, ten musí být stejný jako přihlašovací jméno), shell.



Poznámka:

Většina řádků v souboru `/etc/default/useradd` bývá ve skutečnosti zakomentovaná (buď „slovní“ komentáře nebo zakomentované položky, které místo dopisování můžeme jednoduše odkomentovat, či ukázky), protože i samotný příkaz pro přidávání nových uživatelů bere určité hodnoty jako výchozí (tj. do tohoto souboru připsujeme jen to, u čeho chceme změnit výchozí chování příkazu).



Protože čísla UID bývají do určité hranice (čísla menší než 500 nebo 1000) vyhrazena systémovým uživatelům (root, ale také některé speciální procesy, démoni, mají své UID z důvodu vyšších přístupových práv), volíme UID vždy větší nebo rovno dané hranici (a samozřejmě takové číslo, které ještě není vyhrazeno pro jiného uživatele), nicméně obvykle býváme volby ušetření, příkaz sám dokáže najít a zvolit vhodnou volnou hodnotu.

 **Příkaz `useradd`** slouží k vytváření uživatelských účtů. Musíme zadat především parametr s přihlašovacím jménem uživatele, to bude poslední parametr na řádku, volitelné jsou další parametry (vpodstatě to, co se má uložit do řádku v souboru `/etc/passwd`).

Pro názvy uživatelských účtů platí podobná pravidla jako pro názvy skupin – první znak má být malé písmeno nebo podtržítka, následovat mohou písmena, číslice, podtržítka, pomlčky, pomlčka nesmí být posledním znakem názvu.



Příklad

Pár ukázek použití příkazu `useradd` včetně některých parametrů a přepínačů (každý typ údaje má svůj přepínač):

```
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 "Jan Novák" novak  některé volby zadáváme
                    ručně (domovský adresář, GID, shell a komentář), zbylé položky budou brány buď ze souboru
                    /etc/default/useradd nebo budou použity implicitní volby příkazu

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



Příkaz `userdel` dokáže odstranit uživatelský účet. Buď odstraníme jen samotný účet, nebo si vynutíme i smazání domovského adresáře souvisejícího s tímto účtem. Bezpečnější je tento domovský adresář zatím nechat, zlikvidovat se dá kdykoliv jindy (mohou tam být důležité dokumenty, taky pro případný forenzní audit).

Pokud chceme domovský adresář zatím nechat, použijeme jako parametr jen přihlašovací jméno uživatele, jehož účet rušíme. Pokud chceme zlikvidovat domovský adresář, použijeme přepínač `-r`, případně taky můžeme přidat přepínač `-f`, který nebude varovat při problémech (v domovském adresáři jednoho uživatele například mohou existovat i soubory vlastněné jiným uživatelem, a je otázkou, zda se mají taky odstranit). Takže u „přísnější“ varianty bychom napsali

```
userdel -rf účet.
```



Příkaz `usermod` slouží ke změně vlastností uživatelského účtu. Na rozdíl od skupin je zde víc možností ke konfiguraci. Můžeme nastavovat jednotlivé parametry podobně jako u vytváření nového uživatele (například měnit hlavní skupinu, přidávat či odebírat členství v dalších skupinách, měnit řetězec GECOS, domovský adresář, shell, dokonce i přihlašovací jméno nebo UID), a také můžeme účet (de)aktivovat či uzamknout či odemknout možnost změny hesla.



Příklad

Možnosti příkazu `usermod` jsou výrazně rozsáhlejší než u dříve probíraného příkazu `groupmod`. Následují ukázky nejběžnějších operací prováděných s tímto příkazem.

```
usermod -c "Jana Nováková" novakova   změna komentářového řetězce účtu

usermod -l janackova novakova   uživatelka novakova změnila příjmení, tak tomu přizpůsobíme při-
                                hlašovací jméno (login, proto přepínač -l), změníme ho na janackova

usermod -g users novak   primární skupinou uživatele novak se tímto stává skupina users

usermod -aG marketing novak   uživatele novak jsme zařadili do skupiny marketing (ale nebude to
                                primární skupina tohoto uživatele, bude to jedna z dalších jeho skupin) – práci se sekundárními
                                skupinami zvládá přepínač -G, přepínač -a způsobí, že provedeme přidání (append) zadané sku-
                                piny do seznamu skupin uživatele a nikoliv nahrazení původního seznamu skupin

usermod -s /bin/tcsh novak   uživateli novak jsme určili shell /bin/tcsh

usermod -L novak   (nebo --lock) uzamkne heslo k účtu, uživatel si nebude moci změnit heslo (k
                    hashi hesla do /etc/shadow se přidá symbol vykřičníku)
```

`usermod -U novak` (nebo `--unlock`) odemkne heslo účtu

`usermod -e 15092 novak` (nebo `--expiredate`) nastaví se datum expirace účtu (po tomto datu bude účet deaktivován); datum se stanovuje jako číslo znamenající počet dnů od 1. 1. 1970 (viz str. 57)

`usermod -e 1 novak` datum expirace (ukončení platnosti) účtu jsme nastavili na 2. 1. 1070, čímž jsme tento účet znepřístupnili (ale není smazán, může být po úpravě data expirace znovu používán)

`usermod --inactive 3 novak` pokud uživatel nezmění včas heslo (pokud je vyžadována jeho pravidelná změna), pak po uplynutí doby platnosti hesla má 3 dny „hájení“ účet sice nebude přímo přístupný, ale bude neaktivní (při pokusu o přihlášení musí uživatel nastavit nové heslo, jinak se nepřihlásí – po změně hesla se účet aktivuje)

Pozor – příkaz `usermod -G marketing novak` (tj. bez přepínače `-a`) by provedl jednu nepěknou věc: sice by zařadil daného uživatele do skupiny `marketing`, ale zároveň by ho odstranil z jakékoliv jiné skupiny.



Poznámka:

Příkaz `usermod` sice umí změnit UID uživatele, ale uvědomte si, co taková změna znamená pro soubory, u nichž je daný uživatel vlastníkem: tyto soubory se stanou opravdovými „sirotky“ (což je ještě horší než být „skupinovým sirotkem“ bez přidružené skupiny), budou bez vlastníka. Ve struktuře obsahující informace o souboru je totiž vlastník souboru zadán pomocí UID, takže změna přihlašovacího jména uživatele (loginu) není problém, ale změna jeho UID ano.



Příkaz `newusers` slouží k hromadnému přidávání nových uživatelů. V případě, že musíme do systému přidat větší množství nových uživatelů, nemusíme je postupně jednoho po druhém zadávat pomocí příkazu `useradd` (ani psát skript, který by to udělal za nás), ale postupujeme takto:

- vezmeme textový soubor obsahující seznam uživatelů, které máme přidat do systému, formát toho souboru musí být v takovém formátu, jako je `/etc/passwd` (tj. řádky by měly být ve formátu `login:heslo:UID:GID:řetězec GECOS:homedir:shell`, ale heslo nešifrované),
- tento soubor použijeme jako parametr příkazu `newusers`,
- původní soubor se seznamem uživatelů znepřístupníme (smažeme nebo bezpečně zašifrujeme a upravíme přístupová oprávnění), protože v souboru jsou hesla uživatelů v nešifrované podobě.



Příklad

Hromadné přidání nových uživatelů trochu podrobněji:

Vytvoříme textový soubor, kde každý řádek bude patřit jednomu uživateli. Formát je stejný jako u řádků v souboru `/etc/passwd`. Buď tento soubor vytvoříme ručně, nebo použijeme některou automatizovanou metodu (původní zdroj můžeme zpracovat třeba pomocí Perlu nebo jiného programu, či v tabulkovém editoru a exportovat jako text do souboru csv s dvojtečkou jako oddělovačem polí). Na řádku pro každého uživatele budou postupně tyto údaje:

- 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 – prostě budou dvě dvojtečky za sebou),
- 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 (pokud používáme systém skupin UPG, taky necháme prázdné),
- 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, například

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

Tento soubor použijeme jako parametr příkazu `newusers`. Například pokud jsme soubor pojmenovali `uzivatele.t`, příkaz vypadá takto:

```
newusers uzivatele.t
```



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

Úkol

Pokud máte dostatečná přístupová oprávnění, vyzkoušejte si vytvoření nového uživatele, nastavení některých vlastností jeho účtu a následně jeho zrušení.



3.1.5 Vlastnosti hesla

 Program `passwd` je jedním z nejdůležitějších příkazů pro nastavení vlastností hesel uživatelských účtů. Nejčastěji je používán ke změně hesla, ale také dalších informací souvisejících s heslem účtu (například lze zadat dobu platnosti hesla), příkaz může být používán buď majitelem účtu nebo rootem. Můžeme nastavovat především údaje ze souboru `/etc/shadow`.

Běžný uživatel může pomocí tohoto příkazu nastavovat jen své vlastní heslo, root může nastavovat heslo k jakémukoliv účtu. Možnosti si ukážeme opět na příkladu.

Příklad

Ukážeme si používání příkazu `passwd`:

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

```
passwd -S novak    (nebo --status) vypíše nastavení (stav) účtu uživatele (zda má účet zamknuté
                    heslo, je bez hesla nebo s heslem, datum poslední změny hesla, jak často musí být heslo měněno
                    apod.), v podstatě se jedná o obsah řádku ze souboru /etc/shadow
```

```
passwd -x 90 novak    (nebo --maxdays) nastavíme maximální stáří (dobu platnosti) hesla na 90 dnů;
                      do té doby by si uživatel měl změnit heslo
```

`passwd -w 10 novak` (nebo `--warndays`) pokud je nastavena doba platnosti hesla, 10 dnů před vypršením doby platnosti se uživateli bude při přihlašování zobrazovat varování o nutnosti změnit heslo

`passwd -i 8 novak` (nebo `--inactive`) pokud si uživatel nestihl změnit heslo včas (když je nastavena doba platnosti hesla), po vypršení doby platnosti se heslo na 8 dnů dostává do stavu neaktivního; po tuto dobu ještě má uživatel možnost napravit svou chybu a změnit si heslo (účet se zaktivní hned po změně hesla), přičemž změnu musí provést během přihlašování (pokud nezmění heslo, nesmí se přihlásit)

`passwd -n 5 novak` (nebo `--mindays`) pokud si uživatel změnil heslo, během příštích 5 dnů si je nebude moci změnit znovu

`passwd -e novak` (nebo `--expire`) okamžité „vypršení“ hesla; uživatel je přinucen změnit heslo při příštím spuštění (používá se často u nově vytvořených účtů, aby si uživatel vytvořil heslo hned při prvním přihlášení)

`passwd -d novak` (nebo `--delete`) smaže heslo zadaného uživatele, tedy účet bude od této chvíle bez hesla

`passwd -l uživatel` (nebo `--lock`) zamkne heslo, uživatel se nebude moci přihlásit zadáním hesla (ale jiné formy přihlašování budou fungovat)

`passwd -u uživatel` (nebo `--unlock`) odemkne heslo



Poznámka:

Funkčnost příkazů `passwd` a `usermod` se částečně překrývají (například pomocí obou se dá uzamknout heslo účtu nebo nastavit doba expirace), je víceméně jedno, který příkaz v těchto případech použijeme. Jen pozor na to, že ne vždy se pro totéž používá jako přepínač stejné písmeno, případně je rozdíl malé/velké písmeno – například pro uzamknutí hesla to je buď `-L` (pro `usermod`) nebo `-l` (pro `passwd`).



Pro práci s hesly skupin existuje příkaz `gpasswd`.



Jak víme, hashe hesel jsou uloženy jako řetězce v souboru `/etc/shadow`, v druhém poli řádku pro daného uživatele. Zamknutí hesla probíhá tak, že se na začátek tohoto řetězce přidá symbol `!`. Protože ověřování hesla při přihlašování probíhá tak, že uživatel zadá heslo, to projde šifrovacím algoritmem, jehož výsledkem je hash tohoto hesla, a ten se následně porovnává s příslušným řetězcem v souboru `/etc/shadow`, srovnání vždy selže – hash jako výstup (jakéhokoliv) šifrovacího algoritmu totiž nikdy nezačíná symbolem `!`.



Poznámka:

Pozor – existuje víc manuálových stránek s názvem `passwd`, každá z nich v jiné sekci. Pokud chceme informace o *konfiguračním souboru* `/etc/passwd` a ne o *příkazu* `passwd`, měli bychom zvolit záznam ze sekce 5 a nikoliv 1:

`man -s 5 passwd`

(bez udání sekce se zobrazuje první nalezená stránka s daným názvem, sekce 1 se prohledává první, takže příkaz `man passwd` zobrazí informace o *příkazu* `passwd`).





Úkol

Zjistěte údaje o svém účtu, které se vztahují k heslu (vypište si status svého hesla). V manuálové stránce zjistěte, co který vypsaný údaj znamená.



3.2 Přístupová oprávnění k souborům

3.2.1 Vlastník a přidružená skupina

Každý soubor má svého vlastníka a přidruženou skupinu, přičemž přístupová oprávnění se stanovují ve třech kategoriích – oprávnění pro vlastníka, pro členy přidružené skupiny a pro „zbytek světa“. Z toho vyplývá, že základním způsobem ovlivnění přístupových oprávnění k souboru je právě určení vlastníka a přidružené skupiny.

 Z předchozího semestru víme, že vlastníkem souboru je obvykle jeho tvůrce nebo u systémových souborů uživatel root. S přidruženou skupinou je to trochu složitější – když vytvoříme nový soubor (nebo adresář, ten je ve skutečnosti taky souborem), přidruženou skupinou se stane ta skupina, pod kterou zrovna pracujeme (pozor – když jsme změnili aktivní skupinu pomocí `newgrp`, bude to ta, kterou jsme si tím nastavili). Jestliže má nadřazený adresář nastaven příznak SGID, dědí nový soubor přidruženou skupinu od svého nadřazeného adresáře.

 **Příkaz** `chown` slouží ke změně vlastníka souboru (change owner). V parametrech příkazu `chown` zadáváme nového vlastníka a soubor, případně můžeme použít různé přepínače. Základní syntaxe:

```
chown uživatel /cesta/soubor
```

Změna vlastnictví může být i rekurzivní (použili bychom přepínač `-R` nebo `--recursive`, obvykle se přidává i přepínač `-h`, abychom neovlivnili cíle symbolických odkazů). Další přepínače bychom našli na manuálové stránce příkazu.

Zároveň s vlastníkem můžeme změnit i přidruženou skupinu:

```
chown uživatel:skupina /cesta/soubor
```



Poznámka:

Vlastníka souboru ale nemůže změnit jen tak někdo – pokud pracujeme pod běžným uživatelským účtem, nemůžeme dokonce ani přenechat vlastní soubor někomu jinému (předat vlastnictví), kdežto root může měnit vlastnictví souborů podle libosti.

Proč? Protože vlastnictví znamená určitou odpovědnost. Pokud by se během auditu přišlo na „závadný obsah“ (hanlivá tvrzení či jiný eticky problematický obsah, upíratěné filmy, uniklé tajné informace apod.), kdo bude hnán k odpovědnosti? Vlastník.



 **Příkaz** `chgrp` mění přidruženou skupinu souboru. Taktéž zadáváme novou skupinu a soubor. Parametry a přepínače jsou podobné jako u `chown`, včetně zajištění rekurze. Základní syntaxe je

```
chgrp skupina /cesta/soubor
```

Běžný uživatel může tento příkaz použít pouze na ty soubory, jichž je vlastníkem, a jejich přidruženou skupinu změnit jen na takovou, jejímž je členem. Root není v používání tohoto příkazu nijak omezen.



Úkol

Proveďte následující posloupnost operací:

- Pomocí příkazu `id` zjistěte, do kterých skupin patříte. Jednu z nich si vyberte.
- Ve svém domovském adresáři vytvořte nový soubor s názvem `test.txt` (je jedno jak, například příkazem `touch test.txt`).
- Zjistěte vlastníka a přidruženou skupinu tohoto souboru, třeba takto:

```
ls -l test.txt
```

(obojí by mělo být vaše, a pokud jste v systému používajícím UPG, pak vlastník i skupina budou stejný řetězec).

- Změňte přidruženou skupinu pro tento soubor na tu, kterou jste si vybrali v prvním bodu tohoto postupu:

```
chgrp xxx test.txt
```

(dosadte za `xxx` tu vybranou – opravdu by to měla být jedna z vašich skupin).

- Znovu si vypíšte vlastníka a přidruženou skupinu tohoto souboru:

```
ls -l test.txt
```

- Odstraňte soubor `test.txt` (například příkazem `rm test.txt`).



3.2.2 Změna přístupových oprávnění souboru

Z předchozího semestru víme, že přístupová oprávnění se zapisují buď řetězcem skládajícím se z písmen `r` (čtení), `w` (zápis), `x` (spouštění) a případně pomlček, nebo číselně.

Taky víme, že pro adresáře znamená oprávnění `r` možnost zobrazit si obsah tohoto adresáře (ať už v textovém režimu pomocí `ls` nebo `echo`, nebo v aplikaci v grafickém režimu), oprávnění `w` je možnost upravovat obsah adresáře (především vytvářet či mazat soubory uvnitř), oprávnění `x` dovoluje použít tento adresář jako pracovní (třeba příkazem `cd`). Pokud se chceme dostat dovnitř takového adresáře a pracovat s něčím v jeho podstromě, potřebujeme také právo `x`.



Příklad

Na zopakování:

- Řetězec `rwrx-rx-x` znamená, že vlastník má všechna oprávnění – `rw`, členové přidružené skupiny mohou číst a spouštět (u adresáře zobrazit obsah a použít ho jako pracovní) – `r-x`, pro ostatní uživatele platí totéž. Velice pravděpodobně se jedná o adresář.
Řetězec `rwrx-rx-x` se převádí na číselný mód 755 (čtyřciferně 0755).
- Řetězec `rw-rw-r--` znamená, že vlastník a členové přidružené skupiny mají právo číst a zapisovat, ostatní uživatelé mohou jen číst. Pravděpodobně to bude běžný soubor (text, obrázek, video, apod.).
Řetězec `rw-rw-r--` se převádí na číselný mód 664 (čtyřciferně 0664).
- Řetězec `rwsr-rx-x` (přísluší souboru `/usr/bin/passwd`) určuje plná oprávnění pro vlastníka (to je root), členové přidružené skupiny a ostatní mohou číst a spouštět. Místo symbolu `x` je v první

trojici symbol `s`, takže je nastaven SUID bit. Protože jde o spustitelný soubor a jeho spuštěním vznikne proces, SUID bit způsobí, že efektivním uživatelem (od kterého proces bere přístupová oprávnění) není ten, kdo soubor spustil, ale vlastník souboru (tj. dotyčný proces získá práva roota).

Řetězec `rwsr-xr-x` se převádí na číselný mód 4755.



 Při změně přístupových oprávnění souboru použijeme jeden z těchto typů reprezentace:

- *číselně* – zadáme číselný mód, a to tříciferný nebo čtyřciferný (pokud chceme nastavit i speciální bity),
- *symbolicky* – používá se hlavně pro relativní reprezentaci změn (chceme některá oprávnění přidat nebo ubrat, ale ostatní oprávnění neměníme), ale existuje i možnost absolutního tvaru (měníme vše).

Symbolický tvar se skládá ze tří částí určujících:

- komu se mají oprávnění změnit – `u`=vlastník, `g`=skupina, `o`=ostatní (others), `a`=všichni (all),
- zda se mají přidat (+), ubrat (-) nebo přímo tak nastavit (=),
- o jaká oprávnění jde – `r`, `w`, `x`.

Absolutní tvar určuje plnou změnu (přímo zadáváme výsledná oprávnění), kdežto *relativní* tvar znamená, že měníme jenom některá oprávnění a ostatní nepřepisujeme.

U první a třetí části může být i více symbolů (například můžeme oprávnění měnit vlastníkovi i skupině, nebo třeba můžeme nastavovat čtení i zápis).

 **Příkaz** `chmod` slouží k nastavení módu souboru, tedy k ovlivňování přístupových oprávnění k souboru. Oprávnění můžeme zadávat číselně i symbolicky, absolutně nebo relativně.

Posledním (a zároveň povinným) parametrem příkazu je název souboru, jehož mód chceme změnit. Před něj zadáváme, jak chceme oprávnění nastavit/změnit (tj. číselný mód nebo symbolický řetězec), a před to můžeme umístit přepínače (například `-R` určující rekurzi).



Příklad

Ukážeme si typické způsoby použití příkazu `chmod`, nejdřív u symbolického zápisu.

`chmod g+w soubor` členům přidružené skupiny jsme přidali právo zápisu

`chmod -R g+w adresář` totéž, ale rekurzivně

`chmod ug+x,o-w soubor` vlastníkovi a členům přidružené skupiny jsme přidali právo spuštění, ostatním jsme naopak ubrali právo zápisu

`chmod g=r-x,o=r soubor` členům přidružené skupiny jsme nastavili řetězec `r-x` (tj. mohou číst a spouštět), ostatním jsme nastavili řetězec `r--` (mohou jen číst, nadbytečné pomlčky zprava se nemusejí psát); použili jsme typ modifikace „=“, tedy napevno (absolutně) nastavujeme oprávnění pro skupinu a ostatní (jejich původní oprávnění budou přepsána)

`chmod u=rw,go-w soubor` kombinujeme absolutní a relativní přístup: vlastníkovi (absolutně) nastavujeme právo čtení a zápisu, členům přidružené skupiny a ostatním odebíráme právo zápisu (ale pokud dosud měli právo čtení, budou ho mít i nadále)

`chmod a-x soubor` všem (all) odebíráme právo `x` (například u souboru rušíme jeho spustitelnost)

Symbolickým způsobem můžeme nastavovat či rušit i speciální příznaky pro soubor:

```
chmod u+s soubor    souboru jsme nastavili příznak SUID
chmod u-s soubor    souboru jsme zrušili příznak SUID
chmod g+s soubor    nastavení příznaku SGID, rušení by bylo podobně, s mínusem
chmod ug+s soubor   nastavujeme příznaky SUID a SGID (v praxi se nepoužívá, ale teoreticky to je možné)
chmod +t soubor     nastavení příznaku sticky (všimněte si, že neudáváme, ve které třetině řetězce oprávnění se má změna provést – to plyne už z reprezentace příznaku písmenem t), odebrání se provede s mínusem
```

Ted' se podíváme na číselný zápis, který je v principu absolutní:

```
chmod 754 soubor    řetězec oprávnění pro daný soubor nastavujeme na rwxr-xr--
chmod 640 soubor    řetězec oprávnění pro daný soubor nastavujeme na rw-r-----
chmod 4755 soubor    řetězec oprávnění pro daný soubor nastavujeme na rwsr-xr-x (nastavujeme také SUID bit, což na první pohled poznáme podle toho, že mód je čtyřciferný a první číslice je 4)
```



Pozor, speciální příznaky se číselně sice dají nastavovat, ale ne rušit, na to potřebujeme symbolický zápis.



Příklad

Ted', když už víme, jak se pracuje s účty, skupinami a oprávněními, si na souhrnném příkladu ukážeme vytvoření pracovního prostoru pro skupinu zaměstnanců spolupracujících na projektu.

1. Přihlásíme se jako root nebo jiným způsobem získáme vyšší oprávnění.
2. Vytvoříme novou skupinu, v ní mají být zaměstnanci spolupracující na projektu:

```
groupadd nova_skupina
```

3. Uživatelé u1, u2, u3, ... mají spolupracovat na projektu, tudíž je zařadíme do nové skupiny:

```
usermod -aG nova_skupina u1
usermod -aG nova_skupina u2
...
```

4. Vytvoříme adresář, ve kterém budou uloženy soubory projektu:

```
mkdir /vol/novy_adresar
```

5. Nastavíme pro tento adresář přidruženou skupinu a vlastníka (vedoucího skupiny – uživatele u1), dále přístupová práva:

```
chgrp nova_skupina /vol/novy_adresar
chown u1 /vol/novy_adresar
```

6. Přístupová oprávnění obvykle nemusíme nijak moc řešit, budou zřejmě stačit stávající, jen „zbytku světa“ zakážeme přístup a dále pro jistotu nastavíme příznak sticky:

```
chmod o-rwx /vol/novy_adresar
chmod +t /vol/novy_adresar
```



Přístupová oprávnění pro daný soubor může měnit buď root nebo vlastník tohoto souboru.



Úkol

Zaexperimentujte si s přístupovými oprávněními tak, jak vám dovolí vaše přístupová oprávnění. V domovském adresáři si vytvořte nový soubor a přidávejte a odebírejte oprávnění především členům přidružené skupiny a ostatním. Vyzkoušejte si i přidání a odebrání některého speciálního příznaku (SUID apod.). Potom nezapomeňte experimentální soubor smazat.



3.2.3 Maska oprávnění

Když vytváříme nový soubor, pro tento soubor je třeba určit vlastníka, přidruženou skupinu a přístupová oprávnění. Kdybychom to měli provádět pokaždé, když vytvoříme nový soubor, bylo by to zbytečné zdržování. Už víme, jak funguje mechanismus určení vlastníka a přidružené skupiny (včetně role SGID bitu), zbývá nám mechanismus určení implicitních oprávnění.

 Maska oprávnění určuje implicitní oprávnění pro nově vytvářené soubory. Většinou je stanovena řetězcem 022, přičemž při vytvoření nového souboru se provádí bitová operace

maximum AND NOT *maska* (vpodstatě bitové odečtení)

kde *maximum* je maximální hodnota pro daný typ souboru (což je 777 pro adresáře a 666 pro ostatní soubory). Masku nám tedy říká, „která oprávnění chceme odebrat z maxima“.



Příklad

Pokud je tedy maska stanovena na 022, implicitní oprávnění budou

- pro adresář:

desítkově	binárně	řetězec
777	111 111 111	rw-rw-rwx
AND NOT		
022	000 010 010	----w--w-
755	111 101 101	rw-r-xr-x

- pro běžný soubor:

desítkově	binárně	řetězec
666	110 110 110	rw-rw-rw-
AND NOT		
022	000 010 010	----w--w-
644	110 100 100	rw-r--r--

Pokud si špatně pamatujete bitovou operaci AND NOT, všimněte si, že jsme vlastně masku desítkově po číslicích odečítali od maxima.



Pokud bychom v masce povolili některé oprávnění, které není v „maximu“, toto oprávnění nebude uděleno. Například v maximu pro běžné soubory není právo x. I kdyby bylo v masce povoleno, stejně při použití masky nebude ve skutečnosti nastaveno.

 **Příkaz** `umask` nám buď zobrazí momentálně platnou masku, nebo ji umožní změnit. Změna platí jen do příštího restartu, pokud tento příkaz nedáme do některého skriptu spouštěného při startu systému či přihlášení uživatele.

Zatímco ve starších UNIXových systémech tento příkaz pracoval jen s absolutní číselnou reprezentací, v novějších systémech je možné oprávnění zadávat i symbolicky, včetně relativního tvaru, podobně jako u příkazu `chmod`.



Příklad

Použití příkazu `umask` si ukážeme na příkladu.

`umask` (bez parametrů) zobrazí masku v číselném tvaru, pravděpodobně 0022

`umask -S` podobně, ale výstup bude v symbolickém tvaru, pravděpodobně `u=rwx,g=rx,o=rx` (všimněte si, že symbolický tvar je už „převrácený“ a přímo udává, jaká oprávnění má která kategorie uživatele, a to k adresáři)

`umask 002` změnili jsme masku, tj. u nově vytvářených souborů budou mít členové přidružené skupiny také právo zápisu (mód pro nově vytvořený adresář bude 775, pro běžný soubor 664)

`umask 023` mód pro nově vytvořený adresář bude 754, pro běžný soubor 654 (tj. u nových adresářů jsme „zbytku světa“ znemožnili se do tohoto adresáře dostat)

`umask g+w` oproti dřívější podobě masky jsme členům přidružené skupiny přidali právo zápisu

`umask g=rw,o=-` členům přidružené skupiny jsme přidělili právo čtení a zápisu, ostatním jsme vše zakázali



Úkol

Ověřte si, jak je nastavena maska pro soubory a adresáře na vašem systému. Vypište jak číselný, tak i symbolický tvar nastavení masky.



3.3 Práce pod jiným uživatelským účtem

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

- `su`
- `sudo`

Každý z nich má trochu jiné výhody a nevýhody, a taky způsob použití.

3.3.1 Příkaz `su`

 **Příkaz `su`** (Substitute User) slouží k získání oprávnění jiného uživatele. Sice se typicky používá k získání oprávnění roota, ale je možné použít i jiného uživatele (ovšem musíme znát heslo).

Výhodou je jednoduchost – tímto příkazem dáme na vědomí, že chceme používat oprávnění jiného uživatele, zadáme heslo a od té chvíle se stáváme „jiným uživatelem“. Návrat k původní identitě provedeme příkazem `exit`.



Poznámka:

Ve skutečnosti tímto příkazem spouštíme nový shell (nový proces, který bude potomkem našeho původního shellu), efektivním uživatelem nového procesu je ten uživatel, jehož oprávnění jsme takto získali. Proto se také veškeré příkazy/programy/procesy spouštěné v tomto shellu provádějí se získanými oprávněními. Příkaz `exit` nový shell ukončí, proces s „cizími“ oprávněními přestane existovat a my se

vrátíme do shellu, jehož proces má jako efektivního uživatele naši původní identitu. Alternativně lze použít příkaz `logout`.

S tímtož mechanismem jsme se už setkali – vzpomeňte si na příkaz `newgrp` pro změnu aktivní (pracovní) skupiny – vytvořil se nový shell, který sice měl jako efektivního uživatele pořád nás, ale jeho efektivní skupina byla ta, na kterou jsme se přepnuli.



Pokud příkaz zadáme bez parametrů, znamená to, že chceme získat oprávnění roota (to je nejběžnější případ). Jinak jako parametr zadáme přihlašovací jméno uživatele, pod jehož účtem chceme pracovat.

 Výchozí chování příkazu je takové, že se sice změní UID, pod kterým pracujeme (tj. změní se efektivní uživatel procesu shellu) a vlastně i aktivní skupina se změní na primární skupinu daného uživatele, ale veškerá další nastavení zůstávají „naše“ – domovský adresář, proměnné prostředí (včetně obsahu `PATH`), atd. Pokud chceme přenastavit i tato nastavení, provedeme to uvedením parametru `--login` nebo `-l` nebo jednoduše samotnou pomlčkou. Tyto možnosti jsou ekvivalentní:

- `su --login uživatel`
- `su -l uživatel`
- `su - uživatel`

Takže pokud chceme pracovat jako root včetně domovského adresáře a proměnných a nechce se nám zbytečně opotřebovávat klávesnici, zadáme:

`su -`

(nezapomeňte na tu pomlčku).



Příklad

Předpokládejme, že v systému existuje uživatel `anna`. Nyní jsme přihlášení jako `anna`. Podíváme se na tuto posloupnost příkazů:

- `whoami`
výstup: `anna`
- `id`
výstup: `uid=1005(anna) gid=1005(anna) skupiny=1005(anna),...`
- `su -`
(jsme požádáni o heslo roota, předpokládejme, že ho máme a získali jsme vyšší oprávnění)
- `useradd -m beata`
`passwd beata`
(vytvořili jsme testovací účet, nastavujeme mu heslo; jsme požádáni o heslo, pak o jeho opakování; odteď existuje účet `beata`, jehož heslo jsme taktéž nastavili, aby byl funkční)
- `exit`
(opustili jsme rootovský shell)
- `su beata`
(zadáme heslo uvedeného uživatele)
- `whoami`
výstup: `beata`
- `id`
výstup: `uid=1008(beata) gid=1008(beata) skupiny=1008(beata)`

- `ps tree`
součástí výstupu bude:


```
... bash
    └─ su
       └─ bash
```
- `ps aux | grep beata`
výstup:


```
root    ...  su - beata
beata   ...  -su
beata   ...  ps aux
beata   ...  grep beata
```

 (je zvláštní, že `ps` a `ps tree` každý trochu jinak interpretují procesy/úlohy)
- `exit`

```
su -
userdel -fr beata
exit
```



 Pokud chceme jako jiný uživatel (případně root) provést jen jediný příkaz a zbytečně kvůli tomu nespouštět vnořený shell, použijeme přepínač `-c`:

```
su - uživatel -c "nějaký_program"
```

(případně s celou cestou, pokud ten program není v žádném adresáři z proměnné `PATH`).



Úkol

Pokud je ve vaší distribuci podporován příkaz `su` a máte dostatečně vysoká oprávnění, vyzkoušejte si postup uvedený v příkladu z této sekce.



Poznámka:

Pokud jste si toho už dřív nevšimli, všimněte si alespoň teď: na konci promptu běžného uživatele je obvykle symbol `$`. Jestliže pracujete s oprávněními roota (třeba díky příkazu `su`), místo `$` se na konci promptu objeví symbol `#`. Taky se pravděpodobně změní barva promptu na červenou (ale záleží na konfiguraci).

Je to z toho důvodu, aby uživatel, který čas od času potřebuje pracovat jako root, měl přehled o tom, s jakými oprávněními právě pracuje, aby třeba omylem při vyšších oprávněních nespustil proces, jehož skutečnou činností si není jist (tj. když hrozí bezpečnostní riziko).



3.3.2 Příkaz `sudo`



Příkaz `sudo` (S**U**peruser **D**o) je přímo určen k dočasnému získání vyšších oprávnění ke konkrétnímu účelu. Výchozí chování tohoto příkazu je takové, že obvykle jako parametr uvádíme přímo příkaz, který chceme provést s vyššími oprávněními.

Na rozdíl od příkazu `su` je tento příkaz konfigurovatelný a také lze použít celou škálu dalších přepínačů a parametrů, můžeme tak velice podrobně určit, co vlastně chceme udělat, či oprávnění mají být použita a co vše konkrétně se má přenastavit. Ovšem obvykle se po nás nic moc složitého nechce.

Oproti `su` je tady z hlediska uživatele ještě jeden podstatný rozdíl – zatímco příkaz `su` po nás chce heslo uživatele, kterého jsme zadali (tj. typicky heslo `roota`), příkaz `sudo` po nás bude chtít (většinou, podle konfigurace a podle toho, co vlastně chceme spustit) naše vlastní heslo.

Konfigurace je uložena v souboru `/etc/sudoers`. Zde se stanovuje, kdo a jakým způsobem může příkaz `sudo` používat, případně kdo naopak tento mechanismus používat nemůže.

 To vyplývá z principu fungování *sudo mechanismu*:

- V souboru `/etc/sudoers` je stanoveno, že pokud konkrétní uživatel (třeba `anna`) pomocí `sudo` chce spustit program `xyz`, tak to bude povoleno, přičemž je program `xyz` spuštěn s oprávněními uživatele `bolek`.
- Jestliže tedy uživatel `anna` zadá příkaz

```
sudo xyz
```

musí nejdřív řádně prokázat, že je opravdu `anna`, a pak je jako podproces původního shellu spuštěn nový proces z programu `xyz`, jeho efektivním uživatelem bude `bolek`.

- Prokázání totožnosti je platné po určitou dobu (podle konfigurace, bývá to 5 až 15 minut), takže pokud `anna` v této době znovu použije `sudo` mechanismus, není nucena pokaždé zadávat své heslo.

V praxi se `sudo` mechanismus používá ke spouštění procesů s oprávněními `roota`, tedy k *eskalaci (navýšení) oprávnění*. Takže v konfiguračním souboru typicky máme zadáno, že určitý uživatel může spouštět určité (v jednodušším případě všechny) programy s oprávněními `roota` (ovšem musí vždy zadat své heslo, aby prokázal, že k dané identitě přišel legálně a je „fyzickým“ uživatelem).

 Jak bylo výše uvedeno, `sudo` mechanismus se konfiguruje v souboru `/etc/sudoers`. Je to sice textový soubor, ale není dobrý nápad do něj zasahovat v (jakémkoliv) textovém editoru, protože případná chyba by se mohla dost vymstít. Edistace se provádí v programu `/usr/sbin/visudo`, což je ve skutečnosti některý jiný editor, ale navíc se barevně zvýrazňují určité prvky syntaxe typické právě pro tento konfigurační soubor. Ve většině linuxových distribucí je jako základní editor použit `vi` (což není zrovna pohodlné, pokud uživatel není zvyklý na určité speciality tohoto programu), v jiných distribucích (často vycházejících z Ubuntu) se setkáme s editorem `nano`.

Samozřejmě ne každý může příkaz `/usr/sbin/visudo` použít, k tomu potřebujeme vyšší přístupová oprávnění.

Postup

Podíváme se podrobněji na soubor `/etc/sudoers`. Předpokládejme, že máme dostatečná oprávnění a použili jsme příkaz `/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. 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.

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
    www

pepa mailserver = /bin, !/bin/su
    zadanému uživateli na mailserveru dovolíme spouštět cokoli 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
```



Postup

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“.

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 sekretárky (pokud nastoupí nová sekretárka, 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)



Příklad

Pokud pracujeme v systému používajícím sudo mechanismus, zadáváme před každým příkazem vyžadujícím vyšší oprávnění příkaz `sudo`. Například chceme vytvořit nového uživatele a nastavit mu heslo:

```
sudo useradd -m beata
```

```
sudo passwd beata
```

Při zadání prvního uvedeného příkazu budeme požádáni o (své) heslo, u následujícího příkazu už ne, systém si nás určitý počet minut „pamatuje“.

Sudo mechanismus se dá použít i při náhledu do souboru s hashi hesel:

```
sudo cat /etc/shadow
```



Postup

Příkazy `su` a `sudo` se také mohou kombinovat. Pokud zadáme

```
sudo su -
```

jsme požádáni o *naše heslo* (to zajistí příkaz `sudo`), ale zároveň se vytvoří nový shell s oprávněními roota, získaná oprávnění roota platí i pro další uvedené příkazy (až do zadání příkazu `exit` nebo `logout`). Takže z obou příkazů si bereme jen samé výhody.



Úkol

Pokud používáte distribuci podporující sudo mechanismus a můžete tento mechanismus používat, vytvořte si nového testovacího uživatele, nastavte mu heslo a případně další vlastnosti. Zkontrolujte si výsledek svého snažení:

- v souboru `/etc/passwd` zkontrolujte, zda přibyl nový záznam a jaké údaje o něm jsou tam uvedeny,
- v souboru `/etc/group` taktéž zkontrolujte, jestli se pro tohoto uživatele vytvořila skupina (předpokládejme, že máte systém používající skupiny UPG),
- pomocí sudo mechanismu se podívejte v souboru `/etc/shadow`, jaký je hash hesla nového uživatele,
- v souboru `/etc/login.defs` si můžete ověřit, jaký typ hashe to vlastně je:

```
less /etc/login.defs
```

nebo

```
cat /etc/login.defs | grep -i ^encrypt (nezapomeňte na „^“ před vyhledávacím řetězcem)
```

Pak testovacího uživatele odstraňte.



3.4 Pokročilé mechanismy řízení přístupu

3.4.1 Atributy

 **Základní 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
- e se souborem je v souborovém systému zacházeno v režimu extentů (sdružování malých souborů do větších bloků, aby se ušetřilo místo na disku), což je například podporováno v souborovém systému ext4
- 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, se kterým se seznámíme v následujícím textu.

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



Úkol

Ve svém domovském adresáři vypište atributy všech skrytých souborů:

```
lsattr .*
```



3.4.2 POSIX ACL

 V standardu POSIX, který již (alespoň podle názvu) 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 v podstatě zachován, jen máme více možností pro jejich přiřazování. Rozlišujeme tři *typy ACL*:

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

 Pro práci s POSIX ACL se používají především příkazy `getfacl` a `setfacl`, které slouží k zobrazení a změnám ACL seznamů. Existující nastavení můžeme také exportovat (zálohovat) do souboru. Základní syntaxe:

```
getfacl soubor    výpis ACL pro daný soubor
```

```
setfacl -m u:uzivatel:opraveni soubor    přidání záznamu ACL o uživateli pro daný soubor (do
řetězce opraveni dosadíme co potřebujeme, víc ACL záznamů oddělujeme čárkou, například
setfacl -m u:sarka:rw,bolek:r,anna:rwx soubor
```

```
setfacl -m g:skupina:opraveni soubor    přidání záznamu ACL o skupině pro daný soubor (pokud
chceme členům konkrétní skupiny explicitně přidělit některá oprávnění)
```

```
setfacl -x u:uzivatel soubor    odstranění záznamu ACL o uživateli pro daný soubor
```



Příklad

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:jmeno:opravneni
```

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



Příklad

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. V řetězcích oprávnění můžeme většinu pomlček umazat:

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



Příklad

Opět jeden souhrnný příklad: vytvoříme nového testovacího uživatele, nastavíme mu heslo a v jeho domovském adresáři vytvoříme nový soubor. Potom sami sobě přidáme speciální ACL záznam, kterým si do tohoto souboru povolíme čtení i zápis, třebaže původně jsme tato oprávnění neměli. Nakonec se vrátíme do svého shellu a pokusíme se do nového souboru zapisovat.

- Nejdřív přípravné operace:

```
sudo su -
useradd -m novak
passwd novak
getfacl -m u:sarka:x /home/novak    (proč myslíte, že mi toto bude dobré?)
exit
```

- Vytvoříme nový soubor a upravíme jeho ACL:

```
sudo su - novak
touch test.txt
getfacl test.txt    (vypsali jsme si ACL našeho souboru)
setfacl -m u:sarka:rw test.txt
getfacl test.txt
exit
```

- A teď vyzkoušíme přístup pod svým účtem:

```
echo "zapisuji" > /home/novak/test.txt ; cat /home/novak/test.txt
```

- Úklid:

```
sudo su -
userdel -fr novak
exit
```





Úkol

V manuálových stránkách zjistíte, jak lze při nastavování ACL některého adresáře použít rekurzi. Pokud máte možnost (dostatečně vysoká oprávnění), vyzkoušejte si postup z posledního uvedeného příkladu.



3.4.3 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. Můžeme si to představit třeba tak, že systém do zásobníku (jako LIFO struktury) předem vloží určitá pravidla, která mají být splněna, a proces, který má projít autentizací, ze zásobníku postupně jednotlivá pravidla vyndává tím, že je splňuje. Pokud některé pravidlo nedokáže splnit, v zásobníku zůstane a není možné dostat se k dalšímu pravidlu. Úspěšná autentizace odpovídá vyprázdněnému zásobníku.

PAM využívají různé služby, ke kterým se autentizují uživatelé (kromě jiného také služba přihlašování uživatele do systému nebo některé demony 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` (tj. knihovny). 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` (omezení míst, ze kterých se lze přihlásit) je v souboru `/etc/security/access.conf`. Tyto soubory jsou obvykle dobře okomentovány, tedy pokud chceme konfiguraci změnit, obvykle nebývá problém.

Příklad

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.



3.4.4 Capabilities

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).

3.4.5 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 nejnútnejší 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)



Poznámka:

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.



Úkol

Vyzkoušejte si z výše uvedeného postupu alespoň příkaz *ldd* – zjistěte, které knihovny potřebují ke svému běhu programy *cat*, *find*, *usermod* a *iwlist*. Nezapomeňte, že příkaz *ldd* vyžaduje zadání parametru včetně celé cesty, tedy byste si měli předem zjistit, kde v adresářové struktuře se jednotlivé programy nacházejí (stačí použít příkaz *which* nebo *whereis*).



3.5 Příkaz stat

Příkaz `stat` zobrazuje podrobnější informace o souboru, zejména o přístupových oprávněních k němu. Výstup je přehlednější a má v sobě víc informací než klasický `ls -l`.



Příklad

Vytvoříme nový soubor (jsme ve svém domovském adresáři) a vypíšeme si jeho vlastnosti:

```
touch pokusny
```

```
stat pokusny
```

```
Soubor: "pokusny"
Velikost: 0          Bloků: 0          I/O blok: 4096      běžný prázdný soubor
Zařízení: 804h/2052d I-uzel: 1704009    Odkazů: 1
Práva: (0644/-rw-r--r--) UID: (1000 / sarka) GID: (1000 / sarka)
Přístup: 2017-04-28 19:06:21.705897187 +0200
Změna obsahu: 2017-04-28 19:06:21.705897187 +0200
Změna i-uzlu: 2017-04-28 19:06:21.705897187 +0200
```

Vidíme, že není nastaven příznak `x` (právo spouštění). Není divu, je to běžný soubor, není ani spustitelný ani adresář. Co se stane, když tomuto souboru nastavím příznak SUID?

```
chmod u+s pokusny
```

```
stat pokusny
```

```
...
```

```
Práva: (4644/-rwSr--r--) UID: (1000 / sarka) GID: (1000 / sarka)
Přístup: 2017-04-28 19:06:21.705897187 +0200
Změna obsahu: 2017-04-28 19:06:21.705897187 +0200
Změna i-uzlu: 2017-04-28 19:12:20.853760773 +0200
```

Změnil se řádek, na kterém jsou uvedena přístupová oprávnění. Všimněte si, že v řetězcovém vyjádření máme velké „S“ místo malého. To znamená, že sice je nastaven SUID bit, ale nastavení není funkční, protože vlastník nemůže provést operaci spuštění. Takže nastavíme příznak „x“:

```
chmod u+x pokusny
```

```
stat pokusny
```

```
...
```

```
Práva: (4744/-rwsr--r--) UID: (1000 / sarka) GID: (1000 / sarka)
```

```
...
```

Teď už je příznak nastaven správně, je funkční. Všimněte si ještě jedné věci – časových údajů. Po změně příznaků se změnil pouze časový údaj u parametru „Změna i-uzlu“. My jsme totiž vůbec neměnili obsah souboru a vlastně jsme k němu ani nepřistupovali (k jeho obsahu), pouze jsme měnili jeho parametry, které se nacházejí ve speciálním záznamu o souboru (i-uzlu).



Správa zařízení

4.1 Paměťová média

4.1.1 Příkazy pro paměťová média

 **Příkaz df** (Disk Free) vypisuje volné místo na jednotlivých paměťových médiích včetně diskových oddílů. Pomocí přepínačů můžeme podrobněji určit, co a jak se má vypsát.

df ke každému oddílu na disku se vypíší 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 vypíší se informace o všech připojených souborových systémech včetně mnoha virtuálních (ne všech), tento výpis má smysl například tehdy, když chceme přehled připojených souborových systémů



Poznámka:

I-uzel (i-node) je struktura obsahující informace o souboru (přístupová oprávnění, čas vytvoření a poslední změny, umístění na paměťovém zařízení, apod.).



Příklad

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

Vidíme, že je připojeno poměrně hodně souborových systémů (většinou jde o oddíly disku), zde se zobrazují i virtuální. Ke každému je vypsána velikost, kolik je používáno, kolik volného místa, procento využití a přípojný bod.



 **Příkaz du** (Disk Usage) vypisuje obsazené místo na disku zabrané konkrétním souborem nebo adresářem (včetně jeho podadresářů, rekurzivně). Protože souborů obvykle bývá na disku poměrně hodně, typické použití tohoto příkazu je s parametrem, který je buď název souboru nebo seznam názvů nebo maska, která představuje (omezenou) množinu názvů souborů.

`du` vypíše rekurzivně 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 prepínač 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)

 **Příkaz lsof** (LiSt Open Files) vypíše seznam otevřených souborů a procesů, které s těmito soubory pracují (vychází ze seznamu procesů, ke každému zjišťuje, které soubory má tento proces otevřené). 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 knihovny (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)

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

 **Příkaz `sync`** slouží k synchronizaci paměťových médií s cache pamětí. Pokud ukládáme data na paměťové médium (včetně USB flash disků), často se nejdříve uloží do mezipaměti (cache) a teprve pak se postupně ukládají tam, kam je třeba. Tímto příkazem si vynutíme jejich okamžité uložení, třeba před odpojením média.

 **Sada příkazů `fsck`** (File System ChecK) je určena ke kontrole vadných sektorů na disku (parametry najdeme v nápovědě), existují varianty pro konkrétní souborový systém – `fsck.ext2`, `fsck.ext3`, `raisefscck`, `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 předem odpojen)

Krátká kontrola nejdůležitějších oddílů se automaticky provádí i během startu systému, jak zjistíme později (když budeme probírat soubor `/etc/fstab`), protože později (za běhu systému) ji nelze provést.

 **Sada příkazů `mkfs`** (MaKe FileSystem) dokáže na paměťovém médiu (oddílu disku, USB flash disku) vytvořit souborový systém (v terminologii Windows zformátování, parametry najdeme v nápovědě), opět existují varianty pro různé souborové systémy – `mkfs.ext2fs`, `mkfs.ext3fs`, `mkfs.vfat`, apod.



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`.



4.1.2 Soubor `fstab`

Soubor `/etc/fstab` slouží k *evidenci připojitelných logických disků* – těch, které se mají připojit během startu systému, a případně těch, které se mohou připojit kdykoliv později (výměnných paměťových médií).

 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 novějších verzích distribucí se zde setkáváme s označováním oddílů nikoliv názvem speciálního souboru, ale identifikátorem UUID (Universally Unique ID). Je to hexadecimální řetězec, který

není zrovna „uživatelsky příjemný“, ale na rozdíl od (dynamicky generovaných) názvů speciálních souborů je jednoznačný.

2. Název přípojného bodu (mounting point), přes který bude zařízení přístupné (například `/media/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`, `ext4`, `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 `udf` (Universal Disk Format), pro swap partition je to hodnota `swap`.

Můžeme se setkat s volbou `auto`, to znamená automatické rozpoznání souborového systému při připojování. Je to vhodné pro výměnná média, u kterých nevíme předem, který souborový systém používají.

4. Seznam parametrů pro připojení. Nejdůležitější jsou

- `ro` (pouze pro čtení), `rw` (i pro zápis),
- `user` (připojit může i běžný uživatel, odpojit tentýž uživatel), `users` (připojit běžný uživatel, odpojit kdokoli, 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 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, 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 nejdříve), 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. nespustění) Linuxu.



Postup

Vraťme se k identifikátorům UUID používaným k označování oddílů místo speciálních souborů. Pokud si nejsme jisti, kterému speciálnímu souboru přísluší určitý konkrétní identifikátor UUID, můžeme si vztahy zobrazit příkazem `blkid`. Výstup bude vypadat zřejmě nějak takto:

```
...
/dev/sda4: UUID="d4f25949-0d2..." TYPE="ext4"
/dev/sda5: LABEL="DATA" UUID="1F0801CAA09..." TYPE="ntfs"
...
/dev/sdb1: LABEL="TRANSCEND" UUID="B49B-633..." TYPE="vfat"
```



Příklad

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

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

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
```

Druhý výpis se může zdát méně přehledný, protože jednotlivá pole na řádku jsou oddělována mezerou, ne tabulátorem. Poslední řádek patří právě připojenému výměnnému zařízení (USB flash disku).





Úkoly

1. Projděte si obsah souboru `/etc/fstab` a vypište si obsah souboru `/etc/mtab`. Srovnajte jejich řádky a zjistěte, ve kterých se liší. Dále se podívejte, jak jsou nastaveny vlastnosti těchto souborů (vlastník, skupina, oprávnění) a kdy naposledy se k nim přistupovalo:

```
ls -l /etc/*tab
```

2. Připojte do systému některý USB flash disk a v souboru `/etc/mtab` zjistěte, s jakými nastaveními byl připojen. Pokud některému parametru nerozumíte, podívejte se do manuálové stránky příkazu `mount`.



4.1.3 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

 **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.

Ve skutečnosti za nás tento úkol většinou plní některý program, který je napojen na mechanismus HotPlug v jádře. Takže vložíme optické médium do mechaniky nebo zasuneme SD kartu či USB flash disk, dojde k automatickému rozpoznání média a jeho připojení. Tak to funguje v distribucích pro běžné uživatele, ale v distribucích určených pro testování sítě a forenzních distribucích obvykle uživatel o připojování rozhoduje ručně. Ruční připojení často funguje i v grafickém prostředí – v kontextovém menu ikony představující paměťové médium zvolíme *Připojit (Mount)*, pro odpojení *Odpojit (Unmount)*, v různých distribucích mohou být tyto položky různě nazvány. Nebo na to máme aplikaci.



Příklad

V shellu použijeme například tyto příkazy:

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

`mount /media/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 ro /media/winD` připojí systém pouze pro čtení

`mount -o rw /media/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 /media/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

Ukážeme si připojování a odpojování disků:

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

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

```
ntfs-3g /dev/sda1 /media/C    na zadaném oddílu je souborový systém NTFS a používáme ovladač ntfs-3g (což je momentálně u většiny linuxových distribucí), tak použijeme rovnou obslužný program tohoto ovladače
```

```
mount -t ntfs-3g /dev/sda1 /media/C    totéž, ale jdeme oklikou přes příkaz mount
```

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

```
mount -t vfat /dev/sdb1 /media/usb    připojí (zřejmě) USB flash disk se systémem FAT32, předpokládáme, že adresář /media/usb existuje, případně také můžeme zadat další parametry
```

```
mount -t vfat -o ro /dev/sdb1 /media/usb    totéž, ale připojili jsme jen pro čtení (ro je read-only)
umount /media/usb    odpojení připojeného média (stačí zadat přípojný bod)
```

V některých případech je dobré přidat i parametr pro znakovou sadu, aby se správně zobrazovaly názvy souborů a adresářů v případě, že někdo použil v názvu diakritiku (ano, i takoví divní lidé jsou). Nicméně většina souborových systémů dnes využívá UTF8, takže problém nastat nemusí.



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`).



Poznámka:

Příkaz `mount` má mnoho volitelných parametrů. Některé z nich jsou víceméně podporovány pro všechny běžné souborové systémy (například možnost připojit pouze pro čtení), jiné jsou používány jen pro konkrétní souborový systém. Manuálová stránka tohoto příkazu uvádí souhrn parametrů platných obecně, a pak speciální parametry zvlášť pro různé souborové systémy, tedy máme tam část stránky pro parametry souborového systému `ext3`, další pro `ntfs`, pro `vfat`, `xfs`, `smbfs`, `tmpfs`, `raiserfs`, `udf`, atd.

Pokud pro určitý souborový systém používáme jiný program než `mount` (třeba `ntfs-3g`), pak tento program taky bude používat určité parametry, a není řečeno, že se budou zapisovat stejně jako u `mount`. Opět je najdeme na manuálové stránce (například `man ntfs-3g`).



Připojení USB flash disku. Následující postup je v novějších distribucích plně řešen automatickým připojováním, ale hodí se u hodně starých distribucí nebo u distribucí určených k opravám či forenznímu auditu.

Když chceme připojit USB flashdisk, nejdřív vytvoříme odpovídající adresář v `/media`, pokud tam ještě není:

```
mkdir -m 666 /media/flash
```

(parametrem příkazu mohou být i přiřazená přístupová práva, ať tento krok nemusíme dělat dalším příkazem, případně další, třeba připojení jen pro čtení).

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

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

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

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

 **Připojení obrazu disku.** Připojovat nemusíme jenom (fyzická) média, můžeme připojovat také obrazy disků, třeba ve formátu ISO nebo IMG. Pak se k nim dá přistupovat naprosto stejně jako k jakémukoliv jinému médiu.



Příklad

Připojení provedeme vytvořením smyčky (loop) k danému médiu:

```
mkdir /media/PripojnyBod
```

```
mount -o loop soubor.iso /media/PripojnyBod
```

Pak už můžeme k obsahu tohoto virtuálního média přistupovat naprosto stejně jako k jakémukoliv jinému paměťovému médiu, třeba:

```
ls /media/PripojnyBod
```



4.1.4 Program fdisk

 Jedním z nejdůležitějších programů pro správu disků je program `fdisk`. Programem `fdisk` je možné vytvářet i jiné než linuxové oddíly, včetně i `msdos` a `vfat`, ale nedoporučuje se to, DOS a Windows s takto vytvořenými partitions nemusejí pracovat správně.

Práce s oddíly na disku probíhá v interaktivním prostředí programu, do kterého se dostaneme tak, že jako parametr příkazu zadáme název speciálního souboru pro daný disk (třeba `/dev/sda`).

Příkazy (reprezentované jednou klávesou) používané uvnitř programu jsou v tabulce 4.1. Nemusíme si je pamatovat, stačí vědět, že stisknutím klávesy  získáme nápovědu (a i tento fakt je nám sdělen hned po spuštění programu).

Klávesa	Význam
	nápověda při práci s programem
	zobrazí oddíly disku
	vytvoří nový oddíl (program nás vede, občas stiskneme nějakou klávesu, udáme začátek a velikost oddílu)
	odstraní oddíl

Tabulka 4.1: Klávesové příkazy uvnitř interaktivního prostředí `fdisk`



Příklad

Ukážeme si, jak se program `fdisk` spouští. Pozor, příkaz `fdisk` vyžaduje vyšší přístupová oprávnění (jinak by nevypsalo ani písmeno), takže například v distribuci používající `sudo` mechanismus před každý příkaz předsuňte `sudo` `fdisk...`).

`fdisk -l /dev/sda` vypíše informace o zadaném disku (jeho oddíly, včetně souborového systému)

`fdisk -l` vypíše informace o všech dostupných discích, ke každému to, co by vypsal v předchozím případě

`fdisk /dev/sda` chceme pracovat s konkrétním diskem, tedy jsme přesunuti do interaktivního režimu, ve kterém můžeme zadávat „písmenkové“ příkazy (viz tabulku 4.1).



Stručný přehled disků a oddílů získáme také příkazem `lsblk` (vlastně vypisujeme údaje o blokových zařízeních, což jsou typicky vnější paměťová zařízení).

4.1.5 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. Podobně všechny disky a oddíly budou mít stejné hlavní číslo, ale odlišná vedlejší čísla. Taktéž virtuální zařízení mají stejné hlavní číslo a různá vedlejší čísla.



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



Souborový systém `udev` slouží ke správě přístupu k zařízením. Je to virtuální souborový systém, který poskytuje rozhraní mezi procesy a ovladači zařízení běžícími převážně v jádře, dynamicky generuje hlavní a vedlejší číslo pro každé zařízení, které je připojeno ať už při startu systému, nebo později, a také generuje název speciálního souboru zařízení.

Jeho činnost zajišťuje démon `udev`, který periodicky zjišťuje informace ze souborového systému `sysfs`. Díky tomu má neustále přehled o změnách v seznamu připojených zařízení a může včas na změny reagovat.



Úkoly

1. Vypište obsah adresáře `/dev` tak, aby se u každého speciálního souboru zobrazilo hlavní a vedlejší číslo zařízení.
2. Zjistěte hlavní a vedlejší čísla příslušející oddílům na vašem pevném disku.
3. Zjistěte hlavní a vedlejší čísla pro virtuální zařízení typu `/dev/null`, `/dev/random` apod.



4.2 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.), pomocí přepínačů můžeme řídit formát výpisu (například `-m` určí výpis množství paměti v MB, `-g` zase v GB)

`cat /proc/meminfo` podrobnější informace o využívání operační paměti, tedy vypisujeme obsah dynamického souboru (v textovém editoru tento soubor neotevřete, musíte použít tento způsob)

`mpmap 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)

Některé údaje o využívání paměti procesem získáme také příkazem `top`.

Postup

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 souboru na typ swap (upraví se jeho vnitřní struktura)

`swapon /novy_swap` zapnutí používání souboru pro swapování

Pokud chceme používat swap oddíl, stačí, když je uveden v souboru `fstab` s označením `swap` (může jich být víc než jeden), zapnutí používání těchto oblastí pro swap se provádí také příkazem `swapon`.



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

Úkoly

1. Vypište si podrobné informace o operační paměti. V manuálové stránce příkazu `free` zjistěte, jaké přepínače se dají použít.
2. Vypište obsah souboru `/proc/meminfo`.
3. Vyberte si některý proces, zjistěte jeho PID a zobrazte údaje o paměti využívané tímto procesem.



4.3 Procesor

 Informace o procesoru můžeme získat třemi základními způsoby:

- příkaz `arch` jednoduše zobrazí architekturu procesoru,
- příkaz `lscpu` (zkratka z „list CPU“) vypisuje podrobnější informace o procesoru, včetně údajů o jádrech a cache pamětí,

- příkaz `cat /proc/cpuinfo` vypíše obsah zadaného dynamického souboru (v textovém editoru ho neotevíre!), obsah je ještě podrobnější než u předchozího příkazu.



Úkol

Vyzkoušejte výše uvedené příkazy.



4.4 Informace o architektuře a základní desce

4.4.1 Základní deska a přehled komponent



Příkaz `uname` nám dokáže říct, na jakém zařízení vlastně pracujeme. Ovšem záleží, který přepínač použijeme:

`uname` zjistíme pouze název operačního systému (tj. obvykle Linux)

`uname -n` vypíše název zařízení

`uname -v` vypíše verzi operačního systému

`uname -r` vypíše verzi jádra („release“)

`uname -a` vypíše všechno předchozí kromě verze jádra



Pomocí příkazu `dmidecode` získáme informace o základní desce a jejích komponentách. Tento program vyžaduje vyšší přístupová oprávnění. Tento příkaz se napojuje na rozhraní SMBIOS (jeho dřívější označení bylo DMI, od toho název příkazu) a odtud získává velmi podrobné informace o komponentách základní desky.



Příklad

Příkaz `dmidecode` používáme buď na terminálu s vyššími přístupovými oprávněními nebo použijeme `sudo` mechanismus.

`dmidecode` vypíše se informace o BIOSu/UEFI, základní desce, chassis, jednotlivých komponentách základní desky včetně různých radičů; u každé komponenty máme kromě jiného sériové číslo a údaj o typu zařízení (číslo), například u BIOSu je 0, u samotné základní desky je typ 2, různá komunikační rozhraní mají typ 8, svůj typ mají cache paměti, SATA radiče, sloty pro rozšiřující karty, paměťové moduly, procesor, atd.

`dmidecode -t processor` výpis podrobnější informace o komponentách typu „processor“ (pokud je v systému jenom jeden procesor, tak jenom o tom jednom)

`dmidecode -t 4` totéž, protože typ „processor“ má číslo 4

`dmidecode -t connector` výpis informací o konektorech základní desky vyvedených přímo ze skříně (USB, HDMI, eSATA, audio, LAN, apod.), ne těch na rozšiřujících kartách; případně se dá použít jako typ kód 9

`dmidecode -t system` získáme podrobné informace o systému (výrobce, sériové číslo, apod.)

`dmidecode -t` když nevíme, jaké typy komponent jsou takto přístupné, použijeme tento příkaz – v podstatě se vypíše chybová hláška, ve které je i seznam komponent



 **Příkazem** `lshw` získáme informace o hardwaru (zkratka z „list hardware“, vyžaduje vyšší přístupová oprávnění). Celý výstup je velmi dlouhý a trochu připomíná `dmidecode`, kratší verzi získáme zadáním `lshw --short`.

 **Informace o hardwaru na nejnižší úrovni** (na úrovni spodní části jádra): na starších systémech (s jádrem verze 2.6) se používala vrstva HAL, démon `hald` a příkaz `lshal` (vypsal informace o zařízeních napojených na vrstvu HAL, v podstatě přehled hardwaru). Na novějších nic z toho nemáme. Funkce vrstvy HAL převzal modul UDEV a k souvisejícím informacím se dostaneme díky příkazu `udevadm`. Má trochu složitější syntaxi, například:

`udevadm info --name=/dev/sda --query=property` zobrazí vlastnosti zařízení `/dev/sda`, tedy prvního pevného disku

`udevadm info --name=/dev/sda --query=path` takto zjistíme, přes která rozhraní se s daným zařízením komunikuje (cesta může být složitější než myslíme)

`udevadm info --name=/dev/sda --attribute-walk` „procházka“ po cestě, po které probíhá komunikace s daným zařízením (vše, co máme v řetězci výsledku předchozího dotazu)

`udevadm monitor` spustí interaktivní sledování hardwaru na nejnižší úrovni (připojování, odpojování, změny konfigurace zařízení)



Úkoly

1. Vyzkoušejte si použití příkazu `uname` včetně jeho různých parametrů.
2. Pokud máte dostačující přístupová oprávnění, vyzkoušejte si příkaz `dmidecode`. Zjistěte především údaje o BIOSu, systému a základní desce, vypište podrobnější informace o procesoru a pamětech. Zjistěte, které komponenty se takto dají lustrovat (vypište si jejich seznam).
3.  Vyzkoušejte si monitorování hardwaru přes `udevadm monitor` – zkuste spustit, pak třeba připojte a softwarově odpojte USB flash disk, atd.; sledování skončí, když stiskneme .



4.4.2 Sběrnice PCI, PCIe, USB



Poznámka:

Určitě jste si už všimli, že hodně příkazů vypisujících přehledy začíná zkratkou `ls` (tedy „list“), za kterou následuje zkratka pro to, co má vypsat. S některými jsme se už setkali (`lscpu`, `lshw`, `lsblk`), další budou následovat zde i později.



 **Příkaz** `lspci` vypisuje údaje o sběrnících PCI a PCI Express a o tom, co přes ně komunikuje.

`lspci` takto získáme přehled radičů připojených přes PCI a PCIe, jednotlivé komponenty jsou uvedeny svými názvy

`lspci -n` totéž, ale komponenty jsou uvedeny svými identifikátory (tj. není to moc přehledné), identifikátor je ve tvaru `[vendor:zařízení]`, například `[8086:2a40]`

`lspci -nn` totéž, ale je uvedeno obojí (identifikátor i název), takže zjistíme, co k čemu patří, identifikátory jsou v hranatých závorkách

`lspci -d vendor:zařízení` vyžádáme si informace, které se týkají jen konkrétní komponenty (vypíše se jenom jeden řádek)

`lspci -v` podrobnější výpis

`lspci -vv` ještě podrobnější výpis (tři věčka by byla ještě podrobnější)

`lspci -vv -d vendor:zařízení` vyžádáme si podrobnější informace o jedné konkrétní komponentě (pozor, pro některé údaje je třeba vyšší oprávnění – jinak by se příslušná položka vůbec neobjevila nebo by u ní informace o nedostupnosti)

 **Příkaz** `lsusb` vypisuje údaje o sběrnici USB a tom, co přes ni komunikuje.

`lsusb` přehled USB hubů a připojených zařízení (jenom seznam s identifikátory a stručným názvem)

`lsusb -s sběrnice:zařízení` zobrazí se jen řádek pro konkrétní zařízení; číslo sběrnice (bus) a zařízení (device) bychom zjistili z výpisu předchozího příkazu, v číslech nemusíme psát nuly zleva, takže kód 002:003 můžeme napsat jako 2:3

`lsusb -v` k jednotlivým komponentám se vypíší podrobnější informace

`lsusb -t` ve stromové struktuře vidíme, co je ke kterému hubu připojeno, tak dokážeme snadněji identifikovat hub, přes který komunikuje konkrétní zařízení

`lsusb -vs sběrnice:zařízení` podrobný výpis vlastností pro jedno konkrétní USB zařízení

Příklad

Připojíme USB flash disk a zobrazíme seznam pro sběrnici USB:

`lsusb`

```
Bus 002 Device 003: ID 04f2:b036 Chicony electronics Co., Ltd Asus Integrated 0.3M UVC Webcam
Bus 002 Device 006: ID 8564:1000 Transcend Information, Inc. JetFlash
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
...
```

(výstup je zkrácen). V zobrazeném úseku máme tři položky ze sběrnice 2 a jednu ze sběrnice 8, přičemž na sběrnici 2 jsou tři z těchto zařízení (číslo 3, 6 a 1). První vypsaná položka je zjevně webkamera (připojená přes rozhraní USB, to není až tak divné), další je náš USB flash disk, zbývající dvě položky jsou USB huby (rozbočovače). První z nich komunikuje podle USB verze 2.0, druhý je nižší verze.

USB flash disk má tedy číslo sběrnice 2 a číslo zařízení 6. Podrobnější informaci vypíšeme takto:

`lsusb -vs 2:6`

Pokud nás zajímá, kde ve struktuře je naše zařízení zapojeno, zadáme:

`lsusb -t`

```
...
/: Bus 03.Port 1: Dev 1, Class=root_hub, Driver=uhci-hcd/2p,12M
/: Bus 02.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/6p,480M
|__ Port 2: Dev 6, If 0, Class=Mass Storage, Driver=usb_storage, 480M
...
```

Takže náš USB flash disk je připojen k hubu komunikujícímu podle USB 2.0 (podle toho ostatně vypadají i konce těchto řádků). Mohlo to být horší.



**Poznámka:**

Pokud víte, že příkaz začíná řetězcem `ls`, ale nemůžete si vzpomenout, jak pokračuje, máte tyto možnosti:

- zadáte `ls` a stisknete klávesu  (může být nutné ji stisknout dvakrát – vyzkoušejte),
- zadáte `apropos -r "^ls"` (v regulárním výrazu určíte, že na začátku řádku má být daný řetězec),
- zadáte `apropos -s 1,8 -r "^ls"` (totéž, ale navíc omezíte hledání jen na příkazy – jejich manuálové stránky jsou totiž v sekcích 1 a 8).

**Úkol**

Nejdřív si vypíšte seznam všeho, co je napojeno na sběrnici USB, pak totéž ve stromové struktuře. Připojte USB flash disk a vše proveďte znovu, sledujte, kde konkrétně se toto zařízení objevilo a jak je reprezentováno. Zjistěte identifikátor tohoto zařízení (sběrnice a číslo zařízení) a vypíšte o něm podrobnější informace.



4.5 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 je najdeme v adresáři `/lib/modules/...kernel/`, kde jsou 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 proprietární nebo neuvedená).

Pro práci s moduly jádra můžeme využít například tyto příkazy:



```
/sbin/lsmmod
```

(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/modinfo
```

vypíše informace o zadaném modulu; jako parametr zadáváme název modulu, třeba zjištěný předchozím příkazem



`/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 ten původní 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)



Příklad

Příkaz `lsmod` vypisuje všechny právě zavedené moduly jádra a výpis je velmi dlouhý. Pokud hledáme něco konkrétního, je dobré výstup profiltrovat. Například pokud chceme údaj o modulu pro wi-fi:

```
lsmod | grep wifi
```

```
iwlwifi      188416 1 iwlwifi
cfg80211     524288 3 iwlwifi,mac80211,iwlwifi
```

Vypisuje se název modulu, jeho velikost, kolika jinými moduly jádra je používán a které to jsou. Podrobnější informace třeba o modulu `iwlwifi` získáme takto:

```
modinfo iwlwifi
```

Vypíše se název a umístění souboru s modulem, licence, popis, závislosti na jiných modulech, ale také údaje o digitálním podpisu (moduly jádra totiž potřebují digitální podpis) a další informace.



Poznámka:

Pokud se při pokusu o spuštění výše uvedených příkazů vypisuje chybové hlášení o tom, že příkaz nebyl nalezen, bude to tím, že v proměnné `PATH` zřejmě není uvedena cesta `/sbin`. Můžeme si ověřit příkazem `which` nebo `whereis`, kde konkrétní „odmítaný“ příkaz leží, ale velice pravděpodobně to bude právě tam. Pak stačí příkaz spustit s plnou cestou, takže třeba `/sbin/modinfo`. Dalším důvodem problémů jsou nízká oprávnění, ovšem vyšší oprávnění se zde vyžadují pouze u příkazu `modprobe`.



Úkoly

1. Zjistěte, které moduly jsou právě načteny v jádře.
2. Vypište seznam souborů modulů jádra, které ve svém názvu obsahují řetězec „video“ (tj. budou souviset s videovýstupem).
3. Vyberte si některý z modulů a zjistěte o něm podrobnější informace.
4. Najděte v adresářové struktuře místo, kde jsou uloženy soubory s moduly jádra, projděte si strukturu podadresářů.



Kapitola 5

Správa sítě

5.1 Soubory

 V souboru `/etc/hostname` je název našeho zařízení. Takže jak zjistíme název našeho zařízení:

```
cat /etc/hostname
```

 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 i IPv6, takže na systému podporujícím oba protokoly zde budou i řádky pro protokol IPv6, například

```
::1          localhost ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters
```

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).

 V souboru `/etc/resolv.conf` nastavujeme kromě jiného adresu DNS serveru. Jde o řádek (nebo 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.

Souborů souvisejících se sítí je víc, k některým se dostaneme později.



Postup

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 dotýčných 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.



Úkol

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



5.2 Několik základních příkazů



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 .

`ssh počítač` podobně jako `telnet`, ale spojení je šifrované, proto bezpečnější, doporučuje se používat místo `telnetu`. Zadává se často i s názvem uživatelského účtu:

```
ssh uživatel@počítač (počítač může být zadán názvem nebo třeba IP adresou)
```

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

5.3 Starší příkazy pro práci se síťovým rozhraním

Z nejdůležitějších příkazů pro práci s adresami (IP, MAC apod.) a obecně síťovým rozhraním:



ifconfig

zjištění MAC adresy, a také IP adresy pro určité rozhraní, masky sítě, atd., přidělení IP adresy určitému rozhraní (například ethernetové kartě – jako rozhraní použijeme eth0), tento příkaz 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 prepínač 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 prepínačem -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)

**Poznámka:**

Od používání příkazů `ifconfig`, `arp` a `route` se v UNIXovém světě postupně odkláníme, přechází se na mechanismus *iproute2*, kterému se budeme věnovat dále v této kapitole (používá se příkaz `ip`). V některých distribucích dokonce starší příkazy ani nejsou funkční.

**Úkol**

Pokud ve vaší distribuci ještě výše uvedené příkazy fungují, vyzkoušejte si vypsání vašich adres, ARP tabulky a směrovací tabulky. Také si vyzkoušejte deaktivaci síťové karty a její opětovnou aktivaci.



5.4 Překlady a testování

5.4.1 Mechanismus DNS

Pro překlad názvů (mechanismus DNS) máme v UNIXových systémech poněkud více možností než ve Windows.

**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`. Nicméně od používání tohoto programu se upouští (sice obvykle existuje, ale jeho některé interní příkazy nebývají implementovány, což zjistíme v manuálové stránce).

**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.



Úkoly

1. Vyzkoušejte DNS překlad pomocí příkazu `host` oběma směry (nejdřív přeložte zvolenou jmennou adresu, například `www.slu.cz` na IP adresu, pak některou z vypsaných IP adres přeložte na jmennou adresu).
2. Podobně si vyzkoušejte i příkaz `dig`. Pro vybranou jmennou adresu zvolte různé podrobné výpisy. Všimněte si, že formát podrobnějšího výpisu vlastně odpovídá členění DNS paketu.



5.4.2 Testování dostupnosti a výpis statistik

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 určitého zařízení. Funguje prakticky stejně jako ve Windows. Odesílá cílovému počítači testovací zprávy ICMP request (každou sekundu) a čeká na odezvu. Příkaz v základní syntaxi pracuje „do nekonečna“, jeho činnost přerušíme například stiskem kláves `Ctrl+C`, pak vypíše souhrnnou statistiku (kolik paketů bylo odesláno, přijato, podíl ztracených, odezvu cílového počítače). Cílový počítač zadáváme buď jeho IP adresou, nebo jeho URL (tj. `www.xxx.zz`).

Přepínač `-c` určuje, kolik paketů má být odesláno (takže když chceme odeslat jenom čtyři pakety, zadáme `ping -c 4 adresa`), 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

vypisuje statistiky související se síťovými protokoly, kontroluje konfiguraci a provoz na síti, je to 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` (bez parametrů) zobrazí přehled všech otevřených socketů, tj. celkový přehled o síťovém subsystému

`netstat -a` vypíše všechny nejdůležitější informace o probíhající komunikaci (otevřené síťové porty, vzdálený systém, se kterým se komunikuje, stav spojení, TCP i UDP komunikaci)

`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)

`netstat -natpl` víceméně totéž (naslouchající), ale zahrne i další (ty, co kromě naslouchání také spojení aktivně používají nebo právě ukončují spojení)

Další várka použitelných přepínačů je v tabulce 5.1.

Přepínač	Význam
<code>-a</code>	souhrnný přehled informací o probíhající komunikaci (<code>--all</code>)
<code>-n</code>	místo doménových názvů zobrazí IP adresy (rychlejší výpis)
<code>-p</code>	doplní sloupec s PID a názvem procesu, který přes síť komunikuje
<code>-t</code>	pouze protokol TCP
<code>-u</code>	pouze protokol UDP
<code>-l</code>	(listen) zobrazí pouze sockety ve stavu LISTEN (naslouchání)
<code>-i</code>	zobrazí stručnou statistiku <i>síťových rozhraní</i> (TX, RX, atd.)
<code>-ei</code>	taktéž, ale mnohem podrobnější informace
<code>-s</code>	zobrazí statistiku <i>síťových protokolů</i>
<code>-g</code>	seznam skupin (multicast komunikace), do kterých dané zařízení patří
<code>-r</code>	výpis hlavní směrovací tabulky (routing table)

Tabulka 5.1: Výběr přepínačů programu `netstat`

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 84).

 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.



5.5 Mechanismus iproute2 pro práci se sítí

 Ve všech distribucích se setkáme s 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.

Výhodou mechanismu *iproute2* je komplexnost (zahrnuje nejrůznější typy úloh souvisejících se síťovým rozhraním, včetně takových, které nelze pomocí starších příkazů provádět) a podpora novějších standardů včetně plné podpory IPv6.

5.5.1 Konfigurace síťového rozhraní a adres

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

 **Příkaz `ip link`** se vztahuje k vrstvám L1 a L2 modelu ISO/OSI, 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 výpisy 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`; pozor, změna MAC nemusí někdy dopadnout dobře, jde o potenciálně nebezpečnou operaci

 **Příkaz `ip addr`** pracuje 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

Podíváme se na výstupy několika příkazů:

`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
```

```
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.).

```
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). Srovnajte 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.



5.5.2 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
```

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.

 **Příkaz `ip route`** pracuje se směrovacími tabulkami.

`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

`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 (zadááme 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 (mimo chodem, je to jedna z IP adres serverů Facebooku), paket bude zahozen a dotyčný nebude informován



Příklad

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

```
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
```

```
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 mít přístup k serveru s citlivými údaji, kdežto uzly z jiné (prověřené či administrativní) sítě k němu mít přístup budou.



Úkol

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



5.5.3 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` (může se zkracovat na `ip neigh`, `ip n` apod.).



Příkaz `ip neigh` slouží k zobrazování a upravování vazeb 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*.

```
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 (zadáваме IP adresu, MAC adresu – Link Layer Addr, rozhraní, přes které je soused dosažitelný, a pak stav)

`ip neigh del 193.168.200.1 dev eth2` odstranili jsme záznam z tabulky

 Pokud přidáváme nový záznam, můžeme hodnotu `nud` nastavit na jednu z hodnot `noarp`, `reachable`, `permanent` nebo `stale`, čímž určíme, zda záznam bude mít omezenou platnost a jestli má být ověřován.

Příklad

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:

```
ip neigh show
```

```
10.0.0.138 dev eth0 lladdr 00:21:63:e2:0f:98 STALE
```

```
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 řádce 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).

Pokud jsme připojeni bezdrátově, naším sousedem je AP (příp. wi-fi router), například:

```
ip neigh show
```

```
193.84.204.1 dev wlan0 lladdr 00:15:17:f0:0f:4d STALE
```



Úkol

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.



5.5.4 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 `ip6v6`) – slouží k propojení IPv6 sítí přes IPv4 síť.

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 příkaz `ip tunnel` (nebo jeho zkrácenou variantu).

`ip tunnel show mujtunnel` zobrazí informaci o vytvořeném tunelu s názvem `mujtunnel`, 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 mujtunnel` 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 105

`ip tunnel add mujtunnel 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 mujtunnel` zrušili jsme tunel

`ip tunnel change mujtunnel 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.



Poznámka:

Příkaz `ip` má i další podpříkazy. Jejich seznam najdeme v manuálové stránce:

`man ip`



5.6 Ovlivňování runtime informací o síti

V souborovém systému `proc` je většina souborů jen „pro čtení“ (k jejich obsahu se dostáváme pomocí `cat`, `head`, `tail` a podobných příkazů). Výjimkou je obsah podadresáře `/proc/sys`, do kterého se dá i zapisovat a tím ovlivňovat systém za jeho běhu (runtime).

To ale neznamená, že bychom tyto soubory mohli otevírat v běžném textovém editoru – zapisujeme pomocí *směrování* z příkazu `echo` nebo jiným způsobem generujícím textový výstup (ale `echo` je nejbezpečnější, jiné možnosti by mohly mít vedlejší efekty a nelze je mít pod přímou kontrolou). Další bezpečný způsob změny runtime informací je příkaz `sysctl`.



Příklad

Běžně se za běhu ovlivňuje třeba chování sítě. Například nastavení související s protokolem IPv4, ICMPv4 a dalšími najdeme v adresáři `/proc/sys/net/ipv4`. Je tam řada (pseudo)souborů, z jejichž názvů se dá poznat jejich účel, v obsahu těchto souborů často najdeme jen číslo (většinou 0 nebo 1, podle toho, zda daná vlastnost je či není nastavena).

Takže provedeme tyto příkazy:

- Přesuneme se do výše uvedeného adresáře:

```
cd /proc/sys/net/ipv4
```

- Ověříme si, co je v souboru `icmp_echo_ignore_all`:

```
cat icmp_echo_ignore_all
```

(velmi pravděpodobně tam bude 0). To znamená, že ICMP pakety nejsou ignorovány a například pokud někdo „pouští ping“ na náš systém (tj. posílá nám pakety ICMP Echo Request), tak je přijímáme a odpovídáme pakety ICMP Echo Reply.

- Můžeme si to ověřit, třeba budeme posílat ICMP Echo Request sami sobě:

```
ping -c4 127.0.0.1
```

(posíláme tyto pakety sami sobě, na loopback, celkem 4). Zobrazí se nám 4 odpovědi.

- Pokud budeme chtít toto chování upravit tak, že budeme chtít všechny ICMP pakety zahazovat (ignorovat) a ani neodpovídat na příkaz `ping`, provedeme to takto:

```
su - (potřebujeme oprávnění roota, nestačí sudo mechanismus)
```

```
cd /proc/sys/net/ipv4
```

```
echo 1 > icmp_echo_ignore_all
```

- Ověříme si, že to funguje, opět spustíme

```
ping -c4 127.0.0.1
```

Teď bychom měli dostat chybové hlášení, resp. informaci, že 4 pakety byly odeslány, ale žádný přijat, 100% paketů ztraceno.

- Návrat k původnímu stavu:

```
echo 0 > icmp_echo_ignore_all
```

Pokud teď zkusíme příkaz `ping` na loopback, už to bude zase fungovat, všechny 4 pakety dorazí. Případně ještě použijeme `exit`, abychom opustili rootovský shell.

Totéž se dá provést příkazem:

- `sysctl -w net.ipv4.icmp_echo_ignore_all=1` (zákaz ICMP)
- `sysctl -w net.ipv4.icmp_echo_ignore_all=0` (povolení ICMP)



Příklad

Pokud budeme chtít ne zakázat, ale omezit zpracování ICMP paketů jen na určitý počet za sekundu, máme dokonce možnost určit, kterých ICMP zpráv konkrétně se to má týkat. Znamená to zásah do dvou míst:

- V souboru `icmp_ratemark` je určeno, které ICMP zprávy budou takto omezeny a které ne. Je to číslo, u kterého po převedení do binární formy jednotlivé bity přísluší konkrétním typům zpráv, a pokud je příslušný bit nastaven na 1, znamená to, že pro daný typ platí omezení.

- V souboru `icmp_ratelimit` je limit (počet milisekund). Pak platí, že pokud přijde „omezený“ typ ICMP paketu, další „omezený“ bude přijat (ne zahozen) až po uplynutí daného limitu. Výchozí hodnota je 1000, co znamená 1 sekundu (1000 ms).

Pokud jsme to už neudělali dříve, provedeme přípravné práce:

```
su -  
cd /proc/sys/net/ipv4
```

Momentální nastavení zjistíme takto:

```
cat icmp_ratemask
```

Pravděpodobně se vypíše číslo 6168, což je hexadecimálně 0×1818 . V manuálové stránce ICMP(7) se dají zjistit pozice bitů pro příslušné typy zpráv. V našem případě jde binárně o řetězec 0001 1000 0001 1000 a příslušné pozice patří typům zpráv Parameter problem, time Exceeded, Source Quench a Destination Unreachable. Pokud bychom chtěli přidat na blacklist ještě Echo reply, musíme nastavit první bit zprava, což je binárně 0001 1000 0001 1001, hexadecimálně 0×1819 , dekadicky 6169.

Změnu bychom provedli takto:

```
echo 6169 > icmp_ratemask
```

Případnou změnu časového limitu bychom provedli v souboru `icmp_ratelimit`.



Další informace:

Řešení pro IPv6: <http://www.brocade.com/content/html/en/configuration-guide/fastiron-08030b-l3guide/GUID-72969BA1-DA31-4BA6-B96A-18DC9F94B270.html>



Proč se taková nastavení vlastně dělají? Protože některé typy DoS a DDoS útoků spočívají v tom, že se zařízení bombarduje obrovským množstvím ICMP zpráv, zejména výše uvedených typů. Pokud omezíme počet těchto paketů, kterými se opravdu budeme zabývat, zachováme funkčnost ICMP mechanismu a zároveň omezíme riziko útoku na naši síť či dané zařízení.



Příklad

Pokud chceme dané zařízení používat jako router, kromě jiného na něm musíme povolit přeposílání (forwarding). Předpokládejme, že máme více než jedno síťové rozhraní (jinak by moc nemělo smysl z takového zařízení dělat router). Přeposílání povolíme v runtime takto:

```
su -  
cd /proc/sys/net/ipv4  
echo 1 > ip_forward
```



Příklad

V runtime (tedy `/proc/sys`) můžeme také nastavit hodnotu pro výchozí TTL, které bude dosazováno do záhlaví většiny IP paketů (kromě těch, u kterých má smysl spíše TTL=1). Nastavení najdeme v souboru

```
/proc/sys/net/ipv4/ip_default_ttl
```

 (bude tam zřejmě 64, možná 128).

**Poznámka:**

Prakticky všechna zde uvedená nastavení platí pro IPv4 a typicky jsou společná pro různá síťová rozhraní. Nastavení IPv6 jsou již v naprosté většině konkrétně rozdělená podle rozhraní. Takže pokud například máme na zařízení jak ethernetový port, tak Wi-fi, nastavení IPv6 určená pro tato síťová rozhraní najdeme zde:

- `/proc/sys/net/ipv6/conf/eth0`
- `/proc/sys/net/ipv6/conf/wlan0`

(případně se mohou jmenovat jinak, podle toho, jak máme tato síťová rozhraní nazvaná). Například výchozí TTL (což je v IPv6 hop limit) se pro eth0 v runtime nastavuje změnou obsahu souboru `.../conf/eth0/hop_limit`.



Výše popsané postupy platí jen do nejbližšího restartu počítače, uvedená nastavení se provedou jen v runtime. Permanentní nastavení provádíme v konfiguračních souborech, tato konkrétně v souboru `/etc/sysctl.conf`. Například nastavení z prvního příkladu (ignorování všech ICMP Echo paketů) by představoval toto nastavení v daném souboru:

```
net.ipv4.icmp_echo_ignore_all = 1
```

Pokud provedeme změnu v konfiguračním souboru, musíme ji ještě dostat do runtime. Změna platná až po restartu systému nebo příslušné služby nebo provedením tohoto příkazu:

```
sysctl -p /etc/sysctl.conf
```

Změny v konfiguraci ICMP se dají provádět i ve firewallu pomocí příkazu `iptables` – taktéž můžeme nastavit například zahazování určitého typu ICMP paketů, a to třeba konkrétně jenom pro vnější rozhraní.

**Další informace:**

Vhodná nastavení souboru `/etc/sysctl.conf` pro server:

<https://www.cyberciti.biz/faq/linux-kernel-etcsysctl-conf-security-hardening/>



5.7 Počítání adres

Ukážeme si práci s příkazem `ipcalc`, který podle zadané IP adresy podsíť vypíše masku, inverzní masku, rozsah adres uzlů v zadané podsíti, broadcast adresu a počet možných uzlů v síti.

**Příklad**

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:

```
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). Takže nainstalujeme.

```
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
...
```

Instalace je hotova, konečně můžeme spustit příkaz, jako parametr zadáme IP adresu (pod)sítě včetně délky prefixu:

```
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*¹ (System Administrators Tool for Analyzing Networks) prochází celou sítí a provádí bezpečnostní testy. Je oblíbený jak u administrátorů, tak i u hackerů, i když každý z nich má samozřejmě jinou motivaci.

Také existují distribuce určené mimo jiné k testování sítě, známý je třeba Kali Linux.

¹Ke stažení na <http://www.porcupine.org/satan/>.

Nasazení systému

6.1 Běh systému

6.1.1 Start systému a proces `init`

Víme, že před startem systému je spuštěn zavadač (u Linuxu obvykle GRUB nebo LILO nebo ELILO). Tento zavadač nejdřív nabídne možnost spuštění nainstalovaných operačních systémů nebo program na kontrolu paměťových modulů.

 Vše, co souvisí se startem systému, je umístěno v adresáři `/boot`. Tento adresář musí být během startu vždy připojen, bez toho by žádný systém nemohl naběhnout, kdežto třeba adresář `/home` klidně může být v oddílu, který se při údržbě systému uživatelem `root` nebude připojovat (proto je domovský adresář uživatele `root` rovnou v kořenovém adresáři – aby bylo možné potenciálně rizikový `/home` nepřipojovat). Buď máme pro `/boot` vlastní oddíl (stačí malý; je to bezpečnější a odolnější proti chybám při startu systému), nebo je součástí hlavního oddílu v bodu připojení `/` (v `rootu`).

V adresáři `/boot` je následující:

- soubory, z nichž lze zavádět operační systémy (například jádro Linuxu `vmlinuz-verze`) nebo jiné nástroje, které běží bez asistence operačního systému (třeba `MemTest86+`),
- bootovací manažer (obvykle je tam adresář `grub` nebo podobný, podle toho, co používáme), včetně jeho konfiguračního souboru (u GRUBu to je `/boot/grub/grub.cfg` nebo podobně), v jednom z podadresářů je kód a moduly zavadače.

 Pokud máme GRUB verze 2 nebo vyšší, do konfiguračního souboru `/boot/grub/grub.conf` se nezasahuje, ale konfigurace je v jiných souborech:

- `/etc/default/grub` je textový soubor se základní konfigurací typu jak dlouho se má čekat na to, až si uživatel vybere systém ke spuštění, výchozí vybraná položka, jestli má systém při spuštění GRUBu pípnout, atd.
- `/etc/grub.d` je adresář obsahující řadu (spustitelných) souborů odpovídajících jednotlivým položkám v menu zobrazovaném GRUBem (různé operační systémy apod.). Součástí názvu každé položky je i určení pořadí v seznamu.

 Prvním (nultým) procesem podílejícím se na startu systému je tedy zavadač, třeba GRUB (ten funguje jako boot manažer i zavadač), a samotné jádro, které je zavadačem načteno do paměti.

Dalším důležitým prvkem při startu systému je proces `/sbin/init`. V běžných UNIXových systémech včetně Linuxu, se obvykle používá, ale někdy se můžeme setkat s jeho alternativami (například `systemd`, `upstart`, `SystemStarter`, atd.), což obvykle znamená, že sice existuje proces `init`, ale část jeho původních funkcí provádí jiný proces.

Úkolem tohoto procesu je dokončení procesu startu systému, jeho inicializace, načtení veškerých inicializačních skriptů, spuštění dalších procesů potřebných pro běh systému a přechod do normální fáze běhu systému. Proces `init` existuje po celou dobu běhu systému a zajišťuje také jeho korektní ukončení. Pokud z nějakého důvodu přestane `init` fungovat, jádro se dostane do stavu „kernel panic“, což je obdoba „modré obrazovky“ BSOD z Windows.

 Předpokládejme, že jsme na obrazovce zobrazené boot manažerem 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 přerušování, 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.
5. Proces `init` řídí zbytek zavádění systému, a to pomocí úrovně běhu (viz dále). Provádí inicializaci uživatelské části systému. Jsou připojeny souborové systémy, aktivovány bezpečnostní technologie, síťová rozhraní, spuštěny demony, nastaveny a nastartovány konzoly, a nakonec také na jedné z konzol (obvykle poslední) grafické prostředí.



Postup

Proces `init` má `PID=1` a `PPID=0` (to znamená, že nemá žádného rodiče). Teoreticky by měl být jediným procesem bez rodiče, ovšem hloubavý experimentátor by zjistil, že na systémech s novějším jádrem je ještě jeden takový proces. Jak si to ověřit: spustíme

```
ps -eo pid,ppid,comm,cmd
```

První řádek tohoto výpisu patří procesu `init`, a pokud máte správnou verzi systému, pak na druhém řádku je proces `kthreadd`, jehož `PID=2` a `PPID=0` (takže taky je bez rodiče):

```
1 0 init /sbin/init
2 0 kthreadd [kthreadd]
```

Všimněte si zvláštní položky v sloupci, ve kterém má být spustitelný soubor – `kthreadd` je totiž démon, který nevznikl z žádného souboru (žádné klonování a změna zdroje strojového kódu), ale je to „umělý“ proces, pod kterým běží vlákna jádra (kernel threads).

Zatímco `init` nemá rodiče, protože jeho rodič už nepracuje, tak `kthreadd` nemá rodiče, protože nikdy žádného neměl a nepotřeboval.





Úkol

Projděte si adresář `/boot`. Najděte soubor jádra a zjistěte, jaké další programy se pomocí tohoto mechanismu mohou spouštět (asi tam bude i MemTest86+). Zjistěte, jaký používáte zavaděč.



6.1.2 Informace o startu a běhu systému

V Linuxu máme běhové informace včetně těch o startu systému uloženy v adresáři `/proc`. Takže se můžeme podívat, co tam všechno je.

 `cat /proc/cmdline` vypíše informaci o tom, s jakými parametry náš systém nastartoval. Dozvíme se, jaké jádro bylo při bootování použito (můžeme mít víc jader, GRUB dokonce umožňuje volbu jádra při startu systému), dále jaké je UUID oddílu na disku, který je kořenovým oddílem (připojen jako `/`) a případně další informace.

 `cat /proc/version` vypíše verzi jak celého systému, tak i samotného jádra. Zjistíme verzi jádra (pravděpodobně i včetně toho, jak bylo přeloženo) a verzi své linuxové distribuce (nebo jiného UNIXového systému) včetně doby sestavení.

 `sudo ls -la /proc/1` vypíše přehled všeho, co souvisí s procesem `init`, protože PID tohoto procesu je právě 1 (případně můžeme přidat přepínač `-R` pro rekurzi).

Další užitečná informace je doba běhu systému – buď se podíváme do určitého souboru v `/proc`, nebo použijeme příkaz. Na obojí se podíváme v následujícím příkladu.

 `cat /proc/uptime` nám řekne, jak dlouho náš systém běží. Výstupem jsou dvě čísla, mohou být třeba takováto: 25801.56 19304.19. První číslo je počet sekund běhu systému, druhé je počet sekund, po které byl procesor ve stavu „idle“ (tj. nic nedělal). Za normálních okolností by druhé číslo mělo být menší než první, ale ve skutečnosti se běžně stává, že druhé je větší – čas strávený ve stavu „idle“ se totiž sčítá přes všechny procesory a přes všechna jádra procesoru. Takže když máme vícejádrový procesor...

 `uptime` jako příkaz přidá další informace. Výstupem je poněkud delší řetězec, ve kterém je druhou informací v pořadí délka běhu systému v delších jednotkách než sekundách (minuty, hodiny, dny – podle toho, co je aktuální). Na konci řetězce jsou tři údaje o vytížení systému (době práce jader) v uplynulých 1, 5 a 15 minutách. Může to vypadat třeba takto:

```
16:35:01 up 19 days, 4 users, load average: 0.45, 0.20, 0.14
```

 Všechny údaje můžeme zjistit také z prvního řádku výstupu příkazu `w`, je naprosto stejný jako výše uvedený řetězec.



Úkoly

1. Vyzkoušejte vše, co je v této sekci uvedeno – žádný z příkazů není destruktivní a většinou není nutné mít vyšší oprávnění (kromě plného přístupu do `/proc/1`, kde se bez vyšších oprávnění některé položky nevypíší).
2. Srovnajte výstup příkazů `uptime` a `w`.



6.1.3 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 je použita, se určuje, co má/nemá být spuštěno.

Úroveň	Význam
0	Zastavení systému (systém neběží)
1	Jednouživatelský režim
2–5	Uživatelské režimy (běžný provoz)
6	Znovuzavedení systému (reboot)

Tabulka 6.1: Obvyklé úrovně běhu systému

Jednouživatelský režim je používán pro administraci systému, když je potřeba, aby se rootovi 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 stanovenou výchozí volbu (v rozmezí 2–5).

 Momentálně platnou úroveň běhu zjistíme příkazem `runlevel` nebo `who -r`. V prvním případě se vypíše jednopísmenná zkratka (třeba „N“ pro normální režim) a číslo úrovně běhu. V druhém případě je výstup poněkud uživatelsky přívětivější, může vypadat třeba takto:

úroveň běhu 2 2017-04-27 16:28 (tedy druhá úroveň, a pak se vypíše momentální datum a čas).

 Změna úrovně běhu se provádí příkazem `telinit úroveň`, například přechod na třetí úroveň:

```
telinit 3
```

Pokud takto přejdeme na úroveň běhu 6, vyvoláme restart, přechod na úroveň 0 zase znamená vypnutí systému (nicméně je lepší pro tento účel používat příkaz `shutdown`). Při přechodu do jednouživatelského režimu můžeme místo čísla úrovně použít písmeno `S` nebo `s`:

```
telinit s
```

 Jinou než výchozí úroveň běhu také můžeme nastavit před startem systému, zavaděč obvykle umožňuje ji explicitně určit při výběru operačního systému, případně alespoň dovoluje spustit systém v `single-user` režimu.

 Každá úroveň běhu je charakteristická tím, že v ní určité služby či procesy běží/neběží. Přechod na jinou úroveň běhu znamená, že jsou zjištěny změny mezi původní a novou úrovní, a příslušné služby se buď spustí (když v původní nebyly a v nové jsou) nebo naopak ukončí (v opačném případě). Služby, které patří do obou úrovní, zůstanou běžet.

Například pokud přecházíme z úrovně používající grafické rozhraní na úroveň bez grafického rozhraní, je ukončen systém `X Window` a grafické rozhraní přestane fungovat (přejdeme do čistě textového režimu). Při přechodu opačným směrem je naopak `X Window` nastartován. Nebo pokud přecházíme z víceuživatelské úrovně na jednouživatelskou, všichni uživatelé kromě `roota` jsou automaticky odhlášeni a je ukončena podpora multiuživatelského režimu.

Úkol

Zjistěte momentální úroveň běhu systému, ve kterém pracujete. Vyzkoušejte obě možnosti.



6.1.4 Ukončení běhu systému

 Jak bylo výše uvedeno, systém se dá ukončit nebo restartovat přechodem do příslušné úrovně běhu, ale lepší je použít příslušné tlačítko nebo příkaz v textovém režimu – `shutdown`.



Příklad

Příklady použití tohoto příkazu:

- `shutdown now` způsobí okamžité vypnutí.
- `shutdown 18:20` systém se vypne v zadanou dobu (hodiny:minuty).
- `shutdown +10` systém se vypne za 10 minut.
- `shutdown +10 "Pozor, systém se za 10 minut vypne."` přihlášeným uživatelům se objeví zpráva o vypnutí a systém se vypne za 10 minut.
- `shutdown -r now` způsobí restart.
- `shutdown -c` přeruší se proces vypínání systému.



Přepínač `-P` (velké „P“) se použije, pokud systém po vypnutí nedokáže vypnout celý počítač (napájení) – power off.



Úkol

Vyzkoušejte si restart nebo vypnutí pomocí příkazu `shutdown`.



6.2 Inicializační systémy a služby

 Inicializační systém určuje, jak konkrétně má probíhat spouštění operačního systému, restart a další změny úrovně běhu. Účelem je především určení služeb, které se na dané úrovni běhu mají spustit či ukončit.

V každé linuxové distribuci se používá některý inicializační systém, tři nejběžnější si zde popíšeme – původní *sysvinit* ze System V, novější *upstart* a nejnovější *systemd*. Dnes většina linuxových distribucí buď používá třetí uvedený způsob nebo na něj postupně přechází.

Jednotlivé inicializační systémy se liší v těchto parametrech:

- konfigurační soubory nebo inicializační skripty služeb – jejich forma, celková struktura a umístění,
- konfigurovatelnost běhu služeb, případné závislosti mezi službami,
- postupy, příkazy.

 *Služba* (service) je aplikace běžící na pozadí bez jakékoliv vazby na přihlášené uživatele. *Démon* (daemon) je implementací některé služby, tedy konkrétní proces (takže démon je proces běžící na pozadí bez vazby na uživatele). Procesy, které jsou démony, obvykle mají na konci svého názvu písmeno „d“, například `sshd` (démon služby pro navazování a udržování SSH spojení), `syslogd` (démon služby pro logování), `httpd` (démon Apache pro službu webového serveru), `cupsd` (démon pro tiskové služby), `crond` (démon pro službu plánování spouštění úloh),...

Většina démonů má jako rodiče proces `init`, a spouští se přes systém úrovní běhu, jak bylo výše naznačeno.

 Jak zajistit, aby se určitá služba spustila či ukončila:

- přidáme službu do dané úrovně běhu, resp. odebereme ji z dané úrovně běhu (to se obvykle provádí automaticky při instalaci služby), někdy je třeba ručně přidat automatické spouštění služby po crashi (pádu) služby,
- použijeme příkaz pro spuštění či ukončení služby.

Pokud instalujeme novou službu (s démonem, konfiguračními soubory apod.), součástí instalačního procesu by měla být registrace (někdy je třeba ji udělat ručně).

 Postupy pro práci se službami se liší podle toho, jestli distribuce používá původní proces `init` (systém *sysvinit*) nebo některou jeho novější variantu či doplnění (v Linuxu *upstart* nebo *systemd*). V následujících podsekcích jsou postupy charakteristické pro všechny tři inicializační metody.

6.2.1 Služby podle *sysvinit*

Původní inicializační systém pro UNIXovou větev System V (*sysvinit*) dnes používá jen málokterá distribuce, je to spíše záležitost starších verzí. Například Debian používal *sysvinit* do verze 6, Ubuntu do verze 9.04, RedHat do verze 5.

 Konfigurace je vícestupňová. První stupeň: *sysvinit* používá hlavní konfigurační soubor procesu `init` v „rozstřelovém“ souboru `/etc/inittab`, kde je například informace o tom, co se má dělat při stisknutí `Ctrl+Alt+Del`, která úroveň běhu je výchozí a který skript se má spustit při přechodu na konkrétní úroveň běhu – obvykle `/etc/init.d/rc`, jehož parametr je číslo úrovně.

Příklad

Po zadání příkazu `cat /etc/inittab` na takto koncipovaném systému se mohou objevit například 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 jednorázovém 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, v našem případě jde o soubor `/etc/rc.d/rc` (jeho první parametr bývá číslo úrovně běhu, která se má nastavit).



Druhým stupněm je tedy soubor `/etc/init.d/rc` spouštěný vždy s číslem úrovně jako parametrem. Hlavním úkolem tohoto skriptu je spuštění spustitelných souborů pro danou úroveň, které najde v adresářích `/etc/rcx.d` (místo písmene „x“ dosadíte čísla úrovně běhu, třeba `/etc/rc2.d` pro druhou úroveň).

Právě obsah adresářů `/etc/rcx.d` je třetím stupněm. Kromě symbolických odkazů tam najdeme vždy i soubor `README`, ve kterém je popsána daná úroveň.

Čtvrtým stupněm jsou samotné inicializační skripty jednotlivých služeb v adresáři `/etc/init.d/`, tyto soubory jsou pojmenované podle jednotlivých služeb. Výše zmíněné symbolické odkazy v adresářích `/etc/rcx.d` vedou právě na ně.



Příklad

Například cesta v této hierarchii ke konkrétní službě běžící v úrovni běhu 2 by byla taková:

`/etc/inittab` $\Rightarrow$ `/etc/init.d/rc 2` $\Rightarrow$ `/etc/rc2.d` $\Rightarrow$ `/etc/init.d/služba`

Vypíšeme si obsah adresáře `/etc/rc2.d`:

```
ls -l /etc/rc2.d
```

```
-rw-r--r-- 1 root root 677 ... README
lrwxrwxrwx 1 root root 21 ... S05loadcpufreq -> ../init.d/loadcpufreq
lrwxrwxrwx 1 root root 22 ... S19cpufrequitls -> ../init.d/cpufrequitls
lrwxrwxrwx 1 root root 17 ... S20hddtemp -> ../init.d/hddtemp
...
```

Takže opravdu je tam soubor `README` a spousta symbolických odkazů. Obsah souboru `README` bychom si zobrazili takto:

```
cat /etc/rc2.d/README
```

(pozor na velká písmena, UNIXové systémy jsou case-sensitive)



Pokud služba patří do více úrovní, není její inicializační skript ve více kopiích, ale na jeden skript vede více symbolických odkazů z adresářů pro různé úrovně běhu. Tyto symbolické odkazy mají ve svém názvu dva údaje – pořadové číslo služby a název služby. Pořadové číslo určuje pořadí, ve kterém mají být služby spouštěny při přechodu na danou úroveň, při opouštění úrovně se služby vypínají v opačném pořadí (tj. od nejvyššího čísla).



Příklad

Pro službu synchronizace (démon `rsync`) existují tyto položky:

- `/etc/init.d/rsync` je inicializační soubor služby, můžeme si jeho obsah prohlédnout třeba pomocí příkazu `cat` (je to textový soubor, a ve výstupu příkazu `ls -l` si můžeme všimnout, že má nastaven přístup „x“ pro spouštění, je to tedy shellový skript, jeho první řádek je `#!/bin/sh`),
- `/etc/rc2.d/S20rsync` je symbolický odkaz; umístění v `rc2.d` znamená, že patří do úrovně běhu 2, číslo 20 v názvu znamená, že služba má být spouštěna až po službách s nižším číslem a před službami s vyšším číslem, písmeno „S“ určuje, že je to spouštěcí skript (ukončovací mají „K“).

Pomocí příkazu `ls -l /etc/rc2.d` bychom zjistili, že cílem symbolického odkazu je soubor z první odrážky. Přehled symbolických odkazů přes všechny úrovně běhu bychom získali takto:

```
ls /etc/rc*.d
```



 Vytvoření inicializačního skriptu služby a příslušného symbolického odkazu v dané úrovni běhu je obvykle součástí instalace služby a tím je zajištěn automatický start služby (což vlastně znamená zařazení do dané úrovně běhu). Pokud ale chceme, aby služba automaticky naběhla i po crashi (pádu) služby, musíme do příslušného skriptu ručně zasáhnout.

 Výchozí úroveň běhu je určena buď v souboru `/etc/inittab`, nebo nověji ji najdeme spíše v souboru `/etc/init/rc-sysinit.conf`. Tam je například řádek (zde určující druhou úroveň):

```
env DEFAULT_RUNLEVEL=2
```

Základní parametry služby (i případná registrace pro automatické spuštění, pokud neproběhla automaticky) se nastavují některým z následujících programů.

 Příkaz `service` umí předávat příkazy službám. Především dokáže službě říct, že se má spustit, zastavit a vypsat svůj stav, případně restartovat (pokud to podporuje daná služba).

```
service služba start    spustí zadanou službu
service služba stop     zastaví zadanou službu
service služba status   zobrazí stav služby
service --status-all    zobrazí stav všech služeb, v abecedním pořadí, výstup může vypadat takto:
```

```
[ + ] acpid
[ - ] anacron
[ + ] avahi-daemon
...
```

 Poněkud zvláště nazvaný příkaz `update-rc.d` slouží k podrobnější práci se službami. Můžeme službu registrovat k automatickému spuštění při startu systému, zařadit do konkrétní úrovně, atd.

```
update-rc.d služba default    registrujeme službu, přičemž se použijí výchozí nastavení (zařadí se do
                               úrovně pro běžný provoz 2–5)
```

```
update-rc.d služba start 20 3 4 5 . stop 80 0 1 2 6 .    určíme, že služba má běžet v úrovních
                                                           3–5 s pořadovým číslem 20, nemá běžet v úrovních ostatních (pořadové číslo pro ukončení je
                                                           doplněk do 100, takže 100 – 20 = 80), tečky jsou nutné
```

```
update-rc.d služba remove    odstraníme všechna přiřazení služby do úrovně, ale předem musíme sma-
                               zat skript služby (pokud jsme to neudělali, musíme použít přepínač -f)
```

Tento příkaz se moc nepoužívá, jeho používanější alternativou je následující. Začínající uživatele totiž u `update-rc.d` poněkud mate „adresářová“ přípona.

 V mnoha distribucích se místo předchozího příkazu používá novější příkaz `chkconfig`. Jeho syntaxe je trochu jednodušší.

```
chkconfig --list    zobrazí se seznam dostupných služeb, u každé přehled, na kterých úrovních běží,
                    řádky vypadají například takto:
```

```
ip6tables    0:off    1:off    2:on     3:on     4:on     5:off    6:off
```

```
chkconfig --add služba    služba se zařadí do úrovně běhu s výchozími nastaveními (tj. obvykle do
                            úrovně běhu 2–5)
```

```
chkconfig --level 23 služba on    službu jsme aktivovali v úrovních 2 a 3 (čísla úrovní píšeme prostě
                                   do řetězce bez mezer)
```

```
chkconfig --level 5 služba off    naopak teď jsme službu vypnuli na úrovni 5
```

```
chkconfig --del služba    odstranění služby ze všech úrovní
```



Další informace:

- <https://www.debuntu.org/how-to-managing-services-with-update-rc-d/>
- <http://manpages.ubuntu.com/manpages/precise/man8/service.8.html>
- <http://www.thegeekstuff.com/2011/06/chkconfig-examples>



Úkoly

1. Novější inicializační systémy *upstart* a *systemd* jsou do určité míry kompatibilní, tedy výše uvedené adresáře a soubory (asi kromě `/etc/inittab`) s největší pravděpodobností budete mít, i když nepoužíváte *sysvinit*. Projděte si tyto soubory, vyberte si některou službu a vypište si její inicializační skript.
2. Zjistěte, ve kterém souboru je zadána výchozí úroveň běhu (tedy taková, ve které má systém trávit nejvíc času, běžný provoz). Zřejmě to bude soubor `/etc/init/rc-sysinit.conf`. Vypište si obsah tohoto souboru, včetně řádku s výchozí úrovní. Toto opět platí i pro další inicializační systémy.
3. Pokud vaše distribuce používá *sysvinit*, zjistěte, který příkaz platí pro vaši distribuci (bývá to jeden z výše uvedených). Pro tento příkaz byste měli mít k dispozici i manuálovou stránku. Vyzkoušejte si výpis stavu služeb.



6.2.2 Služby podle *upstart*

Systém *upstart* používal například RedHat 6, CentOS 6, Ubuntu do verze 14.10 a dodnes se používá v ChromeOS a některých dalších systémech včetně embedded zařízení. Také ho můžeme najít v některých distribucích vycházejících z Ubuntu, které ještě nepřešly na *systemd*, nebo na strojích, kde byla instalována starší verze distribuce a na novou verzi se přešlo upgradem, ne přeinstalováním.

Oproti *sysvinit* přidává určité možnosti konfigurace služeb (u *sysvinit* jsou prostě symbolické odkazy na to, co se má spustit) a možnosti závislostí mezi službami, závislostí je míněno třeba to, že daná služba se má spustit hned po spuštění „nadřízené“ služby. Výhodou systému *upstart* je, že služba je nejen automaticky registrována, ale také je (většinou) automaticky zajištěno zotavení po pádu služby.

 Pro služby se zde používá pojem *job* (úloha), s úlohami souvisejí *events* (události) – událostí může být přechod na jinou úroveň běhu, a úloha na tuto událost má reagovat (spuštěním, zastavením apod.).

 Konfigurační soubory (pozor, ne skripty) jednotlivých služeb jsou v adresáři `/etc/init`, každý soubor je pojmenován podle příslušné služby a přípona souboru je `.CONF` (například `udev.conf`). Význam adresářů `/etc/rcx.d` je stejný, také obsahují symbolické odkazy na služby. Služby kompatibilní se systémem *upstart* mají své inicializační soubory (na které vedou symbolické odkazy) taktéž v adresáři `/etc/init.d`, jen navíc mají i konfigurační soubory v `/etc/init`.

Systém je zpětně kompatibilní s *sysvinit*, starší služby jednoduše nemají konfigurační soubor.

 V konfiguračním souboru služby najdeme jeden nebo více bloků, kterým se říká *stanza*, v nich je stanoveno, co se má provést v konkrétním případě. Například tam bývá buď stanza, která říká, že se

při spouštění služby má spustit konkrétní spustitelný soubor, nebo stanza určující, že se v tom případě má spustit určitý skript s konkrétními parametry (resp. bývá zde zapsán přímo kód skriptu), případně tam může být určeno, že se má služba spustit po startu jiné služby (tj. i mezi službami mohou být závislosti), přičemž se opět spustí určitý spustitelný soubor nebo zadaný skript.

Nejjednodušší forma prostě říká, za jakých okolností má být služba spuštěna či zastavena (uvedením úrovně běhu nebo závislosti na jiné službě) plus informace o tom, co má být spuštěno (třeba spustitelný soubor).



Příklad

Vypíšeme si obsah jednoho z konfiguračních souborů:

```
cat /etc/init/acpid.conf

# acpid - ACPI daemon
#
# The ACPI daemon provides a socket for other daemons to multiplex kernel
# ACPI events from, and a framework for reacting to those events.

description "ACPI daemon"

start on runlevel [2345]
stop on runlevel [!2345]

expect fork
respawn

exec acpid -c /etc/acpi/events -s /var/run/acpid.socket
```

Všimněte si, jak je určeno, na kterých úrovních běhu má či nemá tato služba běžet – „povolené“ úrovně jsou 2, 3, 4 a 5, „zakázané“ jsou všechny kromě těchto (vykřičník zde znamená negaci). Příkaz `respawn` určuje, že pokud dojde k pádu služby, má být znovu spuštěna.

Poslední řádek určuje, jak má být daný proces spuštěn, když se tedy přechází na některou z „povolených“ úrovní. To si můžeme ověřit tak, že si zobrazíme seznam procesů s tímto názvem:

```
ps -fC acpid

UID  PID  PPID  C  STIME  TTY      TIME  CMD
root  945    1   0   16:56  ?        00:00:00  acpid -c /etc/acpi/events -s /var/run/acpid.socket
```



 Výchozí úroveň běhu je stanovena v souboru `/etc/init/rc-sysinit.conf`, v tom je *upstart* zpětně kompatibilní. Takže pokud chceme nastavit jinou výchozí úroveň, provádí se to zde.

Pro práci se službami používáme následující příkazy.

 Příkaz `service` funguje podobně jako u *sysvinit*, lze použít podpříkazy `start`, `stop`, `status` a případně `restart`.

 Příkaz `initctl` taktéž umí pracovat se stavy služeb:

```
initctl start služba    spustí zadanou službu, volitelně je také možné předat službě parametry
initctl stop služba    zastaví službu, opět můžeme přidat parametry
initctl restart služba  restartuje službu, taktéž s případnými parametry
```

`initctl reload služba` donutí službu, aby znovu načetla svou konfiguraci (ale ne všechny služby tento příkaz podporují), provádí se posláním signálu SIGHUP zadané službě

`initctl status služba` zobrazí informace o službě (stav)



Příklad

Pokud chceme znát stav služby bluetooth, zadáme jeden z těchto příkazů:

- `service bluetooth status`
- `initctl status bluetooth`

(všimněte si, že druhé a třetí slovo je v příkazech naopak). Výstup bude v obou případech (samozřejmě s jiným PID procesu) takový:

```
bluetooth start/running, process 1094
```



Další informace:

- <https://www.digitalocean.com/community/tutorials/the-upstart-event-system-what-it-is-and-how-to-use-it>
- <https://linux.die.net/man/8/initctl>
- <http://upstart.ubuntu.com/cookbook/>



Úkoly

1. Pokud vaše distribuce používá inicializační systém *upstart*, podívejte se do adresáře `/etc/init`, vyberte si několik konfiguračních souborů (například pro služby `udev`, `shutdown` a `cups`) a zobrazte si jejich obsah pomocí příkazu `cat` (nezapomeňte na příponu `.CONF`).
2. Vypište si stav všech služeb pomocí `service --status-all`, některou z vypsaných služeb si vyberte a vypište pouze její stav.



6.2.3 Služby podle *systemd*

Inicializační systém *systemd* se používá v naprosté většině současných distribucí. Celý mechanismus je robustnější a poskytuje víc možností například pro práci se závislostmi, paralelní činností služeb a jejich různými stavy.



Službu nebo cokoliv jiného, co má být takto plánováno, označujeme pojmem *unit* (jednotka). Kromě služby může být jednotkou například socket, přípojný bod, timer, swap, zařízení, atd.



Konfiguraci jednotlivých služeb najdeme v adresářích `/etc/systemd/system` (konfigurace určená adminem) a `/usr/lib/systemd/system` (konfigurace určená instalačním procesem služby). Jsou tady soubory pojmenované podle služeb s příponou `.SERVICE` (kdežto soubory jednotek typu zařízení mají příponu `.DEVICE` apod.).

 Mohou tady být i adresáře, když služba vyžaduje podrobnější konfiguraci (stejně jméno, jen přípona například .D). Adresáři s určitými příponami a obsahem ve formě symbolických odkazů jsou řešeny i závislosti mezi službami.

V adresářích `/etc/rcx.d` (místo „x“ číslo úrovně) jsou pak symbolické odkazy na služby patřící do konkrétních úrovní běhu. Výchozí úroveň běhu se opět stanovuje v souboru `/etc/init/rc-sysinit.conf`.

 Používá se příkaz `systemctl`, který má celou řadu parametrů a přepínačů.

`systemctl` (bez parametrů) vypíše seznam jednotek (tj. služeb)

`systemctl list-units` totéž

`systemctl list-unit-files` vypíše seznam souborů pro jednotlivé služby

`systemctl status` zobrazí momentální stav služeb

`systemctl --failed` vypíše seznam služeb, které „spadly“

`systemctl status služba` zobrazí stav konkrétní služby

`systemctl start služba` spustí službu

`systemctl stop služba` ukončí službu

`systemctl restart služba` restartuje službu

`systemctl reload služba` donutí službu k znovunačtení konfigurace

`systemctl show služba` vypíše vlastnosti dané služby

`systemctl enable služba` registruje službu k automatickému spuštění při startu systému (například po předchozím odregistrování příkazem `disable`)

`systemctl list-dependencies služba` zobrazí strom závislostí pro zadanou službu

`systemctl is-enabled služba` ověřujeme, zda je zadaná služba registrovaná ke spuštění při startu systému, podobně lze ověřovat i další stavy (`is-failed`, `is-active`, atd.)

Podpříkazů je ve skutečnosti mnohem více (podrobnosti na manuálové stránce příkazu). Místo „služba“ by byl název konkrétního démona, třeba `httpd` nebo `mysqld` či `crond`. Správně bychom měli psát vždy i příponu `SERVICE` (například `crond.service`), ale příkaz si tu příponu obvykle dosazuje sám.

Místo názvu jedné služby můžeme zadat víc služeb nebo použít zjednodušené regulární výrazy (hvězdičku apod.) k interpretaci shellem.

Také existuje grafický frontend *systemadm* v balíčku `systemd-ui`.

 Poznámka:

Zpětná kompatibilita jde do důsledků – v systému mohou běžet i starší služby programované pro *sysvinit*. Pokud chceme pracovat pouze s těmito staršími službami, používáme i „starší“ příkazy včetně `chkconfig`. Také obvykle funguje příkaz `service` (buď přímo nebo je překládán na volání `systemctl`).

 Další informace:

- <https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units>
- https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/sect-Managing_Services_with_systemd-Unit_Files.html
- <https://www.cyberciti.biz/faq/check-running-services-in-rhel-redhat-fedora-centoslinux/>

- <https://www.freedesktop.org/software/systemd/man/systemctl.html>
- <https://wiki.archlinux.org/index.php/systemd>



Další informace:

Ještě pár odkazů vztahujících se k inicializačním systémům obecně:

- <https://www.digitalocean.com/community/tutorials/how-to-configure-a-linux-service-to-start-automatically-after-a-crash-or-reboot-part-1-practical-examples> (je zde na příkladu se službou `mysqld` ukázán postup pro všechny tři typy *initu*)
- https://wiki.gentoo.org/wiki/Comparison_of_init_systems



Poznámka:

Démoni služeb se od běžných procesů liší v mnoha ohledech. Kromě toho, že rodičem většinou bývá `init`, je také zajímavé, že nedokážou přijímat data ze standardního vstupu (`stdin`), je automaticky směřován na `/dev/null`. Je to bezpečnostní opatření, aby se zabránilo uživatelským pokusům o ovlivnění běhu služby.



6.3 Instalace aplikací

Jak už víme, aplikace pro Linux jsou obvykle dostupné v *repozitářích*. 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 jeden nebo více oficiálních repozitářů na webu naší linuxové distribuce. Z těchto repozitářů můžeme instalovat nový software, ale především jsou zde zajišťovány automatické aktualizace.

6.3.1 Binární a zdrojové balíčky

Postup instalace se liší podle toho, v jaké formě jsou instalační soubory. Použijeme příslušný nástroj (pro RPM je to program `rpm`, `yum` nebo `urpmi` (některé distribuce přidávají i další podobné programy), pro DEB je to `dpkg` nebo `apt`, atd.), ve většině distribucí existují grafická rozhraní k těmto programům. S některými jsme se seznámili v minulém semestru.

 *Binární balíčky* obsahují přeložený kód (spustitelný soubor, knihovny apod.), konfigurační soubory a instalační rutinu spolupracující s daným balíčkovacím systémem, to vše zabaleno do „archivu“, kdežto *zdrojové balíčky* v sobě mají místo přeloženého kódu zdrojové soubory a součástí instalačního procesu je jejich překlad. Není to však jen surový zdrojový kód, ale opravdu balíčky s dodanou instalační rutinou.

Následující výklad je opět rozdělen, protože většina linuxových distribucí používá jeden ze dvou systémů pro softwarové balíčky – RPM nebo DEB. Postupy pro textový režim si ukážeme na příkladech.

 **RPM balíčky.** Balíčky v systému RPM jsou soubory s příponou .RPM, tvar názvu souboru zachovává tuto strukturu:

název_balíčku-verze-release.architektura.rpm

Většinou budeme instalovat balíčky pro architekturu i686 (64bitové systémy kompatibilní s Intelem). Pokud u balíčku není uvedena přípona, pak obvykle stačí uvést první část (název balíčku); verze a release se doplní aktuální, architektura dle daného systému.

 Příklad

 V distribuci s RPM balíčky (tj. v distribucích odvozených z RedHat):

`rpm -q balicek` vypíše, zda je balíček nainstalován

`rpm -qi balicek` zobrazí podrobnější informaci o balíčku (query for information)

`rpm -qip balicek.rpm` zobrazí informaci o souboru balíčku

`rpm -ivh balicek.rpm` (případně další parametry, jsou v `man rpm`) – 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 (pokud není nainstalován, tak ho nainstaluje)

`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.

 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

 Další alternativou je nástroj `yum` (v parametru můžeme uvést i víc balíčků oddělených mezerou):

`yum install balicek` instalace balíčku

`yum remove balicek` odstranění balíčku

`yum update balicek` aktualizace balíčku



 **DEB balíčky.** Balíčky v systému DEB jsou soubory s příponou .DEB, tvar názvu souboru zachovává tuto strukturu:

název_balíčku-verze-release.architektura.deb

(takže stejná struktura jako u RPM). Příkazy pro práci se softwarem vyžadují vyšší oprávnění, takže správně bychom měli vždy před příkazem na začátek řádku připsat `sudo`.



Příklad

V distribucích s DEB balíčky (odvozených od distribuce Debian) se setkáme s nástrojem `dpkg` a hlavně 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 vypsaný)
apt-get -u dist-upgrade        aktualizace celé distribuce (význam přepínače je stejný jako u předchozího
                                příkazu), předem bychom měli provést update zdrojů softwaru
apt-get check                  zkontroluje databázi nainstalovaných balíčků a zjistí případné nesrovnalosti (poško-
                                zenou instalaci, problémy v závislostech mezi balíčky apod.), nic neinstaluje, pouze zkontroluje
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
```



Poznámka:

Místo `apt-get` je obvykle možné zadávat i jen zkrácenou variantu `apt`. Tedy například místo `apt-get update` stačí zadat jen `apt update`.



Příklad

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/ updpgk/
deb-src http://www.tug.org/texlive/Debian/ updpgk/

# 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).

Po každé změně je třeba aktualizovat vnitřní databázi mechanismu *apt*, například příkazem `apt-get update`.



Další informace:

Další informace najdeme například na

- <http://www.debian.org/doc/manuals/apt-howto/index.cs.html>
- <http://www.tfug.org/helpdesk/linux/rpm.html>
- <https://www.debian.org/doc/manuals/debian-faq/ch-pkg-basics.en.html>
- <https://www.tecmint.com/20-practical-examples-of-rpm-commands-in-linux/>



Úkol

Ujasněte si, který typ balíčků používá vaše distribuce. Podle typu balíčkovacího systému vyzkoušejte alespoň příkazy pro zjišťování informací o balíčcích a repozitářích.



6.3.2 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 popsáním 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 (předpokládáme, 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

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).

6.3.3 Aktualizace balíčků

Co se týče aktualizací, jsou řešeny tímtož mechanismem jako instalace – v repozitáři postupně přibývají nové verze daného balíčku (jak víme, v názvu souboru balíčku je i číslo verze a release, takže se tyto soubory „nebijou“). Aktualizační systém pravidelně prochází repozitář a srovnává seznam verzí

instalovaných balíčků se seznamem jejich verzí v repozitáři. Pokud najde aktuálnější verzi, buď ji automaticky nainstaluje nebo se zeptá, zda tak má učinit (podle nastavení).

Velkou výhodou je (oproti Windows), že v jediném systému je zahrnuta aktualizace jak systému, tak i všeho ostatního, i běžných aplikací. Výjimkou jsou jen balíčky, které jsme instalovali jinak než z repozitáře, třeba ručně stažené z internetu.

Většina uživatelů aktualizuje přes příslušnou aplikaci v grafickém rozhraní, ale samozřejmě to jde i příkazem.

 V systému používajícím balíčkovací systém DEB můžeme použít tyto příkazy:

- `apt-get install balíček` buď provede instalaci (když balíček není nainstalován) nebo aktualizaci (když je instalována starší verze)
- `apt-get update` pouze aktualizuje seznam balíčků podle repozitáře (aby bylo možné zjistit nový software nebo nové verze existujícího softwaru)
- `apt-get upgrade` aktualizuje všechny instalované balíčky



Příklad

Z toho vyplývá, že pokud v systému založeném na Debianu (vč. Ubuntu, Mint a dalších) chceme provést aktualizaci všech balíčků, provedeme následující dva příkazy:

```
sudo apt update
```

```
sudo apt upgrade
```

Můžeme být také na něco dotázáni nebo požádáni o potvrzení některé dílčí operace.



 V systému používajícím balíčkovací systém RPM máme k dispozici tyto možnosti:

- `rpm -Fvh balíček` zadaný balíček aktualizujeme na novou verzi (můžeme použít i parametr `-U` místo `-F`, ale pak se balíček přeinstaluje i tehdy, když už máme aktuální verzi)
- `rpm -Fvh *.rpm` takto aktualizujeme všechny nainstalované balíčky
- `yum upgrade balíček` pokud máme v distribuci program `yum`
- `up2date -i balíček` pokud máme v distribuci program `up2date`



Úkol

Ve své distribuci si vyzkoušejte aktualizaci v textovém režimu.



6.3.4 Přechod na novou verzi distribuce

Přechod na novou verzi distribuce je věc často odkládaná, ale může přijít doba, kdy ho musíme provést. Na klientských zařízeních bývá obvykle bezproblémová a většinou není důvod váhat, ale na serverech může určitý problém nastat, proto se obvykle na servery instalují déle podporované verze (LTS – Long Term Support), které delší dobu dostávají aktualizace. Ale i přesto může být nutné na novou verzi přejít.

Víme, že balíčky v repozitáři jsou členěny podle typu, v podstatě tvoří strom. V repozitáři jsou pro každou verzi samostatné stromy balíčků, takže přechod na novou verzi znamená, že nejdříve se

v repozitáři přepneme na strom pro novou verzi, pak synchronizujeme své informace o verzích balíčků a provedeme hromadný upgrade, také je v některých případech nutné řešit nové závislosti nebo změny v konfiguraci.

 Nejdrív se podíváme na postup pro distribuce používající balíčkovací systém DEB:

1. aktualizujeme seznam balíčků:
`apt update`
2. provedeme upgrade celé distribuce:
`apt dist-upgrade`

Proces bude samozřejmě nějakou dobu trvat, ale v jeho průběhu můžeme systém dle libosti využívat. Průběžně můžeme být žádáni o potvrzení některých kroků, hlavně v případě, že je třeba změnit obsah některých konfiguračních souborů, většinou můžeme s klidným srdcem potvrzovat. Alespoň na serveru si však pro jistotu ověřujeme dopady takových rozhodnutí.

Po provedení aktualizace je vhodné restartovat a ověřit, zda všechno proběhlo v pořádku. Mělo by být použito nové jádro (jádro se totiž taky dá aktualizovat) a měly by najet všechny potřebné služby.

 V distribucích používajících balíčkovací systém RPM je situace různorodější a složitější, protože celý balíčkovací systém je jinak postaven než u systému DEB (jinak je řešeno oddělení verzí). Často se používá aplikace v grafickém rozhraní, může být k dispozici program DNF (příkaz `dnf`) nebo jiný, specifický pro konkrétní distribuci.



Další informace:

- https://fedoraproject.org/wiki/DNF_system_upgrade
- <https://wiki.centos.org/TipsAndTricks/CentOSUpgradeTool>
- <https://serverfault.com/questions/298146/yum-equivalent-to-apt-get-upgrade-vs-apt-get-dist-upgrade>
- <https://wiki.centos.org/TipsAndTricks/YumAndRPM>



6.4 Log soubory

 Soubor `/var/log/dmesg` obsahuje informace o průběhu bootování (startu systému). Tento soubor se při každém startu systému celý přepíše, najdeme tam jen poslední provedené bootování (starší verze se archivují). Obvykle neprohlížíme přímo tento soubor, ale pro jeho výpis používáme příkaz `dmesg`.

Je zde zdokumentován celý postup startu včetně načítání jádra, inicializace nejrozličnějších rozhraní, protokolů, spouštění správy paměti, přidělování adres, atd.

Příkaz `dmesg` samozřejmě používáme, pokud systém nespouští podle našich představ, ale také v jiných situacích. Součástí vypisovaných informací jsou i řádky o připojených ovladačích zařízení a jiných modulech, které se při startu systému načítají do jádra. Takže když si chceme ověřit, zda se nový ovladač či jiný modul načel bez problémů, můžeme pro to použít příkaz `dmesg`.

 V souboru `/var/log/boot.log` máme stručnější verzi popisu startu systému, v podstatě to je totéž, co se vypisuje na jednu z textových konzol během startu systému (to může být buď první nebo předposlední, na poslední konzoli obvykle běží X Window). Nenajdeme tady záznamy o modulech jádra (jen info o tom, že byly načteny) ani inicializovaných hardwarových rozhraní či adresách, ale

spíše informace o připojování souborových systémů (včetně `proc`), spuštění inicializačního systému (`sysvinit`, `upstart` apod.), informace o spuštění démonů, atd.

Takže je to obraz startu spíše vzhledem k uživatelskému prostoru, zatímco `dmesg` je obraz startu z pohledu jádra.

 Informace o činnosti jádra jsou v jednom ze souborů `/var/log/messages` nebo `/var/log/syslog`. Do tohoto logu se ukládají veškeré zprávy od jádra, včetně oznámení o fungování služeb nebo případných problémů s moduly jádra včetně ovladačů zařízení.

Pokud jsme připojeni k síti, měli bychom v tomto souboru kromě jiného najít i průběh získávání IP adresy, zjišťování adresy DNS serveru, u wi-fi použité frekvence, atd.

Příklad

Pokud chceme zjistit, kdy a jak probíhala autentizace *na síti*, stačí použít příkaz

```
cat /var/log/syslog | grep -i authentication
```

(příp. dosaďte jiný název souboru – podle toho, co se používá ve vaší distribuci). Také je možné použít řetězec `eap` nebo název síťového zařízení, třeba `eth0` nebo `wlan0`.

Seznam chyb si zobrazíme takto:

```
cat /var/log/syslog | grep -i error
```

(ale pozor, ve výstupu budou i řádky, kde se tento řetězec vyskytuje v parametru určujícím, co se má dít při výskytu chyby, třeba při připojování souborových systémů, pak by o chybu samozřejmě nešlo). Jiné klíčové slovo je `failed`, při chybě se na daném řádku vypisuje jedno nebo druhé nebo obě.



 Pokud se nacházíme v systému založeném na Debianu, pak v souboru `/var/log/auth.log` najdeme záznamy o autentizaci na daném zařízení (všimněte si rozdílu oproti postupu v předchozím příkladu). Takže poslední záznamy o tom, kdy se kdo přihlašoval, ale taky kdy kdo použil příkaz `sudo` (a tedy získal práva roota) nebo se přihlašoval přes SSH, zjistíme takto:

```
cat /var/log/auth.log
```

 V systémech založených na RedHatu se pro tyto účely používá soubor `/var/log/secure`.

V adresáři `/var/log` jsou i další log soubory, také od jednotlivých služeb. Mnohé jsou ve vnořených adresářích. Jejich seznam získáme jednoduše tak, že si vypíšeme obsah adresáře `/var/log`.

Úkoly

1. Prohlédněte si adresář `/var/log` včetně vlastností souborů (takže `ls -l /var/log`). Všimněte si, které soubory jsou nejdelší (údaj před datem poslední změny). Všimněte si, že některé soubory mají varianty s „číselnou“ příponou – to jsou starší verze, aby aktuální log nebyl moc velký.
2. Pomocí příkazu `dmesg` si projděte log o startu systému, pokud na to máte dostatečná oprávnění. Také si prohlédněte soubor `/var/log/boot.log` a srovnajte.
3. Zjistěte, jestli se ve vaší distribuci používá pro záznam běhu systému soubor `/var/log/messages` nebo `/var/log/syslog`. Prohlédněte si alespoň konec tohoto souboru (například příkazem `tail`), nebo použijte stránkovač (například `less`).

Pokud máte možnost, vyzkoušejte postupy z uvedeného příkladu.

4. Vypište si obsah souboru s údaji o místní autentizaci (název souboru zvolte podle toho, v jakém jste systému). Používal někdo v poslední době mechanismus `sudo`?



6.5 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*.

6.5.1 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.

6.5.2 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  cíl           kategorie.!priorita  cíl
kategorie.=priorita  cíl           kategorie.!=priorita  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` nebo `/var/log/syslog`, 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

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 události 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 popsáním 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.

6.5.3 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,

- kategorií (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 přídatné nástroje. Například rotaci logů (včetně označování či odstraňování starších záznamů) zvládne *logrotate*.

Existují také propracovanější verze *syslogu*, 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 šířitelný 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 peru, je to velmi pružný nástroj využívající regulární výrazy.

6.6 Firewall

 V linuxových jádrech 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.

6.6.1 Ovládání firewallu

Jedná se o modul jádra, ke kterému se nepřistupuje přímo, ale přes některý obslužný program. Už mnoho let je základním obslužným programem *iptables* (příkaz `iptables`), místo něj se v posledních letech v mnoha distribucích objevuje program *nftables* (příkaz `nft`). Tyto programy už můžeme použít k ovlivňování pravidel firewallu.

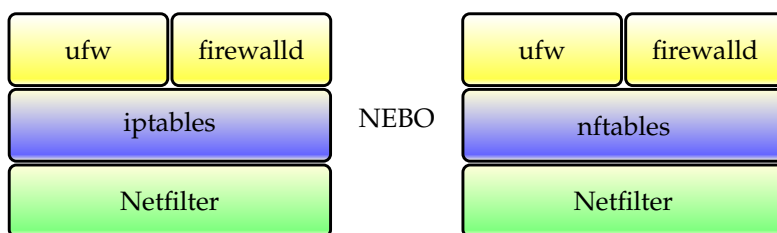
Kromě *iptables* a *nftables* však máme k dispozici jejich front-endy, tedy obslužné programy vyšší úrovně, které svůj výstup předávají programům *iptables* a *nftables* (jednomu z nich), což nám dává hierarchickou „ovlivňovací“ strukturu naznačenou na obrázku 6.1. Můžeme se setkat s top-level programy *ufw* (příkaz `ufw`) nebo *firewalld* (příkaz `firewall-cmd`).

¹*NTSyslog* najdeme na <http://ntsyslog.sourceforge.net>.

²*logwatch* získáme na <http://www.logwatch.org>.

³Program *swatch* (Simple Watch) je dostupný na <http://swatch.sourceforge.net>.

⁴Stavové 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.



Obrázek 6.1: Ovládání firewallu

V některých linuxových distribucích může být člověk poněkud zmaten – sice existuje příkaz `iptables`, ale pokud použijeme některý top-level program a myslíme si, že takto vytvoříme `iptables`-pravidlo, s překvapením zjistíme, že tabulky zobrazené přes `iptables` jsou prázdné. Protože pravidlo bylo přidáno do `nftables`-tabulky.

V následující sekci si vysvětlíme princip firewallu Netfilter (ten je stejný, ať používáme kterýkoliv z výše jmenovaných programů), ovládání si ukážeme na `iptables`. Top-level programy jsou jednodušší na používání (alespoň co se týče jednodušších úloh), jejich použití se dá snadno dohledat na internetu a samozřejmě v manuálových a info stránkách.

6.6.2 Struktura firewallu

 Základem je *tabulka* (table), přes tabulky prochází několik řetězců *chain* (čti: [čejn]), které již obsahují pravidla uplatňovaná na pakety (tato pravidla jsou zřetěžená a na paket se uplatňují postupně, dokud se nenajde „pasující“ pravidlo, proto se tomu říká chain).

Tabulka obvykle určuje, co konkrétně se má v dané části řetězce dít (například jen filtrování, nebo překlad adres či změna některých dalších údajů v záhlaví procházejícího paketu), chain obvykle určuje, pro jaký typ paketů se to má dít (například pro příchozí pakety, odchozí, průchozí u routeru, připravené na směrování, zpracované směrováním, apod.).

Příklad

Pokud jsme na routeru se zapnutým přeposíláním (forwarding), je příchozí paket nejdříve zařazen do chainu *PREROUTING* (tj. pro činnosti, které se mají provést ještě před směrováním), v něm projde určenými tabulkami (například tabulkou *nat*, pokud se provádí překlad adres).

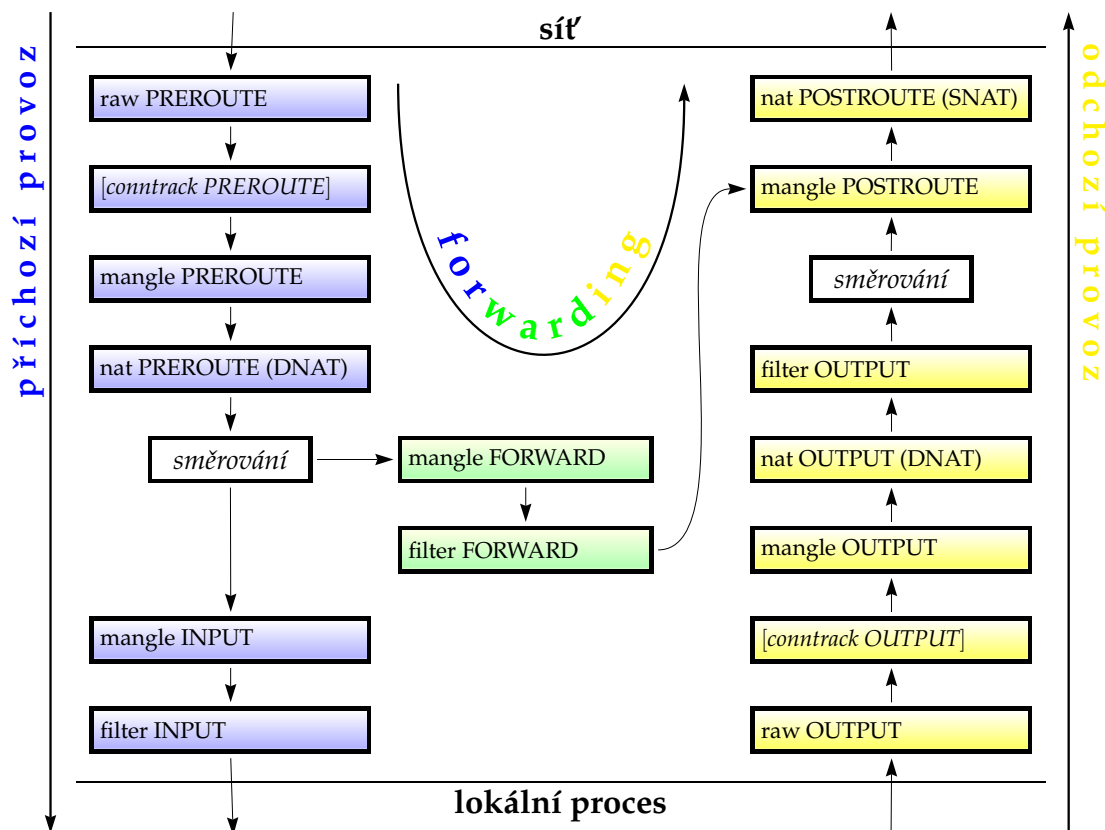
Pak proběhne směrování, které určí, jestli se tento paket má přeposílat (tedy je přeražen do chainu *FORWARD* pro přeposílané pakety) nebo je určen přímo pro toto zařízení (půjde do chainu *INPUT* pro příchozí pakety), a v rámci daného chainu projde stanovenými tabulkami. Přeposílaný paket pak ještě projde chainem *OUTPUT*, protože se stává odchozím paketem.



Takže chain vlastně určuje „kategorii“ paketu, tabulky pravidla pro zacházení. Pokud přes tuto tabulku procházejí dva pakety takové, že každý je v jiném chainu, bude se s každým zacházet trochu jinak, třebaže typ prováděných operací bude stejný (filtrování je ve více chainech, ale jinak se filtrují příchozí, jinak průchozí a jinak odchozí pakety).

 Standardně existují tyto tabulky (v příkazu `iptables` se před název tabulky dá přepínač `-t`):

- *filter* je výchozí, umožňuje základní filtrování paketů hlavně podle adres a některých dalších kritérií, do záhlaví nezasahuje, jen některé pakety zahodí,



Obrázek 6.2: Dráha paketu mezi výchozími tabulkami a chainy v Netfilteru

- *mangle* obsahuje pravidla pro transformace – upravujeme záhlaví, hodnoty TTL, pole TOS/QoS, apod.,
- *nat* pro mechanismus NAT (překlad adres, takže taky zasahuje do záhlaví),
- *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),
- *security* je přidávána bezpečnostními moduly jádra jako je například SELinux.

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, modifikace se provede 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.
- Tabulka *security* může následovat po tabulce *filter*, obsahuje chainy *INPUT*, *OUTPUT* a *FORWARD* a dokáže podle potřeby pakety označovat a přesměrovávat k jiným modulům jádra.

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 6.2 je znázorněna dráha paketu mezi jednotlivými tabulkami a chainy. Všimněte si, že přeposílané (forwarded) pakety nedorazí do žádného chainu typu INPUT ani OUTPUT.

 Program (ve skutečnosti služba, démon) *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).

V různých distribucích se však můžeme setkat i s dalšími obslužnými programy, například *uwf* (Uncomplicated FireWall) s grafickým rozhraním *gufw*, *shorewall*, *sanewall*, *apf-firewall* a další. Často jde spíše o nástavby *iptables*.



Další informace:

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/sect-Security_Guide-IPTables.html



6.6.3 Základní vnitřní příkazy a parametry pro filtrování

Jak bylo výše řečeno, pro konfiguraci firewallu slouží program *iptables*. 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.

Základem je možnost vypsat si existující pravidla, k tomu slouží parametr *-S* (velké „S“), všechna pravidla ze všech tabulek a chainů vypíšeme

```
sudo iptables -S
```

V parametrech pak můžeme určit jen konkrétní tabulku nebo chain (najdeme na manuálové stránce).

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** zahod' 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 („upovídáném“) 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 určení pravidla

`-I` nebo `--insert` připojí nové pravidlo na začátek chainu (pokud ne zadáme 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), na rozdíl 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 daného 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 zablokuje 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 PRE-ROUTING, 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 směřován (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 směřující 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

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    takto jsme povolili 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`.



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.



6.7 Bezpečnost systému a další nástroje

 **SELinux a Bastille Linux.** 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.

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 týká 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 přihlášení s dostatečnými právy. Projekt je dostupný zatím pro některé nejznámější distribuce a jejich klony na stránkách <http://freshmeat.net>.

 **Antiviry.** 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 scanner, 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.

 **Další nástroje.** Výborný nástroj pro správu systému je *Webmin* (<http://www.webmin.com>), pro jeho provoz stačí prakticky jen internetový prohlížeč (třeba Firefox) a samozřejmě nainstalovaný

a spuštěný démon 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 různé servery (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 v jiných předmětech.



Další informace:

Příručka správce zabezpečení Linuxu je na <http://www.securityportal.com/lasg>.

