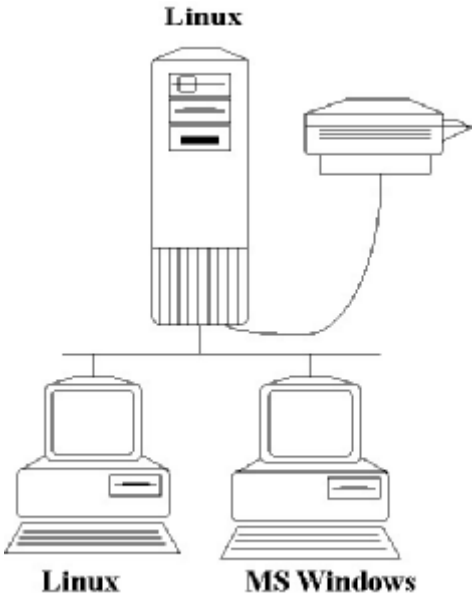
Linux prakticky ako server/ 1.časť

Aj vás - serveristov- vítam u prvej časti nového seriálu. Prečo vznikol tento seriál, to si prečítajte na inom mieste tohoto časopisu. Verím, že vás toto delenie potešilo, a tak aj my (serveristi) máme konečne dôstojné miesto na riešenie našich problémov.
Linux ako server je pomerne zložitá technológia. Jej výhodou je určitá modularita, keď sa nemusí nastaviť celý server naraz, ale postupne. Môžeme povedať, že Linuxový server je skupina rôznych dielčích serverov, bežiacich naraz na jednom počítači. My túto výhodu samozrejme využijeme. Začneme tými jednoduchšími časťami, niečo sa na nich naučíme, potom pridáme ďalšie a ďalšie súčasti. V našich cvičných podmienkach si vyskúšame (skoro) všetko, čo je v Linuxe z pohľadu servera dostupné. Je zrejmé, že v skutočnej praxi využijeme len niečo, čo konkrétne potrebuje tá-ktorá sieť, škola či firma. Ako odmenu však získame stabilitu, o ktorej sa správcom iných sietí ani nesníva!
Na rozdiel od desktopovej časti by bolo vhodné, keby sme si občas prečítali aj seriál Linux teoreticky. Objasní to niektoré následné súvislosti.

Čo budeme potrebovať
Používateľom - "desktopistom" postačuje pre výuku, ale aj samotnú činnosť jeden štandardný počítač, prípadne ešte modem a tlačiareň.

Server je však stavaný na prácu v sieti. Okrem toho, čo potrebuje bežný desktop, základnou podmienkou je sieť. Predpokladám, že málokto má možnosť cvičiť na funkčnej, "ostrej" sieti v práci! (A ani ja to tak nerobím. Najprv si doma niečo odskúšam, keď som si istý "ako na to", presuniem to na ozajstnú podnikovú sieť!)
Preto si musíme postaviť doma vlastnú cvičnú sieť.
Stačí minimálna, skladajúca sa z dvoch počítačov. Ideálnym modelovým riešením je však sieť o troch počítačoch, kde jeden bude linuxový server a tie dve budú klientske pracovné stanice - jedna s operačným systémom MS Windows a druhá s Linuxom. Nemusia to byť žiadne zázračné delá, stačia bežné dostupné počítače, ktoré sa nám alebo našim kamarátom doma povaľujú - tak od 486 vyššie s pár megami RAM, priemernou grafikou a harddiskom (aby na tom behali Win95/98).
Server by mal byť už celkom dobrý stroj - také bežné Pentium či AMD s rýchlosťou okolo 200 MHz a vyššie. U servera je tiež dôležitá veľkosť harddisku, tá by mala byť minimálne 2,5 GB a viac. Najdôležitejšou súčasťou servera je operačná pamäť - 64 MB RAM je naozaj minimum! Možno si poviete, že je to všetko málo, ale hovorím o modelovej sieti, na ktorej sa budeme učiť! Nechceme sa predsa finančne zruinovať! (Kto má na viac, tým lepšie!)
A u všetkých troch počítačov je dôležitá ešte jedna komponenta - tou je sieťová karta s príslušnými kábelmi a inými sieťovými prvkami. Vzor takej modelovej siete je na obr.č.1-1:



Kde na to vziať?
Ako hovorím, treba sa pozrieť, či sa niekde niečo staršieho doma neváľa. Keď nie u nás, tak u kamarátov. Občas (a nielen počítačové) firmy robia výmenu techniky a tú starú odpredávajú za bagateľ. Takto, alebo na inzerát sa dá zakúpiť zostava s Pentiom či AMD za jednu -dve tisícky aj s monitorom. Ale netreba kupovať ku každému počítaču monitor. Po prvé, je drahý a po druhé, môžeme ho pomocou prepínača (ktorý sa dá bežne kúpiť) zdieľať až medzi 4 počítačmi. Tieto prepínače okrem monitoru prepínajú aj klávesnice a prípadne aj myši.
Na serveri nemusí byť ani myš a postačuje tá najjednoduchšia grafika. Maximum práce v linuxovom serveri budeme vykonávať z konzoly, teda príkazového riadku. Nie že by to nešlo "oknoidne", ono sa to tiež tak akosi dá, ale ručná práca je ručná práca!
Ak sme trochu hardvérovo zruční (a to spravidla správcovia siete musia byť), môžeme si také počítače poskladať aj sami zo starších dielov. Sám mám doma sieť o 5 (piatich) počítačoch, kde za niektoré som nedal ani groš (no, možno pár flaštičiek dobrého vínka). Jeden z nich je môj "pracovný" stroj, na ktorom tvorím aj tento článok.

Poznámka:
Niektoré témy zo serverovej oblasti sa dajú vyskúšať aj na jednom jedinom počítači - na samotnom serveri. Ale je to také ako sex - vo dvoch je to predsa len lepšie!

Po softvérovej stránke budeme na serveri a tej jednej klientskej stanici používať linuxovú distribúciu RedHat (7.3). Ak máte inú distribúciu, ako je Mandrake alebo SuSE, nebojte sa, tieto distrá vyšli z RedHatu a v drvivej väčšine príkazy, cesty k súborom a názvy súborov sú totožné. Na prípadné rozdiely sa budem snažiť včas poukázať.

Ako vytvoriť sieť
Predpokladám, že máte základné skúsenosti s tvorbou siete, hlavne čo sa týka hardvérovej stránky, teda aké karty existujú, aké kábely, konektory a prípadne ďalšie technické prostriedky, ako sú huby (čítaj haby, nemyslím tie jedlé alebo nejedlé čudá, čo vyrastajú na poliach a v lese).
Na doma nám plne postačuje sieť postavená na koaxiálnom kábeli, ktorá je asi najlacnejšia.
Z pohľadu softvéru nepotrebujeme žiadne znalosti, všetko si vysvetlíme.

Okrem "železa" budeme potrebovať ešte základné vedomosti o Linuxe v rozsahu seriálu *Začíname s Linuxom*, niektoré znalosti z operačného systému MS Windows, aspoň minimálne základy teórie sietí a ... trpezlivosť.

Čo by sme sa mali naučiť
Cieľom tohto seriálu je naučiť sa **vybudovať** a **spravovať** linuxový server v heterogénnej počítačovej sieti.
Úlohou servera bude poskytovať svoje služby a prostriedky klientským počítačom, nezáležiac na ich operačnom systéme.
To značí, že na jeden a ten istý server budú pristupovať klientské stanice ako s operačným systémom MS Windows, tak aj so systémom Linux.

Aké služby očakávame od nášho servera?
V prvom rade musí overovať totožnosť klientských staníc (klientov) a používateľov, pracujúcich s nimi. Na základe autentifikácie im stanoví, čo môžu a čo nemôžu.
Ďalej bude používateľom poskytovať diskový priestor výhradne iba pre nich, ako aj spoločný diskový priestor, do ktorého majú prístup všetci ostatní.
Chceme, aby zdieľal nielen priestor, ale aj dáta a prípadne aplikácie.
Svetu vládnu informácie. Naučíme sa inštalovať a spravovať databázový server a príslušné aplikácie.
Budeme požadovať, aby vykonával automatické zálohovanie dát a aplikácií v uvedených adresároch, a to na napalovačku, alebo ešte lepšie, na prepisovačku.
Dnes je samozrejmosťou tlač. Ukážeme si, ako zdieľať tlačiareň, pripojenú k serveru, a to z windowsovských aj linuxových klientov.
Ak máme nejakú možnosť pripojenia do Internetu (či už modemom alebo pevným pripojením), mal by slúžiť ako jeho súčasť, teda poskytovať poštové a iné internetové služby.
Naučíme sa vytvoriť www server a používať na ňom HTML stránky, či len pre našu lokálnu sieť, alebo - ak máme prístup do Internetu - tak aj ostatnému svetu.
Pripojíme ho na inteligentnú UPS-ku, aby sme minimalizovali škody pri výpadku elektrickej energie.
Máte radi, keď vás váš serverík pravidelne informuje o svojom stave, prípadne okamžite hlási rôzne problémy, výpadky, poruchy či pokusy o napadnutie po sieti? Donutíme ho, aby nám teda poslal es-ém-esku alebo inak vyhlásil poplach!

Ukážeme si, ako sa pripraviť na prechod k čisto Linuxovej sieti - teda ako server, tak pracovné stanice sú na linuxovej báze. Pracovné stanice budú pristupovať technológiou "tenkých klientov", čo značí, že na strane pracovných staníc postačuje minimálny, ale naozaj minimálny hardvér. Tenký klient, bežiaci na 486 beží celkom svižne, až máme pocit, že pracujeme s Pentiom. Môžeme mať spustený OpenOffice, webovský prehliadač a pracovať s grafikou súčasne, a pritom naša pracovná stanica nemusí mať ani harddisk. Že to funguje, to je overené a poznám niekoľko firiem a dokonca úradov, kde týmto spôsobom značne ušetrili peniaze za obnovu techniky a samozrejme za softvér!
Ak to bude čo len trochu možné (to záleží, či mi jedna firma poskytne sľúbené zariadenia!), ukážeme si spojenie pomocou bezdrátového mikrovlného spoja - tzv. Wi-Fi. Je to veľká novinka, v poslednej dobe veľmi obľúbená pri vytváraní privátnych alebo komerčných sietí, a to aj na väčšie vzdialenosti.
Nemenej dôležitou činnostou správcu servera je aj pravidelné "upgradeovanie" systému a príslušných programov.
Na záver sa budeme venovať toľko žiadanej bezpečnosti, a ukážeme si aj pár hackerských trikov, aby sme zistili, či nemáme niekde v systéme pootvorené dvierka.
Možno sa to zdá nezasvätenému málo. Ten, kto už má nejaké skúsenosti s inou ne-linuxovou sieťou, mi dá za pravdu, že za dosiahnutým cieľom sa skrýva značné úsilie, hodiny a hodiny štúdia, skúšania, nastavovania a vylaďovania.
Ale stojí to za to!

Linux prakticky ako server/ 2.časť

V minulej časti sme si povedali, čo budeme očakávať od serveru. Ale server, to je iba stroj. A ten spravuje nejaký človek, osoba, muž či žena (áno, áno aj žena!), ktorej sa hovorí aj ***správca***.
Správca má rôzne ďalšie mená. Spravidla to býva správca servera, správca systému, superpoužívateľ, supervisor, administrátor, admin alebo root.
Zatiaľ čo správca, admin a administrátor sú synonymá (teda slová rovnakého významu) a titulujú osobu, ktorá zodpovedá za systém alebo sieť v širšom meradle, vrátane servera, klientských staníc, tlačiarní, kabeláže a chladničky s kávovarom, root alebo superpoužívateľ značí používateľa s najvyššími právami v operačnom systéme Linux. Tá istá osoba, nech sa volá *oravmir*, môže byť naraz správcom celej siete (systému), rootom, ale aj bežným používateľom Linuxu s prihlasovacím menom *oravmir*.
Pasujme teda sami seba do všetkých troch pozícií v našom cvičnom informačnom systéme.

Správca systému
Správca systému je tá osoba, ktorú nikto a nikdy nedocení! Jej úlohou je starať sa o celý informačný systém, aby bol vždy a riadne funkčný a v plnej výkonnosti, a to tak, aby to čo najmenej alebo vôbec nezaťažovalo používateľov siete. Takým dobrým ukazovateľom, či sme správnym správcom je fakt, ako často sa na nás obracajú práve používatelia. Čím menej nás otravujú, tým viac to značí, že je sieť stabilnejšia a tým sme lepšími správcami.
Bohužiaľ, má to aj druhú stranu mince – keď o nás používatelia nevedia, považujú nás za nepotrebných, prebytočných ľudí, ktorí im ukrajujú z finančných prostriedkov a ktorých výplatu by si mohli rozdeliť medzi sebou.
Pracovná vyťaženosť správcu je však menej ružová. Dobrý správca sa o systém stará! Pravidelne kontroluje chod zariadení, vykonáva údržbu, zálohovanie, ale väčšinu času odstraňuje závady.
Závady? Aké závady? Veď keď je systém raz dobre nastavený, tak musí fungovať tip-top, nie?
Omyl! Náš systém je živý organizmus! Veď ho používajú (živí) ľudia a práve oni sú zdrojom rôznych porúch a problémov.
Ale dobrý správca všetko zvládne!
Ako?
O tom si teraz niečo povieme.

Správa systému
Čo je teda presne správa systému?
Pretože výpočtovým – informačným systémom môže byť všeličo od jedného po stovky počítačov, a pretože každý počítač v prostredí siete obsahuje niečo iné, tak aj tento pojem je veľmi rozsiahly a nie je možné do pár slov zahrnúť všetky činnosti, obsahujúce správu systému.
Preto len veľmi obecne:
Správa systému sú všetky úlohy správcu systému, ktoré zabezpečujú plynulý a bezporuchový chod systému s čo možno s najväčšou výkonnosťou.
Tieto úlohy správca plní pomocou rôznych *prostriedkov*.

1. Úlohy správcu systému
Pokúsme sa teraz vytvoriť akýsi zoznam úloh, ktoré každý správca systému vykonáva. Tento zoznam samozrejme nebude úplný, pretože to vždy záleží od konkrétnej situácie.

Môžeme povedať, že správca systému vykonáva tieto úlohy:
- konfigurácia a údržba hardvérových prostriedkov
- inštalácia a opravy operačného systému
- inštalácia a údržba aplikačného softvéru
- ochrana a zabezpečenie systému
- vytváranie a mazanie používateľov a skupín v systéme
- aktualizácia aplikácií
- archivovanie dát a obnova prevádzky
- ladenie výkonu
- plánovanie modernizácie systému

Konfigurácia a údržba hardvérových prostriedkov
Každý správca systému je tak trochu aj hardvérista. Musí byť schopný pridať do servera nový disk, vložiť novú sieťovú kartu alebo vybudovať novú sieťovú prípojku pre klientskú stanicu. Je síce možné o to požiadať niektorého z technikov, ale hardvérové schopnosti patria k imidžu administrátora.

Inštalácia a opravy operačného systému
Inštalácia a opravy operačného systému patria výlučne do právomoci správcu alebo roota. To sa netýka iba servera, ale aj klientských staníc. Niekedy sa stáva, že vplyvom objektívnych udalostí dôjde k poškodeniu operačného systému. Medzi tieto objektívne udalosti patrí poškodenie hardvérovej súčiastky servera, klientskej stanice alebo časti siete vplyvom tepla, výrobných vád, atmosférických porúch, ale aj násilným poškodením. Najčastejšou závadou je však neodborná manipulácia zo strany používateľa, ktorý „všade bol, všetko videl a všetko vie“.

Inštalácia a údržba aplikačného softvéru
Boli by sme ako správcovia veľmi neradi, keby si používatelia sami inštalovali aplikačný softvér podľa svojej ľubovôle. Už len z toho dôvodu, že chceme mať centrálnu správu aplikácií, vrátane prideľovania prístupových práv a zdieľania dát. Preto táto úloha patrí správcovi.

Ochrana a zabezpečenie systému
Bezpečnosť systému je asi najobtiažnejšia úloha správcu. Je to oblasť, ktorá spôsobuje najviac problémov. V dnešnej dobe čakajú tisíce crackerov a hackerov, ktorí majú veľkú záľubu v dolovaní dát a v spôsobovaní kolapsu systému. Ochrana zabezpečuje celú škálu opatrení od pravidelnej zmeny hesiel až po kamerové systémy a mreže.

Vytváranie a mazanie používateľov a skupín v systéme
Veľmi často sa stáva, že nám do systému vstupujú noví používatelia. Im musíme stanoviť, čo môžu a nemôžu, prístup k dátam, aplikáciám alebo k pošte. Veľmi často sa stáva, že po odchode bývalého používateľa z organizácie pozabudneme vymazať jeho účet, čím sa tento stáva potencionálnou dierou do systému.

Aktualizácia aplikácií
Celá počítačová sieť vrátane servera a klientských staníc vyžadujú príležitostnú aktualizáciu aplikačného programového vybavenia. Tú môžu vyžadovať samotní používatelia alebo aj vedenie organizácie.

Archivovanie dát a obnova prevádzky
Mimoriadne dôležitou úlohou je vytváranie a ukladanie kópií operačného systému, aplikačných dát, konfiguračných súborov, ale aj domovských adresárov jednotlivých používateľov. Dnes, vďaka cenovo dostupným technickým zariadeniam a cenám médií nie je problém vykonávať pravidelnú archiváciu celých systémov. Neexistujú horšie chvíle v živote správcu ako „kľaknutie“ systému s kompletnou stratou dát, ktorý nemá po ruke archívne médiá s aplikačnými dátami. Takáto chyba môže spôsobiť aj znefunkčnenie firmy alebo organizácie na dlhú dobu, prípadne aj krach.

Ladenie výkonu
Žiadny systém nikdy nie je taký rýchly, aby uspokojil požiadavky používateľov. Preto musí správca pristupovať k optimalizácii systému, ktorej hovoríme aj ladenie. Ladenie zahŕňa súbor činností od efektívnej správy operačnej pamäte až po rýchlosť výmeny vadnej súčiastky.

Plánovanie modernizácie systému
Informačné technológie sa na rozdiel od iných technológií vyvíjajú veľmi rýchlo. Každého pol roka je na trhu nový hardvér alebo nová verzia príslušného softvéru. Preto správca musí popri svojej činnosti sledovať novinky v týchto oblastiach a držať s nimi krok.

2. Prostriedky správcu systému

Každý správca má na plnenie svojich úloh príslušné prostriedky. Tie môžeme deliť na *technické* a *programové*. Medzi technické prostriedky patrí okrem klasického skrutkovača aj merací prístroj. Tým sme schopní zmerať základné parametre siete, a to siete elektrickej alebo počítačovej. My si neskôr ukážeme, ako pomocou najjednoduchšieho meracieho prístroja (alebo aspoň baterky a žiarovky) identifikovať najbežnejšie poruchy kabeláže počítačovej siete (pre tých zručnejších „fajnšmekrov“ mám pripravený jednoduchý, ale účinný tester štrukturovanej kabeláže – pár lediek, odporov a integrovaný obvod – PIC od Microchipu).

Viac ako technické prostriedky však každý správca použije prostriedky programové. Tie sa delia na:
- konfiguračné súbory
- príkazy operačného systému
- skripty
- špecializované utility a nástroje
- logovacie súbory
- protokoly

Konfiguračné súbory
Na rozdiel od operačného systému MS Windows, v Linuxe sa každý program konfiguruje pomocou konfiguračných súborov. Tieto konfiguračné súbory sú textové súbory, preto ich môžeme editovať v nám najvyhovujúcejšom editore. Ja obľubujem editor, ktorý je súčasťou *Midnight Commandera* (zlozvyk či nostalgia z *Norton Commandera*) pod klávesom F4.

Poznámka:
*Správny linuxový guru používa zásadne len a len **vi** editor. Pre mňa je však pomerne zložitý a tak som stále iba čarodej II.kategórie.*

Konfiguračné súbory sa v Linuxe spravidla nachádzajú v adresári */etc/meno_súboru* alebo */etc/názov_programu/meno_súboru*, v závislosti od distribúcie a verzie Linuxu.
Každý program alebo príkaz v Linuxe pri svojom spustení prečíta svoj konfiguračný súbor alebo súbory, aby načítal jeho parametre, podľa ktorých sa bude riadiť. Preto pri zmene parametrov konfiguračných súborov musíme zabezpečiť reštart programu alebo príkazu, aby došlo k znovunačítaniu parametrov.
Na rozdiel od MS Windows v Linuxe nie je potrebné po každej zmene konfiguračného súboru vykonávať reštart celého operačného systému!

Príkazy operačného systému
Existujú stovky príkazov, ktoré správca používa k spravovaniu serveru. Môžu byť vo forme binárnych, teda skompilovaných príkazov jazyka C alebo vo forme príkazov interpretera *shellu* alebo iného programovacieho interpretačného jazyka ako je *Perl* alebo *Python* .
Použitie akéhokoľvek typu príkazu vo väčšine prípadov predstavuje zadanie príkazu spolu s príslušnými parametrami napr. *mount –t msdos /dev/fd0 /mnt/floppy*. Najčastejšie požívané príkazy správcu si v našom seriáli vysvetlíme.

Skripty
Veľmi často sa stáva, že denne používame určitú skupinu funkcií a príkazov interpretera k dosiahnutiu požadovaného výsledku. Aby sme tieto nemuseli zadávať pracne každý deň, vytvoríme si *skript*. Skript je v podstate textový súbor, ktorý obsahuje požadované príkazy a funkcie. Skript v Linuxe sa veľmi podobá dávkovému súboru .BAT v MS DOS/Windows. My si niekoľko funkčných skriptov vytvoríme a budeme ich samozrejme využívať.

Špecializované utility a nástroje
V poslednej dobe sa v Linuxe rozšírili špecializované utility a nástroje, ktoré v sebe zahrňujú nástroje na komplexnú správu servera alebo systému. Tieto nástroje sa vyznačujú jednotným vzhľadom a uceleným prostredím v textovej alebo grafickej podobe. Najznámejším nástrojom v Red Hat Linuxe je *linuxconf* (ktorý bol z nepochopiteľných príčin od verzie 7.x vypustený).
Druhým výborným nástrojom je *webmin*, ktorý má vďaka http protokolu grafický vzhľad a ktorý si preberieme.

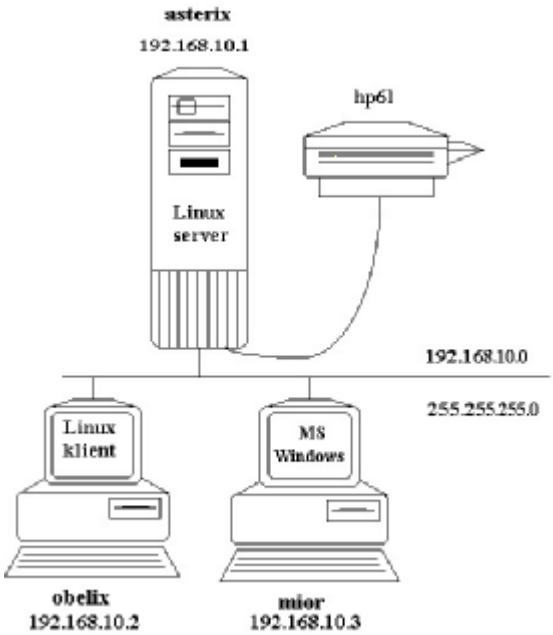
Logovacie súbory
Mnoho programov a projektov, vrátane jadra systému (kernelu) vytvárajú o svojej činnosti, prípadne chybách záznam do zvláštneho súboru. Tomuto súboru hovoríme *logovací súbor*, skrátene *log*. Aj keď sa nám to teraz nezdá, budeme sa vo svojej praxi na tieto logy veľmi často obracať. Logovací súbor kernelu je dokonca základným kameňom systémovej bezpečnosti.

Protokoly
Je veľmi vhodné si o každom počítači alebo serveri založiť akýsi denník, do ktorého budeme zapisovať nielen zmeny hardvérovej, ale aj softvérovej konfigurácie. Sem zapisujeme aj rôzne postupy riešenia konkrétnej úlohy a poznámky. Tomuto denníku sa hovorieva *protokol*. Nepodceňujme tieto protokoly, pretože ľudská pamäť nie

je nekonečná a zvlášť v prostredí, kde môže dôjsť k zmenám na pozícii správcu, sú tieto protokoly neoceniteľné pre prípadného nástupcu.

Dosť bolo úvodných teoretických rečí, prejdime k praxi.

Staviame cvičnú sieť
Aby sme sa mohli stať správcami siete, musíme nejakú sieť mať. Preto si teraz jednu – cvičnú - sieť postavíme. Na obrázku č.1 je schéma našej siete aj so základnými údajmi:



Každý počítač v sieti býva reprezentovaný svojim menom. Nech sa náš linuxový server volá ***asterix***, linuxový klient ***obelix*** a pracovná stanica na báze MS Windows prezmenu ***mior*** (môžete si dať aj *vilko*, ale podobnosť s krstným menom jedného softvérového magnáta by asi nebola náhodná -;))).
Každá sieť a počítač v nej v un*xovom prostredí sú prezentované číslom. My už vieme, že to číslo je číslo IP adresy a masky a na cvičné účely existuje určitý rozsah voľných sietí. Nech má naša sieť IP adresu 192.168.10.0 a masku 255.255.255.0 a počítače budú v rozsahu 192.168.10.1 až 192.168.10.254.
V zmysle určitých zvyklostí, ktoré sme si vysvetlili v seriáli *Začíname s Linuxom*, sa servery číslujú od začiatku siete, aktívne prvky siete (huby, prepínače a pod., ktoré tu teraz nemáme) od konca siete a ostatné počítače po poriadku. Náš server *asterix* má teda adresu IP 192.168.10.1, druhý Linux *obelix* IP 192.168.10.2 a „okienkový“ mior má IP 192.168.10.3. Masku majú totožnú a tou je číslo 255.255.255.0.
A vieme ešte, že *broadcast* adresa bude posledné číslo v sieti a to je 192.168.10.255.
Poslednou položkou našej siete je laserová tlačiareň, ktorú budeme v sieti poznať pod menom ***hp6l***. Keďže je pripojená k paralelnému portu servera, nemá žiadnu IP adresu.

A vieme ešte jednu vec – my sa nemusíme v sieti odvolávať iba na IP adresy. Môžeme komunikovať s iným počítačom pomocou jeho mena. To môžeme zabezpečiť dvoma spôsobmi:
- využitím služby DNS (Domain Name Service)
- pomocou súboru *hosts*

V prípade, že máme malú sieť a tá nie je pripojená do ďalšej siete, DNS nepotrebujeme. Samozrejme, že sa tento systém naučíme, ale ešte nemáme tie správne znalosti. Preto zatiaľ využijeme druhú možnosť.
Súbor *hosts* sa v Linuxe nachádza v adresári */etc*.
Ako na serveri (asterix), tak na klientovi (obelix) vytvoríme príslušný súbor. Ten obsahuje dva stĺpce. V prvom je IP adresa a v druhom je meno počítača.

Poznámka:
Toto je príklad mojej cvičnej siete. Vy si môžete prideliť vlastné mená. Pokiaľ možno však dodržte čísla IP adries, pretože sa v mojich výpisoch budem na ne odvolávať.

Prejdime teraz k jednotlivým počítačom:

Server
Ako sme si povedali, za server by sme mali vybrať čo najsilnejší počítač (samozrejme podľa našich možností). Predpokladám, že už máme skúsenosti s inštaláciou Linuxu, takže iba pár poznámok či krokov:

A) Ak máme dostatok miesta, nainštalujeme celú distribúciu. Vyhneme sa tým nedostupnosti niektorého balíku, ktorý by sme mohli prípadne potrebovať. Ak nemáme dostatok miesta, použijeme voľbu „server". Tá by mala zabezpečiť najpodstatnejšie súbory na činnosť serveru siete, ostatné si podľa potreby doinštalujeme neskôr.
B) Nastavme systém tak, aby server bootoval do textového režimu a nie grafického. Ak sme náhodou pri voľbe inštalácie omylom nadefinovali bootovanie do grafického režimu, toto odstránime takto:

- po nabootovaní do grafického režimu (ľudovo povedané do X-ov) sa prihlásime ako root
- spustíme ľubovoľný súborový mamžér, môžeme aj *Midnight Commander* (pomocou xterm-u) alebo pomocou kláves *Ctrl-Alt-F1* prejdeme do prvej textovej konzoly, nalogujeme sa ako root a spustíme *mc*
- prejdeme do adresára */etc* a vyeditujeme súbor *inittab* – výpis č. 2:

```
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Author:       Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#               Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:

# System initialization.
```

- vyhľadáme sekvenciu znakov začínajúcu na *id*, napr.: *id:5:initdefault:*
- číslo *5* nahradíme za číslo *3* (platí pre Red Hat, v iných systémoch si treba prečítať poznámky v súbore *inittab*)
- súbor uložíme
- reštartujeme systém

Po novom spustení systém automaticky nabootoje do textového režimu.

C) Vytvoríme na serveri niekoľko nových používateľov (pomocou useradd a passwd)
D) Nastavíme sieťové parametre (pozri seriál *Začíname s Linuxom*)
E) Vytvoríme súbor */etc/hosts* takto – výpis.č.3:

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1         asterix localhost.localdomain localhost
192.168.10.1      asterix
192.168.10.2      obelix
192.168.10.3      mior
```

Linuxový klient
Ako linuxový klient môžeme použiť aj menej výkonný počítač, to záleží na našich možnostiach. Pre začiatok by bolo dobré, aby mal harddisk, neskôr, až budeme mať dostatok skúseností a vedomostí, si ukážeme, ako použiť bezdiskovú stanicu. Inštalačné pokyny nech sú takéto:

A) Ak máme dostatok miesta na harddisku, nainštalujme celú distribúciu. Ak nemáme dostatok miesta, použijeme voľbu „desktop“ alebo „workstation“ (neskôr si ukážeme, ako podstatne zmenšiť celú inštaláciu)
B) Ak máme dostatočnú operačnú pamäť a aspoň trochu schopnú grafickú kartu, nastavíme systém tak, aby bootoval do grafického režimu. Ak nie, nastavíme boot do textovej konzoly
C) Pridáme do systému okrem roota aspoň dvoch bežných používateľov
D) Nastavíme sieťové parametre
E) Vytvoríme súbor */etc/hosts* takto – výpis č.4:

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1         obelix localhost.localdomain localhost
192.168.10.1      asterix
192.168.10.2      obelix
192.168.10.3      mior
```

Všimnime si, že sa výpisy líšia u položky 127.0.0.1, čo je spätná smyčka, tzv. *loopback*

Pracovná stanica s MS Windows
Tou nech je bežný počítač, na ktorom sú schopné bežať „okná“ W9x.
Urobme tieto nastavenia:

A) nastavme meno počítača a jeho IP adresu
B) povolíme zdieľanie súborov a ak máme pripojenú aj tlačiareň, tak aj tú
C) Vytvoríme súbor *c:\Windows\hosts* takto – výpis č.5 (môžeme použiť vzorový *hosts.sam*):

```
# Copyright (c) 1998 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP stack for Windows98
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

127.0.0.1         localhost
192.168.10.1      asterix
192.168.10.2      obelix
192.168.10.3      mior
```

Skúška spojenia
Aby sme sa presvedčili, či sú naše spojenia funkčné, musíme urobiť skúšku. Znova sa odvolávajúc na seriál *Začíname s Linuxom* vieme, že najvhodnejším nástrojom je príkaz **ping**. Div sa svete, pozná ho aj operačný systém MS Windows!

Prejdime teraz za konzolu servera – asterixu a zadajme príkaz:

[root@asterix root]# ping 192.168.10.2

a ak dostaneme výpis podobný č.6, vieme, že spojenie z asterixu na obelix funguje:

```
PING 192.168.10.2 (192.168.10.2) from 192.168.10.1 : 56(84) bytes of data.
64 bytes from 192.168.10.2: icmp_seq=1 ttl=128 time=0.373 ms
64 bytes from 192.168.10.2: icmp_seq=2 ttl=128 time=0.371 ms
64 bytes from 192.168.10.2: icmp_seq=3 ttl=128 time=0.343 ms

--- 192.168.10.2 ping statistics ---
3 packets transmitted, 3 received, 0% loss, time 4997ms
rtt min/avg/max/mdev = 0.339/0.352/0.373/0.014 ms
```

Teraz vyskúšame, či funguje aj preklad mien počítačov na IP adresy.
Zadáme príkaz:

[root@asterix root]# ping obelix

Ak uvidíme výpis č.7, je všetko v poriadku:

```
PING obelix (192.168.10.2) from 192.168.10.1 : 56(84) bytes of data.
64 bytes from doma (192.168.10.2): icmp_seq=1 ttl=128 time=0.374 ms
64 bytes from doma (192.168.10.2): icmp_seq=2 ttl=128 time=0.357 ms
64 bytes from doma (192.168.10.2): icmp_seq=3 ttl=128 time=0.352 ms

--- doma ping statistics ---
3 packets transmitted, 3 received, 0% loss, time 4996ms
rtt min/avg/max/mdev = 0.346/0.355/0.374/0.017 ms
```

Takto vyskúšame spojenie na mior:

[root@asterix root]# ping 192.168.10.3
[root@asterix root]# ping mior

Ak dostaneme podobné obrazovky, je to v poriadku.

Obdobný spôsobom prejdeme za konzolu obelix-u a zadáme príkazy:

[root@asterix root]# ping 192.168.10.1
[root@asterix root]# ping asterix
[root@asterix root]# ping 192.168.10.3
[root@asterix root]# ping mior

a nakoniec prejdeme do MS Windows. V ponuke ***Štart – Spustiť*** zadáme ping a príslušný parameter. Či spojenie funguje, zistíme z výpisu č.8:

```
C:\WINDOWS>ping asterix

Testovanie dostupnosti asterix [192.168.10.1] s 32 bajtov údajov:

Odpoveď od 192.168.10.1: bajty=32 čas<10 ms TTL=255
Odpoveď od 192.168.10.1: bajty=32 čas<10 ms TTL=255
Odpoveď od 192.168.10.1: bajty=32 čas=1 ms TTL=255
Odpoveď od 192.168.10.1: bajty=32 čas<10 ms TTL=255

Štatistika testovania dostupnosti pre 192.168.10.1:

    Pakety: odoslané = 4, prijaté = 4, stratené = 0

        (strata:0%),

Približný čas odozvy v milisekundách:

    Minimum = 0ms, Maximum =  1ms, Priemer =  0ms
```

Ak sme dosiahli uspokojivé výsledky, máme sieť riadne nastavenú. To je prvý predpoklad, akýsi základný stavebný kameň nášho malého informačného systému.

Opakovanie
Na záver si dáme domácu úlohu. Zopakujme si:
- ako sa zapína a vypína operačný systém Linux
- ako sa pridávajú a mažú používatelia v systéme
- ako sa pridávajú a mažú skupiny používateľov v Linuxe a ako sa do konkrétnej skupiny pridávajú rôzni používatelia
- čo sú to prístupové práva a ako sa nastavujú
- ako sa v Linuxe narába so súbormi a adresármi
- adresárovú štruktúru v Linuxe
- možnosti a položky menu Midnight Commandera

Na všetky otázky nájdete odpoveď v seriáli *Začíname s Linuxom*.

Linux prakticky ako server/3.časť

Minule sme si postavili našu cvičnú sieť. Verím, že funguje dobre a dokážeme sa *pingnúť* z každého počítača na ostatné. Ak pingy prešli, máme istotu, že sieť je po hardvérovej, ale aj po softvérovej stránke priechodná. To je prazáklad úspechu pri budovaní informačného systému.
Vždy, keď zistíme nefunkčnosť určitej služby alebo procesu, ešte predtým, než všetko „rozhasíme“ pri hľadaní potencionálnej chyby, overme, či je sieť „živá“. A na to niet lepšieho prostriedku, ako je *ping*.
Dnes sa budeme venovať základom administrácie serveru. Niežeby sme už nemali nejaké základy, ale dnes sa na to pozrieme trochu pokročilejšie. To preto, aby sme vedeli, čo robíme, keď niečo robíme.

Pravidlo č.1 (najhlavnejšie)
Keď všetko zlyhá, otvoríme manuál. *V tomto prípade manuálové stránky, ktoré sú súčasťou operačného systému Linux.*
Prečo?
Po prvé, človek nemá sloniu pamäť. Nemôže si zapamätať všetko, obzvlášť nie rôzne parametre príkazu a ich formálny zápis.
Po druhé, my si tu povieme tie najčastejšie používané parametre toho - ktorého príkazu. Ale spravidla každý z nich má ešte niekoľko ďalších, menej používaných, ale za to cenných parametrov.
Len pre zopakovanie (veď sme už na linuxovej strednej!), manuálové stránky vyvoláme príkazom
man *meno_príkazu*.
Za meno_príkazu môžeme niekedy použiť aj meno súboru. Manuálové stránky sa stále dopracovávajú, preto skúšajme spýtať sa manuálových stránok na všeličo. Budeme prekvapení, koľkokrát nájdeme správnu odpoveď.

Jednou zo základných činností správcu servera je vytváranie, modifikácia alebo mazanie používateľov v systéme. Poďme si povedať niečo o používateľoch a ich skupinách:

Používatelia
Filozofia Linuxu je založená na tom, že každý program alebo súbor niekomu patrí. To znamená, že Linux nemôže nemať používateľov. Má teda aspoň jedného používateľa a tým je *root*. V Linuxe existuje niekoľko typov používateľov. Delíme ich na *bežných* a *špeciálnych* používateľov. Bežní používatelia sú tí, ktorých do systému zavedieme my - správcovia, teda napr. *anicka*, *jkovac*, *matejb*, *direktor*, *skladnik1* a podobne. Špeciálnych používateľov vytvorí Linux sám, buď pri inštalácii systému, alebo niektorého programu. Títo používatelia sú pevne zviazaní s činnosťou systému alebo programu, ktorý ich vytvoril. Príkladom takého špeciálneho používateľa je napr. *bin*, *adm*, *lp*, *sync*, *apache* a iní. Prečo existujú, to sa dozvieme neskôr. Zatiaľ nám stačí vedieť, že existujú.

Skupiny
Ako s používateľmi je to podobné aj so skupinami. Tiež existujú skupiny, ktoré vytvoril systém sám a tie, ktoré sme vytvorili my správcovia. V distribúcii RedHat sa pri vytvorení nového používateľa vytvorí zároveň skupina toho istého mena. Tejto skupine hovoríme implicitná skupina. V iných distribúciách môžeme nájsť jednu spoločnú skupinu s názvom *users*, ktorej členovia budú nami nadefinovaní používatelia. Ako správcovia však najčastejšie budeme definovať skupiny, ktoré budú zobrazovať reálnu podobu systému. Ak budeme používať sieť v niektorom výrobnom podniku, pravdepodobne vytvoríme skupiny *menezment*, *vyroba*, *sklad*, *ucto*, *expedicia*, *logistika* a podobne. Do skupín priraďujeme používateľov so spoločným pracovným zameraním. Je to logické, pretože ľudia z jedného oddelenia pracujú na spoločných úlohách a preto chceme, aby mali spoločné – skupinové práva.

Každý používateľ (bežný či špeciálny) je v Linuxe identifikovaný pomocou identifikačného čísla používateľa. Označuje sa **UID** (*User IDentificator*).
Takisto skupina má priradené identifikačné číslo skupiny – **GID** (*Group IDentificator*).
Pri prideľovaní UID a GID existujú v Linuxe určité pravidlá:
Administrátor *root* má UID a GID najnižšie - rovné nule. Špeciálni používatelia majú UID a GID v rozsahu 1 až 499. Bežní, správcom definovaní používatelia a skupiny majú UID a GID rovné 500 a vyššie.

My ako správcovia sa nemusíme trápiť pri definovaní nových používateľov a skupín do systému, aké že to máme prideliť UID a GID. To za nás zabezpečia pomocné nástroje, ako je ***useradd*** alebo ***groupadd***.
Na čísla UID a GID sú v Linuxe naviazané prístupové práva k jednotlivým súborom.

(Spomeňme si! V Linuxe sa **VŠETKO** javí ako súbor!!! Teda aj rôzne periférne zariadenia, ako je modem, céderomka, tlačiareň, ale aj príkaz či dáta).

Vytváranie používateľov
Každý používateľ v Linuxe je charakterizovaný týmito prvkami:
- prihlasovacím menom
- heslom
- priradenou skupinou
- domovským adresárom

Pri prihlasovaní do systému sa musí každý používateľ identifikovať prihlasovacím menom (*login*) a heslom (*password*). Na tieto dve položky sa ho operačný systém Linux spýta vždy, keď sa používateľ pokusí vstúpiť do systému.

Ako sme si povedali, každý používateľ by mal byť priradený do niektorej skupiny (ale striktne nemusí). A každý bežný používateľ (až na pár výnimiek) má pridelený pracovný adresár, ktorému hovoríme *domovský adresár*. Domovský adresár bežného používateľa (špeciálni používatelia domovský adresár spravidla nepotrebujú) sa nachádza na jednotnom mieste ako podadresár, ktorého názov je zhodný s logovacím menom používateľa, napr. adresár */home/oravmir*. Výnimku tvorí používateľ *root*, ktorý má svoj pracovný adresár priamo v koreni súborového stromu, teda */root*.

Súbory passwd a shadow
Všetci používatelia v Linuxe (a unixe všeobecne) sú evidovaní v súbore *passwd*. Tento sa nachádza v adresári */etc*.
Pozrime sa na výpis č.1, kde je fragment súboru */etc/passwd*:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
apache:x:48:48:Apache:/var/www:/sbin/nologin
oravmir:x:500:500::/home/oravmir:/bin/bash
home-pc$:x:501:501::/home/home-pc$:/bin/bash
mirka:x:502:502::/home/mirka:/bin/bash
katka:x:503:503::/home/katka:/bin/bash
```

Každý riadok súboru predstavuje jedného používateľa. Riadok sa skladá z položiek, ktoré sú oddelené dvojbodkou.
Čo znamenajú jednotlivé polia:
- **logovacie meno používateľa** (*username*) - je to meno, ktoré používateľ zadáva pri prihlasovaní na výzvu *login*. Spravidla sa píše malými písmenami, aby nedošlo ku zbytočným zmätkom. Logovacie meno používateľa musí byť jednoznačné, nemôžeme mať v Linuxe dvoch používateľov s menom *oravec*, aj keby sa líšili heslom!
- **zašifrované heslo** (*password*) – technicky táto položka obsahuje heslo používateľa. Ak na tomto mieste vidíme znak *x* (niekde aj ***), jedná sa o *tieňové heslo* (vysvetlíme nižšie)
- **UID** – ako sme si povedali, toto číslo je spojené s používateľom a linie sa celým systémom
- **GID** – implicitný skupinový identifikátor. Je to číslo skupiny v čase prihlásenia
- **popis používateľa** (*user description*). Táto položka obsahuje popisné informácie o používateľovi. To môže byť jeho skutočné meno, napr. *Mirka Oravcová*, alebo drobná poznámka, napr. *programátor v Kylixe*. Ak nie je táto položka vyplnená, v súbore *passwd* sa javí ako prázdny znak (nič medzi dvoma dvojbodkami).
- **domovský adresár používateľa** – názov adresára je zhodný s login menom používateľa na danom mieste, teda */home/login_meno_používateľa*, napr. */home/mirka, /home/katka*
- **používateľský príkazový procesor** – *shell*. V Linuxe môžeme používať niekoľko rôznych príkazových procesorov (prosím nezamieňajme si v tomto prípade slovo „procesor“ so súčiastkou s tým istým menom, ktorá je srdcom počítača. Slovo procesor značí „vykonávač“. *Shell* je obdobou *command.com*-u v prostredí DOS/Windows). Počas vývoja Linuxu sa stal najpoužívanejším shellom tzv. ***bash*** (*Bourne Again Shell*).

Špeciálni používatelia spravidla nepotrebujú shell, a preto je táto položka nastavená na */sbin/nologin* alebo */bin/false* a podobne, čím sú nasmerované „do stratena“.

Všimnime si priradenie UID a GID. Neplatí, že UID musí byť rovnaké s GID. Zároveň sme schopní z výpisu vyčítať, koľko je skutočných bežných používateľov. Sú to *oravmir, home-pc$, mirka, katka* a samozrejme *root*. Zvyšok sú špeciálni používatelia.

Heslá
Vráťme sa na chvíľu k heslám. Heslo je kombinácia znakov, skladajúca sa z písmen, číslic a špeciálnych znakov, ktorá slúži na overenie totožnosti používateľa. Heslo by malo byť minimálne 6 znakov dlhé a nemalo by prekročiť rozsah riadku, teda 80 znakov. Je zrejmé, že čím dlhšie heslo, tým náročnejšie je jeho odhalenie. Aby nebolo možné heslo odhaliť, heslo sa šifruje.

Nedávno
V minulosti sa heslá šifrovali šifrovacím algoritmom *DES* a ich šifrovaná podoba sa ukladala priamo do súboru */etc/passwd* na druhú pozíciu (tam, kde je dnes znak x alebo *).
Pri prihlasovaní používateľa sa po zadaní mena toto overilo v súbore *passwd*. Po zadaní hesla sa toto zadané heslo zašifrovalo algoritmom *DES* a tento zašifrovaný reťazec znakov sa porovnal s tým, ktorý bol uložený v súbore *passwd*. Ak došlo k zhode mena a zašifrovaných reťazcov, používateľ bol identifikovaný a bol mu umožnený vstup do systému.
Ale ako to už vo svete chodí, medzi nami existujú aj rôzni nenechavci a potencionálni škodiči, ktorí sa pod účtom niektorého používateľa snažia vstúpiť do systému a narobiť tam šarapatu. Samozrejme, najradšej si vyberajú účet toho najvyššieho, teda *roota*. Jeho login je jasný, je to *root*, stačí len uhádnuť jeho heslo.
Keďže je súbor *passwd* bežne prístupný na čítanie všetkým používateľom v Linuxe, stačilo z neho zistiť zašifrovaný reťazec, ktorý ukrýval heslo *roota*. Potom pomocou slovníkovej metódy stačí odhaliť heslo *roota*. Vzhľadom na stále rastúci výkon výpočtovej techniky to už netrvá mesiace, ale je to otázka hodín!!!

Dnes
Preto sa pre zvýšenie bezpečnosti pristúpilo k dvom krokom:
Po prvé, heslá sa šifrujú dokonalejším algoritmom *MD5* a po druhé, už sa neukladajú do súboru */etc/passwd*, ale do pomocného súboru */etc/shadow*. Týmto heslám hovorievame ***tieňové heslá***.
Do súboru *passwd* sa na druhú pozíciu u každého používateľa vloží znak *x* (alebo *), čo značí odvolávku na súbor *shadow*. Skutočné zašifrované heslo sa uloží práve v tomto súbore *shadow*. Ten, na rozdiel od súboru *passwd,* už nie je voľne prístupný, ale môže do neho zapisovať a čítať iba *root*.
Pozrime sa na vzor výpisu súboru */etc/shadow* – výpis č.2:

```
root:$1$9zYuijoY$R2TF0m9VQDNt.RWNY9vz./:12283:0:99999:7:::
bin:*:12283:0:99999:7:::
daemon:*:12283:0:99999:7:::
adm:*:12283:0:99999:7:::
lp:*:12283:0:99999:7:::
sync:*:12283:0:99999:7:::
shutdown:*:12283:0:99999:7:::
halt:*:12283:0:99999:7:::
apache:!!:12283:0:99999:7:::
oravmir:$1$SIf01A1y$VpRuw8EiprTRy.AKaTcDy1:12316:0:99999:7:::
home-pc$:!!:12316:0:99999:7:::
mirka:$1$Rz7js9C2$tV2LoGOB38vvjYg9Mbowy/:12321:0:99999:7:::
katka:$1$CsGJzICP$DgCfqQnWpzP/midEArDJi1:12321:0:99999:7:::
```

Vidíme, že sa v tomto súbore nachádzajú login mená používateľov, definovaných v súbore *passwd*. Okrem zašifrovaných hesiel sa tu nachádzajú aj ďalšie dôležité informácie, viažúce sa k heslu používateľa. Poďme sa pozrieť na jednotlivé položky (opäť sú jednotlivé položky oddelené dvojbodkou):

- **logovacie meno používateľa** – je totožné s prvou položkou v súbore *passwd*
- **zašifrované heslo** – tu sa nachádza zašifrovaná podoba skutočného hesla, tzv. tieňové heslo. Všimnime si, že u používateľa *root* je to reťazec *$1$9zYuijoY$R2TF0m9VQDNt.RWNY9vz./*, čo je už pomerne dosť náročné na odhalenie. Ak je na tejto pozícii namiesto reťazca prázdny znak, to znamená, že konkrétnemu používateľovi nebolo heslo stanovené
- **posledná zmena hesla** – toto číslo udáva počet dní od 1.januára 1970, kedy bola vykonaná posledná platná zmena hesla

- **počet dní, po ktorých môže byť heslo zmenené**. Obvykle je to hodnota 0 (nula), čo značí, že heslo môže byť zmenené tak často, ako to používateľ potrebuje
- **počet dní, po ktorých musí byť heslo zmenené**. Toto číslo reprezentuje počet dní, po ktorých uplynutí musí byť heslo zmenené. Ak zmeny hesla nie sú vynútené, položka obsahuje hodnotu 99999
- **počet dní napred, kedy bude používateľ upovedomený, že platnosť jeho hesla vyprší**. Obyčajne je používateľ upovedomený o vypršaní platnosti hesla týždeň dopredu, preto je na tejto pozícii hodnota 7.
- **počet dní medzi skončením platnosti hesla a zablokovaním účtu**. Ak nepožadujeme blokovanie účtu, táto položka obsahuje hodnotu –1 alebo je prázdna
- **počet dní od 1.januára 1970, kedy došlo (alebo dôjde) k zablokovaniu účtu**. Ak nie je požadované zablokovanie účtu, položka obsahuje hodnotu –1 alebo je prázdna
- **rezerva** – táto položka je pripravená pre budúce použitie a býva prázdna.

Teraz je namieste otázka, či nie je jednoduchšie zašifrované heslo uvádzať priamo v súbore */etc/passwd* a tento súbor znepristupniť všetkým okrem roota, tak ako je to u súboru */etc/shadow*.
Nemožno! Prečo?
Z historických dôvodov niektoré programy pracujú so súborom */etc/passwd*. Keby sme ho znepristupnili, tieto programy by prestali fungovať. Preto sa súbor *passwd* necháva naďalej prístupným na čítanie všetkým a samotné heslá sa nachádzajú v pre ostatných používateľov neprístupnom súbore *shadow*.

Slovníková metóda
Už sme tu spomenuli odhalenie hesla pomocou slovníkovej metódy. Tomuto odhaleniu sa hovorí slangovo *prelomenie* hesla alebo skrátene *prielom*. Táto oblasť patrí už do bezpečnosti systému, ale urobme si malú prestávočku a povedzme si, v čom spočíva princíp slovníkovej metódy:
Aby sme to správne pochopili, vžime sa do role hackera, ktorý chce vniknúť do nášho systému.
Najprv musíme získať zašifrovaný reťazec hesla roota. (Ako to urobíme, pre túto ukážku to nie je dôležité!).
Máme ho a vieme, že je to reťazec *$1$9zYuijoY$R2TF0m9VQDNt.RWNY9vz./*. Použijeme ten istý šifrovací algoritmus, akým je toto heslo šifrované. Ten je – bohužiaľ - verejne známy a býva to DES alebo modernejší MD5.
Potom použijeme zoznam rôznych slov, mien, názvov a podobne, o ktorých sa domnievame, že ich mohol používateľ pri tvorbe hesla použiť. Vezmeme prvé slovo, zašifrujeme algoritmom, porovnáme zo získaným reťazcom. Ak sa nezhodujú, pokračujeme s nasledujúcim slovom dovtedy, až sa reťazce zhodujú. Ak nájdeme zhodný reťazec – bingo! – odhalili sme heslo roota. Jednoduché, však? Musíme mať na pamäti, že zoznam týchto slov, ktorému hovoríme *slovník* (odtiaľ aj názov slovníkovej metódy) dnes už číta milióny slov a je bežne dostupný na Internete, vrátane programov na automatické šifrovanie a porovnávanie reťazcov.
Aké je z vyššie napísaného poučenie?
Voľme správne heslá! ***Slovník obsahuje slová, ktoré sa bežne nachádzajú v hovorovej reči a v rôznych jazykoch, mená, priezviská, dátumy a podobne. Ak ste sa už stretli s niektorou sieťou, viete, že používatelia si dávajú ako heslá mená svojich blízkych, mená zvieracích miláčikov alebo ich dátumy narodenia! Tak mi verte, že toto odhalí slovníková metóda za niekoľko minút!***
Meňme často heslá! Nielen svoje rootovské, ale núťme k tomu aj našich používateľov, napr. vhodným nastavením položiek v súbore */etc/shadow*!
Heslá utajujme! Nie je nič horšieho, ako keď si používateľ po donútení zmení heslo a aby ho náhodou nezabudol, tak si ho napíše ceruzou na monitor alebo klávesnicu! (Mávam z toho srdcový kolaps!)

Poznámka:
Veľmi často sa mi stáva, že mi zavolá používateľ a povie mi, aby som mu povedal jeho heslo. On si z neznalosti myslí, že ja ho viem prečítať „tam niekde v tých súboroch". Ale ako je z vyššie popísaného zrejmé, ani root (bez použitia hackerských metód) nedokáže zistiť heslo používateľa. A to je asi jediná vec, čo root – pán všetkých pánov (capo di tutti capi) v Linuxe nedokáže. Inak dokáže naozaj všetko!
Preto takémuto zábudlivcovi môžem ja – ako root – iba heslo zmeniť, povedať mu ho a následne ho donútiť, aby si ho zmenil, aby ani root nevedel, aké heslo má.

Ako zvoliť správne heslo?
To je naozaj zložitá otázka. Človek je tvor, čo si všetko rád zjednodušuje. Preto treba zvoliť také heslo, aby bolo naozaj unikátne, ale aby som si ho zapamätal, alebo aby som si ho vedel odvodiť. Na to si musí každý odvodiť svoj vlastný algoritmus. Jeden, ako príklad, vám prezradím:
Nech som používateľ (trebárs aj root) a nech sa volám *oravec*. Posledné štvorčíslie môjho rodného čísla nech je *4596*. Postupujem takto:

oravec odzadu je *cevaro*. Každé párne písmeno nahradím jednou číslicou z rodného čísla, teda e =4, a=5, o=9. Zvyšnú číslicu 6 pridám na koniec. Potom heslo vyzerá **c4v5r96**.
Tento reťazec sa nenachádza ani v slovníkovej metóde!
Stačí si zapamätať algoritmus a je to v pohode!
Moment, minútočku....
(Práve som bol zmeniť rootovské heslo na všetkých našich serveroch...pre istotu!)
Môžeme pokračovať...

Skupiny
Zoznam skupín, vytvorených v prostredí Linuxu sa nachádza v súbore */etc/group* – výpis č.3:

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
tty:x:5:
disk:x:6:root
lp:x:7:daemon,lp
mem:x:8:
kmem:x:9:
wheel:x:10:root
mail:x:12:mail,postfix
news:x:13:news
uucp:x:14:uucp,nut
apache:x:48:
oravmir:x:500:
home-pc$:x:501:
mirka:x:502:
katka:x:503:mirka
ucto:x:504:katka
```

Tak ako súbor */etc/passwd*, aj súbor *group* musí byť čitateľný (to značí dostupný na čítanie – argument *r*) pre všetkých používateľov systému, aby rôzne aplikácie a programy mohli testovať asociácie medzi používateľmi a skupinami.
Každý riadok súboru *group* predstavuje jednu definovanú skupinu.
Záznam sa skladá v týchto položiek:

- **meno skupiny** – musí byť, tak ako meno používateľa jednoznačné, teda nemôžu v systéme existovať dve skupiny rovnakého mena. Pozor! Nie je podmienkou, aby sa meno skupiny zhodovalo s menom používateľa! Ako sme si povedali, môžeme založiť skupinu účtovateliek s názvom **ucto** a pritom nemusí existovať používateľ **ucto**
- písmeno *x*, nahrádzajúce heslo skupiny
- **GID** – identifikátor skupiny
- **ďalší členovia skupiny** (okrem implicitného člena). Pozrime sa na posledné dva riadky výpisu. Pri vytváraní používateľa **katka** implicitne vznikla skupina s názvom **katka**, ktorej členom sa automaticky stal používateľ **katka**. Pridali sme do skupiny **katka** používateľa **mirka**, takže v skupine **katka** sú títo dvaja členovia – **katka** (implicitne) a **mirka**. Ale my sme vytvorili aj skupinu **ucto**. Keďže sa jedná len o skupinu a nie aj používateľa, v skupine **ucto** nie je implicitný používateľ, ale iba jeden člen, ktorého sme tam pridali a tým je používateľ **katka**. V prípade, že pridáme do skupiny **ucto** ďalších používateľov, ich mená budú zapísané za sebou, oddelené čiarkou bez medzery, ako je to napr. u *uucp:x:14:uucp,nut*.

Vytváranie, editovanie a mazanie používateľov a skupín
Tak ako všetko v Linuxe..., tak aj vytváranie, editovanie a mazanie (jedným slovom administráciu) používateľov a skupín môžeme vykonávať niekoľkými spôsobmi:

- ručne
- pomocou riadkových príkazov
- pomocou (semi)grafických utilít a programov

Ručná administrácia
Ručná administrácia spočíva v úprave jednotlivých súborov */etc/passwd, /etc/shadow a /etc/group*. Takto sa pracovalo v unixovej dobe kamennej. Ale aj dnes, v niektorých prípadoch, keď chceme pozmeniť len jeden argument, je jednoduchšie siahnuť priamo do daného súboru.

Administrácia s využitím riadkových príkazov
Riadkové – inak aj nazývané *konzolové* príkazy sa dnes používajú najčastejšie. Ich použitie je predsa len bezpečnejšie ako priame editovanie súborov. Taký príkaz vie, čo má kde a ako zapísať.
My sme si v seriáli *Začíname s Linuxom* ukázali podrobné použitie riadkových príkazov, takže teraz len zrýchlene letom – svetom na zopakovanie:

a) vytvorenie používateľa - **useradd**
b) stanovenie hesla používateľa - **passwd**
c) mazanie používateľ - **userdel**
d) modifikácia používateľa - **usermod**
e) vytváranie skupiny – **groupadd**
f) mazanie skupiny – **groupdel**

Ak si nevieme spomenúť na všetky dostupné parametre, platí Pravidlo č.1!

Administrácia s využitím (semi)grafických utilít a programov
Medzi najznámejšie utility v textovej aj grafickej podobe patrí program **linuxconf**. Ako som už spomínal, bol (pre mňa z nepochopiteľných príčin) z distribúcií Red Hat Linuxu od verzie 7.x vyňatý.
Druhým efektívnym projektom je **webmin**.
Ak sa teraz pýtate, prečo sa trápime s obsahom súborov alebo riadkovými príkazmi, a nepoužívame „oknoidné“ programy, tak vedzme, že:
a) nie sme obyčajní používatelia, ale rootovia!
b) veľmi často budeme nejaký systém spravovať na diaľku, a to ide iba cez súbory a riadkové príkazy!

Na záver ešte jednu dôležitú poznámku:
Ak zo systému odstránime niektorého používateľa (skupinu), musíme odstrániť aj všetky jeho súbory. Pri pridaní nového používateľa (skupiny) by sa mohlo priradiť uvoľnené UID (GID), ktoré patrili predtým zrušenému vlastníkovi. Tým by nový používateľ (skupina) zdedil aj všetky súbory! To preto, že súbory nie sú viazané na meno, ale UID (GID)!

Linux prakticky ako server/4.časť

V minulej časti sme si povedali niečo o tom, ako Linux rieši používateľov – *userov* svojho systému. Vieme, aké sú to súbory ***passwd, shadow a group***. Pridávanie používateľov je však iba jedna „povinnosť“ správcu. Nie každý deň sa však do systému pridávajú noví používatelia alebo odchádzajú už existujúci.
Taký správca má aj „denné“ povinnosti. A o tých si dnes budeme rozprávať.

Tieto denné činnosti môžeme rozdeliť do niekoľkých skupín:

- pomoc / help
- informácie o používateľoch
- práca s adresármi a súbormi
- práca s diskom
- práca s pamäťou
- práca s procesmi
- ostatná práca na konzole

Pomoc - help
„***Keď všetko zlyhá, otvorím manuál***“ je môj teorém č.1. My sme si už čo to o manuálových stránkach hovorili. Čo ešte nevieme je, že sa stránky delia do desiatich sekcií:

Skupina	Popis
1	Popis používateľských príkazov
2	Popis programových knižníc
3	Popis knižníc jazyka C
4	Popis konfiguračných súborov
5	Popis syntaxu konfiguračných súborov
6	Popis hier
7	Popis práce s textom
8	Popis príkazov pre správu systému
9	Popis linuxového jadra
10	Novinky a ostatné manuálové stránky

Rozdelenie bolo zavedené preto, aby bolo možné sa lepšie v manuálových stránkach orientovať.
Musíme si uvedomiť, že určitý príkaz či program nemusí mať vytvorené všetky sekcie manuálovej stránky. To záleží od tvorcu príkazu (programu) či vývojového týmu, či danú sekciu vytvorí alebo nie. Je jasné, že taký príkaz ***shutdown*** isto nebude mať vytvorenú sekciu 6 (hry).

Manuálovú stránku s určitou sekciou si môžeme nechať vypísať príkazom, napríklad:

```
[root@asterix root]# man 8 smbd
```

Možno si poviete, že prečo by sme mali používať jednotlivé sekcie, keď si môžeme nechať vypísať celé manuálové stránky jednoduchým príkazom

```
[root@asterix root]# man smbd
```

veď je to to isté.

Nie je to úplne pravda. V Linuxe existujú príkazy a súbory rovnakého mena. Ak by sme nepoužívali sekcie v manuálových stránkach, mohlo by sa stať, že sa o tom druhom príkaze či súbore nič nedozvieme.
Takým vhodným príkladom je ***passwd***.
Vieme, že to môže byť meno príkazu na vytváranie hesiel jednotlivým používateľom, ale zároveň je to aj meno súboru, kde sú uložené údaje o jednotlivých používateľoch. Jedná sa teda o dve odlišné veci!

Ak zadáme príkaz:

[root@asterix root]# man passwd

dozvieme sa niečo o programe *passwd* – výpis č.1:

```
PASSWD(1)                 User utilities                PASSWD(1)

NAME
       passwd - update a user's authentication tokens(s)

SYNOPSIS
       passwd [-k] [-l] [-u [-f]] [-d] [-S] [username]

DESCRIPTION
       Passwd is used to update a user's authentication token(s).

       Passwd is configured to work through  the  Linux-PAM API.
       Essentially,  it  initializes itself as a "passwd" service
       with Linux-PAM and utilizes configured password modules to
       authenticate and then update a user's password.
```

Ak však zadáme príkaz:

[root@asterix root]# man 5 passwd

dozvieme sa podstatu súboru *passwd*, ktorý sa nachádza v adresári */etc* - výpis č2:

```
PASSWD(5)                  File formats                 PASSWD(5)


NAME
       passwd - password file

DESCRIPTION
       Passwd  is  a  text file, that contains a list of the sys-
       tem's accounts, giving for each account some useful infor-
       mation like user ID, group ID, home directory, shell, etc.
       Often, it also contains the encrypted passwords  for  each
       account.   It  should  have  general read permission (many
       utilities, like ls(1) use it  to  map  user  IDs  to  user
       names), but write access only for the superuser.
```

Informácie o používateľoch

Veľmi často potrebuje root ale aj bežný používateľ vedieť, kto okrem neho ešte v systéme pracuje. Root to najčastejšie použije vtedy, keď z určitých dôvodov potrebuje systém vypnúť alebo zresetovať. Vtedy je slušnosť preveriť si, či niekto nie je zrovna pripojený. Ak áno, požiadame ho, aby prácu korektne ukončil.
Na zisťovanie informácií o používateľoch je v linuxe niekoľko príkazov.

w

Tento príkaz nás informuje o aktivite používateľov na sieti.
Zadajme

[root@asterix root]# w

Príklad výsledku tohto príkazu je na výpise č.3:

```
 11:39am  up 24 min,  3 users,  load average: 0.28, 0.23, 0.16
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                11:22am  0.00s  0.23s  0.22s  /usr/bin/mc -P
```

```
oravec   tty2     -                11:38am 17.00s  0.04s  0.04s  -bash
oravmir  tty6     -                11:37am  1:53   0.51s  0.01s  /bin/sh /usr/X1
```

V záhlaví nás informuje, ako dlho už systém beží - *up*. Nasleduje podrobný výpis, kde je uvedené prihlasovacie meno (USER), terminál (TTY) z ktorého je pripojený, čas prihlásenia do systému (LOGIN@), dobu jeho neaktivity (IDLE), čas využitý procesmi na danom termináli (JCPU), proces, ktorý práve beží (WHAT) a jeho čas (PCPU).

who
Tento príkaz je o niečo jednoduchší vo svojom výpise.
Ked zadáme

[root@asterix root]# who

na výpise č.4 vidíme zoznam práve prihlásených používateľov, terminál, z ktorého sa prihlásili, dátum a čas prihlásenia:

```
root     tty1     Oct 28 11:22
oravec   tty2     Oct 28 11:38
oravmir  tty6     Oct 28 11:37
```

finger
Tento príkaz nás informuje okrem prihlasovacieho mena používateľa aj o jeho plnom mene, ak je v systéme uvedené. My už vieme, že toto plné meno je uložené v súbore */etc/passwd*. Príklad použitia príkazu ***finger*** je na výpise č.5:

```
Login     Name              Tty      Idle  Login Time   Office     Office Phone
oravec    Miroslav Oravec   tty2       13  Oct 28 11:38
oravmir   mior              tty6       14  Oct 28 11:37
root      root              tty1           Oct 28 11:22
```

Práca so súbormi a adresármi
Pamätáme sa na poučku *nomero uno* – „***V linuxe sa na všetko hľadí ako na súbor***“? Je to naozaj tak a preto práca so súbormi je v Linuxe tá najdôležitejšia. V seriáli *Začíname s Linuxom* sme si čo-to o práci so súbormi povedali. Dnes sa na to pozrieme ďalej.

ls
Príkaz ***ls*** je asi najčastejšie používaným príkazom v príkazovom riadku. Jeho úlohou je vypísanie obsahu príslušného adresára. To, o ktorý adresár sa jedná, záleží od parametrov príkazu *ls*.

Syntaktický zápis je ***ls [voľby][adresár]***

Ak použijeme príkaz bez parametrov, vypíše sa obsah aktuálneho adresára v najjednoduchšom tvare – výpis č.6:

```
Desktop     lsla.txt                samba-2.2.7a-1.i386.rpm
dfh.txt     ls.txt                  samba-3.0.0rc2-2_rh73.i386.rpm
df.txt      manpasswd.txt           samba-3.0.0rc2-2_rh9.i386.rpm
finger.txt  OpenOffice.org1.0.3
freemt.txt  ping1.txt
```

Len pre pripomenutie – aktuálny adresár je ten, v ktorom sa práve nachádzame

Tento stručný výpis môžeme predĺžiť voľbou –*l* (long).

Poznámka:
Voľby začínajú znamienkom mínus -, za ktorým bez medzery nasleduje písmeno označujúce voľbu. Voľby môžeme zlučovať tak, že následnú voľbu uvádzame bez znamienka mínus, napr ls –la.
Toto pravidlo platí pre všetky príkazy v Linuxe.

Ak chceme zobraziť aj skryté súbory, pridáme voľbu *–a* (výpis č.7:)

```
[root@asterix oravmir]# ls -la

celkom 52668
drwx------   22 oravmir  oravmir      4096 okt 28 11:56 .
drwxr-xr-x    6 root     root         4096 okt 28 11:38 ..
-rw-------    1 oravmir  oravmir         0 sep 11 18:19 .autorun.lck
-rw-------    1 oravmir  oravmir       147 sep 27 18:28 .bash_history
-rw-r--r--    1 oravmir  oravmir        24 sep  1 10:14 .bash_logout
-rw-r--r--    1 oravmir  oravmir       191 sep  1 10:14 .bash_profile
-rw-r--r--    1 oravmir  oravmir       124 sep  1 10:14 .bashrc
drwx------    3 oravmir  oravmir      4096 sep 27 18:19 Desktop
-rw-rw----    1 oravmir  oravmir  11832412 dec 11  2002 samba-2.2.7a-1.i386.rpm
-rw-rw----    1 oravmir  oravmir  21155910 sep  2 22:06 samba-3.0.0rc2-2_rh73.i386.rpm
-rw-rw----    1 oravmir  oravmir  20456987 sep  2 22:07 samba-3.0.0rc2-2_rh9.i386.rpm
```

Musíme si uvedomiť, že skryté súbory sa v Linuxe označujú bodkou pred menom súboru. S tým súvisia aj atribúty súborov. Nalistujte si v seriáli *Začíname s Linuxom* časť, kde si vysvetľujeme rozdiel medzi atribútmi súborov v prostredí MS Windows a v Linuxe! Pokým nezvládneme prístupové práva, ich charakteristiku a nastavovanie, nedosiahneme v linuxovom serveri nič pozitívne!
Doporučujem (nariaďujem!) naštudovať túto tématiku v tom rozsahu, ako je uvedená v seriáli *Začíname s Linuxom*! My sa o tieto základy budeme opierať v následnom vysvetľovaní a nebudeme mať čas sa k nim vracať!
Okrem základných príkazov so súbormi a adresármi, ako je *pwd, cd, rm,mv,cp, mkdir a rmdir*, ktoré sú vysvetlené v už vyššie spomenutom seriáli, dobrý správca – *root* používa ešte tieto:

touch
Tento príkaz má viac spôsobov použitia, my sa naučíme ten, ktorý v praxi použijeme najviac a to je možnosť vytvárať nové prázdne súbory.
Ak zadáme príkaz

```
[root@asterix root]# touch vysledok.log
```

tak sa v aktuálnom adresári vytvorí súbor s názvom *vysledok.log*. Ak vás zrovna nenapadá, ako toto využiť, tak počkajte a uvidíte!

cat
Okrem vytvárania prázdnych súborov veľmi často v praxi potrebujeme vypísať obsah súboru (na obrazovku). Na to je vhodný príkaz *cat*:

```
[root@asterix root]# cat subor.txt
```

vypíše obsah súboru na obrazovku.

Ak chceme vypísať súbor v obrátenom poradí, teda od konca súboru až na začiatok, použijeme príkaz ***tac***.
(Ak nás napadá, že názov príkazu je adekvátny jeho funkcii, sme na správnej ceste stať sa linuxovým guru.)

more
Mnohokrát je však vypisovaný súbor dlhší ako je počet riadkov obrazovky a tak sa začiatok súboru síce mihne na obrazovke, ale potom utečie mimo obraz a my si tak môžeme prezerať iba jeho posledné riadky, ktoré na obrazovke zostali stáť. Aby sme *scrollovaniu* (čítaj skrol...) zabránili, použijeme príkaz ***more***. Jeho úlohou je zobraziť súbor po jednotlivých obrazovkách – akoby stránkach. Prečítanú časť posunieme ďalej stlačením medzerníka.

less
less neznamená v tomto prípade *menej*, ale je to vylepšený variant príkazu *more*. Príkaz

```
[root@asterix root]# less subor.txt
```

spôsobí takisto výpis na obrazovku, ale má komfortnejšie ovládanie. Môžeme nielen posúvať výpis dopredu, ale aj naspäť. Stačí, ak použijeme klávesy **PageUp**, **PageDown**, kurzorové šípky, **Enter** a samozrejme medzerník.

file
Príkaz *file* používame na zistenie typu súboru a pritom ten súbor nemusíme otvárať. Syntaktický zápis je *file meno_súboru*.
Príkazom

```
[root@asterix root]# file *
```

dostaneme popis všetkých súborov a podadresárov v danom adresári. Pripomínam, že adresár je tiež určitá forma súboru a tak väčšina príkazov pre súbory platí aj pre adresáre (až na výnimky ako *mkdir*, *rmdir* a pod).

du
Chceme vedieť veľkosť niektorého súboru alebo celého adresára na disku?
Stačí, ak použijeme *du*.
Syntaktický zápis je *du [voľby][súbor alebo adresár]*.

Ak chceme zistiť veľkosť všetkých súborov, pridáme parameter *–a* (all).
Ak sa nám nepáči výpis veľkosti v blokoch, pridáme voľbu *-b* pre výpis v bajtoch alebo *-k* pre výpis v kilobajtoch.

Vyhľadávanie súborov
Na vyhľadávanie súborov sa v Linuxe (a samozrejme aj unixe) používa veľmi mocný príkaz ***find***. *find* má mnoho rôznych volieb a kritérií, podľa ktorých vyhľadáva.
Predstavme si, že chceme vyhľadať súbor podľa mena *zaloha.tgz*, ale nevieme, kde sa môže nachádzať, takže radšej začneme hneď od koreňa. Výsledok chceme vypísať na obrazovku.
Zadáme teda príkaz

```
[root@asterix root]#  find / -name zaloha.tgz –print
```

Keď *find* nájde viac súborov s tým istým menom, vypíše ich všetky. Výpis spôsobí práve parameter *–print*.
Ak chceme vyhľadávať podľa iného kritéria, napr. veľkosti, použijeme namiesto parametra *–name* iný parameter *–size*. Ako sme si povedali, *find* je mocný nástroj a ak chceme používať aj iné voľby, preštudujme si manuálové stránky príkazu *find* (no ako? Predsa *man find*, nie...?)

locate
Druhou možnosťou, ako nájsť súbor, je použitie príkazu *locate*. Ten však pracuje na inom princípe ako *find*. *locate* nevyhľadáva súbor skutočne na disku, ale vo svojej databáze, kde má zapísané umiestnenie všetkých súborov, stačí sa len pozrieť. Tým dosahuje rýchlejších výsledkov ako *find*. Nevýhodou je, že tá databáza nemusí byť vždy aktuálna. Preto ak chceme využívať *locate*, musíme zabezpečiť aktualizáciu databázy príkazom *locate –U*. Pre zautomatizovanie toto môžeme robiť denne alebo týždenne s použitím programu *cron*.

grep
Príkaz *grep* (*Global Regular Expression Printer*) vyhľadáva zadaný reťazec znakov v príslušnom textovom súbore. Ak tento reťazec nájde, vypíše na obrazovku riadok, kde sa reťazec v súbore nachádza.
Syntaktický zápis je

grep [voľby] reťazec meno_súboru

Najčastejšie voľby sú *–i* a *–l*.
-i použijeme v prípade, ak nám nezáleží na veľkosti písmen v hľadanom reťazci (vieme, že Linux je citlivý na malé a veľké písmená).

-l použijeme vtedy, ak namiesto riadku s výskytom reťazca požadujeme meno súboru, kde sa reťazec nachádza, napr. *grep –l SMB *.log* vypíše všetky súbory s príponou *log*, v ktorých sa vyskytuje reťazec *SMB*.
Najčastejšie však *grep* budeme používať v spojení s rúrou (nie mikrovlnou ani na pečenie).

Presmerovanie vstupu a výstupu
So súbormi veľmi tesne súvisí presmerovanie vstupu a výstupu.
Niekedy budeme potrebovať presmerovať výstup niektorého programu alebo príkazu namiesto na obrazovku do určitého súboru. Na to využijeme operátor presmerovania výstupu. Je to znak >.
Ukážeme si úplne konkrétny príklad. Myslíte si, že som vyššie spomenutý výpis č. 7 opisoval z obrazovky ručne? Ale nie, stačilo len presmerovať výstup príkazu *ls* do súboru napr. s menom *vypis_ls.txt* takto:

```
[root@asterix root]#  ls –la>vypis_ls.txt
```

Výstup príkazu, čo je v tomto prípade obsah aktuálneho adresára sa nezobrazil na obrazovke, ale sa uložil do súboru *vypis_ls.txt* na disk. Potom som len jednoducho vzal súbor *vypis_ls.txt* a nakopíroval do tohto textu.
V prípade, že súbor *vypis_ls.txt* predtým neexistoval, bude automaticky vytvorený. V prípade, že existoval, bude pôvodný obsah zmazaný a nahradený novým obsahom.

Niekedy sa stáva, že sa nejaký príkaz spúšťa opakovane a jeho výsledok si chceme ukladať do súboru tak, aby sa pôvodný obsah nevymazal, ale aby sa nový obsah pripojil na koniec pôvodného obsahu. Vtedy stačí použiť operátor doplňovania >>.
Opakované spúšťanie príkazu

```
[root@asterix root]#  ls –la>>výpis_ls.txt
```

zabezpečí uloženie všetkých výsledkov daného príkazu do jedného súboru.
Na tomto princípe sú v Linuxe založené logovacie súbory.

Predstavme si, že často vykonávame určitú činnosť, kde sme nútení zadávať určité hodnoty z klávesnice. Aby sme ich nemuseli stále naťukávať, stačí, ak tieto hodnoty uložíme do určitého súboru a tento použijeme ako vstup pre daný program.
Presmerovanie vstupu programu zabezpečíme operátorom <.
Ak spustíme príkaz na vytvorenie FTP spojenia takto

```
[root@asterix root]#  ftp<pokyny.txt
```

bude si tento príkaz odoberať príslušné hodnoty zo súboru *pokyny.txt* tak, ako keby sme mu ich zadávali ručne z klávesnice.

Rúra – pipe
Rúra (angl. *pipe*) slúži ku kombinácii príkazov a ich rozširovaniu. Takto môžeme zreťaziť niekoľko príkazov za sebou, pričom každý z nich bude považovať svoj vstup napojený na výstup toho predchádzajúceho. Operátorom rúry je |.
Príkaz

```
[root@asterix root]#  ls –la|grep holababa.gif
```

spôsobí výpis obsahu príslušného adresára, ale tento výpis nebude na obrazovke, ani presmerovaný do súboru na disk, ale priamo pretečie rúrou ako vstup príkazu *grep*. Ten v tomto výpise vyhľadá reťazec s menom *holababa.gif* a ak ho nájde, zobrazí riadok, kde sa tento vyskytuje. Vidíme, že je to meno súboru, je toľko a toľko veľký a jeho vlastníkom je ten a ten.
(Využitie rúry spolu s grepom si ukážeme ešte efektívnejšie ako je hľadanie nahých ženských...)

Práca s diskom
Môj teorém č.2 hovorí: ***Čím väčší disk, tým sa rýchlejšie zaplní***.
Vieme, že disky nie sú nekonečné a preto treba čas do času sledovať ich zaplnenosť. Na to slúži príkaz ***df*** (disk full). Obecný zápis je *df [voľby]*.
Zadajme teraz príkaz

```
[root@asterix root]# df
```

Pozrime sa na výpis.č.8:

```
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda3             20018668   3924792  15076964  21% /
none                    127920         0    127920   0% /dev/shm
```

Trochu nejasné, nie?

Preto pridáme voľbu *–h* :

```
[root@asterix root]# df -h
```

a pozrime sa na výpis č.9:

```
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda3              19G  3.8G   14G  21% /
none                  125M     0  124M   0% /dev/shm
```

Tomuto už rozumieme lepšie, pretože veľkosť udávanú v blokoch previedol na zrozumiteľnejšiu hodnotu.

free

Už vieme, ako máme obsadený harddisk, ale ako je to s operačnou pamäťou? Na to slúži príkaz *free*.
Má niekoľko pekných volieb, napr:
-*b* pre výpis v bajtoch
-*k* pre výpis v kilobajtoch
-*m* pre výpis v megabajtoch
-*t* napíše aj celkový súčet.

Tak zadajme:

```
[root@asterix root]# free -mt
```

a zistíme skutočný stav – výpis č10:

```
             total       used       free     shared    buffers     cached
Mem:           249        240          9          0         11         76
-/+ buffers/cache:        152         97
Swap:          298          0        298
Total:         547        240        307
```

Tak, na dnes by to bolo všetko, nabudúce sa budeme venovať procesom.
Čo to vravíte? Že je to nezáživné a nudné?
No, to možno je, ale patrí to k správe servera tak, ako volant k autu – bez toto sa nepohneme!

Linux prakticky ako server/ 5.časť

V minulej časti sme sa venovali „jednohubkám“ – dennodenným úlohám a činnostiam dobrého správcu siete. Tieto činnosti sme si rozdelili do niekoľkých skupín. Aj keď práca s manuálovou stránkou, pamäťou, diskom či používateľmi je naozaj dôležitá, prím v Linuxe hrá práca so súbormi a procesmi. O súboroch už toho vieme pomerne dosť, dnes si budeme hovoriť o procesoch.
So starým známym, ale zato pravdivým heslom, že „základným stavebným prvkom spoločnosti je rodina“ ma napadla asociácia: „základným stavebným prvkom Linuxu je proces“. A skutočne, po súboroch sú procesy to najdôležitejšie v Linuxe.

Základné pojmy
Aby sme správne pochopili podstatu činností, ktoré budeme ako správcovia vykonávať, musíme si uviesť niekoľko základných pojmov. Všade v odbornej literatúre sa to len tak hemží výrazmi ako *program, súbor, proces, démon, úloha, služba* a podobne. Ale nikde (skutočne nikde) som nenašiel podrobnejšie vysvetlenie týchto pojmov. No a aby sa nám to neplietlo, pokúsim sa tu dnes uviesť moje definície.

Súbor (File)
Súbor je presne definovaná postupnosť určitých znakov. Typ týchto znakov určuje využitie súboru. Poznáme súbory textové, ktoré obsahujú znaky, tvoriace textový dokument, alebo súbory datové, ktoré obsahujú znaky, tvoriace informácie, súbory grafické, ktoré obsahujú znaky tvoriace obrázky, filmy a podobne. Súbory sa ukladajú na určité médium, napr. disketu, harddisk, CD ROM, ale aj do pamäti zariadenia.

Program (Program)
Program je *súbor* základných elementárnych inštrukcií (príkazov) pre daný počítač, ktorého cieľom je splnenie určitej požadovanej úlohy.

Inštancia programu
Program môže splniť svoju úlohu len vtedy, ak je spustený. To znamená, že pokiaľ program nie je spustený, nemôže vykonať žiadnu činnosť. Program sa môže spustiť len vtedy, ak je zavedený do operačnej pamäte počítača. Tomuto zavedeniu programu do operačnej pamäte hovoríme *inštancia programu*.

Proces (Process)
Nezávisle bežiacej *inštancii programu*, ktorá má vlastné systémové zdroje (napr. veľkosť operačnej pamäte) hovorievame *proces*.

Skúsime to zhrnúť:
Proces je určitý program, ktorý sa skladá z postupnosti znakov, ktoré procesor interpretuje ako strojové inštrukcie, dáta a zásobník. Jadro Linuxu dômyselne plánuje beh procesov tak, že vzniká dojem, ako keby boli procesy spracovávané naraz – simultánne. Niekoľko procesov môže byť inštanciou toho istého programu.

Každý proces počas svojho života prechádza tromi štádiami. Najprv *vznikne*, potom *je* určitú dobu *v činnosti* a nakoniec *zaniká*. Ako príklad si môže predstaviť program, ktorého úlohou je sčítať dve čísla. Nech sa takýto program volá – no, čo tak ***suma***.

Je program ***suma*** súborom? Je, veď je to postupnosť príkazov a istotne sa bude v počítači nachádzať na niektorom médiu – najčastejšie na harddisku. Pokiaľ leží len tak na disku a nič nevykonáva, je to len jeden z typov súborov – *programov*. Keď však z klávesnice zadáme meno tohto programu – ***suma***, zavedieme ho do operačnej pamäte počítača, kde sa následne spustí. Takto vznikne *proces*. Ten bude určitú dobu v činnosti, napr. si vyžiada vstup daných čísiel, potom ich spočíta a vypíše výsledok. Keď splní svoju povinnosť, proces zanikne. Uvoľní operačnú pamäť počítača a tak je program znovu pripravený k činnosti.
Jasné?

Úloha (Task)
Čo bolo povinnosťou procesu ***suma***? Splniť *úlohu*! Môžeme povedať, že *úloha* je jednorázová činnosť daného procesu.

Démon (daemon)
Existuje však proces, ktorý po svojom vzniku zostáva dlhodobo, spravidla natrvalo v operačnej pamäti počítača (samozrejme, že len do vypnutia počítača, potom sa – logicky – stratí). Takémuto „stále žijúcemu“ procesu sa hovorieva *démon* (myslím, že to meno je naozaj výstižné).

Služba (Service)
Aj démon vykonáva svoju činnosť a plní zadané úlohy. Keďže však po splnení úlohy nezaniká, ale ostáva v pamäti počítača, aby mohol opakovane plniť úlohy, môžeme povedať, že trvalo slúži a tak vykonáva určitú *službu*. *Služba* je opakovaná činnosť daného procesu – démona. Ako príklad si môžeme uviesť taký web server Apache. Je to vlastne démon, ktorý opakovane vykonáva určitú činnosť – koná službu. Ak ho niekto požiada o poskytnutie niektorej web stránky, on mu ju poskytne. Neopúšťa operačnú pamäť , ale zase načúva požiadavkám svojich klientov a plní ich požiadavky.

Rodina procesov
Keďže je Linux viacúlohový systém, môže v ňom – a spravidla aj naozaj beží viac procesov naraz. Teraz nie je podstatné, ako to je zariadené. Dôležité je jednotlivé procesy od seba odlíšiť. Jadro si o všetkých spustených procesoch udržuje prehľad pomocou jednoznačne priradeného čísla **PID** – *Process Identification*.
Procesy v Linuxe tvoria hierarchickú štruktúru, teda každý proces (okrem toho prvého) má svojho predka – rodiča, ktorému sa hovorí ***rodičovský proces***. Zároveň ten istý proces môže (ale nemusí) vytvoriť svojho ***potomka***.
To znamená, že každý proces nesie okrem svojho čísla PID aj informáciu o svojom rodičovi – tzv. **PPID** (*Parent Process Identification*).
A načo je to dobré? Podľa čísla PPID sme schopní určiť, ktorým procesom bol daný proces vytvorený. Neskôr uvidíme, že PPID má aj iné využitie.

Procesy sa číslujú od čísla 1. Jednotka je priradená najdôležitejšiemu procesu v celom systéme Linuxu. Tento proces sa volá ***init*** a už z názvu si môžeme domyslieť, že inicializuje – spúšťa všetky ostatné procesy a teda služby v celom systéme. My sa procesom *init* budeme ešte patrične zaoberať na inom mieste. Zatiaľ nám stačí vedieť, že init je taký „praotec“ všetkých ostatných procesov.

Informácie o procesoch v systéme
Zoznam procesov v systéme Linux je možné získať pomocou príkazu **ps**.
Program má veľa rôznych parametrov, ktorými môžeme príkaz modifikovať.
Ak spustíme samotný príkaz *ps* bez parametrov, vypíše sa zoznam aktívnych procesov spojených s aktuálne použitým terminálom alebo konzolou - výpis č.5-1:

```
[root@asterix root]# ps
  PID TTY          TIME CMD
 1327 pts/1    00:00:00 bash
 1683 pts/1    00:00:00 ps
```

Pozrime sa teraz na predchádzajúci výpis:
Prvý stĺpec je označený *PID* a my už vieme, o čo sa jedná. Pripomeňme si, že PID musí byť v celom systéme jednoznačné, teda nemôžu existovať dve rovnaké čísla PID. Stĺpec *TTY* popisuje aktuálny terminál. Stĺpec *TIME* popisuje čas trvania procesu. Posledný stĺpec *CMD* charakterizuje príkaz shellu, ktorý spôsobil vytvorenie procesu.

Vidíme, že na danom výpise sa nachádzajú iba dva procesy. Prvý je shell *bash* a druhý je samotný príkaz *ps*.
Ale prečo sú hodnoty stĺpca TIME pre obidva procesy nulové?
Proces *ps* trvá veľmi malú nezaznamenateľnú dobu a prebehne takmer okamžite a bash väčšinu času strávi čakaním na vstup s klávesnice

Výpis príkazu je možné ďalej upravovať. Môžeme použiť dva druhy parametrov príkazu *ps*. Prvým druhom parametrov môžeme špecifikovať množinu vypisovaných procesov. Druhým typom parametrov definujeme spôsob zobrazovania vypisovaných informácií.
Množinu zobrazovaných procesov môžeme ovplyvniť týmito parametrami:
Parameter **–a** („mínus a“, aj s tým znakom mínus!) zobrazí informácie o všetkých aktívnych procesov, riadených nejakým terminálom. Zadajme teda príkaz:

```
[root@asterix root]# ps –a
```

a pozrime sa na výpis č.5-2:

```
  PID TTY          TIME CMD
 1269 tty1     00:00:00 mc
 1477 tty6     00:00:00 startx
 1488 tty6     00:00:00 xinit
 1497 tty6     00:00:00 startkde
 1593 tty6     00:00:00 kwrapper
 1680 pts/1    00:00:00 ps
```

Záhlavie jednotlivých stĺpcov je totožné s predchádzajúcim príkladom, ale obsah je už trochu iný. Všimnime si, že sú vypísané aj procesy, bežiace na iných termináloch. Ak sa pozrieme aj na posledný stĺpec, všimneme si, že na *tty6*, čo predstavuje siedmu konzolu na klávesnici (Alt-F7) sa spustilo prostredie KDE (začalo to príkazom *startx*, to ostatné sa od neho odvinulo automaticky).

Zadajme prezmenu príkaz *ps* s prepínačom **–e**, ktorý spôsobí zobrazenie úplne všetkých procesov, teda i tých, čo nie sú nejakým spôsobom napojené na terminál:

```
[root@asterix root]# ps -e
```

Zobrazenie je na Výpise č.5-3:

```
  PID TTY          TIME CMD
    1 ?        00:00:04 init
    2 ?        00:00:00 keventd
    3 ?        00:00:00 kapmd
    4 ?        00:00:00 ksoftirqd_CPU0
    5 ?        00:00:00 kswapd
    6 ?        00:00:00 bdflush
    7 ?        00:00:00 kupdated
    8 ?        00:00:00 mdrecoveryd
   12 ?        00:00:00 kjournald
   91 ?        00:00:00 khubd
  618 ?        00:00:00 eth0

--- skrátené ---

 1269 tty1     00:00:00 mc
 1270 ?        00:00:00 cons.saver
 1271 pts/0    00:00:00 bash
 1477 tty6     00:00:00 startx
 1488 tty6     00:00:00 xinit
 1489 ?        00:00:01 X
 1497 tty6     00:00:00 startkde
 1622 ?        00:00:00 kdeinit
 1625 ?        00:00:00 kalarmd
 1629 tty5     00:00:00 bash
 1692 ?        00:00:00 kdesktop_lock
 1693 ?        00:00:00 kblankscrn.kss
 1718 pts/1    00:00:00 ps
```

Všimnime si úplne prvého riadku! Vidíme, že to začína praotcom všetkých procesov – procesom *init* s hodnotou PID = 1. Tento proces, tak ako niektoré ostatné nemajú žiadnu väzbu na terminál, lebo ich spustil samotný systém Linuxu. Preto je v stĺpci TTY namiesto hodnoty otáznik (výpis je skrátený, vy si to vyskúšajte na svojej klávesnici a budete prekvapení, koľkože to procesov beží na vašom miláčikovi).
Že je výpis veľmi dlhý a len tak vám prebehol obrazovkou? Ale, ale, vy beťári, však ste nedávali minule pozor! Vyskúšajte spojenie rúry s príkazom *more*, napr. takto:

```
[root@asterix root]# ps –e|more
```

Ak nás zaujíma, ktoré procesy patria konkrétnemu používateľovi, použijeme parameter **–u**. Napríklad nás zaujímajú procesy používateľa s logovacím menom *oravec*:

```
[root@asterix root]# ps –u oravec
```

A na výpise č.5-4:

```
  PID TTY          TIME CMD
 1629 tty5     00:00:00 bash
```

Vidíme, že používateľom *oravec* je spustený iba jeden proces – je to bash na šiestej konzole (prvá konzola tty0 je Alt-F1, tty5 je Alt-F6).

Ako sme si už povedali, programu *ps* môžeme zadať aj parametre, ktoré ovplyvnia spôsob vypisovania daných informácií. Poznáme tri možnosti vypisovania informácií:

- normálny výpis – zapína sa parametrom –f
- dlhý výpis – zapína sa parametrom –l („mínus el“, nie jedna!)
- používateľom definovaný výpis – tu môže používateľ sám definovať, ktoré stĺpce sa majú zobraziť

Pozrime sa na použitie obidvoch výpisov, pričom môžeme vyššie spomenuté parametre spolu kombinovať:

```
[root@asterix root]# ps –a –f
```

(výpis č.5-5:)

```
UID        PID  PPID  C STIME TTY          TIME CMD
root      1269  1216  0 08:40 tty1     00:00:00 /usr/bin/mc -P
root      1477  1427  0 09:00 tty6     00:00:00 /bin/sh /usr/X11R6/bin/startx
root      1488  1477  0 09:00 tty6     00:00:00 xinit /etc/X11/xinit/xinitrc --
root      1497  1488  0 09:00 tty6     00:00:00 /bin/sh /usr/bin/startkde
root      1593  1497  0 09:01 tty6     00:00:00 kwrapper ksmserver --restore
root      1838  1327  0 09:48 pts/1    00:00:00 ps -a -f
```

alebo

```
[root@asterix root]# ps –a –l
```

(výpis č.5-6:)

```
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
000 S     0  1269  1216  0  75   0    -   910 schedu tty1     00:00:00 mc
000 S     0  1477  1427  0  76   0    -   552 wait4  tty6     00:00:00 startx
100 S     0  1488  1477  0  75   0    -   569 wait4  tty6     00:00:00 xinit
100 S     0  1497  1488  0  75   0    -   556 wait4  tty6     00:00:00 startkde
000 S     0  1593  1497  0  75   0    -   355 schedu tty6     00:00:00 kwrapper
100 R     0  1836  1327  0  76   0    -   780 -      pts/1    00:00:00 ps
```

Niektoré stĺpce už poznáme – *UID* je predsa identifikátor linuxového používateľa.
Pribudol stĺpec *PPID* a my vieme, že sa jedná o PID rodiča – teda toho procesu, ktorý vytvoril daný proces. Veľmi dobre je to vidieť u procesov *startx, xinit, startxkde*.
startx má PID 1477, *xinit* má PID 1488, ale PPID má 1477, čo je PID procesu *startx*. To značí, že proces *xinit* bol vytvorený procesom *starx*. A následne, *xinit* vytvoril *startkde*.

Stĺpec *C* predstavuje percentuálnu vyťaženosť procesora. Vidíme, že ani jeden proces nedáva procesoru zabrať.
Stĺpec *STIME* definuje systémový čas vzniku procesu.
Stĺpec F predstavuje masku procesu a tak ako stĺpec *WCHAN*, zobrazujúci akciu, na ktorú proces čaká, nedokážeme ich zatiaľ plne využiť.

Stĺpec *S* zobrazuje stav (status) procesu. Dosahuje týchto hodnôt:
- **S** – spiaci proces
- **R** –bežiaci proces
- **T** – pozastavený proces
- **W** – proces nie je momentálne v pamäti

Stĺpec *SZ* zobrazuje počet blokov pamäti, zabrané obrazom programu a *PRI* predstavuje plánovanú prioritu procesu. Čím je číslo väčšie, tým je priorita menšia.

Používateľ si môže sám nadefinovať, ktoré stĺpce by chcel zobraziť. Na to slúži voľba **–o**, za ktorou nasledujú názvy jednotlivých stĺpcov, oddelené čiarkou, napr:

```
[root@asterix root]# ps –u root –o pid, ppid, s, cmd
```

Asi však najčastejšie používaným parametrom pre nás bude parameter **a** (samotné „a“ bez mínusu). Tento parameter zabezpečí výpis všetkých (*a = all = všetky*) bežiacich procesov.
A druhým parametrom bude **x**, ktorý zabezpečí výpis aj tých procesov, ktoré nie sú naviazané na žiadny terminál.
Zadajme teda

```
[root@asterix root]# ps ax
```

alebo

```
[root@asterix root]# ps ax|more
```

a na výpise č.5-7 uvidíme žiadané informácie:

```
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:04 init
    2 ?        SW     0:00 [keventd]
    3 ?        SW     0:00 [kapmd]
    4 ?        SWN    0:00 [ksoftirqd_CPU0]
    5 ?        SW     0:00 [kswapd]
    6 ?        SW     0:00 [bdflush]
    7 ?        SW     0:00 [kupdated]
    8 ?        SW     0:00 [mdrecoveryd]
   12 ?        SW     0:00 [kjournald]
   91 ?        SW     0:00 [khubd]
  618 ?        SW     0:00 [eth0]
  701 ?        S      0:00 syslogd -m 0
  706 ?        S      0:00 klogd -x
  726 ?        S      0:00 portmap
  754 ?        S      0:00 rpc.statd
  882 ?        S      0:00 /usr/sbin/apmd -p 10 -w 5 -W -P /etc/sysconfig/apm-sc
  937 ?        S      0:00 /usr/sbin/sshd
  970 ?        S      0:00 xinetd -stayalive -reuse -pidfile /var/run/xinetd.pid
  988 ?        S      0:00 smbd -D
  993 ?        S      0:00 nmbd -D
 1011 ?        S      0:00 gpm -t ps/2 -m /dev/mouse
 1029 ?        S      0:00 crond
 1117 ?        S      0:00 xfs -droppriv -daemon
 1153 ?        S      0:00 /usr/sbin/atd
 1190 ?        S      0:00 cupsd
 1197 tty1     S      0:00 /sbin/mingetty tty1
 1198 tty2     S      0:00 /sbin/mingetty tty2
 1199 tty3     S      0:00 /sbin/mingetty tty3
 1200 tty4     S      0:00 /sbin/mingetty tty4
 1201 tty5     S      0:00 /sbin/mingetty tty5
 1202 tty6     S      0:00 /sbin/mingetty tty6
 2093 ?        S      0:00 /usr/sbin/sshd
 2096 pts/0    S      0:00 -bash
 2146 pts/0    R      0:00 ps ax
```

Utilita top

Okrem príkazu ps existuje aj interaktívna utilita, ktorá sa nazýva **top**. Na obr.č.5-7 je príklad výpisu informácií pomocou tejto utility:

```
  1:40pm  up  1:42,  1 user,  load average: 0,00, 0,00, 0,00
34 processes: 32 sleeping, 2 running, 0 zombie, 0 stopped
CPU states:  0,0% user,  0,0% system,  0,0% nice, 100,0% idle
Mem:   255844K av,  245988K used,    9856K free,       0K shrd,   34140K buff
Swap:  305224K av,       0K used,  305224K free                   34600K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
    1 root      15   0   476  476   420 S     0,0  0,1   0:04 init
    2 root      15   0     0    0     0 SW    0,0  0,0   0:00 keventd
    3 root      15   0     0    0     0 SW    0,0  0,0   0:00 kapmd
    4 root      34  19     0    0     0 SWN   0,0  0,0   0:00 ksoftirqd_CPU0
    5 root      15   0     0    0     0 SW    0,0  0,0   0:00 kswapd
    6 root      15   0     0    0     0 SW    0,0  0,0   0:00 bdflush
    7 root      15   0     0    0     0 SW    0,0  0,0   0:00 kupdated
    8 root      25   0     0    0     0 SW    0,0  0,0   0:00 mdrecoveryd
   12 root      15   0     0    0     0 SW    0,0  0,0   0:00 kjournald
   91 root      15   0     0    0     0 SW    0,0  0,0   0:00 khubd
  618 root      15   0     0    0     0 SW    0,0  0,0   0:00 eth0
  701 root      15   0   588  588   500 S     0,0  0,2   0:00 syslogd
  706 root      15   0   444  444   384 S     0,0  0,1   0:00 klogd
  726 rpc       15   0   544  544   464 S     0,0  0,2   0:00 portmap
  754 rpcuser   18   0   720  720   624 S     0,0  0,2   0:00 rpc.statd
  882 root      15   0   480  480   424 S     0,0  0,1   0:00 apmd
  937 root      15   0  1232 1232  1120 S     0,0  0,4   0:00 sshd
  970 root      15   0   960  960   756 S     0,0  0,3   0:00 xinetd
  988 root      15   0  2648 2648  1732 S     0,0  1,0   0:00 smbd
  993 root      15   0  2252 2252  1488 S     0,0  0,8   0:00 nmbd
 1011 root      18   0   440  440   384 S     0,0  0,1   0:00 gpm
 1029 root      15   0   616  616   540 S     0,0  0,2   0:00 crond
 1117 xfs       15   0  4248 4248   840 S     0,0  1,6   0:00 xfs
 1153 daemon    15   0   524  524   460 S     0,0  0,2   0:00 atd
```

Táto utilita je interaktívna, bližšie info zistíme po stlačení klávesu **h** (help).

Nabudúce budeme pokračovať v procesoch. Ja viem, že sa vám to nie veľmi páči, ale je to naozaj nutné. V tejto druhej časti uvidíme, ako sa nám táto teória zíde pri ovládaní serveru.

Linux prakticky ako server/6.časť

V minulej časti sme si povedali niečo o procesoch. Už vieme , čo je to proces, ako vzniká, ako pôsobí a čo vykonáva. Keďže sme minule všetko nestihli, dnes budeme pokračovať v rozprávaní o procesoch.

Beh procesu na popredí a pozadí
Povedali sme si, že všetky procesy (okrem démonických) vzniknú, prebehnú a zaniknú. Ako príklad sme si uviedli program **suma**. Ten mal jednu špecifickú vlastnosť – bežal na popredí (*foreground*). To znamená, že počas behu procesu bola konzola, z ktorej bol program spustený, obsadená, takže sme ju nemohli používať. Príkazový interpreter shell čoby rodičovský proces čakal na dokončenie synovského procesu – programu **suma**.
Predstavme si, že máme iný program, ktorý vykonáva časovo veľmi náročnú úlohu – napríklad vykoná kompresiu súboru, v ktorom je uložený nejaký film. Nech sa tento program nazýva **komprimuj**.
Keďže beh takéhoto procesu je časovo náročný, nebolo by veľmi výhodné, aby bola na dlhú dobu klávesnica nepoužiteľná.

Existuje však jedno riešenie – presunúť činnosť procesu na pozadie (*background*). Od procesu, bežiaceho na pozadí sa očakáva, že nebude obťažovať používateľa – nebude ho obmedzovať v práci v shelli, bude samostatným a nebude interaktívny. Čo to znamená? Že nebude očakávať vstup z klávesnice – v opačnom prípade by bola jeho činnosť zastavená a nebol by to už proces *bežiaci*, ale *pozastavený*.
A ešte jedna dôležitá vlastnosť – proces bežiaci na pozadí by nemal nič vypisovať na obrazovku, lebo by to bolo značne rušivé.
Pozrime sa znova na náš imaginárny program **komprimuj**. Je to vzorový príklad procesu, vhodného pre beh na pozadí - jeho činnosť trvá veľmi dlho (skomprimovať taký dvojhodinový film o veľkosti niekoľkých gigabajtov môže trvať aj hodiny!), nepožaduje žiadny vstup z klávesnice od používateľa a nič nevypisuje na obrazovku. (Nech nás teraz netrápi, ako by taký program vyzeral z programátorského hľadiska.). Takémuto programu sa hovorí „úplne tichý“.

Beh programu na pozadí môže nastať v týchto dvoch prípadoch:
- spustenie na pozadí
- presun na pozadie

Aký je medzi nimi rozdiel?
Spustenie na pozadí sa vykonáva hneď – zadaním z klávesnice. *Presun na pozadie* sa vykonáva u programu, ktorý pôvodne bol spustený na popredí a ktorý chceme z určitých dôvodov na pozadie presunúť.
Poďme si rozobrať obidva prípady:

Spustenie na pozadí
Predstavme si, že chceme skomprimovať súbor s filmom pomocou programu **komprimuj**. Zo skúsenosti vieme, že jeho činnosť trvá veľmi dlho a tak by obsadila na značný čas klávesnicu, preto spustíme program na pozadí. To dosiahneme tak, že za názov súboru v príkazovom riadku pridáme znak **&** (*ampersand*).
Zadáme

```
[root@obelix root] # /bin/komprimuj &
```

Na obrazovke sa objaví niečo podobné:

```
[1] 14778
```

Prvé číslo v hranatých zátvorkách je *identifikátor procesu* v rámci aktuálneho shellu. To druhé číslo je už nám dobre známe číslo PID.
V rámci každého shellu sú procesy číslované od čísla 1 (jedna). Na proces spustený v aktuálnom shelli je možné sa naďalej pri práci odkazovať obidvoma číslami. Na proces spustený z iného shellu je možné odkazovať sa iba číslom PID. Je to aj zrejmé – čísla PID registruje samotné jadro systému, čísla procesov v rámci shellu uchováva samotný shell.
Úspešné dokončenie činnosti programu **komprimuj** sa oznámi používateľovi hlásením na obrazovke:

```
[1] +  Done                    /bin/komprimuj
```

V prípade, že program je ukončený volaním jadra *exit( )*, hlásenie bude mať tvar:

```
[1]+    Exit 10                                    /bin/komprimuj
```

Číslo za slovom *Exit* vyjadruje návratovú hodnotu programu.

Poznámka:
Každý program v Linuxe končí svoju činnosť s určitou návratovou hodnotou. Štandardná hodnota je 0 (nula) a znamená úspešný koniec programu. Ostatné hodnotu znamenajú chyby alebo výnimočné situácie. Jednotlivé hodnoty je možné nájsť v manuálových stránkach daného programu.

Zapamätajme si!
Všetky démoni, poskytujúce služby v systéme, sú programy spúšťané na pozadí!

Presun na pozadie
Predstavme si situáciu, že sme spustili program **komprimuj** na popredí konzoly. Ale potom sme si uvedomili, že činnosť bude trvať dlhší čas a my budeme mať blokovanú konzolu. Preto presunieme program na pozadie.
Je to trochu iný problém, ako priame spustenie na pozadí.
Keďže program už beží na popredí, je nutné ho najprv *pozastaviť*. To dosiahneme stlačením **Ctrl–Z** (najprv stlačíme kláves *Ctrl*, pridržíme a následne stlačíme kláves *Z*).
Stlačíme teda *Ctrl-Z* a na obrazovke sa vypíše:

```
[1] +   Stopped                    /bin/komprimuj
```

Hlásenie *Stopped* oznamuje pozastavenie procesu . Ako overíme, či proces je naozaj pozastavený?
Zadáme

```
[root@obelix root] # ps –u root –o pid, s, cmd
```

a dostaneme výpis:

```
  PID  S  CMD
13502  S  -bash
14159  T  /bin/komprimuj
14187  R  ps –u root –o pid, s, cmd
```

Písmeno **T** symbolizuje pozastavený proces.
(Pozastavený proces nie je zaraďovaný plánovačom medzi vykonávané procesy. Nie je mu prideľovaný strojový čas a tým pádom procesor nevykonáva žiadnu jeho inštrukciu.
Pozor! Pozastavený proces neznamená zrušený proces! Pozastavený proces nestráca svoje dosiahnuté výsledky, má ich uložené a bude v práci pokračovať, ak bude znovu spustený).
Presun pozastaveného procesu na pozadie zabezpečí príkaz **bg**. Parametrom príkazu *bg* je číslo – identifikátor procesu v rámci shellu (teda to v hranatých zátvorkách), v našom prípade číslo 1, ktorému predchádza znak %.

Zadajme príkaz:

```
[root@obelix root] # bg %1
```

a dostaneme hlásenie:

```
[1] +    /bin/komprimuj &
```

A to už poznáme, nie?

Overme si znova výpisom procesov status procesu komprimuj:

```
[root@obelix root] # ps –u root –o pid, s, cmd
```

a na výpise uvidíme:

```
  PID  S  CMD
13502  S  -bash
14159  R  /bin/komprimuj
14187  R  ps –u root –o pid, s, cmd
```

Písmeno **R** symbolizuje bežiaci proces. Musíme si uvedomiť, že príkazu *bg* nie je možné zadať ako parameter číslo PID. To znamená, že príkaz môžeme používať na proces, ktorý je spustený z tej istej konzoly či terminálu. Ak nie je u príkazu *bg* použitý žiadny parameter, berie sa posledné použité identifikačné číslo procesu v rámci aktívne shellu.

Presun na popredie
Veľmi často sa stáva, že potrebujeme úlohu bežiacu na pozadí presunúť na popredie, aby sme ju mohli sledovať alebo riadiť.
K presunu na popredie sa používa príkaz **fg**. Ten nemusí mať žiadny parameter, ak pracujeme s posledne spracovávaným procesom. Ak máme procesov viac, ako parameter použijeme číslo procesu v rámci aktívne shellu, predchádzané znakom %.

Zadáme

```
[root@obelix root] # fg %1
```

čím prenesieme program z pozadia na popredie.

Príkaz **fg** má ešte jednu vlastnosť. Presunúť na popredie totiž môžeme nielen procesy bežiace na pozadí, ale aj procesy pozastavené klávesmi *Ctrl-Z*.
Na čo je to dobré?
Niekedy, keď už budeme naozaj veľmi zdatní a budeme spravovať niekoľko linuxových serverov za slušné peniažky na diaľku (Bože daj, aby to už bolo!!!) pekne v teplúčku z nášho domova, budeme používať prostriedky prístupu, ktoré budú mať iba jednu konzolu – teda žiadne Alt – F1 až F6. Budeme si musieť vystačiť s jednou.
A môže nastať situácia, že budeme upravovať niektorý konfiguračný súbor, nech je to napr. *smb.conf*. A zrazu počas konfigurácie sa potrebujeme pozrieť do manuálových stránok na správny zápis určitého parametra, nachádzajúceho sa v súbore *smb.conf*.
Máme dve možnosti:
Prvá - ukončiť editáciu súboru *smb.conf*, uložiť ho na disk, ukončiť editor ***vim***, spustiť manuálové stránky, naštudovať ich, ukončiť a nakoniec znovu spustiť editor ***vim*** a pokračovať v editácii súboru *smb.conf*. (Kto už pracoval s editorom vim, vie o čom rozprávam)
Druhá možnosť – editáciu súboru *smb.conf* neukončíme, len pozastavíme klávesmi *Ctrl-Z*. Nemusíme nič ukladať ani ukončovať editor. Spustíme manuálové stránky, naštudujeme a ukončíme. Potom príkazom **fg** aktivujeme pozastavený proces editácie súboru *smb.conf* a to presne na tom mieste, kde sme ho opustili.
V praxi by to vyzeralo takto:

```
[root@obelix root] # vim /etc/samba/smb.conf     {spustíme editáciu súboru a editujeme}

{stlačíme Ctrl-Z, čím pozastavíme tento proces, na obrazovke sa zjaví hlásenie:}

[1] +   Stopped                 vim /etc/samba/smb.conf

[root@obelix root] # man smb.conf        {spustíme manuálové stránky, naštudujeme a ukončíme}

[root@obelix root] # fg      { aktivujeme proces a pokračujeme v editácii súboru}
```

Ukončenie procesu
Povedali sme si, že stlačením *Ctrl–Z* daný proces pozastavíme. Čo ale v takom prípade, ak chceme daný proces úplne ukončiť? Na to existuje niekoľko dôležitých postupov, ktoré závisia od toho, či je proces bežiaci na popredí alebo na pozadí.

Proces bežiaci na popredí, teda ten, ktorý je aktívny priamo v danej konzole, ukončíme stlačením kláves *Ctrl-C*. Na rozdiel od *Ctrl-Z*, takto ukončený proces nie je možné spustiť tak, aby pokračoval vo svojej činnosti. Proces je ukončený, takže novým spustením začne svoju činnosť znovu od začiatku.

Horšie je to s procesom, ktorý beží na pozadí. Ten nemôžeme len tak jednoducho ukončiť klávesmi *Ctrl-C*. Mohli by sme síce žiadaný proces vytiahnuť na popredie, ale nie vždy sa to podarí. Spravidla potrebujeme ukončiť taký proces, ktorý nevykonáva to, čo by sme od neho očakávali, ale sa niekde „zatúlal“ alebo „zacyklil“. Preto je jednoduchšie taký proces nevyťahovať z pozadia, ale ho priamo *zabiť*! (Strašné slovo, však?)

K zabitiu procesu slúži príkaz – no akože iináč – **kill** (angl. zabiť).
Jeho syntaktický zápis je:

kill {-signál} proces

Poznámka:
V skutočnosti príkaz kill procesy nezabíja. On im len zasiela signály. A práve tieto signály spôsobujú „zabitie“ procesov. Pozri nižšie.

Signál je mechanizmus jadra systému, s ktorého pomocou sú procesy informované o rôznych udalostiach. Signál je jednoznačne určený svojím číslom alebo skratkou. V súčasnosti je v systéme definovaných základných 28 signálov, celkom ich je 64 (ale nám zatiaľ postačí poznať iba pár).
Na príchod signálu môžu procesy reagovať rôznymi spôsobmi. Môžu signál prenechať na spracovanie jadru systému, môžu ho ignorovať alebo ho môžu zachytiť a reagovať naňho nejakou činnosťou.
(Všetky signály ale zachytiť nemožno).
V tabuľke č.1 sú uvedené najzákladnejšie signály:

signál	číslo	význam
SIGHUP	1	odpojenie terminálu
SIGINT	2	prerušenie z klávesnice
SIGQUIT	3	koniec s uložením obrazu pamäti
SIGILL	4	použitá inštrukcia je neznáma pre procesor
SIGTRAP	5	ladiace prerušenie
SIGABRT	6	ukončenie z dôvodu vstupno/výstupnej operácie
SIGFPE	8	chyba aritmetiky s pohyblivou desatinnou čiarkou
SIGKILL	9	okamžité ukončenie procesu
SIGSEGV	11	zlyhanie segmentácie
SIGPIPE	13	pokus o zápis do rúry, ktorú nikto nečíta
SIGALRM	14	vypršanie časového intervalu
SIGTERM	15	ukončenie
SIGUSR1	16	používateľský signál 1
SIGUSR2	17	používateľský signál 2
SIGCHLD	18	zmena stavu synovského procesu
SIGSTOP	23	pozastavenie procesu
SIGCONT	24	pokračovanie v činnosti

V prípade, že je signál ponechaný na spracovanie jadru systému, dochádza spravidla k ukončeniu behu procesu. U niektorých signálov naviac jadro uloží obsah pamäti, v ktorej operoval daný proces na disk a to do súboru s názvom ***core***. Obsah tohto súboru je možné čítať špeciálnymi programami – *debuggermi*. Skúsený programátor vie analýzou súboru *core* zistiť, kde nastala chyba, ktorá spôsobila ukončenie programu. Ale nás nech to zatiaľ netrápi, stačí nám vedieť, že ak niekde na disku objavíme súbor *core*, niektorý program sa ukončil nekorektne. Pre dnešok stačí informácia – súbor *core* zmažme, aby zbytočne nezaberal diskový priestor!
Za parameter *proces* zadáme číslo PID daného procesu alebo *%id* číslo v rámci shellu. Ak chceme killnúť viac procesov naraz, zadáme ich parametre oddelené medzerou.

Použitie príkazu kill a killall
Používatelia môžu procesom zasielať signály pomocou príkazov ***kill*** a ***killall***. Už názov hovorí, že reakcia na takýto signál je očakávané ukončenie (zabitie) procesu. Ale vždy to tak byť nemusí. Napríklad systémové procesy – démoni reagujú na dohodnutý signál tak, že znova načítajú konfiguračné súbory. Tým môžeme upraviť správanie sa démonov bez nutnosti ich reštartovať.
Bežný používateľ môže „killnúť“ len jeho vlastné procesy, root – samozrejme – všetky procesy.
Predstavme si, že sme spustili program **komprimuj** na pozadí:

```
[root@obelix root] # /bin/komprimuj &
```

Na obrazovke sa objavia čísla:

```
[1] 14778
```

Po určitom čase sa program niekde „zasekne“ a nie je možné ho bežným *Ctrl-C* ukončiť. Jednoducho nereaguje na nič. Neostáva nám nič iné, iba ho zabiť – *killnúť*.
Zadáme

```
[root@obelix root] # kill 14778
```

alebo

```
[root@obelix root] # kill %1
```

V tomto prípade sme nezadali žiadny signál – automaticky bol priradený signál **SIGTERM**. Ten je zachytitelný a program naň reaguje tak, že uloží rozpracované dáta na disk a až potom sa ukončí.
Môžeme zadať aj signál **SIGKILL**, ktorý daný proces okamžite ukončí, alebo **SIGSTOP**, ktorý daný proces pozastaví. Ak teda chceme proces okamžite ukončiť, zadáme

```
[root@obelix root] # kill –SIGKILL 14778
```

alebo

```
[root@obelix root] # kill –9 14778
```

kde číslo *9* je identifikátor signálu *SIGKILL* (pozri tabuľku).

Na obrazovke sa objaví hlásenie

```
[1] +    Killed                    /bin/komprimuj
```

Príkaz ***killall*** taktiež zasiela signály, ale procesy nie sú identifikovaní číslom *%id* alebo *PID*, ale svojim menom. Má to svoje výhody a nevýhody – na jednej strane nepotrebujeme vedieť žiadne číslo, na druhej strane môžeme nechcene zastaviť procesy s rovnakým menom. (Hovorili sme si, že niektoré procesy vytvárajú svoje „deti“, a to aj niekoľko. Napr. program ***apache*** vytvára až 10 potomkov, každý z nich obsluhuje požiadavky používateľov, lebo sám by to nezvládol. Všetci majú rovnaké meno, ale líšia sa číslom PID).
Môžeme zadať

```
[root@obelix root] # killall /bin/komprimuj
```

a na obrazovke bude

```
[1] +    Terminated                /bin/komprimuj
```

Pomocou signálu *SIGSTOP* je možné pozastavovať procesy a signálu *SIGCONT* presúvať na pozadie podobne ako s použitím *Ctrl-Z* a príkazu **bg**. Nech má náš program komprimuj PID s hodnotou 14778.
Postup bude takýto:

```
[root@obelix root] # kill –SIGSTOP 14778
```

```
[root@obelix root] # kill –SIGCONT 14778
```

Aká je v tom výhoda?
Spomeňme si! Príkazu **bg** je možné zadať ako parameter číslo procesu v rámci aktuálneho shellu. PID zadať nemôžeme! A nastanú situácie, keď proces bol spustený z iného shellu, z inej konzoly, takže priame riadenie príkazom **bg** nie je možné! Vtedy nám prídu vhod uvedené signály.

Praktický príklad
Na záver využijeme naše znalosti z predchádzajúcich častí, navzájom ich pospájame do funkčného celku a dosiahneme požadovaný výsledok.
Dajme si reálny príklad, na ktorom si predvedieme najbežnejší postup manipulácie s procesmi:
Nech náš server beží určitú dobu a na ňom program – démon, ktorý z určitého dôvodu chceme ukončiť. Ten démon nech sa volá ***apmd***. Taký démon v prostredí Linuxu naozaj existuje. V pravidelných intervaloch kontroluje napájanie z akumulátorov a pri poklese napätia pod hraničnú úroveň vypne celý systém. (Isto nás napadne, že to má význam iba pre notebooky, ale to teraz nie je vôbec dôležité). Z určitých dôvodov chceme tento proces ukončiť.
Najprv zistíme PID číslo procesu *apmd*.
Zadáme príkaz

```
[root@obelix root] # ps ax|grep apm
```

(Príkaz *ps* vypíše všetky procesy v systéme. Tento výpis sme pomocou rúry – *pipe* poslali na vstup príkazu *grep*, ktorý z daného výpisu vyselektuje riadky, v ktorých sa nachádza text „apm“. Výsledok takto zreťazeného príkazu vidíme nižšie:

```
    3 ?        SW     0:00 [kapmd]
  797 ?        S      0:00 /usr/sbin/apmd -p 10 -w 5 -W -P /etc/sysconfig/apm-sc
 2015 pts/0    S      0:00 grep apm
```

Výsledok má tri riadky. Prvý síce obsahuje text „apm“, ale jedná sa o „kapmd“, a to je niečo iné. Tretí riadok je samotný príkaz *grep apm*.
To, čo hľadáme, je na druhom riadku. To je ten hľadaný démon *apmd*. Jeho PID číslo je 797.

Pre ukončenie činnosti démona zadáme príkaz

```
[root@obelix root] # kill 797
```

Proces bol ukončený.

Príkaz *kill*, s parametrom PID môžeme úspešne použiť aj z inej konzoly alebo dokonca pri riadení servera na diaľku, pomocou iných prostriedkov.

Možno si teraz položíme otázku, že načo je to celé dobré. Z týchto niekoľkých dôvodov:
- patrí to k všeobecnému vzdelaniu linuxového správcu
- manipulácia s procesmi je jedna z najčastejších činností správy serveru
- na týchto základoch budeme stavať naše ďalšie vedomosti

Tieto princípy, ktoré sme si dnes ukázali, sú všeobecne platné v celom Linuxe, bez rozdielu distribúcie. Preto, keď zasadneme za ľubovoľný server, po chvíľke rozhliadnutia sme schopní s nim správne narábať.

Práve na týchto základných činnostiach s procesmi je založený štart a ukončenie práce celého systému Linuxu.

Ale o tom nabudúce.

Linux prakticky ako server/ 7.časť

V minulej časti sme si povedali niečo o samotných procesoch, o ich spúšťaní a zastavovaní. Takéto spúšťanie démonov z príkazového riadku a opätovné zastavovanie príkazom *kill* je síce v Linuxe bežné, ale nie je úplne čisté. Prečo?
Viete si predstaviť, že by sme museli všetky služby po zavedení jadra spustiť z príkazového riadku na pozadí a pred ukončením zase všetky *killnúť*?
Preto v Linuxe existuje nástroj, ktorý sám určité služby spustí automaticky pri štarte systému a zároveň automaticky ukončí pri vypínaní systému.

Zavedenie Linuxu
Predpokladám, že ste sa už stretli aj s inými operačnými systémami a tak už vieme, čo si pod pojmom zavedenie operačného systému máme predstaviť.

Poznámka:
Dnes nie je našou úlohou rozobrať si do podrobných detailov spôsob zavedenia – spustenia, alebo iným slovom – nabootovanie systému. Pre dnešnú lekciu stačí toto stručnejšie vyjadrenie.

Po zapnutí sieťového vypínača na zdroji príslušného počítača sa aktivujú najprv všetky hardvérové zložky počítača – ako prvé sa overí správnosť napájacích napätí, potom sa aktivuje krátky program, ktorý sa nachádza v EEPROM pamäti základnej dosky počítača. Ľudovo tomu hovoríme, že sa spustí BIOS. Tento program skontroluje a zaktivuje vloženú grafickú kartu, potom otestuje veľkosť a dostupnosť operačnej pamäte, otestuje pripojené zariadenia, ako sú disketové mechaniky, harddisk alebo céderomka, vložené zvukové alebo sieťové karty a ich konfiguráciu. Potom kontaktuje prvý – zavádzací sektor prvého harddisku. Ak nájde na tomto mieste ďalší kratučký program, ktorému aj hovoríme *zavádzač* alebo *bootloader*, odovzdá mu riadenie plne do jeho rúk. A tu už nastáva čas na softvérové ovplyvňovanie následného priebehu.
Predstavme si teda, že na príslušnom harddisku máme nainštalovaný Linux a na tom malinkom kúsku v zavádzacom sektore leží jeden z jeho zavádzačov – bootloaderov (**lilo** alebo **grub**). Keď sa bootloader aktivuje, jeho prvou úlohou je do operačnej pamäte počítača zaviesť jadro systému. Jadro je pri zavádzaní komprimované. V hlavičke komprimovaného jadra je obsiahnutý špeciálny kód, ktorý prepne procesor počítača do chráneného režimu a komprimované jadro dekomprimuje.
Ak je už jadro plne zavedené do operačnej pamäte, je zahájené jeho spustenie. Musíme si uvedomiť, že jadro je v tejto fáze schopné vykonávať iba tie funkcie, ktoré sú v ňom zabudované. To preto, že ešte neexistuje prístup k disku, kde ležia zvyšné súčasti jadra systému vo forme modulov. Tie teda ešte nie sú dostupné a použiteľné. Nasleduje aktivácia podsystému virtuálnej pamäte. Potom sa vykoná test hardvéru, aby sa určilo, ktoré ovládače zariadení majú byť aktivované a po tomto okamžiku je už jadro schopné pripojiť koreňový súborový systém (toto je podobné tomu, ako keď MS Windows je schopné rozpoznať disk C:). Jadro pripojí koreňový systém s príslušným súborovým systémom (najčastejšie ext3) a spustí proces **init**.

Proces init
Proces *init*, tak ako sme si už povedali minule, je najzákladnejším procesom v celom systéme Linuxu.
A povedali sme si, že je to taký praotec všetkých procesov a preto má číslo PID rovné jednej. Tento program sa nachádza v niektorom z adresárov */etc/*, */bin/*, ale najčastejšie v adresári */sbin/*.
Init je obecný program, ktorého úlohou je vytváranie nových procesov (svojich potomkov) a opakované spúšťanie niektorých programov – procesov pri ich ukončení. Napríklad každá konzola (na klávesnici) je obsluhovaná procesom *mingetty* (v starších systémoch *getty*), ktorý je spúšťaný z procesu *init*. Proces *mingetty* nám umožňuje prihlásenie sa do systému a ak sa zo systému odhlásime, proces *mingetty* sa síce ukončí, ale je procesom *init* nahradený novou inštanciou programu *mingetty*.
Všetko, čo program *init* vykonáva, je riadené súborom ***inittab***. Tento leží v adresári */etc/* a je to klasicky čistý textový súbor, takže ho môžeme nielen preštudovať, ale aj pozmeniť a tým ovládať spôsob štartovania samotného Linuxu.
Ak si spomenieme, my sme už súbor */etc/inittab* používali. To bolo ešte v seriáli *Začíname s Linuxom*, keď sme určovali, či má Linux naštartovať do textového alebo grafického režimu.
Stavu, do akého sa dostane Linux po úplnom nabootovaní hovorievame aj **úroveň behu**.

Úroveň behu
Úroveň behu (angl. *runlevel*) zásadne určuje správanie sa celého systému. Každá úroveň behu je označená číslom od 0 do 6, a zastupuje určitú funkciu. Ak je použitá štandardná úroveň behu – označovaná ako

initdefault, je táto priamo spustená. Ak sa *initdefault* v súbore *inittab* nenachádza, budeme pri boote na úroveň behu dopytovaní.
Môžeme voliť medzi úrovňami behu, ktoré sú uvedené v tabuľke č.1:

Tabuľka č.1 - úrovne behu:

Úroveň	Význam
0	Pozastavenie spustenia systému
1	Jednopoužívateľský režim bez pripojenia siete
2	Multipoužívateľský režim bez NFS
3	Plný multipoužívateľský režim - textový režim
4	Nepoužité
5	Plný multipoužívateľský režim - grafický režim
6	Reštart systému

Po zadaní úrovne behu proces *init* pokračuje v spúšťaní ostatných skriptov, definovaných v súbore *inittab*.

Pozrime sa bližšie na výňatok súboru *inittab*, ktorý sa nachádza na výpise č.2:

```
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Author:       Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#               Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Things to run in every runlevel.
ud::once:/sbin/update

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
```

Ak odhliadneme od prvých niekoľko riadkov, ktoré sú aj tak len komentármi, ako prvý nastavovací prvok súboru je voľba štandardnej úrovne behu – **initdefault**. V našom prípade je definovaná úroveň č.3, takže systém nabehne do multipoužívateľského textového režimu. Keby sme číslicu 3 nahradili číslicou 5, Linux by po boote prešiel do grafického režimu. My už vieme, že úroveň behu 3 je vhodná pre serverový režim, úroveň behu 5 sa viac hodí pre grafické stanice.

Adresáre úrovní behu
Už v úvode sme si povedali, že by nebolo veľmi príjemné, kedy sme mali všetky služby, ktoré po našom linuxovom serveri požadujeme, spúšťať a ukončovať ručne. Ba dokonca, museli by sme si pamätať, ktoré služby je vhodné použiť v danej úrovni behu.
Preto je lepšie, ak to necháme na Linux samotný.
Všetky služby, všetky démoni v Linuxe sa (pri boote a ukončení) spúšťajú a zastavujú pomocou špeciálnych súborov, ktorým sa hovorí **rc skripty**.

Poznámka:
Slovom **skript** *(angl.* script*) sa označuje program, ktorý je tvorený súborom v textovej podobe, teda čistý text (nie skompilovaný do binárnej podoby). Jeho obsahom sú príkazy, parametre, premenné a konštanty určitého programového prostredia. Ak je týmto prostredím príkazový interpreter* shell*, hovoríme o* shellovom skripte *alebo aj* bash skripte*. Pre bližšie pochopenie môžeme povedať, že shell skript je to isté, čo BAT súbor v prostredí MS DOS/Windows. Ak je programovým prostredím iný – skutočný programovací jazyk, hovoríme o skripte v tomto jazyku, napr.* perl skript, python skript *a podobne. Tieto skripty sú analogické súborom, napísaných v jazyku BASIC. Treba si uvedomiť, že tieto skripty sú textové a NIKDY nenadobudnú binárnu podobu. Možno sa to bude zdať niekomu na závadu, ale opak je pravdou. Kedykoľvek môžeme do týchto súborov zasiahnúť, pozmeniť alebo upraviť, lebo sú v otvorenej podobe. A to je plne v súlade s nosnou myšlienkou Linuxu.*

rc skript je teda (zase, akože iná č **J**) program *shellu*, ktorý slúži na riadenie požadovanej služby.
rc skripty boli zavedené z dôvodu veľkého množstva služieb, ktoré je nutné spravovať.
V Linuxe sa nachádzajú tieto rc skripty:
- **rc.sysinit**
- **rc**
- **rc skripty** v jednotlivých úrovniach behu
- **rc.local**

Skript *rc.sysinit*
Pozrime sa ešte raz na výpis súboru *inittab*. Hneď za voľbou štandardnej úrovne behu *initdefault* sa nachádza sekcia *System initialization*. Všimnime si, že sa odkazuje na súbor *rc.sysinit*, ktorý sa nachádza v adresári */etc/rc.d/*.
rc.sysinit je prvý rc skript, ktorý sa procesom *init* spustí. Tento súbor je jednoduchý shell skript, obsahujúci príkazy pre nastavenie základnej systémovej konfigurácie – aktivuje sa odkladací mechanizmus tzv. *swap*, kontrolujú a pripojujú sa ostatné súborové systémy, synchronizujú sa systémové hodiny s časom v CMOS pamäti počítača a podobne. (Ak sme nedočkaví, môžeme sa na neho pozrieť a preštudovať, ale to dnes nie je našou úlohou).

Skript *rc*
Po spustení a vykonaní skriptu *rc.sysinit* sa spustí skript *rc*. Ten sa nachádza v adresári */etc/rc.d/*. Všimnime si (pohľadom do súboru *inittab*), že sa tento spúšťa pri ľubovolnej úrovni behu, pričom zodpovedajúca úroveň je jeho argumentom. Skript *rc* je generický spúšťací skript, ktorý spúšťa iné skripty, platné v danej úrovni behu. Každá úroveň behu má samostatný podadresár, umiestnený v adresári **/etc/rc.d/rc*X*.d**, kde je ***X*** je číslo úrovne behu (napr. */etc/rc.d/rc0.d*, */etc/rc.d/rc1.d*, */etc/rc.d/rc2.d*, */etc/rc.d/rc3.d*....).
Práve v stanovenom adresári sú uložené skripty, ktoré sa v danej úrovni behu budú spúšťať.
Ak je napríklad nastavená štandardná úroveň behu na č.3 (*initdefault* =3), argumentom skriptu *rc* je číslo 3, a tak skript *rc* sa pozrie do adresára */etc/rc.d/rc3.d/* a spustí všetky skripty, ktoré sa v danom adresári nachádzajú. Ak teda chceme ovplyvniť, ktoré služby sa budú v tejto úrovni behu spúšťať, pridáme do tohto adresára skript, ktorý bude ovládať žiadanú službu.

rc skripty v jednotlivých úrovniach behu
Pozrime sa teraz, ako to vyzerá v jednotlivých podadresároch úrovní behu. Pre skutočný príklad budeme uvažovať, že používame úroveň behu číslo 3, a tak sa zameriame na adresár */etc/rc.d/rc3.d*.

Pozrime sa na obsah uvedeného adresára. Môžeme použiť Midnight Commander alebo priamo v riadku príkaz:

```
[root@obelisk root]#  ls /etc/rc.d/rc3.d
```

Na výpise č.3 uvidíme tento obsah uvedeného adresára:

```
K05innd         K30sendmail     K61ldap         K84ripngd               S25netfs
K09junkbuster   K34yppasswdd    K65identd       K85zebra                S26apmd
K12mysqld       K35atalk        K65kadmin       K90isicom               S28autofs
K15httpd        K35dhcpd        K65kprop        K99microcode_ctl        S55sshd
K15postgresql   K35vncserver    K65krb5kdc      S05kudzu                S56rawdevices
K16rarpd        K40mars-nwe     K65krb524       S06reconfig             S56xinetd
K20bootparamd   K45arpwatch     K70aep1000      S08ipchains             S81smb
K20iscsi        K45named        K70bcm5820      S08iptables             S85gpm
K20netdump-serverK45smartd      K74ntpd         S08ip6tables            S90crond
K20nfs          K45winbind      K74ups          S09isdn                 S90xfs
K20rstatd       K46radvd        K74ypserv       S10network              S95anacron
K20rusersd      K50netdump      K74ypxfrd       S12syslog               S95atd
K20rwalld       K50snmpd        K75gated        S13portmap              S97rhnsd
K20rwhod        K50snmptrapd    K84bgpd         S14nfslock              S98wine
K24irda         K50tux          K84ospfd        S17keytable             S99cups
K25squid        K54pxe          K84ospf6d       S20random               S99local
K28amd          K55routed       K84ripd         S24pcmcia
```

Všimnime si, že sa tu nachádzajú súbory – rc skripty, ktoré začínajú iba na písmeno **K** alebo **S**. Za písmenom nasleduje dvojciferné číslo a potom slovo, tvoriace názov služby. Písmeno **S** značí, že skript spustí danú službu, písmeno **K** značí, že skript danú službu ukončí.

Pozor!
Obe písmená musia byť napísané VEĽKÝMI písmenami!

Poradie spúšťania a zastavovania je určené vyššie spomínaným dvojciferným číslom. Skripty s nižším číslom sú spúšťané skôr ako skripty s vyššími číslami. Na poradí spustenia skriptov niekedy naozaj záleží. Je zrejmé, že sieť musí byť nastavená skôr ako sieťové súborové systémy, teda skript */etc/rc.d/rc3.d/S10network* bude musieť byť spustený pred skriptom */etc/rc.d/rc3.d/S25netfs*.

Ak som pred chvíľkou povedal, že v adresári */etc/rc.d/rc3.d/* sa nachádzajú skripty, tak som trošku klamal.
V tomto adresári sa nenachádzajú skripty, ale **symbolické odkazy** - linky na skutočné skripty, ktoré sa nachádzajú v adresári */etc/rc.d/init.d/* (pod symbolickými odkazmi (linkami) si môžeme predstaviť niečo ako zástupcov v prostredí MS Windows).
Prečo je to tak?
Predstavme si, že by sme museli do každého adresára danej úrovne behu (*/etc/rc.d/rc**X**.d*) prekopírovať skript, ktorý chceme v tejto úrovni behu spúšťať. Tak by sa mohlo stať, že ten istý súbor by sme mali v 7 adresároch (od */etc/rc.d/rc0.d* po */etc/rc.d/rc6.d*). Ak by sme ho časom pozmenili, museli by sme to urobiť vo všetkých siedmych adresároch. Po prvé by to bolo zbytočne pracné a po druhé by sme mohli vniesť niekoľko nerovnakých chýb.
Preto je výhodnejšie, aby sa iba na jednom mieste – a to v adresári */etc/rc.d/init.d* nachádzali skutočné skripty. A do adresárov jednotlivých úrovní behu (napr. */etc/rc.d/rc3.d/*) sú umiestnené iba symbolické odkazy na skutočný súbor.
Že je to skutočne tak, môžeme sa presvedčiť príkazom

```
[root@obelisk root]#  ls –la /etc/rc.d/rc3.d
```

Na výpise č.4 vidíme jednotlivé odkazy na skutočné súbory (symbolický odkaz je prezentovaný znakom ->):

```
drwxr-xr-x    2 root     root         4096 dec 14 16:12 .
drwxr-xr-x   10 root     root         4096 dec 14 14:57 ..
lrwxrwxrwx    1 root     root           14 okt 25 13:43 K05innd -> ../init.d/innd
lrwxrwxrwx    1 root     root           16 okt 25 13:43 K12mysqld -> ../init.d/mysqld
lrwxrwxrwx    1 root     root           15 okt 25 13:43 K15httpd -> ../init.d/httpd

--- skrátené ---

lrwxrwxrwx    1 root     root           15 okt 25 13:43 S05kudzu -> ../init.d/kudzu
lrwxrwxrwx    1 root     root           17 okt 25 13:43 S10network -> ../init.d/network
lrwxrwxrwx    1 root     root           16 okt 25 13:43 S12syslog -> ../init.d/syslog
```

```
lrwxrwxrwx    1 root     root           17 okt 25 13:43 S13portmap -> ../init.d/portmap
lrwxrwxrwx    1 root     root           17 okt 25 13:43 S14nfslock -> ../init.d/nfslock
lrwxrwxrwx    1 root     root           18 okt 25 13:43 S17keytable -> ../init.d/keytable
lrwxrwxrwx    1 root     root           16 okt 25 13:43 S20random -> ../init.d/random
lrwxrwxrwx    1 root     root           16 okt 25 13:43 S24pcmcia -> ../init.d/pcmcia
lrwxrwxrwx    1 root     root           15 okt 25 13:43 S25netfs -> ../init.d/netfs
lrwxrwxrwx    1 root     root           14 okt 25 13:43 S26apmd -> ../init.d/apmd
lrwxrwxrwx    1 root     root           16 okt 25 13:43 S28autofs -> ../init.d/autofs
lrwxrwxrwx    1 root     root           13 okt 25 13:43 S85gpm -> ../init.d/gpm
lrwxrwxrwx    1 root     root           14 dec 14 15:46 S98wine -> ../init.d/wine
lrwxrwxrwx    1 root     root           14 nov 17 15:54 S99cups -> ../init.d/cups
lrwxrwxrwx    1 root     root           11 dec 14 14:57 S99local -> ../rc.local
```

Všimnime si, že skripty, nachádzajúce sa v adresári */etc/rc.d/init.d/* nemajú meno totožné so symbolickou linkou v adresári */etc/rc.d/rc3.d/*.
A práve skripty v adresári */etc/rc.d/init.d/* spravujú vlastný proces spúšťania a zastavovania jednotlivých služieb. Ak skript */etc/rc.d/rc* prechádza adresárom zvolenej úrovne behu – napr. */etc/rc.d/rc3.d/*, volá (aktivuje = zavádza do operačnej pamäte a následne vykonáva) skripty podľa číselného poradia. Najprv volá skripty začínajúce písmenom **K**, potom volá skripty začínajúce písmenom **S**. Ak začína skript písmenom **K**, je volaný s parametrom **stop**, ak začína písmenom **S**, je volaný s parametrom **start**.
Pozrime sa na výpis č.4 ešte raz:
Ak je volaný skript ***K12mysql***, je to v skutočnosti tak, ako keby sme z príkazového riadku zadaný príkaz:

```
/etc/rc.d/init.d/mysqld stop
```

Ak je volaný skript ***S85gpm***, bol by v skutočnosti zadaný príkaz:

```
/etc/rc.d/init.d/gpm start
```

Na výpise č.5 je obsah skriptu **gpm**:

```
#!/bin/bash
#
# chkconfig: 2345 85 15
# description: GPM adds mouse support to text-based Linux applications such \
#              the Midnight Commander. Is also allows mouse-based console \
#              cut-and-paste operations, and includes support for pop-up \
#              menus on the console.
# processname: gpm
# pidfile: /var/run/gpm.pid
# config: /etc/sysconfig/mouse

# source function library
. /etc/init.d/functions
[ -e /etc/sysconfig/gpm ] && . /etc/sysconfig/gpm

MOUSECFG=/etc/sysconfig/mouse

RETVAL=0

start() {
        echo -n $"Starting console mouse services: "
        if [ -f "$MOUSECFG" ]; then
                . "$MOUSECFG"
        else
                echo $"(no mouse is configured)"
                exit 0
        fi

        if [ "$MOUSETYPE" = "none" ]; then
                echo $"(no mouse is configured)"
                exit 0
        fi


        if [ "$MOUSETYPE" = "Microsoft" ]; then
```

```
                MOUSETYPE=ms
        fi

        if [ -z "$DEVICE" ]; then
           DEVICE="/dev/mouse"
        fi

        if [ -n "$MOUSETYPE" ]; then
                daemon gpm $OPTIONS -t $MOUSETYPE -m $DEVICE
        else
                daemon gpm $OPTIONS -m $DEVICE
        fi
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/gpm
}

stop() {
        echo -n $"Shutting down console mouse services: "
        killproc gpm
        RETVAL=$?

        echo
        [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/gpm
}

case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  restart|reload)
        stop
        start
        RETVAL=$?
        ;;
  condrestart)
        if [ -f /var/lock/subsys/gpm ]; then
           stop
           start
           RETVAL=$?
        fi
        ;;
  status)
        status gpm
        RETVAL=$?
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart|condrestart|status}"
        exit 1
esac

exit $RETVAL
```

Keďže väčšina skriptov je si veľmi podobná, pozrime sa na chvíľu na tento súbor. Nebojte sa, nepotrebujeme teraz zvlášť veľké programátorské skúsenosti, aby sme úplne porozumeli tomuto skriptu. Čo je podstatné, aby sme si všimli tieto dôležité veci:

- Úplne prvý riadok je veľmi dôležitý!
  Text **#!/bin/bash** – značí, že sa jedná o skript, ktorého obsah bude vykonávaný príkazovým interpreterom *shell* – v našom prípade *bash*.
- Riadky označené **#** (mreža, angl. *hash*) označujú poznámky. Popisujú použitie skriptu, meno jeho autora a podobne. Výnimku tvorí už spomenutý prvý riadok (ktorý je tiež „zamrežovaný“)

- Prvé naozajstné programové riadky sú definície požadovaných premenných, (napríklad *MOUSECFG=/etc/sysconfig/mouse*)
- Nasledujú definície jednotlivých argumentov, spravidla **start** a **stop**
  Všimnime si, že v argumente **start** je výkonným riadkom príkaz ***daemon gpm $OPTIONS -m $DEVICE***. Tento príkaz volá program *gpm*, ktorý sa nachádza v adresári */usr/sbin/*. A to je už samotný skompilovaný program v jazyku C, ktorému hovoríme aj **binárka**. Tento program už nie je bežne čitateľný, a to že sa jedná o *binárku* spoznáme podľa prvých štyroch znakov obsahu súboru. Tie znaky sú **.ELF** (ale nebojte sa, že by sme nemali možnosť ho zmeniť. Predsa je dostupný jeho zdrojový kód, ktorý upravíme a znovu preložíme).
  U argumentu **stop** je výkonným riadkom zápis ***killproc gpm***, ktorý korektným spôsobom *killne* daný binárny program *gpm*, čím ho korektne ukončí.
- Nasleduje hlavná časť skriptu, ktorá je uložená medzi príkazmi **case "$1" in** a **esac**. Pre zaujímavosť si všimnime, ako je vytvorený argument **restart** alebo **reload**. (Úplne jednoducho – najprv sa využije argument **stop** a potom **start**!)

Linux prakticky ako server/ 8.časť

Keďže sme v minulej časti nepovedali o ovládaní procesov v Linuxe všetko, dnes budeme v tejto téme pokračovať. Pre občerstvenie našej pamäte si predchádzajúcu tématiku zhrňme:

Zhrnutie
Po tom, čo sme si to teoreticky vysvetlili, pre lepšie ujasnenie si to celé ešte raz zhrňme:
Po zapnutí vypínača na počítači dôjde k privedeniu napájacích napätí na komponenty počítača. Spustí sa program BIOS, ktorý sa nachádza v pamäti na základnej doske počítača a jeho úlohou je otestovanie funkčnosti a dostupnosti počítačových súčastí. Potom je práca odovzdaná programu, ktorý sa nachádza v zavádzacom (bootovacom) sektore harddisku (diskety, cédečka...). Boot loader zavedie z disku do operačnej pamäte počítača komprimované jadro, kde sa následne dekomprimuje, otestuje dostupnosť hardvéru z pohľadu Linuxu, pripojí koreňový súborový systém a spustí prvý proces *init*.
Ten sa riadi konfiguráciou, ktorá je definovaná v súbore *inittab*. Ako prvé je priradenie štandardnej úrovne behu systému (*initdefault*), potom sa spustí rc skript s názvom *rc.sysinit*, ktorý vykoná určité softvérové nastavenia systému a odovzdá prácu ďalšiemu skriptu s názvom *rc*. Ten podľa nastavenia úrovne behu (nech je to číslo 3) prejde do príslušného adresára */etc/rc.d/rc3.d/*, kde zavolá každý skript, ktorý sa v tomto adresári nachádza a začína na písmená **K** a **S**.
Najprv začne so skriptami, začínajúcimi na písmeno *K*, potom pokračuje skriptami, začínajúcimi na písmeno *S*. Zároveň záleží na poradí, ako prvé volá skripty s nižším číslom, potom skripty s vyšším číslom.
Keďže sa však jedná len o symbolické odkazy na skutočné skripty, ktoré sa nachádzajú v adresári */etc/rc.d/init.d/*, sú skutočné skripty volané s príslušným argumentom. Ak začína odkaz na písmeno *K*, volá sa skript s argumentom **stop**, ak začína odkaz na písmeno *S*, je volaný skript s agrumentom **start**.
Agrument *start* v danom skripte v adresári */etc/rc.d/init.d/* spôsobí volanie binárky, ktorá zabezpečuje požadovanú službu.
Argument *stop* v danom skripte v adresári */etc/rc.d/init.d/* spôsobí korektné ukončenie binárky pomocou príkazu *kill*.
Ako posledný sa pustí odkaz *S99local*, ktorý zavolá ďalší skript s menom *rc.local* z adresára */etc/rc.d/*. Do tohto súboru je nám umožnené dopĺňať naše vlastné nastavenia alebo príkazy (ako uvidíme v neskorších lekciách). Tým je bootovanie systému Linux ukončené.

Ukončenie činnosti Linuxu
Vypínanie a ukončenie činnosti Linuxu je vykonané obdobným spôsobom. Už vieme, že ak chceme Linux ukončiť, zadáme príkaz ***shutdown***. Ten zavolá súbor *init*, ktorému predá parameter 0 alebo 6 (podľa parametru u súboru shutdown, h – halt = 0, r - reboot =6), ktorý značí úroveň behu. Keď sa pozrieme do tabuľky č.1, vidíme, že úrovniam behu 0 a 6 zodpovedá zastavenie (halt) alebo znovu-spustenie (reboot) systému. Vtedy už dobre známy skript s názvom *rc* vstúpi do príslušného adresára /etc/rc.d/rc0.d/ alebo /etc/rc.d/rc6.d/, kde následne zavolá všetky uložené odkazy na skripty. Ak sa dobre pozrieme do uvedených adresárov, vidíme, že sa tu nachádzajú hlavne skripty – odkazy, začínajúce na písmeno *K*, ale len dva na písmeno *S*. Jeden je *S00killall*, ktorý ukončí všetky ostatné procesy. Druhý je *S01halt* (u runlevelu =0) alebo *S01reboot* (u runlevelu = 6), ktorý zastaví alebo rebootne celý systém.

Využitie rc skriptov
Teória je jedna vec, ale ako môžeme vyššie uvedené znalosti využiť v našej praxi?
Dvoma spôsobmi:
- ovplyvnenie štartu systému
- riadenie jednotlivých služieb počas behu systému

Ovplyvnenie štartu systému
Ovplyvňovanie samotného štartu systému je veľmi jednoduché. Ak chceme, aby sa niektorá služba automaticky spúšťala v danej úrovni behu, stačí ak zaistíme, aby sa v určenom adresári príslušnej úrovne behu nachádzal skript (lepšie povedané symbolický odkaz), začínajúci písmenom **S** a príslušným číslom, ktorý bude odkazovať na konkrétny skript do adresára */etc/rc.d/init.d/*.
Ak nechceme, aby sa konkrétna služba v danej úrovni behu spúšťala, symbolický odkaz zmažeme alebo premenujeme tak, že zameníme písmeno **S** (veľké S) za písmeno **s** (malé s - musíme si uvedomiť, že služba sa spúšťa iba vtedy ak začína na veľké písmeno *S*). Druhá možnosť je výhodnejšia, pretože nám umožňuje kedykoľvek sa vrátiť k aktivácii služby jednoduchým zmenením prvého písmena názvu odkazu.

Nástroje na manipuláciu so službami
Musím uznať, že ručné editovanie súborov v príslušných adresárových štruktúrach jednotlivých úrovní behu je trochu nemotorné.
Preto väčšina distribúcií obsahuje nástroje, ktoré za nás upravujú jednotlivé skripty – odkazy v adresároch úrovní behu.
Najznámejšie sú tieto nastroje:

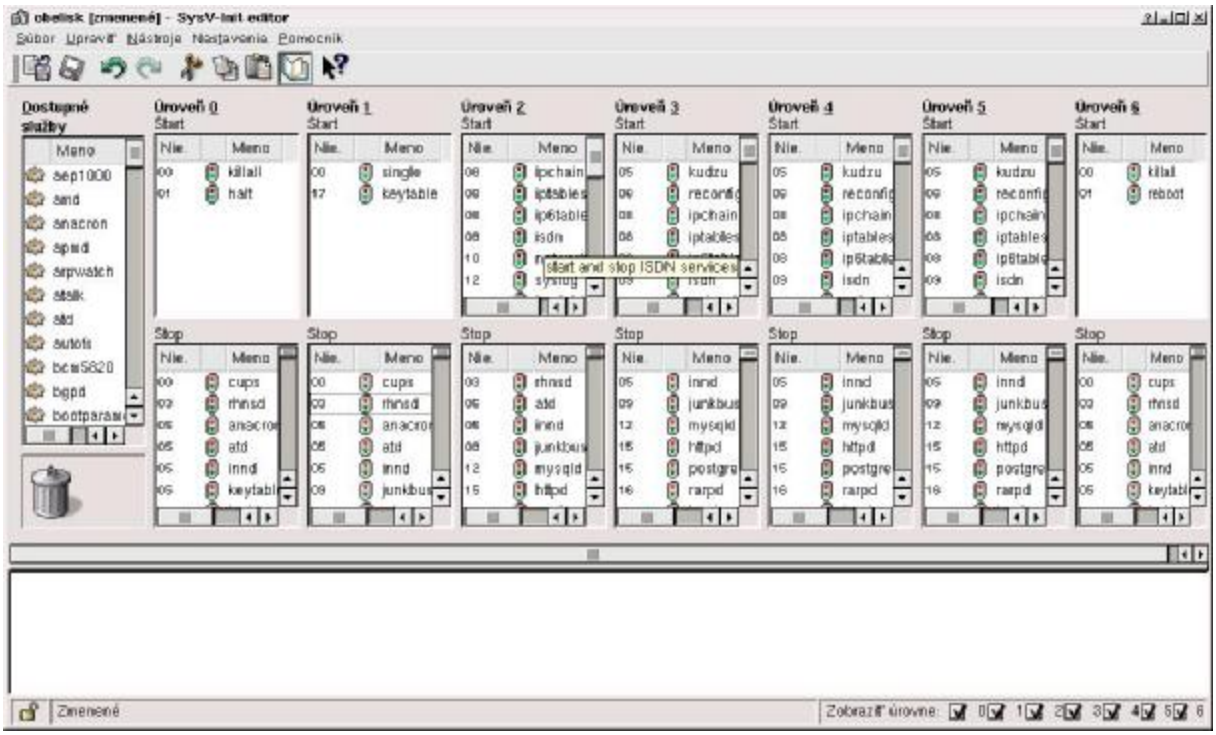
- **ntsysv** – textový nástroj
- **ksysv** – grafický nástroj
- **chkconfig** – textová utilita

ntsysv
Tento program sa nachádza hlavne v distribúciách RedHat a Fedora a od ich odvodených klonov. Program spustíme z konzoly zadaním príkazu **ntsysv**. Vzhľad nástroja *ntsysv* je na obrázku č.6:

Tento nástroj je veľmi intuitívny. Kurzorovými šípkami vyberieme požadovanú službu a stlačením medzerníka určujeme (de)aktivovanie služby počas bootu systému. Či bude pri boote služba aktivovaná, je symbolizované hviezdičkou (*) u názvu služby. Naopak, ak chceme službu deaktivovať, stlačením medzerníka hviezdičku zmažeme.
Nakoniec presunieme tabulátorom kurzor na tlačidlo OK a stlačením klávesu Enter zmeny uložíme.

Pozor!
Musíme si uvedomiť, že zmeny sa prejavia až pri reštarte systému, kedy dôjde k znovunačítaniu adresára príslušnej úrovne behu.

ksysv
Nástroj *ksysv* je grafickou obdobou utility *ntsysv*, preto ho musíme spustiť len v prostredí X Window – obr.č.7:



Ani tento nástroj nie je zložitý. Ak chceme niektorú službu v danom runleveli aktivovať alebo deaktivovať, jednoducho ju prenesieme myškou metódou *drag and drop* medzi oknami *Štart* a *Stop* Aj tu platí, že príslušné zmeny sa prejavia až v novom štarte systému.

chkconfig

Všimnime si, že nástroj *ksysv* na rozdiel od nástroja *ntsysv* umožňuje nielen danú službu zapnúť alebo vypnúť, ale môže presnejšie stanoviť, v ktorom runleveli danú službu chceme spúšťať. Nie vždy je na serveri dostupný nástroj *ksysv*. Preto v textovej konzole existuje jeden veľmi mocný nástroj, ktorý sa nazýva **chkconfig**. Nemá síce žiadne okienka, menu či kurzor, zato dokáže toho neskonale viac. Práve rôznymi parametrami tohto príkazu môžeme riadiť náš server.

Syntaktický zápis príkazu **chkconfig** je:

chkconfig *--parameter názov_služby*

kde za *parameter* môžeme zadať :

- **list** – vypíše aktuálny stav danej služby v jednotlivých úrovniach behu
- **add** – pridá danú službu do menežovania príkazom chkconfig
- **del** – výjme danú službu z menežovania príkazom chkconfig
- **level** – nastaví v konkrétnej úrovni behu danú službu

Poznámka:
Každý parameter musí začínať dvomi znakmi -- (mínus mínus)!
Názov_služby nie je povinný, vtedy sa vypíše nastavenie všetkých služieb v systéme.

Ak zadáme príkaz

```
[root@obelisk root]#  chkconfig --list gpm
```

dostaneme výpis podobný tomu na výpise č.8:

```
gpm             0:vypnuté   1:vypnuté   2:zapnuté   3:zapnuté   4:zapnuté
      5:zapnuté   6:vypnuté
```

Z uvedeného výpisu vidíme, že služba ***gpm*** (pridávajúca myši nové možnosti, ako je *cut and paste* a podobne) je aktívna v úrovniach behu č. 1, 2, 3, 4 a 5, a neaktívna v úrovni behu č.1 a 6.

Takto to Linux prednastavil už pri inštalácii.
Môže sa však stať, že myš v 3.runleveli, ktorý zodpovedá serverovému uplatneniu, nechceme používať. Preto potrebujeme dosiahnúť, aby sa z adresára */etc/rc.d/rc3.d/* odstránila (vypla) symbolická linka na súbor */etc/init.d/gpm*.
Preto zadáme:

```
[root@obelisk root]#  chkconfig --level 3 gpm off
```

Pohľadom do adresára */etc/rc.d/rc3.d/* zistíme, že sa skutočne stratila symbolická linka **S85gpm**.

Ak zadáme príkaz:

```
[root@obelisk root]#  chkconfig --level 3 gpm on
```

linka **S85gpm** sa v uvedenom adresári znovu objaví. Takto zabezpečíme, že po reštarte sa v tejto úrovni behu daná služba (ne)spustí.

Ako však príkaz *chkconfig* vie, aké číslo linky má priradiť?

Príkaz *chkconfig* číta komentáre uvedeného súboru v adresári */etc/init.d/*. Pozrime sa ešte raz na začiatok súboru *gpm* – výpis č.9:

```
#!/bin/bash
#
# chkconfig: 2345 85 15
# description: GPM adds mouse support to text-based Linux applications such \
#              the Midnight Commander. Is also allows mouse-based console \
#              cut-and-paste operations, and includes support for pop-up \
#              menus on the console.
# processname: gpm
```

Na treťom riadku je komentár - **# chkconfig: 2345 85 15**
Prvá skupina číslic (12345) reprezentuje jednotlivé úrovne behu, v ktorých bude daná služba aktívna, teda v 2., 3., 4., a 5.úrovni. Druhá skupina (85) je číslo linky s príznakom **S** v uvedených úrovniach behu. V tomto prípade to bude *S85* a meno súboru, teda *S85gpm* vo všetkých vyššie spomínaných úrovniach behu.
Posledná skupina číslic (15) udáva číslo, ktoré bude mať linka s príznakom **K** v ostatných – chýbajúcich úrovniach behu. V tomto prípade to bude *K15gpm* a táto linka bude v 1. a 6.úrovni behu. Pohľadom do jednotlivých adresárov sa môžeme presvedčiť, že je to tak.
Tieto čísla zadáva autor daného skriptu pri jeho tvorbe. Ak máme pocit, že z nejakého dôvodu potrebujeme tieto čísla zmeniť, môžeme tak urobiť. Nesmieme zabudnúť, že len zmena číslic v hlavičke súboru nepomôže, musíme vykonať konkrétne postupy s príkazom *chkconfig*. Ako a čo všetko sa robí sa naučíme ďalej.

Riadenie jednotlivých služieb počas behu systému
Veľmi často sa však stáva, že my – ako správcovia systému – potrebujeme spustiť alebo zastaviť určitú službu bez reštartu celého systému, teda počas jeho samotného behu (spomeňme si na teorém, že linuxový server sa reštartuje iba dva razy do roka **J**).
Ako na to?
Adresárovú štruktúru jednotlivých úrovní behu, bohužiaľ, nemôžeme využiť. Je zrejmé, že preto, lebo jednotlivé odkazy (na skripty) spúšťa iba skript s názvom *rc*.
My však dokážeme využiť priamo skripty v adresári /etc/rc.d/init.d/. Jednoducho zavoláme daný skript, ktorému ako parameter predáme niektorý argument – najčastejšie **start**, **stop**, **restart**, **reload**, **status** a podobne. Aké sú možnosti argumentov v danom skripte zistíme tak, že skript spustíme bez parametrov. Každý skript je napísaný tak, aby nám ponúkol svoje možnosti.
Napr. zadajme (a na obrazovke uvidíme):

```
[root@obelisk root]#  /etc/rc.d/init.d/smb
Usage: /etc/rc.d/init.d/smb {start|stop|restart|reload|status|condrestart}
```

Ak chceme službu smb spustiť, na príkazovom riadku zadáme príkaz:

```
[root@obelisk root]#  /etc/rc.d/init.d/smb start
```

Na obrazovke sa objaví hlásenie:

```
Starting SMB services:                    OK
Starting NMB services:                    OK
```

Ak chceme z určitého dôvodu službu zastaviť, použijeme príkaz:

```
[root@obelisk root]#  /etc/rc.d/init.d/smb stop
```

a na obrazovke objaví:

```
Shutting down SMB services:               OK
Shutting down NMB services:               OK
```

Veľmi často v našej budúcej praxi budeme potrebovať službu zastaviť a znova spustiť, preto použijeme argument **restart**. A ešte častejšie nastane prípad, keď niečo zmeníme v príslušnom konfiguračnom súbore, ale nechceme celú službu reštartovať, aby sme nezrušili už nadviazané spojenia s klientmi. Vtedy stačí, aby táto služba iba znovu načítala svoj konfiguračný súbor.
Vtedy zadáme príkaz:

```
[root@obelisk root]#  /etc/rc.d/init.d/smb reload
```

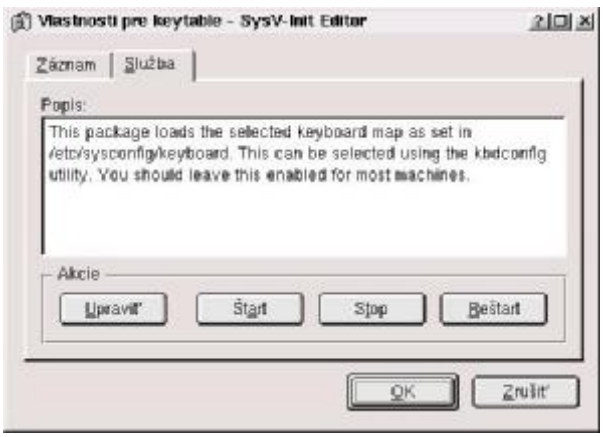
Ak sa chceme dozvedieť, v akom stave sa nachádza daná služba, použijeme argument **status**.

Ešte jednoduchšie – utilita service
Aby sme nemuseli vypisovať plnú cestu do adresára */etc/rc.d/init.d/*, existuje v niektorých distribúciách Linuxu malá utilitka s názvom **service**. Jej použitie je veľmi podobné:

```
[root@obelisk root]#  service smb start
```

je adekvátny príkazu /etc/rc.d/init.d/smb start.

Spustenie, zastavovanie alebo reštart určitej služby môžeme vykonávať aj v prostredí X Window. Využijeme na to už spomínaný nástroj ***ksysv*** (obr.č.7).
Samotnú službu môžeme ovládať tak, že na ňu klikneme myškou a stlačíme pravé tlačidlo myši. Objaví sa okno vlastností danej služby – obr.č.10:



Na záložke *Služba* môžeme pomocou dolných štyroch tlačidiel službu ovládať – upraviť, spustiť, zastaviť alebo reštartnúť. Dosiahneme podobného efektu, ako keby sme používali príkazový riadok a utilitu *service*.

Praktická ukážka
Na malej ukážke si precvičíme vyššie popísané teoretické znalosti v praxi. Ukážeme si vytvorenie svojho vlastného skriptu a zabezpečenie jeho spúšťania v jednotlivých úrovniach behu pri štarte systému a počas behu systému.
Predstavme si, že na našom milom serveríku chceme mať nainštalovanú wi-fi kartu a chceme zabezpečiť, aby sa po štarte počítača táto karta aktivovala a pri vypnutí deaktivovala. Preto si vytvorme skript s názvom **wifi**, ktorý všetko za nás automaticky zabezpečí.

Poznámka:
Pre tých, čo nevedia, čo je to wi-fi karta, tak len tak na okraj: je to karta, ktorá zabezpečuje spojenie počítačov pomocou bezdrôtového – teda rádiového prenosu. Dnes je táto technológia dostupná aj normálnym smrteľníkom. Cena takej karty sa pohybuje v reláciách od 1500 Sk do 3500 Sk. Záleží na type a výrobcovi. Je jasné, že pre spojenie musíme mať minimálne dve wi-fi karty. Nie je podmienkou, že musia byť rovnakého typu alebo od rovnakého výrobcu. Ale o wi-fi inokedy.

Ďakujem týmto firme **Floppy** v Novom Meste nad Váhom za bezplatné poskytnutie rôznych wi-fi kariet za účelom testovania ich funkčnosti pod Linuxom. Skoro všetky karty bolo možné pod Linuxom sprevádzkovať.

Vráťme sa k nášmu skriptu. Jeho obsah vidíme na výpise č.11:

```
#!/bin/bash
#
# chkconfig: 2345 13 86
# description: Spustanie wifi na XI-626 wlan0
#
# mior 02/2004

# source function library
. /etc/rc.d/init.d/functions

RETVAL=0

case "$1" in
  start)
      echo -n "Startovanie wifi: "
      modprobe hostap_pci
      ifconfig wlan0 up
      iwconfig wlan0 mode master
      ifconfig wlan0 inet 192.168.100.1 netmask 255.255.255.0
      iwconfig wlan0 essid "skuska"
      iwconfig wlan0 txpower 1
      RETVAL=$?
      [ $RETVAL -eq 0 ] && touch /var/lock/subsys/wifi
      echo
      ;;
  stop)
      echo -n "Vypinanie wifi: "
      ifconfig wlan0 down
      rmmod -r hostap_pci
      RETVAL=$?
      [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/wifi
      echo
      ;;
  restart)
        $0 stop
        $0 start
      RETVAL=$?
        ;;
  *)
```

```
      echo "Pouzitie: wifi {start|stop|restart}"
      exit 1
esac

exit $RETVAL
```

Nie je dôležité, aby sme teraz porozumeli celému obsahu skriptu. Stačí, ak spoznávame časti **start**, **stop** a **restart**. Teraz je však dôležitá hlavička skriptu. Všimnime si, že za povinným riadkom ***#!/bin/bash*** (dúfam, že už všetci vieme, čo tento riadok znamená!) sa nachádzajú dva veľmi dôležité riadky:

```
# chkconfig: 2345 13 86
# description: Spustanie wifi na XI-626 wlan0
```

Tieto dva riadky sú – ak avšak chceme využívať príkaz *chkconfig* – povinné!
Prvý už poznáme. Hovorí, že v úrovni behu č.2, 3, 4 a 5 bude vytvorená symbolická linka *S13wifi* a v úrovni behu č.1 a 6 bude vytvorená symbolická linka *K86wifi*. Obe tieto symbolické linky budú odkazovať na súbor *wifi* do adresára */etc/init.d/*.
Druhý riadok je tiež povinný. Musí začínať reťazcom **#description:** (popis). Text za nim je už ľubovolný (v prípade, že by popis presahoval jeden riadok, musíme na konci každého riadku napísať znak \ (takzvaný backslash – spätná lomka), ktorý poznáme z prostredia MS DOS/Windows).

Aby sme zabezpečili automatické zavádzanie skriptu **wifi** pri štarte systému, musíme ho pridať do systému menežovania príkazu chkconfig. To zabezpečíme takto:

- do adresára */etc/init.d* prekopírujeme súbor **wifi**
- zadáme príkaz:

```
[root@obelisk root]#  chkconfig --add wifi
```

- skontrolujeme, či sa požadované linky v daných adresároch skutočne vytvorili.

Odteraz vždy pri boote systému sa spustí aj príslušný skript, ktorý nastaví wifi spojenie v Linuxe.

Ak budeme chcieť riadiť službu počas behu systému, využijeme príkaz *service*, napr. takto:

```
[root@obelisk root]#  service wifi start
```

alebo

```
[root@obelisk root]#  service wifi stop
```

alebo

```
[root@obelisk root]#  service wifi restart
```

Ak všetko funguje ako má (musí!), otvoríme fľaštičku dobrého sektu a oslávime tvorbu našej prvej vlastnej služby v systéme.
(Súbor **wifi** si uchováme na neskôr – je naozaj funkčný!).

Blížime sa ku koncu na prvý pohľad nezaujímavých častí, ale opakovane pripomínam, že sa jedná o veľmi dôležitú – možno povedať, že najdôležitejšiu súčasť bežnej agendy správcu systému.
Nabudúce sa budeme venovať modulom a ich správe v systéme. Moduly sú tiež veľmi dôležitá súčasť Linuxu. Pod pojmom modul si môžeme predstaviť akýsi ovládač – drajver – zariadenia, ktorý poznáme už z prostredia MS DOS/Windows.
A potom, keď to už budeme mať za sebou, môžeme sa pustiť do skutočnej stavby linuxového servera.

Linux prakticky ako server/9.časť

V predchádzajúcich dvoch častiach sme si vysvetlili základy práce s procesmi – službami. Ukázali sme si, ako takéto služby spúšťať, ukončovať a ako riadiť ich beh buď hneď od štartu systému alebo počas práce systému. Nie však všetky procesy je potrebné mať stále spustené. Veď iba zaberajú miesto v pamäti a ukrádajú zo systémových zdrojov servera. Preto by bolo vhodné, kedy sa vybrané procesy aktivovali iba vtedy, keď to je naozaj potrebné. Ale ako to vyriešiť? O tom a ešte o moduloch jadra si dnes budeme rozprávať.

xinet démon
Predstavme si takýto príklad:
Na našom linuxovom serveri máme spustenú službu ***smb*** (samba). Pre tých, čo nevedia, na čo slúži Samba, tak len v krátkosti:
Samba je služba, ktorá beží na linuxovom serveri a tvári sa ako server MS Windows NT alebo 2000/XP. Pre bežné klientské stanice s operačným systémom MS Windows 95/98/Me/2k/XP sa javí ako primárny radič domény alebo ako iná klientská stanica s operačným systémom MS Windows. Tieto klientské stanice nedokážu spozorovať, že pracujú s Linuxom. Výhodou tohto je, že nie je potrebné v serverových aplikáciách zakupovať licencie na MS Windows Server. Stačí použiť Linux a Sambu, pretože ako sa správne domnievate, Samba je zadarmo.

Samotná Samba je služba, ktorá – ak ju chceme používať - musí bežať na Linuxe trvalo. Preto je zrejmé, že sa v príslušných adresároch jednotlivých úrovní behu bude nachádzať skript na jej spustenie po štarte systému. V prípade, že to tak nie je, môžeme spustiť Sambu nám už dobre známym príkazom

```
[root@obelisk root]#  service smb start
```

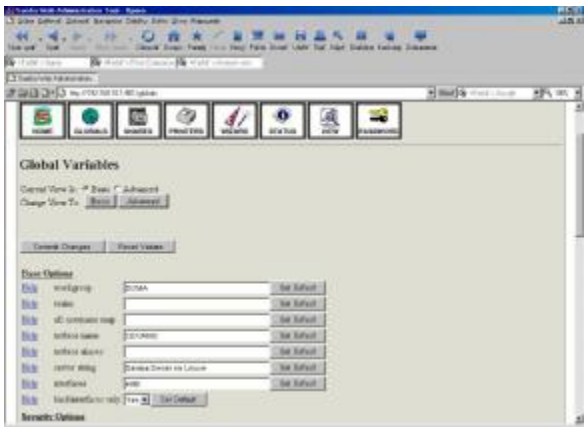
alebo zastavovať príkazom

```
[root@obelisk root]#  service smb stop
```

Toto je pre nás už jasné, vysvetlili sme si to minule.

Balík Samby však obsahuje aj jednu zvláštnu utilitku – konfiguračný nástroj **SWAT** (*Samba Web Admin Tool*). Je to nástroj na konfiguráciu Samby pomocou grafického internetového rozhrania. Na používanie SWAT-u stačí používať bežný internetový prehliadač, ako Opera, Mozilla alebo dokonca aj Internet Explorer (samozrejme stále existuje možnosť konfigurovať Sambu priamo editovaním textového konfiguračného súboru *smb.conf*)

Vzhľad nástroja SWAT je na obrázku č.1:



Ale konfiguračný nástroj sa predsa nepoužíva každý deň!
Preto nie je nutné, aby tento proces, táto služba bežala neustále v pamäti počítača. Stačí, ak bude aktivovaná iba vtedy, keď ju budeme chcieť použiť.

Môžeme síce pred použitím túto službu spustiť a po použití zase ukončiť nám známymi príkazmi *service*, ale je to veľmi nepraktické. Preto by bolo dobré, keby toto za nás strážil a riešil niekto iný. Máte nápad, kto by to mohol byť? No predsa zase nejaký iný proces – služba, ktorá bude trvalo spustená v operačnej pamäti servera. Naozaj, v Linuxe na to existuje zvláštny proces, ktorý sa volá ***inetd***. Ten je už trochu starší a preto bol vytvorený jeho novší klon, ktorý sa volá ***xinetd*** (*eXtended IntErneT services Daemon*). Sú to akési superdémony, ktorých úlohou je spúšťanie služieb na požiadanie.

Poznámka:
V jednom systéme môže existovať iba jeden superdémon. V starších distribúciách alebo v niektorých malých – jednodiskových distribúciách sa dodnes nachádza inetd, v novších modernejších distribúciách je už implementovaný xinetd. Preto sa dnes budeme venovať tomuto procesu.

xinetd sleduje porty, nadefinované v konfiguračných súboroch a v prípade, že zistí požiadavku na komunikáciu na tomto porte, spustí službu, ktorá danú požiadavku vybaví.
Konfigurácia služby *xinetd* sa skladá z dvoch častí:

- *xinetd.conf* – je hlavný konfiguračný súbor samotného procesu *xinetd*
- konfiguračné súbory jednotlivých služieb, ktoré *xinetd* spustí, aby vybavili požiadavku na danom porte.

xinetd.conf
sa nachádza v adresári /etc/. Jeho obsah vidíme na výpise č.2:

```
#
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/

defaults
{
        instances               = 60
        log_type                = SYSLOG authpriv
        log_on_success          = HOST PID
        log_on_failure          = HOST
         cps                    = 25 30
}

includedir /etc/xinetd.d
```

Pozrime sa na jeho obsah podrobnejšie:
Znaky # už poznáme a vieme, že symbolizujú komentár (okrem zvláštnych prípadov).
Celý konfiguračný súbor, vrátane konfiguračných súborov jednotlivých služieb je zložený z bloku parametrov v tvare:

```
názov
{
        premenná = hodnota
}
```

názov predstavuje názov definovaného bloku, *premenná* obsahuje názov premennej, ktorá nadobúda hodnotu *hodnota*.
Zložené zátvorky sú povinné!

V súbore *xinetd.conf* je iba jeden blok, ktorý sa nazýva ***defaults***. Premenné v tomto bloku sú platné pre všetky ostatné bloky.
Obsahuje tieto položky:

- **instances** – určuje maximálny počet bežiacich démonov, poskytujúcich tú istú službu naraz.
  V tomto prípade je dovolených maximálne 60 spojení tej istej služby, napríklad *telnet*. V prípade, že by bola požiadavka na 61. telnetové spojenie, nebude spojenie vykonané, pokiaľ sa niektoré z predchádzajúcich telnetových spojení neukončí.

- **log_type** – definuje, kde budú zaznamenávané správy. Táto premenná môže nadobúdať dve hodnoty:
  - ***FILE meno_súboru [soft_limit [hard_limit]]*** – Nastavenie *FILE* hovorí démonovi *xinetd*, aby zaznamenal aktivitu do súboru zadaného parametrom *meno_súboru*. Horné limity súboru sa nastavujú pomocou parametrov *soft_limit* a *hard_limit*. Keď je prekročený *soft_limit*, *xinetd* vydá varovné hlásenie o veľkosti súboru. Keď je prekročený *hard_limit*, *xinetd* zastaví záznam správ do súboru
  - ***SYSLOG syslog_falicity*** – toto nastavenie hovorí démonovi *xinetd*, aby k záznamu aktivity použil štandardného démona *syslogd*. Parameter *syslog_falicity* musí byť nastavený na jeden z prostriedkov definovaných v súbore */etc/syslog.conf*, napr. *authpriv*
- **log_on_success** – udáva informáciu, ktorá je zaznamenávaná pri úspešnom pripojení k požadovanej službe, napr. adresu vzdialeného klienta (HOST) a ID procesu obsluhujúceho klienta (PID). Existuje ešte niekoľko iných volieb.
- **log_on_failure** – je opakom predchádzajúcej položky, teda zaznamenáva adresu klienta (HOST) pri neúspešnom spojení k požadovanej službe.
- **cps** – nastavuje limit služby v pripojení za sekundu. Prvá hodnota predstavuje maximálny počet pripojenia jednej služby, ktorý bude akceptovaný za jednu sekundu. Ak je toto číslo prekročené, *xinetd* prestane požiadavky akceptovať a počká počet sekúnd daný druhým číslom než začne akceptovanať ďalšie pripojenia. Tým sa zabráni tomu, aby jedna služba alokovala všetky systémové prostriedky.

Okrem bloku ***defaults*** sa v súbore *xinetd.conf* nachádza ďalší výraz **includedir**. Ten hovorí procesu *xinetd*, kde má hľadať konfiguračné súbory jednotlivých požadovaných služieb. V tomto prípade je to adresár */etc/xinetd.d/*. Pozrime sa príkazom *ls* na obsah adresára */etc/xinetd.d/* - výpis č.3:

```
[root@obelisk root]#  ls /etc/xinetd.d/
```

```
amanda          daytime-udp     gssftp          ipop3   pop3s   sgi_fam         time-udp
amandaidx       dbskkd-cdb      chargen         klogin  rexec   swat
amidxtape       echo            chargen-udp     krb5-telnet     rlogin talk
comsat          echo-udp        imap            kshellrsh       telnet
cups-lpd        eklogin         imaps           ktalk   rsync   tftp
daytimefinger   ipop2           ntalk           services        time
```

Toto je príklad možných služieb, ktoré môžu byť spúšťané pomocou procesu *xinetd*, ak vznikne požiadavka na jednu z nich.
Povedali sme si, že sú to konfiguračné súbory jednotlivých služieb. Je medzi nimi aj spomínaná služba SWAT. Pozrime sa teraz na jej konfiguračný súbor */etc/xinetd.d/swat* – výpis č.4:

```
# default: off
# description: SWAT is the Samba Web Admin Tool. Use swat \
#              to configure your Samba server. To use SWAT, \
#              connect to port 901 with your favorite web browser.
service swat
{
        disable         = no
        port            = 901
        socket_type     = stream
        wait            = no
#       only_from       = 127.0.0.1
        user            = root
        server                  = /usr/sbin/swat
        log_on_failure    += USERID
}
```

Už z pohľadu vidíme, že má podobnú štruktúru ako *xinetd.conf*. Tiež sa skladá z bloku, v ktorom sa nachádzajú jednotlivé premenné s príslušnými parametrami.

Názov bloku je **service swat** a má tieto premenné:

- **disable** – táto premenná hovorí démonovi *xinetd*, či je vypnutá alebo nie. Ak je ***disable = no***, služba je zapnutá. Ak je nastavená na ***yes***, služba je vypnutá
- **port** – určuje, na ktorom sieťovom porte služba po spustení pobeží
- **socket_type** – určuje typ soketu použitý pre túto službu. Býva to spravidla ***dgram*** pre datagramovú službu protokolu UDP alebo ***stream*** pre bajtovú službu protokolu TCP. Existujú aj iné hodnoty, v tomto prípade nepodstatné
- **wait** – táto premenná hovorí démonovi *xinetd*, či má pred prijímaním ďalších požiadaviek na spojenie k tejto službe čakať na uvoľnenie portu. Hodnota ***yes*** značí čakaj, hodnota ***no*** značí nečakaj. Spravidla datagramové sokety požadujú čakanie, streamové sokety nevyžadujú čakanie na uvoľnenie portu
- **only_from** – určuje siete, z ktorých môžu prichádzať oprávnené požiadavky na využitie služby. Číslo *127.0.0.1* je označenie spätnej slučky. Ak potrebujeme zabezpečiť prístup aj z iných adries, vypíšeme ich sem. Ak vôbec nechceme obmedzovať spojenie, túto položku zakomentujeme znakom # (ako to máme my)
- **user** – definuje meno používateľa, pod ktorého právami služba pobeží. Spravidla to býva *root*, ale niekedy z bezpečnostných dôvodov aj *nobody*
- **server** – toto je cesta k skutočnému programu, ktorý má proces *xinetd* spustiť. V našom prípade sa spustí program ***/usr/sbin/swat***
- **log_on_failure** – určuje, aká informácia bude zaznamenaná v prípade neúspešného spojenia. Znaky += značia, že sa uvedená hodnota pridá k už defaultne nadefinovaným hodnotám v súbore *xinetd.conf*. V tomto prípade sa bude ešte naviac zaznamenávať používateľské meno použité pri prihlásení k službe.

Aby sme to správne pochopili, zhrnieme si vyššie uvedené nastavenie:
Predstavme si, že chceme použiť službu *swat*. Musíme vedieť, že táto služba funguje na porte 901. Do internetového prehliadača zadáme adresu (napr. 127.0.0.1:901) a klávesom *Enter* požiadavku na spojenie odošleme. Požiadavka príde na linuxový server na port 901. V tomto momente superdémon *xinetd* zistí, že niekto požaduje spojenie na tomto porte. Preto sa pozrie do dvoch špeciálnych súborov */etc/protocols* a */etc/services*, kde vyzistí, že port 901 je priradený službe *swat*. Preto sa pozrie do konfiguračného súboru */etc/xinetd.d/swat*. V prvom rade overí, či je služba zakázaná (*disable*). Ak áno, spojenie ukončí. Ak nie, tak skontroluje, či súhlasí číslo portu (*port*) a z kadiaľ môžu prichádzať požiadavky (*only_from*). Ak všetko súhlasí a požiadavky sú oprávnené, *xinetd* spustí program */usr/sbin/swat* s právami roota. V prípade, že nie sú splnené požiadavky premenných, spojenie sa neuskutoční.

Ovládanie služieb, spúšťaných pomocou xinetd
To, či môže byť daná služba spúšťaná alebo nie rozhoduje správca systému – root. Už vieme, že zakázanie či povolenie služby sa určuje premennou *disable* v jej konfiguračnom súbore.
Jej zmenu môžeme dosiahnuť dvoma spôsobmi:
- priamou editáciou príslušného súboru, napr. pomocou Midnight Commandera alebo iného editora (vi)
- pomocou už známeho príkazu ***chkconfig***

Ak zadáme príkaz

```
[root@obelisk root]#  chkconfig  --list smb
```

zistíme, ako či je služba dovolená, napr:

```
smb                     on
```

To značí, že premenná je *disable = no*

Preto ak chceme zmeniť hodnotu na *disable = yes*, musíme zadať príkaz

```
[root@obelisk root]#  chkconfig  smb off
```

Musíme si uvedomiť, že samotná zmena premennej *disable*, či iný zásah do konfiguračného súboru danej služby nestačí. Ešte musíme dať vedieť procesu *xinetd*, že sme zmeny vykonali a že ich musí znova načítať. To dosiahneme príkazom

```
[root@obelisk root]#  service xinetd restart
```

Až potom sa uvedené zmeny prejavia!

Zapamätajme si!
V Linuxe existujú dva druhy služieb:

- **stále služby, spúšťané už pri štarte systému alebo spustené z príkazového riadku a ktoré až do odvolania zostávajú trvalo v pamäti počítača**
- **služby na požiadanie, ktoré spustí superdémon xinetd (alebo inetd) na základe požiadavky, ktoré sa po vykonaní svojej práce automaticky ukončia.**

To, ktorá služba bude stála a ktorá na požiadanie, je už prednastavené pri inštalácii systému, ale nič nám nebráni, aby sme toto zmenili (zatiaľ som nemal dôvod).

Linux prakticky ako server/ 10.časť

Moduly
Doteraz sme sa rozprávali o programoch, ktorých úlohou je komunikovať medzi sebou navzájom alebo medzi systémom a používateľom. Nesmieme však zabudnúť, že každý výpočtový systém – odhliadnúc od výrobcu, je postavený na nejakom „železe“ – hardvéri, teda na technických zariadeniach. A aby mohol ľubovolný operačný systém k týmto technickým hardvérových komponentám pristupovať, potrebuje na to malé časti strojového kódu, ktoré sprostredkujú komunikáciu medzi technickým prostriedkom a operačným systémom. Tomuto malému programu hovorievame ovládač zariadenia (občas sa môžeme stretnúť aj s termínom *driver* – drajver, čo je fonetický prepis anglického slova ovládač).
Isto ste sa už vo svojej informačnej praxi s týmito ovládačmi stretli – prinajmenšom v prostredí MS Windows. Nie je tomu inak ani v Linuxe.
Boli však doby (ešte úplne v začiatkoch Linuxu), keď boli príslušné ovládače súčasťou jadra Linuxu. Jadro potom použilo ten správny ovládač ku každému prostriedku. Keďže sa informačné technológie rozrastajú neuveriteľným tempom, vznikajú tisíce nových technických zariadení - počnúc rôznymi sieťovými kartami, cez grafické karty a končiac nepreberným množstvom tlačiarní – je nepredstaviteľné, aby existovalo akési univerzálne jadro, ktoré by v sebe nieslo všetky príslušné ovládače (teda, ono by existovať aj mohlo, ale bolo by velmi veľké, ťažko by sa s ním narábalo a manipulovalo pri dekompresii u bootu systému).
Preto sa pristúpilo k tomu, že sa (až na malé výnimky) ovládače nachádzajú mimo jadra, a to vo forme malých prípojných programčekov, ktorým sa hovorieva **moduly**. Tento princíp modulov zostal zachovaný dodnes, a ako potvrdila prax, bol to výborný nápad.

Modul je kus programového kódu, ktorý po svojom zavedení do operačnej pamäte komunikuje medzi jadrom Linuxu a hardvérom. Z toho je zrejmé, že pre každý technický prostriedok musí byť iný príslušný ovládač a pre každý operačný systém takisto. Nie je možné používať tie isté ovládače pod Linuxom a v MS Windows a zároveň nie je možné použiť ovládač pre grafickú kartu na ovládanie modemu.

Bohužiaľ, výrobcovia hardvéru zanedbávajú Linux pri tvorbe príslušných ovládačov k svojmu zariadeniu. Spravidla na priloženom cédečku alebo diskete nájdeme ovládač pre MS produkty, ale pre Linux nie. Musíme ale priznať, že sa situácia zo dňa na deň zlepšuje a tak už dnes nie je neobyčajné nájsť na diskete od sieťovej karty aj ovládač pre Linux. U iných hardvérových komponentov to už tak časté nebýva, ale uvidíme, čo prinesie budúcnosť.

Preto vývojárom Linuxu nezostávalo nič iné, len si príslušné ovládače vytvoriť sami. Jasné, že vznikali najprv pre najrozšírenejšie produkty, ale dnes je situácia veľmi dobrá aj u iných komponentov. Tieto moduly často bývajú kolektívnym dielkom, keď sa na jeho vývoji podieľala veľká skupina ľudí. Spravidla autor myšlienky na modul pre konkrétne zariadenie vytvorí akúsi beta verziu ovládača a dá ho k dispozícii linuxovej komunite. Hneď na to vzniknú rôzne opravy – takzvaný patch (čítaj peč, z anglického slova záplata), ktoré odstránia najmarkantnejšie chyby. V zmysle licencie GPL je to všetko dovolené a nakoniec niekto pridá rôzne vylepšenia a pomerne slušný ovládač je na svete.
Ak sa aj my stretneme s tým, že nejaký modul má určitú chybu, máme dve možnosti – buď túto chybu oznámime autorovi (autorom) a požiadame ich o odstránenie (hovoríme tomu aj bugfix), alebo ak sa na to cítime, danú chybu opravíme v zdrojových kódoch, modul prekompilujeme a opravu sami zverejníme na Internete.

O tom, že nadšenie pre tvorbu vlastných ovládačov linuxovou komunitou je častokrát efektnejšie, ako u výrobcov iných proprietárnych operačných systémov, svedčí aj moja skúsenosť:

Prednedávnom, keď som si zakúpil svoj prvý (starší) notebook, rozhodol som sa vytvoriť *dualboot* medzi Linuxom a MS Windows 2000. A neverili by ste, aký som bol prekvapený, keď som zistil, že MS Windows 2000 nedokázali správne nakonfigurovať grafickú kartu ani zvukovú kartu, zatiaľ čo Linux ich spoľahlivo identifikoval a nastavil najvhodnejšie parametre. (Tie MS Windows som potom pracne dolaďoval pomocou driverov z Internetu – to bola fuška!).

Vráťme sa k modulom.
Výhodou linuxových modulov je, že ich môžeme zaviesť alebo odstrániť z operačnej pamäte za behu systému. Jednoducho zavedieme príslušný ovládač, zistíme ako funguje, ak sa nám niečo nepáči, ovládač odstránime,

môžeme dať iný alebo tento upravíme a znova zavedieme. A to všetko bez jediného reštartu systému. (Skúste to urobiť v MS Windows!)

Zapamätajme si, že moduly majú v Linuxe príponu .o (o ako object), pretože sú to objektové moduly. Modul je tesne zviazaný s príslušným jadrom, a to dokonca s jeho číselnou verziou. Preto ak chceme použiť nejaký modul, buď musíme použiť ten, ktorý bol skompilovaný pre to isté jadro, ktoré vlastníme aj my alebo si ho musíme sami skompilovať. Inak nemusí správne fungovať, ba často nám sám oznámi, že je skompilovaný pre iné jadro! Možno sa nám to teraz zdá veľmi zložité, ale nebojme sa, zvládneme to!

Moduly sú v adresárovej štruktúre Linuxu uložené vo viacerých adresároch podľa príslušnosti k zariadeniu. Základný adresár je */lib/modules/číslo_jadra/kernel/drivers/*, kde sa nachádzajú ostatné podadresáre podľa zariadení, napr. */net*, */cdrom*, */sound*, */usb* a iné. Napríklad driver na moju sieťovú kartu s názvom ***e1000*** v distribúcii *Fedora Core 1*sa nachádza v adresári */lib/modules/2.4.22-1.2115.nptl/kernel/drivers/net/*. Môže sa stať, že v iných distribúciách sa bude nachádzať na inom mieste, spravidla však koreňový základ */lib/modules/* býva zachovaný.

Samotným modulom, ich kompilácii, tvorbe a podobne sa budeme postupne (neskôr) venovať podrobnejšie, dnes si povieme úplné základy práce s nimi.

Utility na prácu s modulmi
Na prácu s modulmi sa používajú štyri základné nástroje, ktoré sa nachádzajú v balíku *modules* (*modules.tar.gz* alebo *modules.rpm*). Tento balík sa nainštaluje už pri inštalácii celého systému, ale ak chceme, môžeme si stiahnúť poslednú verziu napr. z Internetu a sami preinštalovať.

Sú to tieto utility:
- **lsmod**
- **insmod**
- **modprobe**
- **rmmod**

lsmod
slúži – podobne ako príkaz *ls* – na výpis nainštalovaných modulov v systéme.
Zadáme príkaz

```
[root@obelisk root]#  lsmod
```

Na obrazovke sa objaví výpis príslušných modulov – výpis č.5:

```
Module                  Size  Used by    Not tainted
parport_pc             19076   1  (autoclean)
lp                      9060   0  (autoclean)
parport                37056   1  (autoclean) [parport_pc lp]
autofs                 13364   0  (autoclean) (unused)
hostap_pci             42076   1
hostap                 71652   0  [hostap_pci]
hostap_crypt            2768   0  [hostap]
ne2k-pci                7136   1
8390                    8600   0  [ne2k-pci]
e1000                  71616   1
floppy                 58012   0  (autoclean)
---- skrátené ---
ide-cd                 35776   0
cdrom                  33728   0  [sr_mod ide-cd]
ohci1394               29160   0  (unused)
ieee1394              204676   0  [ohci1394]
keybdev                 2976   0  (unused)
mousedev                5556   0  (unused)
ext3                   71300   2
jbd                    52084   2  [ext3]
```

Prvý stĺpec s názvom *Module* popisuje názov modulu, druhý stĺpec s názvom *Size* udáva jeho veľkosť. Zaujímavý je pre nás posledný stĺpec. Ten hovorí o závislosti jednotlivých modulov navzájom.

Modulová závislosť
Modulová závislosť (*module dependence*) je v Linuxe bežná, ale dôležitá vec.
Nie je potrebné vytvoriť modul, ktorý musí obslúžiť úplne všetko sám. Výhodnejšie je napísať modul, ktorý pre svoju činnosť môže využiť iné, už hotové a funkčné moduly. Tým sa dosiahne zmenšenie modulov, a tak zbytočne neobsadzujú drahocennú operačnú pamäť počítača. Pozrime sa uvedený výpis ešte raz. Modul, ktorý obsluhuje moju druhú sieťovú kartu, sa nazýva *ne2k-pci*. Všimnime si, že tento modul je závislý na ešte jednom module, ktorého názov je *8390*. To, že je na ňom závislý modul *ne2k-pci*, spoznáme podľa toho, že je uvedený v hranatých zátvorkách.

Všimnime si parameter *autoclean*. Ten znamená, že sa modul automaticky uvoľní z pamäte, ak nebude 60 sekúnd používaný.

insmod
Tento príkaz slúži na ručné zavedenie modulu do operačnej pamäti. Je veľmi jednoduchý a jeho použitie je

```
[root@obelisk root]#  insmod meno_modulu
```

Zapamätajme si, že sa meno modulu zadáva bez prípony! Teda ak chceme zaviesť modul *e1000.o*, zadáme iba

```
[root@obelisk root]#  insmod e1000
```

Tento príkaz má však jednu nevýhodu – nerozumie vyššie spomínanej modulovej závislosti. Predstavme si, že chceme zaviesť modul *ne2k-pci*(.o) do pamäti:
Ak zadáme príkaz

```
[root@obelisk root]#  insmod ne2k-pci
```

modul sa síce do pamäti počítača zavedie, ale nebude dobre pracovať, pretože podľa uvedeného výpisu je zrejmé, že je závislý na module 8390.o a príkaz *insmod* nevie, že musí do pamäti zaviesť zároveň modul *8390*.

modprobe
Preto je pri ručnom zavádzaní modulov vhodnejšie použiť príkaz **modprobe**. Jeho použitie je z používateľského hľadiska jednoduché, zadáme

```
[root@obelisk root]#  modprobe ne2k-pci
```

a modul si sám zistí, na ktorých moduloch je tento modul *ne2k-pci* závislý. Zistí, že je potrebné ešte zaviesť modul *8390* a ako ho nájde, zavedie ho automaticky. V prípade, že potrebný modul nie je dostupný, príkaz *modprobe* vypíše hlásenie, ktoré nás na chýbajúci modul upozorní.

Ale odkiaľ príkaz *modprobe* vie, ktoré moduly sú na ktorých závislé? To si zistí zo súboru */lib/modules/verzia_jadra/modules.dep*, ktorý bol vytvorený príkazom **depmod**.
V prípade, že do systému pridávame nové moduly alebo vykonávame kompiláciu nových modulov, je potrebné zadať príkaz

```
[root@obelisk root]#  depmod -a
```

Ten skontroluje všetky moduly na disku, zistí ich vzájomné závislosti a vytvorí súbor *modules.dep*

rmmod
Už vieme modul do pamäte zaviesť, ale často je potrebné modul z pamäti aj odstrániť. Na to slúži príkaz **rmmod**. Ak chceme odstrániť modul *usb*, zadáme príkaz

```
[root@obelisk root]#  rmmod usb
```

a modul bude z pamäte odstránený.

Automatické zavedenie modulu
Obidva príkazy – *insmod* aj *modprobe* vykonávajú ručné zavedenie modulov do pamäte počítača, teda je potrebné zadať tieto príkazy z konzoly. Aj keď môžeme vytvoriť príslušný skript a uložiť ho do niektorých štartovacích skriptov systému, aby sme si to čiastočne zautomatizovali, nebude to automatizácia v pravom zmysle slova.
Na automatické zavádzanie modulov je určená funkcia jadra s názvom **kmod**. Tá zavedie potrebné ovládače – moduly bez zásahu správcu systému už pri štarte systému. Ak nie sú moduly 60 sekúnd používané, budú z pamäte automaticky uvoľnené. Takto zavedené moduly sú označené príznakom *autoclean*.

Na dnes stačí, a nabudúce začíname na ostro!

Miroslav Oravec