E-közigazgatási Szabad
Szoftver Kompetencia Központ

Mátó Péter, Rózsár Gábor, Őry Máté,
Varga Csaba Sándor, Zahemszky Gábor

20/80

Linux rendszerek alapvető beállításai, üzemeltetése

ÚJ SZÉCHENYI TERV

20/80 – Linux rendszerek alapvető beállításai, üzemeltetése

Módszertan és tartalmi tervek: Mátó Péter, Varga Csaba Sándor, Zahemszky Gábor

Írták: Mátó Péter, Rózsár Gábor, Őry Máté, Varga Csaba Sándor, Zahemszky Gábor:

0.9-es változat, 2014. április 11., elektronikus kiadás

Kiadó: E-közigazgatási Szabad Szoftver Kompetencia Központ

Honlap, javított kiadások: http://szabadszoftver.kormany.hu/sajat-oktatasi-anyagok/.

ISBN 978-963-08-8299-6

Szerzői jog

Ez a könyv a Creative Commons Attribution-ShareAlike 3.0 Unported (CC-BY-SA 3.0) licenc szerint szabadon terjeszthető és módosítható.

További információk: http://creativecommons.org/licenses/by-sa/3.0/

A dokumentumban található összes védjegy azok jogos tulajdonosait illeti meg.

Tartalomjegyzék

Előszócska

Ezzel a könyvvel és testvéreivel az a célunk, hogy viszonylag tömören összefoglaljuk azokat az információkat, amiket egy szabad szoftvereket használó szakembereknek tudnia illik.

20/80. Mit akar ez jelenteni? Tapasztalatunk szerint a létező eszközöknek és információknak csak egy kis része szükséges a mindennapok tipikus feladatainál. Igyekeztünk kiválogatni nektek a tudásnak azt a 20%-át, ami az általában előforduló feladatok 80%-ánál elegendő lesz. Célunk ezen elv alapján összeszedni, rendszerezni és átadni a leghasznosabb dolgokat. Hiába próbálnánk mindent elmondani – nekünk nincs időnk mindent leírni, nektek meg nincs időtök elolvasni. Ezért sok minden kimarad. Ha úgy gondolod, hogy fontos, kimaradt vagy bővebben kellene beszélni róla, szólj! Ha valami hibás, szólj! E-mail címünk: esz2k2@gmail.com. De ha írsz, légy türelemmel, valószínűleg 200 másik levél is vár még megválaszolásra. A továbbfejlesztés során minden konstruktív javaslatot igyekszünk majd az anyagba építeni.

A tárgyalt megoldások és szabad szoftverek legtöbbször több operációs rendszer alatt is használhatóak. Amikor viszont operációs rendszer szintről esik szó (telepítés, csomagkezelés vagy finomhangolás), akkor ez most – népszerűsége miatt – nálunk Linuxot jelent.

A könyvben időnként kérdéseket teszünk fel, de néha nyitva hagyjuk a választ. A cél: gondolkozz, olvass utána, használd az agyadat! Ha egy témát alaposabban meg akarsz ismerni, akkor nincs mese, alaposabban utána kell olvasnod. Minden területnek megvannak a maga – tőlünk sokkal mélyebb ismereteket tárgyaló – szakkönyvei, előttük azért érdemes a mi összefoglalónkat elolvasni, mert ezekben – reményeink szerint – az adott terület esszenciája található. Ez alapján már könnyebben eligazodsz majd a 6-700 oldalas, lényegesen kimerítőbb anyagokban is.

Virtualizációról dióhéjban, mini virtuális tesztlabor kialakítása

A ma kapható legkisebb teljesítményű számítógép is olyan erős processzorral, annyi memóriával rendelkezik, hogy könnyedén el tud látni több független feladatot is. Különösen akkor tudjuk optimálisan kiterhelni a rendszert, ha olyan feladatokat tudunk rábízni, melyek nem egy időben okoznak nagyobb terhelést. Például egy gépen remekül egymás mellett futhat a pénzügyi rendszer adatbázis kezelője, mely főként hónap végén, munkaidőben kap nagyobb terhelést, és a belső rendszerek mentésére szolgáló szoftver, mely főképp éjszaka dolgozik. De ezeket a funkciókat érdemes lenne biztonsági és kényelmi okokból is elválasztani. A virtualizáció segítségével egy számítógép erőforrásait úgy oszthatjuk meg, hogy a virtualizációs szoftver valamilyen módon különálló környezetet hoz létre a szeparálandó szolgáltatásnak.

Erre alapvetően kétféle módszer van. Az egyik, hogy a szoftver lényegében egy teljes számítógépet szimulál vagy emulál. Ezt nevezzük virtuális gépnek (VM, virtual machine). A VM-ek a rajtuk futó operációs rendszerek és szoftverek szempontjából látszólag ugyanolyan felépítésűek mint egy általános célú asztali számítógép, van bennük processzor, memória, háttértár és hálózati eszköz, természetesen akár több is. A virtualizációs szoftvert futtató gépet általában host-nak nevezik, a virtuális gépeket pedig guest-nek. Ekkor tehát a virtuális gépen lényegében egy teljesen független operációs rendszer indul el, azt fel kell telepíteni, be kell állítani és utána a host futása közben tetszőlegesen elindítható vagy leállítható, a benne lévő virtuális eszközök kikapcsolt állapotban (bizonyos esetekben bekapcsolva is) módosíthatók. Például nagyon gyorsan lehet egy VM-be több memóriát tenni.

A másik módszer egy szeparált környezet létrehozása az adott számítógép környezetében, ekkor nem egy független operációs rendszer indul el, hanem a host kernelén fognak futni a processzek, csak általában leválasztott fájlrendszert használnak és a legtöbbször független hálózati eszközük van (akár valódi, akár virtuális). Ez megvalósítható az Linux alaprendszer részét képező „chroot” rendszerhívással, de ilyen funkciót nyújt például az LXC is. Ha az azonos kernel használata elfogadható, akkor jó választás lehet, mert teljesítményben valamivel jobbak, mint a virtualizált gépek használta. Mivel azonban használatuk kissé nehézkesebb, mint a virtuális gépek használata, mi azt ajánljuk, hogy szerencsésebb virtuális gépeket használni.

Linuxos környezetben több különböző lehetőségünk is van virtuális gépeket használni. Van szabad és tulajdonosi szoftver, ingyenes és pénzes megoldás is. Néhány ezek közül:

– KVM és QEmu – a KVM a Linux kernel része, a QEmu pedig egy korábbi független virtualizációs eszköz mely a KVM-hez is használható. Ez a két eszköz kiválóan használható szerveren és munkaállomáson egyaránt, bár a virtuális gépek menedzsmentjére további szoftvereket kell majd használni.
– VirtualBox (OSE) – a VirtualBox kereskedelmi termék, aminek az OSE (Open Source Edition) verziója nyílt forráskódú, de kevesebb funkciója van, mint a zárt verziónak. Kényelmes felhasználói felülete, több platform támogatása és praktikus funkciói miatt desktop felhasználásra nagyon ajánlott.
– VMware Player/Workstation/szerver család – úttörő, piacvezető zárt forrású szoftver család. A Player ingyenesen használható freeware. Noha sok tulajdonságában fejlettebb szabad konku-

renseinél, közepes méretű vállalatig ezek a funkciók általában nem kritikusak. Speciális igények esetén, amennyiben a szabad szoftverek valamilyen problémára nem tudnak megnyugtató megoldást nyújtani, beszerzése megfontolandó.

– És még sokan mások. Néhány név a jelentősebb további versenyzők közül: XEN, Proxmox (mely egy kényelmes előtét a KVM-hez és az LXC-hez)

Ezek közül az első 3 egészen más kategóriát képvisel, mint az utolsó 2:

– A QEmu egy általános célú emulációs rendszer, amely többek között x86 architektúra emulációját is képes megvalósítani (illetve megfelelő hardveres támogatás esetén virtualizációra is képes);
– a VirtualBox kereskedelmi termék, aminek az OSE (Open Source Edition) verziója nyílt forráskódú, de némileg kevesebb funkciója van, mint a zárt verziónak;
– a VMware Player és a VMware Workstation szintén kereskedelmi termékek, melyeket (hasonlóan a QEmuhoz és a VBoxhoz) elsősorban asztali felhasználásra szántak. Ezek is a kezdetektől használhatók Linux rendszerek alatt.

A XEN és a KVM sokkal inkább javasolható szerver felhasználásra[1]. Míg a XEN egy paravirtualizációs technológia segítségével a processzorok hardveres támogatása nélkül is képes virtualizációra, addig a KVM működtetéséhez feltétlenül szükséges a processzorok hardveres virtualizációs támogatása. Ma ez azért már nem akkora probléma, mivel mind az Intel, mind az AMD által gyártott modern processzoroknál jó pár éve megtalálható ez a támogatás (Intel VT vagy VMX, illetve AMD-V vagy SVM néven nevezett processzor tulajdonságok).[2] Ennek ellenére fontos tudni, hogy ha valahol olyan x86-alapú gépen szeretnének szerver virtualizációt, amely gép processzora hardveresen nem támogatja azt, akkor nekik a KVM nem alkalmas.

A mi választásunk ennek ellenére a KVM-re esett. Az ok: XEN esetén egy kifejezetten a XEN használatára előkészített másik kernelt kell használni a virtualizációs támogatáshoz, míg KVM esetén elegendő egyetlen[3] kernel modul betöltése, és máris rendelkezésünkre áll a virtuális gépek használatának lehetősége. A KVM támogatás egy ideje (2.6.20-as kernelverzió óta) része a hivatalos Linux kernelnek, ráadásul egyre több disztribúció szállítja alapértelmezett virtualizációs eszközként (esetleg egyedüliként, mint pl. a kereskedelmi RedHat – és klónjai: CentOS, Scientific Linux – a 6-os verzió óta).

Virtualizáció esetén elterjedt kifejezés a gazda (host) és vendég (guest) számítógép (esetleg operációs rendszer) – az előző a fizikai gépet, az utóbbi a virtuális gépet (vagy az abban futó oprendszert) jelenti. KVM használata esetén a gazda gépen fut egy teljesen általános Linux disztribúció[4], amelynél amennyiben használni szeretnénk a virtuális gépeinket, akkor csak betöltjük a kvm nevű kernel-modult.

Virtuális gépek használatához létre kell hozni azokat, majd az így létrehozott VM-ekbe operációs rendszert kell telepíteni (ez utóbbi nem lényegesen tér el a valódi hardveres telepítéstől). Ahogyan

1. A dokumentáció írásakor a XEN kicsit többféle architektúrán működik, mint a KVM, de mivel jelenleg az x86 (és a 64-bites x86_64) alapú gépek a legelterjedtebbek még a közepes méretű vállalatoknál is, ez nem jelent valódi problémát.
2. Működő Linux rendszerben a `grep -E 'vmx|svm' /proc/cpuinfo` paranccsal ellenőrizhető a megléte (ha XEN-kernel alól próbáljuk lekérdezni, akkor nem látszik; ezen kívül néha a gépek hardveresen támogatják, de a BIOS-ban alapból le van tiltva a funkció)
3. igazából kettő, mert a fő (`kvm` nevű) modul mellett kell még egy, amely a használt processzortól függő – vagy `kvm_intel`, vagy `kvm_amd` névre hallgat; ez utóbbiak közül a megfelelőt a `kvm` modul automatikusan betölti
4. amennyiben szervereket virtualizálunk, akkor erőteljesen javasolt a host rendszer funkcióit minimalizálni, és ténylegesen csak a nélkülözhetetlen szolgáltatásokat futtatni rajta (pl. NTP és SSH)

a valódi számítógépek, úgy a virtuális gépek is a következő fő komponensekből állnak, a VM létrehozásánál ezekre kell odafigyelni:

– CPU – a processzorral sok gond nincs, a gépben fizikailag meglevő processzort fogják a VM-ek látni, legfeljebb azt kell beállítani, hogy hány darab, hány magos, hány szálúnak lássa a VM. Egy VM-be sem konfigurálhatunk több processzort, mint amennyi a gazda számítógépben létezik (ebbe viszont már beletartozik a gazda gép HyperThreading, illetve multi-core tulajdonsága is). Azaz egy 2 fizikai processzoros, amúgy processzoronként duplamagos (dual-core), ráadásul Intel gyártmányú (így HyperThreading támogatással rendelkező) gazdagép esetén, mivel ez látszólag 8 processzor, ezért maximum 8 processzoros virtuális gép hozható létre. (Igaz, akár több is.)
– memória – természetesen itt is a gazdagép által elérhető memória szab határt
– háttértár – egy virtuális géphez egy vagy több, IDE (ATA), SATA, esetleg SCSI, SAS-interfészen keresztül elérhető virtuális diszket, és elsősorban a telepítéshez valamilyen CD/DVD eszközt kell konfigurálni. Ezek a virtuális diszkek lehetnek a akár gazda számítógép csak erre a célra fenntartott fizikai eszközei, vagy a gazda számítógép fájlrendszerében létrehozott un. diszkképek (disk image). (Természetesen a virtuális diszkekhez virtuális háttértár-vezérlőre is szükségünk van – és az egyes virtualizációs rendszerek között elég nagy eltérések lehetnek, hogy milyen vezérlőket szimulálnak.)
– hálózati hozzáférés – valódi gépek esetén (szerver környezetben) jellemzően hagyományos Ethernet-vezérlőket használunk. A virtuális gépek eléréséhez szükséges hálózatot pedig a VM-hez létrehozott virtuális hálózati interfészek szolgáltatják.
– egyebek – a valódi számítógépekben ezen kívül elő-előfordulnak egyéb hardverelemek, mint a géppel való kapcsolattartásra jellemzően használt billentyűzet, egér, grafikus kártya, soros, párhuzamos és USB-csatlakozók, egyebek.

A virtuális gépek kezelésére több eszköz is rendelkezésre áll. Általában van egy specializált parancskészlet, aminek segítségével az adott virtualizációs eszköz speciális szolgáltatásai érhetőek el. Ez tipikusan parancssoros eszköz (pl. a XEN-hez tartozik egy `xm` nevű eszköz, a VirtualBox-hoz egy `VBoxManage` nevű segédprogram, és í. t.), de sokszor grafikus felületű adminisztrációs szoftver is létezik. A KVM praktikus okokból a fent említett QEmu parancssoros eszközének módosított verzióját használja. A használt terjesztéstől némileg függő módon, ez a parancs hol `qemu-kvm`, hol csak egyszerűen `kvm` néven érhető el – Ubuntu 12.04 LTS alatt `kvm` a neve. (A KVM-hez elérhető különböző menedzsment eszközök listája a http://www.linux-kvm.org/page/Management_Tools címen található, innen mindenki választhat magának, ha a mi javaslataink nem nyerték el a tetszését.)

Ezeknek a fent említett egyedi eszközöknek a használata ma egyre ritkábban szükséges, ugyanis folyamatos fejlesztés alatt áll egy `libvirt`[5] nevű eszköz, mely az elterjedtebb virtualizációs környezeteket támogatja – egységes felületet biztosítva hozzájuk. Azaz ha valamely eszköz a libvirtre épít, akkor gyakorlatilag mindegy, hogy a háttérben milyen virtualizációt használunk, ugyanazon a felületen keresztül kell karbantartani. A libvirt pozitívumai közé tartozik, hogy ha szükségünk van rá, akár a lokális, akár a távoli gépen futó virtualizációs szoftverrel képes kommunikálni (távoli gép esetén természetesen biztonságos kapcsolat kiépítésével).

Két ilyen libvirtre épülő eszköz a libvirt terjesztés szerves részét képező `virsh`, valamint a Python-nyelven írt Virtual Machine Manager (vagy `virt-manager`)[6]. Míg az első parancssoros eszköz, addig az utóbbi egyrészt nyújt parancssoros eszközt (`virt-install`) másrészt egy kényelmes grafikus

5. http://libvirt.org
6. http://virt-manager.org/

felületű alkalmazást – ez maga a VMM. (A libvirt-ről tudni kell, hogy a virtuális gépekre vonatkozó információkat XML-formájú fájlokban tárolja. Ez elsősorban akkor kellemetlen, amikor az ember élete első virtuális gépét készíti el kézzel.)

KVM használat natív eszközei

Lássunk egy virtuális gép létrehozást és indítást a KVM natív eszközeivel:

Első lépésként hozzuk létre a virtuális gép diszkjét. (A példához nyilván elég a 100MB, normális esetben valószínűleg nagyobb diszk kellene.)

```
qemu-img create -f qcow2 vm1diszk1.img 100m
```

A példában szereplő qcow2 elég hasznos paraméter, ez a virtuális diszkformátum nyújtja a legtöbb hasznos szolgáltatást, ha lehetőségünk van rá, használjuk mindig ezt. (A dokumentáció írásakor pl. a grafikus felületű VMM-ben sajnos ezt nem lehet a létrehozás pillanatában beállítani, hanem trükközni kell vele – remélhetőleg ezt a hiányosságot a közeljövőben megszüntetik.)

A létrehozott 100 MB-os diszket odaadjuk a virtuális gépnek, ami kap egy korábban letöltött DVD-képet amiről bootolhat, kap 768MB memóriát, 1 db duplamagos, HyperThread-támogatású processzort[7] és 1 db Pcnet típusú hálózati interfészt (a Pcnet egy nagyon régi hálózati kártya típus, sok évvel ezelőtti operációs rendszerek esetén jöhet jól – ma az alapértelmezett Intel e1000 típusú kártya szinte biztosan jobb választás). Boot eszközként a DVD-t (d) és a diszket (c) állítjuk be

```
kvm -name vm1 -m 768 -smp 4,cores=2,sockets=1 -hda vm1diszk1.img -cdrom DVD.iso
  -net nic,model=pcnet -boot order=dc
```

A paraméterként megadott név a virtuális gépben semmilyen szerepet nem látszik, csak az adminisztráció megkönnyítésére szolgál, és természetesen ott válik majdnem kötelező egyedi paraméterré, amikor nem egyetlen példa virtuális gépünk, hanem sok (10, 100) VM fut ugyanazon a gazdagépen, és ezeket kell valamilyen módon megkülönböztetni. Értelemszerűen ilyen esetekben valami ennél jobb névválasztási konvenció javasolt – aki sok oprendszert ki szeret próbálni, az használja az OS-nevét, verziószámát; aki különböző funkciójú M-ket hoz létre, az használhat funkció alapú nevezéktant – ezt mindenki maga dönti el, de érdemes valami vm1, vm2, vm3, stb.-nél logikusabbat használni.

Természetesen mind a qemu-img, mind a kvm parancs ennél sokkal bonyolultabb lehet, attól függően, hogy milyen egyéb dolgokat szeretnénk beállítani (például ha a gépünkön ahonnan a VM-et el fogjuk érni magyar billentyűkiosztás van érvényben, akkor kényelmesebb, ha a VM-ben is olyat konfigurálunk be a -k hu paraméterrel). Mint látható, a dolog nem túl bonyolult, bár kétségtelen, hogy ha a későbbiekben is szeretnénk ezt a virtuális gépet használni, akkor nem túl kényelmes indításonként mindig ezt a hosszú (vagy esetleg még ennél is sokkal hosszabb) parancssort begépelni. Ráadásul még csak nem is pont ez kell, hiszen a későbbiekben már nem valószínű, hogy a telepítő CD/DVD-képről szeretnénk indítani a virtuális gépet, hanem valószínűleg a diszkre telepített operációs rendszerről (azaz napi használathoz valószínűleg -boot order c kell. Ha a DVD-képet a telepítés utáni futásokkor változatlanul odaadjuk a VM-nek, akkor az úgy látja, mintha a telepítő CD/DVD mindig rendelkezésre állna – ez akkor érdekes, ha esetleg újabb alkalmazásokat szeretnénk a későbbiekben a CD/DVD-ről telepíteni) . Hosszú távon tehát jól jö-

7. mivel egy processzorfoglalatot és processzoronként 2 magot írtunk elő, muszáj HyperThreadnek lenni, különben nem lenne meg a szintén előírt 4 (virtuális) processzor, de persze elő is írható a példában nem szereplő threads=2 paraméterrel

het, ha a megfelelően módosított parancssort egy egyszerű szöveges parancsfájlba elmentjük – mondjuk startvm1.sh néven.

KVM a libvirt saját virsh parancsával

Most nézzünk egy virtuális gépet a libvirtre épülő virsh segítségével.

Pár szakkifejezés[8]:

– node – a korábbiakban host gépként aposztrofált fizikai gép
– hypervisor – a virtualizációs szoftverréteg
– domain – egy operációs rendszer (vagy alrendszer) egy példánya, amely valamely hypervisor által szolgáltatott virtuális gépben fut (korábban vendég-ként említettük)

Ha a libvirttel dolgoznánk, elsőként telepítenünk kell a libvirtet és a hozzá tartozó segédeszközöket. (Ubuntu 12.04 LTS esetén ez a sudo apt-get install libvirt0 libvirt-bin.) Ez nyilván egyszeri feladat. A továbbiakban már első lépésként mindig kapcsolódni kell a konkrét virtualizációs alrendszerhez – ez lehet a helyi gép vagy a hálózat egy másik gépén futó hypervisor – majd a már élő kapcsolatot felhasználva kezelhetjük virtuális gépeinket. Az alapbeállítás ellenőrzéséhez adjuk ki a

```
virsh uri
```

parancsot. Lokális gépen futó KVM esetén a qemu:///system a megfelelő. Ha esetleg nem az, akkor csatlakozzunk a

```
virsh connect qemu:///system
```

paranccsal. Ubuntun ez az alapértelmezett hely, ezért ott ez el is hagyható. (Mint korábban elhangzott, a KVM erősen épít a QEmu egyes részeire, ezért kell KVM esetén is qemu-típusú kapcsolatot felépíteni.) Új virtuális gép létrehozásakor a következő lépésként létre kell hozni a virtuális gépet leíró XML-fájlt.

A fenti VM XML-formátumú konfigfájlja

```
<domain type='kvm' id='1'>
<name>vm1</name>
<memory>786432</memory>
<vcpu>4</vcu>
<cpu>
<topology sockets='1' cores='2' threads='2' />
</cpu>
<devices>
<disk type='file' device="disk'>
<source file='/path/to/vm1diszk1.img'/>
<target dev="hda' />
</disk>
<disk type='file' device='cdrom'>
<source file='/path/to/DVD.iso' />
<target dev='hdc' />
```

8. http://libvirt.org/goals.html

```
<readonly/>
</disk>
<interface type='network'>
<source network='default'/>
<model type='pcnet'/>
</interface>
</devices>
<os>
<type>hvm</type>
<boot dev='cdrom'/>
<boot dev='hd'/>
</os>
</domain>
```

Szerencsére a kézi XML-fájl létrehozáson kívül vannak más lehetőségek is.

Ha már létezik egy libvirt által kezelt virtuális gépünk, akkor könnyű helyzetben vagyunk, mert a

```
virsh dumpxml VM-neve > vm1.xml
```

paranccsal legalább egy szintaktikailag megfelelő, a fontosabb információkat tartalmazó sablonhoz juthatunk.

Másik lehetőség, hogy ha – mint ebben a példában – készen van a parancssorból elindított QEmu/KVM virtuális gép (és adott a futtató parancssor, amivel a virtuális gépet indítjuk), akkor a virtuális gépet indító parancssort a virsh képes XML-fájllá konvertálni, azaz nem kell teljesen nulláról indulnunk. Tehát, ha a fenti indító parancsot egy startvm1.sh nevű fájlba tettük, akkor a

```
virsh domxml-from-native qemu-argv startvm1.sh
```

parancs eredménye az előző virtuális gép XML-formátumú leírása lesz. Ha a parancs kimenetét egy fájlba átirányítjuk, akkor akár azonnal használhatjuk is azt, vagy kézzel (vagy esetleg másik libvirt-eszközzel) később módosíthatjuk. (Annyit hozzáfűznénk, hogy a parancs még messze nem tökéletes, pl. a CPU-topológiát leíró `-smp 4,cores=2,sockets=1` paramétert nem ismeri, azt ki kell törölni ahhoz, hogy egyáltalán működjön a konverzió. A létrehozott XML-fájlból aztán nem lehetett létrehozni a virtuális gépet, mert hiányolta a kvm binárist. Azaz elindulásnak éppen jó lehet, de tökéletes eredményt ne feltétlenül kell várni tőle.)

Ha készen vagyunk a virtuális gépet leíró XML-lel, biztosítani kell, hogy a leírófájlban szereplő fájlok létezzenek, Mivel a diszket reprezentáló fájl még nincs meg, azt most létre kell hozni:

```
touch /path/to/vm1diszk1.img
```

és ha ez kész, már indíthatjuk is:

```
virsh create vm1.xml
```

Ez a parancs tulajdonképp két dolgot művel: létrehozza a virtuális gépet, és egyúttal el is indítja azt. Ha nem szeretnénk a létrehozás pillanatában el is indítani, akkor a létrehozás helyett csak definiálni kell:

```
virsh define vm1.xml
```

(A virsh define-nal definiált – és nem virsh create-tel létrehozott virtuális gépeket a későbbiek során a virsh start vm1 paranccsal lehet elindítani.)

Amennyiben a VM létrehozása/indítása (vagy bármely egyéb VM-művelet) hibaüzenetet ad, érdemes megnézni a naplófájlokat. Egyrészt van egy globális a libvirthez, másrészt van egy a konk-

rét virtuális géphez tartozó fájl. Az előbbi fájl (Ubuntu 12.04 LTS alatt) a /var/log/libvirt/libvirtd.log, míg a VM-specifikus a /var/log/libvirt/qemu könyvtárban található, a fájl neve megegyezik a VM nevével egy .log hozzáfűzésével (azaz mivel a példában vm1-nek hívtuk a gépet, ez /var/log/libvirt/qemu/vm1.log). Részletesen látható, hogy milyen műveletek hajtódtak végre, és pontosan mi is a hiba. Pl. ha nem hoztuk létre a virtuális diszkként működő fájlt (fenti touch parancs), akkor a 'No such file or directory' – vagy más nyelv beállítása esetén az adott nyelven a fájl hiányára utaló – hibaüzenet segíthet. Eléggé tipikusak a különböző jogosultságkezelésből adódó problémák. Sok Linux-terjesztés alapértelmezetten bekapcsolt állapotban szállít valamilyen hozzáférés-szabályozó alrendszert, mint pl. a Selinux, vagy épp az AppArmor. Ezek lehetnek oly módon konfigurálva, hogy a virtuális gépekhez tartozó fájlok (pl. a diszkek) csak bizonyos könyvtárakban érhetőek el. Előfordulhat, hogy a VM fájljainak saját fájljogosultsága nem teszi lehetővé a hozzáférést, vagy ad absurdum az, hogy a VM-hez tartozó diszkfájlt egy mások által nem kereshető könyvtárban hoztuk létre (pl. eléggé elterjedt, hogy a felhasználó saját könyvtárához másoknak semmiféle hozzáférése nincs, így a $HOME-ban létrehozott diszkkép fájlhoz a libvirt alrendszer sem fér hozzá).

A sikeres virsh define vagy virsh create paranccsal létrehozott (tehát álló vagy futó) virtuális gépünknek immár látszania kell a virtuális gépek listájában:

```
virsh list --all
```

A meglevő virtuális gépeinkkel pedig szintén a virsh paranccsal további műveleteket hajthatunk végre:

– egy futó gépet szabályosan leállíthatunk: virsh shutdown GÉPNÉV

– egy álló gépet elindíthatunk: virsh start GÉPNÉV

– egy futó gépet újraindíthatunk: virsh reboot GÉPNÉV

– egy virtuális gépet nem szabályosan leállítunk: virsh destroy GÉPNÉV

– majd pedig megszüntetése: virsh undefine GÉPNÉV

A felsorolt parancsok a jéghegy csúcsa, részletesebb leírás a virsh dokumentációjában (és a libvirt.org oldalon) találhatóak.

Lemezkezelés. Partíciók, LVM, RAID, md-eszközök

Nevezéktan

Linux alatt a lemezes háttértárakat, diszkeket blokkos eszközfájlokon keresztül érjük el. (Részletek az eszközfájlokról a Hardverek kezelése részben.) Külön eszközfájl van magához a teljes diszkhez, és a diszk egyes partícióihoz. A rendszerben levő diszkek és azok partíciói lekérdezhetők az

```
fdisk -l
```

paranccsal.

Régebben az ATA, ATAPI eszközök korában a merevlemezeket (hard disc drive) hdX és hdXY néven lehetett elérni:

– /dev/hda az első vezérlőn (primary) keresztül elérhető ún. master diszk
– /dev/hdb az első vezérlő ún. slave (szolga) diszkje
– /dev/hdc a második vezérlő (secondary) master diszkje
– /dev/hdd a második vezérlő slave diszkje

és így tovább.

A diszkeket a klasszikus PC-n megszokott particionálási mód szerint osztják logikai részekre, ebben van ún. elsődleges partíció (primary partition), és bővített partíció (extended partition), amelyet tovább lehet osztani logikai meghajtókra. Az elsődleges partíciók 1-4-ig számozottak, míg a bővített partícióban levő logikai eszközök 5-től fölfelé kapják meg a számokat. Így valamely diszken levő partíciók neve:

– /dev/hda1 a hda diszk első elsődleges partíciója
– /dev/hdc3 a hdc diszk harmadik elsődleges partíciója
– /dev/hdd6 a hdd diszk bővített partíciójában levő második logikai meghajtó

Ebben a furcsa elnevezési konvencióban a lukak természetesek: lehet, hogy nincs az elsődleges vezérlőn slave eszköz, attól még a második vezérlő master eszköze hdc lesz. Hasonlóan, lehet, hogy csak egy elsődleges és egy bővített partíció van a hdb diszken, a bővített partícióban 2 logikai meghajtóval, akkor /dev/hdb1, /dev/hdb5 és /dev/hdb6 lesz a nevük.

A SCSI, SATA, SAS vezérlőkön és az USB-n keresztül elérhető diszkeket hdXY helyett sdXY-nak hívjuk, ahol is az első periféria amit a rendszer felismer sda, a következő sdb, sdc, és í. t. néven jelenik meg. Azaz ma ez a jellemző:

– /dev/sda
– /dev/sdc3

A diszkeket használathoz először is particionálni kell, erre jó a parancssoros fdisk és parted parancsok, illetve a grafikus felületű gparted.

A particionálás során az egyes partíciók típusát is be kell állítani. Ezt linuxos fájlrendszer esetén a hexadecimális 83, míg a Linux által használt swappartíció esetén 0x82 értékre kell beállítani.

A diszkek egyértelmű azonosítása miatt egyre elterjedtebb egy, a fenti elnevezésnél bonyolultabb konvenció használata:

```
/dev/disk/disk-by-uuid/c5b875fc-50eb-41c4-af76-af37cead305a
```

Megjegyezni ugyan nem nagyon lehet ezt a fajta nevet, de mint az elérési útban látszik, alapja az adott eszközhöz tartozó UUID[9] – azaz teljesen egyedi név. Egyre inkább úgy tűnik, hogy ez a névkonvenció terjed az egyes terjesztésekben. Mivel viszont ilyen hosszú neveket senki nem szeret begépelni, általában elérhetők a diszkek a régi néven is. A két név közötti összerendelésre a

```
blkid
```

nevű parancs szolgál.

Kitérő - Tesztkörnyezet létrehozása

Amennyiben valaki a következőket ki szeretné próbálni, de nem célja az esetlegesen működő rendszerének szétrombolása, akkor egy lehetséges módszer a következő. Ha van működő Linux (e nélkül nem fog menni), aminek van egy kis szabad hely valamelyik fájlrendszerében, akkor használjuk ki az ún. hurok (loop) eszköz nyújtotta lehetőségeket. A /dev/loopX arra szolgál, hogy egy közönséges fájlt a rendszer számára virtuálisan úgy láttassunk, mintha valódi blokkos eszköz lenne. Használata:

– létrehozunk egy, a virtuális diszk adatait majdan tároló fájlt. Pl. a dd paranccsal:
  dd if=/dev/zero of=file0 bs=1M count=32
  Ekkor lesz egy 32 MB méretű, csupa 0-kkal feltöltött, file0 nevű fájlunk.
– ebből a fájlból megcsináljuk a virtuális diszkeszközt:
  losetup -v -f file0
  A -f opció hatására a losetup kikeresi az első szabadon használható eszközt a loop0, loop1, loop2, stb listából, és létrehozza a (pl.) /dev/loop0 nevű virtuális diszket. (Ha biztosan egy konkrét eszközt szeretnénk használni, a -f helyett megadhatjuk annak nevét is paraméterként a losetup-nak.) A file0 paraméter pedig azt adja meg, hogy melyik fájl játssza el a diszk szerepét. Valójában a loop0-n keresztül ezt a file0-t kezeljük, de immár diszkként. Fontos, hogy a továbbiakban a file0-t ne piszkáljuk, csak a loop-eszközt! A -v opció pedig megmutatja, hogy melyik eszközt konfigurálta fel. (Ez sajnos nem minden verziójú losetup paranccsal működik, ilyenkor a losetup -a paranccsal lehet lekérdezni a hozzárendelést.)

Ettől a pillanattól kezdve a rendszerben látszik egy /dev/loop0 néven elérhető újabb diszk. Ezzel a diszkkel aztán már ugyanúgy dolgozhatunk, mint a valódiakkal.

9. http://en.wikipedia.org/wiki/Universally_unique_identifier

Diszkek használata

A gépben levő valódi (vagy virtuális, mint az előzőekben létrehozott loopX-eszköz) diszkeket használhatjuk immár magukban (azaz rakhatunk rá fájlrendszert vagy konfigurálhatunk rá swap-et), de a Linux-kernelben elérhető funkciók használatával bonyolultabb virtuális diszk-alrendszereket is kiépíthetünk.

mdadm

Az ún. md (multi-device, multi disk) segítségével tetszőleges RAID-tömböket építhetünk. Az elterjedtebbek közül elérhető pl. RAID0 (stripe), RAID1 (mirror) és RAID5 – illetve ezek kombinációja. A konfiguráláshoz az mdadm nevű eszköz használatát kell elsajátítani. Nézzük sorban a lépéseket:

– fdisk-kel 0xfd típusú partíciókat csinálunk

```
fdisk /dev/sda
```

– mdadm-mel létrehozzuk a RAID-tömböt – a példában RAID1-et (a szögletes zárójelek között álló opciók ugyanazt jelentik, értelemszerűen csak az egyik megadására van szükség)

```
mdadm [ -C | --create ] /dev/md0 \
  [ -l 1 | --level=1 ] \
  [ -n 2 | --raid-devices=2 ] \
  [ -x 1 | --spare-devices=1 ] \
  /dev/sda1 missing missing
```

A fenti mdadm paranccsal létrehoztunk egy, a továbbiakban /dev/md0 néven elérhető, elvben 3 diszkből álló, tükrözött diszktömböt, amibe jelenleg csak egyetlen diszket konfiguráltunk be, az sda1-et – a két „missing” nevű diszk a szintaxis miatt szükséges. (Tesztkörnyezetünkben ez nyilván a loop0, majd a továbbiakban újabb és újabb loop-eszközöket használjunk ezek helyett.)

Ha lekérdezni szeretnénk, akkor használjuk a

```
mdadm --detail /dev/md0
```

parancsot, vagy pedig a

```
mdadm --detail --scan [--verbose]
```

formát (ez utóbbi esetben majd az mdadm megkeresi nekem, hogy milyen eszközök állnak rendelkezésre).

Mivel ez a tükör jelenleg eléggé féllábú, és nem kifejezetten az adatbiztonság netovábbja, egészítsük ki, adjunk hozzá újabb diszkeket. A további

```
mdadm [ --manage ] /dev/md0 -a /dev/sdb1
mdadm [ --manage ] /dev/md0 --add /dev/sdc1
```

parancsok eredményeként a virtuális md0 eszközünk szépen kiegészül ezzel a két diszkkel, teljessé válik a tükör és a tartalék is hadrendbe állt.

Amennyiben egy hardver meghibásodik, azt el kell távolítani, és jót rakni a helyére. Ezt a tesztkörnyezetben az

```
mdadm /dev/md0 -f /dev/sdb1
```

paranccsal tudjuk megtenni. (A -f opció hatására hibás (faulty) állapotba kerül a diszk.) Ekkor a tartalék a helyére lép, a háttérben megindul az adatszinkronizáció a jó és a tartalék diszk között. Ezt követően távolítsuk el a hibásat (-r, mint remove):

```
mdadm /dev/md0 -r /dev/sdb1
```

Majd tegyünk bele egy újat:

```
mdadm /dev/md0 -a /dev/sdd1
```

Ha akarjuk, ezeket akár mind, egyetlen lépésben is megtehetjük:

```
mdadm /dev/md0 -f /dev/sdb1 -r /dev/sdb1 -a /dev/sdd1
```

(Az utóbbi parancsokban egyébként mindenütt kihagytuk a --manage opciót. Mert megtehettük.)

lvm

A dm (device mapper) alrendszer segítségével igen rugalmasan használható virtuális diszkeket lehet létrehozni (akár az előzőekben tárgyalt dm-eszközön is). Főleg a hagyományos particionálás rugalmatlanságának kiküszöbölése a hasznos funkciója. Ha lehet egy javaslatunk, akkor: építsünk RAID tömböket a diszkjeinkből, majd az így létrejött tömbre rakjunk LVM-et, és csak utána akarjunk fájlrendszert rátenni.

Az LVM (Logical Volume Manager, logikai kötetkezelő) felépítése a következő: diszkből (ami persze RAID-tömb is lehet), partícióból ún. fizikai kötetet (physical volume) kell csinálni. Ebből aztán párat összefogunk, és kötetcsoportot (volume group) hozunk létre. Ez kb. egy virtuális diszknek felel meg. Ahogyan a valós diszkeket logikailag különálló részre bontjuk a particionálással, ugyanúgy kisebb részekre bontjuk a virtuális diszkeket is. Csak nem particionálunk, hanem logikai köteteket (logical volume) hozunk létre. Ezzel az utolsó lépéssel aztán készen vagyunk, a logikai kötet ugyanúgy használható, mint egy rendes diszkpartíció. Nézzük a lépéseket:

– fdisk-kel 0x8e típusú partíciókat csinálunk

```
fdisk /dev/sda
```

– pvcreate – a diszkpartícióból fizikai kötetet csinálunk

```
pvcreate /dev/sda1
```

– vgcreate – több PV-ből névvel ellátott kötetcsoportot csinálunk (létrejön a „virtuális diszk”)

```
vgcreate volgroup1 /dev/sda1 /dev/sdb1 /dev/sdc1
```

– lvcreate – VG-ből kivágunk egy – a példában 100 MB-os – szeletet (ez lesz a „virtuális diszkpartíció”)

```
lvcreate -L 100 -n logvol0 volgroup1
```

Készen is vagyunk: az eredmény a /dev/volgroup1/logvol0 nevű virtuális eszköz (más néven: /dev/mapper/volgroup1-logvol0).

Az így létrejött eszköz felhasználásra kész. Minden olyasmit csinálhatunk vele, amit valódi diszkkel megtehetnénk[10]. Viszont vannak speciális parancsok a kezelésére.

Ez a készlet a névből egyértelműen kikövetkeztethető típusú objektum jellemzőinek megjelenítésére, feltérképezésére szolgál:

10. http://linuxgazete.net/114/kapil.html

```
pvdisplay / vgdisplay / lvdisplay
pvs / vgs / lvs
pvscan / vgscan / lvscan
```

Az LVM diszkekkel néhány hasznos dolgot meg lehet csinálni, amit valódi diszkelle már nem

– valamely objektum eltávolítható, ha nincs használatban

```
lvremove / vgremove / pvremove
```

– Ha van szabad hely a kötetcsoportban, nem csak új logikai kötet hozható létre, de egy meglevőt ki is bővíthetünk – azaz átméretezhetjük azt. (Vigyázat, ez a rajta levő fájlrendszert nem méretezi át, az egy különálló lépés.) Ha pedig nincs szabad hely a kötetcsoportban, a kötetcsoportot is kibővíthetjük újabb fizikai diszkekkel.

```
vgextend/ l vextend
```

– Az előzők fordítottja is lehetőség. Azaz egy logikai kötet mérete csökkenthető, illetve egy üres fizikai kötet a kötetcsoportból eltávolítható.

```
vgreduce / lvreduce
```

Ha a logikus elnevezésű, egyes objektumok kezelésére szolgáló parancskészletet kicsit soknak tartanánk, ott van helyette a mindent tudó lvm parancs. Annak használata esetén is ismerni kell az LVM elvi felépítését, de parancsból csak egyet kell megjegyezni :-)

cryptsetup

Titkosított diszkeket is csinálhatunk, ehhez a dm-crypt alrendszer és az un LUKS szolgáltatja a hátteret. A kiindulási állapot egy diszk, amin szeretnénk az adatokat titkosítva tárolni. Ebben az esetben első lépésként bekonfiguráljuk:

```
cryptsetup luksFormat /dev/sda1
```

A titkosításhoz kér egy jelszót, a továbbiakban a jelszó nélkül az adatok nem nagyon értelmezhetők. Ezt csak egyetlen egyszer kell megtenni.

A használathoz a következő lépésben a bekonfigurált eszközt elérhetővé kell tenni:

```
cryptsetup luksOpen /dev/sda1 valami
```

Ekkor újra bekéri a jelszót, hisz hozzá szeretnénk férni a titkosított tartalomhoz. Ha sikerül, innentől kezdve megjelenik egy /dev/mapper/valami nevű virtuális diszkeszköz. Ugyanúgy használható, mintha valódi diszk lenne, csak épp a valódi diszket olvasva az adatok értelmezhetetlenek.

A következő lépést szintén egyszer, a legelső luksOpen után szabad csak megcsinálni. Teleírjuk a virtuális diszkünket nullákkal, azaz a valódi diszkünket titkosított nullákkal – így az egész diszk véletlenszerű adatokat fog tartalmazni.

```
dd if=/dev/zero of=/dev/mapper/valami
```

Az előkészítéssel végeztünk, használhatjuk az eszközt. Ha fájlrendszert akarunk, akkor mkfs-sel csinálunk rá egyet, ha swapként, akkor engedélyezzük a használatát. (Részletekért lásd a „Fájlrendszerek, swap” fejezetet.)

Amikor már nem óhajtjuk az eszközt használni, akkor

```
cryptsetup luksClose valami
```

paranccsal elengedjük. A későbbiekben valahányszor használni szeretnénk, akkor elegendő a cryptsetup luksOpen parancs és a jelszó begépelése, és az eszköz használatra kész. Amennyiben automatikusan szeretnénk rendszerindításkor elérhetővé tenni a titkosított eszközt, akkor a /etc/crypttab nevű fájlban fel kell venni a jellemzőit és máris automatikusan elérhetővé válik.

Fájlrendszerek típusai és kezelése, swap használata

A diszkekre, illetve a diszkpartíciókra vagy valamilyen adatot teszünk, vagy a fizikai memória kiegészítésére használjuk, swap-et konfigurálunk rá. Ha adatot akarunk tárolni, ahhoz jellemzően fájlrendszert kell létrehozni. Linux alatt a bőség zavarával küzd az átlag kezdő; annyiféle fájlrendszer van, hogy nehéz dönteni.

Fájlrendszerek

Talán a legelterjedtebb az ext2 / ext3 / ext4 család (és igen, volt ext nevű is, de az végképp kihalt). Ezek közül az ext3 és ext4 naplózó fájlrendszer, ami azt jelenti, hogy egy nem szabályos rendszerleállás után a fájlrendszer használható állapotba hozása elhanyagolható ideig tart[11]. Stabil, megbízható, a legnagyobb kereskedelmi Linux-terjesztés gyártója igen régóta ezt szállítja alapértelmezetten. Régebben az ext3-t, újabban az ext4-et – ez mindenképpen bizakodásra adhat okot. Rendkívül sokat tud, fájlokhoz pl. törlést meg felülírást megtiltó jellemzőt is lehet rendelni, ACL-ek és kibővített attributumok kezelése támogatott rajta (ez utóbbiakról részletesebben olvashatsz a “Jogosultságkezelés” c. részben).

A másik mostani sláger a btrfs. Egyetlen negatívumát lehet felhozni. Fiatal. Noha több terjesztésben is elérhető, mostanra már stabilnak is minősítették, egyelőre korából adódóan nyilván nincs annyira kitesztelve.

(Ezen kívül elérhető az IBM JFS-fájlrendszere, az SGI-féle XFS, a Sun ZFS-e, és noha már nem fejlesztik, még mindig sok rajongója van a ReiserFS nevű fájlrendszernek. A kereskedelmi kínálatból valószínűleg a valamikori Veritas VxFS fájlrendszere a legkomolyabb – de a sor folytatható.)

Mivel alapvetően nem kívánunk javaslatot tenni, legyen ez az alapelv: használjuk azt, amit az általunk választott terjesztés alapértelmezetten felkínál. Ha ilyen nincs, akkor általános célokra talán az ext4 javasolható, speciális esetekben viszont mindenképpen próbáljuk tüzetesen letesztelni az alternatívákat.

Fájlrendszer létrehozása

Fájlrendszerek létrehozására a mkfs parancs szolgál. Két paramétert igényel: az eszközt, amin a fájlrendszert létre kell hozni, és azt az információt, hogy milyen típusú fájlrendszert hozzon létre. Ez utóbbit a -t opció után kell megadni.

```
mkfs -t ext4 /dev/sdb1
```

Helyette gyakran használják a mkfs.FSTYPE jellegű parancsnevet:

```
mkfs.btrfs /dev/sdc2
```

11. Nem, most szólunk – a naplózó fájlrendszer nem azt jelenti, hogy nem lehet adatvesztés. Lehet. De legalább gyorsan kiderül.

(Némelyik fájlrendszernél használhatók e mellett mindenféle egzotikus nevek is, mint mondjuk az mke2fs; sejthetően az ext2-höz – meg a család többi tagjához is.)

Fájlrendszer csatolása

A fájlrendszeren tárolt adatok eléréséhez a fájlrendszert csatolni kell – azaz a már látható könyvtárfa egy könyvtárára – úgymond – rálógatni. Ez persze felvet egy érdekes kérdést – hogyan lesz elérhető a legelső könyvtár: a gyökérkönyvtár[12]. Egy fájlrendszer felcsatolása – angolul mountolása – a következő előkészületeket igényli: legyen egy ismert nevű eszközfájlon egy ismert típusú elkészített fájlrendszer (létrehozni az előző pontban említett mkfs paranccsal lehet), és legyen egy lehetőleg üres[13] könyvtár (létrehozni a jól ismert mkdir paranccsal lehet), amin keresztül a fájlrendszert a későbbiek során elérhetjük. Ha ezek adottak, akkor egyetlen parancs:

```
mount -t FSTYPE /dev/eszköz /könyvtár
```

után az adatok elérhetőek. Fenti parancsból a típus megadása elhagyható, ekkor majd a mount kitalálja, hogy milyen az a fájlrendszer. A csatolásnál megadhatók különböző opciók, amelyek egy része minden típus esetén érvényes (például: -r, vagy másként -o ro; jelentése csak olvasható módú csatolás) mások az adott fájlrendszer sajátosságainak beállítására szolgálnak (pl. ext4 esetén a "-o discard" opcióval kérhetjük, hogy a kernel adatblokk felszabadításakor küldjön erről értesítést a hardvernek – ez SSD használata esetén jöhet jól). Példa[14]:

```
mount -t ext4 -r -o discard /dev/sdb1 /backup
```

A továbbiakban a /dev/sdb1 partíción levő EXT4-típusú fájlrendszer tartalma elérhető a /backup könyvtáron keresztül.

Fájlrendszerek lecsatolása

Egy felcsatolt fájlrendszer csatolásának megszüntetésére az umount parancs szolgál. (Hiába hallatszik úgy, nincs a parancs nevében az "u" után egy "n".)

```
umount /backup
umount /dev/sdb1
```

Mi fogja a fájlrendszert?

A lecsatoláshoz az szükséges, hogy a fájlrendszer éppen ne legyen használatban[15]. Ha mégis megpróbáljuk, akkor a "Fájlrendszer elfoglalt" (Filesystem is busy) hibaüzenetet kapjuk. Ekkor érdemes megkeresni, hogy mely processzek használják:

```
fuser -c /backup
```

A kimenetben látható PID-ekkel azonosított folyamatokat érdemes vallatóra fogni, hogy mi dolguk az adott fájlrendszerrel (esetleg meg is szüntethetjük őket; erre pl. a fuser -k opciója is jó lehet).

12. A kernel teszi láthatóvá. Nyilván nem csatolhatja, mert nincs hova, De úgy mutatja, mintha.
13. Nem kell üresnek lenni, de amíg a fájlrendszer fel van csatolva, addig a könyvtár eredeti tartalma nem látható.
14. Mi a logikai hiba a példában?
15. Ha erről a fájlrendszerről indítottunk el egy programot; ha valamilyen program ezen a fájlrendszeren megnyitott (és még mindig nyitva tart) egy fájlt; ha egy process ennek a fájlrendszernek valamelyik könyvtárában áll – no ezek tipikusan azok a helyzetek, amikor a fájlrendszer foglalt.

Fájlrendszer-ellenőrzés

Szabálytalan rendszerleállás esetén nagy eséllyel kimarad a fájlrendszer szabályos lecsatolása. Mivel a fájlrendszerek jellemzően pufferelt adateléréssel dolgoznak, ennek egyenes következménye némi adatvesztés lesz, de legalábbis inkonzisztencia. Ezeket javítani illik. Erre szolgál az fsck (fájlrendszer-ellenőrző, File System Check) nevű parancs. Csak fel-nem-csatolt fájlrendszer esetén adjuk ki, használata:

```
fsck -t ext4 /dev/sdb1
```

Szerencsére ritkán van rá szükség.

Automatikus csatolás

Ha azt szeretnénk, hogy a fájlrendszer automatikusan rendelkezésre álljon a rendszer újraindulása után is, akkor fel kell venni az adatait a /etc/fstab nevű fájlba. A fájl 6 db, szóközökkel, tabulátorokkal határolt mezőből álló sorokat tartalmaz, egy sor egyetlen fájlrendszer leírására szolgál:

```
eszköz csatolási_pont fs-típus mount-opciók dump-szint fsck-passno
/dev/sdb1 /backup ext4 ro,discard 0 2
```

A mezők sorban:

– a csatolandó fájlrendszert tartalmazó eszköz neve
– a csatolási pont
– a fájlrendszer típusa
– az extra csatolási opciók
– a dump-szint: a dump névre hallgató mentőprogram számára szükséges mentési szint megadása. (Ezt a programot Linux alatt amúgy nem jellemzően szokták használni, ezért legtöbbször a példában is látható 0 értéket írják ide.)
– az fsck-passno paraméter a rendszerindítás során automatikusan lefutó fsck számára a fájlrendszerek ellenőrzésének sorrendjét határozza meg. Vigyázzunk, mert a kihagyott érték 0-nak, azaz nem-ellenőrizendőnek számít!

Mennyi helyem van?

Rendszeresen vissza-visszatérő kérdés: mennyi helyem van. A válasz a df (disk free space) paranccsal kapható meg:

```
df
```

Az elérhető fájlrendszerek mérete, szabad és elfoglalt területeinek nagysága látszik a kimenetben. Napi használatban jellemzően szeretik a df -h opcióját (ún. human readable), ekkor nem nekünk kell a megjelenő számértékeket a kicsit könnyebben kezelhető MB, GB mértékegységekre átszámolni. Bizonyos fájlrendszerek esetén meglepő tud lenni, hogy a szabad és foglalt területek összege nem adja ki a teljes méretet – ennél már csak az meglepőbb, ha 100%-nál nagyobb telítettséget kapunk. Tudni kell, hogy van egy ún. “fenntartott” terület, amit közönséges felhasználók nevében futó program nem, csak rendszergazda jogú alkalmazások használhatnak el – ez pl. az EXT-családnál alapértelmezetten 5%. (Ezen terület nagyságát a fájlrendszer létrehozásakor – mkfs – lehet állítani. Illetve van egy tunefs – Ext-fájlrendszereknél tune2fs – nevű parancs, utólag azzal is lehet szabályozni a méretet.) Fent említett anomáliák erre vezethetők vissza

Minden fájlhoz – típustól függetlenül – tartozik egy un . i-node. Ebben az adatstruktúrában tárolódnak a fájl különböző adminisztratív adatai, mint pl. tulaj, jogok, méret, típus, stb. (Hard linkek közös i-node-ot használnak.) Az i-node-okhoz tartozik egy ún. i-node-tábla, amelynek mérete némelyik fájlrendszertípus esetén fix, a fájlrendszer létrehozásakor határozható meg. Ha nagyon sok nagyon apró fájlunk, vagy millió üres könyvtárunk van, ez a tábla is betelhet. Ennek ellenőrzéséhez a

```
df -i
```

parancsot kell használni. (És ha betelik, üres könyvtárakat, fölösleges fájlokat kell törölni.) Jellemzőbb a diszkhely fogyása, nem csoda, hogy külön parancs van annak megjelenítésére, hogy egy adott könyvtár fájljai mennyi helyet foglalnak, ez a

```
du
```

(azaz disk usage). Ha tehát egy fájlrendszeren fogy a hely, érdemes ezzel megkeresni, hogy hol, melyik könyvtárban levő fájlok foglalnak el sok helyet. A megoldás a problémára a feleslegleges fájlok törlése, esetleg a ritkán használt fájlok tömörítése. (Ha az i-node-ok fogytak el, akkor a problémás könyvtár megkeresésére egy megoldás található az “Alapvető hibaelhárítás” c. fejezetben.)

Fájlrendszer átméretezés

Ha elfogy a hely a fájlrendszeren, hosszú távon érdemes megnövetni a méretet. Ehhez természetesen előbb a tárolóterületet kell megnövelni (ez LVM esetén nagyon egyszerű, a “Diszkek” c. fejezetben szereplő lvextend paranccsal megtehető). Ha tehát a tárolóterület mérete nagyobb, mint a fájlrendszeré, akkor jöhet a fájlrendszer átméretezése. Növelni gyakorlatilag mindegyik elterjedt linuxos fájlrendszert akár működés közben is lehet, bár eltérnek a művelet végrehajtásához szükséges parancs nevében. Az eddig a példákban prefrált EXT-család átméretezéséhez a resize2fs parancs tartozik. Használata igen egyszerű:

```
resize2fs /dev/sdc1
```

(Csak az érdekesség kedvéért, az eddigi példákban méltatlanul hanyagolt XFS esetén xfs_growfs a parancs neve, míg pl. JFS esetén egyszerűen újra kell csatolni a fájlrendszert a mount parancs -o remount,resize paraméterével.)

Swap

A gépben levő fizikai memória véges, és sajnos szinte sosem elegendő. A modern operációs rendszerek előszeretettel alkalmazzák azt a módszert a memória takarékos használatára, hogy bizonyos memóriaterületeket a fizikai memóriából kimentenek a diszk erre fenntartott részére (aztán később vissza). Hagyományosan ezt swap névvel illetik, bár ma már pontosabb lenne az ún. paging (lapozás) kifejezés használata. Ahhoz, hogy a swap működjön, először ki kell jelölni egy önálló, csak erre a célra fenntartott diszkterületet, és előkészíteni a használatra. Erre szolgál a

```
mkswap /dev/sdd2
```

parancs[16]. Ezt egyszer kell megcsinálni. Az előkészített diszkterületet használatba venni pedig a

```
swapon /dev/sdd2
```

16. Érdekes, hogy az egyéb UNIX, és UNIX-szerű rendszerekben ennek az előkészítésnek nincs igazán megfelelője.

paranccsal lehet. A használt swap területre vonatkozó adatokat (és egyébként a gépben levő memória nagyságát is) a

```
free
```

paranccsal lehet lekérdezni, de használható (csak a swap-re) a

```
swapon -s
```

is. Ha pedig egy már nem használt swap-et el szeretnénk távolítani, akkor a

```
swapoff /dev/sdd2
```

paranccsal kapcsolhatjuk ki a használatát. A swap bekapcsolása hasonló a fájlrendszerek csatolásához: a rendszer leállásakor elvesznek az ezzel kapcsolatos információk. Ha tehát egy swap területet automatikusan használatba szeretnénk venni, ugyanúgy mint a fájlrendszereket, fel kell venni a /etc/fstab fájlba.

```
/dev/sdd2 none swap sw 0 0
```

(Mint látható, mivel ez nem fájlrendszer, ezért néhány mezőnél speciális paramétereket kell megadni.)

Az operációs rendszer telepítése

A szerverek a hozzájuk megvásárolt operációs rendszert is ritkán tartalmazzák előre telepítve, még ritkábban fordul ez elő az ingyenes Linux-disztribúciókkal. Ennek oka kézenfekvő: míg egy lakossági eszköznél a felhasználók jelentős részének megfelel az előre telepített operációs rendszer, szerver környezetben olyan döntéseket kell hoznunk a telepítést megelőzően, amelyek a telepítés menetét érdemben befolyásoljuk. Az informatikai környezetek is egyre több operációs rendszert futtatnak az évek során (először az árak csökkenése, majd a virtualizáció térnyerése miatt), így ma már nem jellemző, hogy egy rendszergazda kikerülheti az operációs rendszer telepítését.

Az operációs rendszer telepítése nagy vonalakban egy a telepítést végző rendszer memóriába töltéséből, a gép hardvereinek felderítéséből, a telepítési cél kiválasztásából és előkészítéséből, a hálózat, az óra, a felhasználók beállításából, az új rendszer komponenseinek felmásolásából és a rendszerbetöltő telepítéséből áll. A folyamatot számos módon végezhetjük – ezek közül hosszú ideig az optikai lemezekről történő telepítés volt a domináns. Mivel az újabb szervereken csak elvétve találunk optikai meghajtót, ezért ezt a lehetőséget jelentős részben kiváltotta az USB kulcsokról és a hálózatról való telepítés.

A Linux-telepítőknek két nagy csoportja van. Az egyik, főleg asztali rendszereknél elterjed módszer a lemezképes telepítés, amely egy telepítés nélkül is elindítható (live) rendszerből áll, amely képes önmagát fölmásolni a merevlemezre. A másik, hagyományos módszer pedig egy csupaszított Linux, amely az új rendszert a csomagkezelő segítségével telepíti. Előbbi előnye a sebessége, utóbbié pedig a szélesebb körű konfigurációs lehetőségek.

Fontos megemlíteni a virtualizációt vagy sok hasonló gépet üzemeltető környezetekben gyakori klónozásos eljárást, amely egy telepített, bekonfigurált mestergép lementett lemezképéből másolással vagy valamilyen differenciális tárolási megoldással hoz létre példányokat.

Végül több megoldás terjedt el lemez nélküli szerverek üzemeltetésére, amikor az egyes gépek a rendszerbetöltést hálózatról memóriába végzik, majd a szükséges tárterületet NAS vagy SAN megoldással érik el.

A telepítés előtt

A rendszer telepítése előtt mindenek előtt válasszuk ki a disztribúciót, annak verzióját és változatát. Szerverek esetében ha bizonytalanok vagyunk a kérdésben, nem tévedhetünk nagyot azzal, ha az általunk vagy az intézmény által előnyben részesített disztribúció utolsó stabil, hosszú távú támogatású verzióját (Ubuntu esetén „LTS”), és annak a Server változatát telepítjük.

Válasszuk ki a megfelelő architektúrát is. Modern x86 alapú rendszerek esetén gyakorlatilag nem lehet okunk a 32 bites változat telepítésére, nyugodtan válasszuk a „64 bit”, „x86-64”, „amd64” vagy „X64” jelölésű kiadást (ezek az elnevezések mind ugyanazt jelentik).

Gondoljuk át, hogy milyen tárolóeszközre kívánjuk telepíteni a rendszert. Szükség esetén alakítsuk ki a hardveres RAID-konfigurációt, hozzuk létre a SAN-on a köteteket.

Készítsük elő a gép hálózati elérését, szükség esetén osszunk ki neki, vagy igényeljünk IP címet, tudjuk meg az alhálózat adatait (hálózati maszk, átjáró és névkiszolgáló címe).

Gondoljuk át, hogy a telepítést hányszor kell elvégezni. Néhány gépnél több esetén keressünk megoldást az automatizálásra.

Ha a lemezeken már van bármilyen, korábban használt rendszer vagy adat, akkor készítsünk róla mentést – akkor is, ha nem szándékozzuk törölni a telepítés során.

Telepítés optikai lemezről, USB kulcsról

Az ingyenes disztribúciók letöltése nem okozhat különösebb gondot, a rendszer honlapján meg fogjuk találni a megfelelő hivatkozásokat. Amennyiben nem szabadon elérhető rendszert szerzünk be, kövessük a forgalmazó tájékoztatását.

A legtöbb esetben CD- vagy DVD-képet tölthetünk le, amelyet a .iso kiterjesztésről ismerünk fel. Amennyiben optikai lemezről kívánjuk elvégezni a telepítést, nincs más dolgunk, mint a lemezt egy megfelelő méretű üres adathordozóra kiírni. Ezt a legtöbb Linux rendszeren a fájlra jobb gombbal kattintva és a „lemezre írás” opciót választva, Windows alatt például az ingyenesen elérhető Infra Recorder nevű alkalmazással tehetjük meg.

Amennyiben USB kulcsról telepítenénk, a lemez kiírásához speciális szoftverre lesz szükségünk. Linux alatt ez az usb-creator, míg Windows alatt a UNetbootin nevű alkalmazással a legegyszerűbb.

A telepítés indításához mindkét esetben helyezzük be a gépbe a telepítő médiát, és indítsuk újra. A gép kézikönyvében szereplő módon válasszuk ki a rendszerindításra szolgáló eszköznek (boot device) az optikai meghajtót vagy az USB kulcsot. Számos esetben ezt az F12 billentyű megnyomásával tehetjük meg.

A telepítés menete

A telepítő elindulásakor általában elsőként ki kell választanunk egy menüben, hogy telepíteni (install) szeretnénk a rendszert. Ne ijedjünk meg, ha ezután hosszabb ideig várnunk kell, ilyenkor töltődik be a telepítéshez szükséges alaprendszer.

A telepítő betöltése után egy varázsló jellegű felület fogad minket, amely az első lépésként kiválasztott nyelven végigvezet minket a telepítés menetén.

A telepítés során számos kérdésre kell válaszolnunk. Az egyik első kérdéscsoport a helyi beállításokra vonatkozik: ki kell választanunk a rendszer alapértelmezett nyelvét, a billentyűkiosztást, valamint az országot vagy az időzónát. Egyes rendszerek megkérdezik, hogy helyi idő vagy UTC szerint jár a gépünk hardveres órája – csak akkor válasszuk a helyi időt, ha az adott gépen Windowst is futtatni szeretnénk. Az órát célszerű pontosan beállítani, és az automatikus időszinkronizációt bekapcsolni (lásd az Egy szerver alapvető beállításai című fejezetet).

A gépünk felhasználóival kapcsolatban is válaszolnunk kell néhány kérdésre. A rendszerek egy része létrehoz egy általános felhasználót, akinek joga van adminisztratív feladatok végrehajtására. Más rendszerek (Debian, CentOS) egy külön, root nevű felhasználónak állítják be az általunk megadott jelszót. Az általános célú felhasználónak mindkét esetben célszerű, hogy a saját nevünket adjuk, és az esetleges további felhasználókat majd később hozzáadhatjuk (lásd a Felhasználók és csoportok című fejezetet.)

A telepítők elvégzik a hálózat alapvető beállításait is. Több hálózati interfész esetén ki kell választanunk azt, amelyen az internetet érjük el – ezt jobb megoldás híján a kártya típusa és MAC címe alapján kell megtennünk. Amennyiben a hálózaton automatikus konfigurálás (DHCP) működik, azt a telepítő automatikusan fölismeri és használja. Ha fix beállításokat szeretnénk megadni, akkor azt a kérdésekre válaszolva tehetjük meg. Ebben az esetben a gép IP címére, az alapértelmezett átjáróra és az alhálózati maszkra, valamint a névkiszolgáló címére lesz szükségünk (bővebb útmutatás az Alap hálózati infrastruktúra című fejezetben).

A telepítés legnagyobb figyelmet igénylő pontja a lemezek beállítása. A legtöbb esetben a telepítő fölajánl néhány szövegesen körülírt beállítást, valamint az egyéni választás lehetőségét. Mindenek előtt döntsük el, hogy szükségünk van-e szoftveres RAID-re vagy LVM-re, valamint hogy van-e speciális igényünk a kötetekkel kapcsolatban (részletes ismertető és tanácsok a Lemezkezelés című fejezetben).

Csomagkezelés

A legtöbb Linux terjesztésben a használt programok ún. csomagokban érhetők el, és az adott csomag aztán kényelmesen telepíthető vagy szedhető le a gépről, ha a későbbiekben fölöslegesnek bizonyul. Két nagyon elterjedt csomagformátum létezik, de mellettük több kevésbé elterjedten használt disztribúció használ sajátot.

Az egyik, az eredendően a RedHat által kifejlesztett RPM (RedHat Package Manager) formátum. Kezeléséhez egy egyszerű parancssoros eszközt biztosítanak, ennek neve szintén rpm. Ezt a csomagformát használja többek között a kereskedelmi RHEL – Redhat Enterprise Linux Server (és az ezek klónjaként létező CentOS, és Scientific Linux), a Fedora, a szintén kereskedelmi SLES – Suse Linux Enterprise Server, és az OpenSuse. (Valamint pár kisebb terjesztés.)

A másik nagyon elterjedt csomagformátum a Debian terjesztéshez kötődik, neve DEB. A kezelőparancs neve dpkg. A DEB-formátumot használja nyilván a Debian, és az abból kinövő Ubuntu (és természetesen az összes Ubuntu gyerek: Xubuntu, Lubuntu, Kubuntu, stb.), illetve néhány más Ubuntu származék, mit pl. a Linux Mint.

Általában az egyes csomagformátumok – noha belülről teljesen másként néznek ki, hasonló funkcionalitást nyújtanak: valamilyen formában tárolódnak benne az adott csomagot alkotó fájlok (pl. az RPM-ben egy CPIO-archívum, míg a DEB-ben egy AR-archívumban), ezen kívül a csomaghoz tartozó különböző metainformációk.

Csomag metainformációk

A csomag jellemzői közé tartoznak a felhasználók számára fontosabb (pl. milyen fájlok alkotják az adott csomagot) és kevésbé fontos adatok (látszólag ide tartoznak pl. az egyes fájlok sértetlenségének ellenőrzésére használható különböző ellenőrző-összegek, hash-értékek).

Fontos szempont egy csomagkezelő rendszer és egy csomagformátum „jóságával” kapcsolatban, hogy milyen lehetőségeket nyújt. Azaz:

– a korábban telepített csomagot könnyedén le lehet-e szedni a rendszerből
– van-e lehetőség a csomagok frissítésére
– van-e függőségkezelés (azaz jelzés arra, hogy „A” csomag csak valamely nem ebbe a csomagba tartozó komponens megléte esetén működőképes, a nélkül esetleg nem is lehet, vagy csak nem érdemes telepíteni). Van olyan csomagformátum, amely fájlokra, és olyan is, amelyek csomagokra vonatkozó függőségkezelést ismer.
– van-e ütközésvizsgálat (az előző ellentéte: csak akkor lehet telepíteni „A” csomagot, ha egy másik komponens nincs telepítve a rendszerben).
– a csomagok megbízhatóságának kérdése (el vannak-e, vagy egyáltalán elláthatóak-e a csomagok pl. digitális aláírással, amely biztosítja a csomagok sértetlenségét)

DEB-csomagok alapszintű kezelése

Ha a kezünkben van (telepítő CD-n, DVD-n, vagy hálózatról letöltött formában) egy *.deb fájl, akkor egyszerűen

```
dpkg -i csomag.deb
```

parancs segítségével telepíthető. Ha szükséges lekérdezni, hogy milyen csomagok vannak a rendszerben telepítve, akkor pedig a

```
dpkg -l
```

teszi ezt lehetővé. Ha körülbelül sejtjük a csomag nevét, akkor szűrhetjük a keresést a

```
dpkg -l zen\*
```

formával.

Amennyiben az érdekel, hogy egy adott nevű, telepített csomag milyen fájlokat tartalmaz, akkor a

```
dpkg -L zenity
```

paranccsal kaphatjuk meg a kívánt információt. Néha hasznos lehet a fordítottja: megtalálni, hogy egy konkrét fájl mely csomag részeként települt a rendszerbe:

```
dpkg -S /usr/bin/test
```

És végül, ha eltávolítani szeretnénk egy csomagot, akkor töröljük le:

```
dpkg -r csomagnév
```

Sajnos ez utóbbi művelettel van egy kis probléma. A csomagokhoz tartozhatnak konfigurációs adatok. A telepítés során ahol szükséges, lefut egy konfiguráló lépés, és a fenti -r opciójú törlés ezt az adatot a rendszerben fent hagyja. (Ennek van jó oldala is.) Amennyiben ezekre az adatokra már nincs szükség, a telepített csomag eltávolításakor érdemes ezen formák valamelyikét használni

```
dpkg -r --purge csomagnév
dpkg -P csomagnév
```

a törléshez.

RPM csomagok alapvető kezelése

A kezünkben levő *.rpm fájlok telepítése hasonlóan egyszerű:

```
rpm [ -i | -U | -F ]csomag.rpm
```

Az install (-i opció) művelet mellett létezik az upgrade (-U) és a freshen (-F) verzió is: az install feltelepít egy csomagot, és az esetleges korábbi verziója ugyanannak a csomagnak megmarad. Az upgrade esetén a korábbi verzió eltávolításra kerül a telepítés során, de ha nem volt fent korábbi verzió, az se baj, települ az új. Ezzel szemben freshen esetén csak akkor települ az adott csomag, ha volt fent korábbi verzió (ekkor viszont azt az upgrade-hez hasonlóan frissíti.) Jellemzően az install helyett az upgrade-et preferálják (nem nagyon van értelme ugyanabból a programból egynél több verziót feltelepíteni, kivételként általában csak a kernel-csomagokat emlegetik).

A feltelepített csomagok listája lekérdezhető a

```
rpm -qa
```

paranccsal (ahol az “a” opció az “all”, minden csomag jelentéssel bír.) Adott nevű csomag telepített állapotának ellenőrzése:

```
rpm -q zenity
```

Egy telepített csomaghoz tartozó fájlok listázása

```
rpm -ql zenity
```

A “melyik csomag telepítésétől lett ez a fájl” kérdésre pedig az

```
rpm -q -f /usr/bin/test
```

parancs adja meg a választ.

RPM alapú rendszer esetén is lehet eltávolítani egy csomagot, ehhez az

```
rpm -e csomagnév
```

parancs használható.

Fenti dpkg és rpm parancsok használatának komoly hátulütője, hogy a függőségkezelés csak félig működik. RPM esetében jellemzően fájl alapú függőségek szerepelnek a metaadatok között, így valahogyan meg kell keresni, hogy a függőségi fájl melyik csomag része ahhoz, hogy a csomagot begyűjthessük, és telepíthessük.

Magasabb szintű csomaghasználat: repók

A rendes függőségkezelést a repók (repository, csomagtároló) bevezetése jelentette. A repókban csomagot tárolunk (és csomag metaadatokból felépített adatbázisokat). A repók lehetnek lokális könyvtárstruktúrák, vagy akár ftp-vel, http-vel elérhető szerveren található távoli tárolók. A repókezeléshez újabb parancsok jelentek meg.

A DEB-alapú rendszerekhez ez lett az APT (A Package Tool). Jellemzően a /etc/apt könyvtárban található az APT konfigurációja. Ezek olyan jellemzők, mint hogyan érjük el a külső szervereket (pl. a proxy adatok), és milyen tárolókat használunk a csomagok keresésekor.

Az APT készlethez tartozik a leggyakrabban használt apt-get nevű parancs, meg az apt-cache (de csomagok telepítésével újabb parancsokkal bővíthető). Ezek parancssoros eszközök, de van aki jobb szereti a karaktergrafikus aptitude-ot.

Az RPM csomagformátum esetében a repó-alapú csomagkezelés kicsit bonyolultabb. A Red Hat vonal egy YUM (Yellowdog Update Manager[17]) nevű eszközt választott, ahol a parancs neve: yum. A Suse-világ e helyett hivatalosan a Zypper nevű eszközt használja. A parancs neve szintén zypper. Ezekhez létezik alternatíva is, talán a legismertebb az Apt4RPM nevű eszköz, amely a DEB-ekhez kifejlesztett APT-t igazította az RPM csomagformához. (Szerzők ismernek olyan kellően nagy rendszert, ahol a Zyppert a YUM váltotta, máshol az Apt4RPM-re cserélték le.)

Funkcióit tekintve nem nagyon térnek el egymástól az egyes eszközök – természetesen van olyan, amit csak az egyik, a vagy másik tesz lehetővé, és természetesen a parancsok neve és paraméterezése is eltér egymástól. Így a folytatásban lehet hasonlóságokat találni (és persze eltéréseket is).

17. A Yellowdog Linux terjesztés fejlesztői alkották meg, onnan jött a név

APT alapszintű használata

Az APT működésének főbb paraméterei az /etc/apt/apt.conf fájlban állíthatók be (pl. proxy elérés). További fontos az ún. repók – beállításuk a /etc/apt/sources.list nevű fájlban történik. Ha fentieket beállítottuk, akkor onnantól egy csomag telepítés egyszerű:

```
apt-get install csomagnév
```

Az APT megkeresi, hogy melyik repóban található meg az adott csomag, ellenőrzi a függőségeit, letölti a csomagot és a függőségeket, és szépen sorban feltelepíti őket. A függő csomagokhoz mellékesen beállítja azt az információt, hogy függőségként kerültek fel, meg azt, hogy mely csomag függőségeként. Az, hogy ez egy függősége valaminek, az azért érdekes, mert ha az összes olyan csomagot eltávolítjuk, amelynek ez a csomag függősége volt, akkor ez a csomag maga is eltávolíthatóvá válik. Előző lépés

```
apt-get remove --purge csomagnév
```

vagy

```
apt-get purge csomagnév
```

paranccsal, míg az “árván” maradt csomagok eltávolítása az

```
apt-get autoremove --purge
```

paranccsal intézhető el.

A csomagoknak időnként újabb verziói jelennek meg. Ezeket a frissítéseket két lépésben érhetjük el. Az

```
apt-get update
```

parancs frissíti a csomagokra (és verziójukra) vonatkozó adatbázist, míg az

```
apt-get upgrade
```

maguknak a telepített csomagoknak a frissítését intézi el. Mivel a csomagok telepítése és frissítése során elég sok csomag kerülhet be a lokális csomag-cache-be (helye /var/cache/apt/archives könyvtár), ezért érdemes rendszeresen lefuttatni az

```
apt-get autoclean
```

parancsot, ami a már felesleges csomagfájlokat törli a cache-ből, illetve ha nagyon szűkösen állunk hellyel, akkor akár a

```
apt-get clean
```

parancsot is használhatjuk, amellyel az egész lokális cache kiüríthető

A feltelepített csomagokkal kapcsolatos különböző információk lekérdezhetők az apt-cache nevű paranccsal. Pl. mely csomagoktól függ (és mikkel ütközik) egy adott csomag:

```
apt-cache depends binutils
```

vagy a fordítottja, az adott csomagtól mely más csomagok függenek (hivatalos neve reverse dependency)

```
apt-cache rdepends binutils
```

ha telepítjük az apt-file nevű csomagot, kapunk néhány érdekesebb funkciót is. Például egy csomag tartalmát a nélkül ki tudjuk listázni, hogy magát a csomagot le kellene tölteni. Persze ez nem megy magától, ehhez az apt-file adatbázisát is rendszeresen frissíteni kell:

```
apt-file update
```

Ha megtörtént a frissítés, akkor már mehet a listázás:

```
apt-file list csomagnév
```

Ugyanígy kereshetjük azt is, hogy egy konkrét fájl mely csomagban van:

```
apt-file search /usr/bin/nvi
```

YUM alapszintű használata

A YUM is konfigurálandó. Ehhez a paramétereket a /etc/yum.conf, a repókat pedig a /etc/yum.repos.d könyvtárban levő fájlokkal kell beállítani. A konfigurálás után a yum használható

Csomagtelepítés yum segítségével:

```
yum install csomagnév
```

A telepített csomag eltávolítása:

```
yum remove csomagnév
```

A telepített csomagok frissítése:

```
yum upgrade
```

Természetesen, mivel YUM alatt is létezik függőségkezelés, itt is előfordulhat, hogy árván maradnak csomagok, ezek eltávolíthatóak a

```
yum autoremove
```

paranccsal.

A telepítések során letöltött fájlok a /var/cache/yum könyvtárban gyűlnek, takarítása a

```
yum clean packages
```

paranccsal történik.

Ha valamely csomagot keresünk, akkor használható a

```
yum whatprovides /usr/bin/test
```

illetve a

```
yum search coreutils
```

parancsok.

Függőségek listázására jó a

```
yum deplist csomagnév
```

parancs.

Zypper alapszintű használata

Konfigurálás: /etc/zypp/zypper.conf és /etc/zypp/repos.d

Csomagtelepítés:

```
zypper install csomag
```

Csomageltávolítás:

```
zypper remove csomag
```

Frissítés: az adatbázis frissítése (ez repónként automatizálható)

```
zypper refresh
```

És a csomagok frissítése:

```
zypper update
```

Cache (helye: /var/cache/zypp/packages) ürítése

```
zypper clean
```

Keresés

```
zypper search coreutils
zypper what-provides coreutils
```

Mint fentiekből látható, a repón alapuló eszközök alapfunkciói meglehetősen hasonlóak, időnként még az alparancs neve is megegyezik. Sajnos az apró különbségek a heterogén rendszert üzemeltetők számára kicsit kellemetlenné teszi a dolgot.

Az egyes csomagkezelő eszközökről fent felvázoltak persze nem tekinthetők teljes körű információnak, azaz a használt rendszer függvényében érdemes jobban elmélyedni a dokumentációban.

Alap hálózati infrastruktúra

Ahhoz, hogy linuxos számítógépünk képes legyen a hálózat többi gépével kommunikálni, minimálisan szükségünk van egy hálózati interfészre és a hálózati paraméterek megfelelő beállítására. Minden gépben van egy speciális, virtuális interfész a lo[18], amit loopback néven is szoktak emlegetni, a hozzá tartozó IP-cím: 127.0.0.1. (Ez jellemzően automatikusan be van konfigurálva.) Ezzel akár valódi hálózati kapcsolat nélkül is lehet hálózati kommunikációt végző programokat futtatni/tesztelni a gépen. Linux alatt az Ethernet-interfészek jellemzően eth0, eth1, stb. néven érhetők el (ez pl. a legújabb Fedora terjesztésben már megváltozott), míg a Wifi-interfész jellemzően wlan0.

A hálózati interfészeket szervereken jellemzően statikusan szokták konfigurálni, míg a desktop gépeken elterjedtebb a DHCP-szerver segítségével dinamikusan kiosztott IP-címek használata. A statikus konfiguráláshoz a UNIX-világból örökölt ifconfig volt használatban, ma viszont egyre inkább egy sokat tudó új eszköz, az ip nevű parancs javasolható. (Ez sok esetben egy iproute – vagy iproute2 – nevű csomag részeként kerül fel a gépre.) A folytatásban mi is ez utóbbit fogjuk használni, de odaírjuk a régebben elterjedtebb formát is. Oka: sokan vannak – főleg egyéb *X-rendszerről érkezők, vagy heterogén rendszereket üzemeltetők, akik a hagyományos eszközöket szeretik a mai napig használni; ráadásul a net tele van ezeket a régebbi eszközöket használó leírásokkal.

Fontos tudni, hogy mint a legtöbb adminisztrációs tevékenység, a hálózat konfigurálása is rendszergazda jogosultságot igényel. E miatt minden parancs esetén mindig mindenki figyeljen, hogy az általunk megadott parancsot sudo parancs esetleg su -c ‘parancs’ formában, vagy bármilyen a használt terjesztésben rendelkezésre álló eszközzel, vagy a céges előírásokban javasolt módon, de a lényeg, hogy rendszergazdaként hajtsa végre.

Az IP parancs

Az ip parancs egymaga szolgál a hálózati interfészek konfigurálására, az útválasztó tábla beállítására, különböző speciális hálózati eszközök (pl. bridge, vagy aggregált interfész) létrehozására és még ezernyi másra. Szintaxisa e miatt kicsit bonyolult, időnként kevéssé intuitív (sőt néha logikátlan), de hamar megszokható. A parancsnak megadhatunk különböző opciókat, mint pl. -o (oneline) egysoros kimenet, -d (detail) részletesebb információk, -s (statistics) forgalmi és egyéb statisztikák, stb. Az opciók teljes listájához illetve az ip parancs tömör leírásának megnézéséhez adjuk ki az ip --help parancsot, és olvassuk el az ip parancs dokumentációját (man ip). Javasolható még ezen kívül az eredeti fejlesztő, Alexey N. Kuznetsov által írt “IP Command Reference” című dokumentum, ami a legtöbb terjesztésben a csomag részeként feltelepül – Ubuntu alatt pl. a /usr/share/doc/iproute-doc/ip-cref.ps.gz fájlban található meg.

18. Egyéb, UNIX és UNIX-szerű rendszereken (sőt korai Linuxokon is) ennek neve lo0 volt.

Az IP parancs felépítése

Az opcionális opciók :-) után kötelezően annak az objektumtipusnak a neve áll, amelyikkel valamilyen műveletet hajtanánk végre (link – hálózati interfész; address – valamilyen címzési rendszerbeli cím; route – útválasztó tábla; és í. t.), majd a végrehajtandó parancs (add – hozzáad; delete – töröl; list/show – listáz, megjelenít; set – beállít; stb), és azok paraméterei. Legtöbb objektumtipus esetén van alapértelmezett parancs, így az akár el is hagyható. Ez az alapértelmezett parancs legtöbbször a listázó parancs. Az objektumok neve, a parancs, sőt maguk a paraméterek nevei is sokszor rövidíthetőek, általában egy-két karakter megadása elég – a lényeg, ha egyértelműen azonosítható, akkor az már jó. Bizonyos objektumtipus és parancs esetén még alapértelmezett paraméter is van, amely neve még akár úgy is elhagyható, ha az adott paraméter saját paramétert igényel. Ez a napi használat során kényelmessé teszi a használatot, de parancsfájlokban, dokumentációban kifejezetten ellenjavalt az olvashatatlanság okán. Mi ebben a könyvben az első egy-két esetben kiírjuk, de a 3., 4. helyen már rövidítünk. Azaz pl. ha létre akarunk hozni egy új interfészt, akkor azt

```
ip link add name dummy0 type dummy
```

formában adjuk meg, de ha egymás után több interfészt, akkor a többi parancsnál már rövidítjük:

```
ip l a dummy1 ty dummy
```

Fenti példa kellően borzalmas, de mind az objektum, mind a parancs nevének rövidítése szerepel, sőt a hiányzó opciónév és az opciórövidítés is (a type egyébként t-ként nem rövidíthető, mert az alapból a txqueuelen opciót jelenti).

E rövid, a parancsot (és szintaxisát) boncolgató kitérő után térjünk vissza a gyakorlati részre.

Interfész IP címének statikus beállítása

A gépben lévő interfészek listája sokféle módon lekérdezhető. Ma már az

```
ip link show
```

parancs javasolható – rövid formájában `ip l sh` . Ha csak `s`-et írunk, az nem a show, hanem a set parancsot jelenti, ezért kényelmesebbek a show helyett a list parancsot használják `ip l l` formában, sőt a parancs a korábban említettek szerint el is hagyható, szimplán csak ennyit írva: `ip l`. (Használható ezen kívül az `ifconfig -a` vagy akár a `netstat -ia` parancs is.) Az előző, az interfészeket listázó `ip` parancs kimenete valami hasonló kell, hogy legyen:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state
   DOWN qlen 1000
   link/ether 5c:9a:d8:67:89:b2 brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
   link/ether 68:5d:43:3f:a0:bb brd ff:ff:ff:ff:ff:ff
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
   link/ether 0e:fb:8d:ae:c9:fb brd ff:ff:ff:ff:ff:ff
```

Ebből szerverünkön a `lo`, és ha van Ethernet-interfész, akkor az `eth0` várható (természetesen lehet más is). Mint látható, a listában minden interfészhez tartozik egy sorszám, e mögött áll kettös-

ponttal elválasztva az interfész neve valamint az interfész paraméterei több sorban. Ha megadjuk a -o opciót, akkor minden interfészről egysoros formában kapjuk meg az adatokat (ahol az eredeti kimenetben soremelés áll, azt az egysoros kimenetben egy \ jelzi).

Ha az interfész nevét már tudjuk (a példánkban eth0), akkor a hálózati beállítások elvégzése egyszerű:

```
ip address add 1.2.3.4/24 broadcast 1.2.3.255 dev eth0
ip link set up dev eth0
```

Az első paranccsal megadjuk, hogy az eth0 interfésznek mi legyen a címe. A törtvonal után az un. netmask paraméter áll un. prefix formában (ami azt mondja, hogy a 32-bites IPv4-es cím első hány bitjét használjuk a hálózat azonosítására – a maradék bitek a konkrét gépet azonosító un. host-bitek). Az ip parancsnál kötelező az un. üzenetszórási címet is megadni (a hagyományos if-configgnál ez elhagyható, a netmask alapján automatikusan számolódik).

A második paranccsal pedig UP, azaz bekapcsolt állapotba hozzuk az interfészt – e nélkül nem forgalmazhatnánk rajta keresztül. Ugyanez a régi ifconfig paranccsal látszólag egy kicsit egyszerűbb:

```
ifconfig eth0 1.2.3.4 netmask 255.255.255.0 up
```

A fenti ip parancsnál a korábban emlegetett rövidítések mellett legtöbbször alkalmazott további egyszerűsítés, hogy a broadcast 1.2.3.255 helyett ennyit írunk: brd +. A +-jel funkciója: a netmask alapján automatikusan kiszámított üzenetszórási cím használata[19].

```
ip a a 1.2.3.4/24 brd + dev eth0
```

Természetesen egy interfész ideiglenesen kikapcsolható, ehhez a fenti ip link parancsot kell az up helyett a down paraméterrel futtatni:

```
ip link set down dev eth0
```

Az interfészhez beállított IP-címet végleg el is dobhatjuk, ekkor az ip addr parancsban az add művelet del-re cserélendő:

```
ip address del 1.2.3.4/24 broadcast + dev eth0
```

Ekkor az interfész visszaáll az eredeti, cím nélküli állapotára.

Útválasztás (routing) beállítása

Ha egy interfészen beállítottuk a használni kívánt címet, akkor már csak az útválasztó tábla beállítása szükséges az alapszintű működéshez. Ez szintén az ip paranccsal tehető meg. Útválasztásnál legjellemzőbb az un. alapértelmezett (default) útválasztó használata, ezen kívül az egy teljes hálózatot (network route) és az egyetlen gépet megcímző (un. host route) beállítás szokott sűrűbben előfordulni. Az útválasztó beállításához szükséges ip parancsok:

```
ip route add default via 1.2.3.250 # default route
ip ro a 1.2.4.0/24 via 1.2.3.254 # network route
ip ro a 1.2.5.6/32 dev eth0 # host route
```

A sorok végén álló, # mögött álló részek megjegyzések, természetesen a parancs futtatásánál elhagyandók. A route objektum neve nem rövidíthető a logikusnak tűnő r formában, az ugyanis a

19. Ezzel kényelmessége közelít az ifconfig parancséhoz.

valószínűleg kezdők által sokkal ritkábban használt, un. szabály alapú útválasztás, azaz policy routing beállításához szükséges szabályok – azaz rule-ok – rövidítése. A statikus útválasztás beállításának szintén van egy hagyományos, a UNIX-világban használt alternatív parancsa, neve: route. Fenti beállítások ezzel a paranccsal így néznek ki:

```
route add default gw 1.2.3.250
route add -net 1.2.4.0 netmask 255.255.255.0 gw 1.2.3.254
route add -host 1.2.5.6 dev eth0 # a “-host” el is hagyható
```

Amennyiben lekérdezni szeretnénk az útválasztó tábla tartalmát, használjuk az ip route show (vagy list) parancs, illetve a régebbi route -n.

```
ip ro l
```

A UNIX világból jött hagyomány ezek helyett a netstat -r -n használata.

Hálózati paraméterek dinamikus beállítása

Asztali gépeknél a statikus beállítással szemben a dinamikus, DHCP-szerver segítségével megoldott hálózati konfigurálás a jellemző. (Sok helyen még a szervereket is DHCP-n keresztül állítják be – igaz, általában fix praméterekt kapnak a szerverek.) Linux környezetben ma a legtöbb terjesztés az un. ISC-féle DHCP megvalósítást használja – kliens oldalon mindenképp. Ez többségében egyetlen parancs futtatását jelenti, neve dhclient. (Ezen kívül előfordul más is, régebben elterjedt volt a dhcpcd – DHCP client daemon –, illetve a pump nevű eszköz használata.) Az esetek többségében, semmi extra konfigurálást nem igényel a dolog, egyszerűen a konfigurálandó interfész nevét kell megadni a klienst konfiguráló parancsnak, a többit az már elintézi. Azaz DHCP-t használó környezetben elegendő csak a

```
dhclient eth0
```

parancs futtatása. (A dhcpcd ugyanígy használható, egyedül a pump igényelt az interfész neve elé még egy “-i” opciót: pump -i eth0.)

Névfeloldás beállítása

Már csak a névfeloldás bekonfigurálása van hátra. A gépek neve és IP-címe közötti összerendeléshez rendívül sok lehetőség adott. Használható a /etc/hosts nevű statikus fájl. Tetszőleges címtárszolgáltatás, mint a kihalófélben levő NIS, vagy NIS+, esetleg egy LDAP-szerver. Nem jellemző, de akár SQL-adatbázisból direktben is kérdezhetők az adatok. Általában DNS-szervert szoktak használni egy erre kitalált protokoll segytségével. (Amely szerver az adatokat akár szöveges fájlban, akár adatbázisban tárolhatja.) Mivel ennyiféle adatforrási lehetőség van, első lépésként ezt kell elmagyarázni a rendszernek.

Name Service Switch (NSS)

Ehhez az un. NSS-keretrendszert kell bekonfigurálni, erre szolgál a /etc/nsswitch.conf fájl. Ebben soronként megadjuk az adatbázist amelynek keresését konfiguráljuk (ugyanis nem csak név-IP hozzárendelést, de pl. felhasználói adatok keresését is szabályozhatjuk az NSS-sel). Az adatbázis

neve után kettősponttal elválasztva a használandó adatforrás(ok) neve(i) áll(nak). Azaz ha a névfeloldást a talán legelterjedtebb módon akarjuk használni, akkor az erre vonatkozó sor így néz ki:

```
hosts: files dns
```

Jelentése: előbb a lokális fájlban keresünk, majd ha nincs találat, a DNS-szerverekhez fordulunk. A működéshez ki kell tölteni a fent említett /etc/hosts fájlt. A szükséges adatok soronként egy IP-cím, majd szóközzel, tabulátorral elválasztva a gép teljes neve, és esetleges alternatív nevei:

```
127.0.0.1 localhost.localdomain localhost loopback
1.2.3.254 defgw.localdomain defgw
```

Viszont ha – mint példánkban – DNS-használatot is konfiguráltunk az NSS-be, akkor még egy /etc/resolc.conf nevű fájl konfigurálása hátravan[20]. Ebben a névkereséseknél használatos DNS-tartományneve(ke)t, és a használandó névszerverek IP-címét kell megadnunk:

```
search example.hu example.com
nameserver 1.2.3.254
nameserver 1.2.5.6
```

Ezzel hálózatunk alapszintű beállításával végeztünk. Fenti beállítások legnagyobb hibája, hogy amiket parancssorban állítottunk be (interfészek címe, útválasztási paraméterek), azok a gép esetleges újraindításakor elvesznek. Ezért fenti parancsok használata kevés, szükség van ezeknek az információknak a maradandóvá tételére. Ezt a különböző terjesztésekben különböző fájlok szerkesztésével lehet megoldani (most eltekintve attól, hogy egyes terjesztésekben ehhez vannak esetleg karakteres, vagy akár grafikus felületű eszközök – melyek általában ugyanezeket a fájlokat módosítják).

Ubuntu: /etc/network/interfaces

RHEL/CentOS/Scientific Linux: /etc-sysconfig-network-scripts/ifcfg-eth0

OpenSuse/SLES: /etc/sysconfig/network/ifcfg-eth0

Ubuntu esetében minden interfészt ugyanabban a fájlban, RedHat és Suse esetén pedig minden interfészt külön fájlban (amelynek nevében benne van az interfész neve) kell beállítani.

20. Érdemes még utánanézni az un. resolvconf nevű eszköznek, kicsit bonyolultabb dolgokat lehet vele megcsinálni, mint ami itt szerepel.

Egy szerver alapvető beállításai

A legtöbb szervertelepítésben van egy közös pont, akármilyen feladatra is hoztuk létre a szervert, ez pedig nem más, mint az úgynevezett alapvető beállítások elvégzése. Ez igaz lehet egy sok felhasználós levelezőszerverre, egy webszerverre, de akár egy beléptető rendszert kiszolgáló beágyazott Linux rendszerre is. Hiszen gondoskodnunk kell róla, hogy az idő be legyen állítva a megfelelő időzóna szerint, az fsck úgy legyen beállítva, hogy ne akkor induljon el egy 4 órás lemezellenőrzés, amikor a legkevésbé kellene, vagy tudjunk róla, ha egy lemez rakoncátlankodik, de természetesen ugyanilyen fontos az is, hogy a cron be legyen állítva, legyen róla napló stb. Ez a fejezet tehát feltételezi, hogy túl vagyunk egy Ubuntu LTS rendszer telepítésén, ahol még nem telepítettünk semmi specifikusat (web, FTP, mail stb. szolgáltatást) és már felraktuk a biztonsági frissítéseket. Nézzük meg tehát, milyen alapvető teendőink vannak.

Hálózattal összefüggő teendők

Ma már alap telepítésben nem kapunk telnetd-t, és portmap sem figyel automatikusan, de első lépésként érdemes feltelepíteni az nmap segédprogramot: *apt-get install nmap* Majd az *nmap localhost* parancs segítségével megnézni, hogy milyen TCP portok vannak nyitva alap kiépítésben a gépünkön. Ha szükségesnek érezzük, akkor vegyük elő a tűzfal fejezetet és az azokban leírtak alapján húzzunk fel legalább egy alap iptables-t. Ezzel együtt érdemes a számunkra feleslegesen futó alkalmazásokat eltávolítani. Azaz, ha valamiért telepítésre került egy POP3 szerver, de funkcionalitása nem lesz, akkor azt távolítsuk el. Egy alap rendszerben jobbára csak az SSH portnak érdemes látszania.

Lemezkezeléssel összefüggő teendők

Alap beállítás szerint minden 20. újraindítást, illetőleg 20 napot követően a soron következő újraindításnál a rendszer kényszeríteni fog egy fájlrendszer-ellenőrzést. Mivel jellemzően a szervereket távolról érjük el (SSH-val) ezért jellemzően nem vagyunk ott, amikor pl. egy kernel frissítés miatt bekövetkezik az újraindítás utáni lemezellenőrzés. Normál esetben ez a lemezméret és a rajta tárolt állományok függvényében lehet 20 perc, de lehet 4,5 óra is. Ha pl. 100 kB-os képállományokból tárolunk 2 TB mennyiséget, akkor ez bizony 4,5 óra is lehet. Megszakítani nem érdemes, és nem is tanácsos. Viszont ha a rendszer talál sérült állományokat, amelyeket szeretne megjavítani, akkor átlép interaktív üzemmódba, és kérdezni fog. Igen ám, de az fsck programot a rendszer hálózat nélküli üzemmódban, alacsony init szinten indítja, így hacsak nem rendelkezünk a gépre fixen vagy ideiglenesen rákötött IP-konzollal, akkor nem fogunk tudni beavatkozni. Ezért ha úgy gondoljuk, hogy jó mentést fogunk felépíteni és bízunk az fsck automatikus javító képességeiben, akkor célszerű megmondani az fsck-nak, hogy alapállapotban a feltett javító kérdésekre válaszoljon igent, amelyet a */etc/default/rcS* állományban az *FSCKFIX=yes* érték beállításával tudunk elérni.

Ha már beállítottuk az automatikus igent, célszerű a lemezeinkre beállítani egy vállalható időtartamot, azaz mondjuk meg mi az fsck számára, hogy mennyi időnként és hány újraindításon-

ként szeretnénk kikényszeríteni az ellenőrzést. Erre a feladatra a tune2fs programot célszerű használni:

```
tune2fs -l /dev/sda3
```

A -l opcióval lekérdezhetjük az alapbeállításokat, és megtudhatjuk, mikor lesz a következő tervezett lemezellenőrzés – ebben a példában – a /dev/sda3 jelölésű partíción. Megváltoztatni pedig a *tune2fs -c 40 -i 30d /dev/sda3* segítségével tudjuk, amikor is 40 csatolás vagy 30 nap eltelte után fogja követelni az ellenőrzést.

Szintén érdemes a tune2fs -m opciójával a csak rendszergazda számára fenntartott terület nagyságát az alapértelmezett 5%-ról alacsonyabbra, de még vállalható méretűre állítani – van, aki a 0,1%-ot tekinti megfelelőnek.

Most, hogy ezeken túl vagyunk, érdemes telepítenünk egy programot, amely figyelni fog a lemezeinkre helyettünk. Még a szerver tervezésénél eldöntöttük, hogy hardveres vagy szoftveres RAID-et alkalmazunk, vagy egyiket sem (ez nem javasolt!). Viszont akármelyik megoldást is választottuk, a lemezeket figyelni kell, hogy az esetleges meghibásodásról még időben értesülhessünk. Erre a feladatra a leginkább alkalmas program a smartmontools. Ez az eszköz a SMART adatokat biztosító merevlemezek megfigyelésére szolgál. A legtöbb ma kapható lemez már rendelkezik azzal a képességgel, hogy folyamatosan és/vagy meghatározott időközönként önellenőrzéseket futtat le, figyeli a lemezek környezeti paramétereit: melegedést, felpörgési időt, élettartam időt, ki/be kapcsolások számát stb. Ezekből az adatokból következtet arra, hogy egy merevlemez lehetséges meghibásodása be fog-e következni. Felhasználói környezetben ezen adatok megtekintésére vagy a parancssori megoldás (smartctl -a /dev/sda1), vagy pedig a jobban átlátható GSmartControl ajánlott. Beállítása viszonylag egyszerű, és a gyakorlati tapasztalat az, hogy a jól beállított önellenőrzési ciklusokat is magában foglaló konfiguráció képes lehet még az adatvesztést megelőzően jelezni. Szerveroldalon képes együttműködni a gyártók saját platformos Linux RAID programjaival, de sajnos nem mindegyikkel.
Telepítése egyszerű:

```
apt-get install smartmontools
```

Ennek kiadása után a függőségekkel együtt települ a smartmoontools. Első teendőnk, hogy eldöntsük, milyen lemezeket akarunk figyeltetni vele. Célszerű az összes SMART-kompatibilis rendelkezésre álló lemezt megfigyelni. Szoftveres RAID esetén az összes fizikai lemezt egyenként be kell fűzni, hardveres RAID esetén pedig érdemes a gyártók hozzá adott szoftverét is használni, illetve megnézni, hogy az adott hardvervezérlő és a smartmon együtt tud-e működni. Nézzük az egyszerű esetet, amikor 1-2 SATA vagy SAS lemez van a gépünkben, vagy SoftRaid vagy single üzemmódban. Nézzük meg, hogy a lemezeink alkalmasak-e a SMART adatok kinyerésére:

```
smartctl -a /dev/sda
  smartctl -a /dev/sdb
```

Valami ilyesmit kellene kapnunk:

```
=== START OF INFORMATION SECTION ===
  Device Model:     ST91000640NS
  Serial Number:
  LU WWN Device Id: 5 000c50 04ed10873
  Firmware Version: FTA2
  User Capacity:    1,000,204,886,016 bytes [1.00 TB]
  Sector Size:      512 bytes logical/physical
  Device is:        Not in smartctl database [for details use: -P showall]
  ATA Version is:   8
```

```
  ATA Standard is:  ATA-8-ACS revision 4
  Local Time is:    Thu Aug  8 14:36:43 2013 CEST
  SMART support is: Available - device has SMART capability.
  SMART support is: Disabled

  === START OF READ SMART DATA SECTION ===
  SMART overall-health self-assessment test result: PASSED
```

Továbbá rengeteg olyan adatot, hogy hány fokos a lemez éppen, illetve hányszor volt ki- és bekapcsolva, vagy hogy mennyi ideje pörög már. A lényeg, hogy a SMART adatok kiolvashatóak, tehát állítsuk be a SMART flag-et ENABLED állásba a következőképpen:

```
smartctl -s on -a /dev/sda
  smartctl -s on -a /dev/sdb
```

Ha ezzel megvagyunk, akkor tudassuk a rendszerrel, hogy szeretnénk, ha bizonyos időközönként figyelné a lemezeinket, szerkesszük a /etc/default/smartmontools állományt:

```
enable_smart="/dev/sda /dev/sdb"
```

Ebben adhatjuk meg, hogy a fizikai lemezeinket hogyan hívják, még akkor is ha valójában mi /dev/md1-ként hivatkozunk rá. Jelen példa mind a két szoftveres RAID lemezt figyelni fogja.

```
start_smartd=yes
```

Alapesetben off értékkel vagy kommentezve található, ha átraktuk on-ra, akkor a rendszer az indításától automatikusan figyelni fogja a lemezeket is.

```
smartd_opts="--interval=1800"
```

Megadhatjuk másodpercben az ellenőrzés gyakoriságát. Figyeljünk oda, hogy ha a smartd-t engedélyezzük, akkor általában nem kell, sőt ellenjavallt felsorolni a figyelendő lemezeket, ugyanis a smartd alapbeállítással automatikusan megkeresi.
Az alapbeállítások megadása után a service smartd restart kiadásával tudjuk root-ként (vagy sudoval) érvényesíteni módosításainkat. A /var/log/syslog alatt pedig már látható is lesz, hogy a smartd figyeli a lemezeinket. Az alapbeállítások mellett a root fog levelet kapni, ha várható vagy bekövetkezett egy lemezmeghibásodás. Az alapbeállításokat a /etc/smartd.conf állományban tudjuk finomítani, pl. ha másnak szeretnénk a figyelmeztetéseket küldetni, vagy ha egy komplett támogatott hardver RAID-et akarunk figyelni.

Hardveres RAID figyelése: számos hardveres RAID megoldás létezik a piacon. Szerencsére a gyártók is felismerték azon igényeket, hogy az eszközeiket nem csak használni szeretnénk, hanem megfigyelni is. Így számos gyártói szoftvermegoldás született, amely nem feltétlen szabad szoftver (l. a Debian Wiki által karbantartott listát). Érdemes egyedileg megnézni ezeket a segédprogramokat, mert felhasználásukat tekintve ingyenesek, de a licencük alapján nem mindig szabad szoftverek.
Jellemzően mindegyik program igényel valamilyen minimális szkriptelést, amely segítségével automatizálni tudjuk pl. az óránkénti állapotlekérdezést, illetve csak a hibák vagy hiba-előrejelzések megfigyelését, de ezek jellemzően 2-3 parancs 1 shell szkriptbe gyűjtése, illetve esetenként egy diff vagy egyéb alap parancs kombinációját jelentik. A szkript eredményét pedig e-mailben elküldve és/vagy naplószerverben feldolgozva kaphatunk értesítéseket a RAID aktuális állapotáról.

Az idő kérdése

Fontos, hogy a szerverünknek be legyen állítva az időzóna paramétere, amit a *tzconfig* parancs segítségével tehetünk meg. A Magyarországon üzemeltetett és használt rendszereknél praktikus Budapest beállítást alkalmazni. A szerveren kliensként tehát a két leginkább elterjedt lehetőség közül választhatunk az idő szinkronizálásához: a napi rendszerességgel ki- és bekapcsolt gépek esetén, közelítőleg megfelelő eredményt érhetünk el egy a gép bekapcsolásakor automatikusan lefutó, a gép óráját a központi órához igazító parancs segítségével. Ez az ntpdate parancs, használata:

```
ntpdate -b time.kfki.hu
```

Míg a „-b” opció használata javasolt, addig a time.kfki.hu helyett inkább válasszunk valami nekünk megfelelőt, ilyen lehet például a hu.pool.ntp.org (Magyarországról – más ország esetén javasolt azon ország országkódját használni a „hu” helyett). Amennyiben a gép folyamatos üzemben működik (a szerverek tipikusan ilyenek, de kényelmi okokból sokan a munkaállomásokat se kapcsolgatják), a bekapcsoláskor történő órabeállítás kevés lehet. Jellemzően a gépek órája sietni vagy késni fog egy idő után. Ebben az esetben a javasolt használat az, hogy az órát folyamatosan igazítsuk a központi órához. Ellenben az, hogy meghatározott időnként (például óránként, naponta) átállítjuk a gép óráját, felvet egy nagyon súlyos problémát – ilyen esetben elveszítjük az idő egy vagy több fontos jellemzőjét – a folytonosságot, illetve a monotonitást. Azaz vagy „luk” keletkezik – kimaradó időpont – vagy ugyanaz az időpont többször is bekövetkezik. Ezért a bevált módszer a régi vekkerórákban alkalmazott „óralassítás” / „óragyorsítás”. Azaz lekérdezzük a központi gép óráját, és ha a miénk késést mutat, akkor felgyorsítjuk, ha pedig a miénk siet, akkor lelassítjuk azt. Ezt szintén megtehetjük az ntpdate parancs segítségével, ekkor ki kell hagyni a fent említett „-b” opciót, helyette a „-B” használandó:

```
ntpdate -B time.kfki.hu
```

Majd ezt a parancsot kell óránként / napi rendszerességgel lefuttatni (tipikusan cron-ból időzítve). Ily módon futtatva az ntpdate-et, a gép szépen lassan „hozzákésik/hozzásiet” a központi időhöz. Komplexebb konfigurációt igényel az ntpd (régebben xntpd) nevű szoftver, amely elindítása után folyamatosan fut, és rendszeresen (alapesetben 64 másodpercenként, ez 16 másodperc és 36,4 óra között változtatható) lekérdezi a pontos időt, majd az előbb tárgyalt módon, a gép belső órájának felgyorsításával / lelassításával hozza összhangba a gép óráját a külvilágéval. Működéséhez viszont szükséges létrehozni egy konfigurációs fájlt (tipikusan /etc/ntp.conf néven létezik), amelyben minimálisan a lekérdezendő szerver(ek) nevét kell megadni, „server time.kfki.hu” formában. Nem árt tudni, hogy jó pár szoftver (például a Dovecot IMAP-szerver, vagy általában az adatbázis-kezelők) rendkívül zokon veszi az „időugrást” – ezek miatt szintén nem javasolt az a fajta óraállítás.
Megjegyzendő, hogy az óra folyamatos szinkronban tartásának két módja (ntpdate -B illetve ntpd) biztonsági szempontból is különbözik. Az ntpdate használatakor csak a parancs futtatásakor lesz egy rövid ideig tartó hálózati forgalom, míg ntpd használata esetén az ntpd szerverprogram folyamatosan fut, és a 123-as UDP porton keresztül elérhető. Természetesen ez utóbbi használat esetén van lehetőség az engedélyezett kliensek korlátozására.

Tudjunk a hibákról, avagy legalább minimális levelezés legyen

Ahhoz, hogy pl. a lemezértesítőket megkapjuk, mindenképpen érdemes egy minimális levelező-szolgáltatást telepíteni a gépre. Ha egy szervert üzemeltetünk, akkor a legegyszerűbb a Postfix-et telepíteni a gépre, és minimális konfigurációval (esetlegesen csak localhoston elérhetően) futtatni. Ha beállítottuk, hogy a root, postmaster, hostmaster stb. aliasok a valós e-mail címeinkre mutassanak, akkor a le nem futott, vagy hibásan lefutott cron rendszerhibáktól kezdve minden olyan üzenetet megkapunk majd, amelyeket a rendszer a root felhasználónak szánt. Ebben az esetben lapozzunk a Levelezés fejezethez, ahol egy komplex leírást kapunk A-tól Z-ig a levelezőszerver üzemeltetésével kapcsolatban. Ha viszont nem akarunk egy komplex SMTPD-t feltenni a gépünkre, mivel az pl. egy beágyazott rendszer, esetleg tűzfal és elegendő számunkra, ha egy jól behatárolt alkalmazási kör tud küldeni, akkor kereshetünk egy egyszerűbb, kliensszerű SMTPD-t, mint pl. az Msmtpd[21], amely a leginkább elterjedt azonosításokat és küldési módokat is támogatja. Ilyen cél feladat lehet, amikor pl. egy mikroszervert építünk RAID-be kötött lemezekkel fájlmegosztási célokra, és éppen csak azt akarjuk tudni, hogy a RAID állapota rendben van-e. Az Msmtpd képes levelet küldeni akár egy GMail fiók[22] segítségével is, így nem kell relay szerverekre hagyatkoznunk, hanem ha kilátunk a 465/587-es portokon, akkor közvetlen a saját fiókunk segítségével is tudunk levelet küldeni.

Ütemezés, Cron

Még egy alap rendszerben is vannak olyan feladatok, amelyeket ütemezetten kell végrehajtani. Tipikusan ilyen a mentő szkript futtatása, a kliens alapú óraszinkron, vagy egyszerű szkriptek futtatása meghatározott időben. Nézzünk egy gyakorlati példát. Képzeljünk el egy olyan, nem magára hagyott rendszert, amely alapbeállításban felrakja magára a szükséges frissítéseket és az adminisztrátor csak akkor néz rá, amikor jelentik neki a felhasználók, hogy egy rég elfeledett rendszer nem működik. Ez persze csak akkor fordulhat elő, ha nem fűztük fel ezt a gépet pl. a Nagios fejezet segítségével a monitoring rendszerünk részévé (lásd „A naplózó alrendszer, naplók elemzése” fejezetet). Ebben az esetben jól jöhet, ha már a teljes lemeztelítettség bekövetkezése előtt tudomást szerzünk erről a veszélyről. Szegény ember ha nem akar valamilyen egyéb lokális eszközt erre használni (Munin vagy Nagios), akkor ír rá egy shell szkriptet, mint pl. ez:

/usr/local/bin/df.sh[23]

```
#!/bin/bash
  FS="/"
  THRESHOLD=90
  OUTPUT=($(LC_ALL=C df -P ${FS}))
  CURRENT=$(echo ${OUTPUT[11]} | sed 's/%//')
  [ $CURRENT -gt $THRESHOLD ] && echo "$FS file system usage $CURRENT" | mail
  -s "$FS file system" sajat@e-mailcimunk.hu
```

21. http://msmtp.sourceforge.net/
22. https://wiki.archlinux.org/index.php/msmtp
23. http://www.cyberciti.biz/faq/mac-osx-unix-get-an-alert-when-my-disk-is-full/

Ezt persze futtatni kell időről időre, amire leginkább a cron[24] alkalmas. A cron egy démon, amely általunk előre meghatározott időközökben képes az előre beállított szkriptek vagy események futtatására. A cron alapesetben telepítésre kerül az Ubuntu LTS esetében, hiszen ez fogja pl. a telepítés során beállított biztonsági frissítések automatikus telepítését végezni. A fő konfigurációs állománya a /etc/crontab, amely valahogy így néz ki:

```
# /etc/crontab: system-wide crontab
  # Unlike any other crontab you don't have to run the `crontab'
  # command to install the new version when you edit this file
  # and files in /etc/cron.d. These files also have username fields,
  # that none of the other crontabs do.

  SHELL=/bin/sh
  PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

  # m h dom mon dow user  command
  17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
  25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
  --report /etc/cron.daily )
  47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts
  --report /etc/cron.weekly )
  52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
  --report /etc/cron.monthly )
```

A leginkább javasolt, hogy ezt az állományt mi ne szerkesszük, hanem használjuk a felhasználóknak szánt *crontab* parancsot, ahogyan a kikommentezett szöveg ajánlja is számunkra. Attól függően, hogy milyen felhasználó nevében szeretnénk időzített parancsot futtatni, vagy annak a felhasználónak a nevében adjuk ki a *crontab -e* parancsot, vagy rendszergazdai jogosultsággal a *crontab -u felhasznalo -e* paranccsal is élhetünk. Ezek után ha még nem választottunk alapértelmezett szövegszerkesztőt, akkor most erre lehetőségünk nyílik majd. Mindenkinek javasolt a saját megszokott szövegszerkesztőjét használnia, azzal a megkötéssel, hogy itt egy elírt karakter, vagy egy véletlen rossz helyre tett szóköz is komoly bajt okozhat, így érdemes olyan komplett megoldást választani, mint a vi szövegszerkesztő. A crontab-ban tehát a következőképpen helyezhetünk el szkripteket. A példa legyen a fenti lemeztelítettség-monitorozó szkript:

```
#perc     óra       nap      hónap  hétnapja           felhasználó
  program
  #(0-59) (0-23) (1-30/31) (1-12)   (0-6)
12 * * * *   root    /usr/local/bin/df.sh
```

Azaz minden óra 12. percében meg fogja hívni a df.sh szkriptünket, amely pedig ha 90%-ban telített a / fájlrendszer, akkor a beállított címre levelet fog küldeni.

24. http://hu.wikipedia.org/wiki/Cron

Távoli elérés: ssh

Történeti áttekintés

A nagygépes UNIX idők kezdetéből (kb. 70-es évek eleje) származik az első olyan távoli karbantartásra szánt eszköz, amely segítségével a rendszergazda úgy volt képes dolgozni helyszíntől függetlenül, mint ha ott a gép előtt tevékenykedett volna. Ennek a protokollnak a neve Telnet[25], amely az RFC 854-es szabvány leírásban van pontosan részletezve. A kornak megfelelő szemléletmód alapján lett kialakítva. Nevezetesen abból indultak ki, hogy aki képes kezelni egy ilyen eszközt és rendelkezik a megfelelő azonosítóval, az a tudását pozitív irányban fogja kifejteni. Talán ezért (vagy mivel akkoriban nem volt tömeges igény a biztonságos hálózati kommunikációra, hiszen akik nagygépes rendszereken dolgoztak, azok jóformán ismerték egymást személyesen), nem építettek titkosítási réteget a telnet protokollba. Igaz, abban az időben elterjedt titkosítási réteg nem is igazán volt. Összefoglalva, a telnet egy flexibilis eszköz, amely a terminháltípusoktól független módon TCP-n keresztül kommunikálva kötötte össze kétirányú kommunikációval a gépeket. A mai napig majd minden Linux terjesztés alap részét képezi, mivel például hálózati protokoll tesztelésre (http, pop3 és sok más) kiválóan alkalmas. Ma azonban már egy magára valamit adó üzemeltető tiltja a gépei telnettel való távoli elérését, azaz nem futtat telnet-kiszolgálót (telnetd, azaz telnet „démon”).

Manapság inkább olyan kontextusban kerül a szerver oldali telnet szóba, hogy például Nmap[26] segítségével megvizsgáljuk, nem maradt-e az alap telepítés része a telnetd. Bár igaz, hogy gyakorlatilag nincs olyan Linux-terjesztés[27], amely a szerver részt alapból telepített állapotban tartalmazná.

A telnet múltjából és jelenéből jól látható, hogy ma már távoli hozzáférésnél alapkövetelmény a titkosított közegben való kommunikáció és a jogosultságkezelés, hiszen gyakorlatilag akinek távolról helyi üzemeltetői (root) joga van a gépre, az majdnem olyan jogosultságokkal rendelkezik, mintha a gép előtt ülne közvetlenül (eltérés például, hogy egyéb eszköz híján, a rendszerindítási folyamatba nem tudunk beavatkozni, illetve bizonyos hardverhez kapcsolódó műveleteket nem hajthatunk végre – CD/DVD-csere például).

SSH

A megoldás egészen 1995-ig váratott magára, amikor is az SSH Communications Security-t a finn Tatu Ylönen megalapította, és létrehozta az első SSH (secure shell) szerver-kliens párost. Lerakták az alapjait az SSH szabványnak. Természetesen ahogy lenni szokott, a piac gyorsan reagált az SSH megjelenésére, és hamarosan számtalan fejlesztőcég alkotta meg a maga verzióját. Ma már

25. http://simple.wikipedia.org/wiki/Telnet
26. http://nmap.org/
27. A BSD-ken alapból telepítve van, persze nincs engedélyezve a futása.

teljesen általános, hogy az OpenSSH[28] implementációját használja minden disztribúció, mivel tudásában kiemelkedő. Az OpenSSH projectet Theo de Raadt[29] vezeti és alapította, aki számos más, főleg BSD operációs rendszerrel kapcsolatos fejlesztésben is vezető.

Az SSH alap eszközévé vált az üzemeltetéssel foglalkozó emberek számára, ugyanis rengeteg adminisztrációs feladat vagy eleve csak karakteresen végezhető el, vagy pedig sokkal gyorsabban megoldható szinte bárhonnan ezen a módon. Köszönhetően a tömeges felhasználásnak, a legtöbb forgalomban lévő okostelefon, vagy egyéb olyan eszköz, amely alkalmas lehet admin funkciók végrehajtására, rendelkezik futtatható SSH kliens programmal. Azaz például Android, iOS vagy akár Windows Mobile operációs rendszert futtató telefonokról is beléphetünk vele távolról gépünkre (ugyanígy régebben a Symbian operációs rendszerekre is számos kliens létezett és még létezik is.). Az egyik legnépszerűbb windowsos SSH kliens a Putty-csomag[30], amely MIT licenc alatt kerül kiadásra (gyakorlatilag a BSD licenccel egyező feltételek mellett). Ez látható a letöltési oldalon is a számtalan platformra fordított bináris letölthetőségéből[31]. Tudása olyan komplex, hogy egy átlagos képességű rendszergazda is csak 5-10%-nyi funkcióját használja ki. Legtöbb esetben a rendszergazda nyit egy titkosított shellt vagy az SSH csomag részét képező SCP kliens segítségével titkosított fájlátvitelt valósít meg. Az OpenSSH weboldalán[32] szereplő előnyök közül kiemelhető, hogy nem csak erős titkosítás valósítható meg például RSA vagy DSA kulcspárok létrehozásával, hanem OTP azonosítás[33], port továbbítása titkosított alagútban, adattömörítés az átvitt adatok esetében automatikusan (egy mobil telefonos kapcsolat esetében – például GPRS kapcsolat – nem mindegy, hogy egy egész képernyő képét kell állandóan le-fel töltögetnünk, vagy csak egy tömörített pár soros szöveget viszünk-e át) és még számtalan olyan előny sorolható fel mellette, amely alapeszközzé teszi a rendszer-üzemeltetésben.

Fontosabb ajánlható használati módozatok

Alapértelmezés szerint az SSH-szerver konfigurációját az /etc/ssh/sshd_config alatt találjuk a legtöbb disztribúcióban. Ha forrásból telepítettük, akkor pedig ahova beállítottuk (az ssh telepítése forrásból általánosságban nem javasolható, támaszkodjunk a csomag kezelőre). Az alapkonfiguráció a PAM-alapú azonosítást és a legtöbb esetben a root alapú ssh bejelentkezés lehetőségét is meghagyja nekünk. Mint jellemzően minden szerver konfig alap kiindulási konfigurációja, ez is a működésre van specifikálva, nem pedig a teljes biztonságra. Érdemes ezért alaposan átnézni a lehetőségeket, és a számunkra elfogadható maximális szintre növeli a biztonságot. Pár példa a szerver beállítási lehetőségeire:

AllowUsers, AllowGroups, DenyUsers direktíva: mivel az sshd a PAM-modullal kommunikálva fogja eldönteni, hogy ki az aki beléphet és ki az aki nem, első közelítésben itt felsorolhatjuk explicit módon, hogy mi kit engedünk. Ez felül fogja bírálni az egyéb felhasználók próbálkozásait. Például megadhatjuk így is

```
AllowUsers felhasznalo1 felhasznalo2 felhasznalo3
```

PasswordAuthentication, RSAAuthentication, PubkeyAuthentication: összetartozó direktívák, amelyek segítségével letilthatjuk, hogy sima jelszó megadásával beléphessen valaki az

28. http://en.wikipedia.org/wiki/OpenSSH
29. http://en.wikipedia.org/wiki/Theo_de_Raadt
30. http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
31. Kevéssé ismert, hogy UNIX, Linux alá is lefordítható; sok Linux disztribúció (például Ubuntu), és a különböző BSD-rendszerek alá is elérhető bináris csomagként
32. http://www.openssh.com/features.html
33. http://en.wikipedia.org/wiki/One-time_password

SSH alrendszer segítségével. A kulcs alapú azonosítás lényege, hogy az **ssh-keygen**[34] program segítségével létrehozunk egy publikus és egy titkos kulcsból álló kulcspárt. A publikus részt feljuttatjuk a szerverünk meghatározott könyvtárába, majd a megfelelő opciók kiválasztásával már csak a titkos kulcs birtokosa fog tudni belépni a rendszerbe, a sima /etc/shadow-ban tárolt jelszóval már nem. A privát kulcsot jelszóval védett formában pedig praktikusan egy erre elkülönített könyvtárban az identity- előtag használatával felcímkézve tároljuk, lásd a példát később. A publikus kulcsú azonosítás nagyban növeli a valós biztonságot és segítségével megnehezíthető például egy hátunk mögül meglesett jelszó, vagy egy keylogger program segítségével rögzített jelszóval való illetéktelen behatolás. A konfig állományba tehát helyezzük el a

```
PasswordAuthentication NO
RSAAuthentication yes
PubkeyAuthentication yes
```

sorokat.

PermitRootLogin: meghatározza, hogy a root felhasználó távoli elérés segítségével beléphet-e a rendszerbe. Érdemes tiltani, és a root felhasználót semmilyen Linux/Unix rendszeren nem használni. Helyette a sudo[35] (vagy su) megoldásait preferálni. Használata:

```
PermitRootLogin NO
```

Port: itt adhatjuk meg, hogy melyik TCP-porton hallgasson az SSH, ide fogunk tudni csatlakozni. Az alap beállítás szerint ez a 22-es lesz. Külső (internet) irányból publikusan elérhető szervereknél ezt érdemes tűzfallal védeni, esetleg port kopogtatásos[36] technológiával ezt nem nyilvánossá tenni mindenki számára. Az alapértelmezettől eltérő port használata néha magának a rendszergazdának nehezíti meg az életét (sok hálózaton belülről a 22-s portra szabad csatlakozni, ellenben máshova nem.)

Például:

```
Port 2022
```

ListenAddress: Több hálózati csatlakozás esetén, megadhatjuk az SSH-nak, hogy melyik „lábon” fogadjon el (tipikusan a menedzsment hálóból, illetve azon a lábon) kapcsolatot.

Pár példa a kliens konfigban javasolható beállításokra:

~ssh/.config: az ssh kliens esetében számtalan olyan megoldást alkalmazhatunk amely egyszerűsíti az életünket, és jó néhány olyat, amelyet más szoftver esetében csak plusz szolgáltatások beüzemelésével érhetnénk el. A leggyakoribb kliens oldali beállítás, amikor az általunk adminisztrált hostokat felvesszük a .config fájlba, és utána alias formájában hivatkozunk rá, ez nem csak azért praktikus, mivel a kulcsot és port számot is meg tudjuk adni, hanem többek között a felhasználót is beállíthatjuk: például:

```
Host szerverem1
IdentityFile /home/user/kulcsok/identity-szerverem1
Port 2222
Protocol 2
User felhasznalo1
HostName 192.168.1.1
PasswordAuthentication no
```

34. http://en.wikipedia.org/wiki/Ssh-keygen
35. http://hu.wikipedia.org/wiki/Sudo
36. http://en.wikipedia.org/wiki/Port_knocking

Hasonlóan hasznos paraméter, ha tűzfalon kell keresztül SSH-zni, ahol a kapcsolat tétlenségi időkorlátja limitálva van, azaz például 5 perc inaktivitás elteltével bontja a kapcsolatot, akkor érdemes ezt a 3 kapcsolót is használni :

```
ServerAliveInterval 60
ServerAliveCountMax 10
ForwardAgent yes (ez az agentet állítja be, amelyről még lesz szó bővebben)
```

Transzparens multi-hop: Ha rendelkezem egy SSHGW géppel, amire bejutva tovább tudok menni a WEB szerver gép felé, és onnan tovább a kvm1-es host gépre (a kvm1 egy virtuális gép, amelyre csak a host közbeiktatásával tudunk jelen esetben belépni), akkor alapesetben a Forward Agent és ssh-key opciókkal operálva mindenhova be tudok ssh-zni, majd onnan tovább a következőre. Ha azt szeretnénk elérni, hogy 1 db ssh parancs segítségével azonnal a 3. azaz a végcél kvm1-es gépre juthassunk el (SSHGW -> WEBSZERVER -> KVM1), akkor a következőt kell kiadni:

```
ssh -A -t SSHGW ssh -A -t WEBSZERVER ssh -A kvm1
```

Az -A opció mondja meg az SSH-nak, hogy a ForwardAgent kapcsoló mindenhol aktív (ON) legyen, attól függetlenül, hogy a klienseken engedve van-e vagy sem. A -t opció pedig erőlteti a Pseudo-TTY allokációt (ez az interaktív parancsfuttatásokhoz kell). A fenti SSH képességet a ~ssh/.config fájlban a következőképpen rögzíthetjük és hivatkozhatunk rá:

```
Host SSHGW
     IdentityFile /home/felhasznalo/kulcsok/identity-MAIN
     Port 22
     Protocol 2
     User update
     HostName 192.168.1.1
     PasswordAuthentication no
  Host WEBSZERVER
     IdentityFile /home/felhasznalo/kulcsok/identity-Test
   Port  22
     Protocol 2
     User update
   HostName 192.168.1.2
     PasswordAuthentication no
  Host kvm1
    ProxyCommand ssh -q MAIN nc -q0 kvm1 22
```

Ha ezt így rögzítettük a konfigban, akkor utána egy egyszerű ssh kvm1 parancs kiadása után (látványosan tovább tart a bejutás, akár 5-10 másodperccel is) egyenesen a kvm1 gépre jutunk. Ugyanígy akár fájlokat is küldhetünk közvetlenül a kvm1 gépre/gépről. Ebben a példában a távoli gépről másolunk a saját gépünkre:

```
scp kvm1:/tmp/proba.txt /tmp/proba.txt
```

Vagy szükség esetén indíthatunk egy távoli (szintén beágyazott, azaz tűzfal és szerver mögötti alhálón) desktop gépen egy Nautilus ablakot, és hozzáférhetünk az állományokhoz a következőképpen (maradva a felsorolt konfig sorainál képzeljük azt, hogy a kvm1 gép egy Ubuntu desktop, tehát fut rajta X szerver):

```
ssh -X kvm1 nautilus
```

kisvártatva egy a kvm1 gépen indított és az ottani állományokat megjelenítő fájlkezelő felületet kapunk. A fenti konfigban természetesen a kulcsokat és a felhasználókat tetszőlegesen variálhat-

juk. Azaz, ha az első SSHGW gépen admin felhasználó amivel bejutunk, a 2.-on admin2, a kvm1 hoston pedig staff, akkor csak át kell írni a konfigot és persze az scp parancs is működni fog utána.

Porttovábbítás: elő szokott fordulni, hogy szükség van egy titkosított adatcsatornára két gép között. Mind az OpenSSH, mint a Putty háromféle porttovábbítási funkciót támogat. Talán leggyakrabban a -L (más néven lokális) formáját használjuk:

```
elso.gep.hu> ssh -f -N -L localhost:1111:harmadik.gep.hu:80 masodik.gep.hu
```

Ezzel a paranccsal a saját gépünk (elso.gep.hu) 1111-es portján keresztül elérjük a harmadik.gep.hu 80-as portját, de a harmadik gép úgy érzékeli, hogy a másodikról jöttünk (tipikusan ilyen, amikor egy olyan eszköz felügyeleti oldalát kell elérni távolról, amelyik csak a (neki) lokális hálózatról fogad el kapcsolódási kéréseket. Miután a fenti parancsot elindítottuk, a böngészőt a http://localhost:1111 címre irányítva máris a harmadik.gep.hu weboldalán találjuk magunkat. Ritkábban használt a -D az un. dinamikus applikációs porttovábbítás, ebben az esetben az ssh SOCKS-proxyként viselkedik. Használatához SOCKS-proxy támogatással rendelkező alkalmazás, vagy a későbbiekben tárgyalt **tsocks** szükséges.

```
elso.gep.hu> ssh -f -N -D localhost:1080 masodik.gep.hu
```

A böngészőnkben beállítva a proxy támogatást, és proxy szerverként megadva a localhost:1080-as adatot, a forgalom az SSH-alagúton keresztül, a masodik.gep.hu -n keresztül jut el a címre.
A harmadik, un. távoli (remote) porttovábbítás használata (-R opció) meglehetősen ritkaságszámba megy.

```
elso.gep.hu> ssh -f -N -R localhost:1111:harmadik.gep.hu:80 masodik.gep.hu
```

eredménye: ha valaki a második gép 1111-es portjára csatlakozik, az az SSH kapcsolaton keresztül átkerül a kiinduló (első) gépre (ahol az ssh parancsot kiadták), majd onnan csatlakozik a harmadik gép 80-as portjára. Ez a példa az első példa (a lokális porttovábbítás) megfordítása, azaz amikor mi szeretnénk elérni azt, hogy valaki kívülről elérhesse a mi belső hálózatunkról elérhető eszközt. (Mi belülről megnyithatjuk az ssh-kapcsolatot, de a távoli ember kívülről nem.)

Korlátozás az SSH-kulcs segítségével: ha az a cél, hogy explicit megmondjuk az ssh-kulcs segítségével, hogy az adott felhasználó azonosítás után milyen parancsokat futtathat, akkor az ssh erre is kínál egy beépített megoldást (ez azonban nem keverendő a chroot direktívával). A kulcs generálása után a publikus oldali kulcsnál a ~/.ssh/authorized_keys fájlban a következő megoldást kell ebben az esetben alkalmazni:

```
from="192.168.1.1",command="/home/update/validate-parancs" ssh-dsa
  AAANas2s4s5za82C1yc2EAAs97mAIPABIwAABAEA8xRLEVyrscvIoJmcWd9/qH......
```

Ahogy látszik, a kulcs mellett egy IP-címet határozunk meg, majd egy parancsot adhatunk meg. Ez akár egy shell parancsállomány is lehet, amiben aztán az SSH_ORIGINAL_COMMAND nevű környezeti változót felhasználva akár többféle parancs engedélyezésére is lehetőségünk van. Itt[37] egy igen szép példa található. Ennek a gyakorlati jelentősége például az olyan jellegű mentéseknél van, amikor egy távoli root hozzáférést kell engedélyezni, amely aztán egy pillanatképet fog készíteni egy távoli mentő szerverre. Persze célszerű ezt a megoldást kerülni, és fordított logikával megoldani, azaz a helyi root felhasználó futtatja a mentést, és küldi át egy ssh-alagút, vagy VPN segítségével a távoli szerverre a pillanatképet, nem pedig fordítva. A példához visszatérve, jelen esetben az /etc/ssh/sshd_config fájlz érdemes a

```
PermitRootLogin=forced-commands-only
```

37. http://troy.jdmz.net/rsync/#validate-rsync

opcióval kiegészíteni a NO helyett, így a root kulcsa mellett engedélyezett fenti parancsok futtatására lesz csak lehetőség.

Egy további lehetőség az azonosításra a **google authenticator** és az SSH összepárosítása. Ha rendelkezünk Androidos vagy IOS-es mobil telefonnal és feltelepítjük a google hitelesítő alkalmazását[38], akkor lehetőségünk nyílik arra, hogy Ubuntu LTS alatt az SSH hoz kapcsoljuk. Fontos tudni, hogy az publikus kulcs alapú azonosítás és a ChallengeResponseAuthentication[39] egyszerre csak a 6.2 es verziójú OpenSSH-tól felfele támogatott. Azonban az Ubuntu LTS nem tartalmaz ilyen verzió számú OpenSSH-t, így a ennek a hitelesítésnek a használata csak akkor javasolt, ha a publikus kulcs azonosítását kizártuk, vagy valamilyen okból kifolyólag ideiglenesen szükséges, hogy a kulcs nélkül, de mégis egy sima jelszavas azonosításnál biztonságosabb módszerrel férjünk hozzá a gépünkhöz. Ilyen indokolt eset lehet, ha pl nyaralás közben egy internet kávézóból Live Ubuntu lemez segítségével, de kulcs nélkül akarunk hozzáférni az SSH hoz és a telefon nálunk van. A google authenticator telepítésének egyszerű lépései:

```
sudo apt-get install libpam-google-authenticator
```

a tárolóból feltelepítjük az authenticator alkalmazást, majd annak a felhasználónak a nevében elindítjuk, amelyiknek a nevében szeretnénk az azonosítást végezni:

```
$ google-authenticator
```

A terminálban megjelenő QR-kódot a telefonunk google authenticator alkalmazásával felolvastatjuk, majd a terminálban közben futó shell script néhány egyértelmű konfigurációs kérdésére válaszolunk. Ezekután az /etc/pam.d/sshd állományhoz hozzáadjuk a következő sort:

```
auth required pam_google_authenticator.so
```

Majd az /etc/ssh/sshd_config állományba pedig a következőt:

```
ChallengeResponseAuthentication yes
```

Figyeljünk arra, hogy a PAM auth be legyen kapcsolva az SSHD beállításokban (alapesetben be van). Ezek után a service ssh restart paranccsal újraindítjuk az SSHD-t. Innentől ha mindent jól csináltunk, akkor a slogin -l user@localhost parancs után kérni fogja a jelszavunkat, majd ha azt jól adtuk meg, akkor Verification code-ot is kérni fog, amelyet a telefonunk generál számunkra. Ha 6.2-es vagy e feletti OpenSSH val rendelkezünk, akkor bekapcsolhatjuk a kulcsos és a google authenticatort egy időben az AuthenticationMethods publickey,keyboard-interactive sshd opció segítségével, de vigyázat, ezt csak a 6.2-es verzió feletti SSH fogja értelmezni!

SFTP alrendszer: az SSH részét képezi az FTP leváltására kiválóan alkalmas és erősen ajánlott SFTP alrendszer. Nagyon sok rendszer-adminisztrátor a mai napig kénytelen az elavult és ezer sebből vérző FTP protokoll fölé ültetni egy TLS/SSL réteget és azt használni, mivel sok esetben a követelmények kimondják az FTP protokoll használatát (pl. elavult weblaptervező szoftver stb.). Pedig az SFTP remek megoldást kínál az SSH összes előnyével együtt. Támogatja a chroot-olt környezet létrehozását, felhasználókra és csoportokra is, vagy akár sftp-only felhasználókat is létrehozhatunk. Ezeket kell hozzá a szerver beállításaiba (/etc/ssh/sshd_config) írni:

```
Subsystem sftp internal-sftp
Match Group sftp-only
AllowTCPForwarding no
X11Forwarding no
ForceCommand internal-sftp
```

38. https://support.google.com/accounts/answer/1066447?hl=hu
39. http://en.wikipedia.org/wiki/Challenge%E2%80%93response_authentication

```
ChrootDirectory %h
```

Főként azért nagyon jó választás az FTP-vel szemben, mivel az SSH titkosítását és opcióit kínálja az FTP-vel szemben, így akár az AllowUsers, DenyUsers stb direktívákat és természetesen az RSA és DSA kulcsokat is használhatjuk. Szerencsére ma már szinte az összes elterjedt operációs rendszerre rendelkezésre áll valamilyen grafikus felület[40], amely segítségével a kezdők is tökéletesen elboldogulnak. Továbbá az olyan mentésekhez is fel lehet használni, ahol parancs állomány írása egészíti a mentést. Tipikusan ilyen terület, amikor az SQL-szerver tömörített dump állományait kell időről időre átvinni valahova (kiegészítő heti, havi mentés stb.):

```
sftp -b /root/sftp.sh -i /root/keyfile 192.168.1.1:/backup/
```

Az SSH számtalan izgalmas lehetőséget tartalmaz még számunkra. Jellemzően egy átlag SSH felhasználó legfeljebb az 1-2%-át használja a lehetőségeknek. Sajnálatos módon azonban ez a fejezet nem az SSH szinte végtelen lehetőségeiről szól, csupán a leggyakoribb lehetőségeket taglalja.

Üzemeltetést segítő alkalmazások

Az SSH kliens-szerver megoldás segítségével most már hozzáférhetünk helytől független módon a távoli szerverünkhöz. A kérdés az, mire is lesz ez jó nekünk? A legtöbb disztribúciónak alapesetben része, vagy csomagkezelővel elérhető számtalan olyan eszköz, amely az alapvető adminisztrációt kezeli. Olyan sok adminisztrációs eszköz létezik, hogy azok felsorolása túlmutat ezen fejezet céljain. Néhány fontosabb eszközt azonban részletezünk:

Nmap[41]

Ezen eszköz segítségével a szerverünk hálózati (lokális és távoli) felderítését tudjuk elvégezni. Segítségével a nyitott/zárt/rejtett portokat tudjuk letapogatni, vagy közelítő információt kaphatunk az operációs rendszer verziójáról. Számos egyéb funkciója van[42].

Sudo[43]

Unix/Linux környezetben lehetővé teszi azt, hogy egy másik felhasználó – általában a root – jogosultságával (nevében) futtassunk programokat. Igen részletesen beállítható, hogy akár csak egyetlen parancs milyen módon és paraméterekkel engedélyezett. A mai modern szemlélet szerint a root felhasználó csupán technikai jelleggel szerepel a rendszerben, azt élesben (belépni a nevében, vagy su parancson keresztül) használni nem tanácsos, éppen ezért érdemes a sudo használatára áttérni. A Sudo konfigurációs állománya az /etc/sudoers fájlban található, jellemzően a mai disztribúciók a visudo parancs segítségével kínálják szerkesztésre a sudoers állományt. Manapság már nem szokás közvetlenül a felhasználókat felvenni a sudoers fájlba, hanem a tipikusan sudonak nevezett csoportba érdemes berakni az adott felhasználót ahhoz, hogy rendelkezzen az ALL jogosultsági szinttel. A sudo számtalan lehetőséget kínál[44], az egyik leggyakrabban használt opció, amikor egy adott felhasználónak nem engedünk meg mindent, csupán egy adott parancs, vagy parancsfájl futtatását engedélyezzük, az alábbi módon:

40. Windows környezetben javasolható a WinSCP (http://winscp.net/) nevű alkalmazás, de parancssoros SFTP kliens a Putty-hoz is tartozik.
41. http://nmap.org/
42. http://en.wikipedia.org/wiki/Nmap
43. http://hu.wikipedia.org/wiki/Sudo
44. Egy üres délutánon vegyük magunk elé a dokumentációt, jól el fogunk csodálkozni a lehetőségein

```
update ALL=NOPASSWD: /usr/bin/apt-get, /usr/bin/aptitude
```

A fenti példában a később tárgyalandó apt-dater program használatához létrehoztunk egy update nevű felhasználót, és ezen felhasználó részére jelszó nélkül engedélyeztük, hogy az apt-get és aptitude parancsokat root jogosultsággal futtassa. Régebben, jellemzően a betárcsázós korszakban, a pppd futhatott hasonló jogosultságokkal, hogy az összes felhasználó vezérelhesse.

GPG[45]

Biztonsági szempontból az egyik legnagyszerűbb szabad szoftver, természetesen GPL licenc alatt elérhető. Legfontosabb funkciója a titkosítás. Segítségével állományokat, leveleket lehet titkosítani oly módon, hogy az harmadik fél számára nem fejthető vissza. Nyílt kulcsos titkosítást használ, amely eljárásnál két kulcsot használunk. Az egyik egy publikus (nyilvános) kulcs, amit titkosítás nélkül kell közzétennünk a másik fél számára – ezt bárki megszerezheti. A másik kulcs a privát (titkos) kulcs, amelyet jelszóval védve biztonságban kell tárolnunk. A rendszer alapelve, hogy a publikus kulcsból nem határozható meg a privát kulcs (és fordítva sem). A másik nagy előnye ennek a titkosítási módnak, hogy már akkor tudok egy általam ismeretlen embernek titkosítottan levelet küldeni, ha még nem találkoztunk és akár nem is ismer. Egyetlen feltétele, hogy az interneten található kulcsszerverek egyikén, vagy a másik ember publikusan elérhető weboldalán szerepeljen a publikus kulcsa, amelyet felfűzve a saját kulcskarikámra máris tudok neki titkos levelet írni, vagy állományt kódolni neki úgy, hogy azt csak és kizárólag ő fogja tudni kibontani/olvasni. Ideális tehát rendszergazdák közötti kommunikációra, hiszen sokszor érzékeny információt kell utaztatni nyílt hálózaton (jelszavak, hozzáférések stb.). Továbbá ideális lokális jelszótárolásra is. Alkalmas továbbá bizalmi körök kialakítására a saját aláírási rendszerével. Fontos funkciója, hogy a kulcsaink rendelkeznek úgynevezett egyedi ujjlenyomattal (*fingerprint*). Azaz ha kulcsot akarok cserélni egy általam még ismeretlen emberrel, akkor első teendő a kulcs felfűzése a karikára, majd az ujjelenyomat egyeztetése a másik féllel, immár egy azonosításra alkalmas csatornán, pl. telefon, videobeszélgetés, aláíró parti, stb. A Debian alapú rendszerek[46] csomagjai a biztonsági csapat GPG-aláírásával vannak ellátva, így azok telepítésnél ellenőrzésre kerülnek. Akár az MD5 szignó helyett is alkalmazható remek és annál megbízhatóbb eszköz. A GPG bővebb ismertetését és végfelhasználói konfigurációját, valamint a hozzá kapcsolódó segédprogramokat a keretrendszer GPG melléklete tartalmazza.

tsocks[47]

Gyakori probléma, hogy olyan nyílt hálózaton keresztül kell távoli adminisztrációt végezni, amely nem titkosított (nincs VPN, nincs SSL, stb), mint például egy nyílt Wi-Fi. Ilyen esetben a legegyszerűbb megoldás, ha egy SSH-alagutat[48] húzunk ki egy megbízható szerverünk és a kliens között. Az alagútba aztán beterelhetjük olyan alkalmazások forgalmát, amelyek ezt támogatják. Például a Firefox, a Chrome (és a Chromium), de még az Opera böngésző is támogatja a SOCKS5 (hivatalos nevén: socket-proxy version 5) üzemmódot, amikor a forgalom ebben az alagútban tud közlekedni, így rejtve el titkosított módon az amúgy titkosítás mentesen menő – például web – forgalmat. Sajnos azonban nem minden program tud alagút technológiával dolgozni, éppen ezért érdemes a tsocks programot használni, amely a legtöbb TCP-alapú program kommunikációját bele tudja terelni egy előtte SSH-val kiépített csatornába.[49] Ha a tsocks csomag már telepítve van, ak-

45. http://www.gnupg.org/
46. a legtöbb terjesztés erős késéssel ugyan, de csatlakozott ehhez a hitelesítési megoldáshoz
47. http://tsocks.sourceforge.net/
48. http://www.linuxjournal.com/content/ssh-tunneling-poor-techies-vpn
49. A tsocks-hoz hasonló elven működő alkalmazás nagyon sok létezik, van aki a dante-client csomagban levő socksify, mások a proxychains nevű eszközre esküsznek, de a sort lehetne még folytatni

kor használata a következő. Az SSH fejezetben már tárgyalt porttovábbítás használatával kiépítünk egy lokális dinamikus porttovábbítást, például így:

```
ssh -i /home/user/kulcsok/identity-szerver  -p 22 -v -N -D localhost:1080
  user@szeveremneve
```

Fenti[50] eredményeként a localhost 1080-as portján elérhetővé válik egy SOCKS5 proxy, amely a hozzá érkező forgalmat az SSH-alagúton keresztül átküldi „szerveremneve" gépre. Onnan pedig már mintha arról a gépről indult volna, megy az adat a célhoz. Ezt követi a tsocks. Először állítsuk be a nekünk szükséges opciókat az /etc/tsocks.conf fájlban. Amelyek az előző ssh-parancshoz igazodva a következők legyenek:

```
server = 127.0.0.1
server_port = 1080
```

Ez a 2 egyszerű sor azt mondja meg, hogy a localhost-on a 1080-as porton hallgat a SOCKS5 proxy, amin keresztül kell majd a tsocks-nak a forgalmat továbbítania. Vigyázat, nincs több azonosítás ez után. Azaz aki elérheti a localhost 1080 portját, az az alagútban fog forgalmazni! Mindezek után már csak a tsocks előtaggal kell indítani az alkalmazást ahhoz, hogy az adott alkalmazás az ssh alagútban forgalmazzon, például így:

```
tsocks pidgin
```

Természetesen böngészőt is indíthatunk ugyanilyen módon, bár a böngészők – mint az fentebb egyszer már szerepelt – általában beállíthatóak SOCKS5 proxy használatára. A legtöbb esetben ez jól működik. Akadnak azonban olyan esetek, például pont böngészők esetében, amikor a forgalmazás például a böngészőből hívott Java alkalmazás miatt csak részben megy az alagútban. Ilyenkor érdemes magának a Java alkalmazásnak is a SOCKET-proxy használatát beállítani.

sshuttle[51]

A sshuttle is egy remek eszköz VPN építéshez, ha a megvalósítandó feladat egyezik a tsocks-szal megoldható feladatkörökkel. A fő különbség, hogy amíg a tsocks esetében az alkalmazásoknak vagy támogatni kell a socket proxy üzemmódot, vagy pedig a tsocks parancs segítségével a tunnelbe kell terelni a forgalmat, addig az sshuttle esetében erre nincs szükség, mivel az IPTABLES-t állítja be a lokális gépen, a forgalmat a távoli szerver fele terelve. A távoli szerveren viszont szükséges a Python interpreter és az ssh-n keresztüli parancsfuttatási lehetőség megléte (lévén az sshuttle szerver komponensét feltöltés után ott futtatja az eszköz). UDP és ICMP adatot nem fogunk tudni küldeni/fogadni, de a legtöbb esetben erre nem is feltétlen van szükség, legalábbis nem feltétlen a tunnelen keresztül. (Kivételként a névfeloldáshoz szükséges DNS-forgalmat lehetne említeni, de ehhez van egy --dns opció, ha szükséges.) Telepítése és használata is egyszerű:

```
apt-get install sshuttle
```

A telepítés után nincs szükség konfigurációk állítására, egész egyszerűen egy létező távoli SSH hozzáférésre van csak szükségünk:

```
sshuttle -r felhasznalonev@tavoli.szerver.kft 0/0
```

Ebben az esetben az összes TCP forgalmat a távoli szerver felé tereltük, de megadhatunk neki különböző alhálózatokat is. Természetesen a korábban már leírt transzparens MultiHop konfigurációk is használhatóak, illetve az ssh .config állományába rögzíthetjük a neveket, portokat, fel-

50. Ha a fenti parancsban a -v opció helyett a -f -et használjuk, akkor az ssh ugyan jóval kevesebb infót küld, cserébe háttérbe megy, így nem szükséges a terminált nyitvatartani a kapcsolat életbentartásához – a tesztek utáni éles használathoz talán alkalmasabb így.
51. https://github.com/apenwarr/sshuttle

használókat előre. A csatlakozás után az összes forgalom a megadott SSH szerver felé fog menni, így például a böngészők (Firefox, Opera, Chrome), csevegőprogramok (Pidgin, Xchat, Skype), levelező programok (Thunderbird) automatikusan az SSH tunnelben fognak végződni. Lehetőség van a -D opció segítségével arra, hogy démon üzemmódban a háttérben fusson, így nem kell azt a konzolt/terminált nyitva tartani, amelyben a parancsot kiadtuk (bár ez utóbbi funkcionalitás akár a screen, akár az fg parancs segítségével is kiváltható).

Hálózati monitorozó szoftverek

Iptraf[52], arpwatch[53], wireshark[54], tcpdump [55], lsof[56] és még számtalan hasonló program létezik lokális és távoli hálózatok forgalmának felügyeletére, hiba keresésére.

Jelszókezelés[57]

Minden rendszer használatának igen sarkalatos pontja a megfelelő jelszó megválasztása. A jó jelszó számok, betűk, írásjelek megfelelő kombinációjából áll, legalább 8-12 karakter hosszúságban. Jelszavak tömeges létrehozása esetén fontos, hogy ne a fantáziánkra bízzuk a jó jelszó kitalálását, hanem egy programra, mint amilyen például a pwgen[58].

Editor

Egy igen sarkalatos és megosztó pontja ez a Unix/Linux közösségeknek. A karakteres felületen végzett munka egyik sajátossága, hogy a konfigurációs állományokat tudni kell szerkeszteni, terminál- és billentyűzetkiosztás-független módon. A disztribúciók általában alap telepítésben tartalmazzák a következők valamelyikét: vi[59], vim[60], joe[61], nano[62], pico[63], emacs[64]

Shell program

Amint az SSH program segítségével azonosítottuk magunkat, a bejelentkezés következő lépése, hogy a /etc/passwd állományban meghatározott programot indítja a rendszer számunkra. Ugyanúgy, mint az editor kiválasztásában itt is az számít, hogy a saját komfort zónánkat megtaláljuk és úgy válasszuk ki a számtalan lehetőség közül a megfelelőt. A legnépszerűbb lehetőségek[65]: bash, tcsh, zsh, ksh, jsh stb.

52. http://iptraf.seul.org/
53. http://en.wikipedia.org/wiki/Arpwatch
54. http://www.wireshark.org/about.html
55. http://www.tcpdump.org/
56. http://hu.wikipedia.org/wiki/Lsof
57. Részletesebb tárgyalása a Központi autentikációról szóló fejezetben
58. http://pwgen-win.sourceforge.net/
59. http://hu.wikipedia.org/wiki/Vi
60. http://www.vim.org/
61. http://joe-editor.sourceforge.net/
62. http://www.nano-editor.org/
63. http://en.wikipedia.org/wiki/Pico_(text_editor)
64. http://www.gnu.org/software/emacs/
65. http://penguin.dcs.bbk.ac.uk/academic/unix/linux/shells/index.php

Apt-Dater[66]

Egy terminál alapú, szervereken csomagok karbantartására és frissítésre használható eszköz. Egy igazi hiánypótló alkalmazás, amely alapjait a nagyvállalati környezetből merítették. Az ötlet lényege, hogy nagyvállalati környezetben több tíz vagy száz (vagy akár ezer) ugyanolyan szervert kell karbantartani. Azaz ha érkezik egy javítás, amelyet azonnal telepíteni kell, akkor az adminnak ne több száz gépre kelljen ssh-val bemennie és ott kézzel frissítenie, vagy a cron ütemezőből megoldani a dolgot (hiszen sokszor ezek a javítások interaktív módon beavatkozást is igényelnek), hanem egy üzemeltetői shellből biztonságos módon tudja figyelemmel kísérni és természetesen be is tudjon avatkozni. Az apt-dater erre kiválóan alkalmas, kezeli a kulcsos ssh azonosítást, használja a screen-t, és ha szükséges az ssh-agent-et. Ha több mint 2-3 Debian vagy Ubuntu alapú szervert kell üzemeltetni, akkor gyakorlatilag nélkülözhetetlen alkalmazás. Igazi előnye, hogy a beállítása szerverenként csak pár percet vesz igénybe. Az apt-dater szóhasználata szerint kliens az a gép, ahonnan a többit menedzselem, és host-nak hívja a menedzselt szervereket. A beállításához először fel kell telepítenünk arra a kliens gépre az **apt-dater**-t amelyikről az összes szervert elérjük. Ez lehet a saját PC/notebook gépünk, vagy akár egy erre fenntartott, biztonságosan elszeparált (például: csak VPN-nel elérhető, titkosított merevlemezű VM) virtuális gép is:

```
apt-get install apt-dater
```

Az apt-dater telepíteni fogja a függőségeit is, ezek után készítsünk egy megfelelően nagy ssh kulcsot a külön felhasználóhoz, amely futtatni fogja a dater-host scripteket. A -C opció egy megjegyzést fűz a kulcshoz, a -f segítségével pedig megadhatjuk, hogy az alapértelmezett id_rsa (és id_rsa.pub) helyett hol tárolja a kulcsokat

```
ssh-keygen -t rsa -b 8192 -C apt-dater-kulcs -f ~/kulcsok/identity-update-
  server[67]
```

Az elmentett titkos kulcsot tároljuk biztos helyen, a publikus párját pedig juttassuk fel a karbantartani kívánt szerverekre az ssh-copy-id parancs segítségével, miután a felhasználót már létrehoztuk. A host gépeken létrehozunk tehát egy update felhasználót, amelynek beállítunk egy kellően véletlen (pwgen -s 16) jelszót (ez a továbbiakban nem fog kelleni).

```
adduser update
```

Az ~/update/.ssh/authorized_keys állományba másoljuk a készített publikus kulcsot. Ha ezzel megvagyunk, akkor az /etc/ssh/sshd_config-ban engedélyezzük az update felhasználó belépését az AllowUsers sor bővítésével:

```
AllowUsers admin update
```

Majd letesztelük, hogy távolról az update felhasználó bejelentkezése működik-e a kulcs segítségével:

```
ssh-add ~/kulcsok/identity-update-server    #(ez az állomány tartalmazza a
  titkos kulcsot)
ssh update@szerverunk.cime
```

Ha minden rendben volt akkor a host gépre felrakjuk az apt-date host script gyűjteményt:

```
apt-get install apt-dater-host
```

66. http://www.ibh.de/apt-dater/
67. A példában RSA kulcsot generálunk. A RSA mellett szól a szinte tetszőleges kulcshosszúság megadási lehetősége, ezért is szerepel a példában 8K. A DSA előnye hogy véletlenül sem tudnánk SSH protokol v1-et használni vele (viszont sajnos nem állítható a kulcshossz.)

Majd beállítjuk a hoston szintén a sudo paramétereit, hogy az update felhasználónak legyen joga frissíteni:

```
visudo
update ALL=NOPASSWD: /usr/bin/apt-get, /usr/bin/aptitude
```

Ezzel lényegében a host gépeken végeztünk a beállításokkal. A továbbiakban a saját gépünkön, vagyis azon a gépen fogunk dolgozni, ahonnan a frissítéseket vezérelni akarjuk. Az első lépésben már felraktuk az apt-dater-t, amely így alap beállításokkal került telepítésre. A konfig állományok a ~/.config/apt-dater/ könyvtárban találhatóak. Amivel első körben érdemes dolgozni, az a hosts.conf állomány. Szintaxisa viszonylag egyszerű és érdemes az ssh .config állományával variálni:

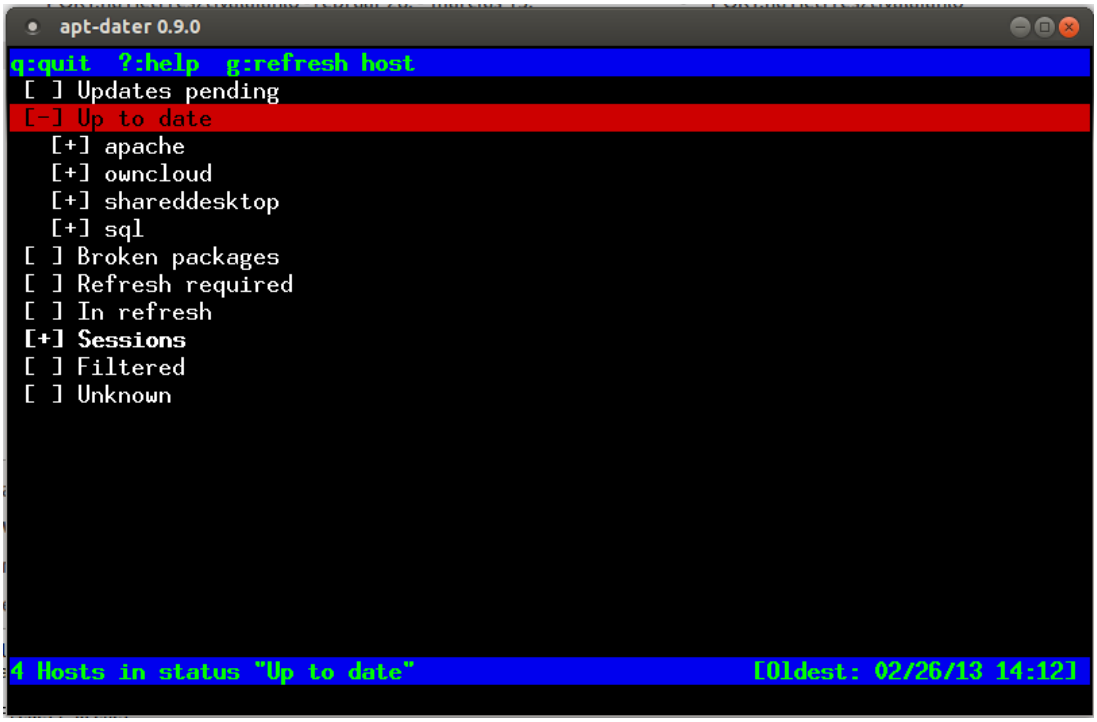
```
[Main]
  Hosts=update@192.168.1.1:22
[Test]
  Hosts=update@192.168.1.2:22
[sql]
  Hosts=update@sql
[apache]
  Hosts=update@apache
```

az első két gép, Main és Test egyszerűbb, közvetlenül ssh-val elérhető gépek. A másik kettő (sql, apache) pedig a Main vagy test gépeken keresztül, mivel azok az itteni példákban virtuális gépei azoknak. Jelen konfighoz az SSH Transzparens Multihop képességét fogjuk használni, amelyet az ssh fejezetben részletesen ismertettünk:

```
Host SSHGW
  IdentityFile /home/user/kulcsok/identity-MAIN
  Port 22
  Protocol 2
  User update
  HostName 192.168.1.1
  PasswordAuthentication no
Host WEBSZERVER
  IdentityFile /home/user/kulcsok/identity-Test
  Port  22
  Protocol 2
  User update
  HostName 192.168.1.2
  PasswordAuthentication no
Host apache
  ProxyCommand ssh -q MAIN nc -q0 apache 22
  Host sql
  ProxyCommand ssh -q MAIN nc -q0 sql 22
```

Az Apt-dater és az ~/.ssh/config paraméterei összhangban kell hogy legyenek a fenti leírás szerint. Jelen esetben, ha a kliensről akarunk a 2. szintű apache virtuális géphez eljutni, akkor az ssh-agent indítása után (amelyet a standard Ubuntu desktop alapesetben elindít) az ssh-add /home/user/kulcsok/identity-MAIN paranccsal a kulcsot hozzáadjuk az ssh-agent tárolójához, és ezek után már csak az ssh apache parancsot kell majd kiadni, és azonnal a 2. szintű gépre lépünk be.

Mindezek után az apt-dater parancs kiadása után már ez látszik előttünk:



Az apt-dater teljesen alapszintű használata viszonylag egyszerű, a súgót a ? lenyomásával tudjuk aktiválni.

Screen[68]

Szintén igen hasznos program, amely segítségével egy belépett shell ablakot sokszorozhatunk meg. Gyakorlatilag egy karakteres ablakkezelő, amelynek segítségével akkor is folytathatjuk a program futását, ha a terminál kapcsolat megszakadt (azaz kiléptünk az ablakból). Ideális több gépen való munka koordinálására, akkor ha például egy ablakban akarunk mindent csinálni. Valamint ideális batch alapú program futtatására, megfigyelésre, adatgyűjtésre. Tipikus felhasználási terület, amikor egy nagy méretű adatbázist állítunk helyre (dump) távolról. Ha ilyenkor megszakad az internet-kapcsolat akár csak 1-2 másodpercre is, az SSH nem biztos, hogy megtartja nekünk a kapcsolatot. Így az adatbázis dump is megszakadhat, ha azt kézzel végeztük. Ha ugyanezt a folyamatot screen alatt futtatva hajtjuk végre, akkor újra belépve a screen -x parancs kiadásával visszakapjuk az éppen aktuális állapotot. Segítségével akár élőben is megfigyelhetünk egy kezdő rendszergazdát (a betanítás során), így elkerülhető, hogy egy esetleges hiányosság nagyobb problémát okozzon.[69]

Távoli asztal alkalmazások

Míg e dokumentum első (nagyobbik) részében a távoli rendszerek elérésének elsősorban karakteres felületű részével foglalkoztunk, ez az utolsó rész a grafikus felületű elérések különböző meg-

68. http://www.gnu.org/software/screen/
69. A screen mellett érdemes megemlíteni, hogy több hasonló funkciókra alkalmas másik program létezik, például tmux (elérhető a http://tmux.sourceforge.net/ oldalról) vagy byobu (http://byobu.co)

valósításairól szól. A klasszikus unixos rendszergazda szemléletbe ez nem fér bele: az alap hozzáállás az, hogy a rendszer adminisztrációjához csak a következő dolgokat kell tudni: hogy hívják a menedzselendő szoftver konfigurációs állományát/állományait, hol van(nak) ez(ek) a fájl(ok) és mi a szintaxisuk. Ezen kívül már csak egy egyszerű szövegszerkesztőre van szükség, és minden megoldható.

Sajnos(?) nem lehet megkerülni a tényt: a Linux disztribúciókban egyre több kényelmi funkció jelenik meg, egyre több szolgáltatáshoz készülnek grafikus adminisztrációs eszközök, amelyek néha meglehetősen komplex szoftverrendszerek viszonylag egyszerű kezelését oldják meg. Ráadásul egyre kevesebb rendszergazda kell felügyeljen egyre több (és többféle) rendszert. Ezért aztán ennek a résznek a végére egy rövid áttekintés keretében megemlítjük azokat a lehetőségeket is, amelyek arra szolgálnak, hogy egy távoli gép grafikus felületét elérjük.

Elsőként egy kakukktojás. A Unix/Linux-világ hagyományos grafikus felülete az X Window System névre hallgató, kliens-szerver alapon működő felület. Az X-szerver egy speciális program, amely a felhasználóval kommunikál, kezeli a beviteli (billentyűzet, egér) és megjelenítő (monitor) eszközöket. Tipikusan azon a gépen fut, amelyik elé a felhasználó leül dolgozni. Az X-kliens pedig az a program, amelyik (esetleg egy másik gépen futva) csinál valamit, aminek az eredményét egy ablakban meg lehet nézni, aminek a futását befolyásolhatjuk a billentyűzetről vagy az egérrel bevitt adatok segítségével. Mivel az X már eredetileg is kliens-szerver felépítésű, természetesen része, hogy a két fő komponens (az X-szerver és az X-kliens) nem ugyanott fut. Azaz a legelső megoldás, hogy ha a saját gépemen fut már az X-szerver, akkor egy távoli gépen futtathatok X-kliens alkalmazást, és megmondhatom neki, hogy a megjelenítést, és adatbeolvasást az én gépemen futó X-szerveren keresztül oldja meg. Ehhez először be kell jelentkezni a távoli gépre – akár parancssoros felületen keresztül. A futtatásnak két egyszerű módja: vagy van az eszköznek erre a célra parancssori kapcsolója – ez tipikusan a -display (ritkábban -d) –, ebben az esetben:

```
xterm -d SZERVEREM:0.0
```

formában kell futtatni, vagy pedig a DISPLAY nevű környezeti változó beállításával jelezhető, hogy hol az X-szerver, amihez a kliens csatlakozni fog. Azaz:

```
DISPLAY=SZERVEREM:0.0
export DISPLAY
xterm
```

formában.

A korábban már említett ssh ehhez is nyújt segítséget, az ssh parancssorában megadható -X (vagy -Y) opció egy ún. X11-forwarding funkciót valósít meg, ez leegyszerűsítve azt jelenti, hogy a távoli gépre ssh-n keresztüli bejelentkezés során az ssh létrehoz egy alagutat, beállítja az előbb említett DISPLAY változót, így tehát tetszőleges grafikus alkalmazás automatikusan az ssh által biztosított titkosított alagúton keresztül, a lokális gépünkre fogja továbbítani a grafikus alkalmazás adatait. Gyakorlatilag a fenti eredményre vezet a következő parancs:

```
ssh -Y user@host /usr/bin/xterm
```

A grafikus eszközök távoli elérésének másik módja, amikor nem csak egy-egy alkalmazást, hanem akár a teljes futási környezetet átvesszük. Ennek oka lehet az, hogy segítenünk kell egy felhasználónak, és könnyebb hálózaton keresztül elérni a gépét, mint odamenni. Vagy az, hogy szeretnénk valamit a csoda grafikus eszközzel elérni, de sajnálatos módon csak az ikonját ismerjük fel az asztalon/tálcán, de a nevét azt nem tudjuk.

Rengeteg népszerű megoldás létezik a távoli asztal megjelenítésére, függően a technológiától amely segítségével kapcsolatot lehet teremteni két kliens között. Lehetséges akár SSH-alagút, akár

VPN megoldások közbeiktatásával, amennyiben 2 publikus IP között, vagy egy belső hálózaton a gépek közvetlen tudnak egymáshoz csatlakozni. Praktikus megoldást jelent sok esetben az IT osztály munkatársai számára, ha az érdekeltségi körükbe tartozó klienseket úgy tudják támogatni, hogy közben pontosan látják a másik gép képernyőjét. Ez azonban felvet természetesen adatbiztonsági és személyiségi jogi kérdéseket is, azonban ha ezek tisztázottak, akkor valóban nagy könnyebbség úgy segíteni egy program telepítésében (vagy akár feltelepíteni neki a programot), hogy a lokális gépünk segítségével „közel hozzuk" a távol gép képernyőjét, billentyűzetét és egerét. Az ilyen szoftverek hasznosak továbbá a keresztplatformok esetében is. Azaz egy linuxos VNC klienssel el tudunk érni egy Windows asztalt, ha azon fut egy VNC-szerver és természetesen viszszafelé is működőképes a módszer[70].

UltraVNC[71]: egy GPL licenc alatt kiadott teljes funkcionalitású távoli asztal elérésére szolgáló kliens-szerver szoftver. Ideális Windowst futtató gépeken, mivel támogatja a titkosítást, azonban a kliens változat Linux alá csak böngészőből, Java használatával lehetséges.

RealVNC[72]: privát használatra ingyenes, de számos fizetős megoldás létezik belőle. A kliens és a szerver is elérhető a legtöbb Linux terjesztésben csomagként. Titkosítást az ingyenes verzió nem támogat.

Rdesktop[73]: a linuxos kliens szoftver GPL licenc alatt elérhető, és mivel natívan támogatja az RDP protokollt, ezért jól használható a Windows kliensek elérésére Linux alól is.

Spice: a Spice protokollt a Red Hat tette nyílt forrásúvá 2009-ben. Számtalan előnye mutatkozik a VNC-vel szemben, tekintve, hogy sokkal hatékonyabb tömörítést és eleve titkosítást is használ. Mára már integrálható a KVM alá is, illetve használható sima asztal-asztal összekötésként is.

Szinte közös jellemzője az összesnek, hogy az ingyenesen használható felületen a titkosítás vagy nem érhető el, vagy nem elégséges, ezért ezek használata jellemzően belső hálózatokon a védett zónán belül javasolt, vagy VPN vagy SSH-alagút segítségével együtt.

70. Ehhez Linux alá sok felhasználóbarát eszköz létezik, mint pl. a Vinagre vagy a Remmina.
71. http://www.uvnc.com/
72. http://www.realvnc.com/
73. http://www.rdesktop.org/

A naplózó alrendszer, naplók elemzése

Talán nincs is különösebben magyarázatra szükség, hogy miért is szükséges a fenti fogalmak megvalósítás minden egyes szerver esetébe. Elég egy felhasználó vagy egy webmester reklamációja az aktuális cron-ból lefuttatott scriptjéről, amely valamiért nem futott le és máris érezni fogjuk annak a hiányát, hogy néhány syslog esetében a cron.log nincs automatikusan bekapcsolva. A legtöbb esetben egy komolyabb problémát gyakran megelőz több kisebb fennakadás, amelyet ha megfelelően monitorozunk és a megfelelő formában értelmezni tudjuk (grafikusan, stb) vagy kapunk róla értesítést, akkor sok esetben megelőzhető a nagyobb baj. Gyakori hiba pl. amikor a RAID-tömbünk aktuális státuszát monitorozzuk, de az egyes diszkek állapotát nem. Ha a raid tömb logikai állapotát kérjük csak le, az még akkor is OK lehet, ha már átállt a meleg tartalék diszkre (hotspare disk). Ha hozzánézzük a fizikai állapotot is, akkor már látni fogjuk, hogy valami nem stimmel. Hasonló hiba lehet az is, ha mindent naplózunk és a nagy adatmennyiség miatt (nem látjuk a fától az erdőt) nem veszünk észre egy behatolási próbálkozást, ami – ha elég kitartó, akkor – akár sikeres is lehet. Egy átlagos szerver üzemeltetésében a naplózás és a folyamatos monitorozás legalább olyan figyelmet kellene, hogy kapjon, mint a biztonság (hiszen része annak) vagy a mentés. Nézzük elsőként a **logolás** lehetséges eszközeit:

Minden Linux terjesztés használ valamilyen log kezelő programot. Az már a terjesztést összeállítók döntése, hogy melyiket. Jellemzően a piacot az rsyslog, a syslogd és a syslog-ng uralja. Persze ezeken kívül más megoldás is létezik, de jellemzően ezek az alaptelepítésben elérhető eszközök.

A részletesebb kifejtés előtt pár alapfogalom tisztázása: a szabványos syslog minden egyes naplóbejegyzéshez két minősítő adatot kapcsol:

– facility (magyarul talán a legjobban a kategória kifejezéssel lehetne leírni a lényegét) – ezzel beskatulyázom a naplóbejegyzés forrását (a kerneltől érkezett, a levelező rendszertől, a nyomtató alrendszertől, és í. t.)
– priority, más néven level (fontosság, vagy szint) – mennyire kritikus az adott üzenet; hagyományosan ha a naplózó szoftvernek megadok egy fontossági szintet, akkor az olyan szintű, vagy annál magasabb szinten levő üzeneteket fogja a megadott célhoz eljuttatni

Azaz egy adott naplóbejegyzés (amihez tartozik természetesen az üzeneten kívül egy időbélyeg is) honnan érkezett (milyen szoftver alrendszer küldte az üzenetet), és mennyire fontos. Általában a különböző syslog-implementációk eltérnek egymástól abban, hogy a forrás-cél meghatározást mennyire finoman lehet hangolni és milyen szintaxissal kell megadni az üzenetek szétválogatását. A forrás helyett eléggé elterjedt a selector (kiválasztó) elnevezés. A facility paraméterek között léteznek teljesen specifikusak – mint lpr vagy mail, és olyan meglehetősen általánosak is, mint daemon vagy user. (A teljes lista egyébként biztosan tartalmazza a következőket: auth, authpriv, cron, daemon, ftp, kern, lpr, mail, mark, news, syslog, user, uucp és local0, local1, ..., local7. Bizonyos implementációkban vannak ettől eltérőek is.) A priority paraméter pedig a következők valamelyike: debug, info, notice, warning, err, crit, alert, emerg (a sorrend egyre magasabb prioritást takar), és léteznek itt nem szereplő (ezek valamelyikével megegyező jelentésű, de már elavult) kulcsszavak is. Megjegyzendő, hogy egyre több szoftver írója gondolja úgy, hogy az alkalmazás naplózására a local0-local7 kategóriák valamelyike a legalkalmasabb – csak éppen ezek a kategóriák sokszor a gyári konfigurációban nincsenek lekezelve, így át nem gondolt konfiguráció esetén elveszhetnek ezek a naplóbejegyzések.

Azok az alkalmazások, amelyek a szabványos openlog(3)/syslog(3)/closelog(3) könyvtári rutinokat használják naplózásra, a /dev/log nevű UNIX-domain socketen keresztül küldik az üzeneteiket, míg a kernel log üzenetei Linux alatt a /proc/kmsg fájlon keresztül érhetők el. Ha pedig engedélyezzük távoli gépekről a naplózást, a szabványos syslog-port az 514/UDP.

Syslogd

A BSD UNIX-rendszerekből átvett syslog megvalósítás alapfunkciókat nyújt: lokális és távoli naplózás. A naplóbejegyzések lokálisan küldhetőek adott (bejelentkezett) felhasználók képernyőjére, (teljes elérési úttal adott) fájlba (ezek természetesen lehetnek eszközfájlok is, ily módon küldhető a /dev/console, vagy a /dev/printer eszközre). Küldhető csövön (pipe, FIFO) keresztül programnak, illetve az összes bejelentkezett felhasználónak (ilyet leginkább kritikusnak minősített rendszerüzenetek esetén szoktak csinálni). Természetesen ugyanazt a napóbejegyzést több különböző címzetthez is el lehet juttatni (azaz a klasszikus, papírra nyomtatva naplózás mellett mehet a központi naplószerverre és a rendszergazdák terminájára is valamely fontos naplóbejegyzés). A hagyományos syslog megvalósítás hiányosságai közé szokták sorolni, hogy a távoli naplózáshoz UDP-protokollt használ, mindenféle titkosítás és hitelesítés nélkül (így viszonylag könnyen lehet naplót hamisítani). (Megjegyzendő, hogy egy elég sajátos megjegyzés-szintaxis használatával a syslogd képes akár a naplót küldő program neve alapján is szelektálni[74], de ezt a tulajdonságát nem nagyon szokták dokumentálni.)

Íme egy példa syslog.conf

```
*.err;kern.warning;auth.notice;mail.crit    /dev/console
*.notice;auth.none;kern.debug                    @logserver.example.hu
security.*                                  root,sysadm
kern.crit                                   *
```

Mint a példából látható, többféle forrás is megadható egyszerre, ezzel lehet különböző alrendszerek üzeneteit azonos helyre tárolva naplózni. Nézzük részletesen a fenti példát:

A *.err jelentése: bármely alrendszertől érkezik ERROR (vagy annál magasabb[75]) prioritású üzenet, azt naplózzuk (a példa szerint a rendszerkonzolra). Mivel ebben a sorban ;-vel elválasztva több forrást is megadtunk, így a konzolon a kerneltől érkező WARNING (vagy magasabb) fokozatú üzenetek, az autentikációs alrendszer NOTICE (és magasabb), valamint a levelező alrendszer CRIT (nyilván kritikus) szintet elérő üzenetei szintén megjelennek.

A következő sorban szereplő bejegyzés azt mutatja, hogy mi módon lehet egy távoli gépre (a példában a logserver.example.hu nevűre) átküldeni a naplókat. (Mivel a távoli gép syslog programja a saját konfigurációjának függvényében szintén szelektál, ügyeljünk arra, hogy ne pingpongozzunk a log üzenetekkel – drága és kevéssé hatékony.) Ebben a sorban még egy furcsaság látszik. A *.notice (bárhonnan érkezik NOTICE szintű vagy annál magasabb prioritású naplózni való) után álló ;auth.none explicit módon kizárja az autentikációs alrendszertől érkező üzeneteket ebből a naplózásból. Ezen kívül a selector-ban szereplő ;kern.debug azt jelenti, hogy a kerneltől viszont a debug vagy magasabb szintű üzeneteket kell a távoli gépre átküldeni (azaz mivel a debug a legalacsonyabb prioritás, így mindent).

74. http://www.**freebsd**.org/cgi/**man**.cgi?query=**syslog.conf**&sektion=5
75. kevéssé ismert, hogy *.=err formával lehet pontosan csak az adott szintű üzenetekre hivatkozni (és megadhatók a matematikából jól ismert !<=> operátorok)

A harmadik sorban szereplő beállítás szerint a biztonsági alrendszer üzenetei a root és sysadm nevű felhasználók képernyőjén jelennek meg (amennyiben be vannak jelentkezve).

Végül az utolsó példa szerint a kernel kritikus (és magasabb szintű) üzenetei minden bejelentkezett felhasználó terminálján megjelennek.

Komoly buktató, hogy a mai napig vannak olyan syslogd implementációk, melyek a konfigurációs fájl szintaxisára nagyon érzékenyek. Így például a sorok elején álló forrást és a sorok végén álló célt eredendően csak TAB karakterekkel lehetett elválasztani, a szokványos szóköz nem megengedett. E miatt a példákban (noha nem látszik) mi is azt használunk és erre buzdítunk mindenkit. Hasonlóan kellemetlen hibája a hagyományos syslogd-nek, hogy amennyiben fájlba naplóz, a fájloknak a syslogd indulásakor már léteznie kell, ugyanis a syslogd nem hozza létre ezeket a naplófájlokat.

Rsyslog[76]

Egy GPLv3-as licenc alatt napvilágot látott projekt, ahol a fő hangsúly a biztonságos logoláson van (talán ezért is állhatott át a Debian[77] és az Ubuntu is rá). Nézzük pontosan mit is tud az Rsyslogd:

- 100%-ban kompatibilis az eredeti Sysloggal: ez a gyakorlatban azt jelenti, hogy a konfig állományok szintaxisa megegyezik.
- Moduláris felépítés
- UDP és TCP-alapú távoli logolási lehetőség és távoli log fogadási lehetőség
- TCP SSL hitelesítéssel is: a távoli logolás kiegészítése SSL titkosítással, ez az egyik legnagyobb különbséges a régi syslogd-hez képest.
- Tömörített küldés és fogadás
- Backup szerverre való automatikus átállás
- Átláthatóbb és pontosabb időbélyeg rendszer
- Különböző szabályos kifejezésekkel (regexp) megvalósítható szűrési feltételek
- IPv6 protokoll ismerete
- natív MySQL/MariaDB és PostgreSQL támogatás: azaz direkt naplózási lehetőség SQL kapcsolattal, amely a kiértékeléseket is nagyban meg tudja könnyíteni.

Az Rsyslog egy igazi hiánypótló megoldás volt, amelyet a terjesztések gyorsan illesztettek a saját rendszereikhez, így ma a sűrűn használt terjesztések a tinédzserkort is bőven megért Syslog helyett szinte mind ezt használják.

76. http://www.rsyslog.com/
77. http://wiki.debian.org/Rsyslog

Syslog-ng[78]

Magyar fejlesztés. Az erdeti syslogból alakították ki, annak hiányosságait szem előtt tartva a fejlesztés során. A projektet 1998-ban indították és mára igen elterjedt naplózó eszköz lett. Elérhető egy nyílt forráskódú megoldás belőle, amely a legtöbb terjesztés alá csomagkezelő segítségével telepíthető, és több fizetős megoldás is a speciális nagyvállalati szegmens részére. Tudása:

– Titkosított naplózás
– Naplóüzenetek biztosított továbbítása
– Szabványos syslog protokollok támogatása
– Lokális üzenetek gyűjtése
– Nagy teljesítményű naplózás, fejlett üzenetfeldolgozási képességekkel és közvetlen adatbázis kezeléssel.
– Üzenetek szűrése és rendszerezése, feldolgozása és módosítás
– Üzenetek klasszifikálása
– Extrém terhelhetőség
– IPv4 és IPv6 támogatás
– Pattern DB: amely segítségével a naplóüzenetek valós idejű feldolgozása megoldott a felhasználói közösség által összegyűjtött mintázatok (pattern-ek) alapján.
– Széles körű magyar és angol nyelvű dokumentáció

Az első példában szereplő syslog.conf fájlnak megfelelő syslog-ng.conf:

```
source s_local {
internal; # syslog-ng generated logs
unix-stream("/dev/log"); # the standard syslog-functions
file("/proc/kmsg"); # kernel logs
}
destination d_console { file("/dev/console"); };
destination d_syslogserver { udp( "logserver.example.hu" port( 514 ) ); };
destination d_user_root { usertty( "root" ); };
destination d_user_sysadm {     usertty( "sysadm" ) ; };
destination d_all_loggedin_users { usertty( "*" ) ; };
filter f_err { level( err .. emerg ); };
filter f_warning { level( warning .. emerg ) ; };
filter f_notice { level( notice .. emerg ) ; };
filter f_crit { level( crit ..emerg ) ; };
filter f_debug { level( debug .. emerg ) ; };
filter f_all_level { level( debug .. emerg ) ; };
filter f_no_level { not level( debug .. emerg ) ; };
filter f_all_facility { facility( auth, authprov, cron, daemon, kern, lpr, mail,
  news, user, cron, local0 .. local7 ) ; };
filter f_kern { facility( kern ) ; };
filter f_auth { facility( auth ) ; };
filter f_mail { facility( mail ); };
filter f_security { facility( security ) ; };
filter f_auth_none { facility( auth ) and filter( f_no_level ) ; } ;
```

78. http://en.wikipedia.org/wiki/Syslog-ng

```
log { source( s_local ) ; filter ( f_all_facility ) ; filter( f_err ) ;
  destination( d_console ) ; } ;
log { source( s_local ) ; filter( f_kern ) ; filter( f_warning ) ;
  destination( d_console ) ; } ;
log { source( s_local ) ; filter( f_auth ) ; filter( f_notice ) ;
  destination( d_console ) ; } ;
log { source( s_local ) ; filter( f_mail ) ; filter( f_crit ) ;
  destination( d_console ) ; } ;
log { source( s_local ) ; filter( f_all_facility ) ; filter( f_notice ) ;
  filter( f_auth_none ) ;  filter( f_kern ) ; filter( f_debug ) ;
  destination( d_syslogserver ) ; };
log { source( s_local ) ; filter( f_security ) ; filter( f_all_level ) ;
  destination( d_user_root ) ; destination( d_user_sysadm ) ; } ;
log { source( s_local ) ; filter( f_kern ) ; filter( f_crit ) ;
  destination( d_all_loggedin_user ) ; } ;
```

Fenti rendszerek általában alap beállításokkal kerülnek a terjesztésekbe, így az esetlegesen nem logolt területek miatt konfigurációjukat érdemes beüzemelés előtt átnézni és azt a saját rendszerünkre szabni.

Mivel a naplózás meglehetősen nagyméretű fájlokat eredményezhet, ezért érdemes a naplófájlok kezelését automatizálni. (Azaz pl. egy adott fájlméret, vagy diszktelítettség elérésekor a régi naplófájlt bezárni, valamilyen konvenció szerint átnevezni, esetleg tömöríteni is, és új naplófájlt létrehozni – esetleg mindezt a naplószerverrel közölni is.) A feladat némi scripteléssel is elintézhető, de léteznek hozzá kész eszközök is (mint pl. a logrotate, vagy a newsyslog).

A logoláshoz kapcsolható még jó pár megoldás, amely esetében a monitorozás és a logolás közös kapcsolódási pont:

snoopy[79]

Egy igazán remek, felhasználói programok aktivitását figyelő program. A snoopy segítségével nyomon tudjuk követni akár egy PHP program futását vagy egy konzol vagy SSH kapcsolattal rendelkező (shell) felhasználó aktivitását az /var/log/auth.log ban naplózva. Segítségével akár egy szándékosan letörölt shell history állomány tartalmát is rekonstruálni lehet. Egy minta log, amelyben egy login parancsot hajtottunk végre sudo-val, majd megnyitottuk az auth.log-ot:

```
Nov 10 09:24:46 mail snoopy[17426]: [user, uid:1000 sid:17426]: -tcsh
Nov 10 09:24:46 mail snoopy[17429]: [user, uid:1000 sid:17426]: ls
  /etc/csh/login.d
Nov 10 09:24:46 mail snoopy[17429]: [user, uid:1000 sid:17426]: ls
  /etc/csh/login.d
Nov 10 09:24:46 mail snoopy[17431]: [user, uid:1000 sid:17426]: uname -n
Nov 10 09:24:46 mail snoopy[17433]: [user, uid:1000 sid:17426]: /usr/bin/whoami
Nov 10 09:24:46 mail snoopy[17435]: [user, uid:1000 sid:17426]: /bin/hostname
Nov 10 09:24:47 mail snoopy[17436]: [user, uid:1000 sid:17426]: sudo -s
Nov 10 09:24:50 mail sudo:    user : TTY=pts/0 ; PWD=/home/user ; USER=root ;
  COMMAND=/usr/bin/tcsh
```

79. http://code.google.com/p/snoopy/

```
Nov 10 09:24:50 mail snoopy[17437]: [user, uid:0 sid:17426]: /usr/bin/tcsh
Nov 10 09:24:54 mail snoopy[17439]: [user, uid:0 sid:17426]: less
  /var/log/auth.log
```

RAID figyelő megoldások

Az talán senkinek nem kétséges, hogy ha már RAID-rendszer van a szerverben, akkor nem feltétlen a véletlenre kell bízni annak észlelését, hogy az egyik vagy másik lemezünk meghibásodott. Természetesen ha nincs hardveres RAID-vezérlő a gépben, akkor még fontosabb a diszkekre vonatkozó figyelmeztetések és egyéb előjelek figyelése. Sok esetben kézzel írt scriptekkel (amelyek a különböző állományokat figyelik: messages.log, syslog, stb) is meg lehet oldani ezeket a dolgokat, de jó pár kész megoldás is rendelkezésünkre áll:

smartmontools[80]

Eszköz a SMART[81] adatokkal rendelkező merevlemezek megfigyelésére. A legtöbb ma kapható lemez már rendelkezik azzal a képességgel, hogy folyamatosan és/vagy meghatározott időközönként önellenőrzéseket futtat le, figyeli a diszkek környezeti pramétereit: melegedést, felpörgési időt, élettartam időt, ki/be kapcsolások számát, stb. Ezekből az adatokból következtet arra, hogy egy merevlemez lehetséges meghibásodása be fog-e következni vagy sem. Felhasználói környezetben ezen adatok megtekintésére vagy a parancssori megoldás (smartctl -a /dev/sda1), vagy pedig a jobban átlátható GSmartControl[82] ajánlott. Beállítása viszonylag egyszerű és a gyakorlati tapasztalat az, hogy a jól beállított önellenőrzési ciklusokat is magában foglaló konfiguráció képes lehet még az adatvesztést megelőzően jelezni. Szerveroldalon képes együttműködni a gyártók saját platformos Linux RAID programjaival, de sajnos nem mindegyikkel.[83]

Telepítése egyszerű:

apt-get install smartmontools

a függőségekkel együtt települ a smartmoontools. Első teendőnk, hogy eldöntsük, milyen diskeket akarunk figyeltetni vele. Célszerű az összes smart kompatibilis rendelkezésre álló lemezt megfigyelni. Szoftveres RAID esetén, egyenként az összes fizikai diszket be kell fűzni, hardveres RAID esetén pedig érdemes a gyártók hozzá adott szoftverét is használni, illetve megnézni, hogy az adott HW-vezérlő és a smartmon együtt tud-e működni. De nézzük az egyszerű esetet, amikor is 1-2 SATA vagy SAS lemez van a gépünkben, vagy SoftRaid vagy single üzemmódban. Nézzük meg, hogy a diszkeink alkalmasak-e a smart adatok kinyerésére:

smartctl -a /dev/sda

és **smartctl -a /dev/sdb**

valami ilyesmit kellene kapnunk:

80. http://sourceforge.net/apps/trac/smartmontools/wiki
81. http://en.wikipedia.org/wiki/S.M.A.R.T.
82. http://gsmartcontrol.berlios.de/home/index.php/en/Home
83. http://sourceforge.net/apps/trac/smartmontools/wiki/Supported_RAID-Controllers

```
=== START OF INFORMATION SECTION ===
  Device Model:     ST91000640NS
  Serial Number:
  LU WWN Device Id: 5 000c50 04ed10873
  Firmware Version: FTA2
  User Capacity:    1,000,204,886,016 bytes [1.00 TB]
  Sector Size:      512 bytes logical/physical
  Device is:        Not in smartctl database [for details use: -P showall]
  ATA Version is:   8
  ATA Standard is:  ATA-8-ACS revision 4
  Local Time is:    Thu Aug  8 14:36:43 2013 CEST
  SMART support is: Available - device has SMART capability.
  SMART support is: Disabled

  === START OF READ SMART DATA SECTION ===
  SMART overall-health self-assessment test result: PASSED
```

és még egy halom olyan adatot, hogy hány fokos a lemez éppen, illetve hányszor volt ki be kapcsolva, vagy hogy mennyi ideje pörög már. A lényeg, hogy a SMART adatok kiolvashatóak, így akkor állítsuk be a SMART flag-et ENABLED állásba a következőképpen:

```
smartctl -s on -a /dev/sda
smartctl -s on -a /dev/sdb
```

Ha ezzel megvagyunk, akkor tudassuk a rendszerrel, hogy szeretnénk, ha bizonyos időközönként figyelné a lemezeinket, szerkesszük a /etc/default/smartmontools állományt:

```
enable_smart="/dev/sda /dev/sdb"
```

itt adhatjuk meg, hogy a fizikai lemezeinket hogyan hívják, még akkor is ha valójában mi /dev/md1 ként hivatkozunk rá. Jelen példa, mind a két soft raid disket figyelni fogja.

```
start_smartd=yes
```

alap esetben off -on vagy kommentezve található, ha átraktuk on-ra, akkor a rendszer indításnál automatikusan figyelni fogja a lemezeket is.

```
smartd_opts="--interval=1800"
```

megadhatjuk másodpercben a figyelés intenzitását. (Figyeljünk oda, hogy ha a smartd-t engedélyezzük, akkor általában nem kell, sőt ellenjavalt felsorolni a figyelendő diszkeket, ugyanis a smartd alapbeállítással automatikusan megkeresi magának.)

Az alap opciók megadása után egy **service smartd restart** -tal tudjuk root-ként (vagy sudo-val) érvényesíteni. A /var/og/syslog -ban pedig már látható is lesz, hogy a smartd figyeli a diskjeinket. Az alap beállítások mellett a root fog levelet kapni, ha várható vagy bekövetkezett egy lemez meghibásodás. Az alap opciókat a /etc/smartd.conf -állományban tudjuk finomítani, pl ha másnak szeretnénk, hogy figyelmeztetés menjen, vagy ha egy komplett támogatott HW RAID -et akarunk figyelni.

HW RAID figyelése

Számos hardveres RAID megoldás létezik a piacon. Szerencsére a gyártók is felismerték azon igényeket, hogy az eszközeiket nem csak használni szeretnénk, hanem megfigyelni is. Így számos gyártói szoftvermegoldás született, amely nem feltétlen szabad szoftver (l. a Debian Wiki által karbantartott listát).[84] Érdemes egyedileg megnézni ezeket a segédprogramokat, mert felhasználásukat tekintve ingyenesek, de a linceszük alapján nem mindig szabad szoftverek.

Jellemzően mindegyik program igényel valamilyen minimális scriptelést, amely segítségével automatizálni tudjuk pl. az óránkénti státusz lekérdezést, vagy csak a hiba vagy hiba előrejelzés megfigyelését, de ezek jellemzően 2-3 parancs 1 shell sciptbe gyűjtése, illetve esetenként egy diff vagy egyéb alap parancs kombinációját jelentik. A script eredményét pedig e-mailben elküldve és/vagy log szerverben feldolgozva kaphatunk értesítéseket a RAID aktuális állapotáról.

84. http://wiki.debian.org/LinuxRaidForAdmins

A hálózati szolgáltatások monitorozása: Nagios

Miért monitorozzunk?

A szervernek mennie kell. A hibamentes működés a felhasználók természetes elvárása hisz ők dolgozni akarnak. Ezért az üzemeltetőnek igyekeznie kell a lehetséges problémákat előre jelezni, és azokat még az előtt elhárítani, mielőtt azok bekövetkeznének. Ha mégis váratlan hiba történik, akkor viszont az adminok szeretnének először tudni róla, hogy a lehető leghamarabb megkezdhessék a hiba elhárítását. Ennek érdekében a rendszer összes vizsgálható paraméterét érdemes folyamatosan figyelemmel kísérni.

Az figyelendő paramétereket több családba sorolhatjuk. A teljesítmény paraméterek megmutatják, hogy egy rendszerben a különböző erőforrások mennyire vannak kihasználva. Ezek a legtöbb esetben valamilyen számmal vagy százalékos értékkel kifejezhető értékek. Ilyen jellemzők például a gépben lévő diszkek foglaltsága, terheltsége, a processzormagok kihasználtsága, a rendelkezésre álló szabad memória, a hálózati csatlakozók ki és bemeneti sávszélesség használata, az adatbázisszerver válaszideje és még folytathatnánk. Ezek változásából előre lehet jelezni, hogy ha a szerver működésében várható hiba fog beállni. Látható például, hogy ha egy szerveren minden egyes új felhasználóval folyamatosan nő a memória felhasználás és a lefoglalt diszkterület, akkor rendszeres méréssel és a felhasználók számának előre becslésével lehetőség van a memória vagy diszk bővítésére még mielőtt azok valamelyike problémát okozna.

A következő ellenőrzendő paraméter típus a szolgáltatások és egyéb jellemzők állapotai. Egy szerveren tipikusan nagyon sok szoftver biztosítja annak működését. Ha valamelyik nem fut (például az időzítő démon, a cron), akkor az előbb-utóbb a szolgáltatások működését veszélyezteti. Ezen állapotokat a legtöbb esetben egy igen/nem vagy 1/0 értékekkel lehet kifejezni. Az előző példánál maradva leellenőrizhető, hogy a cron démon fut-e a rendszeren. Egy másik példa: van-e a rendszeren frissítendő csomag?

Történeti és egyéb paramétereken olyan ellenőrizhető jellemzőket értünk, hogy volt-e újraindítás a legutóbbi adminisztrátor által végzett nyugta óta. Ez olyankor fordulhat elő, ha áramkimaradás miatt a nem biztonságos áramellátásra kötött gép nem tervezett módon újraindul. Ilyenkor az adminisztrátornak ki kell derítenie, hogy mi okozta a problémát, és lehetőleg elejét kell vennie, hogy a későbbiekben hasonló probléma előforduljon.

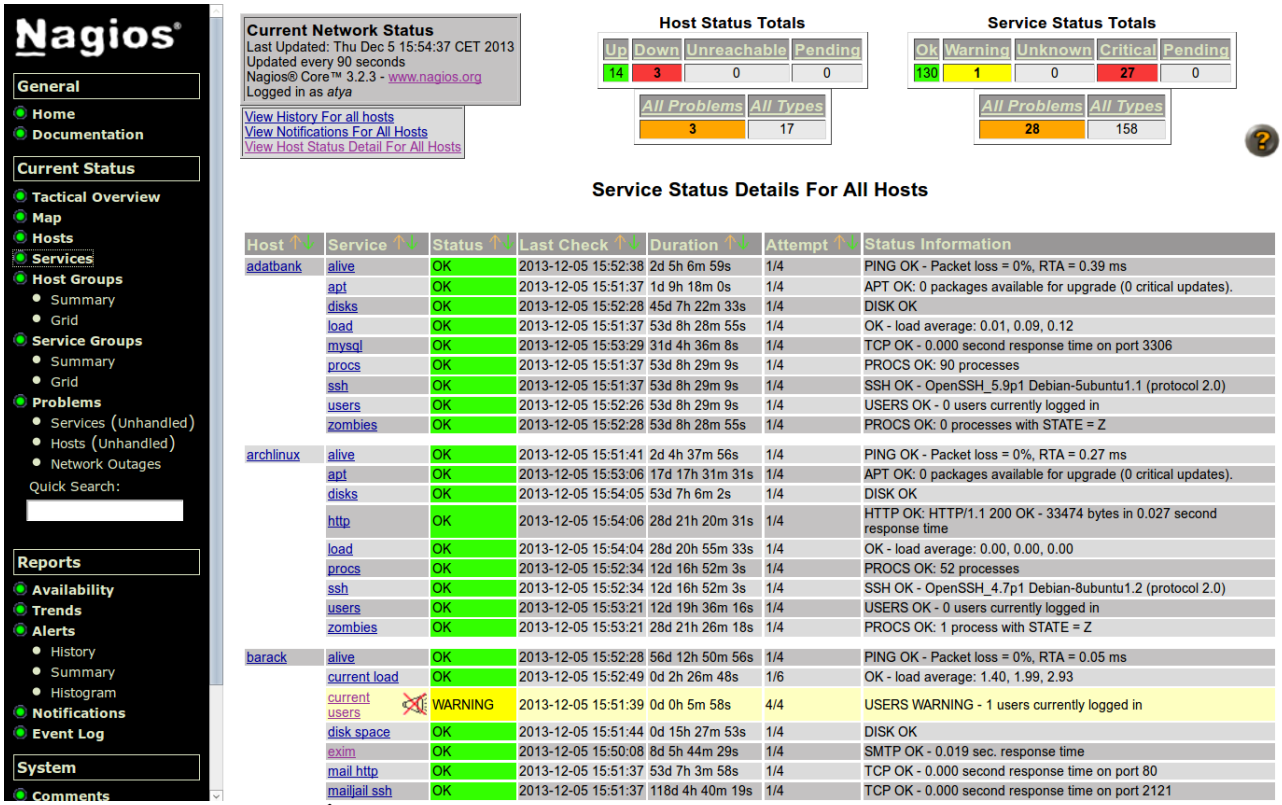
A szerverek jellemzőinek monitorozásával kapcsolatban általánosságban elmondható, hogy ha a jelző rendszer valamilyen hiba bekövetkezéséről tájékoztatja az adminisztrátorokat, akkor a hiba elhárítása mellett minden esetben az a legfontosabb teendő, hogy a személyzet gondoskodjon arról, hogy az adott hiba lehetőleg többé ne forduljon elő. Ha ezt saját hatáskörben nem tudják megoldani (például egy szünetmentes táp vagy egy nagyobb diszk beszerzése szükséges), akkor a problémát az illetékes felé jelezni kell.

Működő képesség monitorozása

A rendszerek különböző paramétereinek ellenőrzésére számos szoftver létezik. Mi az egyik legelterjedtebb, könnyen kezelhető, jól testre szabható megoldást választottuk ki ezek közül. A Nagios nevű szoftver szinte iparági szabványnak számít. A többi hasonló célú rendszer mellett is lehet érveket találni (például egyszerűbb üzembe helyezés, speciális tulajdonságok stb.), azonban a tapasztalataink szerint a Nagios szinte minden feladatra megfelelő. Érdemes ezt megismerni még akkor is, ha eleinte a rendszergazda csak egyetlen rendszert akar ellenőrizni, hisz így olyan eszközt kap, mely később alkalmas lesz sokkal nagyobb rendszerek esetén is.

A Nagios alapvető jellemzői

A Nagios egy remekül skálázható, rugalmas automatikus ellenőrző rendszer. Kényelmes, web alapú felülete egyszerűen átlátható, használható.



1. Ábra: A Nagios szolgáltatások állapotát mutató felülete

A rendszer a következő alapvető tulajdonságokkal bír:

– szabad szoftver, GPL v2 licenc alatt érhető el, így mindenféle célra szabadon használható;
– sokoldalú: lehetőség van alkalmazások, mérhető rendszer jellemzők, hálózati szolgáltatások, operációs rendszerek, hálózati protokollok és infrastruktúra komponensek felügyeletére;
– sokoldalú ellenőrző modul API lehetővé teszi a speciális, saját fejlesztésű alkalmazások felügyeletét, saját ellenőrző modulok fejlesztését;

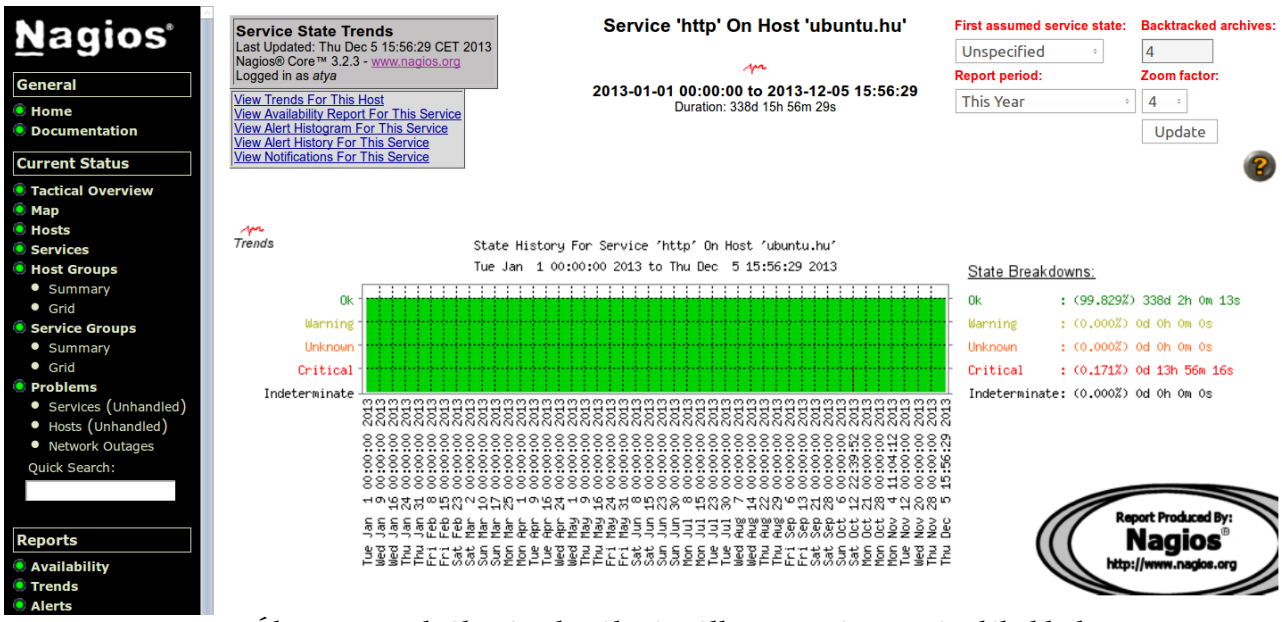
– kényelmesen használható web alapú felület az ellenőrzött rendszerek állapotának megfigyelésére és bizonyos beavatkozások elvégzésére;
– figyelmeztetések e-mail-ben vagy más parancssorból elérhető illesztő felületen keresztül (például webszolgáltatáson elérhető SMS API megfelelő segédprogrammal);
– többfelhasználós, a felhasználók csoportokba rendezhetők;
– az ellenőrzött gépek csoportokba rendezhetők;
– az ellenőrzött gépek és szolgáltatások szabadon felhasználókhoz vagy csoportokhoz rendelhetők, melyek értesítéseket kapnak, a felületen ellenőrizhetik a beállított rendszereket vagy szolgáltatásokat;
– a felhasználók a beállításoknak megfelelően csak azokat a rendszereket láthatják, melyek az ő felügyeletükhöz tartoznak;
– ha a hiba nem hárul el megadott időn belül, akkor a jelzések eszkalálhatók;
– ha az ellenőrzendő szolgáltatás nem érhető el közvetlenül a felügyeleti rendszerről, akkor az ellenőrzéseket táv ellenőrző kiegészítéssel (NRPE) másik elérhető gépen keresztül el lehet végezni;
– eseménykezelők segítségével automatizált reakciók állíthatók be, például egy szolgáltatás elérhetetlensége esetén az adott szerver újraindítható;
– az ellenőrzött rendszerek elérhetőségéről jelentések készíthetők, melyek SLA bizonyítékként használhatók;
– a riasztásokról és egyéb eseményekről beállítható ideig visszamenőleg elérhető a történet, így visszakövethető egy-egy régebbi esemény;
– akár több ezer ellenőrzött rendszerre is képes skálázódni;
– több, mint tíz éves folyamatos fejlesztés, stabil fejlesztői közösség;
– nagy és jól működő felhasználói közösség.

Összességében elmondható, hogy Nagios segítségével lényegében mindent ellenőrizhetünk. Ha valamit nem tudunk megcsinálni, akkor egy nagy, jól működő közösségtől kérhetünk tanácsot, támogatást. Például itt: http://support.nagios.com/forum/

A Nagios technológiai megközelítésből

A rendszer két alapvető részből áll, melyeket kiegészít több kisebb-nagyobb modul, melyek az ellenőrzést, a megjelenítést, a táv-ellenőrzést és egyéb funkciókat látnak el.

A rendszer lelke egy démon, mely a beállítások alapján az ellenőrző, beavatkozó és értesítő modulokat futtatja. Ez az telepítés és a megfelelő beállítások elvégzése után a háttérben fut és szükség esetén például e-mail-ben értesíti a beállított rendszer adminisztrátort, hogy ha a vizsgált szolgáltatásokkal valamilyen gond van.



2. Ábra: Egy szolgáltatás aktuális évi állapot története százalékokkal

A másik alapvető fontosságú rész a web alapú felület, melyet egy cgi-k kiszolgálására alkalmas web szerver alatt kell elhelyezni. Erre a jól ismert és általánosan használt Apache szervert javasoljuk. A rendszer web felület nélkül is működőképes, csak kissé nehezebb az e-mail értesítések alapján átlátni az összes ellenőrzött rendszer aktuális állapotát. Aki attól tart, hogy a felügyeleti rendszer felülete veszélyezteti az őt futtató szerver biztonságát, az megteheti, hogy a web felületet úgy állítja be, hogy csak a 127.0.0.1-es címen figyeljen a web szerver, így a felületet csak helyben, például w3m, vagy ssh és port továbbítás segítségével érheti el (lásd távoli adminisztráció fejezet).

A rendszer számos kiegészítő modullal bővíthető. Az elmúlt évtizedben sok olyan ellenőrző és egyéb modul került be a rendszerbe és érhető el hozzá, melyek segítségével a Nagios amúgy is nagyszerű funkcionalitása tovább bővíthető. Ezek közül egyet szeretnénk bemutatni annak általános használhatósága miatt. Ha egy másik szerver életjeleit szeretnénk ellenőrizni, akkor azt távolról csak nagyon korlátozottan tudjuk elvégezni. Ha a hálózati szabályrendszer lehetővé teszi, akkor ping segítségével le lehet ellenőrizni, hogy a rendszer életben van-e, de nem lehet megnézni, hogy mennyi az aktuális memória felhasználása. Erre a feladatra készítették el az NRPE (Nagios Remote Plugin Executor) kiegészítőt, mely biztonságos módon lehetővé teszi a Nagios modulok távoli futtatását. Bővebben erről a beállítások végén.

A Nagios alaprendszer könnyedén bővíthető további funkciókkal, kiegészítések, kiterjesztések és dokumentációk százai érhetők el a http://exchange.nagios.org/ honlapon.

Telepítés, beállítások

A Nagios csomag telepítése Ubuntu 12.04 LTS szerverre a nagios3 csomag telepítésével végezhető el. Ha még nem volt a rendszeren webszerver vagy kimenő levelezés (utóbbi esetben ejnye!, lásd szerver alap beállítások fejezet), akkor ezeket is magával hozza a telepítés. Web szerverre a felhasználói felület miatt van szükség, melynek alapértelmezetten Ubuntu 12.04 LTS esetén az Apache2 szerver települ. Kimenő levelezés az értesítések kiküldése miatt szükséges. És még vagy ötven másik csomagot is magával ránt a telepítés, melyek az ellenőrző modulok működéséhez szükségesek. Az ellenőrző modulok három csomagban vannak, ezek a nagios-plugins-basic, na-

gios-plugins-standard és a nagios-plugins-extra. Ezek rendre a következő ellenőrző modulokat tartalmazzák:

basic	apt, by_ssh, clamd, cluster, dhcp, disk, dummy, file_age, ftp, host, http, icmp, ide_smart, imap, ircd, jabber, load, log, mrtg, mrtgtraf, nagios, nntp, nntps, nt, ntp, ntp_peer, ntp_time, nwstat, overcr, ping, pop, procs, real, rta_multi, sensors, simap, smtp, spop, ssh, ssmtp, swap, tcp, time, udp, ups, users
standard	bgpstate, breeze, dig, disk_smb, dns, flexlm, hpjd, ifoperstatus, ifstatus, ldap, ldaps, linux_raid, mailq, mysql, mysql_query, oracle, pgsql, radius, rpc, snmp, wave
extra	fping, game

A telepítés közben a rendszer megkérdezi a web-adminisztrátor jelszavát. Ennek megváltoztatását a későbbiekben leírjuk, de nagyon ajánlott bölcsen megválasztani, különösen akkor, ha a felügyeleti szerver felülete minden belső hálózati gépről, vagy akár az egész internetről elérhető lesz. Utóbbit lehetőleg el kell kerülni. Ha külső gépekről is el kell érni a web felületet, akkor szerencsés ssh és port továbbítás vagy VPN használata (lásd távoli menedzsment és VPN fejezetek).

A rendszer egyébként minimális további konfiguráció után üzemkész. Alapesetben elérhető a rendszer IP címén illetve helyben a localhost címen, a /nagios3 könyvtárban. Ha például tisztességesen be van állítva a gép névfeloldása, akkor valami ilyesmi címen:
http://monitor.kisceg.intra/nagios3

Ekkor bekéri a HTTP hitelesítéshez szükséges adatokat, melynél a név nagiosadmin, a jelszó pedig a telepítéskor megadott jelszó. Ha elfelejtettük, akkor lépjük a következő mezőre, ott leírjuk, hogy hogyan kell módosítani vagy úgy felhasználókat felvenni.

Indítás, leállítás, újrakonfigurálás

Mint minden démont, a Nagiost is a következő parancs segítségével kell indítani:

```
/etc/init.d/nagios3 start
```

Leállítani:

```
/etc/init.d/nagios3 stop
```

Egyedül az újraindítással kapcsolatban érdemes tudni, hogy ez általában felesleges és némi kiesést okozhat az ellenőrzésekben, szerencsésebb a beállítások újratöltése. Ha egy konfig. módosítás nem sikerül, akkor az újraindítás le tudja állítani a Nagiost, de már nem tudja újra elindítani. Az újratöltés azonban leellenőrzi a konfigurációs beállítások konzisztenciáját, így ha abban hiba van, akkor nem is kezdi meg az újrakonfigurálást. Tehát újraindítás helyett használjuk ezt:

```
/etc/init.d/nagios3 reload
```

Az Apache beállítása

A Nagios rendszer az Ubuntu 12.04 LTS szerveren alkalmazkodik az Apache beállítási szokásokhoz, és a konfigurációs állományát az /etc/apache2/conf.d könyvtár alá linkeli nagios3.conf néven. Ez a fájl valójában a Nagios beállító könyvtárában, az /etc/nagios3 alatt található. Ennek beállításaiból csak néhány fontosabb megállapítás. A beállítások miatt a Nagios rendszer ellopja a gép webroot-ja alatt a /nagios3 könyvtárat, ahogy ezt már leírtuk. Ebben a könyvtárban leír egy

/cgi-bin könyvtárat, ahol engedélyezi a cgi-k futtatását. Ez valójában a /usr/lib/cgi-bin/nagios3 könyvtár. A html dokumentumok, stíluslapok és cgi-k eléréséhez kötelezővé teszi a HTTP hitelesítést, melynek felhasználói és jelszó hash-ei a /etc/nagios3/htpasswd.users fájlban találhatóak. Ha tehát fel akarunk venni egy új felhasználót vagy valamelyiknek le akarjuk cserélni a jelszavát, akkor ezt a fájlt kell megpiszkálnunk a htpasswd Apache segédprogrammal. Használatáról lásd Apache fejezet.

A beállító fájlok felépítése

A Nagios rendszer konfigurációs fájlja a /etc/nagios3/nagios.cfg. Ebben találhatók a rendszer beállításai, illetve itt van megadva, hogy a további, objektumokhoz (például felhasználók, csoportok, gépek, szolgáltatások stb.) tartozó beállításokat hol találja a rendszer. Ezekhez a legegyszerűbb esetben nem kell hozzányúlni, megfelelőek az alapbeállítások.

Telepítés után a cfg_dir beállítások alapján a /etc/nagios3/conf.d könyvtárban található beállító fájlokat is beolvassa az objektum beállításokhoz (sok más mellett). Mi most csak azokat a fájlokat tárgyaljuk, melyeket a legegyszerűbb beállítások esetén módosítanunk kell. Az alapvető használathoz a beállításokat a /etc/nagios3/conf.d könyvtárban található fájlokban kell megtennünk.

A beállító fájloknak van egy fontos közös jellemzője. Ez pedig az, hogy minden beállításhoz készíthetünk (illetve az alaptelepítés már tartalmaz is) egy-egy mintát (template-et), mely az alapvető beállításokat tartalmazza. Ilyen például a generic-host_nagios2.cfg fájl tartalma, mely egy gép definíciója, melynek neve generic-host. Ha ezek után egy valódi gépet akarunk definiálni a későbbi ellenőrzés céljából, akkor csak át kell vennünk a generic-host tulajdonságait a „use” direktíva segítségével, ahogy a localhost_nagios2.cfg beállító fájlban látszik is. Valahogy így:

```
-- generic-host_nagios2.cfg
define host{
    name                            generic-host    ; The name of template
     notifications_enabled           1 ; Host notifications are enabled
    event_handler_enabled           1 ; Host event handler is enabled
    flap_detection_enabled          1 ; Flap detection is enabled
    failure_prediction_enabled      1 ; Failure prediction is enabled
    process_perf_data               1 ; Process performance data
    retain_status_information       1 ; Retain status across restarts
    retain_nonstatus_information    1 ; Retain non-status info across restarts
    check_command                   check-host-alive
    max_check_attempts              10
    notification_interval           0
    notification_period             24x7
    notification_options            d,u,r
    contact_groups                  admins
    register                        0 ; DONT REGISTER THIS DEFINITION!
    }
```

```
-- localhost_nagios2.cfg
define host{
        use                     generic-host ; Name of host template to use
        host_name               localhost
        alias                   localhost
```

```
        address                 127.0.0.1
        }
...
```

A másodikként idézett konfig „use” helyére oda kell képzelnünk minden olyan beállítást, mely a mintában meg van adva. A mintának mindig van egy „name” értéke, mely megadja a minta nevét, ezt kell majd használni a „use” értékeként.

Ennek óriási előnye, hogy a tipikus beállításokat nem kell újra és újra leírnunk, a konfig jobban átlátható és rövidebb lesz. Szintén nagy előny, de jobban oda is kell figyelnünk, hogy ha a mintában átállítunk valamit, akkor az minden olyan további beállításra hatással lesz, ahol felhasználták.

Felhasználók és csoportok

Az Apache által elfogadott felhasználókról a korábbiakban már volt szó annak beállításainál. Ha nem akarsz több felhasználót létrehozni, több különböző gép vagy szolgáltatás ellenőrzésére, akkor használd a nagiosadmin felhasználót és lépj a következő mezőre. Ha új felhasználókat akarsz létrehozni, akkor tudnod kell, hogy az Apache beállításokban létrehozott felhasználó be tud majd lépni, de amíg nem hozol neki létre egy Nagios rendszeren belüli felhasználót (itt contact), addig nem fog semmilyen gépet és szolgáltatást látni. Az alábbiakban egy egyszerű contact minta és a belőle származtatott felhasználó látható, aki bekerül az admins csoportba. Ha indokolt, akkor a contact-okat több csoportba rendezhetjük, így a megfelelő beállítások mellett mindenki csak a neki érdekes gépek és szolgáltatások állapot változásairól értesül.

```
-- contacts_nagios2.cfg
define contact{
    name                            generic-contact
    host_notification_period        24x7
    service_notification_period     24x7
    host_notification_options       d,r
    service_notification_options    w,u,c,r
    host_notification_commands      notify-host-by-email
    service_notification_commands   notify-service-by-email
}
define contact{
    use                             generic-contact
    contact_name                    atya
    email                           mato.peter@gmail.com
}
define contactgroup{
    contactgroup_name               admins
    alias                           Nagios Administrators
    members                         root,atya
    }
```

Parancsok

A rendszer minden kapcsolatát az operációs rendszerrel parancs objektumokon keresztül definiálhatjuk. Ilyeneken keresztül lehet az ellenőrzéseket végző modulok meghívását specifikálni:

```
define command{
    command_name    check-host-alive
    command_line    /usr/lib/nagios/plugins/check_ping -H '$HOSTADDRESS$' -w
  5000,100% -c 5000,100% -p 1
    }
```

Vagy az értesítések kiküldésének pontos módját megadni:

```
define command{
    command_name    notify-host-by-email
    command_line    /usr/bin/printf "%b" "***** Nagios *****\n\nNotification
  Type: $NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState: $HOSTSTATE$\nAddress:
  $HOSTADDRESS$\nInfo: $HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" |
  /usr/bin/mail -s "** $NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is $HOSTSTATE$
  **" $CONTACTEMAIL$
}
```

Értesítések, időintervallumok

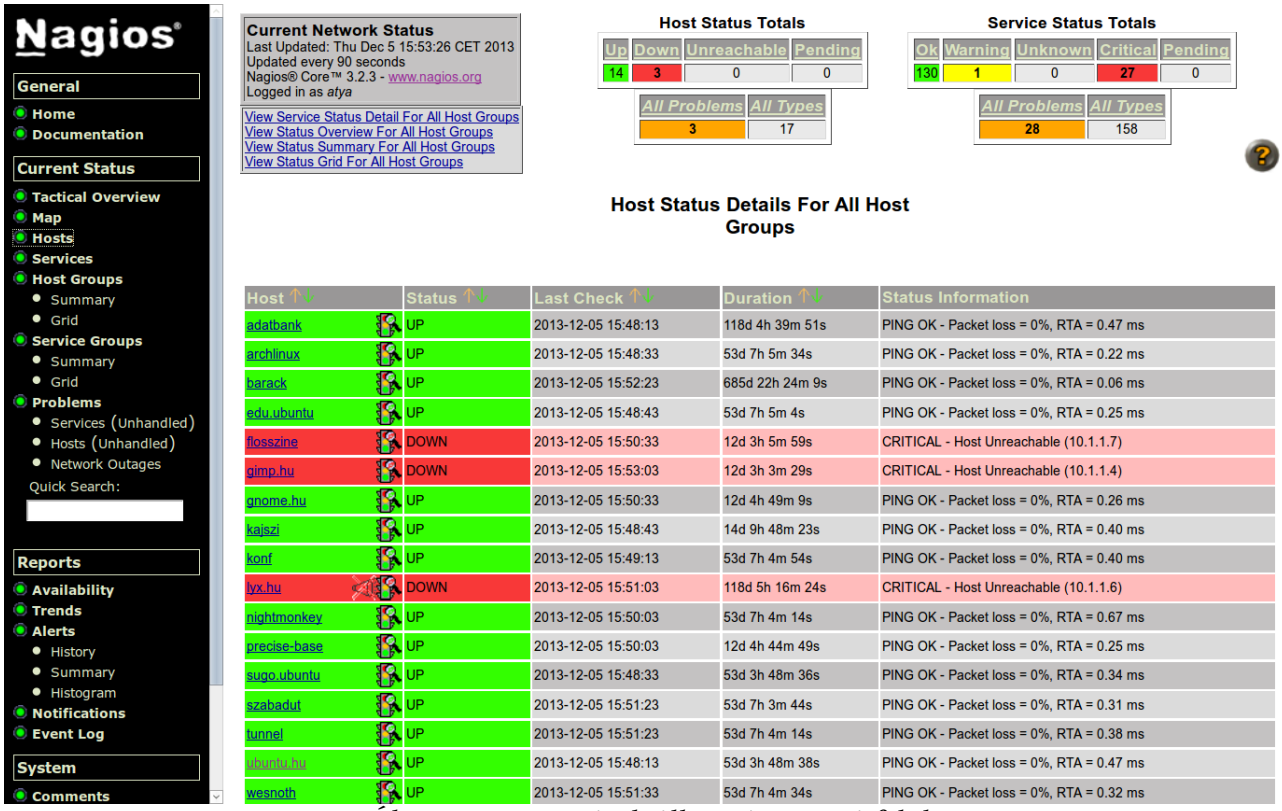
A fenti contact minta tartalmazott két _period végű beállítást. Ezek határozzák meg, hogy melyik intervallumban kell az adott felhasználót kiértesíteni gépekkel vagy szolgáltatásokkal kapcsolatban. Ezen „24x7” beállítás a timeperiods_nagios2.cfg fájlban van definiálva, melynek használata értelemszerű. Ha a felügyeletnek csak munkaidőben kell reagálni, akkor átállítható például „workhours” értékre, melynek pontos adatait szintén a fenti fájlban lehet megtekinteni alaptelepítés után. Egyébként ugyanezekkel az időintervallum beállításokkal lehet majd megadni, hogy egy-egy gép vagy szolgáltatás mikor ellenőrzendő.

A contact-nál láthatóan megadható egy-egy gép (host) és szolgáltatás (service) értesítési parancs. Ezek akkor kerülnek meghívásra, ha a gép vagy szolgáltatás állapotában valamilyen változás áll be. Ezek módosítására általában nincs szükség. Ha azonban a kiküldött értesítő e-mail egy e-mail – SMS csatolón keresztül telefonra is továbbításra kerül (a legtöbb mobil szolgáltatónál elérhető ilyen szolgáltatás), akkor szerencsésebb a parancs paramétereiből csak a leginkább fontosakat megőrizni, hogy egy-egy értesítés ne legyen több SMS hosszú. Ilyen lehet a módosított parancs definíció:

```
define command{
    command_name    notify-host-by-email
    command_line    /usr/bin/printf "%b" "Date: $LONGDATETIME$" | /usr/bin/mail
  -s "Nagios: $HOSTNAME$ is $HOSTSTATE$" $CONTACTEMAIL$
}
```

Gépek és csoportok

Az ellenőrzendő gépek definícióit a generic-host_nagios2.cfg-ben található generic-host mintából érdemes elkészíteni. A gépek ellenőrzését a rendszer a check_command által meghatározott paranccsal végzi el. Ez alapértelmezetten a check-host-alive, mely egy ping kérés. Ha a hálózatban az ICMP echo-request nem tud átmenni a tűzfalon, akkor ezt a gép ellenőrző parancsot át lehet állítani például check_ssh-ra. Az ellenőrzés kikapcsolásra kerül, ha a check_command értéke üres.



3. Ábra: A Nagios gépek állapotát mutató felülete

A gépek lehetséges állapotai:

U, Up	működik
D, Down	nem működik
U, Unreachable	nem elérhető

Így néz ki az alapértelmezett gép beállítási minta:

```
-- generic-host_nagios2.cfg
define host{
    name                            generic-host ; The name of template
    notifications_enabled           1 ; Host notifications are enabled
    event_handler_enabled           1 ; Host event handler is enabled
    flap_detection_enabled          1 ; Flap detection is enabled
    failure_prediction_enabled      1 ; Failure prediction is enabled
    process_perf_data               1 ; Process performance data
    retain_status_information       1 ; Retain status across restarts
    retain_nonstatus_information    1 ; Retain non-status info across restarts
    check_command                   check-host-alive
    max_check_attempts              10
    notification_interval           0
    notification_period             24x7
    notification_options            d,u,r
    contact_groups                  admins
     register                        0 ; DONT REGISTER THIS DEFINITION!
}
```

Ez alapvetően jól is van, csak a következő táblázatot nézzük át a beállítások testre szabására. A gépeknek nagyon sok paraméterét lehet beállítani, amiknek a módosítására leggyakrabban szükség van:

host_name	A gép neve, ez fog megjelenni a felületen és az értesítésekben
address	A gép IP címe, ez szinte minden ellenőrző parancs definíció első paramétere lesz
hostgroups	A gép (vagy minta) gép-csoportjainak neve
check_interval	Ellenőrzések időköze időegységben (a nagios.cfg-ben megadott interval_length adja meg az időegységet, alapértéke 60 másodperc)
max_check_attempts	Újra ellenőrzések száma hibajelzés előtt
retry_interval	Újra ellenőrzések időköze ha a gép nem Up állapotú
check_period	Az ellenőrzések intervalluma, részletesebben szó volt róla az értesítéseknél
contacts	A gépet felügyelő felhasználók vesszővel elválasztott listája
contact_groups	A gépet felügyelő csoportok vesszővel elválasztott listája
notification_interval	Értesítések időköze időegységben. Alapértéke 0, mely azt jelenti, hogy nincs újra értesítés. Ez a beállítás nem javasolt. Jobb választás az 1440, mely egy nap.
notification_period	Értesítések idő intervalluma
notifications_enabled	Az értesítések engedélyezettek-e. Ha átmenetileg ki akarjuk kapcsolni az értesítéseket, akkor 0-át kell beállítani.

A gépeket csoportokba lehet rendezni, így a gépek hasonló jellemzői alapján lehet egy helyen megadni például a hozzájuk tartozó felhasználói csoportokat (hostgroup_members) és később a rajtuk futó szolgáltatásokat egyetlen Nagios konfig beállítással ellenőrizni, valahogy így:

```
define service{
    hostgroup_name  HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    service_description   valamiszolgáltatás
    egyéb szolgáltatás beállítások ...
}
```

Szolgáltatások ellenőrzése

A szolgáltatások a korábban megadott gépeken elérhető ellenőrzendő szolgáltatások, melyek lehetnek valamely helyben ellenőrizhető jellemzők (például memória foglaltság, egy folyamat futása, a diszkek állapota stb.), állapot vagy történet információk (van-e belépett felhasználó, megfelelően működnek-e a RAID tömbök, volt-e újrabootolva a szerver az utolsó kézi ellenőrzés óta stb.) valamint hálózati szolgáltatások életképessége és egyéb jellemzői. Az alaptelepítés után így néz ki egy szolgáltatás minta:

```
-- generic-service_nagios2.cfg
define service{
    name                            generic-service ; The name of template
    active_checks_enabled           1 ; Active service checks are enabled
```

```
    passive_checks_enabled          1 ; Passive checks are enabled/accepted
     parallelize_check               1 ; Active checks should be parallelized
    obsess_over_service             1 ; We should obsess over this service
    check_freshness                 0 ; NOT check service 'freshness'
    notifications_enabled           1 ; Service notifications are enabled
    event_handler_enabled           1 ; Service event handler is enabled
    flap_detection_enabled          1 ; Flap detection is enabled
    failure_prediction_enabled      1 ; Failure prediction is enabled
    process_perf_data               1 ; Process performance data
    retain_status_information       1 ; Retain status info across restarts
    retain_nonstatus_information    1 ; Retain non-status info across restarts
    notification_interval           0 ; Only send notifications on status change
    is_volatile                     0
    check_period                    24x7
    normal_check_interval           5
    retry_check_interval            1
    max_check_attempts              4
    notification_period             24x7
    notification_options            w,u,c,r
    contact_groups                  admins
    register                        0 ; DONT REGISTER THIS DEFINITION!
}
```

A szolgáltatások leggyakrabban állítandó paraméterei:

service_description	A szolgáltatás leírása, ez fog megjelenni a felületen és az értesítésekben
host_name	A szolgáltatást tartalmazó ellenőrzendő gépek nevei vesszővel elválasztva
hostgroup_name	Az ellenőrzendő gépcsoportok nevei vesszővel elválasztva
check_command	Itt adjuk meg, hogy mely parancs segítségével lehet ellenőrizni az adott szolgáltatást. Példákat lásd lent!
active_checks_enabled	Ezzel átmenetileg tiltható az ellenőrzés. 0 - tiltva, 1 - engedélyezve
check_interval	Mint a gépeknél
max_check_attempts	Mint a gépeknél
retry_interval	Mint a gépeknél
check_period	Mint a gépeknél
contacts	Mint a gépeknél
contact_groups	Mint a gépeknél
notification_interval	Mint a gépeknél
notification_period	Mint a gépeknél
notifications_enabled	Mint a gépeknél

A szolgáltatások lehetséges állapotai:

O, OK	rendben
W, WARNING	veszély

C, CRITICAL	kritikus
U, UNKNOWN	ismeretlen

Ha egy szolgáltatás állapota megváltozik, akkor a beállított felhasználók vagy csoportok értesítést kapnak és a felületen az adott szolgáltatás piros színnel jelenik meg.

Távol ellenőrizhető szolgáltatások

Ahogy arról már korábban is szó volt, a távoli ellenőrzések elvégzése az NRPE kiegészítő modul segítségével lehetséges. Működése a következő. A távoli, vizsgálandó szerveren telepíteni kell az NRPE szerver komponensét (nagios-nrpe-server csomag), valamint a szükséges Nagios ellenőrző modulokat (nagios-plugins csomag és barátai). Ekkor a vizsgálandó gépen alapértelmezetten az 5666-os porton figyelni kezd az NRPE szerver. A felügyeleti szerveren van egy NRPE ellenőrző kliens modul (check_nrpe), mely a vizsgálandó szerver megadott portjára kapcsolódik, a kommunikáció SSL titkosítással zajlik, és a vizsgálandó szerveren beállított (szabadon konfigurálható) vizsgáló parancsok valamelyikét lefuttatja. Erre az NRPE szerver (mely a vizsgálandó gépen fut) lefuttatja az adott parancshoz tartozó, előre beállított Nagios ellenőrző modult a konfigjában előre megadott paraméterekkel, majd az eredményt hálózaton visszaadja az NRPE ellenőrző modulnak. Tehát a felügyeleti szerver _nem tud_ tetszőleges parancsot végrehajtani a vizsgált szerveren, de még a paraméterezést sem tudja megváltoztatni. Kizárólag az előre beállított ellenőrzések valamelyikét kérheti.

A vizsgált gépen futó NRPE szerver konfigja így néz ki:

```
--- /etc/nagios/nrpe.cfg
pid_file=/var/run/nrpe.pid
server_port=5666
nrpe_user=nagios
nrpe_group=nagios
# localhost es a felugyeleti szerver
allowed_hosts=127.0.0.1,192.169.1.42
# Tilos a parameterek atvetele es feldolgozasa! Biztonsagi okbol
#    SOHA nem szabad engedelyezni!!!11
dont_blame_nrpe=0
debug=0
command_timeout=60
connection_timeout=300
command[check_apt]=/usr/lib/nagios/plugins/check_apt
command[check_users]=/usr/lib/nagios/plugins/check_users -w 1 -c 4
command[check_load]=/usr/lib/nagios/plugins/check_load -w 15,10,5 -c 30,25,20
command[check_disks]=/usr/lib/nagios/plugins/check_disk -w 20 -c 10 -e
command[check_zombie_procs]=/usr/lib/nagios/plugins/check_procs -w 5 -c 10 -s Z
command[check_total_procs]=/usr/lib/nagios/plugins/check_procs -w 150 -c 200
include=/etc/nagios/nrpe_local.cfg
```

Mentések és archiválás

Az talán senki számára nem vitatható tény, hogy adatmentésnek, azaz biztonsági mentésnek minden rendszer alatt lennie kell. Az is fontos, hogy a mentés automatikusan, beavatkozásmentesen történjen, ugyanakkor szintén fontos, hogy a mentések rendszeresen ellenőrizve is legyenek, hiszen ha nyugodtan bízunk egy mentést végző 5-6 éves héjprogramban (shell szkript), amit sosem ellenőriztünk le, akkor kellemetlen meglepetések érhetnek minket, amikor elő kell venni a régi, megbízhatónak gondolt mentést, és kiderül, hogy egy adatbázist vagy egy könyvtárat mégsem követtünk le. A Linux-alapú rendszerekre számos kiváló mentési keretrendszer és héjprogram érhető el. Gyakorlatilag a legtöbb a unixos alapokkal rendelkező cp, tar[85], cpio[86] programokat használja, amelyek minden Linux rendszer standard tartozékai. Mások az rsync[87]-et és az SSH-t kombinálják, ezzel teszik lehetővé a biztonságos távoli menthetőséget is.

A főbb mentési módszerek

– *Teljes mentés:* ilyenkor a rendszer minden adata mentésre kerül, válogatás nélkül. Előnyre, hogy például egy lemezkép készíthető a szerver aktuális állapotáról, amely egyszerűen visszaállítható (gyakran csak a rendszertöltőt (boot loader) vagy még azt sem kell helyrerakni). Hátránya: sérülékeny és időigényes.
– *Inkrementális mentés:* Használata során csak a változások mentődnek, azaz leképezünk egy nullás mentést és a megváltozott állapotokat mentjük. Ennek a mentésnek 2 alfajtája van: a kumulatív és a differenciális mentés.
– *Kumulatív mentés:* alkalmazása esetén mindig az utolsó teljes mentés óta megváltozott adatok kerülnek mentésre. Adatváltozás esetén csak a változást menti, viszont azt minden esetben. Előnye, hogy jóval gyorsabb a teljes mentésnél, és kevesebb a területigénye is.
– *Differenciális mentés:* ebben az esetben csak az utolsó inkrementális mentés óta megváltozott adatokat mentjük. Előnye, hogy a leggyorsabb stratégia. Hátránya, hogy az első mentés és az összes követő mentés szükséges hozzá, azaz a leginkább sérülékeny az adatstruktúrát illetően.

A mentési eljárások két irányból hajthatók végre. A szerver végezhet adat „tolást” a mentő egység (szerver, NAS, szalag) felé, illetve a mentő egység/szerver is kezdeményezheti a mentési folyamatot, ilyenkor maga fele húzza az adatokat. A két módszer között főként biztonsági különbség van. Ha a backup szerver húzza maga felé az adatot, akkor azt a szervert kell úgy kialakítani, hogy biztonságos legyen. Ilyen esetben az éles szerverre való betörés esetén a backup szerver kevésbé sérülékeny, mint fordítva.

85. http://www.gnu.org/software/tar/
86. http://www.gnu.org/software/cpio/
87. http://en.wikipedia.org/wiki/Rsync

A legfontosabb teendők a mentés kialakításában

– *Felmérés:* összeírni a mentendő szoftverkörnyezet kialakítását. Eldönteni, hogy az adott mentő szoftver alkalmas-e valós időben menteni az adatbázisunk (szükséges-e ez egyáltalán), illetve pontosan mit és mikor célszerű mentenünk. Például: érdemes-e minden nap menteni a teljes fájlrendszert, mindenféle ideiglenes állománnyal együtt, vagy elég csak aktualizálni. Fontos gondolni a Disaster recovery-re[88] (katasztrófa-helyreállítás) is, amikor akár az egész gépterem elpusztulhat. Ismert ilyen példa, amikor a WTC[89] egyik épületében volt az éles alkalmazás szerver és a 2. toronyban pedig a backup. Természetesen ez nem minden nap előforduló lehetőség, de számításba kell venni a felmérés folyamatában. Fel kell tenni magunknak a kérdést, hogy az adataim hiánya, mekkora kárt okoz, illetve mekkora összegbe kerül egy RAID tömb visszaállítása az erre szakosodott szervizben.

– *Tervezés*: a felmérésre alapozva vázlatosan leírjuk a támasztott igényeinket. Betervezzük, milyen gyakran, milyen idősávban lesz lehetőség futtatni a mentést. Szétválasztjuk a távoli, a szalagos és a helyi mentéseket. Ütemezzük azokat, hogy ne fussanak egymásra. A mentési házirend figyelembe vétele, ha már van ilyen, ha nincs akkor írjunk.

– *Kiválasztás*: Az 1-2 pontok alapján kiválasztjuk a megfelelő szoftvert, amely a leírása alapján alkalmas lesz a munkára.

– *Megvalósítás*: A kiválasztott szoftverek segítségével felépítjük a mentési rendszert.

– *Tesztfuttatás:* Leteszteljük a már elkészült mentést. Megnézzük a lehetséges eseteket és azok befolyását a mentésünkre. Elég szalag áll-e rendelkezésre, működik-e az értesítési rendszer, amely tudatja velünk, ha elfogyott a szalag, vagy nincs elég tárterület. Egy híres mentési baki, amikor az egyik ismert banki kártyarendszer teljes hálózata azért állt meg karácsony előtt, mivel az automata leejtette a cserélendő szalagot és a mentés nagyobb prioritást élvezett, mint maga az éles üzem. Gyakran hiába állítjuk össze a legjobb mentő eszközt, és állítjuk be jól, ha egy hibásan beállított e-mail cím, vagy egy blokkolt SMTP szolgáltatás lehetetlenné teszi, hogy az értesítés eljusson hozzánk, akkor hiába volt minden. Figyeljünk arra, hogy a mentés lehetőleg olyankor fusson amikor szükséges, de a mentés ne zavarja az éles üzemeltetést.

– *Ellenőrzés*: A mentés elkészültével fontos, hogy egy próba rendszeren végezzünk egy teljes visszaállítást, és nézzük meg, 100%-ban visszaállítható-e minden csak a backupra támaszkodva.

– *Ellenőrzés* ütemezése: legyen a havi/negyedéves/féléves/éves rutin része, hogy teljes rendszer-visszaállítást végzünk, csakúgy mint ahogy akár naponta-hetente megnézzük, hogy a mentés rendben ment-e. Érdemes a mentés által készített összegző naplót monitorozni.

88. http://en.wikipedia.org/wiki/Disaster_recovery
89. http://hu.wikipedia.org/wiki/Vil%C3%A1gkereskedelmi_K%C3%B6zpont

Szoftverek a mentés megvalósítására

DD[90]

Az egyik legrégebbi Unix eszköz például a teljes lemezduplikációra. Felhasználási területe szélesebb a mentéseknél, azonban egy egyszerű klónozás esetében megoldást jelenthet, fenntartásokkal.

Tar, CP, CPIO

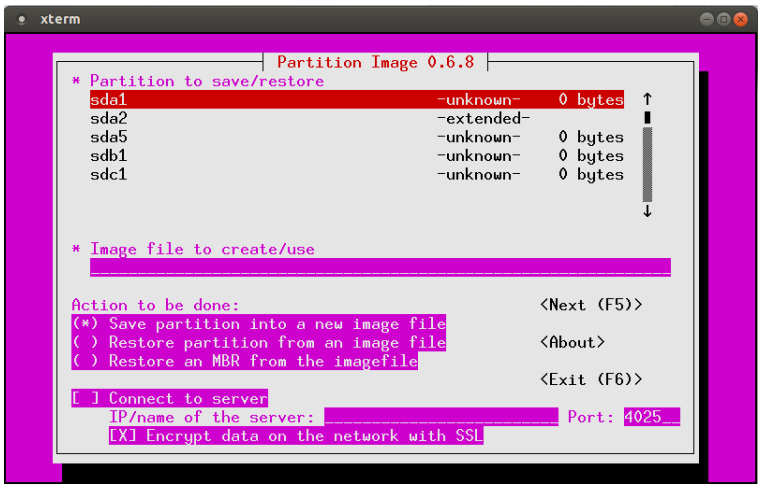
Az operációs rendszerek alapvető kelléktára. Héjprogramok segítségével használható mentésre, akár kazettás egység vezérlésére is. Önmagukban nagyon hasznosak, de egy modern rendszerben a rájuk épülő programokat használjuk inkább.

Partimage[91]

Egy konzol alapú lemezklónozó eszköz. A dd-hez nagyon hasonló, azonban képességei alapján nem mondható egy sima dd-nek. Kezelése egyszerű, egy párbeszédablakos felületen lehet kiválasztani a mentendő területet és azt, hogy hova mentsen. Tud titkosítani és darabolni is, valamint hálózati mentést csinálni. Hátránya, hogy csak offline rendszert lehet vele menteni, célszerűen egy live CD-vel. Szerver lemezének költöztetése esetén, vagy hardvercsere esetében jó megoldás lehet, illetve munkaállomások klónozásra is. Rendelkezik állapotfigyelővel, és számos lehetőséget nyújt a tömörítésre, darabolásra is. Ugyanakkor alap szintű lemez duplázásra remek megoldás, azaz ha elkészítettünk egy szervert (például egy tűzfalat, amelyen jó esély van rá, hogy hosszú távon főként az iptables konfigurációs állományának a változását kell lekövetni, vagy például egy telefonközpont – Asterisk) és mindenképpen szükséges egy hideg backup belőle, akkor egy tökéletes eszköz a partimage. Akár egy Ubuntu Live pendrive vagy CD-vel beindítjuk a gépet, úgy hogy a másolandó (forrás) és a cél merevlemez is benne van a gépben, majd a Partimage párbeszédablakában kiválasztjuk, mit szeretnénk csinálni, és készen is vagyunk. Futó (éles rendszert) értelemszerűen így nem tudunk menteni, de akár arra is használható, ha kliens parkot kell létrehoznunk, azaz adott 200 db ugyanolyan PC, és mindegyikre fel kell telepíteni ugyanazt a terjesztést. Telepítése rendkívül egyszerű:

```
apt-get install partimage
```

90. http://en.wikipedia.org/wiki/Dd_(Unix)
91. http://www.partimage.org/Main_Page



Rsync[92]

Az rsync ideális eszköz gépen belüli szinkronizációra, természetesen lehetőség van távoli elérésre is, akár ssh-n, akár rsyncd-n keresztül. A legtöbb backup megoldás alapeszközként használja, mások kiegészítőként, illetve pár megoldás kifejezetten az rsyncre alapozva igazából csak egy kezelőfelület. Paraméterezhetősége nagyon széles körű, talán pont ezért készült annyi eszköz, amely leegyszerűsíti a vezérlését (például: grsync). Előnye a többi hozzá képest sokkal egyszerűbb (cp, tar stb.) programmal szemben, hogy a számtalan lehetőség miatt rugalmasan kezeli például, ha a forrás nagyobb mint a cél lemez területe, akkor nem írja tele a célt stb. Általában héjprogram formájában vagy beépített eszközként használjuk. Az rsync segítségével nem csak gyorsabban tudunk menteni, hanem a speciális delta kódolásnak köszönhetően csak a változásokat fogja átvinni egyik helyről a másikra. Ennek különösen szűk hálózati keresztmetszetek között van nagy jelentősége, vagy pedig például, ha egy 1 GB-os lemezképben csak 2 MB tartalom változik meg, akkor az rsync delta technológia segítségével nem a teljes 1 GB-ot fogja lementeni, hanem egy összehasonlító vizsgálat után csak a változást – azaz jelen esetben a 2 MB-ot – fogja aktualizálni. A legtöbb terjesztés tartalmazza alap szerver kiépítésben, de telepíthetjük az

```
apt-get install rsync
```

parancs kiadásával. A legtipikusabban használt paraméterezése a következő:

```
rsync -av /forrás/ /cél/
```

ahol a -a az archív készítése, a -v pedig a részletes nézet. Ezt aztán kiegészíthetjük számtalan hasznos paraméterrel is, pl:

```
rsync -r -t -p -o -g -v --progress --delete -l -H /cél/ /forrás/
```

Az Rsync lokális használta esetén előfordulhat, hogy például egy igen gyors SSD lemezről másolunk egy USB1 vagy USB2 szabványon csatlakozó akár Flash memóriára, vagy merevlemezre. Ez persze főként otthoni használatban fordulhat elő, de szerveren belül is megeshet, hogy a különböző lemez alrendszerek más-más sebességgel működnek. Ilyen esetben és főként a tömegesen előforduló kis állományok szinkronizálása esetén előfordulhat, hogy feltorlódnak az I/O műveltek, és ezért emelkedni kezd a rendszerterhelés (load), fogyni kezd a memória, szélsőséges esetben már tárcserét (swap) kezd a rendszer. Ez pedig biztos út az összeomlás felé. Az ilyen összeomlások el-

92. http://rsync.samba.org/

kerülésére két eszköz használata javasolható, az egyik egy monitorozó eszköz, amely a top programhoz hasonlóan képes megjeleníteni és rendezni azokat a futó folyamatokat, amelyek az I/O terhelést okozzák. A program neve iotop,[93] telepítése és használata is igen egyszerű:

```
apt-get install iotop
```

és akár csak a top program esetén, csak indítsuk el. A másik segédeszköz, amely hasznos lehet egy esetlegesen rsync mentés kivitelezéséhez, az ionice[94]. Telepítése:

```
apt-get install ionice
```

Az ionice és iotop programok részletes ismertetésével a tuningolás fejezet foglalkozik.

Rsnapshot[95]

Szintén rsync és SSH alapokon nyugvó mentési eszköz. Jóval fejlettebb lehetőségeket tartogat, mint az eddigi eszközök. Alkalmas teljes fájlrendszer mentésére megszorításokkal (boot loadert nem tud menteni, illetve futási időben például adatbázist sem). Egyszerű paraméterezhetőség és könnyen átlátható konfiguráció jellemzi. Rugalmas eszköz, mivel képes lokális mentést végezni (alapesetben a differenciális mentést támogatja), az indulása cronból ütemezhető, vagy szkriptelhető. Létrehoz egy zároló állományt, amely jelzi az esetlegesen ráfutó következő rsnapshotnak, hogy még az előző mentés fut, ezzel elkerülve az esetleges feltorlódást és ütközést. Együtt tud működni az SSH-val is, így lehetőség van arra, hogy egy távoli backup szerver (amelynek csak ez a feladata és kellően biztosított) kezdeményezze a mentést SSH-val. De természetesen megoldható valamilyen hálózati fájlrendszeren keresztüli mentés, illetve külső illesztésű merevlemezre is. Nagyon hasznos funkciója, hogy a konfigurációs állomány az Apache-hoz hasonló szintaxis-ellenőrzéssel rendelkezik, vagyis az rsnapshot configtest parancs segítségével tesztelhetjük az esetlegesen a szintaxissal ütköző parancsokat, elütéseket. Így nem az éles mentés lefutása közben fog kiderülni a probléma. Külön naplóállományt készít, amelyben jól követhető a tevékenysége. A mentést elkülönítetten egy erre meghatározott területen kezeli, ahol a megadott intervallumtól függően készíti el a könyvtárstruktúrát. Az első alap mentés elkészítése jóval lassabb, a további inkrementális mentések már gyorsabbak, melyeket linkek segítségével köt össze. Az aktuális állapotok pedig hourly, daily, weekly, monthly címkével kerülnek tárolásra. Természetesen egy konfigurációs állományban felsorolhatjuk az óránkénti, napi, heti, havi mentések struktúráját is. Kivételkezelése könnyen paraméterezhető (az rsync használatának köszönhetően). Külön csak erre a feladatra fenntartott mentő szerver esetén (megfelelően védett backup szerver) jól automatizálható a mentés teljes folyamata az SSH kulcs alapú megoldásával. Ebben az esetben a backup szerver kezdeményezi a kapcsolatot SSH-n keresztül. Annak érdekében, hogy mindent le tudjon menteni, root hozzáférés szükséges (a legtöbb esetben), amely távoli elérésre normál esetben nem lenne biztonságos. Ezért az SSH szerver következő konfigurációváltoztatása javasolt: használjuk a

```
PermitRootLogin=forced-commands-only
```

opciót a root távoli hozzáféréséhez. Majd készítsük el a root SSH-kulcspárját jelszó nélkül, hogy a héjprogram ne kérjen majd jelszót. Válasszuk megfelelően nagy (4096 bájt vagy e feletti) kulcs készítését. Az éles mentendő szerveren a /root/.ssh/authorized_keys állományba másoljuk be a publikus kulcsot a következő kiegészítéssel:

```
from="172.20.1.10",command="/etc/validate-rsync"
```

93. http://guichaz.free.fr/iotop/
94. http://manpages.ubuntu.com/manpages/precise/man1/ionice.1.html
95. http://www.rsnapshot.org/

ezt közvetlenül a legelső karakter helyére illesszük be, utána kezdődjön a kulcs. Az IP-cím helyén a backup szerver IP-címe legyen. Az /etc/validate-rsync héjprogramba pedig rakjuk be azokat a korlátozásokat, amelyek segítségével minimalizálni tudjuk a root SSH használatát. Pontos lista is elérhető az egész mentési módszer részletes leírásával együtt.[96] Mint látható, az rsnapshot egy komplex héjprogram-keretrendszer, amely egyszerűvé teszi a strukturált mentést. Azonban teljes visszaállítás csak az alap operációs rendszer újratelepítése után lehetséges egy esetleges teljes adatvesztés után. Ideális eszköz tehát nem HA rendszerek mentéséhez. Valamint gondoskodni kell az adatbázis külön mentéséről, hiszen futási időben erre az rsnapshot konzisztensen nem képes.

Esetünkben az /etc/validate-rsync állomány tartalma a következő volt:

```
#!/bin/sh
  case "$SSH_ORIGINAL_COMMAND" in
  *\&*)
  echo "Rejected"
  ;;
  *\(*)
  echo "Rejected"
  ;;
  *\{*)
  echo "Rejected"
  ;;
  *\;*)
  echo "Rejected"
  ;;
  *\<*)
  echo "Rejected"
  ;;
  *\`*)
  echo "Rejected"
  ;;
  *\|*)
  echo "Rejected"
  ;;
  rsync\ --server*)
  $SSH_ORIGINAL_COMMAND
  ;;
  *)
  echo "Rejected"
  ;;
  esac
```

Amanda Backup[97]

Kliens-szerver felépítésű mentőeszköz, képes Windows rendszereket is menteni. Windows esetén a Cygwin csomag, vagy Samba megosztás vagy pedig saját kliens segítségével végezhető el a mentés. Teljesen szerver alapú működés, előre definiálható adattárolók, amelyek lehetnek lemezről lemezre, lemezről kazettára, illetve lemezről felhő alapú tárterületre való mentések is. Eredetileg kezdetben kazettás mentések kialakítására tervezték, ezért amikor merevlemez alapú mentést

96. http://troy.jdmz.net/rsync/#ref2
97. http://www.amanda.org/

definiálunk, akkor virtuális szalagokkal operálva a merevlemezen hoz létre alkönyvtárakat. A virtuális szalagok definiált méretét és darabszámát nem lépi át így sem. Igazi előnye, hogy nem kell statikus tervet készíteni, meg kell neki adni, hogy hány darab teljes mentést illetve mennyi inkrementálist fogunk készíteni. Rögzítjük a felhasználható szalagok számát. Az Amanda mentés elején kiszámolja, hogy a mentendő könyvtárak vagy inkrementális mentéseihez aktuálisan mennyi helyre van szükség és önállóan eldönti, hogy az adott mentésbe milyen könyvtár milyen szinten kerüljön bele. Ha rendelkezésre áll hely, akkor előre hozza a teljes mentéseket, ha pedig kifut a helyből, akkor késlelteti és a naplóba figyelmeztetéseket ír. Az Amanda telepítésének fontos része az openbsd-inetd csomag is. Mivel titkosítást alapesetben nem használ a távoli gépre való mentés esetén, ezért érdemes VPN-t vagy a külön konfigurálható amcrypt-et[98] használni a nyílt hálózatokon. Belső gépek mentésére titkosítás nélkül is jól használható. Mivel az Amanda több TCP és UDP portot használ és nyit, ezért érdemes a tűzfalunkat felkészíteni a fogadására mind a két gépen, amennyiben a kialakításunk ezt szükségessé teszi.[99]

Bacula[100]

Szintén keresztplatformos biztonsági mentést lehet megvalósítani vele. Létezik hozzá Linux, Windows és OS X alá kliens program, amelyeknek konzol alapú, GTK+ felületű és wxWidgets grafikus variánsai léteznek. Legfőbb erőssége, hogy elosztott működésre képes, azaz különböző részekből áll, amelyeket külön gépekre telepíthetünk. Fő komponensei:

– Director daemon (director): végzi a rendszer irányítását, kapcsolatot tart a többi démonnal, kommunikál a különböző grafikus felületekkel, időzíti a mentéseket, aktualizálja a katalógust.
– Storage daemon (sd): közvetlenül a mentő egységekkel (szalagos meghajtó, HDD, DVD) tartja a kapcsolatot, kezeli a director olvasási/írási kéréseit, illetve fogadja az FD-től (lásd alább) érkező mentett adatokat.
– File Daemon (FD): feladata a mentendő szerverek és asztali gépek mentendő adatainak begyűjtése, tömörítése, titkosítása, majd az adatok továbbítása a kijelölt SD-nek. Valamint a visszaállításnál mindezt visszafelé irányban is ellátja.

A Bacula egy központi katalógusban tárolja a mentett állományokat és a fájlt tartalmazó kötet nevét, a mentés időpontját, valamint egyéb adatokat is, segítve és gyorsítva ezzel az esetleges visszaállítást. Nagy előnye, hogy szinte mindenre kiterjedően konfigurálható, így megoldható az is, hogy a mentés indulása egy tetszőleges egyedi héjprogram futtatásával kezdődjön, például egy adatbázismentést (MariaDB, MySQL, PostgreSQL dump) kezdeményezve. Egy igazi komplex megoldás, amely a titkosítása és az elosztott működése miatt a nagyvállalati felhasználásra is alkalmassá teszi[101].

Dirvish[102]

Szintén szerver alapú mentőeszköz, amely támaszkodik az rsync és a Perl képességeire. Tudása nagyban hasonló az Amanda, vagy a Bacula lehetőségeihez, konfigurálása népszerűsége és a fel-

98. http://wiki.zmanda.com/index.php/How_To:Set_up_data_encryption
99. http://wiki.zmanda.com/index.php/How_To:Set_Up_iptables_for_Amanda
100. http://www.bacula.org/en/
101. http://sugo.ubuntu.hu/10.10/html/serverguide/hu/bacula.html.
102. http://www.dirvish.org/

lelhető dokumentáció miatt talán könnyebb[103]. Ugyanúgy kezeli a legtöbb nagyvállalati szalagos eszközt és lehetőség van egyéb például merevlemez mentésére is. Az adatbázis mentéseket egy mentés indítását megelőző Perl vagy héjprogrammal lehet megoldani akár Oracle, PostgreSQL, MySQL vagy egyéb adatbázis-kezelő esetén is. A kliens-szerver alapú mentés természetesen futhat SSH megoldással[104], így biztosítva a távoli gépek között az adatbiztonságot mentés esetén. Telepítését a megszokott módon az apt-get segítségével kezdjük meg:

```
apt-get install dirvish ssh
```

A telepítés menete a Pull vagy Push (utolsó bekezdések) alapján általunk követendőnek tekintett felépítést elemzi, azaz a központilag védett és elkülönített backup szerver tud csatlakozni az összes klienshez, de a kliensek nem tudnak csatlakozni közvetlenül hozzá.

Az alap konfigurációs állomány mintákat a /usr/share/doc/dirvish/examples könyvtárban találjuk. Célszerűen innen elsőnek a master.conf-ot másoljuk át a /etc/dirvish/master.conf helyre. Akkor valami ehhez hasonlót fogunk látni:

```
bank:
/backup
  exclude:
/etc/mtab
        /var/lib/nfs/*tab
        /var/cache/apt/archives/*.deb
        .cache/*
        .firefox/default/*/Cache/*
        /usr/src/**/*.o
        lost+found/
  Runall:
  eles1.szerver    21:00
eles2.szerver    23:00

  expire-default: +15 days
  expire-rule:
  #       MIN HR    DOM MON       DOW  STRFTIME_FMT
  *   *     *   *         1    +3 months
  #       *   *     1-7 *         1    +1 year
  #       *   *     1-7 1,4,7,10  1
  *   10-20 *   *         *    +4 days
  #       *   *     *   *         2-7  +15 days
```

Nézzük meg pontosan, mi mire való:

A master.conf-ban azokat a változókat fogjuk beállítani, amelyek az összes mentendő gépre vonatkozni fognak, valamint itt definiáljuk azokat a gépeket is, amelyeket menteni akarunk. Értelemszerűen vagy hozzuk létre a /backup könyvtárat 700-as jogokkal root:root felhasználóként, vagy írjuk át arra a könyvtárra, ahova menteni akarunk. A bank változóval határozhatjuk meg, hogy a program hol tárolja a tényleges mentéseket.

Az exclude paraméterrel azokat a könyvtárakat határozhatjuk meg, amelyeket nem akarunk menteni, azaz ki akarjuk zárni őket a teljes mentésből. Mint az látható, megadhatjuk abszolút és relatív módon is. Így vonatkozhat az összes .firefox/ alatt lévő Cache könyvtárra is, de explicit ki-

103. http://www.googlux.com/dirvishconfig.html
104. http://apt-get.dk/howto/backup/

zárhatjuk a /lost+found/-ot is. Ezek a paraméterek központilag fognak vonatkozni az összes később definiált hostra, amelyet menteni fogunk, így ott már nem kell külön ezeket felsorolni.

A Runall paraméterrel azt mondjuk meg a programnak, hogy az egyes gépek mentését mikor futtassa, ide tudjuk felsorolásszerűen befűzni a gépeinket, amelyeket a bank könyvtár alá létre is kell hozni. Azaz, létre kell hozni a /backup/eles1.szerver és a /backup/eles2.szerver könyvtárakat is.

Az expire-default és expire-rule paraméterekkel az elévülési időt határozhatjuk meg, azaz az egyes mentéseket meddig tartsa meg, és mikortól kezdje felülírni.

Hogy tovább tudjunk lépni, a /backup/eles1.szerver alá létre kell hoznunk egy dirvish könyvtárat és abban elhelyezni egy default.conf fájlt. Az elérési útja így nézzen ki:
/backup/eles1.szerver/dirvish/default.conf

a tartalma pedig a példa kedvéért legyen a következő:

```
client: eles1.szerver.kft
  tree: /
  index: gzip
  image-default: %Y-%m-%d
  xdev: 1
  exclude:
  /var/spool/squid/
```

Értelemszerűen a client paraméternél megadjuk a mentendő éles szerver hostnevét vagy IP-címét, ahova a dirvish root felhasználóval fog csatlakozni. A tree paraméter pedig megmondja, hogy honnan indítjuk a mentés. Jelen pillanatban az egész fájlrendszert szeretnénk menteni, így a / került oda. Az image-default paraméterrel meghatározzuk a bank megfelelő alkönyvtárában, milyen struktúrát építsen, így az aktuális dátum lesz az IMAGE neve, azaz például 2013-10-17. Az xdev opció 1-re való állításával megmondjuk az rsync-nek, hogy a távoli gépen csatolt összes lemezt mentse le.

Ha ezzel megvagyunk, akkor – mivel ez egy időzített mentés – biztosítani kell, hogy a root felhasználó jelszó nélkül tudjon csatlakozni. A legjobb és egyben biztonsági szempontból is elfogadható kompromisszumos megoldás erre, ha a mentendő szerver root felhasználója számára létrehozunk egy megfelelő SSH kulcsot, amely nem kér jelszót, a következőképpen:

```
ssh-keygen -t rsa -b 8192 -C elesszerver
```

majd megadjuk neki, hogy mi legyen a kulcsállomány neve:

```
Enter file in which to save the key (/root/.ssh/id_rsa): /root/elesszerver1
```

A következő kérdésre, azaz mi legyen a kulcs jelszava, csak egy Entert nyomunk:

```
Enter passphrase (empty for no passphrase):
```

Az így elkészült kulcs használata esetén nem fog jelszót kérni az SSH kliens, így a dirvish sem, de nézzük mit kell még tennünk, annak érdekében, hogy ez működjön:

a /root/.ssh/config állományt szerkesszük (a backup gépen), és oda vegyük fel a következő sorokat:

```
Host *
  ServerAliveInterval 60
  ServerAliveCountMax 10
```

```
  Host eles1.szerver
     IdentityFile /root/elesszerver1
     Port 22
     Protocol 2
     User root
     HostName 172.27.1.1
     PasswordAuthentication no
```

Ha ezzel megvagyunk, akkor az éles szerverre fel kell juttatni az elkészített SSH-kulcs publikus részét, tehát például scp-vel másoljuk azt oda:

```
scp /root/elesszerver1.pub root@eles1.szerver.kft:.ssh/authorized_keys
```

Teszteljük is le a backup gépről indított ssh parancs segítségével, hogy minden rendben van-e, azaz beengedtük a tűzfalon, az SSH engedi a root hozzáférést, stb.:

```
ssh -i /root/elesszerver1 root@eles1.szerver.kft
```

Ha sikerült bejutnunk, akkor már csak annyi a dolgunk, hogy a dirvish-t is inicializáljuk:

```
dirvish --vault eles1.szerver --init
```

Az elkészített konfiguráció alapján a dirvish az SSH-n keresztül lementi – a kivételek kezelése mellett – az egész gépet. Majdnem készen is vagyunk, most már csak az olyan dolgokat kell még mentenünk, amelyeket futási időben nem célszerű. Ilyen tipikus példa az SQL szerverek (MySQL, MariaDB, PostgreSQL stb.), ezeket nem csak nem célszerű futási időben egy egyszerű cp vagy rsync segítségével másolni vagy szinkronizálni, hanem felesleges is, mivel tipikusan egy nyitott adatbázisba a mentési idő alatt is írnak, azaz a tartalma változik. Az aktuális állapotokat az ilyen esetben érdemes a Dirvish segítségével vagy Pre, vagy Post szkriptek írásával[105], vagy pedig előre ütemezett SQL mentés segítségével a saját segédeszközeikkel dumpolni, majd a konzisztens és mentés értékű dump állományokat akár fájlrendszer szinten már menteni és verziókövetni. De nézzünk konkrét példákat:

Jelen esetben mivel a MySQL és a MariaDB teljesen kompatibilisek (lásd bővebben az Adatbázis-kezelők[106] fejezetben) a mentési parancsok és tervezés sem fog eltérni, ezért a továbbiakban csak SQL-nek fogjuk hívni a MariaDB-t és MySQL-t. Ez a mentési megoldás feltételezi, hogy az adatbázis-szerver külön szerveren szeparálva található, így ütemezetten a cron segítségével és egyedi szkript írásával készítjük el az SQL mentését, nem a dirvish adottságaira támaszkodunk.

Az SQL adatbázisok mentésére több megoldás is adott. Egyrészt van a hagyományos út, amikor a mysqldump-ot használjuk, vagy rendelkezésre áll egy hotcopy nevű kiegészítés, amelyet inkább csak a mysqldump mellé, biztosítékként, vagy más cél mentés készítésére találtak ki.

mysqldump[107]

Kifejezetten a konzisztens adatbázis-mentésre kihegyezett eszköz, a mysql csomag telepítésével a birtokosai leszünk az eszköznek. Természetesen ugyanúgy szabályozható az /etc/mysql/my.conf fájlban a dump által használt erőforrások egy része is, mint a sima MySQL csatlakozások esetén.

105. http://wiki.dirvish.org/ClientScriptsOnServer

106. http://szabadszoftver.kormany.hu/szabad-szoftver-keretrendszer/
107. http://dev.mysql.com/doc/refman/5.1/en/mysqldump.html

Lokális mentés esetében a bevett módszer, hogy a mysqldump parancsot egy szkript részeként crontab-ból hívva egy olyan időintervallumban futtatjuk, amikor a szerver úgynevezett idle (üresjárati, csúcsterhelésen kívüli) periódusban van. Az ilyen időszakok tervezésében nagy segítségünkre lehet a munin, amelyről a Naplózás, monitoring[108] fejezetben olvashatunk bővebben. Egy egyszerű mentő szkript tehát valahogy így nézzen ki (az adatbázis-szerveren)

```
a szkript neve és elérési útja: /backup/ment.sh
#! /bin/sh
  /usr/bin/mysqldump --all-databases -pjelszo --events | /bin/gzip >
  /backup/SQL/mysql-ALLDATABASE-$1.tgz
  /usr/bin/mysqldump -p --skip-lock-tables information_schema | /bin/gzip >
  /backup/SQL/openmail-mysql-information_schema-$1.tgz
  /usr/bin/mysqldump -p --events mysql | /bin/gzip > /backup/SQL/openmail-
  mysql-mysql-$1.tgz
  /usr/bin/mysqldump -p website | /bin/gzip > /backup/SQL/openmail-mysql-
  website-$1.tgz
```

Mint az látható, az első sorban az összes adatbázist kimentjük egy nagy állományba, majd mindent még egyszer kimentünk egyenként külön tömörített állományokba is. Miért szükséges ez? A gyakorlati tapasztalat azt mutatja, hogy ha csak mindent egy nagy állományba mentenénk ki, akkor viszonylag nehezen férhetnénk hozzá, ha csak egy adott rekordot töröl le pl egy programozó a weboldalunk adatbázis-állományából és csak arra van szükség, akkor elég csak azt az egy adatbázist visszadumpolni egy teszt adatbázisba. Viszont ha például egy korrupt fájlrendszer miatt kell teljes visszaállítást végeznünk, akkor célszerű az egész komplett mentést egyszerre visszatölteni (--all-databases), hiszen ilyenkor minden reláció (amelyek az information_schema, vagy mysql nevű adatbázisban vannak tárolva) visszatöltődik. Éppen ezért, ha van rá lehetőségünk (nyilvánvalóan egy sok 100 GB vagy TB-os adatbázis mentését ne így végezzük) és lemezterületünk, akkor mind a két módszerrel mentsünk. Az első inicializáló mentést végezhetjük a time parancs a mentő szkript elé helyezésével, hogy legyen fogalmunk arról, az SQL motor mentés körülbelül mennyi időt vesz igénybe:

```
time /backup/ment.sh
```

Figyeljünk rá, hogy az SQL mentés néha képes hibákat visszaadni, így a cron kimenete legyen mindig egy olyan e-mail címre irányítva, amelyet olvasunk is. Ugyanis annál, hogy nincs mentés, csak az rosszabb, ha azt hisszük, tökéletes mentést készítünk, de már évek óta a /dev/null-ba irányított hibaüzenetekkel fut a mentés, amelynek esetlegesen az az eredménye, hogy üres vagy éppen hibás állományt mentünk. Éppen ezért célszerű néha kicsomagolni az SQL mentés állományait vagy zless segítségével belenézni (persze csak ha nem sok GB nagyságúak). Illetve célszerű betervezni olyan napokat esetleg havonta vagy fél évente, amikor a teljes SQL motort egy teszt környezetben újra felépítjük a mentésből.

Felmerülhet a kérdés, hogy mi van akkor, ha én az összes adatbázist is menteni szeretném egyben és külön-külön az adatbázisokat, de nem érek rá lekövetni, hogy ki mikor hoz létre új adatbázist, de mégis szeretném, hogy azok is benne legyenek a szeparált mentésben?

Erre egy remek lehetőséget ad a következő szkript[109], amely a mysql show és a mysqldump tudását használja fel arra, hogy ha valaki létrehozott a tudomásunkon kívül egy új adatbázist, akkor az is biztosan le legyen mentve:

108. http://szabadszoftver.kormany.hu/szabad-szoftver-keretrendszer/
109. http://dev.mensfeld.pl/2013/04/backup-mysql-dump-all-your-mysql-databases-in-separate-files/

```
#! /bin/bash

  TIMESTAMP=$(date +"%F")
  BACKUP_DIR="/backup/$TIMESTAMP"
  MYSQL_USER="backup"
  MYSQL=/usr/bin/mysql
  MYSQL_PASSWORD="jelszo"
  MYSQLDUMP=/usr/bin/mysqldump

  mkdir -p "$BACKUP_DIR/mysql"

  databases=`$MYSQL --user=$MYSQL_USER -p$MYSQL_PASSWORD -e "SHOW DATABASES;" |
  grep -Ev "(Database|information_schema)"`

  for db in $databases; do
    $MYSQLDUMP --force --opt --user=$MYSQL_USER -p$MYSQL_PASSWORD --databases
  $db | gzip > "$BACKUP_DIR/mysql/$db.gz"
  done
```

Mysqlhotcopy[110]

A hotcopy egy remek megoldást kínál olyan esetekre, amikor csak bizonyos rekordokat kell visszaállítanunk, vagy éppen muszáj futási időben az éles szerveren menteni, de sok konkurens írás/olvasás miatt a mysqldump zárolásos mentési módszere nem várható ki, vagy olyan lassulást eredményezne, amely nem vállalható futási időben. Azaz ezt a módszert főként napközbeni duplikálásra, vagy közbenső mentésre ajánljuk. Használata csak az InnoDB[111] típusú adatbázis esetén engedélyezett, viszont mivel általában ez az alapbeállítás, így általános esetben működőképes. A kézikönyve szerint online mentésnek hívják, ugyanis nem zárolja az egész adatbázist, hanem táblánként használja a lock, a flush és az unlock parancsokat, miközben kimásolja állományba *.frm, *.MYD, *.MYI állományokat. Használhatjuk ugyanúgy szkriptben és cronból meghívva:

```
mysqlhotcopy -u root -p jelszo dbneve /backups/SQL/HOTCOPY/ --allowold --keepold
```

Jellemzően a rendszergazdák az SQL mentő szkripteket a /backup könyvtárban tárolják, és onnan is hívják meg. Két fontos dolog amire érdemes figyelni: a /backup könyvtár mindenestül csak a root számára legyen olvasható, írható és bejárást (+x) is csak számára biztosítsunk. A második fontos tanács, hogy a backup szkriptet is mentsük. Egy esetleges átállás vagy adatvesztés esetén egy rendes backup szkript elkészítése is sok munkaórát takarhat. Ezért érdemes vagy átlinkelni az /etc alá, amit mentünk, vagy eleve ott tárolni.

Dirvish pre szkript vagy ütemezett mentés az SQL gépen?

A válasz viszonylag egyszerű: ha nem akarunk vagy nem tudunk törődni azzal, hogy kézzel ütemezzük be az SQL mentéseket, akkor célszerű post szkriptet készíteni, hiszen így a mentési folyamat fogja megszólítani és utasítani az SQL gépet a mentésre, majd ha az készen van, akkor fogja átemelni a backup gépre a biztosan elkészült mentést. Ha viszont az SQL gépen már eleve kénytelenek voltunk más megfontolásból lefuttatni a mentést például hajnali 2-kor, mivel csak 2-3 között vállalható az, hogy egy ekkora kaliberű terhelés érje a gépet – azaz ez a szempont fontosabb – akkor ehhez kell igazítanunk a dirvish-t, és esetlegesen beépíteni úgynevezett jelzőket a rendszerbe.

110. http://dev.mysql.com/doc/refman/5.0/en/mysqlhotcopy.html

111. http://dev.mysql.com/doc/refman/5.0/en/innodb-storage-engine.html

Azaz, ha valamiért (nem várt lassulás, például a terhelés megugrása miatt) az SQL mentés bár elindult hajnali 2-kor, de nem záródik le a megszokott 45 perc alatt, akkor a mentő szkript elhelyezhet egy LOCK állományt a /backup/SQL alatt (echo /backup/LOCK), amelyet egy dirvish post szkript segítségével figyelhetünk, és kaphatunk róla értesítést, vagy felfüggeszthetjük a mentést a LOCK állomány meglétéig. Felfüggesztés esetén ütemezzük úgy, hogy a dirvish utolsó dolga az SQL állományok áthozatala legyen, hogy egy esetleges SQL szerver fennakadás és ezzel együtt a mentés megakadása miatt a többi rész mentése ne akadhasson el.

Hogyan mentsünk, melyiket használjuk és mikor?

A leggyakoribb mentőeszközöket soroltuk fel, amelyek segítségével össze lehet állítani egy ideális mentést a dokumentum elején felsorolt szempontok figyelembe vétele mellett. Dönteni egyik vagy másik mentőrendszer mellett a mentendő szerverek és kliensek ismerete nélkül nem célszerű. Ajánlásunk az, hogy kombinálva használjuk a fenti rendszereket szükség szerint. Azaz a klienseket sok esetben nem szabad kifelejteni a mentésből, még akkor sem ha belső szabályzat kimondja, hogy a gépen tárolt dokumentumok nincsenek mentve, hiszen az adatvesztés ilyenkor is jelentős lehet, ha egy felhasználó megszegve az előírásokat a saját gépén tárolta a fontos dokumentumokat.

Kisvállalati, kis irodai közegben éppen ezért a szerverek mentésének Bacula, Dirvish használatával vagy egyéb komplex módon történő megoldása esetén is érdemes gondolni a kliensgépekre, és ha esetleg nincs lehetőség azok befűzésére a központi mentő megoldások közé, akkor is megoldás lehet, ha a kliens gépre telepítünk (értelemszerűen asztali Linux esetén) egy rsnapshotot, amely például a halt parancs meghívása előtt, vagy akár ütemezetten cronból a lokális merevlemezre végez egy mentést. Így ha a felhasználó nem is tartotta be a játékszabályokat, nem éri adatvesztés egy véletlen törlés esetén. Természetesen fizikai hiba ellen ez sem véd, hiszen ugyanazon az eszközön tároltuk a mentést és az éles adatokat. Az ilyen esetek ellen csak a független mentő szerver kialakítása védhet, ahol minden alaposan mentve és tesztelve is van.

Szerverek esetében pedig érdemes kombinálni a mentéseket. Ha van rá lehetőségünk, akkor egy teljes gép tönkremenetele esetén nagyban gyorsítja a visszaállítást, ha van egy teljes lemezkép, amelyet esetleg csak a napi mentéshez kell aktualizálni. Ilyenkor egy arra betanított ügyeletes is képes lehet az alap lemezkép visszaállítására, amely órákkal vagy akár napokkal is gyorsabb lehet, mint egy alaprendszer telepítése, majd arra a rendszer visszaszinkronizálása mentésből. Érdemes felkészülni, és akár papíralapú dokumentáció formájában a szerver mellett tartani egy boot loader dokumentációt. Hasznos lehet, mivel tipikusan a boot loader az, amelyet nem használunk nap mint nap. Nagyon ritkán, de előfordulhat, hogy az adataink nem sérülnek, de a partíciós tábla igen. Az üzemeltetők jelentős része ezt soha sem menti, hiszen sok éves tapasztalat alapján nincs rá szükség, ugyanakkor csak egy egyszerű parancs, és ha gond adódik a táblával, akkor van kiindulási alap. Akár így is:

```
dd if=/dev/sda of=/mentes/gepemneve-lemezem bs=512 count=1
```

Helyreállítás a mentés lemezkép alapján:

```
dd if=/mnt/gepemneve-lemezem of=/dev/sda bs=1 count=64 skip=446 seek=446
```

Pull vagy Push?

A felsorolt mentő eszközök jelentős része támogatja lehetőséget ad arra, hogy egy központi mentő szerverről indított távoli általában SSH-n keresztüli bejelentkezésen keresztül a mentő szerver mentsen, illetve azt is, hogy fordítva a kliensek kezdeményezzék a mentést a távoli backup szerverre. Mind a két megoldás lehet jó és rossz is egyszerre.

Ha képesek vagyunk a mentő szervert fizikailag, hálózatilag és szoftveresen teljesen szeparálni, azaz jól megvédeni, akkor adott esetben jó megoldás lehet, ha a mentő szerver rendelkezik a távoli szerverekre root vagy azzal egyenértékű jogokkal és így a mentő szerver fogja ütemezetten kezdeményezni a mentéseket. Ebben az esetben viszont mivel az összes éles szerver SSH kulcsát feltettük a mentő szerverre, a mentő szervert ténylegesen extrém módon meg kell védenünk és bármilyen hozzáférést tiltani.

Ha a másik módszert választjuk, amikor is az éles szervereink jelentkeznek be a mentő szerverre, akkor nem koncentrálódik egy kézben az összes hozzáférés, viszont ebben az esetben az Achilles-sarka a mentésnek az lesz, hogy az éles szerver esetleges kompromittálódása esetén a támadó el fog „látni” a backup szerver irányába. Ez ellen védekeznünk kell mindenképpen. Erre több jó megoldás is van. Az egyik ezek közül, ha a mentő szerveren minden éles szervernek elkülönített chrootolt (SFTP vagy scponly shell) hozzáférése van. Ekkor a támadó „csak” az éles szervert és annak mentését tudja megsemmisíteni. Ennek kikerülésére, ha ezt a módot választjuk, érdemes a chroot környezetből egy időzített szkript segítségével minden mentés végeztével a backup anyagokat elmozgatni egy olyan területre, ahol a chrootolt backup felhasználó – amely az éles szerver felől jön – nem láthatja már. Sajnos számtalan esettanulmány bizonyítja internet szerte, hogy ha egyszer ténylegesen betörtek az éles szerverünkre, akkor a legtöbb esetben – célzott támadás esetén mindenképpen– végigmennek az elérhető összes lehetőségen (mentett gépek továbbtámadása) is, így a backup mint egy könnyen felderíthető (cron szkripteket használó) rendszer, adja magát. Ebből a szemszögből nézve adott esetben jobb megoldás lehet az első változat, amikor a backup szerver kívülről az éles szerverek felől elérhetetlen, csak ő tud kezdeményezni.

Visszaállítási tesztek: Ahogy már említettük mindenképpen legyen egy ütemezett tervünk, mikor melyik mentést teszteljük és állítjuk belőle vissza az adatokat. Ha nincs rá külön fenntartott gépünk, akkor használjunk virtualizációs környezetet.

Kliensek mentése

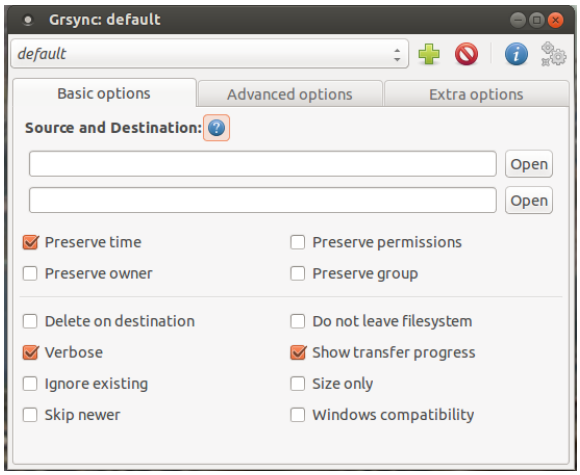
Manapság inkább az jellemző, hogy az asztali számítógépek mint egyfajta vékonykliens viselkednek, azaz tárolunk ugyan adatokat rajta, de a munkával összefüggő dolgok vagy egy böngészőben, vagy egy SSH terminálban, vagy egy távoli SMB megosztásra történnek. Ebben az esetben az alapvető hozzáállás, hogy a klienst 1-2 órán belül ugyanolyanra tudjuk varázsolni, ha tényleg mindent a szerverre mentettünk. Ez így is van, azonban még így is célszerű menteni a kliensek /etc/ alatti konfigurációs állományait és legalább a /home környezetet. Ha a menteni kívánt kliensek Linux vagy BSD, vagy bármilyen Unix-klón alapúak, akkor szerencsések vagyunk, mert a már megismert módszerek egyikét alkalmazhatjuk. Azaz ha az asztali számítógép tud SSH kapcsolatot indítani vagy fogadni, akkor ugyanúgy menthetjük Rsnapshot-tal vagy akár Dirvish-sel is központilag. Ha pedig szeretnénk a felhasználóra bízni (ez általában nem jó ötlet), hogy a saját klien-

sén mit ment és mit nem (hiszen egy központilag mentett meghajtóra dolgozik amúgy is) akkor például a mentendő kliensen futtathatunk Grsyncet.

Grync[112]

Ezt a grafikus felületet az rsync meglehetősen bonyolult opcióinak átláthatóbbá tételére hozták létre. A fejlesztők a leginkább használt funkciókat grafikus formába öntötték. Akár a teljes funkcionalitást könnyen el lehet érni a kiterjesztett nézetben, azonban az igazán hasznos funkciója az, hogy az 1-2 perc alatt egérrel összeállított beállításokat egy nézet menüben parancs formájában is lementhetjük. Kezdő rendszergazdáknak ideális a kapcsolók nem teljes ismerete esetén, ugyanis a tévedést lehet vele minimalizálni. Működtetését szerverek között érdemes kombinálni az ssh-agent[113] és az ~/.ssh/config állományba felvett rövid gépnevek felsorolásaival. Például:

```
Host szervergep
     IdentityFile /home/admin/identity-szerver-titkoskulcs
     Port 22
     Protocol 2
     User root
     HostName szerver.belsohalo.hu
     PasswordAuthentication no
```



Ezzel az apró SSH-kliens beállítással a Grsync menüjében (vagy akár az rsync parancssorában) elég a szervergep hivatkozást használni, és máris tudni fogja a többi (például port és kulcs) paramétert. Ha ez kombinálva van az ssh-agent-tel, akkor még jelszót sem fog kérni. Így például akár automatikus szinkronizálást is megvalósíthatunk.

Asztali gép esetében, ha ugyanúgy be van kapcsolva 0-24 órában, mint egy szerver, akkor használhatjuk a cront időzítésre, ha viszont a munkatársak kikapcsolják a gépeiket távozáskor, akkor érdemes meglovagolni a megfelelő init szintet, azaz a kikapcsolásnál lefuttatott init folyamatba befűzni a mentő szkriptet. Így ha kiadjuk a halt parancsot, vagy megnyomjuk a kikapcsolás gombot, akkor elsőnek a saját mentő szkriptünk fog lefutni a háttérben. Ezzel késlelteti ugyan a gép kikapcsolását, de így nincs szükség a felhasználó interakciójára, azaz minden kikapcsolásnál le fog futni az rsnapshot, vagy akár az rsync és menteni fog a távoli szerverre.

112. http://www.opbyte.it/grsync/
113. http://en.wikipedia.org/wiki/Ssh-agent

Windowsos munkaállomások mentése

Ugyanúgy, ahogyan a Linux alapú kliensek estében is lehetőség van lemezkép alapon az egész lemezterületet menteni, vagy állományonként differenciált mentést készíteni, gyakorlatilag az MS világban is léteznek erre remek megoldások, a teljesség igénye nélkül néhány hasznos sciprt:

Robocopy[114]

Windows Backup VHD image

Adatmegsemmisítés

A mentéshez kapcsolható feladat a nem használt **eszközök biztonságos megsemmisítése**, amelyre oly gyakran nem fordítunk egyáltalán figyelmet. Egy már használaton kívüli szerver merevlemeze, vagy bármilyen adattárolója veszélyforrást[115] jelenthet, ha csak egyszerűen kidobjuk. A szalagos egységek, merevlemezek, Flash drive-ok jelentik a mentéshez kapcsolódóan a legnagyobb veszélyt, mivel a régi használaton kívüli egységeket ritkán szokták biztonságosan megsemmisíteni, egyszerűen csak kidobják a szemétbe. Rendszeresen olvasni a hírekben, hogy különböző adatbiztonsággal foglalkozó cégek véletlenszerűen vásárolnak használt komplett PC-ket, vagy csak adathordozókat, és próba visszaállításokat végeznek. A legtöbb esetben már eleve csak gyors formázást, vagy még azt sem végeznek az emberek, így az adatok könnyedén visszaállíthatóak. A szerverek illetve szalagok esetében valamivel bonyolultabb a helyzet, mert komolyabb szaktudás szükséges a visszaállításhoz például egy RAID eszköz esetén. Érdemes tehát számításba venni, hogy a kidobásra ítélt eszközök adathordozói veszélyforrást jelenthetnek. Egy remek eszköz a **Wipe**[116], amely Linux alatt az adatok biztonságos felülírását végzi. Azonban erre erőforrást kell dedikálni, mivel a Wipe folyamat is egy biztonságos törlés esetén akár napokat is igénybe vehet. Természetesen minden ilyen esetben a leginkább biztonságos megoldás a fizikai megsemmisítés és a szoftveres megsemmisítés kombinálása. De ha erre csak időt fordíthatunk, pénzt nem, akkor legalább az alapos 8-10-szeres Wipe felülírást, azaz a szoftveres megsemmisítést válasszuk az adathordozók nagy részénél.

114. http://en.wikipedia.org/wiki/Robocopy
115. http://www.wikihow.com/Destroy-a-Hard-Drive
116. http://wipe.sourceforge.net/

Rendszerindítás, init, rendszerkomponensek konfigurálása (/etc)

A számítógépes rendszerek egyik fontos folyamata az elindításuk. Bár ideális esetben erre egy szerver teljes életciklusa során csak egyszer lenne szükség, a valóságban ez ritka – a legnagyobb gondosság ellenére is előfordulhat hardverhiba, áramszünet, újraindítást igénylő szoftvermódosítás.

Emiatt is fontos, hogy a rendszer indítása teljesen automatikus, kipróbált és bármikor megismételhető legyen – ne egy áramszünet után derüljön ki, hogy nem indul el minden szolgáltatás, vagy nem töltődik be minden beállítás. Ha még sincs minden rendben, akkor jobb esetben csak nem működik valami, de rosszabb esetben biztonsági hiányosság keletkezik a rendszerben.

> A rendszerindításra úgy is tekinthetünk, hogy egy ismert, a gép bekapcsolását követő, kezdőállapotból jól definiált lépésekkel jut el a kívánt üzemi állapotba.
>
> Napjaink irodai- és kiszolgálógépei bekapcsoláskor alaphelyzetbe állítják a komponenseiket, majd végrehajtják a rendszer indítását végző szoftvert, a BIOS-t vagy az UEFI-t, amelyek elvégzik a szükséges hardverteszteket, felderítik az operációs rendszer betöltésére alkalmas forrásokat, majd a beállítások alapján megkezdik a rendszerbetöltést a megfelelő forrásból (például merevlemezről, cserélhető médiáról vagy a hálózatról).

A bootloader

Az operációs rendszereket általában merevlemezre (vagy újabban szilárdtestmeghajtóra) telepítik. A BIOS-t használó rendszerek esetében a rendszerbetöltésre kiválasztott lemez elején (master boot record, MBR) szerepel egy rendszerindító, amely képes megkezdeni az operációs rendszer betöltését. Ez a rendszerbetöltő (bootloader) modern Linux rendszerek esetében általában a GNU GRUB 2, amely számos összetett tárolási megoldással együttműködve tudja betölteni a megfelelő rendszermagot, valamint lehetővé tenni a rendszerek közti választást. Régebbi rendszerek a jelentősen eltérő működésű GRUB Legacy (GRUB 1) vagy a LILO bootloadereket használják. A GRUB 2 be tudja tölteni az operációs rendszert különböző RAID-tömbökről és LVM-ről is, valamint számos típusú fájlrendszerről. Beállítása rendszerint a telepítő feladata.

Konfigurációját a /etc/default/grub fájlban szereplő változók és a /etc/grub.d/ könyvtárban szereplő héjprogramok alkotják, amelyeket a update-grub parancs futtat, majd kimenetüket a /boot/grub/grub.cfg fájlba fűzi össze (amely kimenet szintén egy héjprogramot ad, azonban ezt a GRUB 2 futtatja a rendszer betöltésekor). A GRUB 2 telepítésére a grub-install parancs szolgál, melynek argumentumaként kell megadnunk a telepítésre kijelölt lemezeszközt (például grub-install /dev/sda).

Amennyiben nem oldható meg, hogy a rendszerbetöltő olvasni tudja a rendszer gyökérfájlrendszerét, akkor szükség lehet arra is, hogy a kernelt és a bootloader moduljait tartalmazó /boot/ könyvtár egy külön, egyszerűbben olvasható – például nem titkosított, a lemez elején lévő – köteten legyen.

Ha a kernel betöltését a bootloader elvégezte, megkezdődhet az operációs rendszer indításának érdemi része, az init folyamat. Ez sem lehetséges azonban azonnal minden esetben. Ennek fő oka az, hogy a folyamathoz szükséges az, hogy a gyökérfájlrendszer elérhető legyen a kernel számára. A disztribúciók azonban olyan kerneleket szállítanak, amelyek az összes támogatott hardveren működnek. A mindezekhez szükséges meghajtók memóriában tartása, a kernelbe történő statikus „beforgatása” azonban nem célszerű, ezért dinamikusan betölthető kernelmodulokkal oldják meg a széles hardverpaletta támogatását. Ezen modulokat viszont valahonnan be kell tölteni, amire az initrd, egy az indítás idejére használt, a memóriába tölthető lemezkép a megoldás. Ez a lemezkép tartalmazza a szükséges modulokat, valamint egy héjprogramot, amely elvégzi a végleges gyökérfájlrendszer csatolását és az ehhez szükséges feladatokat. Egyéni initramfs képek létrehozására szolgál a széles körben használt dracut keretrendszer.

Hálózatról történő rendszerbetöltésnél is ez a megoldás, azonban ilyenkor az initrd feladata a hálózat beállítása és a hálózati fájlrendszer csatolása is. A disztribúciók telepítői is gyakran az initrdben kerülnek megvalósításra.

A GRUB 2 konfigurációjához általában elegendő a /etc/default/grub fájl módosítása, mivel a telepített kernelekhez és a telepítéskor felfedezett más operációs rendszerekhez automatikusan létrejönnek a bejegyzések. A `GRUB_DEFAULT` opcióban az alapértelmezett bejegyzés sorszámát vagy nevét adhatjuk meg; míg a `GRUB_CMDLINE_LINUX` opcióval kiegészíthetjük a kernelnek átadott paramétereket (például ha gondjaink vannak a konzol megjelenítésével, itt adhatjuk meg a `nomodeset` opciót, amely letiltja a felbontás automatikus állítását). A konfiguráció módosítása után adjuk ki az `update-grub` parancsot.

Az alapértelmezettől eltérő rendszer indításához a rendszer indítása közben megjelenő menüből választhatunk a billentyűzettel. A `GRUB_TIMEOUT=0` beállítás esetén azonban ez a menü nem jelenik meg, csak akkor, ha a GRUB 2 betöltése pillanatában a shift billentyű le volt nyomva. Amennyiben a megjelenő menüben nem találunk megfelelő bejegyzést, akkor a képernyőn szereplő útmutatás alapján módosíthatjuk is a meglévőket. Például a `linux` kezdetű sor végére `init=/bin/bash`-t írva a rendszerszolgáltatások elindítása helyett jelszó megadása nélkül egy parancsértelmezőt indíthatunk.

Init

Miután betöltődött a kernel és elvégezte a folyamatok indításához szükséges lépéseket, elindul az init folyamat, amely a felhasználói térben (userspace) futó szolgáltatások indítását és megállítását hajtja végre – ebből adódóan minden folyamat az init leszármazottja.

Az init rendszernek egészen a közelmúltig a SysV[117] megvalósítása és a vele kompatibilis megoldások volt a jellemző, azonban főleg a teljesítményével kapcsolatos kifogások miatt két új megoldás, az Upstart és a Systemd terjedt el. A rendszerek közötti érdemi választás lehetősége általában

117. A System V (System Five, röviden SysV) a nyolcvanas évek egyik sikeres UNIX-változata. Fő konkurense a BSD (Berkeley Software Distribution) volt.

a disztribúció készítőjénél van, így azt kell használnunk, amit az általunk használt disztribúció kínál.

SysV init

A hagyományos SysV inittel kompatibilis megoldás található meg a Debian disztribúció aktuális kiadásában, valamint a beágyazott rendszereken is ez a jellemző. Az újabb init-rendszereket használó terjesztések is igyekeztek lehetővé tenni a SysV inithez készült szolgáltatások használatát kompatibilitási interfészek biztosításával.

A megoldás központi fogalma a futási szint (runlevel), amely a rendszer egy pillanatnyi jellemzője. Az egyes futási szinteket 0 és 6 közötti egész számokkal, valamint „S” betűvel jelöljük, amely futási szintek értelmezése az egyes rendszerekben változatos. Általánosságban elmondható, hogy a „0” jelű speciális futási szint a gép kikapcsolását, a „6” jelű pedig újraindítását jelenti. Az 1-es futási szint általában az egy felhasználós mód, amelyben olyan adminisztratív feladatokat tudunk végrehajtani, ami a szokásos szolgáltatások futása mellett nem lehetséges vagy célszerűtlen. A 2–5 futási szintek értelmezése nagyobb változatosságot mutat, a 2-es vagy 3-as szint a többfelhasználós, a szolgáltatásokat is tartalmazó futási szint, míg a 4–5-ös egyes rendszereken (például Debian) a grafikus rendszerrel bővíti a kisebb szintek szolgáltatáshalmazát. Más rendszereken a 2-es szint jelöli a teljes többfelhasználós módot, míg a nagyobb szinteket nem használják. Az „S” futási szint szerepe is változó, Debian alatt a csak bootoláskor indítandó szolgáltatások helye, más rendszereken nem használják, vagy az egyfelhasználós mód egy szinonimája.

A futási szintek között az init paranccsal válthatunk, például a gép újraindításához: `init 6`. A rendszer betöltésekor az alapértelmezett futási szintet éri el, amelyet az init konfigurációs fájlja határoz meg.

Az init működését a /etc/inittab fájlban állíthatjuk be. A fájl kettőspontokkal elválasztott mezőkből álló sorai egy-egy szolgáltatást írnak le, és egy rövid szolgáltatásazonosítóból, a szolgáltatás számára kijelölt futási szintek listájából, az indítás módjából és végül a végrehajtandó parancsból állnak. Például egy virtuális terminál futtatása a 2–5-ös futási szinteken (a respawn mód azt jelenti, hogy a programot kilépése esetén újból kell indítani):

```
1:2345:respawn:/sbin/getty 38400 tty1
```

Speciális indítási mód az initdefault, amely az alapértelmezett futási szintet jelöli. Például 3-as alapértelmezett szintet állít be a következő sor:

```
is:3:initdefault:
```

A legtöbb SysV initet használó Linux-disztribúció alatt nem kell itt az alapértelmezett futási szinten kívül mást beállítanunk, mivel az alapértelmezett konfiguráció egy dinamikusabb megoldást tartalmaz. Az egyes futási szintekhez tartozik egy könyvtár, általában /etc/rc[0-6S].d/ néven, amelyben init szkriptekre mutató szimbolikus linkek találhatóak.

Az init szkriptek a /etc/init.d/ könyvtárban találhatóak, és szolgáltatások elindítására (start), leállítására (stop), állapotuk lekérdezésére (status), vagy újraindításukra (restart), újratöltésükre (reload) szolgálnak, rendre a megadott argumentummal futtatva. Például Debian alatt a telepített, de jelenleg nem futó Apache webkiszolgálót a /etc/init.d/apache2 start paranccsal indíthatjuk el.

Ezekre az init szkriptekre mutatnak szimbolikus linkek a /etc/rc?.d/ könyvtárakból, kötött elnevezésekkel. A link első karaktere egy „S” vagy egy „K”, amely azt jelöli, hogy az adott szintre lépve az adott szolgáltatást indítani (Start), vagy leállítani (Kill) kell. Ezt követi két számjegy, amely az indítások sorrendjét adja meg – egyes szolgáltatásokat csak akkor lehet elindítani, ha másikak

már futnak. A link neve a szolgáltatás nevével (ahogy az init szkript szerepel) fejeződik be. Például a második futási szinten 91.-ként indítandó Apache webkiszolgáló linkje a következő: /etc/rc2.d/S91apache2 -> ../init.d/apache2 .

Mivel ezeknek a linkeknek a karbantartása is nehézkes, ezt a disztribúciók igyekeznek megkönnyíteni. A Debian az update-rc.d parancsot biztosítja. Egy új szolgáltatás (a példákban: foobar) bekapcsolásához az alapértelmezett futási szinteken, 20.-ként indítsuk és 80.-ként állítsuk le, futtassuk az update-rc.d foobar defaults 20 80 parancsot. Ha az init szkript elején szerepel egy speciális formátumú megjegyzés, akkor az alapértelmezett futási szinteket onnan veszi az update-rc.d, egyébként a 2–5 futási szintekre állít „S” linket.

Egy szolgáltatás tiltásához először töröljük a linkeket (update-rc.d -f foobar remove), majd vegyük föl az összes futási szinthez a „K” linket: update-rc.d foobar stop 20 2 3 4 5 ..

Upstart

Az Upstart az Ubuntu, valamint az aktuális CentOS és RHEL disztribúciók alapértelmezett init rendszere.

Systemd

A systemd a Fedora és az openSUSE disztribúciók alapértelmezett init rendszere.

Alapszintű hálózati hibakeresés

Az alapszintű hibakezelés részben szereplő jó tanácsot ismételjük el:

Az elejére általánosságban 2 alapszabály:

– olvassuk és próbáljuk meg értelmezni a hibaüzenetet
– nézzük meg a hibanaplókat.

Akkor most mehetünk tovább.

Hálózati problémák

Sajnos általánosságban is, a hálózattal kapcsolatban is elmondható, millióféle hibahelyzet lehet. Ha „nem-megy-a-hálózat”, annak oka lehet, hogy nincs bedugva a kábel – ellenőrzése az ip paranccsal:

```
ip link show dev eth0
```

és ellenőrizzük, nem látjuk-e véletlenül a NO-CARRIER jelzőt. A hiba lehet rosszul beállított IP-cím, nyilván lekérdezni az

```
ip addr sh dev eth0
```

paranccsal kell, és az inet kezdetű sorokat ellenőrizzük (ilyen sorokban szerepelnek a gép IPv4-es címei).

Lehet szimpla routing probléma a háttérben, kérdezzük le.

```
ip ro sh
```

Ellenőrizzük az elérni kívánt hálózathoz tartozó bejegyzést. Ha az elérni kívánt hálózatra vonatkozó bejegyzés nincs, akkor ellenőrizzük a default bejegyzést.

getent

Hálózati problémát okozhat névfeloldási hiba, sajnos sokan és sokszor azonnal a névszerverekkel kezdik a dolgot, holott előtte van még egy pont – maga a névfeloldást végző keretrendszer (az NSS) hibás konfigurációja. Tesztelésére teljesen jól használható a getent. Két paramétert vár: melyik adatbázist kell lekérdeznie (felhasználók, csoportok, gépek, stb.) – és mi a lekérdezendő adat. Azaz ha névfeloldási hibára gyanakszunk, akkor

```
getent hosts keresett-szerverneve-vagy-cime ; echo $?
```

Sajnos pl. nem létező szervernév esetén egy hangot nem ad ki magából, csak a 2-s hibastátusz (ennek értékét írja ki a pontosvessző utáni parancs) segít értelmezni, hogy jelenleg valami nem stimmel – nem találja a keresett adatot. Azzal együtt is javasolt a státszkódot ellenőrizni, hogy ha van találat, a kapott választ kiírja, tehát a csönd már önmagában gyanús kell legyen. Természetesen ha a getent szerint nem jó a válasz, akkor érdemes megnézni a konfigurációt tartalmazó

/etc/nsswitch.conf fájlt. Ha ez is jónak látszik, de a konfiguráció szerint névszerverhez is kell fordulni (azaz a hosts: kezdetű sorban szerepel a dns kulcsszó), akkor jöhet a /etc/resolv.conf fájl ellenőrzése, s ha látszólag stimmel, csak akkor a DNS-teszt.

nslookup, host, dig

Az eszközök közül idővel mindenki megszokja valamelyiket a kínálatból, de ma még mindig nagyon sokan a már erősen elavultnak számító nslookup parancsot használják. Javasolható helyette a host, vagy a dig.

Ha gépünk a publikus hálózaton is elérhető névvel/címmel rendelkezik, akkor a külvilág elképzelését a gépünk adatairól könnyedén lekérdezhetjük, ha megjegyzünk 2 publikus névszervert: a 8.8.8.8 és a 8.8.4.4 talán nem feltétlenül javasolható mindennapos használatra, de tesztre szinte biztosan jó:

```
nslookup keresett-szerver-neve-vagy-cime 8.8.8.8
host keresett-szerver-neve-vagy-cime 8.8.8.8
dig +short keresett-szerver-neve @8.8.8.8
dig +short -x keresett-szerver-cime @8.8.8.8
```

amíg hálózaton belül keressük a hibát, addig annyit módosítsunk, hogy az utolsó paramétert (ami egyébként a lekérdezendő névszervert azonosítja) hagyjuk el a parancssorból. Ekkor a fent említett /etc/resolv.conf fájlból veszik elő a lekérdezni kívánt néveszerver adatot.

ping, tcpspray

Gyakran azzal ellenőrzik egy szerver működését, hogy „megpingetik”. A

```
ping szerverneve-vagy-címe
```

többek között azért jó, mert mellékesen a hálózat „sebességéről” is ad némi információt. Hátulütője, hogy a „nem-pingik” állapotot millió dolog okozhatja. Hálózati sebesség ellenőrzésére publikus netet elérő gép esetén jól ismert a www.speedtest.net oldal tesztje, Nem publikus gép esetén érdemes a tcpspray nevű tesztelő eszközt igénybe venni. (A túloldalra szükséges egy, általában az inetd szolgáltatásai között engedélyezhető discard, vagy echo TCP szolgáltatást nyújtó eszköz, ennek bekonfigurálásáról se feledkezzünk meg.)

traceroute, tracepath

Elsősorban távoli szerver elérési problémáinak keresésére szokták használni a traceroute, tracepath nevű eszközöket – amelyek kimenetükben többek között a csomagok útvonalát adják vissza. (Bár komolyabb hálózati határvédelem esetén ezek nagy eséllyel nem jutnak el a célig.)

A továbbiakban megemlítésre kerülő programok jó része csökkentett funkcionalitással, vagy akár sehogy sem működik, ha nem rendszergazda jogosultsággal hajtjuk őket végre, azaz sudo/su nagy eséllyel szükséges a használatukhoz.

iptraf, vnstat

Egyszerű, de jól használható forgalommonitorozó eszköz az iptraf. Szerveren előnyös kis erőforrásigénye miatt, és hogy csak karakteres felületet ad. (Akinek ez negatívum, az nézze meg az ntop-ot is.)

Ha pedig összefoglaló hálózati statisztikákat szeretnénk kapni, akkor nézzük meg a vnstatot. Feltérképezi a gépben levő interfészeket, majd folyamatosan adatbázisba gyűjti a forgalmi adatokat, amiket aztán óránkénti, napi vagy akár havi bontásban kérdezhetünk le tőle.

tcpdump, tshark, Wireshark

Komolyabb hálózati forgalomanalizálásra jól használható (bár viszonylag mélyebb hálózati ismereteket feltételez) a tcpdump nevű program. Kérhetünk tőle a teljes forgalom megjelenítését, de akár meglehetősen bonyolult feltételeket is megadhatunk, amikkel szűrhetjük amit látni szeretnénk. (Pl. host 192.168.1.42, vagy tcp port 25, mindezeket logikai operátorokkal és zárójelekkel kombinálva.) Ma egyre kevesebben használják ezt direktben. Helyette elterjedt, hogy tcpdump -w fájl formában indítják el (ekkor a hálózati forgalmat a paraméterként megadott fájlba menti), majd pedig valamilyen intelligensebb eszközzel vizsgálják meg a lementett adatot. Jellemzően karakteres felületen ez a tshark, grafikus felületen pedig Wireshark névre hallgató feldolgozószoftver. Ez a két parancs ugyanannak az eszköznek a karakteres és grafikus felületre optimalizált változata, és amennyiben azonnali forgalomfigyelést szeretnénk, nem szükséges a tcpdump-ot közbeiktatni, ezek az eszközök is képesek az interfészeken folyó forgalom figyelésére.

További hibakeresési megoldások

Előfordulhat, hogy egy hálózati alkalmazás az indulása után nem sokkal az „Address already in use” hibaüzenettel örvendeztet meg minket, és leáll. Ilyen esetekben jó szolgálatot tehet a korábban már említett strace. Indítsuk el újra az alkalmazást az strace által felügyelt módon. Az strace üzenetei között meg fogjuk találni, hogy milyen hálózati erőforrással van baj. Ez lehet akár lokális (UNIX-domain), akár távoli (Internet-domain) socket. Az strace üzenetei között meg kell keresni a sockethez tartozó paramétert (fájlnevet, vagy portszámot).

```
bind(3, {sa_family=AF_FILE, path="/tmp/socket"}, 11) = -1 EADDRINUSE (Address
  already in use)
```

Fenti példában egy un. UNIX-domain socket és a hozzá tartozó fálnév,

```
bind(3, {sa_family=AF_INET, sin_port=htons(8888),
  sin_addr=inet_addr("0.0.0.0")}, 16) = -1 EADDRINUSE (Address already in use)
```

ebben a példában pedig a 8888-as TCP[118] port foglalt.

Ha ez megvan, akkor már csak azt kell megkeresni, ki használja. Ezt megtehetjük a fuser nevű paranccsal:

118. Hogy TCP, vagy UDP portról van szó, az ebben a sorban nem látszik, hanem vissza kell követni, hogy a példában a bind első paramétereként szereplő 3-s számú socket létrehozásakor SOCK_STREAM (tcp) vagy SOCK_DGRAM (udp) paraméterrel hozták-e létre.

```
fuser -n file /tmp/socket
fuser -n tcp 8888
fuser -n udp 514
```

(A file az a UNIX-domain socketet jelenti, paramétere a socket neve; az udp/tcp értelemszerűen UDP, illetve TCP socket, és a hozzá tartozó port paramétert kell megadni. Az utolsó, udp-s példa csak a teljesség kedvéért szerepel.) A fuser parancs kimenetében egy folyamatazonosítót (PID) találunk:

```
8888/tcp:             8620
```

és már csak ki kell deríteni ez milyen folyamat:

```
ps -fp 8620
```

A kimenetben megtalálható a problémát okozó alkalmazás, aki azt a hálózati erőforrást használja, amelyik az elindítandó programunk útjában áll. A két lépcsőben használatos fuser + ps helyett használhatunk több más eszközt is. Használható akár a netstat, vagy az ss nevű programok:

```
ss -p -l -n -x | fgrep /tmp/socket
netstat -p -l -n -t | fgrep 8888
```

(a netstat egyéb UNIX-rendszerekből lehet ismerős, az ss Linux-specifikus parancs)

de egy sokkal több mindenre szolgáló eszköz is, ez az lsof:

```
lsof -i tcp:8888
```

(Mivel az lsof sokkal több információt ad meg, így kicsit nehezebb értelmezni a kiírt adatokat.)

Még egy apróság. Néha jó lenne, ha egy hálózati kapcsolatot meg tudnánk szüntetni gyorsan, szabályosan. Meglehetősen hatékony eszköz a tcpkill (valószínűleg a dsniff nevű csomagban fogjuk megtalálni). Figyeli a -i opcióval megadott hálózati interfész forgalmát, és a tcpdump kapcsán megtanult kifejezéssel leírható TCP forgalmat szabályosan lezárja.

```
tcpkill -i eth0 port 8888
```

Azaz ha biztosak vagyunk abban, hogy márpedig azon a porton nem kellene másnak kommunikálnia, mint a programnak akit épp most szeretnénk elindítani, akkor ezt is megpróbálhatjuk.

Természetesen okozhat hálózati problémákat a rosszul konfigurált tűzfal, ezért érdemes lehet a tűzfalszabályokat is ellenőrizni:

```
iptables -t filter -v -n -L
```

de az első pillanattól szokjuk meg, hogy a -v opciót is adjuk meg, e nélkül meglehetősen fontos intormációk hiányozhatnak a kimenetből. A követező példa nem szűrő, hanem a NAT – azaz címfordítási – tábla szabályait kérdezi le:

```
iptables -t nat -v -n -L
```

Sok esetben a „nem elérhető a szerver” jellegű hibajegyek kezelésére, tesztelésére is szükség lehet. Nagyon sok applikációs protokoll[119] egyszerű angol nyelvű parancsokkal működik, ezért meglehetősen gyakran használt teszt eszköz a telnet, vagy a ma már egyre több terjesztésben az alaptelepítésben elérhető nc (netcat) parancs. Használatuk meglehetősen egyszerű:

```
telnet mailszerver SMTP
nc webszerver 80
```

119. Az elektronikus levelezésben használt SMTP, IMAP4 és POP3 protokoll, a webböngészésre szolgáló HTTP ilyen, de van még pár.

A netcat annyival tud többet, hogy nem csak TCP, hanem akár UDP protokollon keresztül is képes kommunikálni, sőt akár UNIX-domain socket kezelésére is alkalmas, így ma egyre több rendszergazda használja inkább azt. (Ráadásul nem csak hálózati kliens, hanem szerver is tud lenni a -l opció hatására.) Ezekkel az eszközökkel az ilyen protokollt használó szerverek alapszintű működése könnyedén ellenőrizhető. De mi a helyzet, ha nem HTTP-t, SMTP-t, stb.-t, hanem ezek titkosított verzióit kell használni? Kézzel senki nem fogja a TLS / SSL-t lejátszani. Erre szolgál a meglehetősen elterjedt OpenSSL névre hallgató eszköz s_client paraméterrel indítva. A fenti netcat-es példa HTTPS-re lefordítva:

```
openssl s_client -connect webszerver:443
```

formában írandó. És az OpenSSL szépen elvégzi a a tanúsítványok ellenőrzését, a kommunikáció során a szükséges kódolásokat és dekódolásokat.

IP-multiplexing, Bridge, VLAN, Bonding, és a hálózatkonfigurálás 3-féle módja

Mivel a további műveletek segítségével viszonylag könnyen teljesen elronthatjuk rendszerünk hálózati beállításait, ezért javasolt ezeket a parancsokat nem a valódi hálózati interfészt használva kipróbálni. Ehhez rögtön az elején hozzunk létre egy virtuális hálózati interfészt. A továbbiakat pedig ezzel a virtuális interfésszel csináljuk meg. (Természetesen ahol a művelethez több interfészre van szükség, ott nem egy, hanem több ilyet fogunk használni.)

Virtuális interfész létrehozása

Létezik egy speciális interfész típus, neve dummy. Ilyen interfész létrehozásához szükséges egy „dummy” névre hallgató ún. kernel modul (más kérdés, hogy az ilyen típusú interfész létrehozása a legtöbb terjesztésben automatikusan betölti ezt a modult, így az azonnal rendelkezésre áll). A virtuális interfész létrehozása:

```
ip link add name dummy0 type dummy
```

Innentől kezdve van egy interfészünk, ami nem jó semmire, kivéve, hogy ennek segítségével viszonylag könnyedén ki lehet próbálni a későbbieket. Ezekkel a virtuális interfészekkel szinte mindent meg lehet csinálni, amit a valódiakkal lehet. A továbbiakban feltételezzük, hogy létrehoztunk egy ilyen dummy-típusú interfészt dummy0 néven, illetve ahol kell, ott továbbiakat is, rendre dummy1, dummy2, stb néven.

Interfész konfigurálása

Elevenítsük fel az „Egyszerű hálózati konfigurálás” című részben szereplőket. Egy interfész életre keltésének javasolt metódusa (valós interfészeknél az első parancsra nyilván nincs szükség.)

```
ip link add name dummy0 type dummy
ip link set dev dummy0 up
ip addr add 1.2.3.4/24 broadcast + dev dummy0
ip route add 1.2.3.0/24 via 172.26.4.2
```

Megszüntetni egy interfészt pedig a létrehozása:

```
ip link add name dummy1 ty dummy
```

után a:

```
ip link del dev dummy1
```

paranccsal lehet.

(Interfész létrehozása az elavultnak minősített módszerrel:

```
ifconfig dummy0 1.2.3.4 netmask 255.255.255.0 up
```

```
route add -net 1.2.3.0 netmask 255.255.255.0 gw 172.26.4.2
```

Azaz ifconfig + route parancspáros.)

Pont-pont kapcsolat létrehozása

Időnként szükségünk lehet nem csak hagyományos, ún. „broadcast” típusú interfészre, hanem pont-pont típusú kapcsolatra. Ezt kicsit másként kell konfigurálni, azaz a

```
ip address add 1.2.3.4/24 broadcast 1.2.3.255 dev dummy0
```

helyett pont-pont típusú kapcsolatot az

```
ip address add 1.2.3.4/32 peer 1.2.3.5 dev dummy0
```

paranccsal lehet beállítani.

Statisztika és részletesebb információk lekérdezése

Az interfészek paramétereinek részletes (-detail) lekérdezése

```
ip -d link show
```

Az interfész forgalmi statisztika (-statistics) a

```
ip -s link list eth0
```

paranccsal kédezhető le. Ha az opciót megkettőzzük, részletesebb statisztikát kapunk:

```
ip -s -s link list eth0
```

Egy interfész átnevezése (a biztonság kedvéért állítsuk vissza :-)):

```
ip link set dev dummy0 name eth10
ip link set dev eth10 name dummy0
```

Különböző hálózati objektumok (interfészek, interfészcímek, útválasztó bejegyzések) figyelése

```
ip monitor link | address | route | l | a | r
```

(Könnyű elrontani, mert az „m” rövidítés a „multicast-address”-t jelenti, szóval legalább „mo” vagy „mon” formában rövidítsük.)

Több IP-cím egy interfészen (IP-multiplexing/IP-aliasing)

Rendszeresen előforduló igény, hogy egy hálózati interfészhez több hálózati címet rendeljünk. Ennek javasolt módja a

```
ip addr add 2.3.4.5/24 broadcast + dev dummy0
```

parancs használata. Természetesen akár ez a „második”, akár bármelyik másik bekonfigurált IP-cím ugyanúgy szüntethető meg:

```
ip addr del 3.4.5.6/24 dev dummy0
```

Van két alternatív megoldás:

```
ip addr del dev dummy0
```

az interfészhez tartozó címlista első elemét törli, míg az

```
ip addr flush dev dummy0
```

minden címet töröl a paraméterként megadott interfészről.

Interfészekhez több IP-cím hozzárendelés az elavult megoldás szerint, egy virtuális ún. alias interfész létrehozásán alapult. Ez a virtuális interfész úgy jön létre, hogy a fizikai interfész nevéhez hozzáragasztunk egy kettőspontot, és egy egész számot – jellemzően eggyel indítva a számozást, és ezt az alias interfészt konfiguráljuk fel:

```
ifconfig dummy0:1 2.3.4.5 netmask 255.255.255.0
ifconfig dummy0:2 3.4.5.6 netmask 255.255.255.0
```

Elvenni egy ily módon hozzáadott címet úgy kell, hogy az adott virtuális interfészt „down” állapotba kapcsoljuk:

```
ifconfig dummy0:1 down
```

Arra vigyázzunk, hogy ha a „szülő” interfészt kapcsoljuk le, akkor az alias interfészek és a szülő interfész is látszólag eltűnnek – azaz az ifconfig kimenetében már nem látszanak – de az ip parancsnál igen :-). Viszont a szülő interfész visszakapcsolásakor visszajönnek az aliasok is. Ha viszont az alias interfészt kapcsoljuk „down” állapotba, akkor valóban eltűnik a cím – mind az ip, mind az ifconfig kimenetéből, sőt, később már „up” állapotba se tehetjük, csak ha újrakonfiguráljuk.

Ezen kívül az a furcsa helyzet állt elő, hogy ha plusz IP-címet rakok egy interfészre a régi módon (ifconfig paranccsal beállítva az alias interfészt), akkor ha az ifconfig paranccsal kérdezem le, akkor önálló interfészekként látszanak, az ip parancsban pedig egyszerűen látszik, hogy az eredeti interfésznek több címe van. Viszont ha a ma javasolható ip paranccsal adok több címet egy interfészhez, akkor ezeket a címeket már nem látjuk az ifconfig listájában. Ennel elkerülése érdekében egy apró trükk:

```
ip addr add 3.4.5.6/24 brd + dev dummy0 label dummy0:1
```

Azaz ha plusz IP-címeket adunk egy interfészhez, akkor a második, harmadik, stb IP-cím hozzáadásakor adjunk meg egy plusz „label” paramétert is egy alias interfész-szerű „névvel”, és máris mind az ip, mind az ifconfig paranccsal látható és kezelhető lesz.

VLAN létrehozása

Sok helyen szükséges virtuális hálózatokat, ún. VLAN-okat használni. Az egy VLAN-ba tartozó gépek egymással tudnak kommunikálni, de a különböző VLAN-ok közötti kapcsolathoz már útválasztó szükséges. Az egyes VLAN-okat ún. VLAN-id azonosítja, és valamilyen már létező interfészből csinálunk VLAN-interfészt. Miután létrehoztuk ezt az interfészt, a folytatásban a hálózat

további konfigurálása során a valódi interfész (pl. eth0) helyett erre a VLAN-interfészre kell hivatkozni.

VLAN-ba konfigurálás javasolt módja:

```
ip link add link dummy0 name dummy0.42 type vlan id 42
```

(A valóságban nyilván nem a dummy0-ból, hanem pl. az eth0-ból, azaz a link dummy0 helyett link eth0 kell szerepeljen.) A végeredmény az ip parancs kimenetében dummy0.42@dummy0 -ként látszik de hivatkozni dummy0.42-ként kell rá (az ifconfig kimenetében dummy0.42-ként is látszik). Ha pedig kihagyjuk a „name dummy0.42" paramétert, akkor vlan0@dummy0 (azaz vlan0) lesz a neve (és ugyanúgy nem kell a hivatkozásnál a @dummy0 része az interfész nevének).

VLAN-interfész jellemzőinek lekérdezésére az

```
ip -d link show dummy0.42
```

parancs javasolható, a „-d" kell a VLAN-id megjelenítéséhez (ez nyilván akkor fontos, ha nem a fenti módon nevezzük el az interfészt). Megszüntetni pedig a szokásos:

```
ip link del dev  dummy0.42
```

paranccsal kell.

VLAN elavult módszerrel konfigurálása pedig egy ún. vconfig nevú paranccsal történik.

```
vconfig add dummy0 42
```

Az eredmény a dummy0.42 interfész. Megszüntetni ugyanezen parancs „rem" (azaz remove) paraméterével lehet:

```
vconfig rem dummy0.42
```

Mivel a vconfig nem tesz lehetőséget a VLAN-interfész nevének kiválasztására, hanem gyárilag ilyen neveket használ, remélhetőleg így már érthető, hogy miért ugyanazt a konvenciót követtük mi is az ip parancs használatakor. (Természetesen VLAN esetén is felcserélhető egymással a javasolt, illetve az elavult módszer.)

BOND / Link Aggregate / Fast Etherchannel

Hálózati terhelés elosztásra, vagy épp redundancia előállítására szokták elsősorban használni az ún. BOND, vagy aggregált interfészeket. Ilyenkor több valódi interfészt kapcsolunk össze. Az aggregált interfészek használatához (hasonlóan a dummy interfészhez) egy kernel modul betöltésére van szükség. A modul betöltésekor adhatjuk meg, hogy a sokféle (szabványos és nem annyira szabványos) bond-módból melyiket akarjuk használni[120]:

```
modprobe bonding mode=1 miimon=500
```

A mode= paraméter állításával szabályozható, hogy pontosan milyen módon működő aggragált interfészt akarunk létrehozni, pl a mode=0 az ugynevezett balance-rr – azaz Round Robin elven forgalmaz az így létejött virtuális interfészen. Fenti parancs eredményeként létrejön egy active-backup módú, 500 ms timeout-ot használó bond0 interfész (azaz ameddig az elsődleges – primary – interfész él, addig azon forgalmaz, ha az megszűnik működni, akkor fél másodpercen belül észreveszi, és átvált a második – secondary – interfészre; ha az elsődleges visszatér, akkor visszakapcsol annak a haszálatára). (A bond0 interfész az ún. master szerepet játssza.)

Ha tehát készen van a bond0, akkor a konfigurálás:

```
ip link set dev dummy0 master bond0
```

120. részletesebb leírásért érdemes elolvasni pl. a Linux Ethernet Bonding Driver HOWTO nevű dokumentumot

```
ip link set dev dummy1 master bond0
ip add addr dev bond0 1.2.3.4/32
ip link set dev bond0 up
```

(Ehhez természetesen kell egy másik, dummy1 nevű interfész is.)

Kivenni valamely alárendelt (slave) interfészt:

```
ip link set dev dummy1 nomaster
```

paranccsal lehet, illetve az egész interfész eltávolításához mindegyik slave-et szedjük ki, konfiguráljuk le, majd dobjuk el az interfészt:

```
ip link set dev dummy0 nomaster
ip link set dev bond0 down
ip link del dev bond0
```

A létrejött bond0 interfész a továbbiakban konfigurálható:

```
ip addr add 1.2.3.4 dev bond0
```

Az elavultnak minősített módszerhez egy ún. ifenslave nevű parancsra van szükség:

```
ifenslave bond0 dummy0 dummy1
```

Az elsőként megadott slave eszköz lesz a primary. A bonding megszüntetése ugyanezzel a paranccsal:

```
ifenslave -d bond0 dummy1 dummy0
```

Az érdekesség kedvéért megmutatuk egy harmadik eszközt is. (Mint egy linuxos zenebohóc: „van másik”) Ez a módszer, az ún. sysfs fájlrendszeren keresztül piszkálja a bonding eszközt. Létrehozzuk a fő (master eszközt)

```
echo +bond0 > /sys/class/net/bonding_masters
```

Majd konfiguráljuk:

```
ip link set dev bond0 up
echo +dummy0 /sys/class/net/bond0/bonding/slaves
echo +dummy1 /sys/class/net/bond0/bonding/slaves
```

Eltávolítani egy slave eszközt:

```
echo -dummy1 /sys/class/net/bond0/bonding/slaves
```

Természetesen az egészet itt is úgy kell, hogy szépen lépésenként lebontjuk:

```
echo -dummy0 /sys/class/net/bond0/bonding/slaves
echo -bond0 /sys/class/net/bonding_masters
```

Ethernet Bridge

Időnként szükség lehet ún. Ethernet bridge létrehozására. A javasolt módszer:

```
ip link add name br0 type bridge
ip link set dev dummy0 master br0
ip link set dev dummy1 master br1
```

Ebben az esetben a bridge típusú virtuális interfész lesz a master, és a bridge-t alkotó fizikai interfészek a slave interfészek. Kivenni egy interfészt, illetve megszüntetni magát a bridge-et ugyanúgy kell, mint a bonding interfésznél kellett:

```
ip link set dev dummy1 nomaster
ip link set dev dummy0 nomaster
ip link del dev br0
```

Ehhez is tartozik elavult módszer, a parancs neve brctl. A fenti br0 létrehozása:

```
brctl addbr br0
brctl addif br0 dummy0
brctl addif br0 dummy1
```

Jellemzőinek lekérdezése:

```
brctl show br0
```

Megszüntetése pedig:

```
brctl delif br0 dummy1
brctl delif br0 dummy0
brctl delbr br0
```

TUN/TAP virtuális interfész

Főleg különböző VPN, és egyéb tunneling megoldások használják az ún. tun/tap interfészeket. (Pl. az OpenVPN, de akár a jól ismert OpenSSH is tud teljes tunnelinget.) Az első ún. virtuális Ethernet, míg a második virtuális IP interfész, mindkét esetben pont-pont kapcsolathoz. A konfiguráláshoz javasolt módszer itt is az ip parancs. (Hátulütője a dolognak, hogy nem minden terjesztésben van annyira új iproute-csomag, ami ismeri ezt a funkciót.) Létrehozni a virtuális interfészt:

```
ip tuntap add dev tun0 mode tun
ip tuntap add dev tap0 mode tap
```

paranccsal lehet. Megszüntetni pedig vagy

```
ip tuntap del dev tun0 mode tun
```

paranccsal. Nehezítő tényező ennél formánál, hogy ha lekérdezzük az interfészek részletes információit:

```
ip -d l sh dev tun0
15: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN qlen 500
  link/none
  tun
ip -d l sh dev tap0
14: tap0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 500
  link/ether 4e:c4:3d:28:5a:ee brd ff:ff:ff:ff:ff:ff
  tun
```

akkor az ip parancs mindkét interfészt tun módúnak mutatja, viszont ha ezek alapján próbáljuk törölni, akkor hibaüzenetet kapunk

```
ip tuntap del dev tap0 mode tun
ioctl(TUNSETIFF): Invalid argument
```

Két lehetőségünk marad. Vagy észrevesszük, hogy a tap0 interfésznél (ez ugye a virtuális Ethernet) látszik MAC-cím – ott a link/ether sor, míg a tun0-nál (ez pedig az IP-interfész) e helyett link/none szerepel. Ha ezt meg tudjuk jegyezni, akkor már nem is kell a -d opció. Ha ezt nem tudjuk megjegyezni, vagy eleve szeretjük a könnyebb megoldást, akkor nem az erre felkészített ip tuntap del parancsot használjuk, hanem a hagyományos interfész-törlő módszert:

```
ip link del dev tap0
```

Ugyanis ilyenkor nem foglalkozik a rendszer a mód paraméterrel.

Ha rendszerünkben az iproute csomag nem eléggé új, akkor sajnos valamilyen extra csomagot kell letöltenünk, amelyben esetleg találhatunk eszközt a tun/tap interfész létrehozásához. Talán leginkább az OpenVPN csomag javasolható (eddig még csak komolyabb méretű, több függőséggel rendelkező olyan eszközt találtunk, amiben van valamilyen segédprogram ehhez a funkcióhoz). Az OpenVPN telepítése után az interfészek kezelése. Létrehozás:

```
openvpn --mktun --dev tun0
openvpn --mktun --dev tap0
```

Törlés:

```
openvpn --rmtun --dev tun0
openvpn --rmtun --dev tap0
```

(Bár az ip link del … parancs ekkor ishasználható.) Mint sejthető, a megadott név függvényében jön létre TUN, illetve TAP típusú interfészünk.

Hasonlóan az alap hálózati részben megfogalmazotthoz, most is érvényes, hogy ezen parancsok hatása legkésőbb a gép újraindulásakor elveszik, azaz az ott leírtak most is érvényesek. Ha szükséges ezeket a speciális hálózati eszközöket mindig használni, akkor az abban a fejezetben emlegetett konfigurációs fájlok megfelelő kitöltése a megoldás. (Sajnos néhány terjesztésben ezek némelyikére nincsenek eszközök, azaz rossz esetben marad valamilyen, a fenti parancsokat tartalmazó, kézzel létrehozott parancsfájlnak a rendszerindításba történő beerőszakolása.)

Sávszélesség szabályozás

Sávszélesség. A Wikipédia szerint: egy digitális kapcsolaton adott idő alatt átvihető adat mennyisége. Gyakran bitsebességnek is nevezik. Mértékegysége a bit/sec vagy byte/sec, ami az másodpercenként átvihető bitek vagy byte-ok számát adja meg. Napjainkban a hálózatok sebességének növekedése miatt a kbit/sec vagy inkább a Mbit/sec az elterjedt. Sávszélesség. Ma, a hálózatok korában már majd mindenkinek van, de senkinek nincs elég (minden sávszélességet ki lehet tölteni, mint ahogy tárhelyből sincs soha túl sok). Akinek kevés van, az jól szeretne gazdálkodni vele, akinek sok van, az szeretné okosan elosztani. Hogyan lehet ezt egyszerűen megoldani? Erről szól ez a fejezet.

Hálózati kapcsolatok

Kezdetben volt a telefonos hálózat. Akkoriban a kiválasztottak modemeket használtak, amelyet az egyszerű népek csak csodálhattak. Ebben az időben a sebesség még csak 2400 bit/sec, de ez elsősorban a telefonrendszerek rossz minőségének volt köszönhető. Kalandos idők voltak. Ahogy javultak a vonalak, a modemekkel akár 56\,kbit/sec sebességet is el lehetett érni -- ilyen kapcsolatokat még ma is lehet találni ott, ahol a telefonközpontok vagy a kábeltévé hálózatok nincsenek megfelelően kiépítve. A modemek le- és feltöltési sebessége általában megegyezik (az 56k-s modem esetén 33.6k a feltöltési sebesség), a modem típusától függően vagy párhuzamos a fel- és a letöltés (full-duplex), akár a beszéd a telefonvonalon, vagy egyszerre csak az egyik fél küldhet adatot (half-duplex), mint a CB rádió esetében. Előkerült egy nagyon fontos fogalompár: a fel- és letöltés. Később meglátjuk, hogy ezek a fogalmak nagyon fontosak a további történetünk szempontjából, ezért beszéljünk egy kicsit róluk.

Valamiért az kép alakult ki az emberek fejében, hogy a kliensszámítógép fizikailag és virtuálisan alacsonyabb szinten helyezkedik el, mint a szerver. Ezért letöltésnek hívják, amikor a szerver adatot küld a kliensnek (például egy munkaállomásnak), és feltöltésnek ha a kliens továbbít adatot a szervernek. Gyakran hallhatjuk például, hogy letöltöttem egy remek albumot az internetről, vagy hogy feltöltöttem az új weblapomat a szerverre. A sávszélesség-menedzsment szempontjából a két irány egészen más tulajdonságokkal bír, de erről majd később.

Az analóg modemet szerencsére lassan már mindenhol le lehet vinni a pincébe, mert végre elérkezett a digitális kor. Nálunk a modem után a leginkább elterjedt technológia az ISDN (Integrated Services Digital Network), amely lényegesen nagyobb sávszélességet tesz lehetővé (kb. 128,kbit/s, ha mind a két csatornát használják), de a telefonszámlája miatt anyagilag nem túl barátságos, ezért nem is szeretjük. Jelenleg leginkább az ADSL, WLAN és kábelnet megoldásokat kedvelik a felhasználók. Az ADSL (Asymmetric Digital Subsciber Line) ma már nagyon sok helyen hozzáférhető (igaz, főleg a városokban), a sávszélessége nagyon jó: jelenleg nálunk legfeljebb 4 Mbit/s le, 512,kbit/s fel. Itt már látszik, hogy miért emeltük ki korábban a le és feltöltés különbségét. De ez csak a kezdet. Az egyszerű emberek, akik nem üzemeltetnek webszervert az otthoni számítógépükön, lényegesen többet töltenek le, mint fel, ezért ez a megoldás nekik tökéletesen megfelel. Annyi gond van vele, hogy ha a felfelé irányuló csatorna bedugul, akkor bizony lefelé se nagyon tudnak haladni az adatok. Ennek okáról később, a protokollok tárgyalásánál írunk.

A WLAN és a kábelnet technikailag nagy sávszélességű, szimmetrikus adatátvitelt tesz lehetővé, de általában itt is korlátozzák a feltöltési sebességet. Ennek az az oka, hogy a szolgáltatók nem vesznek felesleges sávszélességet, ezért nem szeretnék, ha a felhasználók otthon hobbiszervereket üzemeltetnének.

Szinte minden kapcsolatfajtánál előfordulhat, hogy ha bizonyos forgalmak túlzottan rátelepszenek a sávszélességre, akkor mások nem tudnak kényelmesen működni. Gyakori eset, mikor néhány nagyobb letöltés (mert kijött az új Debian stabil, és azonnal le kell tölteni a CD-ket) annyira leterheli a vonalat, hogy az internetes telefon nem működik megfelelően. Itt válasszuk szét a hálózati forgalmat két típusra. Az interaktív forgalom általában kicsi, de azonnal továbbítani kell. Ilyenek a már említett IP telefon és videó, az interaktív távoli hozzáférési módszerek (telnet, ssh, vnc...), a csevegő alkalmazások (IRC, webchat, ICQ stb.) és végül is ide sorolható a kisebb weboldalak átvitele is. Ez esetekben nagyon fontos az adatok azonnali átvitele, mert egyébként szaggat a hang, nem jönnek le a lapok és az nem jó. A hálózati forgalmak másik típusa a tömeges adatok átvitele (bulk transfer), ahol az azonnaliság nem annyira fontos szempont. Ilyenek a nagyobb letöltések (web és ftp egyaránt), a levelek küldése és fogadása, rendszerfrissítés és így tovább. Meg kell ismerkednünk két új fogalommal: ezek a throughput és a látencia (latency). A throughput a vonal átviteli sebessége, a látencia pedig a vonal késleltetése.

Az interaktív forgalmakat érdemes lenne egy külön, kicsi VIP csatornán átvinni, ahol a késleltetést igyekeznénk minimalizálni, míg a masszív adatátvitelnél olyan csatorna kellene, ahol az átvitelt maximalizálnánk. De nekünk csak egy vonalunk van, az is bizonyos korlátokkal rendelkezik technológiai sajátosságai miatt. Hogyan lehet mégis használható kompromisszumos megoldást kialakítani? Lássuk!

Protokollok

A továbbiak megértéséhez sajnos le kell szállnunk bit szintre, és meg kell vizsgálnunk az interneten leggyakrabban használt protokollok sajátosságait. Aki viszolyog a hexa editoroktól, annak javasoljuk, hogy a következő bekezdés elolvasása után ugorjon a következő fejezetre.

Vezetői összefoglaló a protokollokról: az IP protokoll adatok átvitelére való számítógépek között, egy forrás- és egy célcím található benne. Ezek az ún. IP címek azonosítják a számítógépeket az interneten. Az IP protokoll nem csinál mást, mint a hátán cipel más protokollokat a címzetteknek és vissza; leggyakrabban TCP és UDP protokollokat szállít. Amik valójában nem is tudják, hogy honnan, hová mennek, csak annyit, hogy ha célhoz értek, akkor melyik szolgáltatásnak vannak címezve. A szolgáltatásokat ezekben a protokollokban a portok címzik meg. Az UDP sajátossága, hogy nagy megbízhatóságú, helyi hálózatokra tervezték (akkoriban az Internet és elődei még igen komoly csomagveszteséggel működtek), és mivel nemigen számítottak csomagvesztésre, ezért az UDP protokoll nem nyugtázza a vételt. A feladó elküldi a csomagot, aztán erősen hisz abban, hogy a másik oldal megkapta. Nincs visszajelzés. Az UDP-t ma már az interneten is használják olyan protokollok hordozójaként (mert ő is csak szállító, mint az IP), ahol nagyon fontos a sebesség, de nem számít, ha egy-egy csomag elvész. Jó példa erre a VoIP, azaz az internetes telefon, azon része, mely a beszédet szállítja. Pompás hibajavító mechanizmusai miatt egy-egy csomag elvesztését a felhasználók nem is hallják meg, a késleltetése viszont igen kellemetlenné teszi a beszélgetést. A TCP olyan mechanizmusokat tartalmaz, melyek kötelezővé teszik a csomagok ellenőrzését és nyugtázását, így a TCP protokollt használó alkalmazások (igen, ő is kizárólag szállít) biztosak lehetnek benne, hogy minden elküldött csomag célhoz ér. Ennek érdekében a TCP szükség esetén újraküld csomagokat, ha a túloldal nem igazolja vissza őket adott időn belül. Ezzel pár-

huzamosan szükség esetén lassítja a csomagok feladási sebességét, hogy ne árassza el a lassabb vevőt. Ez az a tulajdonság, amit a továbbiak megértéséhez meg kellett ismerni. Most szépen elköszönünk a kényelmesebbektől, és leszállunk a protokollok bugyraiba.

Az ICMP és UDP fontosabb jellemzői

A számítógépek kommunikációja nagyon hasonlít az emberi beszédre. A nyelv, amit beszélnek, a protokoll. Sokféle felhasználásra az idők folyamán nagyon sok különböző protokoll alakult ki, minket most leginkább az interneten leginkább használt IP, ICMP, UDP, és különösen a TCP érdekel. Az IP-ről fent már leírtuk, amit most tudni érdemes, térjünk rá az ICMP-re. Az ICMP (Internet Control Message Protocol), ahogy a neve is mutatja, az internetes forgalom irányításában segít. Sok feladata van, minket csak egyetlen része érdekel: a Source Quench üzenet. Ennek feladata, hogy közölje az adóval: a vevő nem képes olyan gyorsan venni az adatokat, ahogy ő küldi. Nézzünk egy egyszerű példát: van egy nagyon nagyon gyors szerver, akitől egy gyenge kliens elkezd letölteni valami nagyot. A szerver elkezdi szórni az adatokat, de szegény gyenge kliens nem tudja olyan gyorsan eltárolni az adatokat, ahogy kapja őket. Ilyen esetben küld vissza a kliens IP stackje egy Source Quench ICMP üzenetet. Erre az adó szépen lecsökkenti a cwnd (lásd TCP protokoll leírás) értékét egy szegmensnyire, és újra kezdődik a slow start. Erről bővebben a TCP protokoll tárgyalásánál. Korábban a kapcsolat útjában lévő router-ek is küldhettek ilyen csomagot, de az aktuális RFC-k szerint már csak a cél küldhet. Az ECN (Explicit Congestion Notification) megjelenése óta a Source Quench jelentősége lassan visszaszorul.

Most ejtsünk szót kicsit bővebben az UDP (User Datagram Protocol) protokollról. Az UDP kapcsolatmentes protokoll, nincs kapcsolatfelépítés és -bontás. Az elküldött csomagok mindenképpen átmennek a csatornán, így a masszív UDP forgalom vevő oldalon nem szabályozható vissza, erre egész egyszerűen nincs semmilyen módszere a vevőnek. Mindez addig teljesen rendben is volt, amíg az interneten csak segédprotokollnak használták az UDP-t. Ilyen felhasználás a domain vagy az ntp. Ma azonban egyre nagyobb teret hódít a VoIP, ahol a csatorna felépítése és a kapcsolat jellemzőinek egyeztetése TCP-n zajlik, de a hang és kép átvitele UDP-n, ezzel komoly fejtörést okozva a sávszélességet szabályozni vágyóknak. Kijelenthető: ha UDP protokollon nagy mennyiségű adattal eltömik a bejövő csatornánkat, akkor szépen elmehetünk teázni, mert annak csak akkor szakad vége, ha a küldő akarja. Van azért egy megoldás, ha nem is túl egyszerű, de erről majd később. Most menjünk rá a nagy vadra: a TCP-re.

A TCP jellemzői

Ma a TCP (Transmission Control Protocol) végzi az internet adattovábbításának az oroszlán részét -- és szerencsére nagyon okos. A TCP kapcsolat alapú protokoll, ügyel a csomagvesztés elkerülésére, a helyes sorrendre és az ismétlődés mentességre.

A protokoll működési mechanizmusai igen összetettek, vaskos kötetek szólnak róla, a sávszélesség szabályzásának szempontjából minket azonban csak néhány dolog érdekel. A TCP egyik legfontosabb fogalma az ablak, mely kettős célt szolgál. Az egyik feladata az újraadás szabályozása, ez minket most kevésbé érdekel, a másik viszont némi forgalom szabályzás: az ablak a csatornára egyszerre kiküldhető még nem nyugtázott csomagok méretét adja meg. Ha a küldő minden csomagra egyenként megvárná a nyugtát, az adatátvitel nagyon lassú lenne. Az adó ablakában már elküldött, de még nyugtára váró, valamint még el nem küldött csomagok vannak. Amint a vevő nyugtáz egy csomagot, azt az adó kiveszi az ablakból, ahová ezután újabb küldendő adatok kerülnek. Ha a vevő nem növeli megfelelő sebességgel az ablakot, akkor csökkentheti méretét. Ezzel a kliens szabályozhatja az adási sebességet, hiszen ha az ablak méretét nullára csökkenti, akkor az

adó csak akkor adhat újabb csomagot, ha nyugtát kapott az előzőről. Így tehát megvalósítható a nagy álom: a bejövő kapcsolatok sávszélessége szabályozható. Ezzel csak annyi gond van, hogy a hálózaton más rendszerek is vannak, így ez néha nem elegendő.

A kommunikációban nagyon fontos timeout-ok megadásakor fontos szerepe van az un. RTT-nek (round trip time), azaz a vonal késleltetésének. A TCP stack-ek ezt folyamatosan mérik a kapcsolat alatt. Amennyiben pl. egy nyugtát késleltetünk, úgy befolyásoljuk az RTT-t is.

A TCP protokoll egy hasznos eszközt ad a kommunikáció hálózati terheléshez igazításához: ez a slow start algoritmus. Az algoritmus lényege, hogy a normál ablak mellett bevezeti a torlódási ablak (congestion window) fogalmát, Az adási jog a normál és a torlódási ablak méretének minimumától függ. A kapcsolat felépítése után a cwnd értéke egy lesz, amit minden csomag veszteségmentes átvitele után megdupláz az IP stack. Az exponenciális növekedés hatására a kapcsolat gyorsan eléri a maximális sebességét, így a slow start hatására a két rendszer kommunikációja lassan indul, és a vevő, illetve a csatorna lehetőségei szerinti szinten megállítja az adási sebességet. Ez azonban még mindig kevés a torlódás elkerüléséhez, ezért a slow start algoritmust kiegészíti a torlódás elkerülési (congestion avoidance) algoritmus. Az algoritmus lényege, hogy a mai nagy megbízhatóságú hálózatokon a csomagok elvesztése egészen biztosan a torlódás jele lesz, így az algoritmus bevezet egy ssthresh nevű változót, amely két lehetséges működési mód közti váltásra szolgál. Amennyiben az ssthresh kisebb, mint a cwnd, akkor slow start fog indulni, ellenkező esetben a nyugták hatására a cwnd-t csak 1/cwnd értékkel fogja növelni. Amikor újraadásra kerül sor, akkor az ssthresh változót beállítja az aktuálisan hasznát ablak (a cwdn és a normál ablak minimuma) felére. Ha timeout történt, akkor a cwnd-t visszaállítja egyre, és újra kezdődik a slow start. Eddig a csomag eldobásával és az ablak méretének csökkentésével jelezhette a vevő az adónak, hogy lassítson. Újabban van azonban erre jobb mód is: az explicit congestion notification.

A tcp újraadás bemutatásakor láthattuk, hogy miként kezelhető a torlódás. Azonban több protokoll is létezik, ahol a csomagvesztés miatt megnövő késleltetés nagyon zavaró lehet. Ezt a hátrányt könnyen meg lehetne szüntetni, ha a router-ek túl hosszú adási sor esetén nem egyszerűen eldobnák a csomagot, hanem torlódási jelzést tudnának küldeni. E célra dolgozták ki az ECN-t, melynek megvalósítása nem csak a TCP protokollhoz kötött, bármilyen protokoll kiegészíthető vele. A protokoll egy része az IP, míg másik része a TCP rétegben helyezkedik el. Az IP rétegben a kommunikáció az IP fejléc két bitjén át zajlik, erre a diffserv mező melletti két, eddig nem használt bitjét jelölték ki. Az előző ECN leírás itt két külön bitről szólt, a végleges szabvány azonban a két bitet egyben kezeli, az így kapott értéket ecn codepoint-nak hívják. A 00 érték azt mutatja, hogy a csomag feladója nem ismeri az ECN-t, vagy nem akarja azt használni. A 01 az ECT(1), míg az 10 az ECT(0) codepoint. Az 11 codepoint jelentése CE (Congestion Experienced). Amennyiben egy router olyan csomagot vesz, amely az ECT(0) vagy ECT(1) codepoint-ot tartalmazza és torlódást érzékel, úgy átállítja a CE codepoint-ra.

Azonban így csak a túloldalt sikerült értesíteni, ezt az infót még vissza kell juttatni megbízhatóan a feladónak. Mindez azért fontos, hogy az újraadási policy-t ennek megfelelően szabályozhassák. A szabályzás elősegítésére a TCP fejlécben bevezettek két új flag-et: ECE (ECN Echo), valamint a CWR (Congestion Window Reduced). Amikor valamely oldal olyan TCP csomagot vesz, amely CE codepoint-ot tartalmaz az IP fejlécben, úgy a nyugtában beállítja az ECE TCP flaget. A túloldal ezt olyan módon nyugtázhatja, hogy a következő csomagjában a CWR TCP flag szerepel. Az ECN jelzések csak olyan kapcsolatoknál használhatóak, ahol már az elején megállapodtak az ECN használatáról. Ez úgy történik, hogy a kapcsolatot felépítő oldal a SYN-es csomagjában az ECE és a CWR bitet is beállítja (ECN-setup SYN packet). Amennyiben a túloldal

szintén ismeri és használni szeretné az ECN-t, úgy a ECE bitet magasra, a CWR bitet pedig alacsonyra állítja (ezt hívjuk ECN-setup SYN-ACK packet-nek).

A legfontosabb hálózati protokollok sávszélesség jellemzői

HTTP, HTTPS	Az interneten leggyakrabban használt protokollok, a webszerverek és kliensek közti kommunikációra és bizonyos esetekben más forgalmakra is (pl. bizonyos esetekben a Skype is használja). TCP szállítja, így kifelé és befelé is jól kontrollálható. Jellemzően kis kifelé és a felhasználástól függően akár hatalmas befelé irányuló forgalmat okoz. Ha web alapú levelező rendszereken csatolt állományokat töltünk fel, vagy valamilyen jelentősebb méretű form-on megnyomjuk a megfelelő gombot, akkor a kifelé irányuló adatmennyiség is jelentős lehet. Általában elegendő a bejövő irányt szabályozni.
VoIP protokollok	Napjaink egyre népszerűbb lehetősége az internet telefon. Az interneten hang és képátvitelre használt protokollok (pl. H.323, SIP stb.) több csatornát használnak, TCP-t és UDP-t egyaránt. Forgalmuk szabályozása nehézkes, bizonyos esetekben (bejövő szabályzás) lehetetlen. A bejövő forgalom szabályzására az egyetlen használható módszer egy külső router használata az interneten, ahol a befelé irányuló VoIP UDP forgalmat kifelé menőként, queue-kkal lehet szabályozni.
SMTP	Levél küldésre használt, TCP feletti protokoll. Munkaállomáson szinte kizárólag kifelé irányuló forgalmat okoz, levelező szervereken általában befelé is ezt használják. Felhasználástól függően kell szabályozni, az otthoni gépeken csak kifelé.
POP3, POP3S	Levelek letöltésére használják. TCP alapú. Minimális a felfelé forgalma, a lefelé viszont a levelek mennyiségétől függően akár nagyon nagy lehet. A bejövő forgalom szabályozása mindenképpen ajánlott.
IMAP, IMAPS	Postaládák elérésére használható, TCP-n közlekedik. Alapesetben a leveleknek csak a fejlécei jönnek át a csatornán, és a levél törzse csak az elolvasás előtt. Általában ritkán mentünk a helyi leveles ládából az IMAP szerverre, de ilyenkor jelentős felfelé menő forgalmat is okoz. Általában a bejövő forgalmának szabályozása szükséges.
SSH	Távoli hozzáférésre használt, TCP szállítja. Alapvetően parancsok kiadására használható, de lehet rajta keresztül fájlokat másolni (scp, sftp) TCP portokat továbbítani (forwarding) és SOCKS proxyként is használható (-D opció). Így tehát a kifelé és befelé irányuló forgalma is a használat módjától függ, így általában mindkét irányban érdemes szabályozni.
FTP	Adatátvitelre használt TCP alapú protokoll. Általában letöltésre használják, de képes feltöltésre is. Felhasználástól függően mind a két irányt kontrollálni kell. A TC szempontjából igen kellemetlen tulajdonsága, hogy több TCP kapcsolatot használ, így szűrése csak az Netfilter ftp conntrack moduljával oldható meg kényelmesen. Ennek használatáról majd később szólunk.
egyéb, kicsi de fontos forgal-	Sávszélesség szempontjából nem jelentős, de kellemetlen problémákat okozhat a domain és az ntp forgalom fennakadása. Mindkettő jellemzően

mak	UDP-t használ (a domain esetén zónatranszferre használnak TCP-t, de ezt a névszerver üzemeltetők nyilván tudják). Ezeknek a forgalmaknak érdemes egy kicsi de fix csatornát biztosítani a fennakadás mentes működés miatt.
egyéb	Ha a hálózaton egyéb forgalmak vannak, akkor azokat egyenként meg kell vizsgálni. Erre nagyszerűen használható például az ntop program.

A sávszélesség-menedzsmentről

A hálózati kapcsolat két "csőként" is elképzelhető, az egyik a kimenő forgalmat viszi, a másik a bejövőt. Világos, hogy a kimenő csőbe olyan sorrendben és olyan logika szerint küldjük az adatokat, ahogy akarjuk. Ha a kimenő sávszélesség szabályzását szeretnénk megoldani, akkor erre minden lehetőségünk megvan (a Linux biztosítja ezeket). A bejövő csőnél viszont könnyen belátható, hogy közvetlenül csak akkor befolyásolhatjuk a bejövő adatok protokollok és gépek közti elosztását, ha a cső másik végén állítani tudjuk a csőbe továbbított adatokat. Erre általában nincs lehetőség, az internet szolgáltatók csak igen különleges esetekben működnek közre ilyen megoldás felállításában. Ez tehát kiesik. Szerencsére azonban az internet forgalmának jelentős része TCP protokollon zajlik, és – ahogy az előző fejezetben láthattuk – ennek a bejövő sebességét indirekt módon befolyásolhatjuk.

A forgalom szabályzásának módszerei

shaping	A csomagok várakoztatása az elvárt átlagos sebesség eléréséig. Kizárólag kimenő forgalomra használható, mivel a bejövő csomagokat nem tudjuk a csatornán várakoztatni, azok bizony bejönnek.
scheduling	Az időzítés (vagy más néven újrarendezés (reordering)) lehetővé teszi a csomagok átadási sorrendjének megváltoztatását.
policing	Valamilyen akció végrehajtása minden csomagon, amely túllépi az elvárt sávszélességet. Általában ez a csomag eldobását jelenti, ami TCP esetén meglepően hatásos módszer, mint azt korábban láttuk. UDP esetén általában semmi másra nem jó, csak csomagveszés előidézésére.

A kimenő forgalom szabályzására használható a shaping, ahol a kimenő csomagokat a hálózati TC alrendszer valamilyen szabályrendszer alapján várakoztatja a szükséges ideig. Ehhez sorokat használ, ahová a beérkező csomagokat besorolja. A bejövő forgalom jelenleg csak policing módszerrel szabályozható, ahol jelenleg nincs lehetőség osztály alapú sorba állítási módszerek használatára, bár némi kiegészítéssel vagy trükközéssel a bejövő forgalomra is használhatunk osztály alapú sorba állítást, de erről majd később.

A Linux TC alapjai

Arról már szóltunk, hogy sorok segítségével befolyásolhatjuk a csomagok kiküldését. A kernel a rendelkezésünkre bocsát számos különböző képességekkel rendelkező besorolási algoritmust, az un. qdisc-eket, a kimenő (egress) és a bejövő (ingress) forgalom szabályozásához. Ezek segítségével tudjuk módosítani az adatok továbbításának módját. Két csoportba sorolhatók. Vannak osztály alapú (classful -- CF) és osztálymentes (classless -- CL) besoroló módszerek. Az osztálymentes

módszerek képesek a csomagok fogadására, újraütemezésére, várakoztatására vagy eldobására. Az osztály alapú besoroló módszerek több osztályt tartalmazhatnak, melyek újabb besorolási módszereket tartalmazhatnak (CL vagy CF) és így tovább. Az osztályok származhatnak (parent) egy qdisc-ből vagy más osztályokból. Levél osztály alatt olyan osztályokat értünk, melyeknek nincs alosztálya (child), és rendelkezik egy qdisc-kel. Mikor létrehozunk egy osztály, akkor alapból egy fifo qdisc kapcsolódik hozzá (lásd később), mely gyerekosztályok hozzákapcsolásánál automatikusan megszűnik. A levélosztályok qdisc-jeit bármikor le lehet cserélni CL vagy CF qdisc-re.

Az osztályoknak szükségük van egy algoritmusra, amely megadja, hogy hogyan kell a beérkező csomagokat elosztani az alosztályok között, ez az osztályzó (classifier). Az osztályzás szűrők (filter) alapján végezhető el. Egy szűrő valamilyen feltételek szerint eldönti, hogy egy adott csomag hozzá tartozik-e vagy nem.

A bal oldalon látható a hálózatról beérkező forgalom, amely egy speciális qdisc-be érkezik, ez az ingress qdisc. Itt végezhető el a beérkező csomagok kezelése. Ehhez szűrők köthetők, melyek lehetőséget adnak a bejövő csomagok eldobására, ezzel a TCP forgalom lassítására. Ha a csomag nem dobódott el, akkor ezek után következik a routing döntés, ahol a kernel eldönti, hogy a csomag helyi folyamathoz tartozik (ekkor az megkapja), vagy tovább kell küldeni egy másik csatolón. Utóbbi esetben a kimenő hálózati interfészhez kapcsolt qdisc folytatja a csomag feldolgozását és a beállításainak megfelelően kezeli azt.

Minden csatolóhoz tartozik egy kimenő (egress) un. root qdisc. Alapesetben ebbe "esnek be" a csomagok, hacsak valamelyik osztályzó nem sorolja őket máshová. Minden qdisc-nek van egy azonosítója (handle), amely két számból, a major-ből és a minor-ból áll. A major-t a felhasználó adja meg, a qdisk-ek esetén minor minden esetben 0. Egy root qdisc esetén az azonosító valahogy így néz ki: 1:0. Az osztályok major-jainak meg kell egyeznie a szülőosztályok major-jával. A major-nak egyedinek kell lennie az ingress-en és az egress-en is. Az osztályok logikai származási fájára mutat egy példát az ábra.

A fa nem a besorolást segíti elő, hanem, ahogy később látni fogjuk, a segítségével a csatorna egyszerűen és logikusan feloszthatő a különböző osztályok között. Ezt úgy oldják meg, hogy minden osztály csak a felette állóval beszélhet, csak az ő sávszélességéből gazdálkodhat. Az osztályba sorolást a szűrők végzik el, minden osztályhoz kapcsolhatók szűrők, és segítségükkel minden csomag átdobható egy másik osztályba.

A Linux TC kernel részei

Ennyi bevezető után ideje lenne rátérni a lényegre. Ha befolyásolni szeretnénk a sávszélességet, akkor szükségünk lesz a kernel támogatására. Az alábbiakat kell a kernelbe fordítanunk (csak a most használt részeket említve):

```
Networking ->
    [*] Networking support
    Networking options  --->
  <M> Firewall based classifier
  <M> U32 classifier
  [*] QoS and/or fair queueing  --->
      <M>   HTB packet scheduler
      <M>   The simplest PRIO pseudoscheduler
      <M>   SFQ queue
      <M>   TBF queue
      <M>   Ingress Qdisc
```

A fenti alrendszereket a komolyabb disztribúciók befordítják a rendszermagba, így ezzel általában nem kell bajlódnunk. A Linux TC beállításához szükségünk lesz még a tc parancsra is, amely általában az iproute2 csomag része.

Besorolási módszerek (qdisc)

Az írás mérete nem teszi lehetővé, hogy minden létező módszert bemutassunk, így egyszerűen csak egy könnyen használható, világos megoldás kivitelezéséhez szükséges részeket ismertetjük. Amennyiben valakinek ettől bonyolultabb igényei vannak, akkor úgysem ússza meg a LARTC elolvasását.

Osztálymentes qdisc-ek (classless)

pfifo_fast

A legegyszerűbb, minden hálózati csatolóhoz alapértelmezetten kapcsolt qdisc. A forgalmat három sávra (band) osztja, minden sáv egy FIFO, ami azt jelenti, hogy az elsőként betett csomag hagyja el először a sávot. A forgalmak sorokba osztását az IP csomag TOS és PRECEDENCE mezői alapján oldja meg, például a Minimum Delay jelzővel ellátott csomag a 0. sorba kerül, és így tovább. A javasolt TOS besorolásokat az RFC-1349 írja le. A sávok egymáshoz képesti kezelése úgy zajlik, hogy amíg a 0. sávon van csomag, addig az 1. sávot nem hagyja el csomag, és így jár el az 1. és 2. viszonylatában is. Ebből azonnal látszik egy kellemetlen mellékhatása: ha egy nagy sávszélességet igénylő forgalom csomagjai rendre Minimum Delay jelzővel vannak ellátva, akkor az lehetetlenné teszi a többi forgalom áthaladását. Ezt a hatást remekül meg lehet figyelni az ssh port továbbítása esetén, ha az átvitt csatornában nagy mennyiségű adatot viszünk át (például egy http port továbbítása esetén). Ilyenkor az ssh képes minden más, jelzővel nem ellátott forgalom elnyomására.

TBF – Token Bucket Filter

A TBF olyan egyszerű, jól használható qdisc, amely lehetővé teszi a rajta átmenő forgalom lassítását. Nagyon precíz, nem igényel sok processzoridőt, valamint hálózatbarát. A hálózati forgalom lassítására kiválóan használható. Csak akkor adja tovább a csomagokat, ha azok továbbításával nem haladja meg az engedélyezett sávszélességet. Az implementáció tartalmaz egy puffert (bucket), amely megadott sebességgel, folyamatosan töltődik virtuális adatdarabkákkal, ún. tokenekkel. A TBF legfontosabb paraméterei a puffer mérete, amely megadja, hogy hány token-t lehet tárolni benne, illetve a korábban említett töltési sebessége. Minden bejövő adatcsomagot egy token-hez kapcsol, amit eltávolít a pufferből. Amennyiben az adatok éppen a megadott sebességgel érkeznek, akkor a pufferben mindig lesz hozzájuk kapcsolható token, ha lassabban, akkor a puffer tele lesz tokenekkel, tehát az adatcsomagok azonnal továbbítódnak. Látható, hogy ha az adatok lassabban érkeznek, akkor kisebb lökések (burst) akár meghaladhatják a beállított bitsebességet, ha azonban az adatok gyorsabban érkeznek, mint a megadott sebesség, akkor a pufferből hamarosan kifogynak a token-ek, így a beérkező csomagok továbbítása átmenetileg szünetel, tehát a sávszélesség a beállított szinten marad. A legfontosabb beállítható jellemzői tehát:

rate	A sebességhatár.
burst	A puffer mérete byte-okban megadva. Vigyázat! Ez megadja egy lehetséges adatlökés maximális méretét. Ha nagyon kicsi, akkor lehetetlenné teszi az adatok továbbítását. A gyakorlat azt mutatja, hogy ha nagyságrendileg nem hasonló a rate értékéhez, akkor a TBF nem tudja a beállított sebességet átvinni. Ennek oka a hálózati forgalom sajátossá-

	gaiban gyökerezik, ugyanis az soha nem egyenletes. Ha a puffer méretét a rate méretére (vagy nagyobbra) állítjuk, akkor a maximális átlagos sebesség biztosan a beállított lesz.
limit	A csomagok sorának mérete byte-okban Nem kötelező meghatározni.
latency	Megadja, hogy egy csomag legfeljebb mennyi ideig lehet a sorban. Nem kötelező meghatározni.

SFQ – Stochastic Fairness Queueing

Az SFQ olyan qdisc, amely igyekszik igazságosan elosztani a hálózatot a beérkező kapcsolatok között. A TCP kapcsolatok és UDP adatfolyamok alapján a beérkező csomagokat nagyszámú sorba osztja szét, és utána minden sorból legfeljebb a quantum által meghatározott mennyiséget továbbítja. Ezzel elérhető, hogy a gyorsabb kapcsolatok nem tudják elnyomni a lassabbakat. Természetesen csak akkor van értelme, ha a hálózati csatoló teljesen ki van használva, mert ha nincs, akkor az SFQ nem csinál semmit, hiszen minden kimenő csomagot azonnal továbbítani lehet. No de kinek nincs állandóan kitömve a kimenő vonala? Az SFQ legfontosabb paraméterei:

perturb	Az elosztási besorolás újrakonfigurálásának ideje. Amennyiben nincs megadva, akkor soha nem lesz újrakonfigurálva, ami nem jó ötlet. A 10 másodperc jó választás.
quantum	A sorból egyszerre kiküldhető adat mennyisége byte-okban Az alapbeállítása egy csomag mérete. Ne állítsuk az MTU mérete fölé!
limit	A sorba állítható csomagok maximális száma. Amennyiben eléri, akkor elkezdi eldobni a csomagokat.

Osztály alapú qdisc-ek (classful)

Itt csak egyetlen, de nagyszerűen használható qdics-ről szólunk. A neve HTB (Hierarchical Token Bucket), és a gyakran használt, de kissé bonyolult CBQ helyettesítésére hozták létre. A HTB tulajdonképpen a TBF osztály alapú változata. Legfontosabb beállítási jellemzői:

rate	Meghatározza a token-ek előállítási sebességét és így az átlagos sávszélesség határt is (lásd még TBF). A root osztálynak a vonal sávszélességét kell megadni.
burst	A puffer mérete (lásd még TBF).
ceil	Az osztály által használható legnagyobb sávszélesség.
prio	Az alacsony várakozási időt igénylő osztályok priorizálására használható, például hang és kép átvitele esetén, vagy távoli shell használatánál. Paramétere egy szám. A magasabb besorolású osztály kap először a sávszélességből (természetesen csak a rate és a ceil által meghatározott mértékig).
default	Az alapértelmezett gyerekosztály azonosító minorja. Hacsak egy szűrő másként nem dönt, akkor ide kerül a csomag.

A root osztályon kívül minden HTB képes sávszélességet kölcsönvenni a testvéreitől. Ám ha közvetlenül a root alá vesszük fel a gyerekosztályokat, akkor azok sem tudnak kölcsönözni, így általában érdemes a root alá felvenni egy osztályt, és a többieket ez alá tenni. Így szükség és lehetőség esetén a levélosztályok már kaphatnak sávszélességet a többiektől.

Az osztályhierarchia megfelelő felépítésével, a ceil és a rate megfelelő megválasztásával szinte minden, gyakorlatban felmerülő feladat megoldható. Ezért használjuk ezt a qdisc-et a példáinkban.

A forgalom osztályozása

A forgalom osztályozására két leggyakrabban használt szűrő a firewall és az u32. A firewall szűrő a csomagszűrő által beállított jelek (mark) alapján osztályozza a csomagokat, ilyen jel helyezhető fel a mangle táblában a -j MARK --set-mark <jel> akció segítségével. A Netfilter hatalmas tudása és rugalmassága ezáltal felhasználható a TC rendszer kialakításánál is. A legjobb az OUTPUT tábla elejére betenni egy jelölő chaint, ami a szükségleteknek megfelelően megjelöli a csomagokat. Érdemes megjegyezni, hogy a bejövő hálózati eszközön is megjelölhetjük a csomagokat, és ez alapján osztályozhatjuk a kimenő oldalon. Ennek felhasználása a szűrőben:

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 handle 10 \
   fw flowid 1:1
```

Ez a 10 jelzéssel ellátott csomagokat az 1:1 azonosítójú osztályhoz rendeli. Az u32 szűrő segítségével közvetlenül a csomag jellemzőire szűrhetünk, és ezek alapján irányíthatjuk a csomagokat. Az u32 legfontosabb paraméterei:

src	Forráscím vagy hálózat. Használata: 'match ip dst 10.1.2.0/24', amely a 10.1.2.0 hálózati című, C osztályú hálózatra illeszkedik. Ha egy bizonyos gépet szeretnénk megadni, akkor a maszk legyen 32.
dst	Célcím vagy hálózat. Használatát lásd a forráscímnél.
sport	Forrásport. Használata: 'match ip dport 110 0xffff', amely a 110-es portról jövő (azaz POP3) csomagokra illeszkedik.
dport	Célport. Használatát lásd a forrásportnál.
protokoll	Protokollra a /etc/protocols fájlban megadott szám segítségével egyszerűen szűrhetünk így: 'match ip protocol 1 0xff'. Ez az ICMP, vagyis az 1 azonosító számú csomagokra illeszkedik.

Tehát egy összetett u32 szűrő valahogy így néz ki:

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match \
   ip src 192.168.1.1 match ip dport 25 0xffff flowid 1:1
```

amely a 192.168.1.1 címre menő, 25-ös portra irányuló forgalmat bedobja az 1:1 azonosítójú osztályba.

A kimenő forgalom szabályozása

Most jöjjön a gyakorlat. A kimenő forgalom szűrését a következő módon állítjuk be:

```
# tc qdisc add dev eth0 root handle 1: htb default 10
# tc class add dev eth0 parent 1: classid 1:1 htb rate 2mbit burst 10k
# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 1536kbit \
   ceil 2mbit burst 10k prio 1
# tc class add dev eth0 parent 1:1 classid 1:20 htb rate 512kbit \
   ceil 512kbit burst 10k prio 2
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
# tc filter add dev eth0 parent 1: protocol ip prio 1 handle 120 \
   fw flowid 1:20
# iptables -t mangle -I OUTPUT -o eth0 -p tcp --sport 25 \
```

```
  -j MARK --set-mark 120
```

Ezek mellett a beállítások mellett a rendszer sávszélességet ketté fogja vágni az 1:10 és az 1:20 azonosítójú HTB osztályra. Az 1:10 osztály maximális sávszélessége 1536kbit/s, és ha a vele egy szinten lévő osztály tud adni, akkor 2 Mbit/s-ig kölcsön kéri annak sávszélességét. Ebbe fog bekerülni minden olyan forgalom, ami nem lesz explicit módon osztályba sorolva. A másik osztály, amelynek azonosítója 1:20 sávszélessége 512kbit/s, és nem kér kölcsön a többiekből. Az osztályzást firewall szűrővel végezzük, a Netfilter jelek alapján. Látható, hogy a kimenő forgalomnak a 25-ös portra menő része (a kimenő SMTP forgalom) lesz 120-as jellel ellátva, amit a firewall szűrő „bedob" az 1:20 osztályba. Az ábra mutatja a hálózat kihasználtságát a beállítás előtt és után. Mindkét levélosztály egy-egy SFQ qdisc-et kapott. A végeredmény: a kimenő levelezés nem tudja elfoglalni a teljes sávszélességet, csak annak egy negyedét.

A bejövő forgalom szabályozása

A betöltött szabályrendszer nagyon egyszerű:

```
# tc qdisc add dev eth0 ingress
# tc filter add dev eth0 parent ffff: protocol ip prio 50 \
  u32 match ip dst 10.1.1.1/32 police rate 512kbit burst 512kbit \
  drop flowid :1
```

Ez létrehoz egy ingress root qdisc-et az eth0 csatolóra, amely valójában nem klasszikus qdisc, csak annyira képes, hogy szabályok alapján eldobjon csomagokat. Egyetlen szabály lesz, ez a 10.1.1.1 gépről jövő forgalmat csökkenti 512kbit/s-ra. Minden olyan csomag, amely ezt a bitsebességet meghaladná, el lesz dobva. A többi bejövő forgalom nem lesz kezelve, így azok tetszés szerint kitölthetik a csatornát. Ahogy az ábrán látszik, ez SSH esetén (és nyilván a többi TCP alapú protokoll esetén is így lenne) szépen működik is. Ha azonban a 10.1.1.1 irányából nagy mennyiségű UDP forgalom érkezik (az nc segítségével állítottuk elő), akkor a szabályzás teljesen tönkremegy.

Ahogy már korábban is említettük, a bejövő forgalomra nem lehet várakozási sorokat használni, mivel a csatornán nem várhatnak a csomagok. Ezt a hiányosságot többféle módon megkerülhetjük. Az egyik lehetőség a virtuális gépek használata, így a bejövő forgalom a gépen belül egy virtuális hálózati csatolón (pl tun eszközön) távozik, és itt már tehetünk hozzá várakozási sorokat tartalmazó qdisc-eket is. A másik módszer jelenleg nem része a hivatalos kernelnek, ez az IMQ. Ez egy virtuális hálózati eszköz, amely használatával a bejövő forgalom kimenőként láttatható és így kezelhető.

Figyelmeztetés: A bejövő forgalom szabályozása csak akkor oldható meg tökéletesen, ha az interneten elhelyezünk egy kis Linux gépet, aki fogadja a befelé irányuló forgalmat (valójában mindenki azt hiszi, hogy az ő IP címe a miénk), és a kimenő csatolóján végzi el a forgalom szabályzását.

Határvédelem, alapvető tűzfal beállítások (netfilter és előtétjei)

A tűzfalakról általában

Tűzfalnak (firewall) nevezzük általánosságban azt a hálózati eszközt, amely célja, hogy az adott hálózathoz vagy számítógéphez csak előre definiált hozzáférést engedélyezzen. A tűzfalak állhatnak speciális szoftver- illetve hardverkomponensekből, és az átlag felhasználó gyakran nem is igazán észleli a létezésüket. Fajtái az egyedfejlődésük során a teljesen kezdetleges határvédelemtől egészen a szofisztikált moduláris applikáció szintű tűzfalig terjedtek, és folyamatosan fejlődnek. Az internet igen korai szakaszában jellemzően nem csak tűzfalakra nem volt szükség, hanem titkosítást sem igen használtak. Azok akik hozzáférhettek a gépekhez és a hálózathoz, mind magasan képzett és jellemzően egy célért a számítógép elé leülő emberek voltak, akik megbíztak a tudás birtokosában. Azt feltételezték, hogy aki a tudást és a fizikai helyet birtokolja, az nem ártó szándékkal tevékenykedik a hálózaton. Ez a fajta hozzáállás nem csak gyökeresen megváltozott, hanem ma már egy átlag üzemeltetőnek muszáj úgy gondolkodnia, hogy amint az asztali vagy a szerver gépet rákötötte a hálózatra, az előbb vagy utóbb célpontja lesz egy betörési kísérletnek, de legalább egy hálózaton keresztüli tapogatózásnak. Számtalan olyan kísérlet dokumentációja olvasható szerte az interneten, ahol egy magányos alaptelepítést tesznek ki védelem nélkül egy publikus címre. Lehet ez akár Linux vagy Windows alapú gép, egy biztos, előbb vagy utóbb felfedezik a véletlenszerű vagy direktben arra a szegmensre indított pásztázások. Onnantól pedig az operációs rendszer komponenseitől, illetve a rajtuk futó szolgáltatásoktól függ, hogy mennyi időn belül „esik el“ a gép, és veszik át rajta az uralmat. Az tehát, hogy a tűzfalra már a legegyszerűbb, hálózatra kötött asztali gépnek és egy magában álló szervernek szüksége van, nem is vitatott kérdés. Jellemzően pedig a legtöbb belső hálózat előtt is legalább egy csomagszűrő router vagy egy célhardver irányítja és szűri a csomagokat.

A tűzfalak fajtái

Csomagszűrő (packet filter, screening router)

Minden egyes csomagról annak fejléce alapján, az előzmények figyelembe vétele nélkül hoz döntéseket. Mivel a csomagok általában fix felépítésű fejléceiben lévő információt használja a döntéseknél, így nincs szüksége bonyolult elemzésekre, ezért nagyon gyors, kicsi processzorigényű. Mivel állapotmentes, így kicsi a memóriaigénye és a kezelhető kapcsolatok számát nem korlátozza semmi. Illetve csak a hálózati eszközök átviteli sebessége.

Állapot mentességéből adódik a hátránya is, ez pedig az, hogy nem tudja megállapítani egy csomagról, hogy egy már fennálló hálózati kapcsolathoz tartozik-e, vagy sem. Ennek következménye, hogy megtéveszthető, még fel sem épült TCP kapcsolatokhoz tartozó csomagok kószálhatnak a hálózaton, és ezt a csomagszűrő nem képes felfedezni. A másik probléma, hogy bizonyos bonyolultabb protokollok esetén, amilyen például az FTP (File Transfer Protokol), nem képes a

több összefüggő hálózati kapcsolatot egyként értelmezni, ezért ezeknek a protokolloknak az átvitele nehézkes és sokszor veszélyesen ki kell nyitni a tűzfalat hozzájuk. A csomagszűrők szinte mindig kernel szinten vannak implementálva. (Lásd folyamatok fejezet.)

Állapottartó csomagszűrő (stateful packet filter, SPF)

Ez is elemzi a csomagok fejléceiben található információkat, de mikor egy TCP kapcsolat felépül, akkor azt feljegyzi egy táblázatba, így mindig tisztában van azzal, hogy egy csomag egy új kapcsolatot kezdeményez, egy már létezőhöz tartozik, vagy egyéb, érvénytelen állapotú.

Az állapottartó csomagszűrő kisebb-nagyobb mértékben elemzi a csomagok adatainak tartalmát is, így például lehetséges, hogy megfelelő elemzőmodul használatával az FTP protokollnál a parancscsatornán egyeztetett adatcsatorna felépítési megállapodást is feljegyzi, így az adatcsatorna felépülésekor tisztában van vele, hogy az adott kapcsolat egy már létező FTP kapcsolat része. Az állapottartó szűrők nem elemzik teljes mélységben a csomagok adattartalmát. A legtöbb ma használatos tűzfal állapottartó, nem is igazán találunk egyszerű csomagszűrőt. Mivel kiváló állapottartó csomagszűrők érhetők el több különböző szabadságú licenc alatt (GPL vagy BSD), így ezek bármely gyártó számára használhatók. Az állapottartó csomagszűrők általában kernel szinten vannak implementálva. (Lásd folyamatok fejezet.)

Alkalmazás szintű tűzfal vagy proxy

Angolul application level firewall, ALF. A hálózati csomagok belsejében lévő, lényegi információk az alkalmazás szintű protokollhoz tartozó adatok. Ilyen protokoll a levelezésnél használt SMTP és IMAPS, a web átviteli protokolljai, a HTTP és a HTTPS és még hosszan sorolhatnánk. Ezeknek az adatoknak a mély elemzésére képesek az alkalmazás szintű tűzfalak. A működésük a következő. Az alkalmazás szintű tűzfalak legtöbbször egy vagy több démonként futnak a rendszeren, a beérkező kapcsolatokat szerver módjára fogadják, majd a kapcsolat eredeti hálózati iránya, vagy ha a protokoll lehetővé teszi, akkor a proxynak szóló információk alapján (pl. HTTP proxy esetén a "Host:" fejléc) kapcsolódnak a valódi szerverhez. Tehát a kliens számára úgy tesznek, mintha ők lennének a szerver, a valódi szerver felé pedig eljátsszák a kliens szerepét.

Mindkét irányból érkező, rajtuk átmenő forgalmat elemeire bontják és lehet, hogy csak megvizsgálják annak részeit, de lehet, hogy szükség esetén akár módosítják is az átmenő adatokat. Ennek mélysége megvalósítás függő. Fontos tudni, hogy a titkosított forgalmak elemzéséhez (mely az esetek döntő többségében SSL vagy valamely szintén aszimmetrikus kulcscserével induló protokoll) a tűzfalnak Man In The Middle (MITM) támadást kell kiviteleznie. Ráadásul ha nem szervert véd, hanem klienseket, akkor a szerverek titkos kulcsa nyilván nem állhat a rendelkezésére, így a kliensek meggyőzése csak úgy lehetséges, ha a szerverek titkos kulcsát hamisítja. Ez úgy lehetséges, ha minden kliensen elhelyeznek megbízhatóként egy olyan CA tanúsítványt, mely a tűzfal üzemeltetőjének birtokában van (pl. céges CA tanúsítványa). Ekkor megfelelően felépített tűzfallal akár on-the-fly készíthetők hamis szerver tanúsítványok. De persze ilyet jóérzésű tűzfal üzemeltetők nem tennének, ugye? Tehát mivel a tűzfal nem tudja visszafejteni a kommunikációt, így a titkosított kapcsolatok általában nem szűrhetők alkalmazás szinten. Ezért valamelyik szabvány SSL porton át felépített OpenVPN, vagy egy SSH kapcsolatba csomagolt port forward vagy SOCKS proxy általában simán átcsúszik a tűzfalon.

Mivel az alkalmazás szűrők teljes egészében elemzik az átmenő forgalmat, sőt, néha a módosítás érdekében az átmenő protokollt szét is kell szerelniük, majd valamely módosításokat végezve visszaszerelni, ezért ezek lényegesen erőforrás igényesebbek (processzor és memória téren is), mint a csomagszűrők. Éppen ezért a nagy sávszélességű hálózatok teljes szűréséhez igen komoly

vasra van szükség a tűzfalakhoz. Az igazán profi tűzfal eszközök támogatják a fürtözést vagy terhelés elosztást a skálázhatóság érdekében.

Hibrid tűzfalak

Az életben legtöbbször hibrid állapottartó-alkalmazás szintű tűzfalakat használnak. Ezeknél az SPF sebessége miatt a hálózati paraméterek miatt elutasított kapcsolatok kezelése minimális erőforrást igényel (akár több tízezer nem engedélyezett kapcsolat felépítő csomag eldobása sem használ szinte semmi processzort). Az alkalmazás szűrő pedig lehetővé teszi a hálózati szinten helyes kapcsolatok teljes mélységű elemzése alapján hozott döntéseket, elvégzett módosításokat.

Egyéb tűzfal funkciók: a NAT

NAT (Network Address Translation, Port Address Translation)

Lényegük, hogy a tűzfalon átmenő kapcsolatban írja át a forrás vagy cél IP-k vagy portok némelyikét az igényeknek megfelelően. Az SNAT (Source NAT) elnevezés a forrás címek és szükség esetén portok módosítását jelenti. Erre akkor van szükség, ha két azonos címtartományú hálózatot kell összekötni. Például egy cég mindkét telephelyén 192.168.1.0/24 C osztályt használnak a korábbi szeparált hálózati állapot miatt. Ezek a saját címtartományukba menő csomagokat nyilván helyben próbálnák elküldeni (lásd hálózati beállítások -> routing). Ha azt akarjuk, hogy tudjanak egymással kommunikálni, akkor kénytelenek vagyunk a két hálózat között egy cím módosító eszközt alkalmazni. Ha például a NAT eszközön a 192.168.1.38 forrású csomagban a forráscím elejét átírom 192.168.2-re, akkor a másik hálózati csatoló felé látható, szintén 192.168.1 kezdetű gépek már nem látják azt sajátnak, a válasz csomagokat a gateway felé fogják küldeni. Az pedig a visszamenő kapcsolatban szépen visszacseréli az (ezúttal már) célcímet 192.168.2-ről 192.168.1-re.

Az SNAT speciális esete, mikor egyetlen címre kell fordítani minden átmenő csomag forrását. Erre kiváló példa egy cég internetes kapcsolata, melynél az internet felé álló lábnak tipikusan csak egyetlen IP címe van, de a cégen belül sok számítógép akar az interneten lévő gépekkel kommunikálni. Ekkor a belső hálózat gépeiről kiinduló kapcsolatokat az internetre csatlakozó SNAT-oló számítógép fogja átalakítani a következőképpen. A csomag forrás címét lecseréli az SNAT gép internetes lábának címére. Ezzel biztosítja, hogy a csomag majd visszatalál erre a gateway-re. Sok kapcsolat esetén van azonban még egy bökkenő. Ez pedig az, hogy a TCP és UDP csomagok forrás portja több számítógép esetén könnyen ütközhet. Ha az aktuálisan módosított forrás portja már foglalt a kimenő oldalon, akkor másik portot kell választani forrás portként. Ezeknek a visszaállítását úgy tudja az SNAT gép megoldani, hogy egy táblázatba feljegyzi, hogy mely forrás IP/port párokat milyen forrás portra cserélt. Így a válaszcsomagokban könnyen be tudja állítani a megfelelő (itt már) cél IP-t és portot.

A DNAT (Destination NAT) egyik formája teljesen hasonló, mint az SNAT első példája. Ezen kívül két speciális alkalmazása lehetséges. Az egyik az, hogy ha egy darab IP címre érkező csomagokat akarunk szétszórni több IP címre (például terhelés kiegyenlítés céljából). A másik lehetséges felhasználás, hogy a gateway nem továbbítja a csomagot, hanem a saját lábának a megfelelő portjára irányítja. Így valósítható meg például az, hogy a kliens gépek beállításainak módosítása nélkül beletereljük a kimenő web forgalmat egy Squid proxy-ba.

Linux alatt a NAT számos formájára képes a Netfilter alrendszer.

Linuxon használható tűzfalak

Az első csomagszűrő tűzfal az 1.1-es kernel változatban jelent meg, a neve a parancssoros eszköze alapján ipfwadm volt. Ezt váltotta az alapjaiban újragondolt és újra is írt ipchains, amely a 2.2-es kernelben jelent meg. Ennek logikája és használata már nagyon hasonlított a mai Linux csomagszűrőhöz. A 2.4-es kernelben megjelent a ma is használt Netfilter (parancssoros kezelő eszköze, az iptables után sokan így nevezik).

A Netfilter projektet Rusty Russell indította 1998-ban az ipchains utódjaként. Russell az ipchains-nek is fejlesztője volt, így annak hiányosságait is figyelembe véve tervezte meg a Netfiltert. Ez egy állapottartó tűzfal, amely támogatja a ma használt fontosabb protokollokat és megoldásokat, nagyon rugalmas és könnyen bővíthető a modulrendszerének köszönhetően. Akár saját modulok is írhatóak hozzá. A Netfilter kernel szinten futó modulokkal van megvalósítva, a felhasználó az iptables/ip6tables (és még pár másik, de ez csak profiknak) parancs segítségével tud szabályokat beállítani.

A Netfilter fejlesztése folyamatos és igen dinamikus. Elmondható, hogy jelenleg az egyik legjobb állapottartó csomagszűrő, számtalan fórum, levelezési lista és HOWTO segíti a felhasználókat az eligazodásban. A rendszer rövid leírása ennek a fejezetnek a további részében megtalálható, ha azonban valaki komolyabban szeretne a rendszerrel megismerkedni, akkor annak különösen ajánlott a Videotorium szemináriuma, ahol Kadlecsik József, a Netfilter magyar fejlesztője – aki 2001 óta a core team tagja –, tart előadást a témában. A Netfilterhez számos olyan GUI készült, amely akár szerveren, akár asztali környezetben leegyszerűsíti a rendszer beállítását.

A teljesség igénye nélkül néhány:

- Fwbuilder
- Turtle Firewall
- Integrated Secure Communications System
- Easy Firewall Generator for IPTables
- Desktop környezethez: Gufw, Firestarter

A Zorp GPL magyar fejlesztésű, moduláris proxy tűzfal. A konfigurációs nyelve Python, ezzel nem csak beállíthatjuk a rendszer konfigurációját, hanem a nyelv lehetőségeit kihasználva rendkívül összetett szűrési feladatokat is megvalósíthatunk. Ugyanakkor meg kell jegyezni, hogy a Python alapú konfiguráció az egyszerű feladatok elvégzésére kissé nehézkes, komoly tudást igényel. A moduláris felépítés lehetővé teszi összetett protokollok elemzését, mint például az SSL-lel védett forgalmak (HTTPS, IMAPS). Sajnos a szabad szoftver verzióban elég kevés protokoll szűrő modul található, így ez valós szituációkban csak részlegesen használható.

A Netfilter keretrendszer alapjai

A Netfilter alapvető működésének megismerése azért szükséges, mert ha nem értjük, hogy mit csinálnak a rendszer beállításait elfedő felhasználói felületek (GUI-k), akkor hiba esetén nem fogjuk érteni, hogy mi a probléma forrása. Egyszerűbb helyzetekben ugyanakkor jobb is a GUI-k

mellőzése, mert egy néhány tíz soros csomagszűrő konfig még kiválóan átlátható és így nem vagyunk kiszolgáltatva a GUI fejlesztők esetleges hibáinak. És komolyan nem bonyolult.

A Netfilter a Linux rendszer általános, IPv4 és IPv6 szűrésére, módosítására is használható moduláris keretrendszer. Legnagyobb egységei a táblák, melyek közül mi a filter és a nat táblákkal fogunk megismerkedni. A filter tábla célja a gépre bejövő, áthaladó és arról kiinduló forgalom szűrésének biztosítása. A nat tábla értelemszerűen elsősorban a különböző NAT funkciók elérésére használható. A táblákban van néhány gyári szűrőlánc, eredeti nevén chain. A filter táblában található gyári szűrőláncok:

INPUT	A gép saját hálózati címeire érkező csomagok szűrő lánca. Ide érkeznek a gép szolgáltatásaira érkező kérések és a gépről kezdeményezett kapcsolatok válaszcsomagjai is.
OUTPUT	A gépről induló csomagok szűrőlánca. Itt jelennek meg a gépről induló kapcsolatok kimenő csomagjainak, illetve a gép szolgáltatásait használó kapcsolatok válaszcsomagjai.
FORWARD	Amennyiben a gépnek több hálózati csatlakozója van, melyek lehetnek fizikai hálózati csatlakozók vagy virtuálisak, akkor az egyik lábon beérkező és a routing tábla szerint egy másik láb felé továbbítandó csomagok ezek a szűrőláncon fognak megjelenni. Virtuális hálózati eszközök például a VLAN vagy VPN végpontok, tun vagy br eszközök, a virtuális gépek hálózati eszközei stb. A legegyszerűbb esetben a FORWARD szűrőt fogjuk használni egy két hálózati eszközzel rendelkező gépnél az áthaladó forgalom szűrésére.

Minden gyári chain-nek van egy alapértelmezett szabálya (default policy), mely csak ACCEPT vagy DROP lehet. ACCEPT esetén azon csomagok továbbításra vagy befogadásra kerülnek, melyekre a láncban szereplő szabályok alapján nem hozott döntést a szűrő rendszer (erről később). DROP esetén ezek a csomagok minden további napló vagy jelzés nélkül eldobásra kerülnek.

Ahogy korábban is elhangzott, a rendszer alapvető beállításait az iptables parancs segítségével lehet elvégezni. További beállításokat végezhetünk még az ip6tables, ebtables, arptables, ipset parancsokkal, de ezekkel most nem fogunk foglalkozni, ha el akarsz mélyülni, akkor olvasd el a kapcsolódó doksikat.

Az iptables parancs alapvetően egyetlen műveletet tud egyszerre végrehajtani. Segítségével képes vagy a chain-ek, vagy azok szabályainak kezelésére. Ezek az elemi műveletek néha jól jönnek, de egy tipikus gyakorlati helyzetben te egy egész csomagszűrő beállítási szabályrendszert akarsz majd betölteni. Erre sokan használnak shell scripteket, de ez nem a legjobb módszer, mert amíg az utolsó szabály is be nem töltődik, a csomagszűrő inkonzisztens állapotban van induláskor. Ha például a legutolsó parancs segítségével állítja be valaki a FORWARD alapértelmezett szabályát, akkor mindaddig szabadon áramlanak a csomagok, amíg oda nem ér a rendszer a shell script végrehajtásában. És egy bonyolult csomagszűrőnél, ha minden szabály egy-egy parancs végrehajtás, akkor ez akár néhány másodperc is lehet. Addig a rendszer nyitva van. Ha nincs más választásunk, mint egy shell script, akkor ügyeljünk rá, hogy először a default policy-t állítsuk be és ha lehet, akkor az első szabály legyen minden láncban egy feltétel nélküli DROP, melyet aztán az összes szabály betöltése után iptables -D-vel eldobunk.

E helyett használható az iptables-restore parancs, mely egy egyszerűen átlátható szöveges fájlból olvassa be a beállításokat és azokat egy elemi műveletként beemeli a kernel memóriájába és csak a végén érvényesíti, az egészet egyben. Mint egy adatbázis tranzakciónál. Itt is commit-nak hívják.

Ha egy épp élő beállítást szeretnél lementeni abban a formátumban, ahogy az iptables-restore értelmezni tudja, akkor az iptables-save parancsot használhatod.

Néhány egyszerű beállítási példa

Az alábbi csomagszűrő beállítások megfelelő védelmet nyújtanak egy asztali gép esetén:

```
# Egyszerű példa desktop iptables konfig
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]

-A INPUT --in-interface lo --jump ACCEPT
-A INPUT --match state --state ESTABLISHED --jump ACCEPT
-A INPUT --match state --state RELATED --jump ACCEPT
-A INPUT --match state --state INVALID --jump LOG --log-prefix "INVALID "
-A INPUT --match state --state INVALID --jump DROP

-A INPUT --jump LOG --log-prefix "catch_all "
-A INPUT --jump DROP
COMMIT
```

A következő pedig épp elegendő lesz egy egyszerű szerverre:

```
# Egyszerű példa szerver iptables konfig
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]

-A INPUT --in-interface lo --jump ACCEPT
-A INPUT --match state --state ESTABLISHED --jump ACCEPT
-A INPUT --match state --state RELATED --jump ACCEPT
-A INPUT --match state --state INVALID --jump LOG --log-prefix "INVALID "
-A INPUT --match state --state INVALID --jump DROP

# admin munkaállomásokról szűrés nélkül minden
-A INPUT --source 10.1.1.56 --jump ACCEPT
-A INPUT --source 10.1.1.111 --jump ACCEPT

# operátor munkaállomásáról megy az ssh
-A INPUT --source 10.1.1.91 --protocol tcp --match tcp --dport ssh --jump ACCEPT

# web szolgáltatások mindenki számára
-A INPUT --protocol tcp --match tcp --dport http --jump ACCEPT
-A INPUT --protocol tcp --match tcp --dport https --jump ACCEPT
```

```
# MariaDB egy megadott hálózatrésznek számára
-A INPUT --source 10.1.2.0/30 --protocol tcp --match tcp --dport mysql --jump
   ACCEPT

-A INPUT --jump LOG --log-prefix "catch_all_"
COMMIT
```

Magas rendelkezésre állás alapok

Az informatikai rendszerek fontos jellemzője a rendelkezésre állás: az idő mekkora részében tudják ellátni a rájuk bízott feladatokat. Egy lakossági szolgáltatásnál elfogadott érték lehet például a 99,5%-os éves rendelkezésre állás, azonban ennek garantálása is erőfeszítéseket igényel a legegyszerűbb szolgáltatás esetén is – két napnyi kiesés egy péntek esti hardverhiba esetén sem fér bele ebbe a szolgáltatási szintbe. Más szegmensekben ennél nagyobb elvárások vannak: gyakran egyszerűen csak a „kilencesek” számát adják meg: egy „öt kilences” rendszer 99,999% rendelkezésre állást vállal. A telekommunikációban nem szokatlan ez az elvárás sem – évi 5 perc kimaradást enged meg. Szervezetek vagy nagyobb szervezeti egységek közötti szolgáltatások esetén az elvárt rendelkezésre állást a szolgáltatásszint-megállapodásban (service level agreement, SLA) rögzítik. Ez a megállapodás általában nem számolja bele a tervezett üzemszüneteket a rendelkezésre állásba, azonban külön megszabja idejüket és gyakoriságukat is.

A rendelkezésre állás növelésére, magas szinten tartására irányuló megoldásokat a HA (high availability, nagy [vagy magas szintű] rendelkezésre állás) betűszóval jelöljük.

Mivel olyan eszköz nincsen, ami garantáltan teljesít egy nagy rendelkezésre állási kritériumot, ezért a megoldások alapja az azonos célt ellátó eszközök többszörözése, redundáns rendszer kialakítása.

> Ha a redundáns eszközök bármikor át tudják venni egymás szerepét, vagy a szerepátvételt vezérlő eszköz jelentősen jobb rendelkezésre állást tud biztosítani az eszközöknél, akkor a két eszköz hibaarányát összeszorozva kapjuk a közös hibaarányt. Például ha két 1% valószínűséggel meghibásodó eszközünk van, amelyek egymástól függetlenül hibásodhatnak meg, és csak egyidejű hibájuk okoz szolgáltatáskiesést, akkor $0{,}01^2 = 0{,}0001 = 0{,}01\%$ közös meghibásodási értéket, vagyis 99,99% működési valószínűséget kapunk. Bár ez az egyszerű modell összemossa a rendszer rendelkezésre állását az eszközök hibátlan működésének valószínűségével, a gyakorlatban jól használható a redundáns eszközök szerepének vizsgálatára. Összehasonlításként: ha mindkét ilyen eszköz szükséges a működéshez, akkor már a rendelkezésre állást szorozhatjuk össze, így a 99%-ból 98% lesz.

A redundancia mértékét mindig az elvárások és a gazdasági lehetőségek alapján kell mérlegelni.

A rendelkezésre állás növelését több irányban kezdhetjük el – azonban a legtöbb rendszernek ezek a módszerek közül többet is kell alkalmaznia a megfelelő szint eléréséhez.

Megbízhatóbb eszközök

Az első lépések általában a hardverhibából eredő szolgáltatáskiesés valószínűségének csökkentését célozzák. Jelentősen csökkentheti a meghibásodás valószínűségét, ha a szolgáltatást folyamatos üzemre tervezett hardverrel nyújtjuk. A leggyakrabban meghibásodó elemek a lemezek és a tápegységek. A lemezek esetében megbízhatóbbak a folyamatos üzemre tervezett változatok, valamint a többportos lemezek ezt támogató vezérlővel.

Mivel egy nagyobb rendszerben elkerülhetetlen, hogy egy-egy lemez időnként tönkremenjen, ezért a RAID konfiguráció minden esetben indokolt lehet. RAID esetén mérlegelnünk kell a szük-

séges teljesítményt és a rendelkezésre álló lemezek számát is (részletekért lásd a Diszkek kezelése című fejezetet).

A redundáns tápegység is jelentősen javíthatja a rendelkezésre állást: nemcsak a tápegység meghibásodása ellen véd, hanem lehetőség van több független feszültségforrás használatára, szünetmentes tápegység cseréjére stb.

Célszerű a hálózati kapcsolatot is redundánsan kialakítani, erre az STP (feszítőfa-protokoll) vagy a link-aggregáció (más neveken: bond, trunk, EtherChannel) jó lehetőség.

Klaszter

Az eszközök megbízhatóságának javításával nem lehet egy szinten túl gazdaságosan növelni a rendelkezésre állást, hiszen az esetleg mégis bekövetkező hiba helyreállítása is időt vesz igénybe. Erre a problémára jelent megoldást a klaszter (cluster, fürt) architektúra.

A megoldás alapja, hogy több azonos célt ellátó rendszer fut szoros hálózati összeköttetésben, amelyek vagy át tudják venni egymás feladatát (HA cluster), vagy elosztják egymás között a terhelést (HPC [high performance computing] cluster). A HPC cluster a legtöbb esetben megköveteli egy, a terheléselosztást végző belépési pont meglétét, ezért önmagában a rendelkezésre állás növelésére nem alkalmas – azonban lehetséges a két megoldás kombinálása is.

A hagyományos HA clustert tekinthetjük melegtartalékos rendszernek is, amelyben a tartalék rendszer állandóan üzemben van, és azonnal képes átvenni a meghibásodott rendszer feladatait. Ennek párja a hidegtartalék, amikor a tartalékgép ugyan elő van készítve a feladat átvételére, azonban kézi beavatkozással el kell indítani.

Heartbeat

A feladatok átvételét a legáltalánosabb esetben úgy oldjuk meg, hogy a tartalék rendszer IP szinten veszi át a pótolt rendszer szerepét: fölveszi IP címét. Ennek a megvalósítása figyelmet igényel, mivel biztosan tudnunk kell, hogy a másik rendszer már nem üzemel (legalábbis azon a címen), hiszen ellenkező esetben címütközést okozunk.

Ennek elterjedt megvalósítása a heartbeat (szívverés) módszer. Ebben az esetben az éles és a tartalék rendszert egy külön erre a célra szolgáló hálózati úton kötjük össze, amelyen az éles rendszer szívverésszerűen jelzi, hogy még működik.

Amennyiben a szívverés megszűnik, a tartalék rendszer körültekintően meg kell állapítsa, hogy a másik gép valóban megállt, és ekkor veheti át a feladatot.

Virtuális IP cím

Az egyes gépek elérhetősége és a visszaállás megkönnyítése érdekében a szolgáltatás igénybevételére egy a gépektől független címet szokás használni, amelyet az éppen aktív gép második címként vesz föl a hálózati interfészén. Ennek esetleges alternatívája az, hogy a tűzfal hálózati címfordítással a megfelelő gépre küldi a kéréseket.

Osztott tár

A feladat átvételéhez általában szükség van arra, hogy a tartalékgép is elérje a szükséges tárterületet. Ennek számos módja van: amennyiben a tárterület SAN vagy NAS eszközön található, a másik gép leállása után nincs akadálya átvételének. Szoftveres megoldások is vannak a tárterület folyamatos elosztott replikálására, ilyen a Ceph vagy a GlusterFS.

Bizonyos esetekben nincs szükség az osztott tárra, mivel a replikációt alkalmazás szinten is meg lehet oldani, vagy nincs is szükség rá. Ilyen lehet egy adatbázis- vagy alkalmazáskiszolgáló, vagy éppen egy tűzfal vagy névkiszolgáló.

Amennyiben van osztott tár, az felhasználható a feladat átadásának visszaigazolására is.

A PAM hitelesítési keretrendszer

Az azonosítási rendszerek gyenge pontja

Az egyes Unix-rendszerek fejlesztői hiába dolgoztak ki közösen, kényelmesen használható megoldást, egy gondot mégsem tudtak orvosolni. Minden program csak az előre tervezett és beépített azonosítási eljárásokat támogatta. Bizonyos programok fejlesztői arra vetemedtek, hogy az általánosan használt azonosítási eljárásokat sem támogatták. Mondanom sem kell, egy új azonosítási eljárás beépítése minden azonosítást igénylő programban nagyon bonyolult és kínos feladat.

Általános megoldás: a PAM rendszer

A fenti feladat megoldására hozták létre a PAM-rendszert (Pluggable Authentication Modules). A PAM ötlete először a Sun fejlesztőinek a fejében született meg. Elsőként részlegesen a Solaris 2.5-ös sorozatában vezették be, a teljes támogatás pedig a Solaris 2.6-ban alakult ki. A PAM finoman hangolható azonosítási rendszer, mely a korábban vázolt gondokra oly módon ad egységes megoldást, hogy az azonosítással kapcsolatos feladatokat elválasztja a programtól, így annak tudomást sem kell vennie róla, hogy éppen milyen azonosítási eljárás lett beállítva. A Linux-rendszer saját PAM-ot használ. A Linux-PAM fejlesztését 1996-ban kezdték meg, legfontosabb fejlesztője a mai napig Andrew Morgan. Az idők folyamán a rendszer rengeteg fejlesztésen ment keresztül. Mi a Linux-PAM 0.72-es vátozatának főbb jellemzőivel ismerkedünk meg.

A PAM lehetőségei

A PAM olyan rugalmas azonosítási rendszert ad a rendszergazda kezébe, mellyel a felhasználókat a rendszeren futtatható bármely PAM-megfelelő program segítségével azonosíthatja. Az azonosítási módszer a legegyszerűbb bugyuta titkos kérdés-titkos válasz azonosítási módtól a legbonyolultabb recehártya- vagy ujjlenyomat-felismerésig terjedhet. Morgan a rendszer lehetőségeinek bemutatására az alábbi példát hozza: ha a rendszergazda (mama) a felhasználói (gyerekek) matematikai tudását szeretné tökéletesíteni, beállíthatja, hogy a kedvenc lövöldözős játékuk (természetesen csak ha PAM-megfelelő) azonosításképpen a 12 alatti számok szorzatát kérdezze. Ha a felhasználó tudja a választ, játszhat. Ha nem... Amennyiben a játék érdekes, a gyerekek egykettőre megtanulják a szorzótáblát. A PAM négy különböző szolgáltatást nyújt: azonosítás, felhasználói név (account), munkamenet (session) és jelszókezelés. Ezeket a beállítási állományokban használjuk: auth, account, session és password. Az azonosítási szolgáltatás két dolgot jelent: egyrészt a PAM valamilyen eljárással meggyőződik a felhasználó személyéről, azaz azonosítja (authenticate), másrészt a PAM a felhasználót jogosultságokkal ruházhatja fel (authorize) (például valamilyen csoportba sorolhatja be a /etc/group alapján vagy más módszerrel). A kapcsolatkezelés a rendszeren azonosítás-független szolgáltatások beállítását teszi lehetővé. Jellegzetesen ilyen a kiszolgálók felhasználó-számának korlátozása, a rendszergazda bizonyos helyekről (például hálózatról) való belépésének megtiltása. A munkamenet-kezelés szolgáltatásai olyan feladatok megvalósítását te-

szik lehetővé, melyeknek a felhasználó hozzáférése előtt vagy után kell végrehajtódniuk (naplózás, chroot stb.). A jelszókezelési szolgáltatás a felhasználó azonosítási zsetonjának módosítását teszi lehetővé. A PAM modulokból áll, amelyek akár több szolgáltatást is nyújthatnak.

A PAM beállítása

A PAM a /etc/pam.d könyvtárban lévő állományokon keresztül hangolható. A beállításokat korábban a /etc/pam.conf nevű állomány tartalmazta, de elavulttá vált, ezért használatát a Linux rendszeren nem ajánlom (akad olyan operációs rendszer, amelyben csak ezt alkalmazzák). A /etc/pam.d könyvtárban lévő állományok neve és a beállítandó szolgáltatás neve kisbetűkkel (su, ssh, ftp stb.) írandó. Az állományokban minden sor (a megjegyzésektől eltekintve) egy PAM-modul valamilyen szolgáltatását hívja meg. Az egyes sorok formátuma a következő:

```
<modul típusa> <ellenőrző zászló (control flag)>
<modulelérési út> <változó (argument)>
```

ahol a modul típusa egy a később megadott modul szolgáltatásai közül. A megadható lehetőségek: auth, account, session, password . Jelentésüket korábban már részletesen ismertettük. Az ellenőrző zászló lehetővé teszi, hogy az adott sorban meghatározott feltétel szükségességét szabályozzuk. Értékei a következők lehetnek:

required (kötelező) a megadott modulnak sikeresen kell visszatérnie, különben a modultípusban megadott szolgáltatás nem térhet vissza hibátlanul. A modul hibája addig nem jut el a felhasználóhoz, amíg az ugyanilyen modultípussal rendelkező sorok mindegyike ki nem értékelődik.

requisite (szükséges) az előzőtől csak annyiban különbözik, hogy a hibát a modul azonnal visszaadja az alkalmazásnak.

sufficient (elégséges) ha a modul sikerrel tér vissza, és a korábban feldolgozott modulok között nincs sikertelenül visszatért required típusú, akkor erre a szolgáltatásra nem hívódik meg több a később felsoroltakból.

optional (tetszőleges) miként a neve is mutatja, ez a bejegyzés nem életbevágóan fontos – amennyiben nem önmagában áll, nem okoz sem elutasítást, sem biztos elfogadást. Ilyen esetben csak a beállításainak köszönhetően teszik be a szolgáltatás végrehajtásába (naplóz, valamit kiír, egyéb).

A modul elérési útja megadja a helyét az állományrendszeren, a változók és azok beállítása pedig modulonként és szolgáltatásonként egyediek. A külön nem beállított alrendszerek az other állományban megadottak szerint vannak azonosítva. Egy kis példa a máshol nem meghatározott PAM-kérések kitiltására:

```
#
# /etc/pam.d/other - default; deny access
#
auth    required    /usr/lib/security/pam_deny.so
accoun  required    /usr/lib/security/pam_deny.so
password required   /usr/lib/security/pam_deny.so
session required    /usr/lib/security/pam_deny.so
```

A PAM rendszer fontosabb alapmoduljai

A PAM rendszer automatikusan telepíti az alapvető modulokat. Az alábbiakban felsoroljuk a leggyakrabban használtakat, és azok fontosabb funkcióit.

Általános PAM hibakeresés: a debug paraméter

A PAM modulok problémáinak felderítésére használható a debug paraméter. Beállításának hatására hiba esetén az adott modul a syslog(3) rendszerhíváson keresztül ír a rendszernaplóba. Mivel minden modul rendelkezik ezzel a paraméterrel, így a továbbiakban nem tárgyaljuk.

A PAM rendszer fontosabb alkotórészei

A pam_unix modul

A Linux rendszereken leggyakrabban használt és legfontosabb modul. A Unix rendszerek hagyományos azonosítási (autentikációs) és feljogosítási (autorizációs) mechanizmusait biztosítja. A rendszer szabványos hívásait használja, tehát a /etc/passwd és a /etc/shadow állományokkal dolgozik. Érdemes megjegyezni, hogy a pam_pwdb modul is hasonló funkciókat nyújt, csak az általa felhasznált adatokat már adatbázisban tartja. Nagy felhasználószámú rendszereken érdemes használni.

account paraméterei: debug; audit
Lehetővé teszi a felhasználói számla érvényességének ellenőrzését. A shadow állomány tartalmaz olyan mezőket (expire; last_change; max_change; min_change; warn_change), amelyek a felhasználó jelszavának kikényszerített lecserélését vagy a számla zárolását teszik lehetővé [1 shad]. Vigyázat! Ha a shadow állomány a fenti mezők valamelyikét nem tartalmazza, akkor az ellenőrzés nem hajtódik végre.

auth paraméterei: debug; audit; use_first_pass; try_first_pass; nullok; nodelay
A felhasználó jelszavas azonosítását teszi lehetővé. Amennyiben több jelszavas azonosítás is be van állítva (lásd később a pam_ldap modulnál), akkor célszerű a try_first_pass paraméter használata. Ilyen esetben a felhasználótól nem kérdezi meg a rendszer újra a jelszavát, hanem az első modul által bekért jelszót használja. Ha azt akarjuk elérni, hogy a felhasználónak több jelszóval kelljen belépnie, akkor ne használjuk. Ilyen esetben a felhasználónak szép sorjában be kell gépelnie a szükséges jelszavakat. A nullok paraméter lehetővé teszi olyan felhasználók belépését a rendszerbe, akiknek a shadow állományban a kódolt jelszó mezője üres. Használata nem javasolt.

password paraméterei: debug; audit; nullok; not_set_pass; use_authtok; try_first_pass; use_first_pass; md5; bigcrypt; shadow; nis; min; max; obscure; remember
A felhasználók szabványos jelszócseréjét teszi lehetővé. Az md5 és bigcrypt paraméterek segítségével elérhető, hogy a jelszó ne a klasszikus crypt [2 crypt] algoritmussal kódolva kerüljön be a végleges helyére, hanem a megadottal. Ne felejtsük ezt beállítani, különben a rendszeren csak nyolc karakter hosszú jelszavakat lehet használni! Itt is használható a try_first_pass akkor, ha a felhasználónak egyforma jelszót szeretnénk beállítani a különböző jelszótárakban. A use_authtok opció kötelezővé teszi a modul számára az előző modul által átadott jelszó beállítását. Erre a pam_cracklib használata ese-

tén van szükség (lásd később). A not_set_pass paraméter segítségével letilthatjuk, hogy a bekért régi vagy új jelszó bármely más modulnak átadásra kerüljön. A nis paraméter hatására a rendszer a NIS RPC-t használja a jelszó beállítására. A min és max beállításával szabályozható a beállítható jelszó legkisebb és legnagyobb hossza. A min paramétert normál felhasználói renszernél célszerű legalább nyolcra, erősen védett rendszernél tízre állítani. Az obscure paraméter néhány alapvető ellenőrzést végez a beállítandó jelszón. Ezek: a jelszó nem hasonlíthat túlzottan az előzőhöz, nem lehet túl egyszerű (jelszóhossz, használt karakterek típusa...), nem lehet az előző jelszó fordítottja, vagy oda-vissza megegyező (például "qwerttrewq").

session Nincs paramétere.
A használatával a felhasználó neve és a szolgáltatás naplózódik a munkafázis (session) elején (a dokumentáció szerint a végén is, de a tapasztalat ennek gyakran ellentmond).

A pam_deny és a pam_nologin modul

A deny segítségével megakadályozható a felhasználó hozzáférése az adott szolgáltatáshoz. auth és account esetén a felhasználó azonosítását és hozzáférését teszi sikertelenné, ha a password komponensben használjuk, akkor a felhasználó nem tudja megváltoztatni a jelszavát. A session-beli használata lehetővé teszi, hogy a felhasználó ne hozhasson létre munkafázist.

A nologin modul a PAM rendszer auth komponenséből elérhető, és a szabványos Unix nologin használatát teszi lehetővé. Amennyiben az /etc/nologin állomány létezik, akkor az azonosítás sikertelen. Leggyakrabban a rendszer felállásakor alkalmazzák, de többfelhasználós rendszeren kényelmes lehetőséget ad egy esetleges karbantartás idején minden felhasználó belépésének időleges tiltására.

A pam_securetty és a pam_shells modul

A securetty és a shells modul meghatározza, hogy az adott felhasználó által használt terminál szerepel-e az /etc/securetty állományban, illetve a felhasználó login shell-je benne van-e az /etc/shells állományban. Mindkettő elutasítja a felhasználót abban az esetben, ha az állomány nem tartalmazza az adott bejegyzést. Mindkét modul a PAM rendszer auth komponenséből elérhető.

A pam_listfile modul

Lehetővé teszi egy szolgáltatás engedélyezését vagy tiltását az azonosítási fázisban egy állomány tartalmán keresztül. Lehetséges paraméterei:

– onerr=succeed|fail
– sense=allow|deny
– file=<állománynév>
– item=user|tty|rhost|ruser|group|shell
– apply=user|@group

A modul veszi az item által meghatározott elemet (ahol a user a felhasználó neve, a tty annak a terminálnak a neve, ahonnan a kérés érkezett, az rhosts a távoli gép neve /ha van/, a ruser megadja a távoli felhasználó nevét /ha van/, a group a felhasználó csoportja), és megnézi, hogy a file által meghatározott állomány tartalmazza-e. Ha tartalmazza, akkor ha a sense értéke allow, akkor a modul sikerrel tér vissza, ha deny, akkor elutasító választ ad. Ha valamilyen hiba történik (például a meghatározott állomány nem létezik), akkor az onerr által beállított értékkel tér vissza. Ezt érde-

mes fail-re állítani. Az apply paramétert akkor célszerű használni, ha a vizsgált elem terminál, távoli gép vagy shell. Segítségével a sikeres visszatérés egy felhasználóhoz vagy egy csoporthoz köthető.

Így tehát egyszerűen korlátozható egy adott szolgáltatás elérése. Használatára a legjellemzőbb példa az ftp, ahol a listfile modult használták fel arra, hogy bizonyos felhasználók számára tiltsák a szolgáltatás elérését. A konfigurációs állományban ez így néz ki:

```
-- /etc/pam.d/ftp részlet
auth  required  pam_listfile.so  item=user sense=deny file=/etc/ftpusers
  onerr=fail
```

Egyéb felhasználására mutatunk példát a későbbi példáknál.

A pam_limits modul

Lehetőséget ad a felhasználók által használható erőforrások korlátozására. Erre azért van szükség, mivel egy többfelhasználós Linux kiszolgálón nem engedhető meg, hogy egy felhasználó olyan mértékben leterhelje a rendszert, hogy a többiek (különösen a rendszer adminisztrátorai) ne tudják zavartalanul a munkájukat végezni. Sajnos a Linux rendszer ezen korlátozások használatát csak kis mértékben támogatja, így a rosszindulatú felhasználóktól tökéletesen csak a hagyományos módszerek védenek meg (időleges vagy végleges kizárás, vasalt orrú bakancs, baseballütő, gyöngyház berakású zsebkés stb.).

Ezen keserű kijelentések a hosszas kísérletezés után alakultak ki. Amennyiben a felhasználók memóriafelhasználását korlátoztuk, a rendszer túlterhelhető volt valamelyik fork bombával. Ha a belépésenkénti folyamatok számát korlátoztuk 4-re, a fork bombák akkor is szinte a teljes processzoridőt fel tudták használni, ráadásul az openssh segítségével nem lehetett belépni a rendszerre. Tapasztalatunk szerint ssh-val csak akkor sikerült belépni a rendszerre, ha a lehetséges folyamatok száma legalább 40 volt. Mivel azonban a sikeres belépést követően a felhasználó 40 folyamatot futtathat, így rossz esetben a teljes processzoridőt el tudja venni. Az ésszerűtlen memória fogyasztást ugyan meg lehet gátolni, de a sok memóriafoglalási kísérlet sajnos szintén megeszi a processzor idejének a jelentős részét. Ebben az az extra kellemetlenség, hogy ilyen esetben a legtöbbet a kernel dolgozik, így még korlátozni sem lehet. Tehát a folyamatok számának korlátozása bizonyos esetekben nem megfelelő a PAM támogatás tökéletlen implementációja miatt.

Ha a rendszeren a kifejezetten rosszindulatú felhasználókat ki tudjuk szűrni, akkor van értelme a határok beállításának, így a felhasználó akaratán kívüli rendszertúlterhelés esélye csökkenthető. Sokat segíthet, ha a felhasználók nice szintjét csökkentjük, így hiba esetén a rendszer adminisztrátorai nagyobb eséllyel tudnak sikeresen beavatkozni.

Ideális az lenne, ha a felhasználónak általános határokat lehetne beállítani (jelenleg csak a belépésenkénti létezik), és a rendszermag lehetővé tenné, hogy egy bizonyos felhasználó által kezdeményezett (felhasználó vagy rendszermag által végzett) feladat legfeljebb mekkora részt kaphasson a rendelkezésre álló processzoridőből (fair share scheduling). Amíg ezek nem lesznek a hivatalos rendszermagban megvalósítva, addig a felhasználók korlátozása csak részleges lehet.

Ugyan nem tartozik szorosan a témához, de itt érdemes megjegyezni, hogy a rendszermag lehetővé teszi annak a helynek a korlátozását, amit a felhasználók merevlemezen foglalhatnak. Így ésszerű mértékűre csökkenthető az egyes felhasználók területhasználata, és a levelesláda (mailbox) sem nőhet egy adott méret fölé a többiek kárára. Ennek beállítása esetén azonban figyelni kell arra, hogy mit tesz ilyen esetben a levelező kiszolgáló.

Egyéb hasznos modulok

A pam_env modul (auth) környezeti változók előzetes beállítását vagy törlését teszi lehetővé. Több felhasználós rendszeren célszerű használni, mivel lehetővé teszi a környezet egységes beállítását a belépési shell-től függetlenül.

A pam_rootok modul (auth) azonosítja a felhasználót, ha a felhasználói azonosítója 0. Ennek akkor lehet értelme, ha a root felhasználót nem akarjuk egy szolgáltatás minden egyes használatakor azonosítani. A Linux rendszerek legnagyobb részén megengedett a root-nak a su használata jelszó nélkül. Ez a konfigurációs állományban így néz ki:

```
-- konfig állomány részlet
    auth       sufficient pam_rootok.so
    auth       required   pam_unix.so
```

Így tehát a rendszergazda mindenkivé annak jelszavának megadása nélkül válhat. Ez felvet bizonyos etikai kérdéseket, de ez szinte minden rendszergazdai jogosítványnál felmerül. Speciális esetben elérhető egy finoman hangolt, külső szakértők által is felülvizsgált rendszernél a rendszer adminisztrátorainak jogainak a szükségesre csökkentése, de ez komoly hozzáértést és erőforrás ráfordítást igényel.

A pam_chroot modul (account, session, auth) segítségével lehetővé válik egy adott szolgáltatás root könyvtárának beállítása a PAM rendszeren keresztül. Ennek használról egy későbbi cikkben részletesebben írunk. A modult jelenleg kissé nehézkes használni, mivel a PAM-ot támogató programok egy része nem megfelelően valósítja meg a session kezelést. Jó példa erre az openssh, ahol a fejlesztők valmilyen okból nem is akarják átalakítani az alkalmazást. Többen kijavították az ssh hibáit, de a fejlesztők nem fogadták be a javításokat.

A pam_mkhomedir modul lehetővé teszi a felhasználók munkakönyvtárának röptében való létrehozását. Mivel a felhasználó adminisztrációs eszközök ezt megteszik, használata csak speciális esetben célszerű.

A pam_motd, pam_mail, pam_lastlog és pam_issue modulok a felhasználók tájékoztatását szolgálják. Belépéskor kiírják a terminálra a /etc/motd és az /etc/issue állományok tartalmát, az utolsó belépés idejét, jelzik ha a felhasználónak új levele érkezett.

A pam_radius és pam_krb4 modulok segítségével a felhasználók azonosítása egy RADIUS [3 rad] vagy Kerberos [4 kerb] kiszolgáló segítségével történik. Ezek az azonosítási eljárások általában nagyobb hálózatokon használatosak, és nagybiztonságú azonosítást tesznek lehetővé. A pam_securetty segítségével egy állományban meghatározható, hogy mely terminálok tekinthetők biztonságosnak.

A pam_time modul használata lehetővé teszi a hozzáférés idő szerinti korlátozását. Minden biztonsági rendszer alapvető eleme a megszokott és elfogadott endegélyezése, és a kirívó esetek tiltása. Ha például nem hihető, hogy a rendszeradminisztrátor reggel tíz óra előtt a konzolról belép, akkor ez letiltható. Vagy egy PAM-ot támogató játék alkalmazással megoldható, hogy csak munkaidőn túl lehessen elindítani.

A PAM extra moduljai

A pam_cracklib modul

A unix modul password komonensének a kiegészítésére szolgál. Jóval finomabb jelszó bonyolultság ellenőrzést tesz lehetővé. A unix modul obscure paraméterénél leírtakon felül képes a szótári szavakon alapuló jelszavak kiszűrésére is. A megfelelő működéséhez szükség van egy megfelelő szavakat tartalmazó szótárra. Ezt a legegyszerűbben úgy állíthatjuk elő, ha összeszedünk nagyobb mennyiségű levelezési lista archívumot, majd szavakra bontjuk. Célszerű olyan csomagokat is gyűjteni, ahol a népek ékezettel leveleznek, mivel a felhasználók előszeretettel tesznek ékezetes szavakat a jelszavukba. Praktikus továbbá összeszedni a felhasználók, kisállataik és szeretteik adatait. A következő kis perl programocska segítségével szavakká vágjuk a szöveg állományokat:

```
#!/usr/bin/perl -wn
next if /^\s*$/;
s/#\S*//g;
tr/A-ZÁÉÖÜÓŐÚŰÍ/a-záéöüóőúűí/;
tr/\012éáöüóőúűía-z0-9 \t//cd;
s/\s+/\n/g;
print;
```

A program standard bemenetén várja a tiszta szövegállományokat (kismértékben akár html lapokat is, ugyan így feleslegesen nő a leendő adatbázis mérete), és a standard kimenetén normál, szavakra vágva jönnek ki az adatok. A Debian által adott eszközökkel az alábbi parancsösszetétellel állítható elő az adatbázis:

```
$ cat <sok szövegállomány neve> | mini_splitter | sort -u |
  crack_packer /var/cache/cracklib/cracklib_dict
```

Nagyméretű adatállományok esetén a sort-nak célszerű megadni egy átmeneti könyvtárak a -T paraméter segítségével. Ezzel előállítottuk a szóadatbázist, amit a későbbiekben célszerű időnként frissíteni. A modul leellenőrzi azt, hogy a megadott jelszó nem képezhető-e valamelyik szótári szóból a kisbetűk és nagybetűk valamilyen kombinációjával.

A modul a PAM password komponensében működik, használata egyszerűsítve a következő: a felhasználó által megadott jelszó minden karaktere egy pontot ér, továbbá minden különböző karakterosztályba tartozó karakter egy jutalompontot. Meghatározhatjuk, hogy egy adott karakterosztályra legfeljebb mennyi jutalompontot adjon a rendszer. Az ismert osztályok: kisbetű (lower), nagybetű (upper), számjegy (digit) és egyéb (other). A jutalompontok hangolása a következő paraméterek beállításával történik:

```
dcredit=<N>; ucredit=<N>; lcredit=<N>; ocredit=<N>,
```

ahol <N> az adott osztály karaktereire adható maximális plusz pontok száma. Ha a megadott szám alatti, vagy azzal megegyező számú adott osztályú karakter szerepel a jelszóban, akkor azok mindegyikére egy plusz pont jár. A karakterszámból adódó és a jutalompontok összegének minimumát a minlen=<N> paraméterrel szabályozhatjuk. Az <N> értéke a megengedhető minimum plusz egy. Így a következő beállításokkal:

```
dcredit=2 ucredit=1 lcredit=1 ocredit=2 minlen=12
```

csak olyan jelszó lesz elfogadható, amely vagy legkevesebb 10 kisbetűből áll, vagy ha van benne nagybetű, akkor nem rövidebb, mint 9 karakter, vagy ha van benne két számjegy és nagybetű, akkor nem rövidebb, mint 7 karakter és így tovább.

A régi és új jelszó elvárt különbsége a difok=<N> paraméterrel állítható be. Segítségével beállítható, hogy egy megadott jelszóban hány karaktert kell mindenképpen lecserélni. Alapbeállítása 10, de ehhez még adódik egy újabb szabály: ha a jelszó karaktereinek legkevesebb a fele lecserélődik, az felülbírálja az itteni beállítást, és a jelszó megfelel.

A pam_ldap modul

Nagyobb hálózatoknál gyakran merül fel a probléma, hogy a felhasználók vándorolnak a munkahelyek között, de mindenhol szeretnék a megszokott munkakörnyezetet látni. A rendszer adminisztrátorok szempontjából komoly probléma lehet nagy tömegű felhasználó kezelése. Ilyen esetekben lehet célszerű az ldap modul. A felhasználók adatai egy központi LDAP kiszolgálón vannak tárolva, így tetszőleges munkaállomásra a megszokott jelszavukkal léphetnek be. A teljes támogatáshoz ne felejtsük el áthangolni az nsswitch lib-eket sem [5 authldap]. A munkakönyvtárak átvitelére megfelel valamilyen hálózati állományrendszer. Erre a célra jelenleg az NFS a legelterjedtebb megoldás, de mivel ez biztonsági szempontból erősen megkérdőjelezhető, ezért használata kizárólag olyan környezetben elfogadható, ahol a kliensek tökéletesen megbízhatóak. Vagyis szinte sehol. Jelenleg a legésszerűbb a felhasználók munkakönyvtárait ssl-sambán keresztül kiajánlani, így lehetővé válik a biztonságos csatlakozás. A kis kitérő után térjünk vissza a központi felhasználó azonosításhoz.

Az ldap modul használatához először is szükségünk lesz egy LDAP kiszolgálóra, ahol a felhasználók adatait tároljuk. Mi az OpenLDAP 2.0.14-es változatát használtuk. A telepítés Debian Woody rendszeren a megszokott apt-get parancs segítségével triviális (a csomag neve slapd). Ha előre megterveztük a leendő LDAP struktúráját, akkor a telepítés közben létre lehet hozni a háttér adatbázist. A rendszer adatinak átemelésére tökéletesen alkalmas a PADL cég által fejlesztett, MigrationTools nevű eszköz [6 ldapmig]. Az OpenLDAP-nál az alap konfigot kissé módosítani kellett, hogy a megfelelő séma definíciókat is betöltse. Ésszerű a hozzáférést is beszabályozni, mert az alaptelepítés bejelentkezés nélkül is olvasási jogot ad. A kissé paranoiásabb beállítás megfelelő része valahogy így néz ki:

```
-- az /etc/ldap/slap.d megfelelő része
# A névtelen azonosíthatja magát, ha sikeresen bejelentkezett, átírhatja a
#   a saját jelszavát. Az adminisztrátor mindenkiét írhatja. Más senki nem
#   csinál semmit.
access to attribute=userPassword
        by dn="cn=admin,o=Andrews,c=HU" write
        by anonymous auth
        by self write
        by * none
# Az adminisztrátornak teljes írás joga van mindenre. A saját adatait minden
#   bejeltkezett felhasználó olvashatja, a többieknek nincs semmilyen joguk.
access to *
        by dn="cn=admin,o=Andrews,c=HU" write
        by self read
        by * none
```

Ezek után kezdődhet az ldap modul üzembe állítása. Ennek beállítása az /etc/ldap.conf állományon keresztül történik. Az állomány tartalma:

```
-- /etc/pam_ldap.conf
# Az LDAP kiszolgáló neve
host tensor.andrews
# A keresés kiindulópontjának a DN-je
base ou=People,o=Andrews,c=HU
# A root DN-je (a jelszava a /etc/ldap.secret állományban található, amely a
# libpam-ldap csomag telepítésekor kitöltésre kerül)
rootbinddn cn=root, o=Andrews, c=hu
# Ennek a DN-nek a nevében keres a pam_ldap modul.
binddn cn=admin, o=Andrews, c=hu
bindpw ubertitkosjelszo
# A jelszavak tárolási formája
pam_password md5
```

Egy példa rendszer beállításai

Legyen példánk tárgya egy kisebb hálózat egyik felhasználói gépe. A felhasználók gyakran vándorolnak a gépek között, így az azonosításukat LDAP-on keresztül oldjuk meg. A felhasználók a rendszert a konzolról való belépéssel érik el, távolról csak az adminisztrátorok léphetnek be ssh segítségével. Megmutatjuk a login és az ssh PAM beállításait, valamint a limits modul konfigurációját.

A login modul beállításai a megszokottól csak annyival térnek el, hogy a felhasználókat ldap-ból is lehet azonosítani. Ha a felhasználó azonosította magát az ldap modul segítségével, akkor beengedjük. Ha nem, akkor megpróbáljuk a helyi felhasználói adatbázisból azonosítani. Kiegészítés továbbá, hogy a pam_listfile modul használatával lehetővé tesszük bizonyos felhasználók rendszerről való időleges kitiltását. Ennek kezelésére az 5. listában megadott egyszerű kis perl program szolgál. Segítségével kilistázhatjuk a tiltott felhasználókat, felvehetünk és törölhetünk tiltást. Használatának megismeréséhez használjuk a -h paramétert.

Példarendszerünkön az /etc/pam.d/login így néz ki:

```
-- /etc/pam.d/login
# PAM konfigurációs állomány a `login' szolgáltatáshoz
auth       requisite  pam_listfile.so item=user sense=deny
  file=/etc/security/deny_users onerr=fail
auth       requisite  pam_securetty.so
auth       required   pam_nologin.so
auth       required   pam_env.so
auth       sufficient pam_ldap.so
auth       required   pam_unix.so try_first_pass
account    required   pam_unix.so
session    required   pam_unix.so
session    required   pam_limits.so
session    optional   pam_lastlog.so
session    optional   pam_motd.so
session    optional   pam_mail.so standard noenv
password   required   pam_cracklib.so retry=3 minlen=6 difok=3
password   required   pam_unix.so use_authtok md5
password   required   pam_ldap.so try_first_pass
```

Az ssh konfigurációja annyiban tér el a Debian által feltelepítettől, hogy fel lett véve egy listfile modul, amely kizárólag az /etc/security/admins állományban felsorolt felhasználókat engedi be. Az ssh pam állománya:

```
-- /etc/pam.d/ssh
# PAM konfigurációs állomány az `ssh' szolgáltatáshoz
auth       requisite  pam_listfile.so item=user sense=allow
  file=/etc/security/admins onerr=fail
auth       required   pam_nologin.so
auth       required   pam_env.so
auth       sufficient pam_ldap.so
auth       required   pam_unix.so
account    sufficient pam_ldap.so
account    required   pam_unix.so
session    required   pam_unix.so
session    optional   pam_lastlog.so
session    optional   pam_motd.so
session    optional   pam_mail.so standard
session    required   pam_limits.so
password   required   pam_cracklib.so retry=3 minlen=6 difok=3
password   required   pam_unix.so use_authtok md5
password   sufficient pam_ldap.so try_first_pass
```

Ha a felhasználó a rendszeren jelszót változtat, akkor azt az ldap-ban is meg kell változtatni. Ennek érdekében a passwd pam beállításait a következőképpen kellett módosítani:

```
-- /etc/pam.d/passwd
# PAM konfigurációs állomány a `passwd' szolgáltatáshoz
password   required   pam_cracklib.so retry=3 minlen=6 difok=3
password   required   pam_unix.so use_authtok md5
password   sufficient pam_ldap.so try_first_pass
```

Ezzel az alap beállításokat megtettük. Innen már mindenkinek az ízlésére bízzuk a finomhangolást. A részletek és mellékhatások tekintetében nézzük meg a PAM rendszer dokumentációját [7 pam]. Azon szolgáltatásokról, amikről hely hiányában nem tudtunk bővebben szólni, találunk elegendő információt a Linux PAM hivatalos honlapján [8 lpam]. Ugyanitt sok egyéb hasznos modulra is ráakadhatunk. Mindenkinek hasznos keresgélést kívánunk.

Hivatkozások

[1 shad] A shadow állomány formátuma: shadow (5)

[2 crypt] A crypt algoritmus leírása: crypt (3)

[3 rad] RADIUS: http://www.gnu.org/software/radius/radius.html

[4 kerb] Kerberos: http://web.mit.edu/kerberos/www/

[5 authldap] Authenticating with LDAP using Openldap and PAM: http://www.imaginator.com/~simon/ldap/

[6 ldapmig] plain to lpdap migration tools: ftp://ftp.padl.com/pub/MigrationTools.tar.gz

[7 pam] A PAM rendszer felhasználói kézikönyve: http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/

[8 lpam] A Linux PAM rendszer hivatalos honlapja: http://www.kernel.org/pub/linux/libs/pam/

E-közigazgatási Szabad
Szoftver Kompetencia Központ

MAGYARY PROGRAM

Nemzeti Fejlesztési Ügynökség
www.ujszechenyiterv.gov.hu
06 40 638 638

MAGYARORSZÁG MEGÚJUL

A projekt az Európai Unió támogatásával, az Európai Regionális Fejlesztési Alap társfinanszírozásával valósul meg.