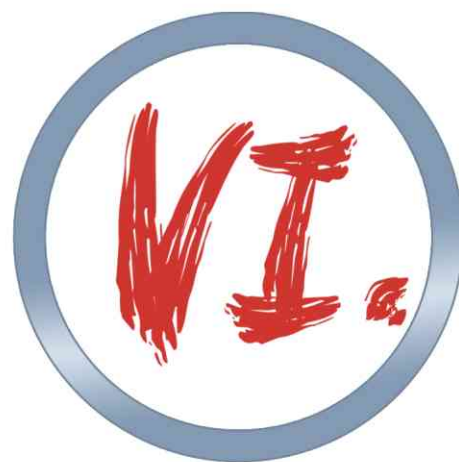


GNU/Linux

Szakmai konferencia

2004



Linux-felhasználók Magyarországi Egyesülete

Budapest, 2004. november 20.



COMPUTERWORLD

SZÁMÍTÁSTECHNIKA

I N F O R M Á C I Ó E L S Ő K É Z B Ő L

Sun Java Desktop System



Kommunikációs és csoportmunka szolgáltatások:

- Mozilla böngésző
- Evolution e-mail és naptárkezelő
- GAIM Instant Messenger

Irodai alkalmazások:

- StarOffice 7 Suite
- Project Manager
- Adobe Reader

Biztonsági szolgáltatások:

- Antivírus szoftver
- Java Card alapú azonosító
- Single Sign-on

Multimédiás eszközök:

- Macromedia Flash
- Real Networks RealONE
- Java Software

Kompatibilitás:

- MS Office
- Exchange Server csatoló
- Sun Java Messaging és Calendar Server csatoló



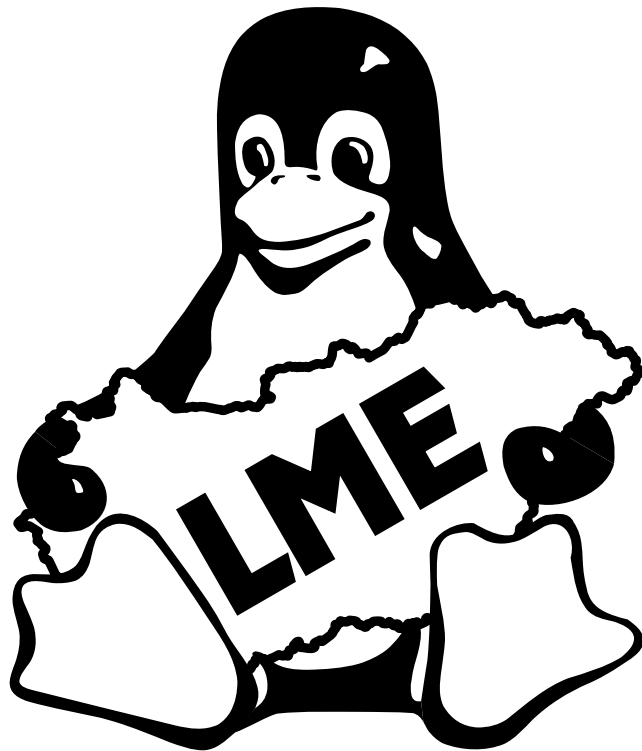
Sun
microsystems

ASBIS

Distribution For Your Success

ASBIS Magyarország Kft. • Hivatalos számítógépalkatrész disztribúció
1139 Budapest, Váci út 81-85. • Tel.: (06 1) 236-1000 • Fax: (06 1) 236-1010
E-mail: infosales@asbis.hu • Web: www.asbis.hu

VI. GNU/Linux Szakmai Konferencia



Budapest, 2004. november 20.

A kiadvány tördelése a \TeX 3.14159 verziójával készült.
Helyesírás-ellenőrzés: Hunspell 1.0-RC1 verzió
Elváltásztás: Huhypn-20041031
A \TeX az American Mathematical Society bejegyzett védjegye.

Szerkesztette: Zelená Endre ezelena@lme.linux.hu

Linux-felhasználók Magyarországi Egyesülete
1395 Budapest 62, Pf. 432,
URL: <http://www.lme.hu/>
E-mail: info@lme.linux.hu

Minden jog fenntartva. Jelen kiadvány elektronikus verziója módosítás nélkül szabadon terjeszthető. A nyomtatott változat terjesztése, másolása, informatikai rendszerben történő további feldolgozása, tárolása csak a szerzők írásos hozzájárulásával lehetséges.

Tartalomjegyzék

Bodnár Csaba: Tartsd az ütemet! – A Linuxos ütemezők fejlődése	9
Czakó Krisztián: Könnyen alkalmazható Linux biztonsági megoldások	17
Deim Ágoston: Központi mentési eljárások	31
Fejős Tamás: Honosítási helyzetjelentés, avagy tényleg bárki megérti azt amit a képernyőn lát?	43
Harka Győző: Az alapértelmezett telepítések biztonsági hiányosságai és orvoslásuk	53
Havasi Ferenc: Szegedi Szemelvények Szabad Szoftvereiről	59
Höltzl Péter: Hálózati határvédelem kialakítása GNU/Linux alapokon	65
Kovács Krisztián: Tűzfalfűrtök állapotartó Netfilter tűzfalakkal	75
László Gábor: A nyílt forráskód társadalmi értéke és a civil szervezetek szerepvállalási formái	89
Légrádi Gábor: UNIX egy kicsit másképp = BSD	95
Mátó Péter: Hibakeresés és elhárítás Linux rendszeren	101
Nemes Dániel: SPAMek és vírusok: elmosódó határok	111
Németh László: Mire jó a Hunmorph?	117
Parragh Szabolcs: Nyílt forráskódú szoftverek a kis- és középvállalatok piacán	125
Scheidler Balázs: Hitelesítési lehetőségek és eszközök egy vállalati információs rendszerben	131
Süveg Gábor: pkgsr – az univerzális csomagkezelő	137
Tomka Gergely: Szerverkonszolidáció UML-lel	141
Interware – Vincze Gábor: Gameserver üzemeltetés Linux alatt (x)	149
Novell – Hargitai Zsolt: Rendszerfelügyelet Linux környezetben (x)	155

A konferencia támogatói

Fő médiatámogató:

Computerworld Számítástechnika

Arany fokozatú támogatók:

- Asbis Magyarország Kft.
- HP Magyarország Kft.
- Interware Internet Szolgáltató Rt.
- Matáv
- Novell Magyarország
- Sun Microsystems Kft.

Ezüst fokozatú támogatók:

- BalaBit IT Biztonságtechnikai Kft.
- Kiskapu Kft. – Linuxvilág magazin
- LSC Linux Support Center Kft.
- Mission Critical Linux Kft.
- RedHat Europe

Előszó

Kedves vendégünk, Tisztelt Olvasók!

Rendhagyó előszó következik – több okból is. Rendhagyó azért, mert nem a szervezők, hanem a kiadvány szerkesztője követi el, és rendhagyó azon egyszerű oknál fogva is, hogy nem az „Üdvözljük...” szóval kezdődik. ☺

Az üdvözlés, a köszöntő szavak persze nem maradhatnak ki, hiszen a konferencia a vendégekért, a kiadvány pedig az olvasókért – azaz önökért – jött létre, készült el. Köszöntök tehát mindenkit, akikért, és akiknek a munkája által immáron hatodszor hallgathatunk meg előadásokat az LME szervezésében zajló szakmai konferencián, és olvashatjuk a cikkeket a konferencia kiadványában.

Az LME életében a tavalyi konferencia óta eltelt időt igen „forgalmasnak” lehetne nevezni; talán a legfontosabb történéseit, feladatát az EU-s szoftverszabadalmak kérdésköréhez tartozó igen komoly munka jelentette.

Azzal kezdtem, hogy rendhagyó előszó következik, így az egyesület életének mozzanatai helyett inkább a kiadvány létrejöttét boncolgatom kicsit – szerkesztői szemmel.

Ez a szem sír is, meg nevet is: Sír, mert sok jó kivonatot kaptunk, azonban többen sajnos megálltak ennél: nem készült minden kiválasztott kivonatból cikk, és nevet, mert örül a sok jó, és formás ☺ cikknek. Persze nem csak ez, hanem a nyers cikkekben talált elgépelések is tudnak kellemes perceket szerezni, idén a képzeletbeli pálmát a „betörélsdetektáló” kifejezés vitte el – gondolom, mindenki rájön, hogy melyik cikkről lehet szó.

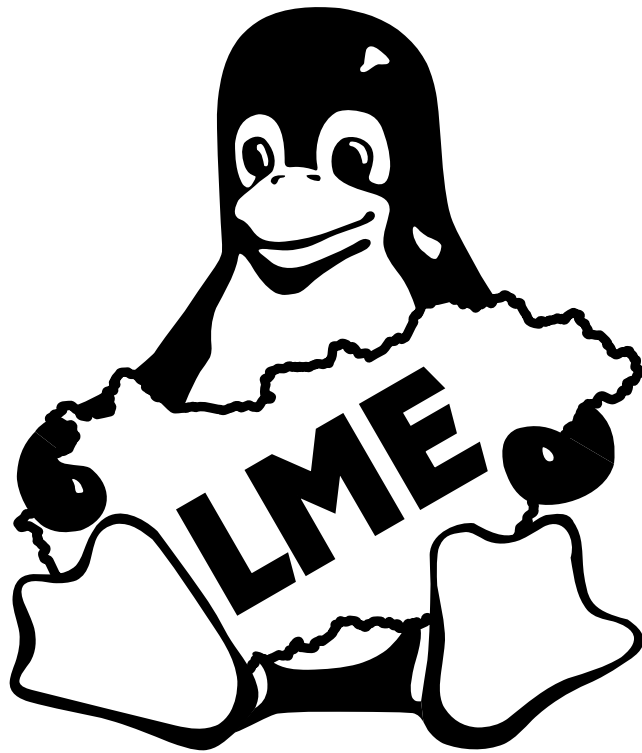
Szeretném megköszönni valamennyi szerző és előadó munkáját, remélem, hogy jövőre is legalább ilyen jó témákkal és cikkekel lepnek meg mindannyiunkat. Ha már a köszöneteknél tartunk, a leadott anyagok átolvasásában segédkező két kollégának, *Tímár Gábornak* és *Sári Gábornak* szeretnék név szerint is köszönetet mondani: nagyon sok hibára, problémás fogalmazásra hívták fel a figyelmemet.

Köszönet illeti a \LaTeX -hez használt *Huhyphn* elválasztási csomag készítőit, és az alapvető nyelvtani ellenőrzésre használt *Hunspell* alkotóit is.

Nincs hibátlan program, és nincs hibátlan kiadvány sem – azonban minden igyekezetünkkel azon voltunk, hogy a zavaró hibákat mindenképp kigyomláljuk a kiadványból, remélem, ezt a célt legalább a tisztelt olvasók által elvárt mértékben sikerült elérnünk.

Bízva abban, hogy a konferencián elhangzottak, és a kiadványban leírtak sok hasznos információval segítik a vendégeket/olvasókat, várunk mindenkit a jövő évi konferencián!

Előadások



Tartsd az ütemet! – A Linuxos ütemezők fejlődése

Bodnár Csaba

<bocs@missioncriticallinux.hu>

Kivonat

Az előadás a különböző Linuxos ütemezőkről szól. Az alapfogalmaknak, és az ütemezőnek a kernelben betöltött szerepének tisztázása után áttekinti a főbb típusokat, majd pedig ismerteti a különböző, egyre fejlettebb Linuxos ütemezőket, különös tekintettel a jelenlegi 2.6-os kernelben használt Molnár Ingo-féle $O(1)$ ütemezőre.

Tartalomjegyzék

1. Alapfogalmak	10
2. Algoritmusok hatékonysága	11
3. Ütemezők típusai	11
4. Az ütemező helye a kernelben, általános viselkedése	11
5. Az „eredeti” Unix-ütemező	11
6. Az 1.2-es Linux kernel ütemezője	12
7. A 2.4.25-es Linux kernel ütemezője	13
8. A Molnár Ingo-féle $O(1)$ ütemező	13
8.1. Miért volt szükség új ütemezőre?	13
8.2. Az új ütemező előnyei	14
8.3. Hogyan működik az ütemező?	14
8.4. Az interaktivitás-becslő (interactivity estimator)	14
9. Az $O(1)$ további hangolása: A Kolivas-féle lépcsős (staircase) ütemező	15

1. Alapfogalmak

program: Végrehajtható állomány.

folyamat v. feladat (process): Egy programnak a számítógép memóriájában lévő példánya, végrehajtás közben.

többfeladatos (multitasking) rendszer: Olyan operációs rendszer, amely egyidejűleg több feladat egymással párhuzamos futtatására képes.

időosztásos (time sharing) működés: Ekkor a többfeladatos rendszer a következő módon kerül megvalósításra: az egyes folyamatok egymás után, de csak egy bizonyos – általában nagyon rövid – időtartamig (időszel) használhatják a rendszer erőforrásait, utána valamely másik folyamat kerül sorra.

ütemező (scheduler): Az ütemező felelős a processzor, mint szűkös erőforrás megfelelő szétosztásáért az egyes folyamatok között.

kooperatív többfeladatos rendszer: Olyan rendszer, amely nem képes a futó folyamattól a vezérlés visszavételére magának a folyamatnak a közreműködése nélkül.

preemptív többfeladatos rendszer: Olyan rendszer, amely képes a futó folyamattól a vezérlés visszavételére annak segítségével nélkül, ha lejárt a számára rendelkezésére álló időszel.

futási sor (run queue): A futásra kész feladatok listája.

nice-érték: Egy feladat indítója által megadott, a feladathoz rendelt, -20 és 19 közötti érték. Alapértelmezése 0 . Minél kisebb a nice érték, a feladat annál nagyobb előnnyel jut processzoridőhöz ütemezéskor. Negatív nice értékeket csak a rendszeradminisztrátor (root) adhat.

prioritás: Szó szerint: elsőbbség; az egyes feladatokhoz egy-egy prioritásérték van hozzárendelve, amely egy rangsort határoz meg.

statikus prioritás: A fenti nice-értékből számított, és a feladathoz rendelt, annak futása közben változatlan prioritás. A feladat indulásakor általában a dinamikus prioritás induló értéke; a dinamikus prioritás újraszámolásakor az egyik döntő tényező.

dinamikus prioritás: Egy feladat futása közben folyamatosan változó, ahhoz tartozó prioritás, amely fontos szerepet játszik annak eldöntésében, hogy melyik legyen a processzorra kerülő következő feladat.

óra-megszakítás (timer interrupt, clock interrupt, clock tick): Fix időközönként, (a hardvertől és a beállításoktól függően $1-100$ ms-onként) végrehajtásra kerülő megszakítás rutin.

interaktív feladat: Viszonylag alacsony CPU-igényű feladat, amely többnyire az interaktív felhasználó beavatkozására vár (sleep on I/O).

köteget feladat (batch task): Általában a háttérben futó, a processzort folyamatosan terhelő feladat.

2. Algoritmusok hatékonysága

Az O (ordó) operátor egy algoritmus hatékonyságát fejezi ki:

- $O(n)$ azt jelenti, hogy a végrehajtás ideje egyenesen arányos a tömb vagy lista elemeinek számával
- $O(1)$ pedig azt, hogy az algoritmus valamennyi esetben közel ugyanannyi – konstans – idő alatt fejeződik be.

3. Ütemezők típusai

- Round robin ütemezés,
- Prioritásos ütemezés,
- Többszörös sorok,
- A legrövidebbet először,
- Garantált ütemezés,
- Sorsjáték ütemezés,
- Valós idejű ütemezés,
- Kétszintű ütemezés.

4. Az ütemező helye a kernelben, általános viselkedése

A különböző Unixos és Linuxos ütemezők nagyjából ugyanúgy működnek. Az ütemező mondja meg, hogy melyik feladat legyen a következő, amelyik végrehajtásra kerül. Ezért a kernel azon pontjairól hívódik meg, ahol annak eldöntésére van szükség, hogy ki kerüljön a jelenleg futó feladat helyére, vagyis:

- amikor egy feladat sleep állapotba kerül (pl. I/O-ra vár)
- amikor egy feladatnak lejár a rendelkezésre álló időszetele

A rendszer folyamatosan (minden egyes óra-megszakításkor – clock interrupt) információt gyűjt a feladatok processzor-felhasználásáról, és ebből (valamint a statikus prioritásból) áll össze azok ún. dinamikus prioritása.

Amikor az ütemezőt meghívják, kiválasztja a legnagyobb dinamikus prioritású feladatot és ő lesz a következő, aki a processzorra kerül.

5. Az „eredeti” Unix-ütemező

Az eredeti System V Unix ütemezője – a „Bach-könyv” [1] leírása szerint – a következőképpen működik¹:

A rendszermag „odaadja” a processzort az egyik feladatnak egy meghatározott időszetre; ha az időszet lejár, elveszi tőle (preempts) és berakja a feladatot valamelyik prioritási sorba.

¹A legtöbb mai gyártóspecifikus Unix-rendszer ütemezője – kisebb-nagyobb módosításokkal (performance, SMP-támogatás vagy egyéb célból) – is ezen alapul.

```

algorithm schedule_process
input: none
output: none
{
    while (no process picked to execute)
    {
        for (every process on run queue)
            pick highest priority process that is loaded in memory;
        if (no process eligible to execute)
            idle the machine;
        /* interrupt takes machine out of idle state */
    }
    remove chosen process from run queue;
    switch context to that of chosen process, resume its execution;
}

```

Azt, hogy melyik feladatot választja ki futtatásra, azt az ütemező algoritmus (lásd a fenti pszeudokódot) dönti el, amelynek lényege a következő: Az algoritmus végigmegy a futási soron, és kiválasztja a legmagasabb prioritású feladatot; kiveszi a futási sorból, majd átkapcsol a feladat környezetére (context switch) és folytatja annak futtatását (ott, ahol előzőleg abbahagyta).

A folyamatok rendelkeznek egy ún. dinamikus prioritással. Minden egyes óramegszakításakor (vagyis hardvertől és beállítástól függően 1–100 ms-onként) az éppen futó folyamat „aktuális CPU-felhasználás” (recent CPU usage) nevű mezője eggyel növekszik – vagyis valaki minél többet használja a processzort, annál magasabb lesz a hozzá tartozó érték. Emellett másodpercenként egyszer az összes feladat ezen mezője megfigyelődik:

$$decay(CPU) = CPU/2$$

így biztosítva a régebbi adatok „elavulását”. Ugyanekkor a futásra kész feladatok dinamikus prioritása is újraszámolódik a következő képlet alapján:

$$prioritás = (aktuális\ CPU-felhasználás/2) + alapprioritás + nice\ érték$$

Ez az a prioritás, amely alapján eldől, mi lesz a következő, aki a processzorra kerül. Minél kisebb a kiszámolt prioritás érték, annál nagyobb egy feladat prioritása.

Látszik, hogy ez a módszer az interaktív feladatoknak kedvez (szemben a háttér-feldolgozás jellegűekkel), ugyanis a kevés processzoridőt felhasználókat részesíti előnyben.

Mellékesen megjegyezzük, hogy a feladatok prioritási szintenként sorokba vannak rendezve; az időnkénti újraszámoláskor pedig átkerül(het)nek egyik sorból a másikba.

6. Az 1.2-es Linux kernel ütemezője

Az 1.2-es kernel ütemezője meglepően hasonlít az eredeti Unixoshoz.

A `schedule()` nevű rutin ugyanúgy 2 helyről hívódik meg:

- amikor egy feladat I/O-ra vár (bizonyos rendszerhívások indítják a `sleep_on()`-on keresztül)
- Minden egyes rendszerhívás és „lassú megszakítás” (ilyen az óra-megszakítás is) végén ellenőrzésre kerül, hogy az éppen futó feladat ideje lejárt-e már (a `need_resched` változó be van-e kapcsolva).

Maga a `schedule()` – egyéb más teendői után – végignézi az összes feladatot és a legnagyobb dinamikus prioritással (counter mező) rendelkező feladatot választja. Ha nincs futtatható feladat, akkor az `init` feladat fog futni. Ha az összes futásra kész feladat counter mezője lement már nullára, akkor az algoritmus végigmegy az összes feladaton és újraszámolja a dinamikus prioritást a következő képlettel:

```
p->counter=(p->counter >> 1) + p->priority;
```

vagyis a dinamikus prioritást elosztja kettővel, és hozzáadja a statikus prioritást (ami a nice érték, csak éppen negatív előjellel). Ha kicsit végiggondoljuk, ez azt jelenti, hogy a dinamikus prioritás soha nem lehet nagyobb a statikus kétszeresénél: ezzel akadályozza meg azt, hogy az éppen várakozó feladatoknál se halmozódhasson fel túlságosan sok „prioritás bónusz”.

Az idő-megszakításkor az éppen futó feladat counter mezője eggyel csökken – szemben az eredeti Unixos megoldással, ahol az aktuális CPU-felhasználás mező eggyel növekszik. Míg a Unixos algoritmusban a prioritások egekbe szökésétől a másodpercenkénti újraszámolás véd meg, addig itt az összes futtatható feladat counterének lenullázódásakor történik újraszámolás. Ott minél nagyobb a prioritás értéke, annál alacsonyabb prioritású (vagyis annál hátrább kerül) a feladat, Linux alatt pont fordítva (és ez a logikusabb megközelítés): a nagyobb counter érték magasabb prioritást jelent.

7. A 2.4.25-es Linux kernel ütemezője

A fentebb leírt igen korai (linux-1.2) kernel ütemezője az évek során nem sokat változott. Az aktuális 2.4-es kernel `sched.c`-jének fejléce szerint:

```
* 1996-12-23 Modified by Dave Grothe to fix bugs in semaphores and
*           make semaphores SMP safe
* 1998-11-19 Implemented schedule_timeout() and related stuff
*           by Andrea Arcangeli
* 1998-12-28 Implemented better SMP scheduling by Ingo Molnar
```

vagyis legutóbb 1998-ban történt benne lényegesebb módosítás. És valóban: az algoritmus lényege ugyanaz, a változások az 1.2-hez képest a következők:

- az SMP-támogatás miatti módosítások: zárolások, CPU-affinitás
- általános, az egész kernelt érintő módosítások átvezetése itt is
- valós idejű feladatok (real time process) támogatása

Úgyhogy valóban ráfért már egy alapos frissítés a Linux ütemezőre...

8. A Molnár Ingo-féle $O(1)$ ütemező

8.1. Miért volt szükség új ütemezőre?

- A régi ütemező eredetileg egyprocesszoros rendszerekre volt kitalálva: egy futási sorral rendelkezett, egyetlen hozzá tartozó retesszel (lock).
- Gyenge CPU-affinitással rendelkezett, emiatt a processzor-cache nem volt megfelelően kihasználva.
- $O(n)$ -osztályú ütemező – Sok feladat esetén gyenge performancia.

Összefoglalva: A régi ütemező nem, vagy legalábbis kevésbé volt skálázható.

8.2. Az új ütemező előnyei

- $O(1)$ – A működése független attól, hogy hány feladat fut a rendszerben Az SMP-rendszerek ütemezése jelentősen javult: a performancia és a skálázhatóság egyaránt sokkal jobb lett. Az ütemező döntési mechanizmusa is robusztusabb, mert az SMP figyelembevételével tervezték.
- Az interaktív feladatok kezelése is javult; ez az, amit a felhasználók leginkább észrevesznek. Az interaktív feladatokat egy különálló, a használatról készült statisztika alapján működő mechanizmus ismeri fel, amely teljesen le van választva a többi ütemező mechanizmustól, pl. az időszelvény kezelésétől.

A végeredmény: az interaktív feladatok még nagy terhelés esetén is „mozgékonyak”, a CPU-intenzív feladatok pedig sokkal jobban elkülönülnek az interaktívaktól, így nem tudják lekötni az összes CPU-erőforrást.

8.3. Hogyan működik az ütemező?

A korábbi ütemezők esetén ha el kellett dönteni, ki kerüljön a CPU-ra, a rendszer végigment a futási soron vagy a feladattáblán, megnézte, hogy kinek a legnagyobb a dinamikus prioritása és azt választotta. Ez így működött a különböző Unix és Linux kernelekben évtizedek óta. Ez az Ingo-féle ütemezővel drasztikusan megváltozott.

A Molnár Ingo-féle $O(1)$ ütemező esetén minden egyes processzorhoz külön futási sor (run queue) tartozik. Ez két részből áll:

- aktív tömb (active array)
- lejárt tömb (expired array)

Amikor egy feladat „felébred” (wake-up hatására), bekerül az aktív tömbbe, mégpedig közvetlenül a nála nagyobb vagy vele egyenlő prioritásúak után. Amikor „elalszik” (sleep), akkor kikerül a futási sorból, vagyis egyik tömbben sem szerepel. Amikor pedig egy feladat felhasználja a teljes időszelvényt, akkor az ütemező annak dinamikus prioritása alapján (lásd lentebb) dönti el, hogy átkerüljön-e a lejárt-tömbbe vagy esetleg kerüljön vissza az aktív tömbbe újbóli ütemezésre. Az aktív tömb tagjai egymás után végrehajtnak, prioritás szerinti sorrendben. A lejárt-tömb tagjai akkor kerülnek sorra, ha az aktív tömb kiürült, vagyis ha az összes eleme „aludni ment” vagy felhasználta az időszelvényt (lejárt); illetve egy bizonyos időkorlát (starvation limit) lejártja után.

Ebből látszik, hogy ez az ütemező valóban $O(1)$ típusú, hiszen mindig ugyanannyi ideig tart annak kiválasztása, hogy melyik feladat kerüljön a processzorra: csupán az aktív tömb legelső, legmagasabb prioritással rendelkező tagját kell vennünk.

8.4. Az interaktivitás-becslő (interactivity estimator)

Az interaktivitás-becslő feladata annak eldöntése, hogy mely feladatok interaktív, és melyek kötegelt jellegűek (batch tasks, „cpu-zabálók”). Ez azon a feltételezésen alapul, hogy a kötegelt feladatok soha nem várákoznak I/O-ra (soha nem „alszanak”), ehelyett az összes rendelkezésükre álló CPU-időt felhasználják; ellentétben az interaktív feladatokkal, amelyek gyakran várákoznak pl. arra, hogy a felhasználó beírjon valamit. Mindegyik feladathoz tartozik egy ún. `sleep_avg` érték, amely minden egyes idő-megszakítás (timer interrupt) idején eggyel növekszik, ha a feladat éppen „alszik” ill. eggyel csökken, ha éppen fut. A magas `sleep_avg` értékkel rendelkezők minősülnek interaktív feladatnak, az alacsonyak pedig kötegeltnek.

$$\text{Dinamikus prioritás} = \text{statikus prioritás} \pm 5$$

Az ütemező attól függően, hogy az interaktivitás-becslő interaktívnek vagy kötegettnak minősíti a feladatot, rendel hozzá dinamikusan prioritást: hogy azt hogyan számolja a feladat statikus prioritásából (amit a felhasználó adhat meg neki indításkor), az a `PRIORITY_BONUS` konstans értékétől függ. Ha a `PRIORITY_BONUS` értéke az alapértelmezett 25%, az azt jelenti, hogy a dinamikusan prioritás a statikustól maximum a teljes prioritástartomány $(-20 \dots +20)$ 25 százalékáig térhet el, vagyis ± 5 -tel: az interaktív feladatoké öttel nő, a kötegetteké öttel csökken.

Ha egy feladat a teljes rendelkezésére álló időszelvet felhasználja, az ütemező még mindig dönthet úgy, hogy ne járjon le; feltéve hogy még mindig interaktív feladatnak minősül (`TASK_INTERACTIVE`).

9. Az $O(1)$ további hangolása: A Kolivas-féle lépcsős (staircase) ütemező

Az ausztrál Con Kolivas módosításokat tett közzé az $O(1)$ -hez. Ezek a foltok (patches) az olyan alkalmazások minél hatékonyabb futtatására koncentrálnak, ahol észrevehető különbséget jelent, hogy megfelelő-e a késleltetés a között az időpont között, amikor a feladat „felébred” és a között, amikor ütemezésre kerül. Ez lényeges különbséget jelent, ami lassú egyre reakcióként, vagy pl. zenehallgatáskor szakadozóként jelenik meg. További cél a rendszer növekvő terheléssel szembeni ellenállásának javítása és a rendelkezésre álló processzoridő igazságosabb elosztása.

A szerző mérései szerint a javított $O(1)$ ütemező – amellett, hogy a fenti rövid késleltetést igénylő alkalmazások jobban viselkednek – általában ugyanolyan, időnként jobb eredményt produkál SMP-környezetben.

A Kolivas-foltokat az újabb 2.6-os kernel tartalmazza.

Hivatkozások

- [1] Bach, Maurice J.: The design of the Unix operating system
- [2] Beck, Böhm et al.: Linux Kernel Internals
- [3] Tannenbaum – Woodhull: Operációs rendszerek
- [4] Kernel források: <http://kernel.org>

Könnyen alkalmazható Linux biztonsági megoldások

Czakó Krisztián
<slapic@linux.co.hu>

Kivonat

Az előadás célja bemutatni olyan Linux alatt használható biztonsági megoldásokat, melyek alkalmazásához nincs szükség hosszas betanulásra, bonyolult szabályok készítésére. Az előadásban megpróbálok egy átfogó képet adni ezen megoldásokról, rövid összehasonlítással, valamint az egyes esetekben a továbblépés (bonyolult biztonsági rendszerek felé) lehetőségének felvázolásával. Olyan biztonsági megoldások kerülnek bemutatásra, mint a Grsecurity, PaX, valamint az Adamantix Linux, ami egy Debian alapú disztribúció, mely számos bemutatott megoldást eleve tartalmaz.

Tartalomjegyzék

1. Bevezetés	18
1.1. Biztonságos a Linux?	18
1.2. Főbb biztonsági kockázatok	18
1.3. Védekezési lehetőségek	18
2. Alkalmazható megoldások	19
2.1. PaX	19
2.2. GrSecurity	21
2.3. Adamantix Linux	25
2.4. Swap és fájlrendszer titkosítása	26
2.5. Mentések védelme	27
3. Továbblépési lehetőségek	28
3.1. LSM	28
3.2. RSBAC	28
3.3. SELinux, LIDS	28

1. Bevezetés

1.1. Biztonságos a Linux?

Ha az operációs rendszereket kell összehasonlítani, sokat halljuk, hogy a Linux milyen biztonságos. Ha a Linux biztonságáról beszélgetünk, folyton a biztonsági problémák hadáról esik szó. Vajon hol az igazság? A Linux valójában biztonságos vagy sem? Erre a kérdésre a legjobb válasz, hogy a Linux lehet biztonságos, de nem feltétlenül az. Az igazi válasz attól függ, milyen Linux disztribúciót alkalmazunk, és mennyire figyelünk oda a biztonságra. Hanyag rendszergazda alkalmazzon bármilyen operációs rendszert, nem fog biztonságos rendszerhez jutni.

Mi kell hát ahhoz, hogy a rendszerünk biztonságos legyen? Ha erre a kérdésre pontos választ akarunk adni, fel kell tenni a kérdést: mennyire legyen biztonságos? A közhelyek kihagyásával nézzük meg, mik a főbb biztonsági kockázatok.

1.2. Főbb biztonsági kockázatok

A biztonsági hibákat jellegük szerint két fő csoportba sorolhatjuk. Az első csoport a helyi biztonságra, a második a távoli (hálózati) biztonságra veszélyt jelentő hibák csoportja. A helyi biztonság kérdése, azaz a szerverünkre bejelentkezett felhasználók korában tartása elsőre nem tűnik annyira fontosnak, mint a távoli (hálózati) biztonságé, azaz a más gépekről érkező kéréseket kiszolgáló programok biztonsága. Ez az első olyan tévedés, mely később nem várt kellemetlenségekhez vezethet. A távolról kihasználható hibák jelentős része megelőzhető, ha a gépünk helyi biztonságára odafigyelünk. A biztonsági hibát a legtöbb esetben egy program nem várt, helytelen működése, azaz programozói hiba okozza. Ne feledkezzünk meg azonban a programozó szándéka szerinti, általunk nem várt működésről, azaz a trójai programokról se.

Egy biztonsági hiba másik jellemzője a rajta keresztül elérhető jogosulatlan hozzáférés mértéke. Ez legtöbbször a root felhasználó jogainak megszerzésének lehetősége vagy csak „sima” felhasználói jogokra korlátozódik, holott sokkal pontosabban illene definiálni, hogy mihez is lehet egy-egy hiba kihasználásával hozzáférni.

Napjaink mumusai a „buffer overflow” (puffer túlcsordulás) és a DoS¹. Kezdő rendszergazdák ijesztgetése mellett ezek igen komoly fejfájást is tudnak okozni.

Ugyan keveset hallani mostanában az adatok biztonságáról, de ez is alapvető része rendszerünk általános jólétének. Az adatok megsemmisülése elleni védelem nem tartozik ezen előadás keretei közé², így maradjunk a lopás elleni védelemnél. A működő rendszer kockázata mellett itt fontos szerepet kap az adattároló eszközök védelme. Hiába készítettünk „atombiztos” hálózati védelmet, ha a gépünket ellopják és bizalmas adataink rossz kezekbe kerülnek. Sajnos ezzel a kockázattal jelenleg még igen kevesen törődnek.

1.3. Védekezési lehetőségek

A védekezés első lépését már megtettük, felmértük a kockázati tényezőket. Most lássuk, milyen lehetőségeink vannak a kockázat csökkentésére. A távoli és helyi, programozói hibából eredő kockázatok jelentősen csökkenthetők, ha a felesleges jogokat

¹Denial of Service, azaz szolgáltatás megtagadás

²Ezzel kapcsolatban lásd *Deim Ágoston*: „Központi mentési eljárások” című cikkét. (A szerk.)

megvonjuk a programoktól, valamint működésüket kontrolláljuk. Erre alkalmas technikák illetve programok többek között a PaX [1], az SSP [2], a GrSecurity [3], az RS-BAC [4], a LIDS [5], a SELinux [6] – és még sorolhatnánk. Ne gondolja senki, hogy ezeket sorban feltelepíti és szuper biztonságos rendszere lesz. Ezek közül számos megoldás kizárja vagy értelmetlenné teszi a másikat vagy épp tartalmazza annak funkcióját. A lista utolsó három tagját az említésen túl nem tárgyaljuk, ugyanis alkalmazásuk direkt ellentmondásban van az előadás címének egyszerű megoldásokat ígérő mivoltával. Ezeken kívül alkalmazhatunk olyan technikákat, melyek megnehezítik rendszerünk kiismerését, így nehezítik a támadásokat. Erre jó példa a véletlenül variált PID (processz azonosító) illetve IP port valamint a véletlenszerű memóriakiosztás.

A trójai programok elleni védekezés elsősorban víruskeresőkkel, valamint a futási jogok pontos definiálásával lehetséges. Utóbbi bonyolult szabályokat igényel, így mi maradunk az első variációnál, illetve a TPE (Trusted Patch Execution) megoldásnál, mely a GrSecurity része.

Az adatok lopásvédelme már érdekesebb kérdés. Az egyik legjobb módszer, amivel egy gépen nem tárolt, de futás közben bevitt információkhoz juthatunk (mint más szerverek jelszava) a swap partíció adatainak elemzése. Ehhez nem kell más, mint megszerezni a kérdéses gép merevlemezét. Ez ellen egyrészt élő erős védelemmel lehet tenni. Ez jól alkalmazható szervertermek védelme esetén, de egy otthoni gép (és ki ne dolgozna otthonról?) vagy egy hordozható gép esetén már nem triviális. Keressünk inkább más megoldás. Gyorsan adódik a swap és más lemezterületek titkosításának lehetősége.

2. Alkalmazható megoldások

2.1. PaX

A PaX [1] nyújtja jelenleg a leghatékonyabb védelmet a túlsordulásos hibákat kihasználó támadásokkal szemben. A PaX alapvetően két dolgot biztosít, ami az alap Linux kernelben nincs, és az x86 platform hardveresen nem nyújt (sajnos a Linux ott sem használja, ahol a platform ezt nyújtja): nem-futtatható memória lapok és teljes címtér kiosztás véletlenszerűsítése ³ a platformok nagy variációján. Ezen védelmek megvalósítására többféle technika létezik, és a PaX-on kívül számos (általában gyengébb) megoldások léteznek, mint a W^X (OpenBSD) vagy a RedHat által (konkrétan Molnár Ingo által) fejlesztett Exec Shield. A PaX technikái a következők:

SEGMEXEC

Ez a technika az oldalankénti nem-futtatható felhasználói oldalak technikáját alkalmazza az x86 architektúra szegmentációs logikájára alapozva virtuális memória tükrözéssel. Különválasztja az adat szegmenst (DS) és a kód szegmenst (CS). Ezáltal két szegmens létezik a felhasználói oldalakban és a kernel oldalakban is. A PaX megfelel a címtérre: az alsó fele az adaté, a felső fele a programkódé. Ez nem okoz teljesítmény csökkenést, de 3 GB-ról 1,5 GB-ra csökkenti a program által elérhető memóriát. A VMA⁴ tükrözés eredményeképpen az alsó adat tér összes futtatható lapját tükrözzük a felső adattérbe. Az összes olyan utasítás lekérés az adat szegmensből, melyhez

³ASLR, azaz Address Space Layout Randomization

⁴VMA = Virtual Memory Allocation

nincs tükrözött kód a kód szegmensben lapozási hibát eredményez. A PaX kezeli ezt és leállítja a processzt.

PAGEEXEC

Ez a technika volt a PaX első megoldása a nem futtatható lapokra. A SEGMEXEC megjelenése óta ez x86 rendszereken már nem használatos. Azon platformokon, ahol a futtatható bit hardver szinten implementált, az a PAGEEXEC technikával van megvalósítva.

A PAGEEXEC célja, hogy a nem futtatható lapokat implementálja az IA-32 alapú processzorok lapozási képességére építve. Hagyományosan a lap védelem a CPU memória kezelő egységének képességeit használja, de az IA-32 architektúrán ez a képesség hardver szinten hiányzik, azaz lehetetlen egy lapot futtathatónak vagy nem futtathatónak jelölni a lapozáshoz tartozó struktúrákban. Ami mégis lehetővé teszi a technika alkalmazását, hogy a Pentium valamint AMD K7 és újabb processzorok osztott TLB-vel⁵ rendelkeznek a kód és adat számára (az AMD K5-ben lévő implementáció nem alkalmas a célra). A TLB szerepe, hogy gyorsítótárként működjön a virtuális/fizikai címek translációjához, amit a CPU-nak minden egyes memória elérésnél el kell végeznie. A TLB nélkül a processzornak minden alkalommal végig kellene lépegetnie a laptáblán, ami jelentős lassulással járna.

Az osztott TLB miatt az utasítás lekérés az ITLB (Instruction TLB), minden más a DTLB (Data TLB) használatát eredményezi. Emiatt a TLB betöltésen teljes szoftveres kontrollt lehet gyakorolni: értesítést kaphatunk a hardveres TLB betöltésekről ha a lap táblát úgy állítjuk be, hogy minden ilyen kísérlet lapozási hibát eredményezzen, valamint TLB betöltést idézhetünk elő, ha megfelelő memória-hozzáféréssel a processzort a laptáblák végigjárására és TLB betöltésre kényszerítjük. Ez a kulcsa a nem futtatható lapok implementálásának. Egy-egy lap megjelölhető nem létezőnek vagy csak a supervisor számára hozzáférhetőnek a laptáblákban. Utóbbi felhasználói hozzáférés esetén lapozási hibát eredményez. A lapozási hibakezelő eldöntheti, hogy utasítás lekérés vagy legális adathozzáférés történt-e. Az első esetben egy futtatási kísérletet detektáltunk egy nem futtatható lapon, amit a feladat leállításával meggátolhatunk. A második esetben ideiglenesen módosítjuk az érintett lap tábla bejegyzést oly módon, hogy a felhasználói szintű hozzáférést engedélyezzük, és a CPU betöltse azt a DTLB-be. A lapozási hiba kiváltására a supervisor módot alkalmazza a PaX, mivel az nem okoz ilyen eseményt akkor, amikor a kernel próbál felhasználói adathoz férni, így csak kisebb teljesítménycsökkenéssel jár.

Amint az implemetációból jól látszik, ez a technika a teljesítmény csökkenésével jár. Ezért x86 rendszereken inkább a SEGMEXEC megoldás javasolt, aminek ilyen mellékhatása nincs.

KERNEXEC

A KERNEXEC a lap védelmet látja el a kernelben. Az alkalmazott technika gyakorlatilag a SEGMEXEC-nél leírtakkal azonos. Ezen felül pár „aprósággal” növeli a biztonságot: a 'const', a rendszer hívás tábla, a megszakítás leíró tábla (IDT) és a globális leíró tábla (GDT) csak olvasható a kernelben, valamint az adatok nem futtathatók. A dokumentáció szerint jelenleg a modul támogatással együtt⁶ nem működőképes.

⁵TLB: Translation Lookaside Buffer

⁶Magasabb biztonsági szint eléréséhez modul támogatás nélküli kernel használata egyébként is javasolt.

ASLR

Az ASLR, azaz Address Space Layout Randomization célja, hogy a memóriacímek véletlenszerű átrendezésével nehezítse egy támadás sikerességét. A technika kizárólag ELF binárisokkal működik, és pozíciófüggetlen kódot igényel (PIC), amit a bináris fordításakor kell biztosítani, azaz használatához számos programot újra kell fordítani és linkelni, nem elég a kernel támogatás bekapcsolása (az Adamantix [7] és a Hardened Gentoo [8] disztribúciók tartalmazzák ezt a változtatást). Az ET_EXEC ELF binárisok a RANDEXEC eljárással, az ET_DYN ELF binárisokat a RANDMAP eljárással lehet véletlenszerű címekre helyezni.

Az ASLR véletlenszerűvé teszi a következő memória objektumok elhelyezkedését: futtatható bináris (executable image), megszakítás vezérelt munkaterület (Break-managed heap), dinamikus könyvtár (library image), mmap által kezelt munkaterület (mmap-managed heap), felhasználói verem (user space stack), kernel verem (kernel space stack). A 32 bites rendszereken a véletlenszerűsítés a következőképpen néz ki: 24 bit, mmap 16 bit, futtatható 16 bit, munkaterületek 12 bit. A véletlenszerűsítés függetlenül történik ezen területekre, így egy támadási kísérlet szinte teljesen reménytelenné válik, hiszen egymástól független, ismeretlen területeket kell egyidejűleg támadnia. Például ha a könyvtárakat (libraries) és a vermet egyszerre kell elérni, a két terület címének bitjeit egyszerre kell kitalálni, ami egy 40 bites véletlenszám kitalálását követeli meg. Fix címen alapuló támadás sikerének esélye ilyen szituációban gyakorlatilag nulla.

A RANDKSTACK funkció véletlenszerűsíti a kernel veremterületét. Mivel a véletlenszerűsítés minden egyes rendszerhívásnál megtörténik, egy már futó kód vizsgálatával nem lehet olyan információhoz jutni, mely elősegítené egy később indított kód ellen irányuló támadást. A verem 5 bitjét teszi véletlenszerűvé. A nyers erő módszere általában nem működik, hiszen szinte minden kísérlet a kernel összeomlásához vezet.

A RANDEXEC funkció véletlenszerűsíti az ET_EXEC binárisok elhelyezkedését a memóriában a SEGMEXEC-nél ismertetett funkciók segítségével. A RANDEXEC elsősorban koncepciót bizonyító megvalósítás. Az alkalmazott technika a RANDMAP megoldás (ET_DYN/PIC binárisok alkalmazása).

2.2. GrSecurity

A GrSecurity [3] egy könnyen alkalmazható gyűjteménye számos olyan megoldásnak, melyek növelik a rendszer belső és külső biztonságát. Egyetlen patch-ként telepíthető a kernelre. Beállításai között vannak előre definiált változatok, melyek mélyebb ismeretek nélkül is gyorsan alkalmazhatók, míg lehetőség van a funkciók egyenkénti beállítására is a kernel konfigurálásakor. Az alábbiakban ismertetem a GrSecurity főbb komponenseit.

RBAC

Egyszerű Role-Based Access Control. A Linux DAC (Discretionary Access Control) rendszere mellett egy nagyobb biztonság lehetőségét adja a szabály-alapú hozzáférés vezérlés. Mivel ennek alkalmazása véleményem szerint túlmutat a „könnyen alkalmazható” feltételen, részletes ismertetésébe nem megyek bele. Megjegyezném azt is, hogy az RSBAC (rövid ismertetését lásd később) rendszer sokkal részletesebb és hatékonyabb hozzáférés-vezérlést tesz lehetővé, igaz azt még nehezebb jól beállítani.

chroot

A chroot technika önmagában már létezik a Linuxban. Célja, hogy egy-egy program futtatásakor a látható fájlrendszert korlátozzuk egy könyvtárra, így gátolva a futó programot abban, hogy olyan helyeket is elérjen, melyeket nem kellene. Sajnos az eredeti implementáció számos gyengeséggel rendelkezik. Például nem korlátozza az így indított program jogait és képességeit (capabilities). Így ha ez a program root joggal fut, képes csatolni fájlrendszereket, vagy ismételt chroot hívással kilépni börtönéből. Ezt és hasonló gyengeségeket hivatott kiküszöbölni a GrSecurity. A korlátozásokat a kernel beállításakor határozhatjuk meg. A következő lehetőségeket választhatjuk:

- A chroot-on kívüli osztott memória elérésének tilalma.
- Szigonók küldésének tilalma (kill) a chroot-on kívülre.
- Ptrace tiltása chroot-on kívülre.
- Capget tiltása chroot-on kívülre.
- Setpgid, getpgid és getsid tiltása a chroot-on kívülre.
- Szigonók küldésének tilalma fcntl segítségével.
- A chroot-on kívüli processzek láthatatlanok akkor is ha a /proc csatolva van.
- Csatolás (mount) és újracsatolás (remount) nem lehetséges.
- pivot_root (rendszer root könyvtár megváltoztatása) nem lehetséges.
- Újabb chroot hívás nem lehetséges.
- A chroot-on kívülre nem lehet fchdir hívást kérni.
- chdir(„”) erőszakolása a chroot híváskor.
- Nem lehet setuid bitet beállítani (chmod +s és fchmod +s tiltása).
- Nem lehet eszközfájlokat létrehozni (mknod tiltása).
- Nem lehet a kernelt vezérelni (sysctl írásának tiltása).
- A processz prioritásának növelése nem lehetséges.
- A chroot-on kívüli abstract unix domain socketekhez nem lehet kapcsolódni.
- Számos veszélyes jogosultság megvonása a képességek (capabilities) megvonásával.
- Minden futtatás naplózása a chroot-on belül.

PaX

A GrSecurity tartalmazza a komplett PaX implementációt, így mindaz amit a PaX-nál írtam igaz itt is.

Auditálás

A védekezés egyik legfontosabb része az auditálás. Segítségével tudomásunkra juthatnak betörési próbálkozások vagy más korlátozás kijátszásának kísérletei. A GrSecurity néhány alapvető auditálási lehetőséget biztosít:

- Kijelölhető egy csoport az auditálásra.
- Programfuttatások és argumentumok naplózása.
- Tiltott erőforrás kérések naplózása.
- Könyvtárváltás naplózása.
- Csatolás (mount) és leválasztás (umount) naplózása.
- IPC létrehozás/megszüntetés naplózása.
- Szignálok naplózása.
- Sikertelen fork() hívások naplózása.
- Idő változtatás naplózása.

Véletlenszerűsítés

A véletlenszerűsítés már szóba került a memóriával kapcsolatban. A GrSecurity tartalmaz további véletlenszerűsítő kódot is, mely a kommunikáció (tcp/ip, rpc) során változtat bizonyos adatokat véletlenszerűen. Ezek segítségével nehezebben lehet kideríteni rendszerünkről a hálózaton keresztül, hogy milyen OS-t futtat, illetve számos tcp/ip támadás válik nehezebbé.

Általában egy-egy operációs rendszerre jellemző az IP csomagjának felépítése. Így van ez a Linux esetében is. Az IP azonosítók véletlenszerű megváltoztatása ezt kevésbé egyértelművé teszi. Egy tcp-kapcsolattal is nehezebb visszaélni, ha az induló csomag sorszám és a forrásport is véletlenszerű. Alapesetben ha egy gép tcp kommunikációját valahol figyeljük, ki tudjuk számítani, hogy a következő kimenő kapcsolata melyik portról indul. Ezzel számos esetben vissza lehetne élni. A véletlenszerűsítés ezt megakadályozza.

Az RPC XID-k véletlenszerűvé tétele hasonló előnyökkel jár. Ugyanezt mondhatjuk a PID-ek (processzek azonosítója) esetén is. A Linux a PID-eket is sorban osztja, míg a GrSecurity alkalmazása esetén ez is véletlenszerű.

Egyéb tulajdonságok

A fentiekén túl találunk sok hasznos változtatást, melyek nem sorolhatók be egy-egy nagyobb kategóriába. Ilyen a /proc használatának korlátozása. Beállíthatjuk, hogy a kernel processzek ne legyenek láthatóak, a felhasználók ne lássák egymás processzeit (így azokról információhoz sem juthatnak). Lehetőségünk van korlátozni a szimbólikus és hard linkek alkalmazását a /tmp versenyhelyzetek elkerülésének érdekében, korlátozható a FIFO használat, felhasználó nem tudja lekérdezni a kernel üzeneteit (dmesg).

Érdekes lehetőség a TPE, azaz a megbízható elérési út futtatás (Trusted Path Execution). Beállíthatjuk, hogy bizonyos felhasználók csak egy előre magadott könyvtárban vagy könyvtárakban található fájlokat futtathatják.

Korlátozhatjuk a socketek, így a hálózat elérését csoport (GID) alapján, például az internet elérését csak bizonyos felhasználóknak engedjük meg.

A GrSecurity alapbeállításai

Alacsony biztonság	Közepes biztonság	Magas biztonság	Egyedi biztonság
Kevés biztonsági kiegészítés, szinte biztosan nem lesznek inkompatibilitási problémák	Több biztonsági korlátozás, várhatóan kevés inkompatibilitás jelentkezik	Majdnem minden biztonsági beállítás bekapcsolva, számos régi vagy rosszul megírt programmal jelentkezhet inkompatibilitás. Ez a szint bekapcsolja a PaX-ot, mely a problémák jelentős részét okozhatja. Ezeket általában a chpax programmal orvosolhatjuk.	Nincsenek előzetes beállítások, minden egyedileg állítható be
<ul style="list-style-type: none"> - linking restrictions - fifo restrictions - random pids - enforcing nproc on execve() - restricted dmesg - random ip ids - enforced chdir("/") on chroot 	<p>Az alacsony szinten leírtak mellett:</p> <ul style="list-style-type: none"> - random tcp source ports - failed fork logging - time change logging - signal logging - deny mounts in chroot - deny double chrooting - deny sysctl writes in chroot - deny mknod in chroot - deny access to abstract AF_UNIX sockets out of chroot - deny pivot_root in chroot - denied writes of /dev/kmem, /dev/mem, and /dev/port - /proc restrictions with special gid set to 10 (usually wheel) - address space layout randomization - removal of addresses from /proc/<pid>/maps/stat 	<p>Az alacsony- és közepes szinten leírtak mellett:</p> <ul style="list-style-type: none"> - additional /proc restrictions - chmod restrictions in chroot - no signals, ptrace, or viewing processes outside of chroot - capability restrictions in chroot - deny fchdir out of chroot - priority restrictions in chroot - segmentation-based implementation of PaX - mprotect restrictions - kernel stack randomization - mount, unmount, remount logging - kernel symbol hiding 	<ul style="list-style-type: none"> - Trusted Path Execution (TPE) beállítható - Socket korlátozások beállíthatók - Sysctl lehetőség bekapcsolható (GrSecurity funkciók ki- és bekapcsolása a sysctl rendszeren keresztül)

A GrSecurity beállítása

A GrSecurity úgy a 2.4-es, mint a 2.6-os kernelhez elkészül, és a projekt weboldaláról [3] letölthető. A letöltött patch általában csak a Linus-féle eredeti kernel-forrásra [9] tehető fel, más kiegészítésekkel gyakran ütközik. A patch telepítését követően a kernel szokásos konfigurálásakor külön menüpontként jelentkezik. Ha bekapcsoljuk, négy alapvető választási lehetőséget kínál. Az első három – paranoiánk függvényében – különböző szintű előre beállított „csomag”, míg a negyedik teszi lehetővé az összes lehetőség kézi beállítását. Utóbbit csak gyakorlott szemeknek javaslom, azonban bonyolultabb esetekben elkerülhetetlen lehet, ha mindent működőképesen akarunk tartani. Aki először találkozik a GrSecurity-vel, érdemes megpróbálnia a magas (high) biztonsági szintet. Ez majdnem minden (de nem minden!) védelmet bekapcsol, és az esetek jelentős százalékában működő rendszert eredményez. A kivételek főleg azon kereskedelmi programoknál van, melyek forráskódja nem áll rendelkezésünkre. Ezeknél főként a PaX által adott védelmek okoznak gondot. Utóbbi a chpax programmal sokszor orvosolható, de vigyázzunk, mert a chpax az információt beleírja magába a programba, így az módosul (integritás ellenőrzőkkel kombinálva okozhat téves riasztásokat, illetve én találkoztam kereskedelmi víruskeresőkkel is, amik viszont beépített integritás ellenőrzőjük miatt egy ilyen módosítást követően nem működtek). Ha a chpax nem segít, nincs más megoldás, mint a GrSecurity egyedi beállítása.

2.3. Adamantix Linux

Az Adamantix [7] egy Debian alapú disztribúció, mely azt a célt tűzte maga elé, hogy biztonságosabbá tegye a rendszert. Ezt meglévő biztonsági megoldások alkalmazásával éri el.

Az Adamantix alapja eredetileg a Debian Woody (3.0) disztribúció volt, az abban lévő csomagok módosításával kezdődött a munka. Azóta számos csomag már sokkal újabb, mint a Woody-ban található változata, így az újdonságokról sem kell lemondani, azaz az Adamantix nem a Debian aktuális stabil verziójának módosítása, hanem immáron egy önálló disztribúció, mely természetesen használja a Debian csomagjait, de a saját szükségleteinek megfelelően módosítva.

A rendszer teljesen „kompatibilis” a Debiannal. Ugyanazt a debconf rendszert, apt és dpkg csomagkezelőt alkalmazza, ugyan ott vannak a beállítások. Így ha valaki jártas a Debianban, azonnal elkezdhet dolgozni az Adamantix terjesztéssel. Olyannyira kompatibilisek, hogy egy Debian Woody frissíthető Adamantixra (Sarge még nem, mert az Adamantixban még a régebbi libc van).

Az egyik legfontosabb kiegészítés a PaX alkalmazása. A kernel patch feltételén túl az összes bináris ET_DYN formátumúra linkelve kerül a csomagba és a PIC opció is szerepel a fordításnál ahol csak lehet. Így a PaX ASLR technikája teljes körűen működik. Ezt kiegészíti az SSP [2] (Stack Smashing Protector), mely a C fordító (GCC) által biztosított védelem (a kernel is ezzel az opcióval kerül lefordításra). A következő kernel kiegészítések kerültek az Adamantixba:

- A Debian által alkalmazott kiegészítők,
- RSBAC (opcionális),
- PaX,
- MPPE (kódolt PPP),

- véletlenszerű PID-ek és IP portok,
- PaX obscurity patch, mely az ASLR-t még hatékonyabbá teszi,
- gyors hálózati eszköz keresés,
- a kernel fordítása SSP-vel,
- transzparens proxy (cttproxy) támogatás,
- a hálózati eszközök hozzáadásra kerületek a kernel entrópia-pooljához,
- XFS fájlrendszer támogatás,
- vezeték nélküli hálózatok kiszolgálása a Host-AP driverrel,
- AES-loop.

Az Adamantix három kernelt készít különböző képességekkel. A „normal” változat tartalmazza a Debian kiegészítéseit, bekapcsolt RSBAC MS (malware scan) támogatást ClamAv [10] használatával, teljes PaX támogatást az ET_EXEC véletlenszerűsítés kivételével, transzparens proxy és MPPE támogatást, hálózati eszközök hozzáadása a kernel entrópia-pooljához kikapcsolva, CAP processz elrejtés bekapcsolva, Host-AP driver bekapcsolva, XFS bekapcsolva. A „softmode” kernel teljes RSBAC támogatással, de az RSBAC soft-mód támogatásának bekapcsolásával, szabályok kapcsolása SysRq-val. A REG modul kikapcsolva, CAP processz elrejtés bekapcsolva, RES module (az AUTH védelemmel együtt) bekapcsolva. A „secure” kernel az RSBAC támogatást jelenti soft-mód nélkül és a szabályok nem kapcsolhatók még SysRq-val sem.

2.4. Swap és fájlrendszer titkosítása

Amint azt a bevezetőben említettem, adataink védelmére egy jó lehetőség, ha a háttértáron azok titkosítva kerülnek rögzítésre. A legfőbb kockázati tényező a swap, azaz a lapozóterület, ahová a memória tartalma kerülhet ki, és belátható, hogy ez jelentheti kényes információk tárolását is, mint jelszavak és más hitelesítő információk. A futó rendszeren a swap védhető megfelelő hozzáférés kontrollal, de a diszkek illetéktelen kézbe kerülésekor a rendszer leállásakor a swap-en tárolt adatokat semmi sem védi meg. A titkosítás ezen segít. Ráadásul a swap tartalmára két újraindítás között nincs szükség, így a titkosításra használt kulcsot nem kell sehol tárolni. Ez lehetővé teszi, hogy a rendszer indulásakor a swap partíciót vagy kötetet véletlenszerűen generált kulccsal titkosítsuk, és azon készítsünk swap területet, amit aktiválunk. Így a kulcsot nem ismerheti senki, az a rendszer leállásakor végleg elvész. A swap-re került adatok visszaállítására ekkor az egyetlen lehetőség a nyers erő módszere, mely kellően nagy kulcs alkalmazásakor igen hosszadalmassá, így szerencsés esetben érdektelenné válik (várhatóan a visszafejtés befejeződésére az onnan nyert jelszavak már megváltoztak).

Partíciók illetve kötetek közvetlen titkosítására több technikai megoldás is létezik. Ilyenek a cryptoloop, aes-loop és a dm-crypt. Én az utóbbit javaslom. A dm-crypt a device-mapper rendszerét használja, mely a 2.6-os kernelek része, 2.4-es kernelhez patch formájában létezik. Ha valaki már dolgozott LVM-el vagy EVMS-el, a logikát már ismeri. A 2.6-os kernelben mind az LVM 2-es verziója, mind az EVMS igényli a device-mappert. A 2.4-es kernel esetén LVM 1-es verzióról könnyedén lehet áttérni a device-mappert használó 2-es verzióra. A device-mapper részeként találkozhatunk a kernel beállításakor a dm-crypt modullal. Ha ezt bekapcsoltuk, a cryptsetup programra

lesz még szükség, melyet letölthetünk [11], illetve a Debian pool-ban megtalálható. Ez az egyszerű kis parancs egy meglévő partíciót vagy kötetet titkosít a parancssorban vagy jelszó promptban megadott kulccsal, és létrehoz egy szintén általunk megnevezendő új eszközt a `/dev/mapper/` alatt. Ezen új eszközt kell `mkswap-el` formáznunk majd a `swapon` parancsal a `swap-et` aktiválnunk. Ha a `cryptsetup` programnak a kulcsot a Linux véletlenszám-generátorától kértük, a titkosító kulcs minden rendszerindításkor más lesz (ezért kell minden esetben az `mkswap` parancs újbóli futtatása is). A `cryptsetup` nem foglalkozik a megadott valódi eszközön található adatokkal, mindössze bekapcsolja az adott kulccsal a titkosítást. Ha egy már titkosított eszköz újbóli olvasását kíséreljük meg, rossz kulcs megadásakor hibás lesz a visszakódolás, így a fájlrendszer nem lesz olvasható. Ha a jó kulcsot adjuk meg, a fájlrendszerünk előkerül és tudjuk csatlakoztatni. Ezzel máris eljutottunk a fájlrendszer titkosításának módjához. Ezúttal a véletlen generált kulcs nem jó, hiszen a fájlrendszerre újból és újból szükségünk lesz. Esetleg még a `/tmp` illetve a `/var/tmp` titkosítható véletlenszerűen, hiszen ezeken normálisan nem tárolunk permanens információkat.

A fájlrendszerek titkosítása felveti azt a problémát, hogy hol és hogyan tároljuk a kulcsot. Ha a rendszer indításához szükséges adatokat nem védjük titkosítással, akkor nincs probléma. Az egyes felhasználók adatait külön-külön kötetben vagy fájlban tárolt fájlrendszeren tárolhatjuk, és pl. a felhasználó belépésekor a PAM rendszert segítségül hívva automatikusan csatlakozhatjuk. Ha a rendszert magát szeretnénk megvédeni, akkor érdemes az összes kötetet vagy partíciót titkosítani. A dekódoláshoz szükséges kulcsot pedig hordozható eszközön (pl. USB memória, CD, stb.) tárolni, és csak a rendszer indításakor betenni a számítógépbe. Esetleg fejben tárolt jelszót alkalmazni a célra. Elegendő a gyökér (`/`) fájlrendszer csatlakozásához szükséges kulcsot megjegyezni, mivel a többi kulcs tárolható a fájlrendszerben, hiszen az titkosítva van, így a rendszer indításakor csak ezt az egy kulcsot kell megadni.

2.5. Mentések védelme

A gondos rendszergazda rendszeresen végez mentéseket a rendszeréről. A mentések túlnyomórészt hordozható eszközökre (DAT, DLT, CD, DVD, stb.) készül, és azokat nem a szerver mellett tároljuk biztonsági okokból. Ez egy újabb lehetőséget nyit az adatok ellopására. Ezért érdemes a mentésre szánt adatokat még a mentendő gépről történő kikerülése előtt titkosítani. Bonyolult mentési megoldások általában külön számítógépet használnak nagy kapacitású háttértárral (cache) és mentő eszközzel (többnyire DLT). Egy ilyen mentést koordináló szerverre nem szerencsés, ha a különböző szervereken más-más bizalmassági szinttel jelölt adat kódolatlanul érkezik. Itt is érdemes tehát kódolni, de hogyan? Egy jó lehetőség, ha az adatokat azok bizalmassági címkéjétől függően más-más pgp kulccsal kódoljuk. A pgp két kulcsos rendszere lehetővé teszi, hogy a mentett szerver csak a nyilvános kulcsot tartalmazza, így a dekódolásra ő már nem képes. A dekódolásra használható kulcsok fokozott védelmét természetesen meg kell szervezni. Érdemes azokat több példányban CD-n vagy más, adatokat hosszú távon stabilan őrző médiumon tárolni. Ezen médiumok fizikai védelmet igényelnek, mely lehet egy páncélszekrény vagy éppen egy bank széfje.

3. Tovább lépési lehetőségek

3.1. LSM

Az LSM az új 2.6-os kernel sorozatban jelent meg. Célja, hogy egységes felületet biztosítson a biztonsági kiegészítések kernelbe illesztéséhez. Így az egyes biztonsági megoldások egymás mellett is létezhetnek, míg előtte ezek olyan módosításokat voltak kénytelen a kernel-forrásban végezni, melyek ütköztek egymással. Az LSM-mel egyidejűleg a SELinux is bekerült a kernel forrásba kihasználva az LSM nyújtotta lehetőségeket. Az LSM megítélése a biztonsági kiegészítéseket készítő fejlesztők körében erősen megosztott. A fő ellenérv az LSM-mel szemben az, hogy nem csak a biztonsági kiegészítéseket engedi a kernellel együttműködni, hanem új távlatokat nyit a rootkitek készítői előtt is, és ez sajnos komoly probléma.

3.2. RSBAC

A Rule Set Based Access Control [4] az egyik legsikeresebb biztonsági rendszer Linux-hoz. Segítségével többféle biztonsági modell alkalmazása válik lehetségessé, melyek egy alap Linux kernellel nem lennének lehetségesek. Az RSBAC cseppet sem sorolható a könnyen alkalmazható megoldások közé, bár az MS (malware scan) modulja még ide sorolható. Így itt csak röviden álljon a lista, minek az ellenőrzésére képes:

- MAC: Bell – LaPadula mandatory access control.
- FC: functional control.
- SIM: security information modification.
- PM: privacy model.
- MS: malware scan.
- FF: file flags.
- RC: role compatibility.
- AUTH: authorization enforcement.
- ACL: access control list.
- CAP: Linux capabilities.
- JAIL: process jails (chroot helyett alkalmazható).
- RES: Linux resources.
- PAX: teljes PaX támogatás.

3.3. SELinux, LIDS

Az említetteken kívül számos többé-kevésbé sikeres megoldás született a Linux biztonságosabbá tételére. A SELinux [6] az RSBAC [4] fő riválisa. Használata kezdetben egyszerűbbnek látszik, bonyolultabb szituációkban nehezebbé válik a használata. Kezdőknek nem javaslom. A LIDS [5] egy szintén közismertebb kezdeményezés, bár a minőségéről megoszlanak a vélemények.

Hivatkozások

- [1] <http://pax.grsecurity.net/>
 - [2] <http://www.trl.ibm.com/projects/security/ssp/>
 - [3] <http://www.grsecurity.net/>
 - [4] <http://www.rsbac.org/>
 - [5] LIDS: Linux Intrusion Detection System:
<http://www.lids.org/>
 - [6] NSA Security-enhanced Linux
<http://www.nsa.gov/selinux/>
 - [7] <http://www.adamantix.org/>
 - [8] <http://www.gentoo.org/proj/en/hardened/>
 - [9] <http://www.kernel.org/>
 - [10] <http://www.clamav.net/>
 - [11] <http://www.saout.de/misc/dm-crypt/>
-

Központi mentési eljárások

Deim Ágoston

<ago@lsc.hu>

Kivonat

Az előadás átfogó képet kíván nyújtani a Linux disztribúciók alatt elérhető központi mentési lehetőségekről, mind szabad szoftveres megoldásokkal, mind kereskedelmi termékekkel. Egy projektet kiemelve részletesen bemutatok egy szabad szoftveres megoldást. Az előadás célja, hogy megmutassa, a vállalati szektorban fontos mentési stratégiák szabad szoftveres módon is kialakíthatók.

Tartalomjegyzék

1. A mentés fontossága	33
2. Mentési stratégiák, módszerek	33
2.1. A mentések gyakorisága és időpontja	33
2.2. Mentés típusa és tartalma	34
2.3. Mentési módszerek	34
2.4. Mentések biztonsága	35
2.5. Helyreállítás	35
3. Hardver alrendszerek	36
3.1. szalagos meghajtók (TAPE)	36
3.2. diszk–diszk	36
3.3. VDL (VTL)	36
3.4. NDMP: Network Data Management Protocol	37
4. Mentőszoftverek követelményei	37
4.1. Platform támogatás	37
5. Kereskedelmi szoftverek	38
5.1. TIVOLI	38
5.2. Veritas	38
5.3. Arkeia	38
5.4. Atempo	38
5.5. BakBone NetVault	38
5.6. BRU	39
5.7. LoneTar	39

6. Szabad szoftverek	39
6.1. „Made by hand”	39
6.2. DAR – Disk ARchiver	39
6.3. Bacula	40
7. Bacula – részletesen	40
7.1. A Bacula felépítése	40
7.2. Amire figyelni kell	41
8. Összefoglalás	42

1. A mentés fontossága

Az informatika középpontjában továbbra sem áll más, mint az adat. Az adatokból élünk, ezekből dolgozunk, még a tudományos kutatásoknál elvégzett számítások adatait is további számításokhoz használjuk. De adatok a számunkra fontos levelek, feljegyzések, szerződések és egyéb információt hordozó fájlok is. Az informatikai rendszereknek folyamatos fejlődése mellett az adatok is egyre nagyobb számban jelennek meg. Ma már sokkal több adatot tudunk tárolni és feldolgozni, azaz több adattal tudunk dolgozni: több kimutatást tudunk készíteni, sokkal pontosabb pénzügyi „jóslásokat” és számításokat tudunk elvégezni és még sorolhatnánk. Ez határozza meg a legnagyobb sérülékenységi pontot is egy informatikai rendszerben, mely nem más, mint szintén az adat. Ezért, szinte már a kezdetektől, de az informatika pénzügyi felhasználásától kezdve mindenképpen az adatok biztonsága és biztonságos tárolása kiemelt helyen szerepelt egy informatikai rendszerben. Az adatokra vonatkozó biztonsági szabályzatok széles területet ölelnek fel, az adatok bizalmasságától és integritásától kezdve a biztonságos tárolásig és az adatmentésig. Ez az írás az adatmentéssel és visszaállítással foglalkozik: a fogalmakkal, módszerekkel és a megfelelő szabad szoftveres valamint kereskedelmi megoldásokkal.

2. Mentési stratégiák, módszerek

A mentési stratégia több elemből épül fel:

- milyen gyakorisággal és mikor mentünk
- milyen típusú a mentés
- hogyan használjuk fel a tároló eszközöket (rotáció)
- hogyan, milyen módon tároljuk az adatokat
- milyen visszaállítási lehetőségeink vannak

2.1. A mentések gyakorisága és időpontja

A mentés gyakoriságának meghatározásánál figyelembe kell venni az adatok fontosságát, mennyire akadályozza a mentés a rendszer működését, azaz mikor hajtható végre a mentés leginkább problémamentesen. Külön kell súlyozni az adatokat, így elképzelhető, hogy míg bizonyos adatokat naponta, addig más adatokat hetente, havonta vagy egyáltalán nem mentünk. További korlátozó tényező lehet az adatok mérete és a rendelkezésre álló mentési kapacitás. Bár ez utóbbi nem szabadna, hogy korlátot jelentsen, van amikor egy szervezet vezetése nem hagyja jóvá a szükségesnek ítélt mentési alrendszert. Ritka példa, hogy egyáltalán nincs mentés, sokkal gyakoribb, mikor az egyszer, egy célfeladatra megvásárolt mentési rendszer(ek)re akarnak minden további mentési feladatot hárítani, a drága bekerülési költség miatt. A mi feladatunk, hogy minden eszközzel ragaszkodjunk az optimális mentési eljáráshoz és eszközökhöz (hardver, szoftver).

2.2. Mentés típusa és tartalma

A mentendő fájloknál fontos meghatározni, hogy melyek azok a fájlok, melyekre szükségünk van és melyek feleslegesek, ezzel ugyanis időt, erőforrást és helyet spórolunk. A mentendő adatoknál, ha nem teljes könyvtárakat mentünk, a legelőnyösebb, ha kivételeket képzünk, mit *nem akarunk* elmenteni. Például ha egy központi, a dokumentumokat tartalmazó könyvtár alatt sokszor és sokan dolgoznak, akkor ott maradhatnak ideiglenes fájlok. Ezekre legegyszerűbb kivételt képezni, és így csökkenteni a mentés méretét és idejét. A mentés típusánál majdnem mindent az adatok mennyisége határoz meg. Ettől függ ugyanis, hogy elégséges-e a mentési eszköz sebessége, a mentési terület, hálózati mentésnél a sávszélesség, stb..

Teljes mentés

Állapot mentésnek is nevezik. Ekkor a teljes kijelölt adatmennyiség mentésre kerül. Előnye, hogy a legkönnyebben visszaállítható mentési forma, hiszen egy mentésből kell összeállítani a volt adatmennyiséget. Hátránya a nagy helyigény és a rendkívül hosszú mentési idő.

Részleges mentés

A részleges mentéseknél két típus különböztetünk meg:

Inkrementális: az inkrementális mentés a legutolsó teljes vagy részleges mentés óta eltelt állapotot rögzíti. Előnye, hogy a legrövidebb idő alatt képes a mentést elvégezni. Kazettás mentésnél a legköltséghatékonyabb, mivel kevesebb kazetta kell hozzá, de diszkre mentésnél is sokkal gyorsabb lehet, mint más mentési módszerek. Hátránya, hogy visszaállításnál a leghosszabb idő ennél a mentési típusnál szükséges, a sok felhasznált adathordozó miatt megnő az adatok sérülésének veszélye.

Differenciális: A legutolsó teljes mentés óta eltelt állapotot rögzíti. Előnye, hogy viszonylag gyors a helyreállítás (és ehhez kevesebb adathordozóra van szüksége, mint egy teljes mentés). Csökken a visszaállításnál jelentkező hiba veszélye. Hátránya, hogy több időt vesz igénybe a mentés és több adathordozóra van szükség, mint az inkrementális mentés esetén.

2.3. Mentési módszerek

Általánosan ideális megoldást természetesen nem tudunk adni, hiszen minden vállalkozásnak egyedi igényei vannak. Az alábbi módszerek a legelterjedtebbek közé tartoznak.

Hanoi tornyai

Egy viszonylag költséghatékony, de összetett rotációs típus. A régen ismert játékra épül, ugyanazzal a logikai megoldással dolgozik. Előnye, hogy kevesebb adathordozó kell hozzá, – nyolc darab kazetta készlettel 128 nap mentését meg tudjuk oldani – de hátránya bonyolultsága és nehéz nyomonkövethetősége. Ha szoftverünk támogatja érdemes lehet használni. Teljes mentésekkel használják [1].

Nagyapa–apa–fiú

Az ötnapos munkahetet veszi alapul. A napi mentéseket a fiú, a heti mentéseket az apa, míg a havi mentéseket a nagyapa kazetták hordozzák [2][3]. A havi mentéseket mindeképpen érdemes eltenni hosszabb időre, például egy éves időtartamra. Ezt meg lehet valósítani az alábbi módon: kell négy készlet kazetta a napi mentésekhez, négy készlet kazetta a heti mentésekhez és tizenkét kazetta a havi mentésekre. Így körülbelül húsz készlet kazetta elég egész évre, a szám a munkahetek arányában változhat, a hónaptól függően. A heti mentéseket érdemes pénteken elvégezni, itt teljes mentést használjunk. Hétfőtől csütörtökig használjunk különbözeti mentést – én a differenciálist ajánlom –, míg havi záráskor készítsünk egy teljes mentést és helyezzük biztonságba az adathordozókat.

6 kazettás mentés

Ez szintén öt napos munkahetekben gondolkodik. A különbség az előzőhöz képest mindössze annyi, hogy két kazettát rotálunk heti teljes mentésekkel, míg a négy egyéb napon egy-egy kazettát (vagy készletet) használunk fel különbözeti mentésre. Főleg kisebb cégeknél vagy egyedi szerver mentéseknél ajánlott használni. A legegyszerűbb és legköltséghatékonyabb megoldás.

Természetesen még több rotációs módszer is létezik és más terminológiát használhatnak ugyanarra a módszerre, de az alapok mindig ugyanazok maradnak.

2.4. Mentések biztonsága

Fizikai tárolás

Tárolásnál figyelembe kell venni az adathordozó típusát, hiszen senki sem szeretne egy nedves helyen tárolt kazettás egységről vagy egy hordozórétégénél megsérült DVD lemezről visszaállítást megkísérelni. Alapelv, hogy a teljes heti illetve havi mentéseket más földrajzi helyen kell tárolni, lehetőleg városon belül is legyen egy mentés, ami védve van a fizikai behatásoktól (elektromágneses sugárzás, víz, tűz stb.)

Titkosítás

Tárolás alatt nem csak a megfelelő fizikai körülményekre, hanem az adatok bizalmasságára is figyelniünk kell. Kiemelten fontos ez akkor, ha személyes vagy pénzügyi adatokat is tartalmaz a mentés. Érdemes olyan szoftver használni, mely támogatja az adatok titkosítását.

2.5. Helyreállítás

Fontos, hogy legyen visszaállítási stratégiánk. Ebben szerepelni kell, hogy miként állítsuk vissza a fájlokat:

- csak a hiányzó fájlokat állítjuk-e vissza
- a fájloknál miként döntünk, ha szerepelnek ugyan a rendszeren de más a méretük, dátumuk stb.
- teljes visszaállítást használunk

Itt mérlegelni kell, hogy mi a fontosabb: a gyors adat-visszaállítás vagy a 100%-os pontosság. Érdeemes az utóbbit választani, akkor is, ha sokkal lassabb. Átmenet lehet, ha van ellenőrző összegünk a fájlokhoz – bár a mentést lassítja – és ez alapján döntünk a visszaállítás szükségességéről.

3. Hardver alrendszerek

3.1. szalagos meghajtók (TAPE)

Mai napig a legelterjedtebb vállalati szintű mentési hardver elem. Attól függően, hogy a kazetta milyen típusú és rendszerű, különböző sebességet és tárolási kapacitást tesznek lehetővé. Alább a legfontosabb típusok, rövid ismertetéssel.

DAT 1987-ben a Sony alkotta meg, digitális hangrögzítés céljából, de hamarosan megjelent az adatmentésre szolgáló változata is.

DDS 1989-ben a DAT speciálisan adattárolásra kifejlesztett verziója, a Sony és a HP együttműködésében. Ma a HP a zászlóvivője a technológiának.

DLT Digital Linear Tape, még nagyobb verziója a Super DLT, a mai modellek fiber channelen és Ultra320-as csatornán kommunikálnak, a legelterjedtebb nagy tárolókapacitású szalagos egység.

LTO A legkomolyabb eszköz, a DLT-k alternatívájaként fejlesztették ki, kezdő kapacitása 100GB, ma már 200GB-osak az elterjedtek, melyek 30MB/s sebességgel írják az adatot.

Természetesen mindegyik modernebb szalagos egységhez kitaláltak autoloader megoldásokat, melyek segítségével a csak több szalagra menthető adatmennyiség is kezelhető emberi beavatkozás nélkül.

3.2. diszk–diszk

A merevlemezek árának csökkenésével együtt egyre jobban terjednek a merevlemezről merevlemezre, általában hálózaton keresztül továbbított adatmentési eljárások. Behatárolja lehetőségeit egyrészt a számítógépek fizikai közelsége – fizikai biztonság –, másrészt a hálózat sebessége, ha távoli helyre visszük a mentést. Ide sorolhatjuk a NAS eszközöket is, amennyiben fájl megosztásként használjuk őket. (Szabványosított protokollokról később ejtünk szót). Nem megoldás, de megjelentek a hordozható „dobozok”, melyek USB2.0 vagy FireWire kapcsolaton kommunikálnak, de ezek leginkább kisvállalkozások vagy egyedi szerver mentésekhez használatosak. Érzékenyek a mechanikai behatásokra, több emberórát igényel használatuk.

3.3. VDL (VTL)

Nagy adatmennyiségnél érdemes megfontolni egy Virtual Disk Library – más terminológiában Virtual Tape Library – használatát. A legegyszerűbben úgy fogalmazhatunk, hogy ez az eljárás diszk–diszk–(tape) mentési sorrendet használ. Ez a megoldás a merevlemezek gyorsaságát, a hálózat egyre nagyobb terhelhetőségét használja ki. Az alkalmazás virtuális meghajtókat és slotokat használ. Megfelelően nagy sávszélesség mellett kitűnően lehet hasznosítani a merevlemez gyorsaságát és olcsóságát a

kipróbált kazettás mentési technológia kiegészítéseként vagy akár helyett. Akkor van leginkább jelentősége ennek a technológiának, ha adott időkeretünk és sok adatunk van a mentés elvégzésére. Ekkor jól alkalmazható az a technika, hogy a virtuális meghajtókban található slotokba helyezzük el az adatot, majd a mentések végeztével folyamatosan írjuk ki kazettára az adatokat vagy azok egy részét.

3.4. NDMP: Network Data Management Protocol

Bár nem hardver eszköz, de megoldás, így került be a listába. Az 1996-ban életre hívott protokoll célja, hogy egységes mentési eljárást teremtsen a heterogén környezetekben, ahol hálózaton keresztül NAS-okról lehet menteni az adatot. Az egységes felületet használva a hardver és szoftver gyártók gyakorlatilag teljesen össze tudják kapcsolni rendszereiket, mivel kompatibilisek lesznek. Jelenleg is sok termék és NAS támogatja [4].

4. Mentőszoftverek követelményei

4.1. Platform támogatás

Fontos, hogy szoftverünk „kliens” oldali része is támogatott legyen valamint az is előny, ha a menedzselhetőség (felület, központi felület) is jó. Egy jó menedzsmenet felület már fél siker. Fontos, hogy mindig látható legyen a mentések állapota, kereshető legyen a katalógus, innen is indítható legyen az adatok visszaállítása, az integritás ellenőrzése. Fontos, hogy mindig hibamentes mentéseink legyenek. Mivel azonban hibalehetőség mindenhol van, ezért fontos, hogy a mentéseket tartalmazó adathordozókat még azelőtt ellenőrizzük, hogy elkezdenénk a hosszas visszaállítási procedúrát. Amennyiben olyan hiba lenne ami problémát okoz, előbb tudjuk meg és kereshetünk egy, a kívánt állapothoz közeli mentést és abból visszaállíthatjuk az állapotot. Állandóan változó adatoknál természetesen ez nem járható út, de legalább tudjuk, hogy mi fog hiányozni, mi nem működik.

Hibatűrés (paritás, redundancia): Az integritás ellenőrizhetősége mellett fontos, hogy a sérülés következtében fellépő adatvesztés veszélyét is minimalizáljuk. A már sok helyen – hardver, szoftver – régóta ismert és alkalmazott paritást használó módszerek itt is használhatók. Egy jobb szoftver – természetesen nagyobb helyigény mellett – ismeri és lehetőséget ad paritásbitek használatára.

Jogosultsági rendszer: A jogosultság szabályozása természetesen itt is fontos szerepet kaphat. Ki melyik tároló eszközhöz férhet hozzá, esetlegesen szabályozhatónak kell annak is lennie, hogy ki, mikor használhatja az erőforrásokat.

Hardver támogatás: Szintén fontos lehet, főleg a már meglévő befektetések védelmében, hogy egy szoftver – de legalábbis a tárolást és rögzítést végző része – a lehető legtöbb hardver eszközt támogassa. Itt nem az számít, hogy a menedzsmenet felület csak egy operációs rendszeren működik, csakis a mentést fizikailag is kivitelező részére.

Biztonságos kommunikáció: A hálózaton keresztül történő mentések alapfeltétele, hogy titkosítottan haladjon az információ. Természetesen ez lassítja a mentést, de az adatok bizalmassága – jelszóállományok, személyes, pénzügyi adatok is

„utazhatnak” a hálózaton! – megköveteli a titkosítást. Természetesen lehet használni már meglévő VPN rendszereket vagy kiegészítő programokat, mint például az stunnel, de az is plusz adminisztrációt és hibalehetőséget eredményez.

5. Kereskedelmi szoftverek

Az alábbi részben rövid áttekintést adok a szabad operációs rendszereken is működő, vagy azt felhasználó kereskedelmi szoftvekről.

5.1. TIVOLI

Az egyik legismertebb, mindenre kiterjedő megoldás, melyet az IBM forgalmaz. Gyakorlatilag majdnem mindenben megfelel egy ideális nagyvállalati megoldásnak. Teljes rendszer, minden elterjedt platform és rendszer támogatott, komplett autentikációs és autorizációs rendszerrel rendelkezik, rengeteg szalagos egységet támogat, természetesen az IBM termékek teljes támogatást élveznek, de rengeteg más megoldáshoz is rendelkezik támogatással. Hátránya a viszonylagos bonyolultsága és az ára.

5.2. Veritas

Kifejezetten adatvédelemre és adattárolásra szakosodott cég, termékei felveszik a versenyt bármelyik megoldással, természetesen a Tivolival is. Minden pozitív tulajdonság amit a Tivolinál felsoroltunk ugyanúgy elmondható róla is.

5.3. Arkeia

Az egyik legkiforrottabb megoldás, mely a Linux alapú rendszereket helyezte a mentés középpontjába. Szerver megoldása bővítmény alapú és széleskörűen támogatja az adatbázis kezelők hot-backup-ját, köztük a MySQL megoldásait [5].

5.4. Atempo

Nagyon jó menedzsment felülettel és megoldással ellátott termék. Az összes nagyobb adatbázis-kezelő támogatott, de széleskörűen támogatja teljes rendszerek mentését. Menthető rendszerei lefedik a teljes palettát: Linux, Windows, MacOS X, VMS, UNIX, NetWare, mind megtalálhatóak benne. Tároló szervernek – storage node – Linuxot vagy MacOS X-et használhat. Teljes helyreállításra képes Windows, MacOS X és Linux esetén is [6].

5.5. BakBone NetVault

A cég az OSDL tagja, részt vesz az NDMP-vel foglalkozó szervezet munkájában is. Szintén széles körű támogatással rendelkezik a támogatott rendszerek és megoldások területén, természetesen ez a megoldás is plugin rendszerű. A legnagyobb adatbáziskezelők mellett támogatja az SAP rendszereket is, valamint rendelkezik titkosítási támogatással [7].

5.6. BRU

Az egyik legrégebbi, Linuxon működő kereskedelmi mentőszoftver. Ma már sok megoldás megelőzi, tudásban ugyanis jelentősen elmarad az eddig felsorolt megoldásoktól. Csak UNIX és Linux alapú megoldásokat valamint a MacOS X-et – ami szintén rendelkezik Unix alapokkal – támogat. Sokat elmondhat, hogy teljesítményét a tar-ral hasonlítják össze a weboldalon is [8].

5.7. LoneTar

Szintén „régi motorosnak” számít a Linuxos és UNIX világban, de mára jelentősen lemaradt a mezőnytől. Támogatja az adatok titkosítását, rendelkezik grafikus felülettel, képes bootolható mentést készíteni, de egyéb módon nem tűnik ki a mezőnyből.

6. Szabad szoftverek

Rengeteg megoldást fel lehetne sorolni, most azonban a klasszikus értelemben vett – és használható minőségű – backup megoldásokat vizsgáljuk meg. Ezért kimarad az ismertetésből, de jó ismerjük a Ghost4Linux és a Mondorescue nevét. Utóbbi nevét azonban jegyezzük meg, mert méltán népszerű és hasznos megoldás, de számomra leginkább rendszerek gyors visszaállítására alkalmas.

6.1. „Made by hand”

Az a megoldás, aminél csak magunkat hibáztathatjuk. Leginkább Linux-alapú szervereink mentésére alkalmas, esetleg egy felcsatolt Windows megosztás mentésére is. Gyakran alkalmazzák az ssh + rsync megoldást vagy az rsyncet VPN csatornán keresztül, hogy biztosítsák az adatok titkosságát adattovábbítás közben. Hátránya, hogy különálló szkriptekkel dolgozunk, nincs központi felület, ahol nyomon követhetnénk mi történt, mi történik, ha tömörítjük az adatokat vagy a tar paranccsal írjuk ki szalagra, akkor nekünk kell elkészíteni a nyilvántartást, hogy melyik fájl melyik kazettán található meg.

6.2. DAR – Disk ARchiver

Egy kifejezetten adatmentési megoldás, melyet rendkívül jó kidolgozottság jellemez [9]. Támogatja a 64 bites rendszereket, saját libraryvel rendelkezik, melyen keresztül API-t biztosít külső programok számára. Támogatja a darabolt mentéseket, igaz, önmagában csak kézi segítséggel. Nagy előnye, hogy készít külön katalógus fájlt, melyet a mentésen kívül is tárolhatunk. Mentésnél meghatározhatunk szűrőket, mit nem szeretnénk elmenteni, de akár azt is, hogy bizonyos mentendő fájlok ne kerüljenek tömörítésre – például kedvenc ogg és mp3 gyűjteményünk, ahol nem csökken a méret, de erőforrást bőven fogyaszt a mentés. Linux esetén támogatja az ACL-ek mentését, visszaállítását is. Az adatok titkosíthatók, melyekhez csak egy passphrase megadása után férünk hozzá. Támogatja a külön fájlok visszaállítását a mentésből, CRC teszt is végezhető valamint támogatja az adatok redundáns mentését! Így ugyan több helyet foglal a mentés, de jól megválasztott arányok esetén ezek a paritásbitek, melyek a redundanciát biztosítják, segíthetnek a sérült mentésekből minél több adatot visszanyerni. Távoli mentésre is rá lehet bízni, kis kézi segítséggel – a netcat program használatával – adattitkosítással természetesen. Kiegészítő programként a SaraB nevű ütemező is elérhető

hozzá, melynek segítségével elérhető például a Hanoi tornyai [1] és a Nagyapa–apafiú [2][3] mentési eljárás is. Hátránya, hogy csak Linux-alapú rendszereket tud kezelni, de szervereinket nyugodt szívvel rábízhatjuk. A megbízhatóságáról és erejéről beszéljen egy számadat: az ismert legnagyobb mentés mérete 1,4 Terabyte adat volt, melyet 200 darab DVD lemezre írtak ki.

6.3. Bacula

A legjobb és vállalati felhasználásra leginkább megérett backup szoftver a Bacula [10]. Támogatja a Windows „klienseket” is, kiemelkedően sok szalagos meghajtót támogat, széleskörűen paraméterezhető, és több felületen keresztül is elérhetjük. Támogatja a slice-okat, rendelkezik ACL rendszerrel, több részre elosztott és rendkívül hatékony szoftver. Ezért ez az a megoldás, amit a leginkább áttekintünk, a következő teljes rész ezzel az alkalmazással foglalkozik.

7. Bacula – részletesen

A Baculáról ezen kiadvány keretében teljes konfigurálást nem tudunk bemutatni, – bár nehéz is lenne, hiszen ahány cég annyi szokás – azonban ismertetjük felépítését, lehetőségeit, megoldásait.

7.1. A Bacula felépítése

A Bacula moduláris felépítésű megoldás, a mentési és visszaállítási eljárások egyes részeit különböző szolgáltatások felügyelik. A teljes működéshez szükségünk van a Bacula Director, Storage, File, Catalog valamint Console szolgáltatására, melyek összessége adja a teljes rendszert. Ezek mindegyike elhelyezkedhet külön, de akár egyazon számítógépen is.

Bacula Director

Ez a démon felügyeli a mentések, visszaállítások, szalagra írák és ellenőrzési eljárások összességét, ő a központ. Az összes többi rész kapcsolatban áll vele. Fontos, hogy állandó kapcsolatban legyen a Catalog-gal, mert ez alapján ad utasításokat, így tudjuk meg mely fájlok mentésére van szükségünk! A kapcsolatoknál meghatározhatjuk, hogy mely Console kliens kapcsolódhat hozzá, adhat utasításokat. TCP kapcsolaton keresztül kommunikál, alapértelmezetten a 9101-es porton.

Bacula Storage

A démon feladata a beérkező adatok szalagra vagy meghajtóra – fájlba – történő mentése illetve kérés esetén ezek visszatöltése a File démon számára. A Director-ral és a File démonnal áll kapcsolatban. Az utasításokat a Directortól kapja, míg az adatot a File démon szolgáltatja illetve fogadja tőle. Alapértelmezetten a 9103-as TCP portot használja.

Bacula File

Klients oldalon futó szolgáltatás illetve démon, mely az adatok szolgáltatásáért illetve helyes visszaállításáért felelős. A helyes adatvisszaállításba beleértjük a jogosultságok helyes visszaállítását is! Windows alapú verziója natív Win32-es szolgáltatás. TCP kapcsolaton kommunikál a Directorral és a Storage démonnal, a 9102-es portot használja alapértelmezetten.

Bacula Console

Az adminisztrátor által használt felület, mellyel a Director számára tud utasításokat adni. Jelenleg két verzió érhető el, egy shell és egy grafikus (GNOME-os). A shell komplett megoldás, míg a GNOME felület felhasználóbarátabb, de korántsem olyan teljes, mint a másik, ahol minden parancsot elérhetünk.

Bacula Catalog

A Bacula másik fontos része, ez az ami megkülönbözteti az olyan megoldásoktól, mint az ssh+rsync páros vagy más házi készítésű tar-al létrehozott megoldások. A katalógusok tárolják az összes mentés összes adatát, melynek segítségével gyorsan lokalizálható a visszaállítani kívánt fájl vagy könyvtár. Az adatok SQL-adatbázisban tárolódnak, jelenleg három választható adatbázis-kezelővel. Az adatbázis-kezelők között a Postgresql és MySQL mellett az SQLite található meg. Utóbbit érdemes használnunk, ha nem akarunk még külön foglalkozni adatbázis adminisztrációval is. Gondunk csak akkor lesz, ha adatbázisunk 2 GB méret fölé nő. Addig azonban ez a legbarátságos megoldás, mivel a Bacula direkt módon kezeli a fájlt, hordozható, könnyen menthető. Az adatbázis a szoftver lefordításakor kell kiválasztanunk illetve a disztribúcióból a megfelelő csomagot telepítenünk.

7.2. Amire figyelni kell

Mivel a teljes konfiguráció ennek az írásnak nem is volt célja, ezért itt „csak” tanácsokat adunk, amit érdemes megfontolni a sikeres mentéshez.

Linux alapú rendszerek mentése nem jelent problémát, azonban a Windows alapú rendszereknél problémákba ütközhetünk. A Bacula első verziói egy Cygwin által megvalósított rendszerhívást alkalmaztak, azonban a Windows 2000 megjelenésével több dolog is megváltozott a rendszerhívások és jogosultságok körül. Ezért a Bacula újabb verziói a natív Windows mentési API rendszerhívásait használják, amivel kitűnően együttműködnek az NT/2000/XP rendszerek, azonban megvan az a nem kis hátrányuk, hogy egy-egy fájl visszaállítására, kinyerésére ugyanilyen rendszer alatt van csak lehetőségünk, azaz Linux alatt nem. A megoldás egy portable nevű kapcsoló használata. Ha hasonló problémába ütköztünk, akkor keressünk rá erre a kapcsolóra a dokumentációban. Azonban vigyázzunk, mert ha ezt a kapcsolót használjuk, akkor nem leszünk képesek visszaállítani a jogosultsági és tulajdonosi jogokat! El kell döntenünk mi az ami kevesebb áldozattal jár. Megoldásként használhatjuk a SetACL [11] rutinyűjteményt, melynek segítségével menteni és visszaállítani is tudjuk a jogosultságokat. Jelenleg készítették hozzá parancssori és Active-X vezérlő elemet (ennek használatával írhatunk Visual Basic szkripteket is például).

Fontos, hogy mindig figyeljünk a rotálásra és megfelelően válasszuk ki a kazetták cseréjének idejét.

SQL adatbázis mérete. Figyeljünk oda az SQLite már említett 2GB-os korlátjára valamint arra, hogy ez a fájl is mindenképpen el legyen mentve (vagy a többi adatbáziskezelő adatait is megfelelő módon mentjük el)

Figyeljünk oda a mentések méretére. Kellemetlen lehet, ha kazettát kellene cserélnünk, mert megtelt, de nem vagyunk ott. Nagy méretű mentéseknél megfontolandó lehet egy autochanger használata. A Bacula természetesen rengeteg eszközt támogat.

Jól állítsuk be a jogosultságokat. Mivel elég kifinomult ACL rendszere van, ezért hajlamosak lehetünk a minden kliens lásson mindent elvet követni, pedig ez akár nagyobb hibákhoz is vezethet.

Ha igazán biztosra akarunk menni, akkor készítsünk minden mentéshez egy úgynevezett Bootstrap fájlt. Ennek a fájlnek a tartalma alapján a Bacula segédprogramjaival olyan mentéseket is vissza tudunk állítani, amihez nem érhető el katalógus. Természetesen egy létező katalógusból is előállíthatjuk a megfelelő Bootstrap fájlt.

8. Összefoglalás

A leírtak mindenkit meggyőzhetnek arról, hogy a Linux-alapú rendszereknek helye van a mentési megoldásoknál, mind kereskedelmi-, mind Linux-alapokon. Az is kiderült, hogy a kereskedelmi megoldások helyenként kevesebbet tudnak, mint szabad szoftveres konkurenseik. A fejlődés mindenesetre öröndetes, hiszen pár éve még nem voltak jó minőségű, elérhető, teljesnek mondható szabad szoftveres megoldások, de a nagy kereskedelmi szoftvergyártók is egyre komolyabban veszik a piac linuxos részét.

Hivatkozások

- [1] <http://www.dlittape.com/ThreeRs/Reliability/Rotation/Tower.htm>
- [2] http://en.wikipedia.org/wiki/Grandfather-Father-Son_Backup
- [3] <http://www.dlittape.com/ThreeRs/Reliability/Rotation/Grandfather.htm>
- [4] <http://www.ndmp.org/>
- [5] <http://www.arkeia.com/>
- [6] http://www.atempo.com/products/backup_restore.php
- [7] <http://www.bakbone.com/>
- [8] <http://www.tolisgroup.com/>
- [9] <http://dar.linux.free.fr/>
- [10] <http://www.bacula.org/>
- [11] <http://sourceforge.net/projects/setacl/>

Honosítási helyzetjelentés, avagy tényleg bárki megérti azt amit a képernyőn lát?

Fejős Tamás

<fejios.tamas@lme.linux.hu>

Kivonat

A felhasználók egy program kezelését könnyen megtanulják, ha megértik a menüket, parancsokat, üzeneteket. Ha megértik. . .

Magyarországon élünk, magyarul beszélünk és sokunk szeretné, ha alkalmazásai is ékes magyarsággal szólának hozzá és főleg angolul kevésbé tudó ügyfeleihez, munkatársaihoz, családtagjaihoz. Előadásom első felében alkalmazáscsoportonként összefoglalom az eddig elért eredményeket, a honosítást segítő technológiákat és a várható folytatást.

Kitérek magára a Linuxra és a rendszer segédprogramjaira, a „legmagyarabb” terjesztésekre, az alkalmazásokra, úgy mint: webböngésző, irodai szoftver, grafikus munkafelületek, stb. és a dokumentáció (folyóiratok, szakkönyvek, kézikönyv oldalak, *Hogyan*-ok, szabad szoftverekhez kapcsolódó írások, esszék).

Az egyes „szakterületekre” általában külön csoportok specializálódtak, rövid bepillantást nyújtok tevékenységükbe, aktuális és tervezett projektjeikbe. További segítőképző önkéntesek toborzása reményében áttekintem legégetőbb problémáikat, hiszen minél többen vagyunk annál több eredményt tudunk felmutatni.

Tartalomjegyzék

1. Bevezetés	45
2. Linux, és a rendszer segédprogramjai	45
2.1. Telepítő	46
2.2. UHU-Linux	46
2.3. SuSE Linux	47
3. „Általános” alkalmazások	47
3.1. \TeX , \LaTeX	47
3.2. Magyar Ispell	47
3.3. Magyarul hogyan, segítség a magyarítások üzembe helyezéséhez . . .	47
4. Grafikus munkakörnyezetek, alkalmazásaik	47
4.1. KDE	48
4.2. Gnome	48
4.3. OpenOffice.org	48
4.4. Mozilla és „társai”	49

5. Dokumentációk	49
5.1. Kézikönyvlapok	49
5.2. <i>Hogyan</i> -ok	49
5.3. Licencek	49
5.4. Egyéb leírások, információk	50
6. A jövő feladatai	50
6.1. Terminológia	50
6.2. Fordítássegítő rendszer	50

1. Bevezetés

Van, aki fordítva szereti. Sokunk munkáját megkönnyíti, szórakozását élvezetesebbé teszi, ha számítógépe anyanyelvén szól hozzá. A Linux és a hozzá kapcsolódó sok ezer szoftver és dokumentáció honosítása több területre osztható, minden területnek megvan a maga gazdája.

A legismertebb „szakterületek”:

- Linux, és a rendszer segédprogramjai,
- „Általános” alkalmazások,
- Grafikus munkakörnyezetek, alkalmazásaik,
- Dokumentációk.

A szoftverhonosítás nem csak a felhasználói felület és a dokumentáció fordításából „nemzetköziesítéséből” (i18n) áll, hanem a hazai sajátosságokhoz (ABC, dátumformátum, valuta stb.) kell igazítani a szoftvert, e tevékenység neve a lokalizáció (l10n).

Kicsit szabatosabban fogalmazva a „nemzetköziesítés” (internacionalizáció) alatt azt értjük, amikor egy program, vagy egy csomagot alkotó programok összessége tudja, hogy a világon több nyelvet beszélnek, és támogatja is ezen nyelvek egy részét. Ez egy általánosításra visszavezethető folyamat, melynek során a programnak el kell vonatkoztatnia az eredeti angol nyelvtől, és az ezzel járó jelölésektől, és meg kell birkóznia a feladattal általános, nyelvtől független módon is. Ebből is látszik, hogy a tevékenység nem csupán nyelvismeretet igényel, hanem az egységesség, valamint a jó minőség érdekében az elkészült fordításokat lektorálni is kell. A programok fejlesztőinek több módszer is rendelkezésükre áll ahhoz, hogy ezt a célt elérjék, létrejöttek erre vonatkozó szabványok. A GNU gettext egyike e szabványoknak.

Lokalizáció alatt azt a műveletet értjük, amikor a már „nemzetköziesített” programokat feltöltjük olyan adatokkal, amik lehetővé teszik, hogy a program egy adott nyelvnek vagy kulturális egységnek megfelelően kezelje a ki- és bemenetét. Ezen folyamat során az általánosított módszereket használó program ezeket specifikusan alkalmazza. A fejlesztői környezet több lehetőséget nyújt a futásidejű konfigurációra. Valamely országra jellemző kulturális szokások formális leírása, és az adott országban beszélt nyelvre fordított karakterláncok halmaza határozza meg az adott országban vagy nyelven használandó „locale”-t. A felhasználók lokalizálhatják a programokat, ha a program indítása előtt, a megfelelő változó beállításával kiválasztják a megfelelő „locale”-t. Az operációs rendszer mindkét feladathoz komoly támogatást nyújt.

2. Linux, és a rendszer segédprogramjai

Mindjárt a legkeményebb dióval kezdjük, hiszen a rendszermag még nincs lefordítva, és a parancsértelmező (shell) sem tud általában magyarul. Ettől természetesen még az angolul kevésbé tudók is képesek használni a rendszert, mert a grafikus felületek általában elfedik ezeket a rétegeket, és magyarul szólnak a felhasználóhoz.

Ha a kernelbe befordítanak egy csomó nyelv támogatását, vagyis ha a rendszermag több nyelven is tudna üzenni, akkor ez óriási méretűvé tenné. Ehelyett járhatóbb út, ha a kernelnek elkészítik a különféle nyelvi változatait. Lehetőség volna még többnyelvű üzenet-katalógusok használatára, ennek viszont megvan az a veszélye, hogyha az előtt

lép fel egy hiba, mielőtt az üzenet-katalógus betöltődik, egyáltalán nem kapunk hiba-üzenetet, illetve ez a megoldás újabb hibákat hozhat a rendszerbe.

Ezen kívül a Linux nagyon jól támogatja a lokalizációt. A legalacsonyabb szintű programkönyvtárak szintjén is (pl. a glibc-locale, locales csomagok) ismeri a helyi dátumformátumot, a napok, hónapok nevét, az ABC-t, a valutát, a mértékegységeket, a szótári rendezési sorrendet, a speciális nyelvi karaktereket tartalmazó karakterkészletet (ez hazánkban az ISO8859-2, de több terjesztés már az UTF-8-at alkalmazza, pl. a RedHat, de a többi is felkészíthető rá). Ezeket a definíciókat elég egyszer elkészíteni, és a rendszer elérhetővé teszi a programok számára, így azok megosztva használhatják. A magyar sajátosságok támogatása már évek óta részét képezi a disztribúcióknak. A hazai beállításokat az alábbi környezeti változók beállításával érhetjük el:

```
LANG=hu_HU
LC_ALL=hu_HU
LC_COLLATE=hu_HU
LC_CTYPE=hu_HU
LC_MESSAGES=hu_HU
LC_NUMERIC=hu_HU
```

Ide tartozik továbbá a fájlrendszerek kérdése is, hiszen koránt sem mindegy, hogy az ékezetes állományneveket milyen kódolással próbáljuk megjeleníteni. A Linux rendszermagjába különféle nemzeti nyelvi támogatás (NLS) fordítható bele, és az adott fájlrendszer csatolásakor megadható a használandó karakterkódolás, természetesen az ISO-8859-2 és az UTF-8 is.

Az alaprendszerhez tartozó segédprogramok többsége már honosított, az előző változók beállításával ezt ki is próbálhatjuk.

A segédprogramok és jó néhány alkalmazás többnyelvűsítését a GNU Gettext [1] [2] segítségével tehetjük meg. Ennek röviden annyi a lényege, hogy a program üzeneteit a fejlesztők különválasztják az üzenetek és a menük szövegétől, melyet egy .po kiterjesztésű fájlba helyeznek. Ebben az állományban megtalálható az eredeti angol és alatta a lefordított szöveg. A fordítatlan eredeti változatban mindkettő angolul van. A fordítók ezeket lefordítják, majd a gettext eszközei segítségével .mo állományt készítenek belőle, melyet a /usr/share/locale/ könyvtár alá¹ másolnak [3] [4].

2.1. Telepítő

A legtöbb, hazánkban elterjedt terjesztés telepítője elfogadható szinten tud magyarul. A jövőben várhatóan ezek is továbbfejlődnek.

2.2. UHU-Linux

Az első igazán magyar Linuxnak mondható terjesztés, teljes mértékben hazánkban tervezték és fejlesztik. Lassan elérkezik az 1.2 verziójához, hiszen weboldalukról [5] már elérhető a második nyilvános tesztverzió.

Tartalmazza az FSF.hu-féle OpenOffice.org irodai csomagot és a Mozillát, egy jDictionary nevű – többek közt – angol–magyar szótárprogramot, mely amúgy teljes értékű dict kliens is, a hazai jogszabályokat feldolgozó Complex jogtár hatályos törvényeket tartalmazó változatát (mely a teljes alkalmazás megvásárlása után a CD-Jogtár teljes körű használatára alkalmas). Természetesen magyar nyelvű felhasználói dokumentáció is elérhető hozzá.

¹pl. /usr/share/locale/hu/LC_MESSAGES/coreutils.mo

2.3. SuSE Linux

Egy másik jó példa arra, hogy nem reménytelen vállalkozás a Linux honosítása. Szintén magyar nyelvű felhasználói dokumentációval felvértezve, és ami fontos, magyar nyelvű, professzionális ügyfélszolgálattal [6].

3. „Általános” alkalmazások

3.1. T_EX, L^AT_EX

A T_EX² nagyon jól használható alkalmazás nyomdakész kiadványok készítésére, nem DTP, hanem tördelő program, jelen konferencia előadásit tartalmazó kiadvány is L^AT_EX-hel készült.

Aki nyomdai környezetben kívánja használni a Linuxot fontos lehet neki, milyen eszközök állnak rendelkezésére. A T_EX igen jól felkészíthető a hazai környezetre némi „bütykölés” után [7] [8] [9] [10].

3.2. Magyar Ispell

Adott nyelvű gépelési, helyesírási hibák felderítésének, javításának támogatása nélkül nem beszélhetünk komoly honosításról. A Linuxban ez többféleképpen is megoldható, általában a legpraktikusabb a Magyar Ispell használata, széleskörű elterjedtsége (integrációja meglevő alkalmazásokkal pl. OpenOffice.org FSF.hu build), jó dokumentációja és tudása miatt.

A rendszert folyamatosan frissíti készítője. A [11] [12] címeken található bővebb információ pl. arról, miként lehet integrálni alkalmazásainkkal. (T_EX, HTML állományok interaktív ellenőrzése parancssorból, Emacs, L_YX, K_YX, KWrite, Abiword, OpenOffice.org, Vim. . .)

A fentiek felül egy hozzá írt spdaemon nevű eszköz segítségével alkalmas webes helyesírás-ellenőrzésre pl. űrlapok kitöltése után. (Megtalálható a letöltött forráskódban az spdaemon könyvtárban.)

3.3. Magyarul hogyan, segítség a magyarítások üzembe helyezéséhez

Ugyan enyhén szólva sem „mai gyerek”, mégis érdemes áttanulmányozni [13], mert jól használható gyűjtemény, ha a lehető legteljesebben honosítani akarjuk Linuxunkat. Nem naprakész, de jobb híján ebben is találunk pár ötletet. . .

4. Grafikus munkakörnyezetek, alkalmazásaik

Az átlag felhasználó által legtöbbet használt felületek meglehetősen jól tudnak magyarul. Nem csoda, hiszen ezek honosításába rengeteg energiát fektetett a hazai szabad szoftver közösség.

²A T_EX név a $\tau\epsilon\chi$ görög betűk latin betűs átírata (ejtsd tech), mely a görög *művészet* szó kezdete.

4.1. KDE

Az egyik legelterjedtebb grafikus munkakörnyezet, honosítói nagyon jól végzik munkájukat, hiszen naprakészen követik az alaprendszer fejlődését. Honosítási munka szempontjából érdekessége egy KBabel-nek nevezett program, mely a .po állományok fordítását hivatott segíteni. Tartalmaz fordítói memóriát is, így régebbi fordításaink bizonyos részét újra felhasználhatjuk, ha a fordítandó szövegek egyeznek (ennek mértéke a felhasználó által beállítható, de vigyázat! rendkívül le is lassíthatja, ha túl nagyvonalúan állítjuk be...). Várhatóan a jövőben is naprakész lesz a magyarítása [14].

4.2. Gnome

A másik legelterjedtebb grafikus felület. Szintén jól áll. ☺ Ennek nagy lökést adhatott az is, hogy emlékeim szerint az UHU-Linux alapértelmezett grafikus felülete. A jövőben is várható, hogy magyarítása naprakész lesz, hála a mögötte álló közösségnek [15] [16].

4.3. OpenOffice.org

Szintén kemény dió, többször többeknek beletört vagy legalábbis meghajlott a bicskája a honosítási kezdeményezésekbe. Mára szerencsére a helyzet megnyugtató, a felhasználói felület nagyon jól magyarított, elkészült a részletes tippek fordítása is (amely gyakorlatilag egy mini súgó).

Az LME több téren is segítette a projektet, tagjai (sok más projekthez hasonlóan) részt vettek a munkában, valamint komoly anyagi segítséget is nyújtott az egyesület a projekt finanszírozásához (pl. a részletes tippek lektorálása).

Érdekessége, hogy a „gyári” változathoz képest több hozzáadott értéket is tartalmaz, pl. a Magyar Ispell helyesírás-ellenőrző (mely lekörözi a más elterjedt szövegszerkesztőkben jelenleg alkalmazott programokat), és néhány, a magyar ékezetes karaktereket is korrekten tartalmazó font.

A menürendszer a fordítást segítő webes rendszer segítségével egy háromnapos, heroikus fordítópartin gyakorlatilag lefordításra került, természetesen hátra volt még a lektorálás.

Amikor 2003-ban készítője bemutatta a hamburgi OO.o konferencián, az ott jelenlevő török csapat azon nyomban neki is kezdett a menürendszer lefordításának. A webes rendszer nagy előnye, hogy bárki, aki jelen volt a fordítóparti helyszínén, vagy internet-hozzáférése volt, pusztán egy webböngésző használatával részt tudott venni a munkában. További kliensprogramra nem volt szükség, és akár már 10 perces munkával is tudott segíteni, hiszen a fordítandó szöveg apró részekre lett felosztva [17].

A részletes tippek fordításához az előző tapasztalatai alapján elkészült egy újabb webes rendszer, mely hasonló előnyöket és fejlettebb szolgáltatásokat nyújtott mint elődje [18].

A közeljövőben várható az 1.1.3 honosított változatának kiadása. A 2005 tavaszán megjelenő 2.0 súgórendszere a fejlesztők ígérete szerint olyan formátumban lesz, mely könnyen fordítható, ezzel lehetőség nyílik a súgó magyarítására is. Adott lesz a lehetőség egy újabb heroikus küzdelemre ☹.

További kiegészítések, sablonok, betűtípusok, valamint telepítési útmutató és felhasználói kézikönyv elérhető [19], továbbá létezik levelezőlista [20] is.

StarOffice és OpenOffice.org felhasználók: [21]

4.4. Mozilla és „társai”

A webböngésző képességeit jól tükrözi, hogy honosított változata az OpenOffice.org mellett beépítésre került a MagyarOffice-ba is. Az egyes Mozilla kiadások után megjelenik a magyarított verzió is [22], ami amúgy teljes körűnek mondható, hiszen a sűgő, sőt egy Enigmail nevű, a gnupg-t a Mozillával integráló plugin is le van fordítva.

A Mozilla projekt további termékei a Firefox és a Thunderbird is honosítás alatt vannak, ám azok kiadása körül adódott némi „gubanc”, ugyanis az előbb említett neveken csak a Mozilla Foundation adhat ki szoftvert, így a fordítások integrálása is az ő feladatuk. A Firefox esetén várható, hogy az 1.0 megjelenésére rendeződik a helyzet, és az angol nyelvűvel egy időben jelenik meg a magyar kiadás is. A Thunderbird levelező kliens esetén ennél valószínűleg kicsit többet kell várni. Magyar nyelvű levelezőlista [23] itt is elérhető.

5. Dokumentációk

A programok honosítása mellett elengedhetetlen, hogy a rendszergazdák, és főleg a felhasználók által használt legfontosabb dokumentáció is rendelkezésre álljon magyarul.

5.1. Kézikönyvlapok

A legtöbbet használt kézikönyvlapok (man) fordítása elkészült, a munka koordinálását volt hivatott segíteni a „Honosítás” [24] oldal. Jelenleg ez a projekt erősen csökkentette aktivitását. Az elkészült kézikönyvlapok többek közt részei a SuSE Linux és az UHU-Linux terjesztéseknek is.

5.2. *Hogyan-ok*

A *Hogyan*³-ok nagyon fontos mankót nyújtanak komplex feladatok megvalósításához, hiszen az első lépésektől az utolsóig részletesen leírják, a megvalósításhoz vezető tennivalókat, komoly segítséget nyújtva ezzel a felhasználóknak. Érdemes átnézni a rendelkezésre álló *Hogyan-ok*at, hiszen nagyon sok témát felölelnek.

A *Hogyan-ok* fordítását az FSF.hu Alapítvány koordinálja [25], a projekt weboldalán fellelhetők a fordítatlan és a már lefordított *Hogyan-ok*, illetve egy webes alkalmazás, amivel nyomon követhető a fordítások állapota. Levelezőlista [26] itt is segíti a munkát.

5.3. Licencek

A szabadszoftver-llicencek általában angol nyelvűek, és bármilyen fordításuk csak tájékoztató jellegű lehet, az irányadó az eredeti nyelven íródott. Egy teljesen magyar licenc az UHU-Linux Licence [27]. Természetesen a GPL magyar fordítása [28] is elérhető. A három GNU-licenc (GPL, LGPL, FDL) egy helyre gyűjtve [29] szintén hozzáférhető magyarul.

³Angolul HowTo

5.4. Egyéb leírások, információk

Egyéb leírások dokumentációk [30], levelezőlista [31], és honosítással foglalkozó közösség [32] is elérhető az internet segítségével.

6. A jövő feladatai

6.1. Terminológia

A fordítók, főleg az alkalmi, vagy kezdő aktivisták munkáját nagyban segítené egy összefogott, több „szakágat” felölelő terminológiai adatbázis, ahol pl. a Mozillában használt fordítás mellett megtalálható volna ugyanazon szó OpenOffice.org környezetben, vagy KDE alatt alkalmazott fordítása is. Ennek segítségével egy–egy ismeretlen vagy nehezen fordítható szó vagy kifejezés fordítása sokkal gyorsabbá válna. Addig is marad az, hogy több helyről [33] [34] [35] [36] [37] [38] kell „összevadászni” a keresett fordítást a legjobb eredmény érdekében.

6.2. Fordítássegítő rendszer

A fordítók munkáját, valamint tevékenységük koordinálását segítené egy, az OO.o-nál már említett, általánosan használható webes rendszer kialakítása, mellyel naprakészen követhető, hogy mely fordítás milyen állapotban van. Fontos, hogy legyen fordító memóriája, hogy a már lefordított szövegeket újra fel lehessen használni, valamint képes legyen adatcserére az elterjedt professzionális fordítást segítő szoftverekkel.

Hivatkozások

- [1] http://www.gnu.org/software/gettext/manual/html_mono/gettext.html
- [2] <http://www.gnu.org/software/gettext/gettext.html>
- [3] A magyar .po fordító csapat oldala:
<http://www2.iro.umontreal.ca/~pinard/po/registry.cgiteam=hu>
- [4] <http://forditas.fsf.hu/>
- [5] <http://www.uhulinux.hu/>
- [6] <http://www.suselinux.hu/>
- [7] <http://www.inf.unideb.hu/~matex/>
- [8] <http://www.inf.unideb.hu/~matex/magyaritas/index.html#eleje>
- [9] <http://www.szgti.bmf.hu/~ahorvath/latex/>
- [10] Elvlasztómodul T_EX-hez, OpenOffice.org-hoz:
<http://www.tipogral.hu/index.rhtml/12>
- [11] <http://www.szofi.hu/gnu/magyarispell/>
- [12] <http://magyarispell.sourceforge.net/>
- [13] <http://www.szabilinux.hu/ismerteto/Magyarul-HOGYAN.html>

- [14] A KDE honosító közösség honlapja:
<http://www.kde.hu/>
 - [15] <http://www.gnome.hu/>
 - [16] <http://linux.vv.hu/faq/gnome-faq/i18n.html>
 - [17] <https://office.fsf.hu/trans/index.php>
 - [18] <https://office.fsf.hu/extips/>
 - [19] Az OpenOffice.org honosítási projekt hivatalos oldala:
<http://hu.openoffice.org/>
 - [20] <https://lists.sch.bme.hu/wws/info/oohu>
 - [21] <http://www.staroffice.hu/forum.php3>
 - [22] A Magyar Mozilla Projekt weblapja:
<http://mozilla.fsf.hu/>
 - [23] Magyar nyelvű levelezőlista Mozilla-felhasználóknak:
http://groups.yahoo.com/group/mozilla_hu/
 - [24] <http://honositas.linux.hu/magyaritas/index.html>
 - [25] <http://tldp.fsf.hu/>
 - [26] A *Hogyan*-ok fordításával foglalkozó TLDP levelező lista:
<https://lists.sch.bme.hu/wws/info/linuxhowto/>
 - [27] <http://www.uhulinux.hu/licenc11/>
 - [28] http://www.lme.hu/forditas/GNU_GPL_hu.txt
 - [29] GNU-licencek magyarul:
<http://www.ham.hu/licencek/>
 - [30] <http://www.szabilinux.hu/>
 - [31] Szabad szoftver fordítással foglalkozó levelező lista:
<http://abos.linux.hu/wws/info/magyar>
 - [32] Az LME Honosítás Csoportjának weboldala:
<http://honositas.linux.hu/>
 - [33] <http://szotar.kiskapu.hu/>
 - [34] <http://www.freeweb.hu/infosz/>
 - [35] <http://i18n.kde.org/cgi-bin/kdedict.cgilang=hu>
 - [36] <http://tldp.fsf.hu/Forditas-HOGYAN/glossary.html>
 - [37] http://www.ebizlab.hit.bme.hu/~vi/ooffice/OpenOffice_Glossary_nontriv.html
 - [38] http://en.wikipedia.org/wiki/Main_Page
 - [39] Alphabet Soup: The Internationalization of Linux, Part 1
<http://www.linuxjournal.com/article.phpsid=3286>
-

Az alapértelmezett telepítések biztonsági hiányosságai és orvoslásuk

Harka Győző

<carlos@gamma.ttk.pte.hu>

Kivonat

A Linux testreszabhatósága és sokfelé ágazó terjesztései pozitív hatással vannak elterjedésére, de sokszor okoznak biztonsági problémákat. A legtöbb disztribúció tartalmaz elavult, vagy kevésbé biztonságos elemeket, illetve a csomagkezelőkre hagyatkozva könnyű átsiklani néhány beállítási problémán. Az előadásban ezekről a problémákról adok áttekintést, és megosztom megoldási ötleteimet is.

Tartalomjegyzék

1. Bevezetés	54
2. Fájlrendszer	54
3. Rendszermag	55
4. Hálózat	55
5. Démonok	55
6. Egyéb beállítások	56

1. Bevezetés

Minden, alapértelmezés szerint telepített rendszer tartalmaz biztonsági vagy teljesítménybeli kompromisszumokat. Sokan nem ismerik fel, hogy a látszólag így is tökéletesen működő rendszer sok olyan biztonsági rést hordoz, vagy hordozhat magában, amely akár percek alatt megszüntethető.

A telepítők a legtöbb disztribúcióban olyan hiányosságokat rejtenek magukban amelyek főleg a kezdő rendszergazdákat akadályozzák meg sok biztonsági fogás alkalmazásában. Kevesen vannak akik első telepítésük után rögtön kernel fordítással, foltozással, vagy akár csak rootkit-, vagy portellenőrző programok telepítésével kezdik.

2. Fájlrendszer

Általában elmondható, hogy minden felhasználónak joga van minden programot végrehajtani, és a legtöbb rendszerkönyvtárba is betekinthez. Ez a hozzáállás teszi lehetővé pl. a felhasználók kényelmes fájlcsereit, a `public_html` használatát, és egyéb kényelmes funkciókat. Ugyanakkor ez információ szivárgást is okozhat.

Az `umask` megfelelő beállítása nélkül (azaz legtöbb esetben a telepítés utáni alapértelmezett állapotában) minden új állomány olvasható mindenki (others) számára.

Érdekes korlátozni a `gcc` és egyéb fordítóprogramok használatát, az `ifconfig`, `route`, `ping` és egyéb hálózati felderítésekre használható programokat, illetve a `chsh` és a `chfn` SUID-es binárisokat.

Természetesen ez a „védelem” csak nehezíti, de nem teszi lehetetlenné egy támadó dolgát (esetleg néhány script-kiddie ellen bizonyulhat hasznosnak a módszer). Hasonló megfontolásból érdemes egyes könyvtáraktól is elvenni jogokat. Például a `chmod 711 /home` paranccsal megoldhatjuk, hogy felhasználóink ne is lássák egymás könyvtárait. Ugyanezt érdemes a `/etc` könyvtárra is alkalmazni.

Ha a rendszert nem egyetlen partíción használjuk, a `/etc/fstab` állományt is érdemes jól beállítanunk, például a `nodev`, `nosuid`, vagy akár a `noexec` opciókat is használhatjuk, tipikusan a `/home`, és a `/tmp` partícióira, illetve érdemes pl. a `/usr`-t normál körülmények között csak olvashatóan (`ro`) mountolni.

A legtöbb, frissen telepített rendszer nem használja ki az attribútumokat, amelyek közül az `a` (append only) – az adott állományhoz csak hozzáfűzni lehet, `i` (immutable) – az adott állományt nem lehet módosítani, `S` (sync) – az állomány változásai azonnal kiíródnak a diszkre támogatott az `ext2` illetve az `ext3` fájlrendszereken. Az `u` (undelete) – törlés utáni visszaállíthatóság és az `s` (shred) – törléskor fizikai adatmegsemmisítés attribútumok nem implementáltak a hagyományos fájlrendszerekben – így ezek mellőzése a telepítő részéről elnézhető.

A rendelkezésre álló attribútumokat azonban érdemes beállítani több helyütt is a telepítés után (ami szintén olyan utómunkálatként jelenik meg, amelyet érdemes lenne telepítés közben végrehajtani, és nem egy esetleg újdonsült rendszergazdára bízni).

Manapság érdemes lehet az `ext2`, `ext3` fájlrendszerek lecserélését is fontolóra venni, és átállni pl. az `XFS`-re.

3. Rendszermag

A Debian GNU/Linux disztribúció „generic” kernelei nem támogatják a kvóta használatát, és az ext3 (journaling) fájlrendszert, ami a naplózás használatával növelheti az adatbiztonságot.

Nem elérhető a csomag adatbázisokon keresztül a biztonsági javításokkal foltozott kernelek sem (*Grsecurity* [1], *Openwall*, *LIDS*, *International kernel patch*, *RSBAC*, *Medusa*...), csupán a forrás foltozásai.

Fordítás nélkül biztonsági kernellel inkább csak a kifejezetten biztonsági irányultságú disztribúciókban találkozhatunk (Pl. Adamantix [2], SELinux [3]).

Természetesen mindezek a problémák kernel foltozással, illetve újrafordítással megoldhatóak. A kernel cseréje illetve újrafordítása körültekintést igénylő feladat, főként ha például egy Grsecurity folt maximális biztonsági beállításai után szeretnénk még X szervert használni¹.

4. Hálózat

Ha hálózati kiszolgálónak használjuk rendszerünket, a biztonság témaköre kibővül a helyi felhasználók korlátozásai mellett a hálózati kliensek kérdéskörével.

Érdemes telepíteni az ismerős SATAN [4], (később SAINT [5], NESSUS [6]) nevű hálózati ellenőrző programot. A tűzfalat is érdemes átnézni illetve beállítani, és számos lehet megfelelően módosítani a `/proc/sys/net/ipv4` könyvtárban található icmp- és tcp-beállításokat is, a `sysctl` segítségével. A snort [7] (open source network intrusion detection system) futtatása is segíthet felfedezni a hálózatunkon kalózkodókat.

5. Démonok

Rengeteg az olyan program amely telepítése során, felhasználói beavatkozás nélkül is működő konfigurációt ad, ami azonban – mint minden rendszernek megfelelő beállítás – se nem biztonságos, se nem optimális.

A BIND (named), ami sokak szerint méltatlanul bitorolja a DJBDNS [8] helyét (ez inkább copyright különbségekre vezethető vissza), telepítés után működő „caching only” DNS szervert hoz létre. Ez viszont – sajnálatos módon – nyitott minden irányba, így névfeloldó szolgáltatásunkat a világon bárki, aki ránk talál használhatja. Hasonló problémát tud okozni a http-proxy (Squid) is, megfelelő korlátozások nélkül.

Az Apache mint az egyik legelterjedtebb webservert, méltán hirdeti alapértelmezett hibaüzeneteiben nevét, de sok „script kiddie”-t meg lehetne akadályozni a támadásokban a verziószám elrejtésével (persze a hibaüzenetek mellett a HTTP fejlécben is rábukkanhatunk a verziószámra).

A syslog gyengesége, hogy a `/dev/log` socket-en keresztül (mivel sok szolgáltatás nem privilegizált felhasználó nevében fut) a rendszer bármely felhasználója tetszőleges üzeneteket juttathat a naplófájlokba, ami nem csak félrevezető, de segítségével megoldható a root számára fenntartott lemezterület eliminálása, megakadályozva egy későbbi támadás naplózását. A naplózást könnyítendő még egy egyszerű bináris (`/usr/bin/logger`) is található a legtöbb terjesztésben, amit a felhasználó közvetlenül is meghívhat. Kiváltására a Syslog-ng [9] alkalmas – természetesen megfelelően beállítva.

¹Bővebben lásd Czákó Krisztián: „Könnyen alkalmazható Linux biztonsági megoldások” c. cikkét. (A szerk.)

A legegyszerűbb gyorsjavítás, ha a felhasználóink mind tagjai egy csoportnak (pl. users), hogy átadjuk a socketet a csoportnak, és kihasználva a Linux jogosultsági rendszerének diszkriminatív tulajdonságait, megvonjuk a csoport hozzáférését. A korrekt megoldás természetesen, ha felvesszük az összes naplózást használó démon felhasználóját egy csoportba, és ennek a csoportnak adunk jogot a socketre. Ez esetben persze ha új szolgáltatásokat telepítünk mindig adminisztrálnunk kell naplózó csoportunkat.

Az RPC (Remote Procedure Call) szolgáltatásokhoz kapcsolódó *portmap*, a Linux disztribúciók alapvető hálózati programjaival együtt kerül fel gépeinkre. A démon automatikusan indul, és a 111-es TCP portot megnyitva, lehetőséget ad a távoli számítógépek számára az esetlegesen a démon implementációiban rejlő hibák kihasználására, valamint az operációs rendszer identifikációjára.

Az *inetd* (internet superserver daemon) alapértelmezett konfigurációs állományai-ban általában engedélyezettek a TCP és UDP *echo*, *chargen*, illetve *discard* belső szolgáltatásai. Kedvelt DOS támadás a *chargen* (karakter generátor) szolgáltatást összefűzni egy *echo* (visszhang) szolgáltatással. Itt az „ököl szabály” annyi, hogy ha valamire nincs szükség, azt nem kell elindítani, azaz ki kell kommentezni.

A *Cron* démon alapértelmezésben bárki által használható, és mivel nem feltétlenül a felhasználó saját héját használja a futtatáskor, a fentebb már említett *chsh* paranccsal a lezárt (tipikusan */bin/false*) héjjal rendelkező felhasználóink, ha „előrelátóak” voltak, visszanyerhetik érvényes shelljüket. A */etc/cron.allow*, illetve a */etc/cron.deny* fájlokkal beállíthatjuk, hogy mely felhasználók használhatják, illetve nem használhatják a démon szolgáltatásait, illetve elvehetjük a SUID bitet a *chsh* parancstól.

Linux rendszereinktől gyakran kapunk hibajelentéseket e-mail formájában. Ezért sok terjesztés alapértelmezésben feltelepít egy, csak a *localhost*-nak relayező levelezőszervert. A 25-ös TCP-portot azonban nem csak a loopback interfészen nyitja meg a rendszer. Ennek következtében – néhány smtp-szervernél – a VRFY paranccsal lekérdezhető a felhasználók teljes neve a rendszerből.

6. Egyéb beállítások

A legtöbb disztribúcióban a */etc/security/limits.conf* beállításai is csak opcionálisak. Itt korlátozhatjuk a felhasználókra illetve csoportokra nézve a CPU-időt, az allokalható memória méretét, az egy időben futtatható folyamatok számát, a maximális veremméretet, az egyidőben megnyitható állományok számát, a maximális fájl méretet, a konkurens belépések számát, a folyamatok prioritását.

Sok program szolgál információval az általunk futtatott disztribúciónkról, illetve kernel verzióinkról is. Például a *BitchX* (egy népszerű konzolos IRC-kliens) alapértelmezett verzió válaszában is szerepel a kernel verzió.

A megoldás globális *.rc* fájlok, létrehozása, illetve ahol erre nincs mód, a beállítófájlok helyes verziójának elhelyezése a */etc/skel* könyvtárban. Természetesen ezek a problémák megjelennek szinte minden démonnál, az Apache-tól a *proftpd*-ig.

Egyes kliens programok alapbeállítása lazán kezeli a biztonságot. Sokszor az *ssh*-kliens telnet kapcsolatra vált ha nem tud *ssh* csatornát nyitni, illetve az engedélyezett cipherek közt is sokszor találunk gyengébbeket.

Néhány általánosabb program ami javítja a rendszer biztonságát:

A *chrootkit* [10] csomag segítségével az ismert „hátsó ajtók”² és rejtett processzek után kutathatunk rendszerünkön.

²backdoor

A tripwire [11] elsősorban a fájlrendszeren történt változásokat jelenti, különös tekintettel a konfigurációs állományokra.

A nagy disztribúciókban a telepített csomagok md5 checksum-jait is ellenőrizhetjük, *deb* csomagformátumnál a *debsums* csomag segítségével, *rpm* csomagformátumnál pl. az *rpm -V* paranccsal.

Természetesen ezek a szoftverek csak aktív rendszergazdai közreműködés esetén javítják a biztonságot!

Hivatkozások

- [1] Grsecurity: <http://www.grsecurity.net/>
- [2] Adamantix: <http://www.adamantix.org/>
- [3] SELinux: <http://www.nsa.gov/selinux/>
- [4] SATAN: <http://www.fish.com/satan/>
- [5] SAINT: <http://www.saintcorporation.com/index.html>
- [6] Nessus: <http://www.nessus.org/>
- [7] Snort: <http://www.snort.org/>
- [8] DJBDNS: <http://cr.yp.to/djbdns.html>
- [9] Syslog-ng: <http://www.balabit.hu/products/syslog-ng/>
- [10] Chrootkit: <http://www.chrootkit.org/>
- [11] TripWire: <http://www.tripwire.org/>
- [12] OpenWall: <http://www.openwall.org/>
- [13] *Gerhard Mourani: Securing & Optimizing Linux: The Ultimate Solution (v.2.0)* (OpenNA Inc., Kanada, 2001.05) ISBN 0968879306
Letölthető verzió (pdf):
<http://www.openna.com/products/books/sol/solus.php>
- [14] Bash Guide for Beginners – The Bash environment
<http://tille.soti.org/training/bash/ch03.html>
- [15] Slackware Linux Essentials – Permissions
<http://slackbook.lizella.net/chapter9-permissions.html>
- [16] Securing Debian HOWTO Chapter 4 After Installation
http://www.linuxsecurity.com/resource_files/host_security/securing-debian-howto/ch4.en.html
- [17] Network Security with /proc/sys/net/ipv4
http://www.linuxsecurity.com/articles/network_security_article-4528.html

- [18] IT Security Cookbook
<http://boran.linuxsecurity.com/security/references.html>
 - [19] Preventing Syslog Denial of Service attacks
<http://www.hackinglinuxexposed.com/articles/20030220.html>
 - [20] OpenSSH
<http://www.dfred.net/public/training/advanced-ssh.pdf>
 - [21] Securing Pam
<http://www.userlocal.com/security/secpam.php>
-

Szegedi Szemelvények Szabad Szoftverekről

Havasi Ferenc

<http://www.inf.u-szeged.hu/opensource/>

Kivonat

Az előadásom kicsit filozofikus hangvételben indul, amit azonban itt nem szeretnék kifejtetni – nagyon más „műfaj”. Amikről azonban írni szeretnék – és ami részletezve szóban talán egyébként is unalmas lenne – azok maguk a szabad szoftveres tevékenységeink.

Tartalomjegyzék

1. Oktatás	60
2. Szabadszoftveres fejlesztéseink	60
2.1. Symbian GCC	60
2.2. GCC ARM	61
2.3. CSiBE	61
2.4. JFFS2	62
2.5. Mozilla forrásvizsgálat	63
3. Szabad Szoftveres Világnap	63
4. Jövőbeli tervek	63

Open Source Laboratory, Szeged

Az Open Source Laboratory a Szegedi Tudományegyetem Szoftverfejlesztés Tanszékén belül működik. Tagjai oktatók, kutatók és diákok közül kerülnek ki, akik valamilyen formában aktív szerepet vállalnak a szabad szoftverek fejlesztésének egyes területein.

1. Oktatás

Az Informatikai Tanszékcsoport – és azon belül a Szoftverfejlesztés Tanszék – nagy hangsúlyt fektet arra, hogy a hallgatók lehetőleg mind kereskedelmi, mind szabad szoftveres technikákban tapasztalatot szerezzenek. A legtöbb, hallgatók számára hozzáférhető számítógép dual-bootos, és a programozási órák többségén nincs megkötve a platform vagy a fejlesztőkörnyezet, még ha ajánlott azért szokott is lenni.

Kifejezetten szabad szoftveres vonatkozásban két speciálkollégiumot hirdetünk meg minden évben *Nyílt forráskódú szoftverfejlesztés*, illetve *Linux kernel* címmel. Mindkét speciálkollégiumra általában nagy a túljelentkezés, ezért lehetőségünk van egy felvételi vizsgával szűrni a hallgatókat. Ez azt a nagyon szerencsés helyzetet idézi elő, hogy tényleg csak olyan diákok maradnak bent, akik hajlandóak erőfeszítést tenni az ügyért és valamilyen szinten értenek is már hozzá. UNIX felhasználói ismeretek, C programozási tudás és angol szövegértés az elvárt követelmények.

Igyekeztünk nem öncélúvá tenni a kurzust: ahelyett, hogy a hallgatók vizsgáznának, vagy olyan feladatokat oldanának meg, amik aztán a kukába kerülnek, inkább létező szabad szoftveres projektekbe próbáljuk bevonni őket. Ennek két előnye is van: a hallgató „valódi” tapasztalatot szerez ilyen téren és egyben a közösségnek is hasznára van.

2. Szabadszoftveres fejlesztéseink

Abban a nagy szerencsében volt részünk, hogy egy ipari partnerünk arra kért fel minket, hogy az általuk támogatott projekt eredményét tegyük szabad szoftverré. Ennek nagyon örültünk még akkor is, ha ez azt jelentette, hogy bizonyos részeket többször is újra kellett írunk, mire azok elfogadásra kerültek.

2.1. Symbian GCC

A Symbian mobiltelefonos operációs rendszerre az alkalmazások jelenleg GCC-vel fordulnak. Ez a GCC verzió (GCC 2.9-psion-98r2) már több mint hat éve, 1998-ban vált külön a GCC fővonalától, így még nagyon régi hibák sincsenek benne kijavítva. Sok esetben 01-nél magasabb szintű optimalizálás bekapcsolásával a GCC fordítás közben hibával leáll.

A projekt feladata ennek a problémának a megoldása volt, amit három módon tettünk meg:

- Az eredeti, GCC 2.9-psion-98r2 verziójú GCC-ben történt javításokkal.
- A GCC 2.95.3-as verziójának módosításával, hogy az Symbian kompatibilis kódot gyártson.

- GCC 3.0 módosításával. Itt már nagyon sok mindent kellett átvariálni ahhoz, hogy tényleg kompatibilis legyen a korábbi binárisokkal is.

Az így kapott GCC verzióknál már mind az `o3`, mind az `os` optimalizáló kapcsolók is működnek, amikkel 10% körüli kódméretcsökkenést, és 20%-os sebességnövekedést is el lehet érni.

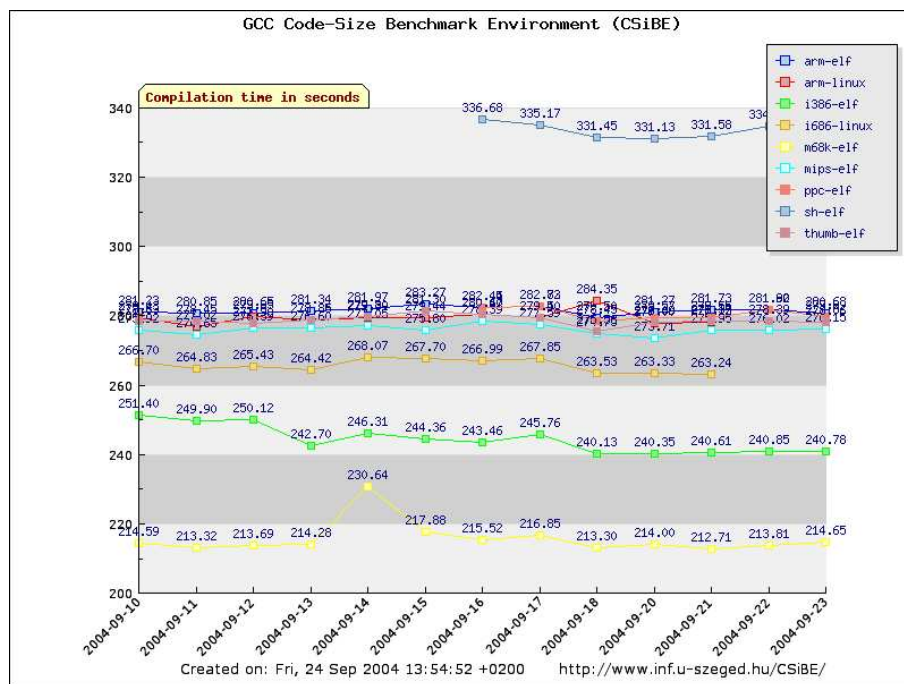
2.2. GCC ARM

Sok, napjainkban használt beágyazott rendszerben (például kézisámítógépekben és mobiltelefonokban) ARM processzort használnak CPU-ként. Ezeknek a rendszereknek van egy sajátosságuk: mindenből kevés van. Kevés memória, kevés háttértár, stb. Léteznek kereskedelmi ARM-os fordítóprogramok és a GCC-nek is van ARM kódot generáló része. Sajnos egy-egy kereskedelmi fordító a GCC-t fordítási méretben jelentős mértékben veri. Bár a különbség az újabb verziójú GCC-knél csökkent, de még így is számottevő.

A projekt célja ennek a különbségnek a csökkentése. A weboldalunkon olvashatóak a GCC általános optimalizáló részeihez és az ARM backendjéhez tett észrevételeink, megoldási javaslataink, amelyek többségét patch formájában el is juttattuk a karbantartóknak – ezek egy része már elfogadásra is került. Ezen patcheknek kb. 2,13% kódméret csökkenés köszönhető, és további jelentős fejlesztések vannak folyamatban.

2.3. CSiBE

A GCC egy nagyon nagy bonyolultságú szoftver, amelyen egyszerre nagyon sok fejlesztő dolgozik. Sokszor nem világos, hogy egy-egy patch pontosan milyen hatással lesz az egész rendszerre. Ezért alkottunk meg egy monitorozó keretrendszert, a Code-Size Benchmark Environmentet, azaz CSiBE-t. A rendszer naponta letölti a GCC változásait, lefordítja több platformra és méri a fordítási időt, a gyártott kód méretét, valamint Intel és ARM platformra a futási sebességet is. Az eredményeket adatbázisban tárolja, ahonnan az különféle lekérdezésekkel összehasonlító táblázatokon, diagramokon megtekinthető.



2.4. JFFS2

A legtöbb beágyazott rendszerben FLASH memória a háttértár – ilyen chip van a pen drive-okban és a digitális fényképezőgépek memóriakártyáin is. Tulajdonságaikban annyira eltérnek a „hagyományos” adathordozóktól, hogy huzamosabb használat esetén mindenképpen olyan fájlrendszer ajánlott rájuk, ami figyelembe veszi ezeket a különbségeket. A JFFS2 egy ilyen, kifejezetten FLASH-re tervezett fájlrendszer.

A FLASH azon kívül, hogy technikailag másképp viselkedik mint a megszokott adattárolók, meglehetősen költséges is. Az optimálisabb kihasználás miatt a JFFS2-be alpból beépítették a zlib alapú tömörítést – ezt használja a gzip is. Az adatok kis (4 kilobájtos) blokkokban kerülnek tömörítésre, majd tárolásra.

A projekt egyik célja volt, hogy ezt a beépített tömörítőt lecseréljük egy keretrendszerre, ahol a tömörítők csak modulok, és minden egyes blokk tömörítése során a rendszer automatikusan választja ki a „legjobbat”. Beállítható az, hogy mit jelent a legjobb (méret, sebesség). Újdonság a modell alapú tömörítők támogatása is – a modell egy különleges fájl, ahova az adott tömörítő tud információkat tárolni a különálló 4K-s blokkok közötti összefüggésről.

A keretrendszer (aminek egy része azóta bekerült a Linux kernelbe is) jó tömörítő modulok nélkül persze nem ér sokat, így a munkánkhoz tartozik egy kifejezetten ARM kódra specializált tömörítő, ami esetenként több mint 10%-kal is felülmúlja a zlib-et.

A projekt másik célja a mount idő csökkentése volt, ami elsősorban NAND FLASH-eknél okozott problémát. Ezt külön helyeken cache-ek tárolásával igyekeztük elérni, csökkentve ezzel a mount időben beolvasandó NAND lapok számát. A cikk írása idejére elkészült az első prototípusunk, ami egy – eddig az egyetlen tesztelt – esetben 52 másodpercről 16-ra javította a mount időt.

2.5. Mozilla forrásvizsgálat

Tanszékünkön folyik a Columbus névre hallgató, sokoldalú C++ elemzőprogram fejlesztése, amit többek között nyílt forráskódú programok (jelenleg Mozilla) elemzésén is teszteltünk. Az eredmények a honlapunkról letölthetők.

3. Szabad Szoftveres Világnap

A nemzetközi Software Freedom Day-t hivatalosan augusztus 28-án ünneplik – idén (2004-ben) először. Magyarországon egyedül nálunk, Szeged tartottuk meg, egyetemi város lévén azonban nem a hivatalos időpontban, hanem szeptember 12-én.

A rendezvény ingyenes volt, és körülbelül 120–150-en vettek részt rajta. Az előadók a Szegedi Tudományegyetem, az FSF.hu, az FSN.hu, az LME, az OnlineWeb Kft. és a Ritek Rt. „aktivistái” közül kerültek ki. A szünetekben open-source sütit osztottunk (forráskóddal, természetesen), könyveket és szabad szoftveres pólókat sorsoltunk, illetve árusítottunk önköltségi alapon. Támogatóink a Szegedi Tudományegyetem Informatikai Tanszékcsoportja, az OnlineWeb Kft. és a Kiskapu Kft. Linuxvilág magazinja volt.

4. Jövőbeli tervek

Szeretnénk az együttműködéseinket tovább erősíteni (elsősorban magyar) szervezetekkel, szabad szoftveres mozgalmakkal, azt használó cégekkel. Tervezzük olyan oktatási segédanyagok kidolgozását, amik megkönnyítik a szabad szoftverekben rejlő óriási tudásbázis oktatásban történő felhasználását, és ezzel a hallgatóknak valódi lehetőséget biztosítani, hogy „gyakorlásukat” másoknak is hasznos területen végezzék.

Hálózati határvédelem kialakítása GNU/Linux alapokon

Höltzl Péter

Kivonat

Előadásomban bemutatom az informatikai biztonság szerepét egy gazdasági szervezetben. Ismertetem a biztonságos határvédelmi rendszer kialakításának legfontosabb szervezési és megvalósítási feltételeit. Bemutatom, milyen GNU/Linux alapú eszközökkel lehet maradéktalanul megfelelni a szigorú védelmi rendszerrel szemben támasztott feltételeknek.

Tartalomjegyzék

1. A hálózati határvédelmi technológiák	66
1.1. Csomagszűrők	66
1.2. Alkalmazásszintű védelem	67
1.3. Tűzfal technológiák összehasonlítása	69
2. Kapcsolódó technológiák	69
2.1. Virtuális Magán-Hálózatok (VPN)	69
2.2. Személyes tűzfal	69
2.3. Betörés detektálás	70
2.4. Vírusszűrés	70
3. Üzemeltetési és támogatási folyamatok	70
3.1. A politika karbantartása	71
3.2. Biztonsági frissítések	71
3.3. Paraméterek monitorozása	71
3.4. Rendszeres naplófájl analízis	71
4. Összefoglalás	72

A biztonság helye a gazdasági szervezetben

A biztonság az a kedvező állapot, melynek megváltozása nem valószínű, de nem is zárható ki. Tehát az informatikai erőforrásainak *bizalmasságát*, *sértetlenségét* és *rendelkezésre állását* semmi nem veszélyezteti, de ez nem is zárható ki teljes bizonyossággal. Erőforrásaink bizalmasságán (Confidentiality) azt értjük, hogy csak korlátozott számú személy férhet hozzá. Sértetlenségen (Integrity) azt, hogy az erőforrás az eredeti állapotnak megfelel, teljes. Végül a rendelkezésre állás (Availability) azt jelenti, hogy a rendszer az eredeti rendeltetésének megfelelő szolgáltatásokat nyújtja a meghatározott helyen és időben. Az utóbbi években további két fontos tulajdonsággal jellemzik az informatikai erőforrások biztonságát. A hitelesség (Authenticity) azt jelenti, hogy az információ forrása az, amit megjelölnek, és tartalma eredeti. A letagadhatatlanság (Non-repuditation) pedig hiteles információ (bizonyíték) egy cselekvéssel kapcsolatban.

Az információ biztonsága manapság ugyanolyan üzleti követelmény, mint az üzemvagyon- és jogbiztonság, melynek jelen kell lennie a termelési, üzleti és irányítási folyamatokban egyaránt. Az előadás ennek gyakorlati megvalósításával foglalkozik GNU/Linux eszközök bemutatásával.

1. A hálózati határvédelemi technológiák

A védelmi intézkedések egyik sarkalatos pontja a számítógépes hálózatok határainak védelme, köznapi értelemben a *tűzfalak*.

Tűzfalak azok a hozzáférés-védelmi eszközök, melyek a Informatikai Biztonsági Szabályzat (I.B.Sz.) ide vonatkozó rendelkezéseit, szabályait kikényszerítik. A tűzfal felfogható olyan hozzáférés-vezérlési eszköznek (Access Control), amit Common Criteria-ban megfogalmazott felhasználói adatok védelmével foglalkozó FDP valamint a felhasználók azonosításával foglalkozó FIA osztályokban részletez. Ennek hatékonyságát az alkalmazott technológia határozza meg.

Napjainkban mind több helyen GNU/Linux rendszereket használnak kiszolgálóként, asztali kliensként és természetesen védelmi eszközként is.

1.1. Csomagszűrők

Az Internet hajnalán a hálózatok mindenféle védelem nélkül voltak összekötve. A kezdeti probléma az volt, hogy az Internetre kapcsolt szerverek minden kéretlen, nekik címzett csomagot megkaptak és feldolgoztak. A megoldás kézenfekvő volt, a hálózatok kijáratán már amúgy is ott lévő routerek fejlesztésével elérték, hogy azok csak bizonyos csomagokat engedjenek át. Ezek az eszközök különböző szabályok szerint csomagokat továbbítanak vagy dobnak el. A döntés kizárólag a TCP/IP protokoll internet (IP) és átviteli (Transport) rétegeinek fejlécein, tehát a csomag forrás- és célcímén, forrás- és célportján, valamint a fejlécekből nyerhető egyéb információkon (ellenőrző értéken és flag-eken) alapult. Erre már a korai Linux rendszerek is képesek voltak. Az első csomagszűrőt (*pf*) Alan Cox portolta az 1.3-as kernel sorozatba. A 2.0-ás sorozatba már a Joss Voss féle *ipfwadm* került. A 2.2-es sorozatban ismét egy új csomagszűrő alrendszer, az *ipchains* jelent meg, melynek alapjait Rusty Russel teremtette meg.

A csomagszűrők szabályrendszerét egy szabálylista (ACL) tartalmazza. A kernel minden beérkező csomag esetén végigfuttatja ezt a listát, és az első egyező szabálynál végrehajtja az ott beállított tevékenységet. Ez lehet elfogadás (ACCEPT), eldobás

(DROP) vagy visszautasítás (DENY). Amikor a csomagszűrő elutasít egy csomagot, a kliens felé ezt jelzi (TCP reset vagy ICMP hibaüzenettel), amikor pedig eldob, azt nem jelzi a kliens felé.

Mivel a TCP/IP megvalósítás nem képes bármekkora adatot egy az egyben átvinni, ezért az átviteli réteg ezeket az adatokat csomagokra bontja, amit szükség esetén az IP réteg tovább bont (fragmentál) még kisebb darabokra. Az egyes rétegek feladata, hogy a megérkezett darabokat összeillessze. Természetesen a megérkezett csomagok nem egyformák, ami támadási lehetőséget kínál. A csomagszűrők képtelenek az összetartozó csomagok (vagyis a kapcsolatok) felismerésére, és az oda nem tartozó „extra” csomagok eldobására. Ez azt jelenti, hogy csomagszűrő az oda nem tartozó csomagokat is átengedi. További problémát jelent, hogy az alkalmazásszintű protokollok nem minden esetben kommunikálnak egyetlen porton (pl. FTP, H.323 vagy az SQLNet). Ezek felismerésére a csomagszűrőknek bele kell látnia az alkalmazási rétegbe, és annak megfelelően a szükséges portra menő csomagokat át kell engednie. Ennek a problémának a megoldására születettek meg az állapottartó csomagszűrők. Igaz, hogy a 2.2-es kernelsorozat már képes volt ilyen kapcsolatok átengedésére kernel modulok segítségével (pl. FTP, RAUIDO vagy IRC), de az igazi állapottartási megoldás a 2.4-es kernelhez a *netfilter*rel [1] született meg.

Az állapottartó csomagszűrők a csomagokból nyert további információk alapján képesek a kapcsolatokat észlelni, az oda nem tartozó csomagokat eldobni. A Rusty Russel, majd később a Netfilter Team¹ által fejlesztett állapottartó csomagszűrő a *netfilter/iptables* a 2.4.6-os kernel verzió óta része a Linuxnak. A Netfilter képes kapcsolatok felismerésére és követésére (connection tracking). A beérkező csomagokhoz a következő állapotokat rendeli: INVALID, NEW, RELATED, ESTABLISHED, DNAT és SNAT.

A csomagszűrők további szolgáltatása a forrás és célcím hamisítás, ami azt jelenti, hogy képesek az eredeti csomag vagy kapcsolat forrását megváltoztatni (eltakarni), illetve eredeti céljától azt eltéríteni. Ezt a technológiát hívják címfordításnak (NAT – Network Address Translation).

A GNU/Linux rendszerekhez sok grafikus felületű szabálylista szerkesztő található. Legismertebb a *firewallbuilder* [2] vagy a *KMyFirewall* [3].

1.2. Alkalmazásszintű védelem

Minden csomagszűrő technológiának hibája, hogy minden kapcsolatot és csomagot átenged, amennyiben annak forrása és célja megfelel a szabályrendszernek. Ez nem véd meg az alkalmazásszintű támadásoktól.

A kliensek és szerverek védelmére elsőként *bastion hosztokat* hozták létre, melyek szervergépek közvetlen kapcsolattal, több hálózati szegmens felé. A bastion hosztok route-olási feladatokat nem látnak el, ezért a felhasználóknak a másik hálózat erőforrásainak eléréséhez be kell jelentkeznie a szerverre. Bastion hoszt alkalmazásakor a szűrési lehetőségek kimerülnek a *felhasználók autentikálásában*, illetve a felhasználói csoportonkénti (esetleg egyéni) *chroot környezetek* használatában. A bastion hosztok adminisztrálása igen nehéz, használhatósága erősen limitált.

Az *alkalmazásszintű tűzfalak* jelentették az első lépést a bastion hosztok esetében kényelmetlen belépési procedúrával. Ehhez speciálisan felkészített kliensek kellenek, melyek – a csomagszűrőkkel ellentétben – nem közvetlenül a szerverhez kapcsolódnak, hanem a *protokoll proxyhoz*. A kapcsolat ezen a ponton végződik, a proxy értelmezi a

¹A Netfilter csapatnak számos elismert magyar fejlesztője van.

kérést, és amennyiben az megfelel a politikának (megfelelő forrással és céllal rendelkezik, illetve betartja az adott protokollt) a proxy újabb kapcsolatot nyit a szerver felé, elküldi a kérést, megvárja a választ, és ha az szintén betartja a protokollt, az eredményt visszaküldi a kliensnek. A protokoll értelmezés megvalósítás függő, de általában a protokoll parancsainak, a parancsok sorrendjének és paramétereinek ellenőrzését jelenti. Természetesen az alkalmazásszintű védelem esetében minden egyes átvitt protokollra célproxyt kell fejleszteni. Mivel ez nem minden esetben megoldható, ennek megoldására született az általános proxyt (Plug, vagy General proxy), melyek minden egy-csatornás protokoll átvitelére alkalmas. Alkalmazásszinten nem véd, viszont a tűzfal politika egységes karbantarthatósága szempontjából hasznos, továbbá véd az IP szintű támadások ellen, mivel a proxyk két, IP szinten is külön kapcsolatot kezelnek.

Ennek a védelmi megoldásnak hátránya, hogy speciálisan felkészített, a proxyt használni képes kliensre van szükség. Sajnos a kliensek nagy része nem alkalmas erre. Ennek a kényelmetlenségnek kiküszöbölésére fejlesztették ki a *transzparens proxykat*. A megoldás lényege, hogy csomagszűrő segítségével a kapcsolatokat eredeti céljuktól eltérítik, és a proxyknak adják. A megoldás haszna, hogy már nincs szükség speciálisan felkészített kliensekre. Linux rendszerekre szabadon elérhető proxy megoldások a FWTK [4] (firewall toolkit) és bővítményei, illetve egy-egy protokollra lehet egyedi projektek keretében készülő proxykat telepíteni.

A proxy tűzfalak is képesek címhamisításra, mely a csomagszűrőkhöz képest fordítottan működik. Mivel a proxy a kliens és szerver oldalon független kapcsolatokat kezel, ezért a szerver oldalon keletkezett kapcsolat forrása a tűzfal IP címe lesz. Ez a funkcionalitás megegyezik a csomagszűrők forrás NAT funkciójával. A proxyk esetében a NAT szolgáltatás azt jelenti, hogy szerver oldali kapcsolat forrása nem a tűzfal IP címe lesz, hanem valami más, pl. az eredeti kliens címe. A proxyk képesek a kapcsolat eredeti célját is megváltoztatni (DNAT).

Az Interneten elérhető szolgáltatások fejlődésével mind több és több protokoll lett elérhető, valamint megszülettek az összetett vagy beágyazott protokollok. Ilyenkor egy protokollt egy másik protokoll adat tartalmába ágyaznak (hasonlóan a tunnelezéshez). Ilyen megoldást alkalmaznak a forgalom autentikálására, az adatok bizalmasságának és sértetlenségének megőrzéséhez használt, a Netscape által fejlesztett SSLv3 [5] (Secure Socket Layer), illetve TLSv1 [6] (Transport Layer Security) protokollok, melyekbe további – hagyományos, pl. HTTP vagy POP3 – protokollokat ágyaznak. Az ilyen forgalommal a hagyományos proxyk nem tudnak mit kezdeni, ezért volt szükség a moduláris proxyk fejlesztésére.

A megoldásban a proxy modulok képesek az adattartalmat további moduloknak átadni, melyek segítségével sokkal mélyebb protokoll ellenőrzésre nyílik lehetőség (pl. SSL proxyba ágyazott POP3 proxy). Adott a lehetőség további proxy kombinációk, akár kettőnél több proxy egymáshoz kapcsolására (pl. SSL proxyba ágyazott POP3 proxy, mely tovább adja az adatot egy MIME proxynak majd az egy vírusszűrő modulnak). A megoldás további haszna, hogy a proxy modulok ezentúl kizárólag a protokoll értelmezéssel foglalkoznak, mert a kapcsolat fogadást, továbbkapcsolódást vagy a policy döntéseket más modulok végzik. Ez által az egyes modulok kevesebb funkcionálitással rendelkeznek, ami nagyobb stabilitást és kevesebb lehetséges hibát eredményez². Ilyen proxy megoldást alkalmaz például a Zorp GPL [7] és Professional.

²Keep It Simple and Stupid, a hagyományos unixos programfejlesztési elv.

1.3. Tűzfal technológiák összehasonlítása

A technológiák kiválasztásánál fontos szempontok a felhasznált sáv szélesség és a felhasználó kliensek száma. Általánosságban elmondható, hogy a csomagszűrők teljesítményét első sorban a route-olt sáv szélesség és a szabályláncok (ACL) hossza határozza meg. A proxyk esetében a párhuzamos (konkurens), és az egyszerre induló kapcsolatok száma sokkal fontosabb, mert a proxyk tipikusan felhasználói térben (userspace) működnek, ezért memóriát, fd-t, processzor időt stb. kell biztosítani számukra. Ez erősen függ az ütemezőtől is. Az alkalmazásszintű védelem tehát erősebb hardvert igényel rendelkezik, amiért cserébe hatékonyabb védelmet kapunk.

2. Kapcsolódó technológiák

A tűzfal technológiák alkalmazása nagyban csökkenti vállalati erőforrások kompromittálódását, ám koránt sem elégségesek a teljes körű védelem elérése szempontjából.

2.1. Virtuális Magán-Hálózatok (VPN)

A moduláris proxy SSL-terminálási képességein kívül szükség lehet a szervezet hálózatán kívül eső kliensek, és a belső szerverek szolgáltatásai között kapcsolatot teremteni. Ezt biztonságosan csak erős autentikációval és kriptográfiával védett kapcsolatok segítségével lehet megoldani. Sok esetben nem elégséges az SSL protokoll használata, melyeknek oka nem abban rejlik, hogy az SSL kriptográfiai vagy autentikációs képességei gyengék lehetnek, hanem sok esetben a szolgáltatások túl sok csatornát (portot) használnak a kommunikációra, vagy más okok miatt ezt kényelmetlen lenne megoldani. Ilyen esetekben használható az IPSec protokoll.

Az IPSec (IP Security) két átviteli módot támogat: a transport és a tunnel módokat. A transport mód csak az IP-csomagok adattartalmát rejtjelezi (payload), míg a tunnel mód az IP fejlécekkel együtt teszi ezt. Az adatforgalom rejtjelezésének neve ESP (Encrypted Security Payload). Lehetőség nyílik az IP fejlécek digitális aláírására is, AH (Authentication Header). Az AH és ESP módok együtt is használhatóak.

Az IPSec működéséhez szükséges, hogy a küldő és fogadó oldal publikus kulcsok segítségével session kulcsot cseréljenek, melyek az úgynevezett SA-kban (Security Association) tárolódnak. A kulccseréhez az ISAKMP/Oakley protokollt (Internet Security Association and Key Management Protocol) használják, ami lehetőséget nyújt a kulccserére RSA kulcsok, jelszavak és X.509-es tanúsítványok segítségével.

A Linux kernelhez a 2.2-es verzió óta létezik IPSec támogatás³, a FreeS/WAN [8], melynek fejlesztése 2004-ben befejeződött. Utódjai az Openswan [9] és strongSwan [10] csomagok.

A csomag két fő részből áll. Az IPSec-et megvalósító KLIPS csomag és a kulccserét (IKE - Internet Key Exchange) megvalósító Pluto. A 2.6-os kernel verzió natív IPSec támogatást tartalmaz KAME [11], a hozzátartozó IKE megvalósítás a Racoon [12].

2.2. Személyes tűzfal

A személyes tűzfalak általában csomagszűrők, melyek az adott lokális gép védelmét szolgálják. Ezek az eszközök általában szolgáltatást nem nyújtanak a hálózat többi

³patch formájában

felhasználója felé, viszont csomagszűrő nélkül védtelenek lehetnek az Internetről érkező támadásokkal szemben. Ezek a támadások gyakran nem a klienst magát, hanem rajta keresztül a hálózat többi elemét próbálják támadni. Ezek védelmére készülnek az MS Windows környezetben megismert személyes tűzfalak (personal firewall). A GNU/Linux felhasználók sokkal szerencsésebbek Windowst használó társaiknál, mivel nincs szükségük extra alkalmazás telepítésére, ha meg szeretnék védeni gépeiket, erre megfelelő egy saját (általában nagyon egyszerű) csomagszűrő szabályokat beállító script, de léteznek grafikus és webes felületű alkalmazások is a feladatra [2] [3].

2.3. Betörés detektálás

A védelmi intézkedések általában nem merülnek ki megelőző intézkedések megtételében. Fontos feladat hárul az esetleges baj érzékelését elősegítő (detektív) eszközökre, intézkedésekre. Ezek az úgynevezett betörés érzékelő rendszerek (Intrusion Detection eszközök, IDS), melyeknek több fajtája lehet.

A hoszt IDS általában a hoszt rendszer felhasználóinak tevékenységét figyeli, ellenőrzi. Ide tartoznak pl. a rootkit keresők (chkrootkit) is. A fájlrendszer sértetlenségének vizsgálatára is több alkalmazás áll rendelkezésre. Ennek célja, hogy figyelemmel követhessük, ha valamilyen állományban (általában a futtatható és konfigurációs fájlokról van szó) változás történt. Több ismert fájlintegritást ellenőrző alkalmazás érhető el. Ilyen a legendás, sajnos nem teljesen szabad Tripwire [13], illetve a GPL-es Aide [14].

A hálózati vagy nIDS a hálózati forgalom figyelésével keres támadási kísérletre utaló nyomokat. Két alapvető fajtája létezik. Az első a víruskeresők mintaillesztési módszerét használva támadási mintákat keres. A második a kliensek viselkedésmintáit figyelve keres támadásra utaló nyomokat. Ezek legjobb megvalósítása a Snort [15]. A Snortot érdemes dedikált hardverre telepíteni, mely legalább két interfésszel rendelkezik. Az egyik egy ún. érzékelő (sensor) interfész, mely ún. promiscuous módban „hallgatózik” IP-cím nélkül és egy menedzsment interfész, mely segítségével a rendszer karbantartható. A snort a megkapott csomagokat preprocessorok segítségével feldolgozza, majd az előfeldolgozás után különböző elemző plugineknek adja tovább. A rendszer képes riasztásokat generálni, illetve lehetőség van a riasztásokat tovább feldolgozva külső (szoftver) eszköz segítségével kapcsolatok megszakítására is.

2.4. Vírusszűrés

A határvédelem másik fontos feladata lehet vírusok keresése és eltávolítása. Ennek leggyakoribb színtere az SMTP protokoll vírusmentesítése. A tűzfalak a levélforgalom átvitelére leggyakrabban natív proxyt (valamilyen MTA-t) használnak. Célszerű ezeket az MTA-kat felkészíteni az átmenő forgalom vírus- és spamszűrésére. A víruskeresők nagyrésze daemonos és parancssori üzemmódban képes működni. Gyakran szükség van az MTA és az víruskereső közé valamilyen illesztő eszközre is, pl. amavis. A legnépszerűbb szabad víruskereső motor a ClamAV [16], melyhez gyakori vírus adatbázis frissítések is tartoznak. A spamek szűrésére pedig széles körben ismert eszközök állnak rendelkezésre (Spamassassin [17], Bogofilter [18]).

3. Üzemeltetési és támogatási folyamatok

A meglévő technológia természetesen csak akkor ér valamit, ha részletesen megszervezett üzemeltetési folyamatok is tartoznak hozzá. Nincs veszélyesebb és sérülékenyebb

egy magára hagyott rendszernél. Ezért fontos az üzemeltetési feladatok meghatározása és alkalmazása.

3.1. A politika karbantartása

A tűzfalak szabályrendszerének karbantartása nagyon fontos üzemeltetési feladat. Minden tűzfal telepítéskor keletkezik egy, a pillanatnyi igényeknek megfelelő szabályrendszer, ami követi az igényeket. Sokszor előforduló hiba, hogy a szabályokat kizárólag bővítik, a nem használt szabályokat nem törlik. Ennek kiküszöbölésére a szabályokat rendszeresen ellenőrizni kell, a szükségtelen szolgáltatásokat le kell törölni. Az új szolgáltatások felvételére használjunk protokoll kérés űrlapot, melyen legyen kötelező feltüntetni a szolgáltatás célját és időtartamát.

3.2. Biztonsági frissítések

Szoftverek nem léteznek hiba nélkül, ez alól a védelmi szoftverek sem kivételek. Javasolt az Internet fórumainak [19] [20], valamint a szoftverek weboldalainak rendszeres ellenőrzése, a hibabejelentések, kiadott javítások nyomon követése, és a javítások mielőbbi telepítése. Igyekezzünk minden alkalmazást úgy frissíteni, hogy minimálisan térjünk el a használt alkalmazás verziószámától, mivel az újabb verziók esetleg nem teljesen azonosan működnek a korábbiakkal. Az ebből adódó hibák elkerülésére inkább foltozzuk (patch) a rendszeren futó alkalmazást, mint újabb verziót telepítünk.

3.3. Paraméterek monitorozása

A védelmi eszközök fontos üzemeltetési eszköze a monitorozás, melynek segítségével üzemelő rendszerünk állapotáról kapunk akár azonnali visszajelzést. A paraméterek monitorozásának két fontos fajtája létezik. Az első esetben a monitorozott eszköz pillanatnyi állapotát követjük figyelemmel. Amennyiben az valamilyen határértéket meghalad, a rendszer riasztást generál. Linux rendszerre elérhető legismertebb monitorozó alkalmazás a Nagios [21]. A rendszer egy monitorozó központból áll, mely periodikusan teszteli a beállított eszközöket. Képes igazodni az üzemeltetők munkarendjéhez. A másik esetben az eszközök állapotát regisztráljuk és ábrázoljuk, melyek segítségével trendeket elemezhetünk és esetleges problémákat előre jelezhetünk. Ennek legismertebb eszközei a The Multi Router Traffic Grapher (MRTG) [22] és a Round Robin Database Tool (RRDTool) [23].

3.4. Rendszeres naplófájl analízis

A logolvasás a legkevésbé kedvelt adminisztrátori tevékenység. A cél, hogy az ismert és veszélyt nem jelentő aktivitást kiszűrve a megmaradó naplóbejegyzés sorokat elemezve a rossz szándékú aktivitásról tudomást szerezzünk. Erre a feladatra több, nagyon jól használható eszköz áll rendelkezésre a Linux rendszereken. A Logcheck [24] periodikusan (crontab) átszűri az utolsó ellenőrzés óta keletkezett naplóállományokat és ismert logmintákat keres. A keresési mintákat szabályos kifejezésekkel (regex) lehet megadni, de a Debian [25] GNU/Linux rendszereken több csomag tartalmazza az előre megírt logcheck mintákat. A logcheck szűrés eredménye lehet ún. paranoid üzemmódú, ekkor az összes ismeretlen logüzenetet elküldi, illetve létezik több más üzemmódja, melyben csak az ismert és támadásra, rendellenességre utaló üzenetek kerülnek

továbbításra. A biztonságkritikus helyeken a paranoid üzemmód javasolt. Az fwlog-watch [26] csomagszűrők naplóinak elemzésére szolgáló eszköz, mely periodikusan (crontab) ellenőrzi a keletkezett naplófájlokat, de csak a csomagszűrő logjait. Az online logellenőrzés eszköze a swatch [27], ami a keletkezett bejegyzéseket azonnal (online) dolgozza fel. Amikor egyezést talál, e-mail riasztást küld. A swatch elsősorban ismert üzenetek azonnali (realtime) keresésére használható, például signalok, egyebek.

4. Összefoglalás

A fent ismertetett eszközökön és példákon keresztül látszik, hogy GNU/Linux eszközökkel azonos, sokszor jobb védelmi értékkel rendelkező rendszer építhető. Fontos, hogy a GNU-rendszerek teljes támogatást adnak komoly tűzfalrendszerek, VPN-ek, vírus és betörésdetektáló rendszerek kiépítéséhez, valamint ezek üzemeltetéséhez.

Hivatkozások

- [1] <http://www.netfilter.org/>
- [2] <http://www.fwbuilder.org/>
- [3] <http://extragear.kde.org/apps/kmyfirewall.php>
- [4] <http://www.fwtk.org/>
- [5] <http://www.openssl.org/>
- [6] <http://www.gnu.org/software/gnutls/>
- [7] <http://www.balabit.hu/products/zorp/>
- [8] <http://www.freeswan.org/>
- [9] <http://www.openswan.org/>
- [10] <http://www.strongswan.org/>
- [11] <http://www.kame.net/>
- [12] <http://www.kame.net/racoon/>
- [13] <http://www.tripwire.org/>
- [14] <http://www.cs.tut.fi/~rammer/aide.html>
- [15] <http://www.snort.org/>
- [16] <http://www.calmav.net/>
- [17] <http://spamassassin.apache.org/>
- [18] <http://bogofilter.sourceforge.net/>
- [19] <http://www.securityfocus.com/>
- [20] <http://lists.netsys.com/pipermail/full-disclosure/>

- [21] <http://www.nagios.org/>
 - [22] <http://people.ee.ethz.ch/~oetiker/webtools/mrtg/>
 - [23] <http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/>
 - [24] <http://www.astro.uiuc.edu/~r-dass/logcheck/>
 - [25] <http://www.debian.org/>
 - [26] <http://fwlogwatch.inside-security.de/>
 - [27] <http://swatch.sourceforge.net/>
-

Tűzfalfürtök állapottartó Netfilter tűzfalakkal

Kovács Krisztián

<hidden@sch.bme.hu>

Kivonat

A Linux csomagszűrője – a Netfilter és az arra épülő `iptables` – igen sokoldalú, rugalmasan használható, és a nagyszámú rendelkezésre álló modullal könnyen bővíthető eszköz. Megjelenésekor egyik fő előnye a korábbi Linuxos csomagszűrőkkel szemben a modularitása, és az állapottartó csomagszűrés lehetősége volt. Az állapottartást a Netfilter kapcsolatkövető alrendszere teszi lehetővé azzal, hogy képes felismerni, hogy egy adott csomag melyik kapcsolathoz tartozik, és nyomon követni, hogy az egyes kapcsolatok milyen állapotban vannak. Az `iptables` szabályrendszerünkben ezeket az adatokat felhasználva igen sokrétű lehetőségeink adódnak, például könnyedén átengedhetjük a már kiépített kapcsolatokhoz tartozó válaszcsomagokat, vagy az FTP kapcsolatok adatcsatornáját anélkül, hogy túlságosan „megnyitnánk” tűzfalunkat. A maszkolást és a különféle átirányításokat (REDIRECT, DNAT) végző NAT alrendszer is a kapcsolatkövető rendszer adataira épül.

Amennyiben azonban nagyobb rendelkezésre állású tűzfalfürtöt szeretnénk építeni, szembetalálkozunk az állapottartó csomagszűrők egy hátulütőjével. Ahhoz ugyanis, hogy failover esetén az újonnan működésbe lépő tűzfal problémamentesen működjön, azonos szabályrendszer mellett is szüksége van a kapcsolat-állapottábla tartalmára. Szükséges tehát egy olyan bővítés, ami lehetővé teszi ezen állapottábla fürtön belüli replikációját: ez a Harald Welte és jómagam által fejlesztett `ct_sync`.

A cikk röviden ismerteti a központi szerepet játszó kapcsolatkövető rendszer felépítését, majd sorra veszi a `ct_sync`, illetve az azzal épített rendszerek alapvető komponenseit és működését. Részletesen kitér a felmerült problémákra, azok megoldásaira, végül a tapasztalatok és a jövőben még megoldandó feladatok összegzésével zárul.

Tartalomjegyzék

1. Csomagszűrő tűzfalfürtök	77
1.1. A VRRP protokoll	77
1.2. Állapottartó csomagszűrők	78
2. A Netfilter connection tracking rendszere	79
3. Állapottartó failover megoldások	80
3.1. „Szegény ember HA fürtje”	80
3.2. Aktív állapotreplikáció	81
4. A <code>ct_sync</code> felépítése és működése	81
4.1. A replikációs protokoll	83

4.2. Egyéb szolgáltatások	84
5. Hogyan használjuk?	85
5.1. VRRP konfiguráció	86
5.2. A <code>ct_sync</code> konfigurációja	86
6. További lehetőségek	86
7. Köszönetnyilvánítás	87

1. Csomagszűrő tűzfalfürtök

A csomagszűrő tűzfalak napjainkban a számítógép-hálózatok igen lényeges és gyakran alkalmazott alkotóelemei. Az egyre nagyobb népszerűségnek örvendő, otthoni felhasználásra készült DSL „routerektől” kezdve a közepes, néhány száz vagy ezer gépes hálózatokat védő vállalati tűzfalakig szinte minden hálózatban megtaláljuk őket. Ezek az eszközök több szempontból is kritikus elemei a hálózatnak. Egyrészt kritikusak biztonsági megfontolásokból, hiszen komoly problémát jelent amennyiben valahogyan meg lehet kerülni a tűzfal szabályrendszere által reprezentált biztonsági házirendet. Másrészt pedig rendelkezésre állás szempontjából is igen kritikusak: a legtöbb esetben az ilyen tűzfal SPOF¹, ami komoly kockázatot jelent a hálózat üzemeltetése szempontjából.

1.1. A VRRP protokoll

A megbízhatóság növelése céljából csomagszűrő fürtöket építettek, általában a routerekhez már rendelkezésre álló, nagy rendelkezésre állást biztosító megoldások felhasználásával. Ilyen például a VRRP, azaz a *Virtual Router Redundancy Protocol* [1], amely lehetővé teszi olyan virtuális routerek (és tűzfalak) kialakítását, ahol a tényleges feldolgozást és továbbítást egy több routerből álló fürt végzi.

A VRRP fürtök n tagja egyetlen virtuális routert alkot. Minden virtuális router esetében egy – a VRRP protokoll alkalmazásával választott – node, a *master* végzi a továbbítást és feldolgozást. A virtuális router címeit² mindig az aktuális master veszi fel, így biztosítva, hogy a virtuális routernek küldött csomagok a megfelelő helyre érkeznek meg. A fürt minden egyes tagján be kell állítani a következőket:

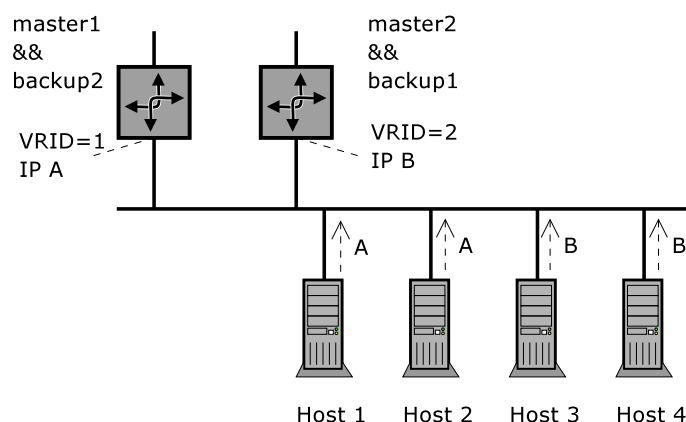
- azon virtuális routerek egyedi azonosítóját (VRID), amelyekben az adott node részt vesz,
- az egyes virtuális routereken az adott node-hoz rendelt prioritásokat,
- a virtuális routerek paramétereit (például a virtuális IP címeket).

A prioritás értékeknek annak eldöntésében van szerepe, hogy a virtuális routereken belül melyik node töltse be a master szerepét: a legnagyobb prioritású működő node lesz a választási protokoll győztese.

A VRRP alapvetően master–slave felépítésű virtuális routerek létrehozására alkalmas, így ha pusztán egyetlen virtuális routert tekintünk, egy kivételével a fürt valamennyi tagja kihasználatlan marad. A gyakorlatban ezért általában egy kicsit bonyolultabb módon használják. Az 1. ábrán látható egy tipikus, két routert (vagy tűzfalat) és két virtuális routert alkalmazó hálózati elrendezés. Lényege, hogy két virtuális routert definiálunk (X és Y), A és B (virtuális) IP címmel. Mindkét virtuális csoportban mindkét valódi node benne van, de a prioritásokat úgy választjuk meg, hogy amennyiben mindkét node működik az egyik az X, a másik pedig az Y virtuális routernek lesz a maste-re. A klienseket úgy konfiguráljuk, hogy egy részük A-t használja alapértelmezett átjáróként, a másik részük pedig B-t, így a terhelés megoszlik a két virtuális – és

¹Single Point of Failure, azaz olyan eszköz, amelynek meghibásodása az egész rendszert működésképtelenné teszi.

²Ez nem csak virtuális IP címeket, de virtuális MAC címeket is jelent; ez a megoldás az adatkapcsolati réteg szintjén is biztosítja a gyors átkapcsolást.



1. ábra. Tipikus VRRP architektúra

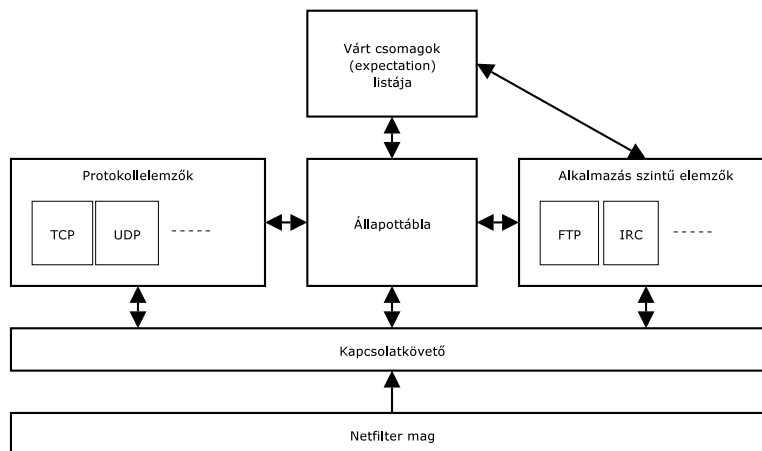
így a két valós router között is. Bármelyik router kiesése esetén a fürt másik tagja átveszi mindkét virtuális routerben a master szerepét, így a kliensek átkonfigurálása nélkül működhet tovább a hálózat.

1.2. Állapottartó csomagszűrők

Sajnos az eddig elmondottaknak van egy szépséghibája: a vázolt módszer csak akkor működik, ha a fürtök tagjai állapotmentesek. Ez egy csomagszűrő esetében azt jelenti, hogy minden egyes csomag esetében pusztán a (statikus) konfiguráció (szabályrendszer) és az adott csomag különféle jellemzői alapján születik meg a döntés, hogy az adott csomaggal mi fog történni. Ezzel szemben egy állapottartó csomagszűrőben lehetőség van arra, hogy a döntési folyamatban a tűzfal aktuális állapotát is figyelembe vegyük, így lehetőségünk nyílik arra, hogy az egyes csomagok közötti összefüggések ismeretében hozzuk meg a döntéseinket.

Az állapottartó csomagszűrők által biztosított lehetőségek bemutatására általánosan használt példa a következő: tegyük fel, hogy van egy tipikus desktop célokra használt számítógépünk, amit egy csomagszűrővel szeretnénk védeni. Mivel a felhasználó általában elvárja, hogy az ő számára ne jelentsen jelentős megszorítást a tűzfal, minden, erről a gépről kezdeményezett TCP kapcsolat csomagjait kiengedjük a csomagszűrőn. Ezt könnyen megtehetjük akár állapotmentes módon is, azonban komoly problémáink lesznek a válaszcsoomagokkal: nem tudjuk ugyanis eldönteni, hogy egy bejövő TCP csomag vajon egy általunk küldött csomagra érkező válasz vagy nem. Állapottartó csomagszűrő alkalmazásával lehetőségünk van ezeket a csomagokat aszerint osztályozni, hogy egy általunk kezdeményezett kapcsolathoz tartoznak-e, és csak ebben az esetben átengedni. Láthatjuk tehát, hogy már egy ilyen egyszerű probléma megoldása is mennyire nehézkes lehet, ha nem tudunk állapotokat tartani az egyes csomagok feldolgozása között.

Amennyiben tehát fürtünkben állapottartó csomagszűrőket használunk, valamilyen módon biztosítanunk kell, hogy amikor egy új node veszi át a master szerepét akkor az rendelkezzen a kiesett node állapottáblájának a lehetőségekhez mérten legteljesebb másolatával. Ennek biztosítására több megoldás is szóba jöhet, ezek tárgyalása előtt azonban előbb tekintsük át, hogy miből is állnak ezek az állapotadatok.



2. ábra. A connection tracking rendszer felépítése

2. A Netfilter connection tracking rendszere

A Netfilter/iptables olyan állapotartó csomagszűrő, amelyben az állapotinformációk az egyes *kapcsolatokhoz* kötődnek. Kézenfekvő, hogy egy TCP kapcsolatot megfeleltessünk egy csomagszűrőbeli kapcsolatnak, de a kapcsolat fogalmának kiterjesztése révén akár más protokoll esetén is beszélhetünk kapcsolatokról³. A *connection tracking*, azaz kapcsolatkövető rendszer feladata az egyes kapcsolatok és az ezekhez kötődő információk nyilvántartása és folyamatos frissítése.

A connection tracking rendszer vázlatos felépítését láthatjuk a 2. ábrán. Mint a Netfilter általában, ez a rendszer is moduláris felépítésű, könnyen bővíthető mind protokoll- mind alkalmazás szintű elemző modulokkal. Ezek mindegyike a kapcsolat-állapot-táblában tárolt adatokkal dolgozik, és a saját adatait is a tábla bejegyzéseiben tárolja.

A különféle elemző modulok regisztrálhatnak úgynevezett *várt kapcsolatokat* (*expectation*). Amennyiben egy ilyen regisztrált kapcsolat kialakul, a kapcsolatkövető rendszer lehetőséget ad a regisztráló modulnak a további speciális feldolgozásra, még mielőtt a kapcsolat első csomagjának feldolgozása tovább folytatódna. Ez a funkcionalitás szükséges például ahhoz, hogy egy FTP kapcsolat esetén felismerhessük az egy sessionhoz tartozó adatkapcsolatokat, és azokon NAT használata esetén a megfelelő címfordítást is végrehajtsuk.

A kapcsolatkövető rendszer nagy vonalakban a következőképpen működik:

1. minden egyes bejövő csomag esetében eldönti, hogy a csomag valamely már ismert kapcsolathoz tartozik-e;
2. amennyiben igen, frissíti a kapcsolat állapotát;
3. amennyiben ez a csomag egy eddig ismeretlen kapcsolathoz tartozik, úgy létrehoz egy új bejegyzést az állapottáblában;
4. új kapcsolat esetén megvizsgálja, hogy az megfeleltethető-e valamelyik várt kapcsolatnak; amennyiben igen, a megfelelő *helper* függvény le is fut.

³Például UDP-nél az azonos végpontok, azaz IP cím – port párok közötti csomagokat tekintjük egy kapcsolathoz tartozónak; hasonlóan más kapcsolatmentes protokollokra is értelmezhető a kapcsolat fogalma.

A csomaghoz a kapcsolatkövető rendszer hozzárendeli a kapcsolat állapotleíróját, valamint a csomagnak a kapcsolathoz fűződő viszonyát:

NEW – a csomag egy eddig ismeretlen, új kapcsolatot hoz létre;

ESTABLISHED – egy már kiépült kapcsolathoz tartozik;

RELATED – a csomag valamilyen módon kapcsolatban van egy kapcsolattal, de nem közvetlenül része annak⁴;

INVALID – a kapcsolatkövető rendszernek nem sikerült eldöntenie, hogy milyen kapcsolathoz tartozik a csomag (például broadcast és multicast csomagok, vagy TCP window tracking használata esetén nem megfelelő szekvenciaszámmal rendelkező csomag).

A tűzfalunk szabályrendszerében ezeket az információkat sokféleképpen felhasználhatjuk, gondoljunk csak a `state` vagy a `connbytes` iptables matchekre, vagy a `CONNMARK` targetre.

A kapcsolatkövető rendszer működéséről részletesebb információ található az [2] és [3] cikkekben.

3. Állapottartó failover megoldások

Állapottartó csomagszűrőkkel épített failover megoldás esetén tehát valahogyan el kell érniük, hogy az egyes node-ok állapotátlábját a lehető legnagyobb mértékben meg egyezzen. Eerre több lehetséges megoldás is kínálkozik, amelyek mind felépítésükben és működési módszereikben, mind pedig a nyújtott „szolgáltatásban” eltérnek.

3.1. „Szegény ember HA fürtje”

Az azonos állapotátlábla eléréséhez kézenfekvő módszernek tűnik, ha a node-okat párhuzamosan működtetjük, valamint biztosítjuk azt, hogy a fürt minden tagja ugyanazt az inputot – esetünkben a hálózati csomagokat – kapja. Determinisztikus működés esetén amennyiben mindegyik csomagszűrő routerünk ugyanazon szabályrendszer alapján hozza a döntéseket, akkor minden egyes csomag után ugyanolyan állapotátláblával fogunk rendelkezni. Ezen megközelítés esetében a megoldás kulcsa egyértelműen annak biztosítása, hogy a fürt minden tagja pontosan ugyanazt az inputot kapja, és azt determinisztikusan és maradéktalanul dolgozza fel.

Ezeket a feltételeket azonban igen nehéz biztosítani. Egyrészt azt, hogy mindegyik node pontosan ugyanazokat a csomagokat kapja meg és dolgozza fel, csak valamilyen relatíve bonyolult hálózati architektúrával tudnánk megoldani. Ezen felül ráadásul egy Netfilter alapú csomagszűrő nem minden tekintetben viselkedik determinisztikusan.⁵ Egyelőre ezektől a nehézségektől tekintsünk el, és tegyük fel, hogy valamiképpen meg tudunk birkózni mind a nem-determinisztikus működés problémájával, mind pedig a

⁴Például: az FTP protokoll esetében az adatkapcsolat első csomagja **RELATED** viszonyban lesz a parancscsatornával.

⁵Bizonyos esetekben – például túlterhelés esetén – a kapcsolat-állapotátláblából a kapcsolatkövető rendszer aktív bejegyzéseket is töröl. Azt, hogy pontosan melyik bejegyzések kerülnek törlésre, a bejegyzéseket tároló hash táblán belüli sorrend dönti el, ami viszont a hash függvény inicializálásához használt véletlenszám függvénye. Amennyiben használunk címfordítást (NAT, *Network Address Translation*), az a TCP/UDP portszámok kiválasztásának módja miatt szintén nem determinisztikus működés forrása lehet.

hálózati környezetre vonatkozó igen szigorú követelményekkel. Vegyük sorra, hogy egy ilyen megoldásnak milyen előnyei és hátrányai lennének a „szolgáltatások” terén.

Előny, hogy a csomagszűrő szoftverünkön nem kell nagyobb módosításokat végezni, az szinte módosítás nélkül használható. A módszer hátrányai között több dologra is ki kell térnünk. Egyrészt nincs lehetőség egy node kiesése majd újraindítása után a konzisztens állapottábla helyreállítására. Ez pedig igen nagy probléma, hiszen így a fürt tagjai „egyszer használatosak”, kiesés után nem lehet őket visszahelyezni a fürtbe, ami a hosszabb távú üzemeltetést lehetetlenné teszi. Másrésztől kétségek merülhetnek fel a módszer skálázhatóságával is. A fürt tagjai számának növekedésével várhatóan egyre több probléma jelentkezik, hiszen egyre nagyobb a valószínűsége, hogy az egyes node-ok belső állapota valamilyen hiba (például egy csomag elvesztése) miatt különbözni fog. Ráadásul ezen konzisztencia-hibákat sem detektálni, sem kijavítani nem tudjuk a hiányzó újraszinkronizációs képességek miatt.

3.2. Aktív állapotreplikáció

Egy másik lehetséges megközelítés az, amikor teljes egészében az aktuális master fogadja és dolgozza fel a csomagokat. A csomagok által okozott állapotváltozásokról a master a fürt többi tagjának a dedikált *replikációs hálózaton* üzeneteket küld, amelyek ezeket fogadva és feldolgozva frissítik a saját állapotinformációikat.

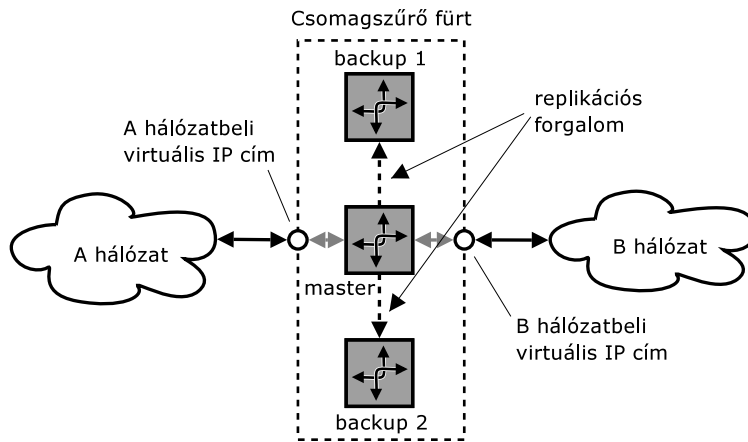
A 3. ábra egy ilyen rendszerre mutat példát. A csomagszűrő fürt virtuális címeire érkező forgalmat a három node közül az aktuális master dolgozza fel és továbbítja, a fürt másik két tagja kizárólag a replikációs üzeneteket fogadja. Az elrendezésnek több előnye is van. Egyrészt a hálózati környezettel szemben nem támaszt komoly követelményeket, pusztán egy dedikált replikációs hálózat szükséges hozzá (például egy külön Ethernet hálózat, amit ráadásul a fürtön belül egyéb célokra is felhasználhatunk). Egy aktív állapotreplikációt használó megoldás felépítése ettől eltekintve tökéletesen megegyezik egy VRRP-alapú fürt esetén szükséges architektúrával. További előny, hogy mivel a node-ok között van kommunikáció, lehetőség nyílik annak detektálására, ha valamilyen node állapotinformációi nem teljesek. Ilyenkor akár a teljes állapottáblát szinkronizálni tudjuk, ami lehetővé teszi akár azt is, hogy a fürtbe menet közben egy új node-ot vegyünk fel. A replikációs üzenetek közvetítésére használt protokoll megfelelő választásával akár az is lehetséges, hogy akár a replikációs hálózaton, akár a backup node-okon belül fellépő, üzenetvesztést okozó hibákat észleljük és javítsuk.

Az előző – a szükséges szoftver-támogatást tekintve meglehetősen minimalisztikus – elrendezéshez képest itt viszont már komoly kiegészítéseket és módosításokat kell eszközölnünk a Netfilter/iptables csomagszűrőnkön. Ez a megoldás azonban olyan előnyökkel kecsegtet – vagy más nézőpontból: a másik lehetőségnek vannak olyan komoly hátrányai – ami miatt ezt a megközelítést választottuk, és a `ct_sync` megvalósításába kezdtünk.

4. A `ct_sync` felépítése és működése

Mielőtt a `ct_sync` használatának tárgyalására térnénk, röviden tekintsük át, hogy hogyan is épül fel ez a megoldás. A 4. ábrán láthatjuk a megoldás belső felépítését, amely három jól elkülönülő részre osztható:

- a Netfilter közvetlen módosítását igénylő, állapotváltozáskor eseményeket generáló, illetve a kapcsolat-állapottábla módosítását végző részekre;



3. ábra. Aktív állapotreplikáció

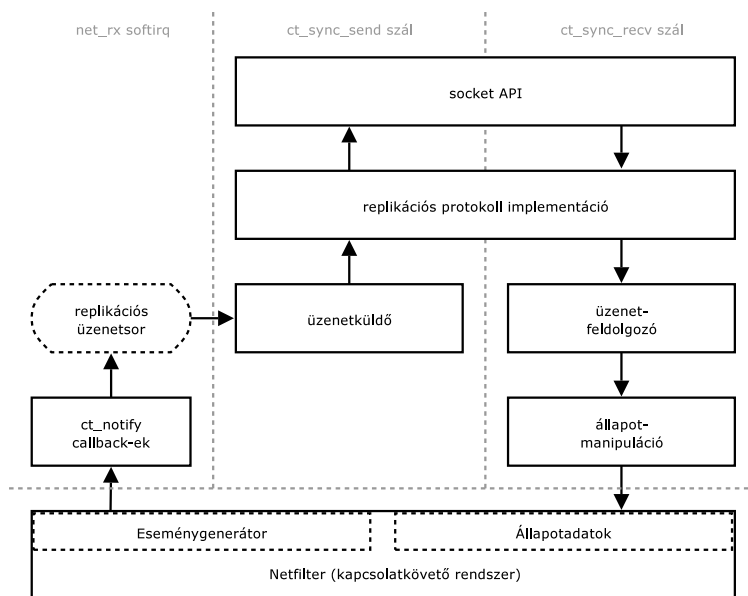
- az egyes események bekövetkeztekor lefutó, a Netfilter belső dinamikus struktúrái alapján egy linearizált üzenetet előállító, illetve azt feldolgozó modulra;
- a node-ok közti kommunikációt biztosító replikációs protokoll implementációjára.

Az első egységtől eltekintve – amelyhez a kapcsolatkövető rendszert is módosítani kell – a `ct_sync` egy betölthető kernel modulként került megvalósításra. A connection tracking rendszert érintő változtatások ráadásul nem kizárólag a `ct_sync` számára szükségesek, hanem például a kapcsolatkövető rendszer állapottáblájához való user-space hozzáférést biztosító `ctnetlink` modul is ezeket használja.

Röviden tekintsük át, hogy hogyan is történik az állapotadatok replikációja. Tegyük fel, hogy a master node éppen feldolgozott egy csomagot, amely továbbításra kerül, és a kapcsolat állapotadataiban valamiféle változást okozott. Ekkor a csomag kiküldése előtt közvetlenül a kapcsolatkövető rendszer meghívja a `ct_sync` által regisztrált callback függvényt, átadva annak a megváltozott adatstruktúrát, valamint az esemény jellegére vonatkozó információkat. Ez a függvény a Linux hálózati kódjának szintjén, megszakítás-kontextusban fut (az ún. `net_rx` softirq-ban), így nem végezhet bonyolultabb műveleteket, egyszerűen a kapcsolat állapotának linearizált⁶ reprezentációját bemásolja a küldő szál által használt kimeneti pufferbe, valamint értesíti a `ct_sync_send` kernel szálát, hogy új elem került a pufferbe. A callback függvény visszatérése után folytatódik az állapotváltozást előidéző csomag feldolgozása. Valamennyi idő eltelte után a kernel ütemezője az üzenetek elküldését végző `ct_sync_send` szálra adja a vezérlést.

Minden egyes állapotváltozást jelző üzenet valamennyi olyan információt tartalmaz, ami a kapcsolat maradéktalan replikálásához szükséges; a `ct_sync` pillanatnyilag nem tud olyan üzenetet küldeni, amelyik a kapcsolat-állapot struktúrájának csak a megváltozott részeit tartalmazná. Egy ilyen update-üzenet körülbelül 250 bájt méretű,

⁶Linearizáció alatt a dinamikusan allokalált struktúrák közötti, mutatókkal reprezentált kapcsolatok olyan reprezentációra alakítását értjük, ami a fürt többi tagján is értelmezhető. Ennek során a különféle struktúrákra mutató pointereket olyan egyedi azonosítókkal kell helyettesíteni, ami nem kötődik a master node-beli lokális információkhoz: például olyan egyedi azonosítókkal, amelyek az egész fürt szintjén értelmezhetőek. Egy kapcsolat esetén ilyen például a két végpont címe.



4. ábra. A ct_sync felépítése

ami viszont ahhoz túl kicsi, hogy minden egyes üzenetet külön csomagban küldjünk el a replikációs hálózaton⁷. Ezért a ct_sync_send szál csak akkor küld el egy csomagot, ha már biztos, hogy nem kerül bele több üzenet (mert legalább egy üzenet már más csomagba fért csak bele, és az események sorrendjén nem szabad változtatni), vagy a csomag ugyan még nincs tele, de az első belekerült üzenet már egy előre definiált időtartamnál régebb óta vár elküldésre. A replikációs protokoll rétege végzi a csomag-fejlécek összeállítását és a csomagok a kernel belső socket interfészén keresztül elküldését.

A slave (backup) node-okon hasonlóan a socket interfészen keresztül fogadja a protokoll-réteg. Itt megtörténik a csomag fejlécének feldolgozása, illetve amennyiben hibát detektálunk, a hiányzó csomagok újraküldésének kérése. Ha minden rendben volt a csomaggal, akkor a benne található üzeneteket feldolgozzuk, az állapotbejegyzések manipulálásához a kapcsolatkövető rendszer módosítása által nyújtott interfészt használva.

4.1. A replikációs protokoll

A replikációs üzenetek célba juttatásához egy UDP multicast alapú, speciálisan erre a célra tervezett protokollt használunk. Az alapvető elvárások a következők:

- hibadetektálási és -javítási képesség,
- kis overhead, különösen a csomagok számát tekintve,
- gyors, egyszerű működés.

⁷Egy hálózati csomag elküldésének költsége ugyanis közel sem lineáris a csomag hosszához képest, egy kisebb csomag elküldése majdnem ugyanannyi időt és erőforrást igényel, mint egy, a hálózat által megengedett maximális méretűé.

Ugyan már többféle megbízható multicast protokoll is rendelkezésre áll (pl. NORM [4]), ezeket elsősorban multimédia streaminghez találták ki, és a mi igen speciális igényeinknek nem felelnek meg bonyolultságuk miatt. Ezért egy NACK (*Negative ACKnowledgement*) alapú UDP multicast feletti saját protokoll került implementálásra. Ennek főbb tulajdonságai a következők:

- Minden csomag rendelkezik egy egyedi sorszámmal, amelyet a küldő node folytonosan növel.
- Amennyiben egy node olyan csomagot fogad, amelynek sorszáma nem pontosan eggyel nagyobb az utolsó sikeresen fogadott csomagénál, akkor egy vagy több csomagot elveszett, és ezek újraküldését kéri egy *negative acknowledgement* üzenetben.
- Minden node rendelkezik az általa utoljára küldött n csomag másolatával; ha újraküldési kérelem érkezik, azokat újra el tudja küldeni.
- Amennyiben már nincs meg a NACK üzenetben igényelt csomagok valamelyike, lehetőség van a teljes újraszinkronizációra.
- Minden egyes újonnan bekapcsolódott node teljes újraszinkronizációt kezdeményez.

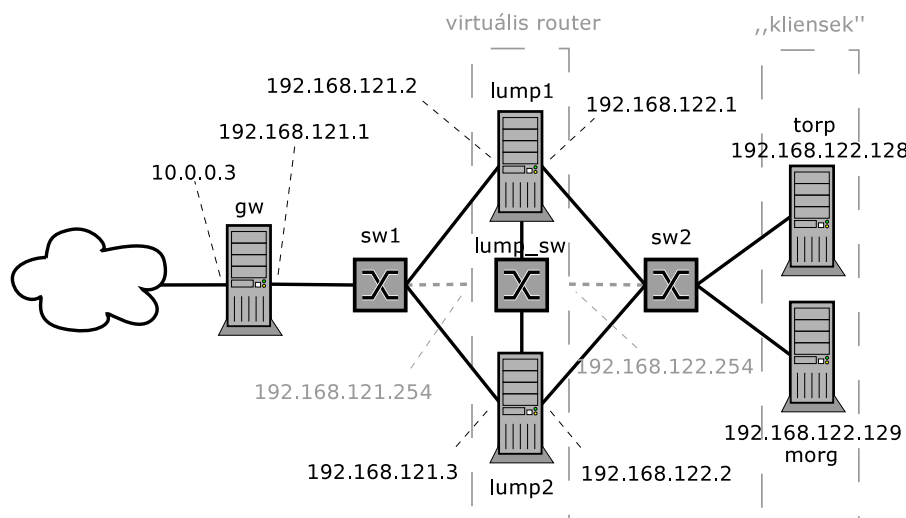
A protokoll tehát csak abban az esetben igényel extra csomagokat, ha tényleges csomagvesztés történt, és a hiányzó üzeneteket újra el kell küldeni. Mivel feltételezésünk szerint a protokollt csak és kizárólag lokális (például Ethernet) hálózaton (LAN) fogják használni, nyugodtan számolhatunk igen alacsony hibaarányral, és így igen alacsony csomagvesztés-aránnyal⁸. A protokoll lehetővé teszi azt is, hogy több üzenetet egy csomagban küldjünk el, így tovább csökkentve a replikációs forgalom csomagjainak számát.

4.2. Egyéb szolgáltatások

A `ct_sync` néhány olyan szolgáltatást is nyújt, ami nem kapcsolódik szorosan az állapotreplikációhoz, viszont igen hasznos lehet tűzfalfűrtünk üzemeltetésénél. Az egyik ilyen a „beépített” NOTRACK funkcionális, azaz a dedikált interfészen át küldött vagy fogadott csomagokat nem dolgozza fel a kapcsolatkövető rendszer. Ezt megtehetnénk az `iptables raw` táblája és a NOTRACK target segítségével is, azonban ez a funkció a replikáció működésének szempontjából annyira fontos, hogy amennyiben elfelejtenénk a megfelelő szabályokkal ezt megtenni, annak igen súlyos következményei lennének. Arról van szó ugyanis, hogy mivel a mi üzeneteink is UDP csomagok, azokat a kapcsolatkövető rendszer feldolgozza, és ez állapotváltozáshoz vezethet. Ez az állapotváltozás azonban újabb csomagok elküldését eredményezi, ami viszont újabb állapotváltozást okoz, és így tovább. Ahhoz, hogy ezt a végtelen ciklust elkerüljük, legalább a `ct_sync` által küldött csomagoknak el kell kerülniük a connection tracking rendszert, amit ez a beépített mechanizmus biztosít.

A másik igen hasznos opcionális szolgáltatás az *l2drop*, azaz *layer 2 drop*. Amennyiben a `ct_sync` betöltésekor engedélyezzük ezt a funkcionális, a slave állapotban levő node-okon a dedikált interfészünket kivéve valamennyi hálózati adapteren bejövő vagy kimenő csomag közvetlenül a második rétegbeli feldolgozás után/előtt (azaz a

⁸Az UDP ellenőrzőösszeg miatt akár egy bithiba miatt is eldobjuk az egész csomagot.



5. ábra. Példa hálózati topológia

csomag a hálózati interfész meghajtójától való fogadása után, vagy a kártya várakozási sorába helyezés előtt) eldobásra kerül. Amennyiben ugyanis a fürt tagjait nem kizárólag csomagszűrésre használjuk, hanem valamilyen egyéb szolgáltatás is fut rajtuk, átálláskor sok időt vehet igénybe, amíg ezek elindulnak. Mivel a szolgáltatásokat biztosító daemonjaink rendszerint a fürt valamely virtuális IP címén várják a kapcsolatokat, azokat nem tudjuk elindítani addig, amíg az adott IP cím nincs valamelyik interfészünkre konfigurálva. Ezt viszont nem tehetjük meg, hiszen ha egynél több node ugyanazzal az IP címmel rendelkezik, az komoly hálózati problémákhoz vezet (gondoljunk például az ARP kérésekre adott válaszokra). Amennyiben azonban a `ct_sync` betöltések engedélyezzük az `l2drop`-ot, nyugodtan bekonfigurálhatjuk hálózati interfészeinket a virtuális IP címekkel, hiszen azokon semmiféle forgalom nem fog keresztüljutni – így például az ARP csomagok sem. Ezáltal átálláskor az új masteren a daemonokat nem kell elindítani, a felkonfigurált virtuális IP címeknek köszönhetően azok akár slave állapotban is is futhatnak, ez pedig az átállási idő rövidítése szempontjából igen fontos lehet.

5. Hogyan használjuk?

A `ct_sync` felépítésének és működésének bemutatása után most nézzünk egy gyakorlati példát is az alkalmazására. Tegyük fel, hogy az 5. ábrán látható rendszert szeretnénk felépíteni: egy router (`gw`), és a klienseink (jelen esetben kettő, `torp` és `morg`) közé szeretnénk tenni egy két tagból álló csomagszűrő fürtöt (`lump1` és `lump2`). A routert és a tűzfalakat, illetve a tűzfalakat és a klienseket egy-egy switch kapcsolja össze (`sw1` és `sw2`), illetve egy dedikált switch biztosítja a tűzfalfürt tagjai közti kapcsolatot a replikációs protokollnak (`lump_sw`).

5.1. VRRP konfiguráció

Első lépésként a fürt számára virtuális IP címeket nyújtó, valamint a master választását is megoldó VRRP daemont, esetünkben a *keepalived*-t [5] kell beállítanunk. A fürtnek két virtuális IP címe van, egy a kliensek, egy pedig a router felé, a két virtuális cím természetesen csak egyszerre migrálható. A node-ok egyenlő prioritással rendelkeznek, és tiltva van, hogy egy újonnan beléptetett node belépésekor rögtön mesterré váljon.

Mivel a *ct_sync* az aktuális master kiválasztását a cluster manager szoftverünkre bízza, a *keepalived*-nek valamilyen módon értesíteni kell a kernel modulunkat amennyiben a node fürtön belüli állapota megváltozik. Ehhez a *ct_sync* egy *procfs* interfészt nyújt, egyszerűen a `/proc/sys/net/ipv4/netfilter/ct_sync/state` fájlba kell 1-et írunk, ha a node master, és 0-t ha slave. A *keepalived* szerencsére lehetőséget ad arra, hogy állapotváltozáskor tetszőleges programot lefuttassunk, így ez igen egyszerűen megoldható.

A *keepalived* konfigurációs fájlját itt nem részletezzük, az megtalálható a *ct_sync* CVS repositoryjában [6].

5.2. A *ct_sync* konfigurációja

A *ct_sync* által nyújtott beállítási lehetőségek jelenleg meglehetősen szegényesek, az állapotátmeneteket vezérlő *procfs* interfészen kívül csak a kernelmodul betöltéskori paramétereire korlátozódnak. Néhány paramétert kötelező megadni, azok nélkül a modul nem töltődik be, míg más paramétereknél ilyenkor az alapértelmezett érték érvényesül. A kötelező paraméterek:

syncdev – a dedikált replikációs hálózati interfész neve;

id – a node egyedi azonosítója (0-255);

state – a node kezdeti állapota (0: slave, 1: master).

Opcionális paraméter jelenleg mindössze egy van:

l2drop – az *l2drop* funkcionalitást tudjuk engedélyezni vagy tiltani (0: tiltva, 1: engedélyezve, alapértelmezett értéke 0).

A modult tehát például a következőképpen tölthetjük be:

```
# modprobe ct_sync syncdev=eth2 id=1 state=0 l2drop=1
```

Ekkor az *eth2* interfészt használja a replikációs protokoll üzeneteinek elküldésére és fogadására, a node egyedi azonosítója 1 lesz, a betöltés után közvetlenül slave állapotban lesz, valamint az *l2drop* engedélyezve van. A példánknál maradva: a tűzfal node-okon mindössze az egyedi azonosítót kell másnak választanunk, ettől eltekintve mindkét node a fenti paraméterekkel tölti be a *ct_sync* modult. Ezt követően a *keepalived* futtatásakor a masternek választott node-on lefutó script állítja master állapotba a *ct_sync* modult.

6. További lehetőségek

Mint az talán már az eddigiekből is kiderült, a *ct_sync* jelenleg is aktív fejlesztés alatt van. Sajnos jelenleg a replikációs lehetőségek nem terjednek ki a kapcsolatkövető rendszer egészére, hiányzik például az *expectation*-ök replikációja. Ez jelenleg azt

okozza, hogy az alkalmazás szintű elemző modulok még csak korlátozásokkal használhatóak a csomagszűrő fűrtünkben.

Szintén problémás pont a kernelmodul nehézkes, csak betöltéskori konfigurációja, valamint az, hogy jelenleg egy node nem lehet több `ct_sync` csoport tagja, azaz például a VRRP protokollnál bemutatott hálózati elrendezés jelenleg nem valósítható meg (1. ábra).

A fejlesztés azonban ha lassan is, de halad, az aktuális változat elérhető a Netfilter CVS-ben [6] (beleértve a szükséges patcheket, a kernelmodul forrását, telepítési és használati útmutatót, valamint néhány példa *keepalived* konfigurációs fájlt).

7. Köszönetnyilvánítás

Itt szeretnék köszönetet mondani azoknak az embereknek, akik valamilyen formában hozzájárultak a projekthez. Kétségtelenül a legfontosabb ebből a szempontból Harald Welte, aki először vetette fel ennek a megoldásnak a lehetőségét [2], valamint az első proof-of-concept implementáció közzététele után valóban használható formára hozta a `ct_sync`-et. Hálás vagyok, hogy olyan emberek segítettek, mint Kadlecsik József és Kis-Szabó András, az ő ötleteikkel felvértezve fogtam neki a tényleges munkának. Köszönet illeti továbbá a BME Méréstechnika és Információs Rendszerek Tanszékét és a BalaBit IT Kft.-t, ők tették lehetővé hogy munkám keretében a `ct_sync`-en dolgozhassak.

Hivatkozások

- [1] R. Hinden, Ed., „Virtual Router Redundancy Protocol”, IETF RFC 3768
<http://www.ietf.org/rfc/rfc3768.txt>
- [2] Harald Welte, „How to replicate the fire – HA for Netfilter based firewalls”, *Proceedings of the Ottawa Linux Symposium*, 2002, pp 565-572
http://www.linux.org.uk/~ajh/ols2002_proceedings.pdf.gz
- [3] Harald Welte, „ct_sync: state replication of ip_conntrack”, *Proceedings of the Ottawa Linux Symposium*, 2004, pp 537-545
<http://www.finix.org/Reprints/Reprint-Welte-OLS2004.pdf>
- [4] B. Adamson, C. Bormann, M. Handley, J. Macker, „NACK-Oriented Reliable Multicast Protocol”, Internet Draft, 2004
<http://www.ietf.org/internet-drafts/draft-ietf-rmt-pi-norm-10.txt>
- [5] <http://keepalived.sourceforge.net>
- [6] Netfilter Project CVS repository, netfilter-ha module
<http://cvs.netfilter.org/netfilter-ha/>

A nyílt forráskód társadalmi értéke és a civil szervezetek szerepvállalási formái

László Gábor

<laszlo.gabor@ittk.hu>

Kivonat

Az előadás egyik fő célja a szabad szoftverek előnyeinek bemutatása a társadalmi hasznosság megközelítésének szemszögéből. A többség által elsőként említett költségvonzatokon kívül még számos más szempont is létezik, amely alapján igazolható a szabad szoftver társadalomra és esélyegyenlőségre gyakorolt pozitív hatása.

A civil szervezeteknek hatalmas szerepe van abban, hogy ezek a szoftverek minél jobban elterjedjenek és beépüljenek a mindennapi életvitelbe. Ebbe a szerepvállalásba beletartozik a kormányzatnál végzett lobbitevékenység is, mivel a legnagyobb informatikai fogyasztó az állam, és – például az oktatásban használt szoftverek esetében – beszerzéseivel közvetve vagy közvetlenül erős befolyást gyakorol a szoftverpiacra, illetve a szoftverhasználat szokásaira is.

Tartalomjegyzék

1. Miért a szabad szoftver?	90
2. A „digitális szakadék”	90
3. Politikai és gazdasági kérdések	91
4. Oktatás, innováció	91
5. Magyarországi kitekintés	92
6. Civil szervezetek	93
7. Összefoglalás	94

1. Miért a szabad szoftver?

Már sok esetben bebizonyosodott, és több független kutatás is kimutatta, hogy a szabad szoftverek megfelelő technikai paraméterekkel rendelkeznek, ami által egyenrangú versenytársá válhatnak a hagyományos kereskedelmi szoftverekkel folytatott versenyben [1]. A gyors technikai fejlődés eredményeképpen a mindennapi élet már elképzelhetetlen lenne informatikai rendszerek használata nélkül. A mindennapokat átszövő számítógépes infrastruktúra megkerülhetlenné vált. Azonban ezek a rendszerek egyrészt sebezhetőek is, ami olyan szoftverek használatát kívánja meg, amelyek ellenőrizhetőek a felhasználó biztonságának szempontjából. A szabad szoftverek számos olyan előnnyel rendelkeznek, amelyek kívánatosá teszik ezen szoftverek használatát a társadalmi esélyegyenlőség biztosításának oldaláról is.

2. A „digitális szakadék”

A társadalmi átalakulások folyamata a történelemben megmutatta, hogy a különböző társadalmi rétegek közötti különbségek soha nem tűntek el, mindig más formában, de léteztek. A XX. század második felétől átmenet figyelhető meg az ipari forradalmat követő és az azt meghatározó időszakból a tudásintenzív folyamatokat igénylő és alkalmazó társadalmi berendezkedés irányába. Az információ szerepe egyre jobban felértékelődött nemcsak a gazdasági életben, ahol ez jelentős versenyelőny-képző tényezővé vált, hanem a hétköznapi életben is, ahol szintén egyre fontosabb szerep jut az információnak.

A tudásintenzív folyamatok, és az információ értékének felértékelődésével egyidőben a társadalmi rétegek között újabb törés következett be, mégpedig az információhoz hozzáférők és az információhoz nem hozzáférők csoportjaira szakadt a társadalom. Ez a törés az egyéb szempontból is hátrányosabb helyzetben lévő rétegeket érinti elsődlegesen. A digitális szakadék terminussal illetett társadalmi különbség kialakulásának, illetve mélyülésének megakadályozása az információs társadalom korában új kihívást jelent nemcsak a társadalom, hanem a kormányzatok számára is.

Ernest J. Wilson hozzáférés-modellje 6 pontban foglalta össze az információkhoz való hozzáféréshez szükséges tényezőket:

- Fizikai infrastruktúra,
- Pénzügyi,
- Kognitív tudás,
- Tartalom,
- Intézményi,
- Politikai.

A modellből jól látható, hogy a hozzáférés nem csak a fizikai infrastruktúrát és a költségvetéseket tartalmazza, hanem megjelenik benne a kormányzatok szerepe is. Az információkhoz, adatokhoz való hozzáférést maguknak az adatoknak a rendelkezésre bocsátásán túl intézményi háttérrel is támogatni kell. A különböző – erősen eltérő összetételű és szerkezetű – társadalmi rétegek „informatikai írástudásának” megteremtésében és a hátrányos rétegek társadalomtól való további leszakadásának megakadályozásában

hatalmas szerepe lehet/van a nyílt forráskódú szoftvereknek. Az oktatásban alkalmazott és oktatott szoftverek képezik a végzett hallgatók jövőbeni szoftverfelhasználásainak alapjait is, mivel az emberi természet olyan adottsággal rendelkezik, miszerint a megszokott környezetből nem szívesen szakadnak ki a felhasználók, valami bizonytalan, új kipróbálása érdekében.

A digitális szakadék néven ismert jelenség csökkentésében, illetve áthidalásában – költségvetési szempontból is – elfogadható alternatívát jelentenek a szabad szoftverek. A jogkövető magatartás elérése, a jogtisztá programok használata sok esetben csak szabad szoftverekkel valósítható meg, főként a hátrányos helyzetű rétegek esetében, ahol a költségek fontos szempontot jelentenek.

3. Politikai és gazdasági kérdések

A másik fontos szempont, amely már a kormányzatok és a politikusok felelősségét is felveti az adófizetők pénzének hatékony kezelésén túl, valamint túllépve a szabad szoftver olcsóbb paradigmán is a *bizalom* kérdésköre. Aki kontrollálja a szoftvereket, kontrollálja az intézményt, szervezetet is. Igaz ez a társadalomra is. Bár az államok – természetükből fakadóan – szeretnék ellenőrizni az állampolgáraikat akár a szoftvereken keresztül is, ld. politikai megrendelésre készülő backdoorok, azt már nem szeretnék és nem is engedhetnék meg maguknak, hogy felettük a szoftvergyártók gyakoroljanak hatalmat. A zárt forráskódú szoftverek használatával – azon kívül, hogy felmerül a függőség kérdésköre – ennek a veszélynek teszik ki magukat a felhasználók.

Az elektronikus választási szoftverek körül elsősorban az USA-ban alakult ki éles vita, amelynek mottóját a „*Ha nem szabad a szoftver – nem szabad a választás sem!*” mondatban foglalták össze a szabad szoftverek támogatói. Ez a nézőpont az állampolgárok szemszögéből is egyértelművé teszi, hogy nem a szoftvergyártóknak kell a kormányokon és a kormányoknak az állampolgárokon hatalmat gyakorolni, hanem a demokrácia adta eszközök és az ellenőrzés lehetőségével élve az állampolgároknak kell biztosítani az ellenőrzés lehetőségét az éppen hatalmat gyakorlók felett.

A gazdasági kérdések tárgyalásakor elsődleges szempontként a költségeket tekintik fő faktorként. A a tulajdonlás teljes költsége (TCO), mint mérőszám – a mérési nehézségek ellenére – összehasonlíthatóvá teszi a szabad szoftverek költségeit, szemben a hagyományos zárt forráskódú kereskedelmi szoftverekkel. Ezek a tanulmányok azt mutatták, hogy a szabad szoftverek teljes tulajdonlásra vonatkozó költségei alacsonyabbak voltak. A szoftverek beszerzésén elért megtakarítások forrásként jelentkeznek a másik oldalon. Amennyiben a szoftverfejlesztési igényeket az adott ország határain belül működő vállalkozások végzik, a kormányzat, illetve a pénzügyi kormányzat többszörösen jól jár. A munkahelyteremtés mellett, ezek a vállalkozások, illetve az alkalmazásukban álló munkavállalók által befizetett adók az ország költségvetési bevételeit növelik.

4. Oktatás, innováció

Egy közmondás szerint: „*Ha egy évre tervezel, vess magokat, ha tíz évre, ültess fát, ha száz évre, oktasd az embereket!*”. Az oktatás kiemelt hangsúlyt kap a nemzetgazdaság területén. Egyrészt hatalmas piacként jelenik meg, másrészt pedig az oktatási rendszer teljesítménye nagymértékben meghatározza egy ország jövőbeli teljesítményét. Ez főként azokra az országokra érvényes, amelyek gazdaságilag nem engedhetik meg maguknak a szellemi tőke importját. Magyarország esetében főként a *brain drain*,

az *agyelszívás* jelenthet reális veszélyt. Az oktatás területén a nyílt forráskódnak is van létjogosultsága. Egyfelől azon szoftverek esetében, amelyeken az oktatás történik – természetesen nem kizárva azon kereskedelmi, zárt forráskódú termékek körét sem, amelyek ismeretére erős piaci igény mutatkozik, pl. integrált vállalati rendszerek – másrészt a nyílt forráskód fejlesztési modellje által a végzett programozók sokkal jobban megértik egy-egy program működését, ami által ők is sokkal jobb szakemberré válnak.

Sajnálatos módon az informatika és a számítástechnika oktatása nem magához az informatikai gondolkodás megteremtéséhez, és a számítógép, számítástechnikai eszközök készség szintű használatának elsajátításához kapcsolódik, hanem elsődlegesen programcsomagok megtanítását jelenti.

Az innovációra, kutatás–fejlesztésre gyakorolt hatás sokrétű a nyílt forráskódú programok esetében. Elsődlegesen a nyílt forráskód modelljének alkalmazása kulcsfontosságú. A tudástermelés és -megosztás területén sokkal hatékonyabb és gyorsabb eredmények érhetők el azáltal, hogy nem kell újra feltalálni a spanyolviaszt egyetlen kutatónak sem, csak amiatt, mert a létrejött eredmények elzártan léteznek csak, és nem elérhetőek. Ezen modell alapján végzett fejlesztések hasznosulásának esélye nagyobb, mint a hagyományos értelemben vett K+F.

5. Magyarországi kitekintés

A Magyar Információs Társadalom Stratégiájában (MITS) [2] is szerepelnek a nyílt forráskódú szoftverek. A stratégiához kapcsolódó „Közzadat” programfüzetben [3] pedig a programok között meghatározott feladatok is szerepelnek. Ezidáig három központi kiemelt program nem indult el, az „Infrastrukturális Szolgáltatások” főirányba tartozó két program, közte a „Közcélú, közhasznú információk infrastruktúrája” program sem, amely tartalmazta a nyílt forráskódhoz kapcsolódó általános programokat is.

Miközben egyfelől az figyelhető meg, hogy a kormányzat és képviselői nem utasítják el a szabad szoftveres megoldásokat, – legalábbis elvi szinten – azonban ezen megoldások politikai támogatásról nincs szó. Hatékonyabb érdekképviselőre és összefogásra lenne szükség a szabad szoftverek témakörében érintett magyarországi civil szervezetek részéről, hogy realitássá válhasson a szabad szoftverek esélyegyenlőségének biztosítása a hazai közbeszerzés során is.

A korábban már említett jogkövető magatartás biztosítása a jelenlegi körülmények között nem biztosítható a hagyományos kereskedelmi szoftvercsomagok alkalmazásával. Ennek okai között nem csak az anyagi oldal, hanem a meglévő szemlélet játszik fontos szerepet.

A kormányzat több megállapodást is kötött a Microsofttal, (CAMPUS, Tisztaszoftver Program) ami alapján az oktatási intézmények ingyenesen használhatják a Microsoft legújabb termékeit. „Ingyen ebéd” azonban nem létezik. Ez a végfelhasználók felé mutatott ingyenesség, a megállapodások keretében a kormányzat súlyos összegeket fizetett a szoftverpiac egyik jelentős súllyal bíró szereplőjének. Kérdés, hogy a végfelhasználók akik ezeket a programcsomagokat szották meg, a megállapodások lejárta után megvásárolják a jogokat, hogy továbbra is jogtiszta használhassák, vagy a programok további használata illegális lesz és bármikor csengethet az ajtón a „szoftverrendőrség”, mivel a legális használat feltétele most is a licencszerződés kitöltése, amelyen szerepelnek a felhasználó személyes adatai is. Ez utóbbi esetben csak jól sikerült reklámfogásról és a piaci működést erőteljesen befolyásoló, az adófizetőknek sokba kerülő marketingakcióról van csak szó.

Mindezen veszélyek kivédésére másik lehetőség a szabad szoftverek használatára történő áttérés.

6. Civil szervezetek

Az a társadalom működik jól, ahol a négy fő aktor – kormányzat, tudomány, ipar és a civil szféra – együttműködése minden irányból kielégítő és kölcsönös, azaz ezek a elemek folyamatos interakcióban állnak egymással. Amennyiben bármely két terület között kommunikációs vagy egyéb problémák keletkeznek, a rendszer kibillen egyensúlyi állapotából és a működésben anomáliák keletkeznek. Az ábra a négy fő terület optimális és kívánatos együttműködését szemlélteti. (A modell kidolgozója, a korábban már említett E. J. Wilson.)



Gyémánt modell

A civil szférának jelentős szerepe van a szabad szoftverek elterjedésének szempontjából. A világ különböző országaiban eltérő erősségű és más-más hagyományokkal rendelkezik a civil szféra. Ebből következően a civil szervezetek szerepköre és lehetőségei országonként változóak.

A civil szerveződések és csoportosulások kulturális hagyománya Kanadában nagyon erős. A GOSLING (Getting Open Source Logic INTO Governments) nevezetű civil szerveződés a nyílt forráskód gondolkodásmódjának bemutatására és átadására törekszik a kormányzati munkában. Két fő csoportjuk van jelenleg, a székhelyük Ottawa, – az ország és a kormányzati munka fővárosa – illetve Toronto – az ország legnagyobb városa. Lobbitevékenységük kiterjed a kapcsolattartásra a kormányzat illetékes képviselőivel, illetve tagjai rendszeres találkozóin túlmutatóan szinte minden fontosabb rendezvényen képviseltetik magukat előadásokkal.

A kanadai példánál a modell működése jól igazolható, miszerint a kormányzat is jelentős erőforrásokat fektet abba, hogy a társadalom jól működjön és képviselőivel és anyagi támogatásával közvetlenül részt vállal a civil szervezetek működtetésében.

A magyarországi helyzet a 4 fő aktor együttműködésének nem tökéletes összhangját mutatja jelenleg. A civil szervezetek már kezdenek megerősödni, azonban működésük és társadalomban betöltött szerepük még elmarad a kívánatos mértéktől. Az összefogás hiánya, sőt sok esetben széthúzás figyelhető meg.

A világ különböző országaiban jó példákat találhatunk, amelyek adaptációjával a hazai érdekérvényesítő képesség is növelhető. Azonban a modell lényegéből fakadóan, ehhez partnernek kell lennie a kormánynak is.

7. Összefoglalás

A szabad szoftverek társadalmi értéke sokkal több területen jól érzékelhető, mint azt eddig kimutatták. Lehetőséget biztosít a hátrányos helyzetű rétegek informatikai le-
szakadásának megakadályozásához, valamint biztosíthatja az esélyegyenlőség megte-
remtését az információkhoz történő hozzáférés területén. Napjaink aktuális kérdései,
milyen módon járulhatnak hozzá a civil szervezetek Magyarországon a MITS-ben és
a hozzá kapcsolódó programfüzetben megfogalmazott célok megvalósításához? Vajon
mit lehetne tenni a hatékonyabb szabad szoftveres érdekképviselésért? Az összefogás,
vagy a széthúzás győz, teret engedve ezáltal a zárt forráskódú szoftvergyártóknak és a
lobbiknak, elszigetelt partizánakciókká degradálva az eddigi kezdeményezéseket?

Hivatkozások

- [1] Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!
http://www.dwheeler.com/oss_fs_why.html
- [2] Magyar Információs Társadalom Stratégia
<http://www.ihm.hu/strategia>
- [3] Magyar Információs Társadalom Stratégia programfüzetei
<http://www.itktb.hu/>
- [4] Saving Cash: A Comparison of Open Source and Proprietary Software
<http://www.researchandmarkets.com/reports/c4117/>
- [5] TCO and ROI
[http://www.osia.net.au/open_source_resources/tco_and_](http://www.osia.net.au/open_source_resources/tco_and_roi)
[roi](http://www.osia.net.au/open_source_resources/tco_and_roi)

UNIX egy kicsit másképp = BSD

Légrádi Gábor

Kivonat

A BSD operációs rendszerek jelentősége napjainkban egyre inkább növekszik, ezért érdemes velük részletesebben is megismerkedni.

Ezen leírás a főbb vonulatokat (FreeBSD, NetBSD, OpenBSD) szeretné bemutatni, de említés szintjén felmutatásra kerül néhány, ezeken az operációs rendszeren alapuló projekt is.

Tartalomjegyzék

1. Előzmények, a BSD kialakulása	96
2. Főbb rendszerváltozatok	96
3. Tulajdonságok	97
4. Néhány BSD alapú fejlesztés	99
5. Összefoglalás	99

1. Előzmények, a BSD kialakulása

A hetvenes években a UNIX rendszert elkészítő amerikai AT&T (American Telephone and Telegraph) cég, mivel irányultságát tekintve alapvetően egy kommunikációval foglalkozó társaság volt, ezért nem rendelkezett semmiféle olyan joggal, amely megengedte volna szoftverek, vagy akár operációs rendszerek fejlesztését és értékesítését.

Ezen oknál fogva a cég, az általa elkészített UNIX rendszert – és annak forráskódját is – ingyenesen odaadta egyetemeknek és kutatóintézeteknek.

A forráskód átadással együtt járt az a jog is, hogy a kódot ki lehetett egészíteni, sőt akár tovább is lehetett fejleszteni.

A fejlesztők sok értékes új ötlettel bővítették ki a UNIX-ot, mely ötletekben rejlő lehetőségek nagyban hozzájárultak a UNIX elterjedéséhez, s a kiegészítések és javítások belekerültek a későbbi UNIX változatokba is.

A legismertebb ilyen UNIX kiegészítéseket a Berkeley egyetemen készítették, megalkotva ezzel a UNIX egyik irányát, amely a BSD (Berkeley Software Distribution) nevet kapta [1].

A BSD rendszer több kommersziális UNIX változat kialakulásában is jelentős szerepet játszott, mint például az AIX (IBM), a HP-UX (HP), a Solaris, vagy SunOS (Sun), az IRIX (SGI), az OSF/1 (Digital), illetve a Tru64 UNIX (Digital, Compaq, HP).

A nyolcvanas években az eredeti AT&T és a BSD rendszerek tovább fejlődtek, de nem alakult ki olyan UNIX változat, amely egységes keretben tartalmazta volna e két eltérő dialektus tulajdonságait.

A BSD rendszer változatait sorszámmal látták el, az itt tárgyalt rendszerek megalkotásában a főszerep a 4.4BSD változaté volt.

A 90-es évek legelején fejlesztők igen szűk csoportja 4.4BSD alapokból kiindulva létrehozta a 386BSD nevű változatot, amely a PC-s felhasználókat célozta meg elsődlegesen, segítve ezzel is a BSD UNIX terjedését. Ez a változat nem lett sikeres, viszont megmutatta, hogy lehet BSD operációs rendszert létrehozni i386-os architektúrára is.

Ez a 386BSD nevű változat hatott más BSD fejlesztőkre is, inspirálta őket, s ezáltal a kilencvenes években több különböző 4.4BSD utód is létrejött.

Ezek az utódok voltak a FreeBSD, a NetBSD és az OpenBSD, melyek a szabad szoftveres, illetve a nyílt kódú oldalt, míg a Mac OS X a kommersziális UNIX világot erősítette.

2. Főbb rendszerváltozatok

A BSD az előbbieken leírt módon több, egymástól eltérő változatban jelent tehát meg.

Az eltérés a változatok között elsősorban a fejlesztési célok és a támogatott platformok tekintetében jelentős, de alapvetően mindhárom rendszer tisztán tükrözi vissza a BSD-filozófiát.

FreeBSD

A FreeBSD [2] rendszer elsősorban az Intel architektúrájú i386-os processzorokat tartalmazó számítógépek felhasználóinak BSD operációs rendszerrel való ellátásának céljára jött létre. Jelenleg a korábban említett három főbb vonulattól ez a változat rendelkezik a legtöbb olyan programmal, amelyek más rendszerekből kerültek át (portolás),

ez több mint 6000 különböző programot jelent. Talán ez a rendszer a legbarátságosabb és legkényelmesebb mindhárom közül.

A FreeBSD megvalósítása jelenleg két – egymástól eltérő – változatban létezik.

A 4-es sorozatú változat (jelenleg a 4.10-es) alapvetően egyprocesszoros gépeken fut, míg az újabb, az 5-ös sorozatszámú (jelenleg a 5.2.1-es), már többprocesszoros rendszereken is alkalmazható. Az 5-ös sorozat a 4-es sorozat nagymértékű és lényeges részeket érintő átírásával (újraírásával) született.

NetBSD

Ez a BSD változat [3] fő célkitűzéseit tekintve megpróbál „az” operációs rendszer lenni, amely szinte minden platformon használható.

Ez azt jelenti, hogy jelenleg már több mint 50 különböző hardverplatformra portolták (vitték át). Maga a portolás folyamata is – mivel egy platformfüggetlen felületet definiáltak erre a célra – más operációs rendszerekhez képest sokkal gyorsabban kivitelezhető.

E rendszer fejlődése viszonylag lassú, mert az újonnan létrehozott tulajdonságokat az eddigi platformokra is következetesen át kell vinni.

A UNIX elterjedését, oktatását, tanulását, az oktatási intézményekben történő felhasználását a NetBSD nagymértékben támogatja azáltal, hogy a régebbi, szerényebb erőforrásokkal rendelkező gépekre is sikeresen telepíthető, és azokon is megbízhatóan használható.

A NetBSD jelenlegi változatának száma 1.6.2, de a napokban adják ki a 2.0-ás változatot is.

OpenBSD

Az OpenBSD-t [4] a NetBSD fejlesztői közül kivált programozók készítették el.

Főbb célkitűzései között ott szerepel a biztonságos üzemmenet, és a megbízható (auditált) forráskód alkalmazása, mind az operációs rendszer, mind pedig a kiszolgáló programok készítésekor.

E rendszer megbízhatóságát jól jellemzi az a tény, hogy a rendszer alapértelmezett installálásának használata során az utóbbi hét évben összesen egy távolról kihasználható hibát sikerült felfedezni.

A jelenlegi legújabb változat (3.6-os), amely már többprocesszoros (SMP) támogatással is rendelkezik. Minden következő változat az év adott időszakában kerül kiadásra, május elsején és november elsején.

3. Tulajdonságok

A fenti BSD-rendszerek alapvetően szerverfunkciókat megvalósító platformnak készültek, s emiatt a szervercélú hardvereket támogatják elsősorban. A megvalósításban természetesen a desktop szoftverek is egyre nagyobb szerepet kapnak (pl. OpenOffice.org), de a fő cél még mindig a szerverfunkciók kiszolgálása, valamint a szerver hardverelemeinek egyre jobb szoftverrel történő ellátása.

Az operációs rendszer fejlesztésének menete igencsak konzervatívnak mondható, amely azt is jelenti, hogy egy frissen megjelent hardverelem szoftveres támogatásának operációs rendszerbe illesztése hosszú ideig tart.

Ez az idő jobb esetben fél, átlagos esetben egy évet jelent. Ez idő alatt azonban a támogató szoftver tesztelése és dokumentálása folyik, mivel ezen feltételek teljesítése nélkül nem kerülhet a rendszerbe semmi.

A fejlesztést végzők átlagéletkora 30 év feletti, s rendelkeznek nagygépes ismerettel és tapasztalattal is.

A fejlesztésben résztvevők három, egymástól jól elkülöníthető csoportba sorolhatók. Az első csoport a legbelső kör, a mag („core”), amely néhány tíz főt jelent. Ők azok, akik döntenek az operációs rendszerbe integrálható funkciókról. Felelőségük kollektív, azaz nem egy személy dönt a felmerülő kérdésekben. Ez azért különösen előnyös, mert nem egy személy pillanatnyi hangulatától függ a projekt további menete. A belső kör a második csoport, a „beajánlók” („committers”) nevű csoporttól kapja az elbírálandó kódot, mely már alapos tesztelésre és dokumentálásra került a kódbeajánlók által. A harmadik csoportba, a legkülső körbe tetszőleges programozó bekerülhet azáltal, hogy hozzáférhetővé teszi az operációs rendszerbe szánt kódját. Ennek a kódnak a további szűrését, tesztelését, javításra vagy kiegészítésre történő visszaadását végzik a beajánló csoport tagjai.

Ez a többszintű szűrés lehetővé teszi a hibák előfordulásának minimalizálását, az elkészült kód minőségének nagymértékű növelését.

Eltérő ezen felül más rendszerektől még a BSD rendszer licencelése is. A BSD típusú licencelés megengedi, de nem teszi kötelezővé a forráskód átadását a bináris (végrehajtható) állományokkal együtt.

További tulajdonság még az is, hogy a BSD rendszereket, és az azokon alapuló fejlesztéseket lehet akár pénzért is árusítani (példa erre a CISCO hálózati eszközökön használatos IOS rendszer, amely „alatt” tulajdonképpen FreeBSD működik).

A BSD rendszerek közötti átjárás nehézségei, az eltérések a NetBSD, FreeBSD, illetve OpenBSD kezelésében kisebbek, mint egyes Linux-disztribúciók között, így „át szokni” egyik BSD-ről a másikra sokkal könnyebb, sokkal kevesebb energiát igényel.

A „disztribúciók” viszonylag kis száma (FreeBSD, NetBSD, OpenBSD) azért is előnyös, mert a fejlesztők erőforrásai jobban koncentrálhatók az adott változatok fejlesztésére.

A BSD rendszerek nagyban hasonlítanak a „klasszikus” nagygépes UNIX rendszerekhez, s ez megkönnyíti a BSD rendszer felhasználóinak szükség esetén a nagygépekre való átállását is.

A BSD rendszerek kernelje nem választható el az operációs rendszer többi részétől. Ez a kernel alapvetően monolitikus jellegű, de néhány hardver, illetve szoftverrész támogatására alkalmazható betölthető modulokat is tartalmazhat szükség esetén. Az operációs rendszer verziószáma tulajdonképpen maga a kernel verziószáma is egyben. A három változat kernelje különböző, egymás között nem cserélhetők.

Bár nem kötelező a forrásprogramot a rendszerhez mellékelni, a kernel forráskódja szinte mindig része a kiadásoknak, s a további programok forrásai is megszerezhetők, ezáltal lehetővé válik akár egész rendszer (kernel + az arra épülő szoftverek) teljes mértékű – az adott gép adott processzorához illeszkedő, annak tulajdonságait figyelembe vevő – újrafordítása is, amely az adott hardverplatform jobb kihasználását eredményezi. (Ez utóbbihoz hasonló, a forrásból történő fordítást valósít meg a manapság egyre népszerűbb Gentoo nevű Linux változat is [5].)

Hiányosságként felróható, hogy a magyar nyelv jelenleg csak a FreeBSD rendszer-nél támogatott, de a másik két rendszer ékezetes betűkkel történő ellátása is megoldható.

4. Néhány BSD alapú fejlesztés

Lássunk néhány, az előbbi rendszerek alapjain létrejött projektet:

picoBSD

A FreeBSD egy korábbi kiadásán (3-as sorozat) alapuló, egy flopis változat. Alkalmas telefonos és hálózatos elérés használatára, illetve router készítésére is [6].

TrustedBSD

A projekt célja a FreeBSD rendszert olyan kiegészítésekkel ellátni, amely alapján megfelel az igen szigorú „Common Criteria” feltételrendszerben foglaltaknak. Ez a fejlesztés magába foglalja az auditáló keretrendszer létrehozását, amely képes a jogok, valamint a futásidejű biztonsági események centralizált kezelésére [7].

DragonflyBSD

A 4-es sorozatú FreeBSD fejlesztői közül néhányan új irányba vitték a fejlesztést. A 4-es sorozat alaptulajdonságait és logikáját megtartva, új rendszert hoztak létre, melynek fejlődési iránya eltér a „hivatalos” 5-ös sorozatétól. Az általuk fejlesztett rendszer több helyen új, a korábbiaktól eltérő részeket tartalmaz (megváltozott az I/O infrastruktúra, a szálkezelési modell, a csomagok kezelése), melyek alapján szinte egy új operációs rendszernek is nevezhetnének [8].

FreeSBIE

FreeBSD alapú live-CD, amelynek segítségével installálás nélkül is kipróbálható a rendszer [9].

ekkoBSD

Az OpenBSD 3.3-as változatán alapuló projekt, melynek célja az igen egyszerű adminisztráció és a nagyfokú biztonság egyidejű megvalósítása [10].

NetBSD/Firewall

A NetBSD rendszert alapul vevő megoldás, amely régi gépek felhasználását segíti elő, s elsődlegesen az állandó internet kapcsolatok – kábel, bérlet vonal, ADSL, PPPoE, PPPTP – megvalósítását és támogatását végzi [11].

5. Összefoglalás

A BSD típusú rendszerek eddigi fejlődése, változatossága, megbízhatósága, valamint a fejlődés irányába mutatott nyitottsága méltán emeli ezeket a rendszereket a többi „jobb” operációs rendszer közé.

A BSD technológia kiforrottsága, a kialakulása óta eltelt közel 30 esztendő méltán bizonyította, hogy ez az irányvonal is jól használható, s a megbízható üzemmenetet biztosító szerverek kialakítására mindenképpen alkalmas.

Természetesen ezek a rendszerek sem tökéletesek, a fejlődés azonban itt is állandóan jelen van, ezért mindenképpen érdemes velük megismerkedni, mert sok tekintetben igen hajlékony és hatékony szerverkörnyezet alakítható ki a segítségükkel.

Jó ismerkedést és használatot kívánok!

Hivatkozások

- [1] <http://www.levenez.com/unix/>
 - [2] <http://www.freebsd.org/>
 - [3] <http://www.netbsd.org/>
 - [4] <http://www.openbsd.org/>
 - [5] <http://www.gentoo.org/>
 - [6] <http://people.freebsd.org/~picobsd/picobsd.html>
 - [7] <http://www.trustedbsd.org/>
 - [8] <http://www.dragonflybsd.org/>
 - [9] <http://www.freesbie.org/>
 - [10] <http://www.ekkobsd.org/>
 - [11] <http://www.dubbele.com/>
 - [12] HUP (Hungarian UNIX Portal) <http://www.hup.hu/>
 - [13] Magyar BSD Egyesület (MBSDE) <http://www.bsd.hu/>
-

Hibakeresés és elhárítás Linux rendszeren

Mátó Péter

<mato.peter@andrews.hu>

Kivonat

Sajnos néha előfordul, hogy valamilyen rendellenes működést tapasztalunk egy szerveren vagy munkaállomásunk használata közben. Nem működik egy csatlakoztatott eszköz, nem indul el egy program, vagy éppen elindul, de nem hajlandó azt csinálni, amit elvárunk tőle.

Kevésbé tapasztalt Linux felhasználók ilyen esetben segítséget kérnek, vagy rosszabb esetben feladják, és nem használják az adott eszközt vagy programot. A legrosszabb eset, hogy a csalódott felhasználó át- vagy visszatér más rendszer használatára. Pedig hosszú évek tapasztalata azt mutatja, hogy általában van megoldás.

Az előadáson olyan általánosan használható módszerek kerülnek bemutatásra, amelyek segítségével minden gyakorlottsági szintű felhasználó képes egy-egy kellemetlen probléma felderítésére és az esetek jelentős részében elhárítására.

Tartalomjegyzék

1. Mi végre is ez az egész?	102
2. A hibák típusai	102
2.1. A program felkészült-e?	102
2.2. A hiba megjelenik-e a felületen?	103
2.2.1. A hibaüzenet megjelenési formái	103
2.2.2. A program bőbeszédűségének befolyásolása	104
2.2.3. Ha van ráutaló hibaüzenet vagy esemény	104
3. Ha nincs értelmezhető hibaüzenet	105
3.1. A program függvényhívásainak követése	105
3.2. Hálózati kommunikációs problémák	108
3.2.1. Nyomkövetés és a core fájlok	109
3.3. Még egy hasznos eszköz, az lsof	109
3.4. A hardver hibák elhárítása	109

1. Mi végre is ez az egész?

Mi az ördög?! – gondolja hősünk felháborodottsággal vegyes kétségbeeséssel. Már megint nem akar ez a macska rúgta gép működni. Mi a baja? Tudja a mér rém. Hogy lehetne megtudni, mi a gond? Legalább értelmesen megmondaná. . .

Ez a hős bármelyikünk lehetne. Időnként kifog rajtunk a gép. Mikor kijött az Unreal Tournament Linuxra, akkor még volt időm ilyen „léha” dolgokkal foglalatoskodni, így gondoltam felteszem a gépemre, és kipróbálom. A dolog azonban koránt sem bizonyult olyan egyszerűnek, ahogy azt én reméltem. Kicsomagoltam és megpróbáltam elindítani, de nemigen akart elindulni. Hosszas kínlódás után odahívtam Bozót, és a segítségét kértem. Ez az alkalom nagyon megmaradt bennem. Nem ekkor láttam először valakit hibát elhárítani, de ennél a programnál szinte minden praktikát latba kellett vetni, hogy hajlandó legyen elindulni. Hosszú, keserves küzdelem volt, de végül megadta magát. Azóta számtalanszor hasznát vettem az ott szerzett ismereteknek, ezért úgy érzem, érdemes másokkal is megosztani a hibaelhárítás általános módszereinek alapjait. Mikor anyagot gyűjtöttem ehhez az előadáshoz, rá kellett jönnöm, hogy nemcsak az iskolák tananyagaiból, de jóformán az Internetről is hiányzik egy jól összeállított hibaelhárítási útmutató.

2. A hibák típusai

A hibákat, mint a bocikat és a tesztaszűrőket, többféle módon lehet csoportosítani. Mi most két fontos csoportosítást fogunk megnézni, mivel a hibák elhárítása szempontjából ezek a legfontosabbak. Az első lehetséges csoportosítási szempont: a program felkészült-e a hiba kezelésére? A második, és a problémák elhárítása szempontjából fontosabb: a program jelzi-e a hibát, és ha igen, akkor érthetően-e?

2.1. A program felkészült-e?

Az, hogy a program felkészült-e egy bizonyos hibára, a tervezőjétől és programozójától függ. Megfelelő alapossággal megtervezett programok a hibák nagyobb részét megfelelően kezelik, bizonyos körülmények között még javítani is képesek, vagy akár megoldási javaslatot is tudnak adni. A tömegesen használt programok nagy előnye, hogy a felhasználók visszajelzései alapján még akkor is fény derül a hibákra, ha a tervezés nem volt elegendően alapos. Mivel a szoftverek tervezésére sokszor kevesebb időt szánnak (és ez nemigen függ attól, hogy szabad-e az adott szoftver), így ez egy nagyon fontos javító mechanizmus.

Érdekes kérdés, hogy a program mit csinál egy adott hiba esetén. Ha a program fel van készítve a rendellenes működésre (például ha egy konfigurációs állomány nem található), akkor helyes reakció várható el tőle. Gyakori hiba azonban, hogy a fejlesztők nem ellenőrzik megfelelően az általuk végrehajtott feladatok helyes kimenetelét, és a program menetét a megfelelő végrehajtás hitében folytatják. Nagyon egyszerű példa egy átmeneti állomány létrehozása az aktuális könyvtárba. Amennyiben a fejlesztő nem ellenőrzi az állomány megnyitásakor, hogy az sikerült-e, és elkezd beleírni, akkor bizony fejre áll a helyes működés. Megfelelő hozzáállással minden meghívott függvény-nél le kell ellenőrizni, hogy az a várakozásoknak megfelelő eredményt produkálta-e, de ez a programozási munkát nagyjából megkétszerezi, így a fejlesztők gyakran eltekintenek tőle. Ha a program olyan visszatérési értékkel találkozik, amire nincs felkészülve, akkor illene neki kilépni, mégpedig olyan hibaüzenettel, mely alapján a hiba helye

később behatárolható. Így a program nem kezdene el szokatlanul, összefüggéstelenül működni.

2.2. A hiba megjelenik-e a felületen?

A felhasználó szempontjából elsődleges fontosságú szempont a hiba jelzésének megléte vagy hiánya. Fontos megérteni, hogy a gyakrabban használt programok a hibák nagyon nagy részét jelzik valahol, valamilyen formában. Ezeknek a hibáknak az aránya 90-95%. Ha a felhasználó képes arra, hogy a hibaüzenetet *elolvassa*, megfelelően *értelmezze*, akkor gyakran máris nyert ügye van. A lehetséges esetek kellemetlenségi sorrendben:

- A hibaüzenetet a felhasználó nyelvén szól és érthető
- A hibaüzenetet angol nyelvű és érthető
- A hibaüzenetet urdu nyelvű és az urduk számára érthető
- A hibaüzenetet tetszőleges nyelvű, de nem érthető
- A hibaüzenetet egy bináris kód
- A hiba megjelenése valami nagyon jellemző rendellenes működés

Ha van hibaüzenet, akkor el kell olvasni. Tudom, ez triviálisnak tűnik, de az a tapasztalatom, hogy a felhasználók nagyon jelentős része nem teszi meg, vagy nem elég alaposan és ezért teljesen félreérti. Vagy azért marad el a hibaüzenet elolvasása mert a felhasználó nem látja értelmét, vagy azért, mert nem tudja, hogy hol tud egy program hibát jelezni. Ezért fontos megvizsgálni, hogy a különböző típusú programok hol jelezhetik a felhasználónak az általuk felfedezett hibát.

2.2.1. A hibaüzenet megjelenési formái

A program a felhasználói felületen jelzi a hibát. Mit is jelent ez? A program felületén ebben az esetben nem csak a grafikus felülettel rendelkező programok ablakait értjük, hanem minden olyan csatornát, amelyen a program képes információt közölni a felhasználóval. Egyszerűbb esetben ez lehet a grafikus felület, de sok programnál gyakori a terminál STDOUT vagy STDERR kimenetek használata visszajelzésre (a parancssorosoknál pedig kizárólag ez használható). A szerverek nem rendelkeznek sem grafikus felülettel, sem STDOUT és STDERR kimenettel, mivel ezeket le kell zárniuk, így az egyetlen jelzési lehetőségük a naplóállományok. (Természetesen ez esetben is vannak ritka kivételek, akik rendelkeznek grafikus felhasználói felülettel, amely kapcsolódik a szerverhez, és jelzi annak állapotát, de ez nem gyakori.) A naplóállományok lehetnek saját vagy a gép naplózó alrendszere által kezelték is. Azt, hogy egy szerver mit használ, annak dokumentációjából lehet megtudni. Nem ritka, hogy egy program levélben vagy webes felületen jelzi a felhasználónak a rendszerben beállt hibákat (pl. cron, monitor rendszerek stb.). Speciális alkalmazásoknál szükség lehet a hang alapú jelzések használatára is. Erre jól ismert példa a BIOS, ahol monitor híján a rendszer nem képes másként jelezni a problémákat, de komolyabb szerverrendszerek összevont monitoring alkalmazás híján csak hangjelzéssel képesek a hibákat jelezni, mert nincs minden rendszerre konzol csatlakoztatva. Kissé más eset, de egyfajta hibajelzés, ha a program valamilyen a hibára nagyon jellemző, speciális körülmények között „leheli ki a lelkét”. Például egy webböngésző nem ír ki hibát, de körülbelül fél percig teljesen

leterheli a processzort, aztán minden figyelmeztetés nélkül elszáll. Ez nem hibaüzenet a szó klasszikus értelmében, de egy tapasztalt felhasználó számára legalább annyira beszédes, mintha egy hibát jelző ablak jött volna fel a kilépés előtt.

Amikor hibát keresünk, akkor alaposan át kell gondolnunk, hogy ha a program közölni szeretne velünk valamit, akkor hol teheti ezt meg. A felhasználók elég nagy része hajlamos megfedkedezni a nem triviális felhasználói felületekről. Egy szerver hibája esetén teljesen természetessé kell válnia a naplóállományok vizsgálatának, egy grafikus program használata esetén pedig nagyon hasznos lehet, ha a felhasználó terminálból indítva ellenőrzi, hogy a program nem ír-e ki valamit a szabvány hibakimenetre (STDERR) mielőtt összeomlik. Ha a felhasználó nem ellenőriz minden kimeneti lehetőséget, akkor hajlamos azt gondolni, hogy a program minden hibaüzenet nélkül lépett ki, és így lényegesen nehezebb a hiba okának felderítése.

2.2.2. A program bőbeszédűségének befolyásolása

A program kiírásainak mennyiségét szinte minden esetben befolyásolni lehet. Ezzel gyakran követhetjük nyomon a hiba kialakulásának lépéseit, így még akkor is könnyebb kitalálni, hogy mi lehet a gond, ha a program nem írja ki. Más esetben a bőbeszédű üzenetek gyakran jóval többet árulnak el a hibáról, és annak lehetséges elhárításáról. Parancssoros programoknál erre a leggyakrabban a `-v` és a `-q` opció van hatással, melyek rendre bőbeszédűbb vagy csendesebb működést váltanak ki. Nem ritka az olyan program, ahol a `-v` opciók számával befolyásolható a kimenet mennyisége (pl. lilo). A szerverek általában a konfigurációs állományból veszik a naplózás helyét és beállításait. Ha egy szerverrel gond van, akkor érdemes megnézni a naplózás beállítási lehetőségeit, ezzel sokszor rá lehet szedni, hogy a működését naplózza, így a hiba helye pontosan láthatóvá válik. A legtöbb szerver naplózási szinteket használ, ami azt jelenti, hogy a naplózás kikapcsolható, az alacsonyabb naplózási szinteken (ami lehet például az INFO szint) csak bizonyos üzenetek jelennek meg a kimeneten, magasabb szinten (pl. DEBUG szint) pedig szinte minden apró műveletről napló képződik. A magasabb szintek használata kizárólag hibakeresésnél célszerű, mivel a nagyon nagy mennyiségű kimenet akár a naplózó partíciót is megtöltheti. Néhány szerver üzenettípus alapján is megengedi a naplózandó adatok szelektálását, itt gyakran egy bitmaszk segítségével állítható be, hogy milyen típusú üzeneteket szeretnénk a naplóban látni (pl. openldap szerver)

2.2.3. Ha van ráutaló hibaüzenet vagy esemény

Ha a program valamit kiírt, akkor arra már rá lehet keresni az Interneten. Az eddigi tapasztalataim szerint az ember nagyon ritkán akad olyan hibára, amely neki okoz először fejfájást. Ha valaki más találkozott már hasonló hibával, akkor az Interneten nagy valószínűséggel többet is megtudhatunk az adott problémáról és gyakran megoldási javaslatot is találhatunk. Kérem higgye el nekem a kedves olvasó: az esetek nagyon nagy részében sikerre vezet, ha sikerül megtalálni a program hibaüzenetét, és arra a Google keresővel rákeresünk. Ennek az információnak az átadása az előadás elsődleges fontosságú célja. Ha azonban nincs megfelelő hibaüzenet, akkor elő kell vennünk a Linux buherátor fegyvertárát.

3. Ha nincs értelmezhető hibaüzenet

Ha a program nem ír ki semmit, csak nem az elvárt módon viselkedik, például érthetetlen okból csak áll és vár vagy éppen elszáll, mint a győzelmi zászló, akkor a program működésének megfigyelésével kaphatunk információt arról, hogy mi lehet a gond.

3.1. A program függvényhívásainak követése

A hibaelhárító eszköztárban a leghatékonyabb fegyver a *strace* program. Segítségével egy futtatott vagy már futó program rendszerhívásai (system call) és üzenetei (signal) figyelhetők meg. Ha valaki átlátja a *strace* parancs kimenetét, akkor képes lesz megérteni, hogy mit csinál a program. Így kis tapasztalattal könnyen eldönthető, hogy egy adott gond esetén mi lehet a valódi hiba. Nézzünk egy példát. A programot C-ben írtam meg, így mindenki által könnyebben áttekinthető, hogy hol a hiba. Természetesen az itt bemutatott technika teljesen megegyezik akár más fordítók által generált bináris, akár más értelmezők által végrehajtott programnál. A kis példaprogramocská:

```
#include<sys/stat.h>
#include<fcntl.h>

int main()
{
    int fd;

    mkdir("fa", 0755);
    chdir("fa");
    fd = open ("alma", O_WRONLY | O_CREAT | O_NONBLOCK | O_NOCTTY,
               S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
    write(fd, "alma\n", 5);
    close(fd);
    return 0;
}
```

Ezt a programot nem a program írójának megfelelő körülmények között lefuttatva nyilvánvalóan hiba keletkezik, a program minden kimenet nélkül fog kilépni, és nem csinálja meg a dolgát. Ennek oka, hogy a fejlesztő nem ellenőrizte le a meghívott függvények visszatérési értékét. Ez sajnos elég gyakori hiba, így felismerésének és az általa okozott gond elhárításának ismerete a hibák újabb óriási részének elhárításában nyújt nagy segítséget.

Ha a program ideális körülmények között fut le, akkor az alábbiakat írja ki a *strace*:

```
1. execve("./teszt1", ["/teszt1"], [/* 21 vars */]) = 0
2. uname({sys="Linux", node="rose", ...}) = 0
3. brk(0) = 0x804a000
4. old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
             MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40017000
5. access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT
   (No such file or directory)
6. open("/etc/ld.so.preload", O_RDONLY) = -1 ENOENT
   (No such file or directory)
7. open("/etc/ld.so.cache", O_RDONLY) = 3
8. fstat64(3, {st_mode=S_IFREG|0644, st_size=59696, ...}) = 0
9. old_mmap(NULL, 59696, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40018000
10. close(3) = 0
11. access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT
   (No such file or directory)
12. open("/lib/tls/libc.so.6", O_RDONLY) = 3
```

```

13. read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\340X\1"... ,
    512) = 512
14. fstat64(3, {st_mode=S_IFREG|0644, st_size=1279140, ...}) = 0
15. old_mmap(NULL, 1289452, PROT_READ|PROT_EXEC, MAP_PRIVATE, 3, 0) =
    0x40027000
16. old_mmap(0x40157000, 36864, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_FIXED, 3, 0x12f000) = 0x40157000
17. old_mmap(0x40160000, 7404, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x40160000
18. close(3) = 0
19. old_mmap(NULL, 4096, PROT_READ|PROT_WRITE,
    MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40162000
20. set_thread_area({entry_number:-1 -> 6, base_addr:0x401622a0,
    limit:1048575, seg_32bit:1, contents:0, read_exec_only:0,
    limit_in_pages:1, seg_not_present:0, useable:1}) = 0
21. mmap(0x40018000, 59696) = 0
22. mkdir("fa", 0755) = 0
23. chdir("fa") = 0
24. open("alma", O_WRONLY|O_NONBLOCK|O_CREAT|O_NOCTTY, 0666) = 3
25. write(3, "alma\n", 5) = 5
26. close(3) = 0
27. exit_group(0) = ?

```

Ez először talán kissé összetettnek tűnik, de ha megismeri valaki, akkor hatalmas segítség egy program hibás működésének felderítésénél. A *strace* program a rendszershívásokat mutatja. A fenti példában a későbbi magyarázatok miatt sorszámoztuk a sorokat, ezt a *strace* nem teszi. A *strace* kimenetének minden sora két részből áll. A sor bal oldalán a meghívott rendszershívás és annak paraméterei láthatók, a jobb oldalán, az egyenlőség jel után a függvény visszatérési értéke. Ha a függvény végrehajtása hibátlan, akkor a visszatérési érték általában 0, a hibás esetekben pedig általában -1. A gyakorlat alapján egy hiba keresésénél a leggyakrabban az *open* rendszershívás végrehajtása közben merül fel hiba. Ez azt jelenti, hogy a program meg akar nyitni egy állományt mert olvasni szeretne belőle vagy írni akar bele, de nem képes rá. Ez általában két okra vezethető vissza: jogosultsági problémára vagy a keresett állomány nem található ott, ahol a program keresi. Ha ennek a két dolognak a jeleit képesek vagyunk felfedezni a *strace* kimenetében, akkor már tipikus a hibák legnagyobb részét is képesek leszünk elhárítani. A *strace* napló első részén a program indulásával kapcsolatos rendszerműködés figyelhető meg.

A program indítási folyamata az 1–21. sorokban követhető nyomon. Ezek után a program által hívott rendszershívások láthatók. Érdeemes megfigyelni, hogy a rendszer a program indulásakor betölti a programhoz linkelt függvénykönyvtárakat. Ezt láthatjuk a 12. sorban. Mikor egy program nem akar elindulni, annak oka lehet az is, hogy az indulásához nem áll rendelkezésre egy szükséges függvénykönyvtár (lib). Ez általában csak akkor fordul elő, ha a lefordított programot nem csomagból telepítettük, hisz ez utóbbi esetben a csomagkezelő jó esetben gondoskodik a szükséges lib-ek rendelkezésre állásáról. A 22. és 27. sorok között láthatjuk a program által hívott függvényeket, és azok visszatérési értékét. Az *open* rendszershívás által visszaadott 3 a sikeresen megnyitott fájl un. filedescriptor-ának azonosító száma. Ha később a program I/O műveletnél ezt a számot látjuk (például a 25. sor első paramétere), akkor az azt jelenti, hogy a megadott számú FD-t használja. Innen később azonosíthatjuk, hogy mely fájlknál vagy folyamatknál lehet írási vagy olvasási probléma. A 25. sorban a *write* a sikeresen kiírt karakterek számával tér vissza.

A példaprogramra tekintve megfigyelhetjük, hogy a program írója nem ellenőrizte, hogy sikerül-e létrehozni a „fa” könyvtárat. Ha már létezik, akkor bizony a program

futása hibás (bár ez ebben az esetben nem okoz gondot). Most ezt írja ki a *strace*:

```
...
mkdir("fa", 0755)                = -1 EEXIST (File exists)
chdir("fa")                      = 0
...
```

Látszik, hogy az *mkdir* rendszerhívás nem nullával tért vissza. Ilyen esetben a rendszertől egy külön függvényhívással le lehetne kérdezni a hiba okát, a *strace* a visszatérési érték után előzőeken odaírja, hogy mi volt a hiba. Innen látszik, hogy az adott könyvtár már létezik. Ez ebben az esetben azért nem jelent problémát, mivel belépve a program helyesen működik tovább, de ha ezt a könyvtárat korábban egy másik felhasználó hozza létre, akkor ezt látjuk:

```
...
mkdir("fa", 0755)                = -1 EEXIST (File exists)
chdir("fa")                    = -1 EACCES (Permission denied)
open("alma", O_WRONLY|O_NONBLOCK|O_CREAT|O_NOCTTY, 0666) = 3
write(3, "alma\n", 5)          = 5
close(3)                      = 0
...
```

Ez lényegesen csúnyább kimenetel mint az előbb. Látszik, hogy a könyvtárat nem sikerül létrehozni, mivel az már létezik. Ez után a program megpróbál belépni, de ez sem sikerül, mivel nincs joga hozzá. És itt máris látszik az, hogy mekkora hibát okozhat a visszatérési érték vizsgálatának hiánya. Mivel a program nem ellenőrizte, hogy sikerült-e belépni a könyvtárba, így minden további nélkül létrehozza az írandó állományt, és beleír. Ha ennek a fájlnak a létrehozása az aktuális könyvtárban hibát okozna, például mert a későbbiek folyamán teljes elérési úttal szeretné majd újra megnyitni, akkor a programunk máris rossz irányba megy. Ettől azonban sokkal nagyobb hibát is okozhatnak az ilyen figyelmetlenségek. A fenti példában követhető, hogy a hibát a könyvtár létezése és a jogosultsági probléma okozza.

Előfordulhat, hogy a programnak nincs joga írni az adott könyvtárba. Ezt illusztrálja a következő kimenet:

```
...
mkdir("fa", 0755)                = -1 EACCES (Permission denied)
chdir("fa")                    = -1 ENOENT (No such file or directory)
open("alma", O_WRONLY|O_NONBLOCK|O_CREAT|O_NOCTTY, 0666) = -1 EACCES
                               (Permission denied)
close(-1)                      = -1 EBADF (Bad file descriptor)
exit_group(0)                  = ?
```

A program végrehajtása helyesen ér véget, a nyomkövetési naplóból azonban látszik, hogy valójában semmit nem csinált. A könyvtár létrehozásához nem volt joga, így nem volt képes belépni sem, az állomány megnyitásához és írásához sem volt joga. A felhasználó hibaüzenet híján azt hiheti, hogy minden rendben, de ez nem igaz. Egy utolsó kis példa a *strace* használatára a következő:

```
...
mkdir("fa", 0755)                = 0
chdir("fa")                    = 0
open("alma", O_WRONLY|O_NONBLOCK|O_CREAT|O_NOCTTY, 0666) = -1 EACCES
                               (Permission denied)
write(-1, "alma\n", 5)          = -1 EBADF (Bad file descriptor)
close(-1)                      = -1 EBADF (Bad file descriptor)
exit_group(0)                  = ?
```

Látható, hogy a könyvtár létrehozása hibamenetesen sikerült, a belépés is, a fájl megnyitása írásra mégis sikertelen. Itt már szükség van a rendszer ismeretére is. Aki Unix rendszeren programoz, az tudja, hogy az *mkdir* függvény figyelembe veszi az umask értékét is. Hiába adta meg nagyszerű programozónk a 0755 jogosultságot, az umask, amely ebben a tesztben 0277-re volt állítva, nem engedte ennek a betartását. Így a keletkezett könyvtárba még a saját tulajdonosa sem képes állományokat létrehozni.

A *strace* programhoz nagyon hasonló az *ltrace*, amely a függvény-könyvtári hívások nyomkövetését teszi lehetővé. Használatával a napló általában jóval hosszabb lesz, de nem csak a libc6 hívások követhetők nyomon. Egyébként használata nagyon hasonlít a *strace* használatához.

Mindkét program legfontosabb paraméterei:

- f Ha a program fork rendszerhívást használ, akkor a gyereket is követi a program. A gyerek folyamatok úgy különböztethetők meg, hogy a *strace* a folyamat azonosítóját (PID) is odaírja a napló sorok elejére.
- F A vfork követésére. Az egyszerűség kedvéért érdemes megjegyezni, hogy a *-f* mellé érdemes mindig odatenni, és így minden folyamat látszani fog a naplóban.
- s Kiírás és olvasás esetén ez a paraméter határozza meg, hogy mennyi karakter (bájt) látszon a naplóban. Megemelésének gyakran vehetjük hasznát, ha a program nem az elvárt I/O-t produkálja.
- o A kimenet nem a szabvány kimenetre kerül, hanem a megadott állományba. Ennek több előnye is van. Az interaktív programok követésére is használható, másrészt később kényelmesen elemezhető a nyomkövetési naplót tartalmazó fájl.
- p Egy már futó folyamathoz való csatlakozásnál ezzel adhatjuk meg a folyamat azonosítóját (PID).

Ha a program működésével baj van, a *strace* használatával nyomkövetési naplót kell készíteni, és annak a vége felé egy hibás visszatérési értéknél általában meg lehet találni a probléma forrását. Tapasztalatok szerint ezzel és némi gyakorlattal a problémák 99,9%-a megoldható.

3.2. Hálózati kommunikációs problémák

Gyakori eset, hogy a program helyesen működne, de valami miatt a hálózati kommunikációja nem sikeres. Ennek a problémacsoportnak az elhárítására is használható a korábban már ismertetett **trace* család, de néha gyorsabb és egyszerűbb a hálózati forgalmat megfigyelni. Azt, hogy a program milyen hálózati kommunikációt folytat, több módszerrel is követni lehet. A legegyszerűbb a *netstat* használata. Ez kiírja, hogy az aktuális rendszeren milyen szolgáltatások figyelnek (ha a szerverünknek nem sikerül egy adott porton figyelni, annak oka lehet, hogy már más figyel ott), és megadja, hogy az adott rendszernek milyen hálózati kapcsolatai vannak. Ha a hálózat forgalmának mélyebb elemzésére van szükség, akkor használható a *tcpdump*, amely lehetővé teszi a kommunikáció teljes nyomkövetését és naplózását, csak bizonyos mélység után már nehezen használható. Ha a protokollok részletes elemzésére van szükségünk, akkor használható az *ethereal* és a *tethereal*, amelyek grafikus és szöveges protokoll elemző programok. Maguk is képesek a forgalom rögzítésére, de a *tcpdump* által rögzített forgalom is elemezhető velük. Megoldható tehát, hogy a *tcpdump* segítségével rögzítsük a hálózati forgalmat egy szerveren, és az adminisztrátori munkaállomáson elemezzük.

Ilyen esetben nem szabad megelégedezni a *tcpdump* program *-s* opciójáról, különben a napló később nem elemezhető teljes részletességgel, mert a csomagoknak csak egy része lesz az állományba írva.

Ha a rendszert valamilyen tűzfal védi, akkor mindenképpen annak naplóállományait is alaposan át kell vizsgálni, hogy nincs-e olyan kapcsolat vagy protokollelem, melyet az visszautasított, és ezzel a kommunikációt lehetetlenné tette.

3.2.1. Nyomkövetés és a core fájlok

Ha az ember megfelelő ismeretekkel és elszántsággal rendelkezik, akkor képes a fenti-ektől mélyebben is elemezni egy program működését. A debuggerek lehetővé teszik a programbeli változók értékének futás közbeni nyomon követését, ún. töréspontok elhelyezését, melyeket akár feltételekhez is lehet kötni. Linux alatt leggyakrabban használt debugger a gdb, amely óriási tudású, egyetlen hátránya, hogy parancssoros volta miatt kissé kényelmetlenül használható. Szerencsére létezik sok nagyszerű kiegészítő eszköz, mely megkönnyíti a használatát. Ilyen például a DDD, amely grafikus felületen teszi láthatóvá a program futtatása közben annak forráskódját (amennyiben elérhető), a program által használt változók értékeit és még sok hasznos dolgot. A gdb-t használhatjuk core fájlok elemzésére is. Mikor a program elszáll, akkor gyakran a munkakönyvtárban hagy egy ilyen core fájlt. Ez az állomány lehetővé teszi a rendszer fejlesztőinek, hogy meghatározzák a hiba helyét, és a program állapotát az elszállás pillanatában, így nagymértékben leegyszerűsödik a hibák okának feltárása. A programok nyomkövetése és a core fájlok elemzése hasznos, azonban igen komoly felkészültséget igényel, a használt módszerek leírása meghaladja a cikk méretét.

3.3. Még egy hasznos eszköz, az lsof

Az *lsuf* parancs a folyamatok által nyitott fájlokról ír ki információt. A fájl lehet hagyományos fájl, könyvtár, blokk vagy karakter orientált fájl, executing text reference, library, stream vagy network fájl (Internet socket, NFS fájl vagy UNIX domain socket). Hasznos segédeszköz lehet például, ha nem tudunk lecsatolni egy partíciót. Ebben az esetben az *lsuf* paranccsal (*lsuf +D <dirname>*) meg tudjuk nézni, hogy ki tart nyitva fájlt az adott könyvtár alatt. A másik gyakori használata, hogy megnézzük, hogy egy processz milyen fájlokat tart nyitva (*lsuf -p <pid>*).

3.4. A hardver hibák elhárítása

Ugyan szorosan nem tartozik a témához, hisz az előadás tárgya a szoftveres hiba, de fontos megemlíteni a vas meghibásodásait is. Ha a rendszer bizonytalan időközönként lefagy, újraindul, a programok néha érthetetlen módon elszállnak, egy eszköz bizonytalanul működik, azt gyakran hardver hiba okozza. A házilag szerelt PC-knél a leggyakoribb hiba a rossz minőségű memória bizonytalan működése. A memória általában nem szokott teljesen meghalni, hanem látszólag helyesen működik, csak a gép működését teszi esetlegessé. A memóriahibák megbízható felderítésére használható a *memtest86* program, amelyet egy hajlékony lemezre írva, majd rábootolva célszerű egy egész napon keresztül futtatni, és csak akkor lehetünk bizonyosak a memória hibátlanságában, ha többször végigellenőrizte a teljes memóriát és nem talált semmilyen hibát. A memória hosszas hibátlan tesztelése a processzor hibátlanságát is bizonyítja, bár a processzor csak nagyon ritkán szokott „kicsit” meghibásodni. Általában egyszerűen feladja, ilyenkor már csak az ajtó kitámasztására használható.

A merevlemez hibái általában nem előzmény nélküliek. Gyakran a rendszermag üzenetei előre jelzik a diszkek későbbi meghibásodását. Ilyenkor például az látszik a naplókban, hogy egy-egy olvasási vagy írási műveletet csak sokadszorra lehetett végrehajtani. Ilyen esetben először ellenőrizni kell, hogy a rendszerben nincs-e fizikai probléma, például hibás vagy kilazult kábel, mert az is okozhat ilyen problémákat. Ha azonban minden rendben lévőnek látszik, akkor az adott diszk cseréjét be kell tervezni, és a lehető leggyakrabban mentést kell készíteni a rajta lévő fontosabb adatokról. Példa kernel napló, ami diszk hibára utalhat:

```
kernel: hdc: dma_intr: status=0x51 { DriveReady SeekComplete Error }
kernel: hdc: dma_intr: error=0x01 { AddrMarkNotFound },
        LBASect=20300322, sector=1263288
kernel: hdc: dma_intr: status=0x51 { DriveReady SeekComplete Error }
kernel: hdc: dma_intr: error=0x40 { UncorrectableError },
        LBASect=20803307, sector=1766272
kernel: end_request: I/O error, dev 16:04 (hdc), sector 1766272
```

Az `e2fsck` program segítségével (`-c` kapcsoló) megkereshetjük a diszken levő hibás blokkokat (bad block) és megjelölhetjük őket, hogy ne is próbálja használni a rendszer ezeket. Ha ezek elkezdenek szaporodni, akkor szintén új diszk után kell néznünk előbb vagy utóbb. Ennek használata azonban kényelmetlen, mivel a lemez tartalmát le kell menteni, és egy mai, nagy méretű diszk esetén vizsgálat akár nagyon hosszú ideig is eltarthat.

A kábelhibák rendszeresen meg tudják keseríteni az ember életét. Egy komolyabb rendező előtt az ember nagyon kicsinek érzi magát (a feladatot meg reménytelennek). A kábel hibáját könnyedén megállapíthatjuk egy kábel tesztelővel. Sajnos nem mindig van ilyen az embernél, ezért inkább próbáljuk meg lecserélni egy biztosan hibátlan kábelre a valószínűsíthetően rossz kábelt. Mi az ami kábelhibára utalhat? Például, ha el van vágva a kábel. Ezt nagyon egyszerű észrevenni. Ezenkívül a nagy csomagvesztés is jelenthet kábelhibát. A kábelhibák esetén először ellenőrizzük, hogy jól be van-e dugva a kábel, illetve előfordulhat, hogy nem megfelelő kábellel próbálkozunk (például két gépet akarunk összekötni egy egyenes kábellel). Azt, hogy egy hálózati eszközön vannak-e hibára utaló csomagok, az `ifconfig` parancs segítségével egyszerűen el lehet dönteni. Az alábbi példában látszik, hogy egyetlen hibás csomag volt a vett csaknem két millióból.

```
scylla:~# ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 00:fa:4e:fa:4e:ff
       inet addr:111.11.111.11  Bcast:111.11.111.255  Mask:255.255.255.0
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:1821683 errors:1 dropped:0 overruns:0 frame:2
       TX packets:1618331 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:369155178 (352.0 MiB)  TX bytes:1351150386 (1.2 GiB)
       Interrupt:5  Base address:0xd400
```

Ez elfogadható eredmény. Ha a hibás csomagok száma elkezd szaporodni, az azt jelenti, hogy vagy az adott gép hálózati kapcsolatával vagy csatlóójával, vagy a hálózatában lévő másik géppel problémák vannak. Ez gyakran a hálózat lassulását is maga után vonja. Az `ifconfig` kimenetből az `errors` (hibás csomagok száma), `carrier` (a carrier változások száma), `collisions` (ütközések száma) magas értéke esetén nem árt szétnézni a hálózati eszközök között.

SPAMek és vírusok: elmosódó határok

Nemes Dániel

dnemes@filtermax.hu

Tartalomjegyzék

1. A vírusok terjedése	112
1.1. A sérülékenység ablaka (window of vulnerability)	112
1.2. Egy életből ellesett példa: MyDoom.A	112
1.3. Gyanúsan ártalmatlan vírusok	113
2. A SPAM-vírus konvergencia	114
2.1. Spyware	114
2.2. Phishing	114
3. Mobil SPAM	115
4. Mít hoz a jövő?	115

1. A vírusok terjedése

Ma már talán senki nem emlékszik a floppyezeken terjedő vírusokra. Ma a digitális kártevők túlnyomó része futtatható állomány, makróvírus és ártó script. A vírusíróknak értelemszerűen céljuk, hogy a legegyszerűbb megoldással a legnagyobb fertőzést tudják okozni, ezért igyekeznek minél inkább sztenderd megoldásokat használni. Ezért a leggyakoribbak a legelterjedtebb operációs rendszerre, esetleg a leggyakoribb böngészőkre, irodai alkalmazásokra, e-mail kliensekre tervezett vírusok. Ma a kártevők több mint 99%-a e-mailen érkezik. Ennek oka, hogy a jól kiépített, világméretű SMTP infrastruktúra kitűnő táptalaj a vírusok terjedésének: biztonsági hiányosságai és gyors működése egyaránt a vírusok terjesztésére predesztinálta, sztenderdizáltsága pedig egyszerű alkalmazását biztosítja a kártevők írói számára.

1.1. A sérülékenység ablaka (window of vulnerability)

Egy vírus életciklusa egy haranggörbe mentén írható le, melynek állomásai az alábbiak:

- a vírus terjedni kezd,
- felismeri a fertőzést egy szakértő felhasználó vagy egy víruslabor,
- az antivíruscégek hozzájutnak a mintához,
- a szignatúra fejlesztése, tesztelése,
- a szignatúra letölthető adatbázisba illesztése,
- az adatbázist az átlagos felhasználó letölti,
- lecsengés.

Egy digitális kártevő elsődleges célja ezért a minél gyorsabb terjedés, hiszen a hagyományos antivírus megoldások a vírus-szignatúrák elkészültéig nem védenek az újfajta támadások ellen, leszámítva azt a heurisztikát, mely szerencsés esetben is csak az újabb variánsok ellen véd. A haranggörbe csúcsa, a vírus leginkább intenzív terjedése tehát addig tart, amíg az átlagos felhasználó letölti (és beilleszti) a legfrissebb vírusadatbázist, amely tartalmazza az új vírus felismeréséhez szükséges kódot.

Ez az idő azonban meglehetősen hosszú: ugyan ma az átlagos felhasználó már kevesebb, mint 24 órán belül hozzájut a friss adatbázishoz, annak elkészítéséhez azonban az első fertőzés detektálásától számított 8–36 órára van szüksége az antivíruscégeknek.

1.2. Egy életből ellesett példa: MyDoom.A

A MyDoom.A, mint az eddigi egyik legfertőzőbb károkozó, kitűnően felhasználható esettanulmányként. Első lépésként vizsgáljuk meg a különböző cégek reakcióidejét:

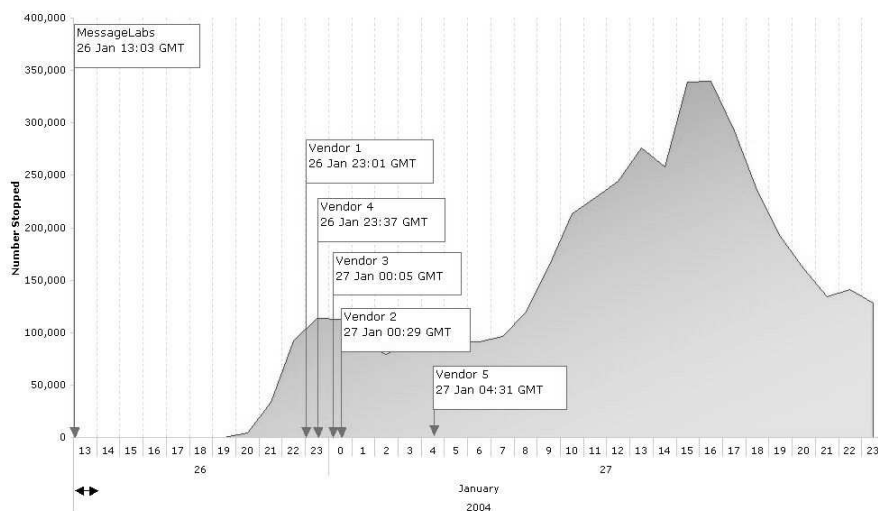
McAfee (BETA)	26.01. 23:15	W32/Mydoom.dll
Trend Micro	26.01. 23:35	WORM_MIMAIL.R
Trend (BETA)	26.01. 23:35	WORM_MIMAIL.R
Virusbuster	27.01. 00:05	Trojan.Mydoom.A
AVG	27.01. 00:15	I-Worm/Mydoom
Symantec (BETA)	27.01. 00:35	W32.Novarg.A@mm
InoculateIT-CA	27.01. 01:20	Win32/Shimg.Worm
Sophos	27.01. 01:40	W32/MyDoom-A
InoculateIT-VET	27.01. 02:30	Win32.Mydoom.A
Esafe	27.01. 02:50	Win32.Mydoom.a.dll
RAV	27.01. 04:10	Win32.Mydoom.A.dll
Dr. Web	27.01. 04:10	Win32.HLLM.Foo.32768
Kaspersky	27.01. 04:35	I-Worm.Novarg
Symantec	27.01. 04:35	W32.Novarg.A@mm
McAfee	27.01. 05:00	W32/Mydoom.dll
Bitdefender	27.01. 05:00	Win32.Novarg.A@mm
Panda (BETA)	27.01. 05:15	W32/Mydoom.A.worm
Quickheal	27.01. 05:50	W32.Novarg
Panda	27.01. 06:00	W32/Mydoom.A.worm
Norman	27.01. 09:05	MyDoom.A@mm
AntiVir	27.01. 12:35	Worm/MyDoom
F-Secure	27.01. 13:05	W32/Mydoom.A@mm
F-Prot	27.01. 19:15	W32/Mydoom.A
Avast	28.01. 17:00	Win32:Mydoom [DLL]
Command	28.01. 18:25	W32/Mydoom.A
A2	28.01. 19:40	Worm.Win32.Mydoom

MyDoom.A reakcióidők [1]

Amint a táblázatból látható, volt cég, amely egy korábbi vírus variánsaként fogta fel az új fenyegetést, mások elég sokat késtek – ezért is javasolt legalábbis több vírusmotor beszerzése. Amíg az átlagos felhasználó az átlagos vírusirtóját nem frissíti, addig a vírus terjedése egyre gyorsul. A MyDoom.A-ból a vezető kihelyezett e-mail biztonsági szolgáltató az első 24 órában 1,2 millió példányt fogott el. A vírus terjedését, valamint más gyártók reakcióidejét mutatja az ő mérésük szerint az 1. ábra.

1.3. Gyanúsan ártalmatlan vírusok

Egyesek szerint összeesküvés-elmélet, mások szerint jóslat, hogy a vírusok napjainkban egy minden eddiginél átfogóbb, elsőprő támadást készítenek elő. Akárhogy is vélekedünk erről, érdemes belegondolni: utoljára mikor lehetett igazán romboló hatású vírusokkal találkozni? Mikor volt utoljára merevlemezt formázó, állományokat törő, dokumentumokat szerteszét küldözgető vírusokkal dolgunk? Ilyenkor érdemes elgondolkodni azon, hogy egyetlen vírusmotorunk elegendő-e, érdemes-e kockáztatni, a véletlenre bízni a megoldást, illetve elfogadható-e a mai gyakorlat, miszerint sok helyen legyintenek egy-egy átcsúszott vírusra.



1. ábra. A Mydoom terjedése

2. A SPAM-vírus konvergencia

Mit is tesz egy modern vírus? Gyakran megnyitja egy támadó előtt a fertőzött PC-t, kiszolgáltatva azt a támadónak. Mit tesz a támadó? Például SPAMmel. SPAM-zombivá alakítja a számítógépet, aminek a gazdája sokáig semmit nem tud a fertőzésről, illetve hogy ő a SPAMmerek kezére játszik. Kimutatható, hogy a vírusírók egyre több SPAM-mer technikát építenek bele a vírus működésébe (mass-mailing, DHA¹, randomizálás), és szintén ismert tény, hogy a SPAMmerek egyre gyakrabban veszik igénybe vírusírók „szolgáltatásait” open proxyk, SPAM-zombik kialakításához. Így borul össze a két „iparág”.

2.1. Spyware

Természetesen nem csak SPAM-zombikká lehet gépeket változtatni, hanem a SPAMnél akár veszélyesebb támadásokra, információlopásra is lehet a nagy mennyiségben, e-mailben szétküldött spyware-eket használni.

2.2. Phishing

Az adatlopás ugyanakkor végtelenül egyszerű technikával, social engineeringgel is megvalósítható, melynek kitűnő példája a phishing, melyet elsősorban bankkártya-információk ellopására használnak. Az autentikusra megtévesztésig hasonló (vagy azonos, illetve annak tűnő) URL, illetve weboldal a gyanútlan felhasználóban bizalmat ébreszt. Több millió, tízmillió áldozat között pedig mindig akad néhány, aki bedől az ilyen cseleknek.

¹Directory Harvest Attack. Az SMTP-szervert „megkínálják” rengeteg generált címmel – amelyeket elfogadja, azt élő e-mail címnek tekintik.

3. Mobil SPAM

Már ma is egyre jobban terjed az SMS-SPAM, külföldön ugyanennek MMS-es formája is gyakori. Történik ez annak ellenére, hogy az SMS-SPAMmer sokkal jobban beazonosítható, mint e-mailes társa, illetve egy SMS-SPAM elküldése több nagyságrenddel nagyobb összegbe kerül, mint e-mailben. Gondoljunk viszont bele abba, hogy – akár a telefon kezelésében is alulképzett, informatikai analfabéta – mobilfelhasználók százezreit, millióit fertőzi meg – mondjuk BlueTooth-on – egy vírus, majd az küld el néhányszor tíz-száz SMS-t – a SPAMmer máris kikerülte a költséges kiküldést, hovatovább az e-mailhez hasonlóan kvázi beazonosíthatatlan lesz.

4. Mit hoz a jövő?

Egyre több intelligens eszköz vesz minket körül. Ma már a mobilon túl is terjed az internet-képes technológia, autókban, háztartási gépekben. Újabb és újabb drótnélküli technológiák jelennek meg és terjednek el (WIFI, BlueTooth, stb.), amik szintén a károkozók táptalaja lehetnek megfelelő védelem nélkül. Hogy a SPAMmelő mosógép és a fagyasztott pizzát leolvasztó vírus csak utópia-e, néhány éven belül kiderül.

Hivatkozások

[1] <http://www.av-test.org/>

Mire jó a Hunmorph?

Németh László

Kivonat

A Hunmorph morfológiai elemző a hozzá kifejlesztett magyar nyelvi erőforrással magyar szavakat elemez: megadja a bemeneti szóalak tövét, szófaját, toldalékainak nevét. A Hunmorph programmal, illetve programkönyvtárral és rokon alkalmazásaival kibővíthetjük a szótőelőállításon alapuló indexelés képességével intranetes keresőnket, vállalati dokumentumkezelő rendszerünket. Mellesleg magyar nyelvi ellenőrzők, fordítógépek, természetes nyelvi interfészek létrehozását teszi lehetővé bárki számára, az LGPL licencnek köszönhetően ingyenesen.

Tartalomjegyzék

1. Bevezetés	118
2. Spell, Ispell, International Ispell, Magyar Ispell	118
3. Hunspell, Hunmorph	120
4. Teljes elemzés	121
5. Összetett szavak kezelése	121
6. Architektúra	122
7. Összefoglalás	122

1. Bevezetés

A Szószablya projekt 2003-ban indult ITEM támogatással. A projekt 18 millió magyar weboldal szövegtartalma alapján kifinomult szűrési módszerekkel elkészített egy egyedülállóan nagy; több, mint 1,4 milliárd szövegszót tartalmazó szövegtörzset, és ennek több, a köznyelvi szókincs arányában különböző részkörzetét. A korpuszok szókincsét tartalmazó 7–19 millió szavas gyakorisági szótárak az `ftp://ftp.szoszablya.hu/` címről tölthetők le. A korpuszból nyert adatokkal kibővítésre került a nyitott forráskódú Magyar Ispell helyesírási és toldalékszótár, valamint folytatódott a Hunspell helyesírás-ellenőrző fejlesztése a morfológiai elemzés irányába.

Az előadáson bemutatjuk a Hunspell helyesírás-ellenőrző, Hunstem tövező, Hunmorph morfológiai elemző legfontosabb felhasználási területeit, valós példákkal illusztrálva.

A következőkben ismertetjük a helyesírás-ellenőrző és morfológiai elemző rendszer felépítését történeti vonatkozásaival együtt. Felsoroljuk a magyar nyelv morfológiai gazdagságából, illetve összetett helyesírási szabályaiból adódó problémákat és megoldásukat.

2. Spell, Ispell, International Ispell, Magyar Ispell

A szabad szoftverek világában a 70-es évektől elterjedt helyesírás-ellenőrzők fejlődése különösebb tervezés nélkül a morfológiailag összetettebb nyelvek támogatásának irányába haladt.

A Spell helyesírás-ellenőrző család őseit folyóírás felismerésének támogatására készítette L. Earnest 1961-ben ([Kuenning]). Az ellenőrző szótára a leggyakoribb 10 000 angol szót tartalmazta ([Earnest1963]). R. Gorin 1971-ben tovább bővítette a szótárt és készített egy hatékonyabb helyesírás-ellenőrző programot Spell néven ([Peterson1980]). W. Ackerman 1978-ban – miközben egy más gépre írta át a Spell programot – jelentős változtatásokat eszközölt a program algoritmusán: a korábbi heurisztikus, és így nem megbízható szuffixumleválasztás helyett bevezette a *szuffixumkapcsolókat* (suffix flags), amelyek egyértelműen meghatározták, hogy mely tövekhez milyen szuffixumok kapcsolódhatnak. A programot Ispellnek (ITS version of Spell) nevezte el.

A következő példa a program erőforrásállományainak felépítését – és részben a program működését – szemlélteti (nem az eredeti szintaxissal). A tőszótár egykarakteres szuffixumkapcsolókat tartalmaz perjellel elválasztva a tőtől:

```
dog/A
drink/AB
```

Az affixumállomány definiálja a szuffixumokat az A és B kapcsolóhoz:

```
SUFFIX A s
PREFIX B able
```

Egy tő a hozzárendelt szuffixumokkal is helyes szó (így a két szótári szón kívül a *dogs*, *drinks*, *drinkable* is helyes).

Az Assembly nyelvű Ispell program C változatát P. Willisson (1983) és W. Buehring (1987) írta meg. Az International Ispell változatban (1998) Geoff Kuenning bevezette az *affixumtömörítést* (affix compression), aminek köszönhetően egy affixumkapcsoló már nem csak egy affixumot, hanem affixumok egy tetszőlegesen nagy halmazát jelölheti. Példával:

SUFFIX A ba
 SUFFIX A ban
 SUFFIX A ból

Ha a szótárban szerepel a *vár/A* definíció, akkor a *várba*, *várban*, *várból* alakot is felismeri a program.

Kuening másik fejlesztése a Munch „affixumtömörítő” program, ami a morfológiailag egyszerűbb nyelvek, mint az angol, nyelvi erőforrásainak elkészítését automatizálja: képes meghatározni az optimális szuffixumokat, prefixumokat és töveket egy nagy szólista alapján. A Munch programmal előállított nyelvi erőforrás egyfajta automatikus tömörítése a bemeneti szólistának, éppen ezért a Munchsal megállapított tövek és affixumok nem fedik pontosan a nyelv morfológiáját. A morfológiailag összetettebb nyelvek esetében azonban kisebb jelentősége van a Munch programnak, mivel lehetetlen a megfelelő méretű szólistát összeállítani a nyelv által helyesnek tartott szóalakokból. Éppen ezért itt számíthatunk arra, hogy a nyelvi erőforrás nem automatikus módszerekkel készül, és követi a nyelv morfológiáját, mint azt a legtöbb Ispell (illetve MySpell) szótár esetében tapasztaljuk.

Az affixumokhoz megadhatunk egy illesztési feltételt és egy levágást is. A következő példában a harmadik mező a *tő* végéről levágandó karaktersorozatot tartalmazza a szuffixum illesztése előtt. Prefixum esetén a *tő* elejére vonatkozik a levágás. Ha a harmadik mező 0-t tartalmaz, akkor nincs levágás. A negyedik mező a *tő* végén alkalmazott illeszkedési feltételt tartalmazza egy karaktertartomány megadását is lehetővé tevő mintával. (A pont (.) tetszőleges karaktert jelöl. Kapcsos zárójelek között karaktertartományt adhatunk meg. Ha a nyitó zárójelet kalap (^) követi, akkor az a megadott karakterek komplementer karakterhalmazát jelöli¹)

SUFFIX B 0 val [óúv]
 SUFFIX B 0 ral r
 SUFFIX B z szal sz
 SUFFIX B 0 zal [^s]z

Az első sor jelentése, hogy az *ó*, *ú* és *v* karakterre végződő tövek megkaphatják a *val* szuffixumot. A harmadik sor jelentése, hogy az *sz*-re végződő szavak *szal* toldalékot kapnak, de a toldalék illesztése előtt levágásra kerül a *tő* végi *z* karakter az egyszerűsítő írásmód miatt. A negyedik sor szerint pedig azok a *z* karakterre végződő tövek, amelyek utolsó előtti betűje nem *s*, *zal* szuffixumot kapnak.

Ahogy a példák sejtetik, az International Ispell egyetlen szuffixumot képes a vizsgált szóról leválasztani, ami szűk lehetőséget kínál a ragasztó nyelvek – mint a magyar – támogatására, hiszen a számításba jöhető toldalékmorféma-kombinációk összes felszíni alakját fel kell tudni sorolni. Az International Ispellhez mégis sikerült használható (bár nem tökéletes) magyar nyelvi erőforrást készíteni ([Németh2002]). A Magyar Ispell keretrendszer állította elő a nagy számú (több, mint 15 000) szuffixumot, valamint a mintegy 200 ezer, felerészben produktív képzővel ellátott szót tartalmazó tőszótárt. Az Ispell által kezelt egyetlen szuffixum nem teszi lehetővé a képzők affixumtáblával történő tárhatékony kezelését. A nagyságrendekkel nagyobb számú szuffixum (pl. *tathatóságaikéiről*) felvétele helyett az igék, melléknevek és tulajdonnevek a leggyakoribb produktív képzős alakjaikban kerültek be a tőszótárba.

¹Magyar vonatkozása az International Ispell programnak, hogy a minta illeszkedésének vizsgálata Dömölki-algoritmussal lett megvalósítva.[Dömölki1967]

3. Hunspell, Hunmorph

Az OpenOffice.org nyitott forráskódú irodai programcsomaghoz Kevin Hendricks készítette 2002-ben egy helyesírás-ellenőrző programkönyvtárat Myspell néven, felhasználva az International Ispell algoritmusait (és a hozzá készült szótárakat). Az Ispell és a Myspell nyelvfüggetlen programok, minden egyes nyelvhez külön szótárat (a mi megfogalmazásunkban nyelvi erőforrást) kell létrehozni. Az OpenOffice.org népszerűségének tudható be, hogy már több, mint 40 nyelvhez érhető el hozzá ilyen nyelvi erőforrás, és folyamatosan készülnek az újabbak. Lényeges ismét kihangsúlyozni, hogy ezek többségéből kis átalakítással tövezési és morfológiai elemzési nyelvi erőforrás is készíthető.

A Hunspell és a Hunmorph fejlesztésének is a Myspell könyvtár képezte az alapját. Elsőként az összetett szavak kezelésének képességével lett kiegészítve a Myspell. (Ez része ugyan az Ispellnek, de a Myspell első változatából még hiányzott.) Számos egyéb bővítés után 2004-ben sikerült lényegesen javítani a magyar és egyéb ragasztó nyelvek támogatásán a *többszörös affixumleválasztás* és a homonimák kezelésének bevezetésével. A többszörös affixumleválasztás jelenlegi megvalósítása csak kétszeres szuffixumleválasztást takar ugyan, de ezzel is már négyzetesen csökkenthető a szuffixumok száma. A magyar nyelvi erőforrás affixumállományában így sikerült a produktív képzőket korlátozások nélkül leírni, miközben a tőszótár mérete is 100 ezer tőre csökkent a képzett alakok elhagyásával.

A bővítésnek köszönhetően a prefixumok már nem csak a tövekhez, hanem az affixumokhoz is köthetők, vagyis alkalmazásuk feltételes lehet: például a mellékneveknél akkor fogadható el a *leg* prefixum, ha a tő már a megfelelő toldalékkal rendelkezik (**legpiros-legpirosabb*). További hasonló problémák, amelyek megoldása a többszörös affixumleválasztás „melléktermékeként” vált egységessé, és pusztán a nyelvi erőforrással megoldhatóvá: egyszerű számnév és képzős főnév szóösszetétele (**ötszoba-ötszobás*), tulajdonnevekből képzett kis kezdőbetűs szavak **budapest-budapesti*, igekötők és névszóból képzett igék (**elkutyá-elkutyásodik*), többszörös prefixumok (**legmegoldhatatlan-legmegoldhatatlanabb*).

Nemcsak a morfológiai elemzés, hanem a helyesírás-ellenőrzés is javult a *homonimák* kezelésével. Az Ispellnél a *megvárából*, *előle* alakok is helyesek voltak a *vár* és *öl* igék igekötőinek és a *vár* és *öl* névszók ragjainak egyidejű megjelenése miatt. A Hunspell esetében megvan a lehetőség a homonimák elkülönítésére és így az igei prefixumok és a névszói szuffixumok szétválasztására.

A következő Hunmorph nyelvi erőforrás összefoglalja a Hunspell és a Hunmorph legfontosabb képességeit. A tőszótár a homonimák megadását a lehető legegyszerűbben, a tövek többszörös megadásának (vagyis halmaz helyett multihalmaz használatának) lehetőségével oldja meg. A második mező tartalmazza a tő morfológiai kódját:

```
drink/RQ <VERB>
drink/S <NOUN>
```

Az affixumdefinícióban két új mező jelenik meg. A hatodik mező a morfológiai kód, a hetedik pedig a „folytatási osztály”, ami az affixum megléte esetén alkalmazható további affixumok kapcsolóit tartalmazza:

```
PREFIX P 0 un . [un]
SUFFIX S 0 s . <NOUN<PLUR>>
SUFFIX Q 0 s . <VERB<PERS<3>>>
SUFFIX R 0 able . [able] PS
```

A példában az utolsó affixum rendelkezik ilyen folytatási osztállyal. Jelentése, hogy az a bizonyos *able* szuffixum együtt járhat a P osztályban található illeszkedő prefixumok valamelyikével, illetve folytatódhat az S osztályban található illeszkedő szuffixumok valamelyikével. Így helyes szó a nyelvi erőforrás szerint az *undrinkable* és az *undrinkables*, viszont az **undrink* már nem.

A Hunspell, Hunmorph affixumállományának formátuma nem szab korlátot a két-tónél nagyobb szuffixumleválasztásnak, de a harmadik szuffixum leválasztását a program jelenleg nem értelmezi, mivel a jelenlegi architektúra nagyobb mérvű változtatására volna szükség. Ezt a korlátot a fejlesztés alatt álló Hunlex előfeldolgozó oldja meg, ami egy általános morfológiai leírásból a többszörös affixumokat is értelmezve automatikusan állítja elő a Hunmorph számára szükséges nyelvi erőforrást.

4. Teljes elemzés

A morfológiai elemző elkészítéséhez nélkülözhetetlen volt, hogy az elemző *teljes elemzést* végezzen, vagyis ne álljon meg az első elemzésnél (ellentétben a helyesírás-ellenőrzővel, aminek elég az első sikeres találat, hogy a szó helyességét megállapítsa):

```
> drink
drink<VERB>
drink<NOUN>
> drinks
drink<VERB<PERS<3>>>
drink<NOUN<PLUR>>
```

A teljes elemzés bevezetése egy helyen lett korlátozva alapértelmezés szerint: a szavak összetett szóként való elemzését csak abban az esetben végzi el a program, ha nem akad más egyszerűbb elemzés. (Például a *halász* elemzésénél nem kapjuk meg a helyesírás szerint elfogadható *hal+ász* felbontást, mivel a szónak van más, egyszerűbb elemzése.) Ha egy-egy szónál mégis szeretnénk összetett szavas elemzést is (mint a *karóra* esetében: *karó-ra* és *kar+óra*), akkor az összetett szót külön fel kell vennünk a tőszótárba.

A Hunmorph morfológiai elemző kimenetének szintaxisa elsősorban a nyelvi erőforrástól függ.

5. Összetett szavak kezelése

A nyelvfüggetlen Hunspell keretrendszer kialakítása a magyar nyelv morfológiai komplexitása és az ortográfiai sajátosságai következtében komoly feladatnak bizonyult. Számos bővítés és új programparaméter került bevezetésre. A kettős affixumleválasztáson, a homonimák kezelésén és a teljes elemzésen kívül az összetett szavak kezelését érdemes kiemelni, ahol

- megadható, hogy mely szavak szerepelhetnek szóösszetételben, akár csak az összetett szó első, vagy utolsó tagjaként,
- megadható a 6–3-as szabály (*kerékpárjavítással*, de *kerékpár-javítási*),
- megadható, hogy mely affixumok megléte esetén szerepelhetnek, illetve nem szerepelhetnek szóösszetételekben a képzővel ellátott szavak.

Erőfeszítéseink ellenére a nyelvfüggetlen összetettség-kezelés nem, vagy csak igen körülményesen valósítható meg pusztán a nyelvi erőforráson keresztül. Ezért a forráskódban elkülönítésre kerül az összetett szavak felismerését végző osztály, lehetővé téve ennek különböző nyelvekhez, és különböző célokhoz (helyesírás-ellenőrzés és morfológiai ellenőrzés) adaptált változatainak elkészítését.

6. Architektúra

A Hunspell, Hunmorph programkönyvtár C++ nyelven íródott. A felhasznált standard programfejlesztési segédeszközök (Make, jelenleg Automake, Autoconf) biztosítja a program széles körű hordozhatóságát, felhasználhatóságát és bővíthetőségét.

A Hunspell helyesírás-ellenőrző, a Hunstem tövező, és a Hunmorph morfológiai elemző nyelvi erőforrásai (vagyis a szótárak) különböznek a különböző felhasználási területnek megfelelően: a morfológiai elemzőtől nagyobb rugalmasságot várunk el az akadémiai helyesírási szabályzat követésében, mint egy helyesírás-ellenőrzőtől (például hasznos, ha elemzi a gyakori **izület*, **lőjünk*, vagy **adatbáziskezelő* szóalakokat is). Egy indexelésre használt szótövezőtől pedig elvárható, hogy a képzőket ne, vagy csak mértékkel vágja le a tövezésre kerülő szóalakról (például a *Sorstalanságról* töve ne a *sors* legyen).

A HunLex előfeldolgozó rendszerrel rugalmasan beállítható, hogy az adott nyelvi erőforrásban hol helyezkedjen el a tövezés szintje, vagy engedélyezhetünk „egészen” szubsztenderd toldalékolást is, ha a morfológiai elemzés úgy kívánja.

7. Összefoglalás

Nyitott forráskódú programunk és programkönyvtárunk egyedülálló lehetőséget teremt magyar nyelvű szövegek elemzésére önállóan, és valamely nagyobb alkalmazás részeként is. A rendszer igény szerint bővíthető, módosítható, de a nyitott forráskódú programok felhasználásának köszönhetően is már több, mint 40 nyelvhez használható helyesírás-ellenőrzőként és részleges szótövezőként. A továbbiakban különösen az Ispell technológia által mostohán kezelt ragasztó nyelvek köréből számítunk nagy érdeklődésre, és új, illetve javított szótárak megjelenésére.

Rendszerünk a <http://www.szoszablya.hu/> oldalon érhető el. Felhasználóink és fejlesztőink részére hibaadatbázist és levelezőlistát is üzemeltetünk, kihasználva a nyitott forráskódú fejlesztésben résztvevők nagyfokú segítőkészségét.

Köszönetnyilvánítás

A Szószablya projekt az Informatikai és Hírközlési Minisztérium ITEM pályázatán nyert támogatással vált lehetővé. A program erőforrásait jelentős részben a MATÁV Rt. és az Axelero Internet biztosította. Fejlesztéseinkhez a Szószablya fejlesztőkön kívül a Magyar Ispell és a Szószablya levelezőlisták olvasóinak észrevételei is hozzájárultak. Segítségüket mindannyiuknak köszönjük.

Hivatkozások

- [Dömölki1967] B. Dömölki. 1967. Algorithms for the recognition of properties of sequences of symbols. *USSR Computational & Mathematical Physics*, 5(1):101–130. Pergamon Press, Oxford.
- [Earnest1963] L. Earnest. 1963. "Machine Recognition of Cursive Writing," Information Processing 62, (Proc. IFIP Congress 1962, Munich), North-Holland, Amsterdam, 1963.
- [1] P. Halácsy, A. Kornai, L. Németh, A. Rung, I. Szakadát and V. Trón 2003. Magyar Számítógépes Nyelvészeti Konferencia, Szeged, 211–217.
- [2] P. Halácsy, A. Kornai, L. Németh, A. Rung, I. Szakadát and V. Trón 2003. Creating open language resources for Hungarian. See LREC Proceedings.
- [Kuenning] Geoff Kuenning. Contributors állomány. Ispell forráskód.
<http://fmg-www.cs.ucla.edu/fmg-members/geoff/ispell.html>
- [Németh2002] L. Németh. 2002. Magyar Ispell. *IV. GNU/Linux Szakmai Konferencia*, 99–107. Linux-felhasználók Magyarországi Egyesülete, Budapest
- [Peterson1980] J. L. Peterson. 1980. *Computer programs for spelling correction: an experiment in program design*, volume 96.
-

Nyílt forráskódú szoftverek a kis- és közép vállalatok piacán

Parragh Szabolcs

Tartalomjegyzék

1. A nyílt forráskódú szoftverek kérdése napjainkban	125
2. A szoftverpiac alakulásáról	126
3. Összefoglalás	129

1. A nyílt forráskódú szoftverek kérdése napjainkban

A következőkben abból a gondolatból indulok ki, hogy a GNU/Linux és egyéb nyílt forrású szoftverek elterjedésének kérdése korunk egyik meghatározó jelensége. És nem is csak az üzleti megközelítésmód, azaz azon emberek számára, akik esetleg üzleti tevékenységük révén is elkötelezettek e szoftverek terjedése mellett. Úgy hiszem, a nyílt forráskód sikere több területen is izgalmas belátásokhoz vezethet.

A legátfogóbb vizsgálódás minden bizonnyal – és itt talán szerencsésebb ezt a kifejezést használni – a szabad szoftverek terjedésének társadalmi vonatkozásait kell érintse. Sajnos jelenleg az akadémiai szinten művelt társadalomfilozófia nem szentel kellő figyelmet a kérdésnek, pedig a szabad szoftver mozgalom sikerének is egyik alapfeltétele, hogy fogalmilag tisztázott, jól artikulált, és a hagyományos társadalomelméleti nyelvhasználattal is megközelíthető alapelvek mentén szerveződhessen. Amennyiben ugyanis a szabad szoftverek elterjedése bizonyos társadalmi reformokat és az információs társadalom új modelljét kínálja, akkor a siker első számú feltétele, hogy a mozgalom alapelvei révén képes legyen a vele jelenleg nem szinkronban érvényesülő, ám meghatározó társadalmi törekvésekkel párbeszédre lépni.

A második a jog tartománya, ahol már kézzel fogható eredményeket is sikerült elérni. A nyílt és/vagy szabad szoftverek további terjedésének számos jogi feltétele van, amelyek közül néhány már megoldottnak látszik – például a szoftverekhez kapcsolódó felelősség kérdése, amely az ingyenesen elérhető programok esetében más jogi formalizálást igényel, mint fizetős társaiknál. Itt komoly előrelépések történtek az elmúlt években szerte a világon, ám sajnos a jogi szabályozás területén is vannak egyelőre még tisztázatlan kérdések, sőt, negatív tendenciákat jelző megoldási javaslatok is, gondoljuk csak az EU-s szoftverszabadalmak kérdésének elhúzódó vitájára. E jogi problémák

vitája minden bizonnyal hosszan elhúzódik még, ám nem árt minél gyakrabban hangsúlyozni: a nyílt forráskódú szoftverek sikerének alapfeltétele, hogy minél hamarabb jogilag is tisztázott környezetben legyenek terjeszthetők.

Végül harmadikként említem azt a területet, amely az előadásom központi témája lesz: a nyílt forráskódú szoftverek elterjedésének üzleti feltételeit. Azzal, hogy csupán harmadikként, azzal jelezni próbálom: a kérdés üzleti vonatkozásait, ha hosszabb távra tekintünk, megelőzik a társadalomfilozófiai és jogi kérdések. Tartós sikerre csak akkor számíthatunk, ha azok is érzékelik a probléma jelentőségét, akik üzleti érdekeltségeik révén maguk nem kötődnek a mozgalom sikeréhez, illetve ha tartós jogi alapokat és a nyílt forráskódú szoftverek melletti széleskörű társadalmi támogatottságot sikerül kialakítanunk.

A továbbiakban tehát ezeknek a szoftvereknek, így például a GNU/Linuxnak az elterjedéséhez szükséges azon feltételekkel és lehetőségekkel foglalkozom, amelyek az üzleti élet tartományába tartoznak, és ott szabályozhatók. Ezt megelőzően azonban szeretnék egy tételt javasolni, melyet a kérdés tárgyalásához elengedhetetlennek tartok. E tétel szerint:

(1) A nyílt forráskódú és/vagy szabad szoftverek belátható időn belül együtt kell létezzenek és működjenek zárt forráskódú, fizetős társaikkal.

Ezzel a tétellel nem azt állítom, hogy egy kizárólag nyílt forráskódú szoftverekre alapozott rendszer önmagában működésképtelen lenne, hanem azt, hogy társadalmi lép-téssel mérve, a nyílt és zárt fejlesztés tartós együttélésére érdemes berendezkednünk. Ennek belátáshoz két érvet szeretnék megemlíteni. Egyrészt: a nyílt forrású szoftverek fejlesztése jelenleg elképzelhetetlen lenne azon vállalatok nélkül, ahol zárt forráskód-dal, üzleti alapon fejlesztenek szoftvereket, hiszen a GPL és egyéb szabad licencek alatt fejlesztő programozók több tízezres táborának nagy része számára a megélhetést ezek a vállalatok biztosítják, ahol az említett programozók kereskedelmi szoftverek fejlesztésében vesznek részt. A másik érv, amit érdemes szem előtt tartanunk: a nyílt forráskódú rendszerek mai alkalmazása jelentős részben zárt, kereskedelmi szoftverekkel való együttélésen alapul – ahol egyébként az együttélési képességekről e szoftverek sokkal jobb bizonyítványt kapnak „fizetős” társaiknál. Gondoljunk csak azokra a vállalatokra, ahol a kereskedelmi szoftverekkel üzemeltetett kliensgépek nyílt forráskódú alkalmazásokkal futtatott fájl- vagy adatbázisszerverekhez kapcsolódnak.

Az említett tétel mellett az előadásom másik központi feltevése, hogy a kis- és középvállalatok (a KKV piac) növekvő érdeklődése az integrált vállalatirányítási rendszerek iránt, a nyílt forráskódú szoftverek terjedésének új lendületet adhat. Ezzel ugyanis e szoftverek a desktop piacon is erős pozíciókat szerezhetnek, ahol eddig kétség kívül a kereskedelmi szoftverek abszolút hegemoniája volt a jellemző. E lehetőséget azonban csak akkor sikerül majd kihasználni, ha a növekvő piaci igényeket fejlett, jól kialakított üzleti stratégiákkal, pénzügyi és technológiai modellekkel próbáljuk kielégíteni. A következőkben e modellek kialakítására vonatkozóan próbálok meg néhány javaslatot megfogalmazni.

2. A szoftverpiac alakulásáról

Az előadás tulajdonképpeni kérdésére rátérve, had kezdjem két széles körben ismert gondolat felidézésével. A Bill Gates és a Microsoft világraszóló üzleti sikerét kommentáló és magyarázni próbáló történetek közül kettő valóban megvilágító erejű. Az

első szerint a redmondi vállalat vezetőjének zsenialitása abban állt, hogy rájött, hogy időnként le kell csapni a biztosítékot a fejlesztők gépei alatt, és akkor azt lehet mondani: ami eddig kész lett, az lesz az aktuális verzió. A másik, immár mélyebb belátásokhoz vezető értelmezés szerint, a Microsoft sikerének titka, hogy az egész világgal el tudták hitetni, a számítógép funkcionális eszköz: olyan gép, amely a kávéfőző és a varrógép mellé állítva e-mail, szövegszerkesztő vagy egyéb konkrét feladatok ellátására szolgáló gépként üzemelhet.

Mi, akik szoftverfejlesztő cégeknél dolgozunk, mindkét problémát jól ismerjük és értjük. Mégis úgy gondolom, hogy a két gondolat sugallta jelenségek: a verzióköz-pontú fejlesztés illetve a számítógépet funkcionális eszközzé redukáló szemléletmód alapvetően ellentétes a nyílt forráskódú és/vagy szabad szoftverek fejlesztésének, és az e mögött álló mozgalomnak a filozófiájával.

A következőkben amellet próbálok majd érvelni, hogy a szívünknek kedves mozgalom sikeréhez a két alapelv közül az egyiknek az erősítésére, a másiknak viszont a visszaszorítására van szükség. Ennek belátásához vessünk először egy pillantást a vállalati szoftverek piacának közelmúltjára.

Az integrált vállalati rendszerek felhasználása egészen a közelmúltig csak a nagyvállalatok számára volt természetes és szükséges. Ennek oka egyrészt az, hogy az ilyen rendszerek fejlesztési és beüzemelési költségeit kisebb vállalatok, lévén sokkal beruházás-érzékenyebbek, képtelenek voltak finanszírozni. A másik ok pedig, hogy a kisebb cégek üzleti logikáját az esetek többségében képes volt számos általános célokra fejlesztett alkalmazás (pl. egy táblázatkezelő) leképezni – még ha ez a hatékonyság rovására is ment –, és így az átállás integrált vállalati rendszerekre nem volt létkérdés. A kis- és középvállalatok piacán ezért még ma is ilyen általános célú alkalmazások, illetve egyes konkrét feladatokra fejlesztett (pl. külön számlázó vagy készletnyilvántartó) szoftverek kerülnek leginkább értékesítésre.

Az elmúlt években azonban az informatika fejlődése révén elérhetővé váltak olyan fejlesztői alkalmazások és környezetek (mint pl. a szabadon elérhető a J2EE platform, az EJB technológia illetve a JBoss alkalmazáserver), amelyek segítségével hatékonyan és olcsóbban fejleszthetők összetett, moduláris felépítésű vállalati szoftverek. Ez már rövid távon is az árak csökkenését eredményezi, amely tendencia a piac növekedésével tovább erősödhet. E fejlődéssel párhuzamosan egy fontos lélektani változás is elkezdődött: a vállalati felhasználók és cégvezetők, mint gyakorló számítógép használók az élet számos területén, egyre jobban elfogadják és megértik azt, hogy az üzleti folyamatok informatikai támogatása nagyban elősegítheti cégük tevékenységének hatékonyságát, így a vállalkozás sikerét. Ez korántsem evidens belátás, és valószínűleg e lélektani átállás nagyobbik szakasza még továbbra is előttünk van.

Az azonban már látható, hogy a kis- és középvállalatok piacán megjelent, és egyre nő az igény a vállalatirányító rendszerek iránt. Hogy hogyan vezethet ez a nyílt forráskódú szoftverek elterjedésének fokozódásához, rögtön láthatóvá válik, de előtte fel kell hívnunk a figyelmet két, a KKV szoftverpiacot meghatározó jellegzetességre.

Egyrészt, a kis- és középvállalatok nagy része ma sem képes egy önálló, teljesen saját célokra fejlesztett rendszer kialakítását finanszírozni. Ezért a fejlesztő cégek általános célú, moduláris alkalmazáscsomagokat fejlesztenek. A potenciális megrendelők között azonban – még az azonos profilú vállalkozások esetében is – kisebb-nagyobb eltérések lesznek a cégek üzleti logikájában. Ezért egy egyszeri fejlesztés csak nagy szerencsével lesz képes akár két cég igényeit is kielégíteni, mindenhol egy kicsit más megközelítésmód, más adatstruktúra vagy éppen más megjelenítés szükséges. Ez a helyzet igazán rémisztően hangzik a legtöbb fejlesztő cég számára.

A KKV piac másik jellegzetessége, hogyha bevezetésre is kerül egy vállalatirányí-

tási rendszer, a legtöbb esetben nem (vagy legalábbis nem elsőre) lesz szükség a teljes üzleti logika leképezésére és informatikai támogatására. A legtöbb megrendelés kihagy a folyamatokból valamit, például könyvelést, banki kommunikációt stb. mert erre már rendelkezésre áll valamilyen alkalmazás.

Mi következik mindebből? Úgy gondolom ez az a pont, ahol beláthatjuk, a nyílt forráskódú szoftverek fejlesztési modellje az, ami ma a leghatékonyabban tud illeszkedni a kis- és középvállalatok piacának igényeihez. Ezen a piacon ugyanis olyan szoftverekre van szükség, amelyek:

- általános megoldásokra alapozva ugyan, de mindig az egyedi igényekhez illesztett alkalmazásokat kínálnak,
- első átadásával a fejlesztési folyamat nem zárul le, hanem folyamatosan, akár havi rendszerességgel újabb fejlesztések és kisebb átalakítások révén bővíthetők,
- értékesítése nem verziókra alapozott dobozolt szoftverértékesítés, hanem szolgáltatás központú, amely szolgáltatás már eleve tartalmaz a karbantartáson túl fejlesztési feladatokat is.

Amit a nyílt forráskódú szoftverek fejlesztésével kapcsolatban, a verziókhoz kötött fejlesztéssel szembeni ellenérzéseként említettem, pontosan az e három pontban megfogalmazott igények megfelelője a fejlesztői oldalon. A nyílt forráskódú szoftverek, mivel az értékesítés logikája nem kényszeríti ki a verziók kiadásának ritkítását, hagyományosan sokkal közelebb áll a folyamatos fejlesztés modelljéhez. Így a legfrissebb verzió adott esetben akár hetente is változhat, bővíthet.

Másrészt, a szabadon elérhető szoftverek már a maguk piaci jelenlétével is erősítik azt a tendenciát – amely a következő években számos elemző szerint jelentősen fokozódni fog – hogy a szoftveres megoldások értékesítése egyre jobban a szolgáltatási struktúrák felé fog elmozdulni. Azaz: a kínált alkalmazás nem egy egyedi termék lesz, hanem egy informatikai megoldáscsomag része, amelynek célja, hogy a megrendelő üzleti tevékenységének informatikai támogatását megoldja. Így egy ilyen csomag adott esetben egyszerre tartalmazhat hardver- és szoftverelemeket csakúgy, mint technikai tanácsadást és a fejlesztői kapacitás rendelkezésre állását – és ezen elemek között nem lesz jelentős értékkülönbség.

Az anyagi vonatkozások tekintetében hasonlóan kedvezőnek látszik a helyzet. A kis- és középvállalatok eleve beruházás-érzékenyebbek, és nem tudnak 10–20 millió forintos rendszerek vásárlásába befektetni. Számukra sokkal kedvezőbb lehet az az üzleti modell, amely nem egyszeri, nagy összegű beruházást, hanem rendszeres, de kisebb, kezelhetőbb költséget jelent a cég informatikai hátterének teljes fenntartására.

Mindez természetesen csupán olyan előrejelzés, amely néhol bizonytalan feltevéseken alapul. A céloom azonban nem erős állítások megfogalmazása, hanem tendenciák és lehetőségek felmutatása. És e lehetőségek megragadásának első számú feltétele, hogy időben észrevegyük, és felkészülten várjuk őket.

Itt térnék rá arra, amit a nyílt forráskódú szoftveres mozgalmak másik alapelveként említettem, s amelyet véleményem szerint – legalábbis részben – fel kell adni egy időre. E szerint az alapelv szerint a számítógép nem funkcionális gép, hanem olyan általános eszköz amelynek segítségével soha nem látott szélességben és mélységben, szinte minden területen képesek leszünk az emberi megismerés és megértés, illetve az emberi tevékeny gyakorlat tartományát kiterjeszteni. Magam is egyetértek ezzel, és elfogadom, sőt vallom Neal Stephenson érveit, aki híres, „In the Beginning... Was

the Command Line” című írásában részletesen megmutatja, hogyan vált a számítógépfelhasználásnak ez a hamis mítosza világszerte elfogadottá és egyes vállalatok üzleti sikerének zálogává.

Mindamellett úgy gondolom, az informatika jelenlegi üzleti felhasználása nem teszi szükségessé, hogy ezt a mítoszt mindenképp elvessük. Az üzleti folyamatok modellezésében és segítésében a legtöbb végfelhasználó szintjén a számítógép valóban csupán egyszerű munkagép, amelynek segítségével ő maga számláz, raktárkészletet kezel, levelet ír vagy e-mailt küld – és sokszor csak egyiket teszi végig a munkája során. Az pedig, hogy a gép, amin dolgozik, valójában többre is alkalmas – kicsit megfordítva Stephenson érvelését – az lehet csupán annak az eredménye, hogy jelenleg a hardveriparban nem rentábilis ilyen szűk keresztmetszetű céleszközök fejlesztése, csak általános célú számítógépeké.

Hogy mi is következik ebből? Úgy vélem, támogatni kell azokat a tendenciákat, amelyek jelenleg már dominánsan jelen vannak a nyílt forrású, szabadszoftveres fejlesztésekben, ti. a felhasználóbarátság, a grafikus megjelenítés és kezelőfelületek funkcionális tagolásának tendenciáit. Ez önmagában nem nagy felismerés, és szemmel láthatóan megoldható oly módon, hogy ne menjen a számítástechnikai eszközök valós, a lehetőségeket maximálisan kihasználó alkalmazhatóságának rovására. Úgy vélem, a szoftverek vállalati felhasználásában a funkcionális, felhasználás-központú fejlesztésnek és tervezésnek jelenleg nem látszik alternatívája. Ezt azért érdemes megemlíteni, mert az a kultúra, ahonnan a nyílt forráskódú és szabad szoftverek fejlesztésének a mozgalma ered, a hacker kultúra, alapvetően ellentétes filozófiát vall ezzel. Én mégis azt hiszem, az üzleti felhasználás igazából azt tudja majd megmutatni – és ez lesz a számítógépeket funkcionális eszközzé redukáló megközelítésmód igazi cáfolata –, hogy a két filozófia az alkalmazás területén nem kerül feltétlenül összeütközésbe egymással, sőt, hatékonyan ki is egészíthetik egymást.

3. Összefoglalás

Úgy vélem, a szoftverpiac átalakulása napjainkban – amely jelenti egyrészt a kis- és középvállalatok növekvő igényét az összetettebb vállalati alkalmazások iránt, illetve a piac lassú, de tartósan ígérkező elmozdulását a szolgáltatás-központú értékesítés irányába – új távlatokat nyit a nyílt forráskódú és/vagy szabad szoftverek elterjedése számára. Ezeknek a lehetőségeknek a kihasználásához azonban több dologra van szükség:

- Egyrészt szükséges a fejlesztő cégek részéről olyan *üzleti modellek kialakítása*, amelyek az említett piaci szegmens részére elfogadható megoldásokat kínálnak. Erre például egy a szolgáltatásért fizetendő havidíjakra, és csak kisebb összegű kezdeti beruházásra alapozott konstrukció látszik megfelelőnek; persze a részletek minden esetben kicsit más megoldást sugallnak majd.
- Ennek sikeréhez szükséges még a *közgondolkodás további átalakulása*, hogy a vállalatvezetők megértsék és belássák, hogyan képes egy ilyen informatikai szolgáltatáscsomag a cégük működésének hatékonyságát fokozni.
- Szükséges olyan *retorika, olyan bemutatóanyagok és portfóliók kialakítása*, amelyek ebben a belátásban segíthetik a cégvezetőket. Ez persze a fejlesztőcégek elemi érdeke is, így nem hiszem, hogy ebben ne lenne gyors az előrehaladás.

- Szükséges a ma még sajnos elterjedt, a nyílt forráskódú szoftverekkel kapcsolatos *hamis mítoszok lerombolása*, melyek szerint ezek alkalmatlanok üzleti felhasználásra – legalábbis desktop környezetben. Ez valószínűleg csupán idő kérdése, és sikeres is lesz, figyelve a kereskedelmi szoftverek árszintjének alakulását. Mégis úgy gondolom, ezen a területen még sok feladat hárul a nyílt forráskódú szoftverek mozgalmát támogató civil szervezetekre, illetve a nagyobb vállalkozásokra egyaránt, elsősorban a Linux disztribúciók forgalmazóira.
- Belátható időn belül szükséges egyes, jelenleg szabad szoftverként még elérhetetlen *üzleti alkalmazások kifejlesztése*. Gondolok itt például a banki kommunikációs, vagy az adóbevallást segítő szoftverekre. Ehhez természetesen szükség van az érintett állami szervezetek illetve nagyvállalatok, pl. bankok támogatásának megnyerésére.
- Fenn kell továbbá tartani azt a magas színvonalat, amelyen jelenleg a nyílt forráskódú alkalmazások kereskedelmi társaikkal *együtműködni képesek*. Véleményem szerint hosszú távon is érdemes a nyílt és zárt forrású rendszerek közötti együttműködésre berendezkednünk.
- Szükség van arra is, hogy a kereskedelmi szoftverek mögött álló vállalatok ne tudják szabadalmi vagy más jogi eszközökkel ezt, a rendszerek közötti kommunikációt lehetetlenné tenni.

Végezetül még egy gondolat a KKV piac informatikai beruházásainak állami támogatásával kapcsolatban. Mint az ismert, az elmúlt években több állami és EU-s pályázat került kiírásra, amelyek célja a kis- és középvállalatok versenyképességének megőrzése volt, informatikai rendszereik fejlesztése révén. Sajnos általános tapasztalat, hogy ezek a pályázatok nem megfelelően, nem éppen a piaci szegmens számára lettek kiírva.

A legtöbb pályázat minimum 10 milliós támogatási limitje, amely a teljes beruházásra számított 50%-os önrésszel 20 millió lesz, messze meghaladja a kis- és középvállalatok informatikai beruházásokra fordítható kereteit. Minden fejlesztő cég tudja, hogy nincs az a vállalat, amelynek a részére ne lehetne 20 millió forintos rendszert fejleszteni. Mindenki tudja azt, hogy nagyon sok esetben nincs szükség ekkora beruházásra, egy ennek negyed-ötödéből kialakított rendszer már nagyban segíteni tudja az üzleti folyamatokat. És már csak ezért is kizárt, hogy egy kis- vagy középvállalat 20 millió forintot vállalati rendszer fejlesztésére beruházzon.

Úgy vélem, hogy amikor állami szinten is felvállalt törekvés a hazai informatikai rendszerek fejlesztése, és általában is, egy jól szervezett információs társadalom kialakítása a cél, akkor komoly kormányzati felelősség, hogy olyan támogatási struktúra jöjjön létre, amely hatékonyabban és jóval szélesebb körben képes a hazai kis- és középvállalatokat segíteni.

Hitelesítési lehetőségek és eszközök egy vállalati információs rendszerben

Scheidler Balázs

Tartalomjegyzék

1. Bevezetés	132
2. Hitelesítés 1x1	132
2.1. Jelszó jellegű hitelesítések	132
2.2. Nem jelszó jellegű hitelesítések	133
2.3. Hibrid rendszerek	133
3. Authentikációs keretrendszerek	133
3.1. PAM, Pluggable Authentication Modules	133
3.2. BSD auth	133
3.3. SASL, Simple Authentication and Security Layer	134
4. Authentikációs protokollok	134
5. Authentikáció a tűzfalon	134
5.1. SOCKS	134
5.2. TLS/IPSec VPN	135
5.3. Egyedi megoldások	135

1. Bevezetés

Ma már teljesen természetes, hogy a számítógépünkkel hálózatban dolgozunk, adatfájljainkat, dokumentumainkat nem a helyi-, hanem egy dedikált szervergép háttértárolóján mentjük el. Amikor elérünk egy hálózati szolgáltatást, a kiszolgáló ellenőrzi, hogy rendelkezünk-e megfelelő jogosultsággal. Ebben az ellenőrzésben kap fontos szerepet a hitelesítés és annak módja, hiszen ha a hitelesítés túl gyenge, akkor az információ nincs megfelelő módon védve, ha pedig túl nehezen használható, akkor az informatikai rendszer használata válik kevésbé hatékonnyá.

2. Hitelesítés 1x1

Mielőtt egy felhasználó egy hálózati szolgáltatást vesz igénybe, hitelesítenie kell magát. A hitelesítés alapvető lépései:

- Azonosítás (identification): a felhasználó bemutatkozik, megadja a számítógép által értelmezhető, egyértelműen azonosító nevét (felhasználói név)
- Hitelesítés (authentication): a felhasználó valamilyen módon bizonyítja, hogy az azonosító mögött valóban az igazi felhasználó van. A bizonyítás alapja mindig valami:
 - amit a felhasználó tud (pl. jelszó), vagy
 - ami a felhasználó elválaszthatatlan része (biometrikus azonosítás),
 - vagy amivel a felhasználó rendelkezik (valamilyen token).
- Engedélyezés (authorization): a felhasználó neve illetve egyéb tulajdonságai (pl. csoport-tagság vagy besorolási szint) alapján eldől, hogy egy adott objektum számára elérhető-e.

2.1. Jelszó jellegű hitelesítések

A fenti áttekintésből látható, hogy hitelesíteni számos különböző módszerrel lehet. A legegyszerűbb és leggyakrabban használt a jelszó alapú azonosítás, amikor a felhasználó a jelszó ismeretével bizonyítja kilétét.

A jelszavakat a számítógép általában egyirányú kódolással tárolja, azok kiolvasása még az adminisztrátor számára sem egyszerű. A számítógép összesen annyit képes eldönteni, hogy a felhasználó által beírt jelszó jó, vagy nem jó.

A jelszó egyszerűen megvalósítható és használható, hátránya viszont, hogy bármennyiszer, változatlan formában felhasználható.

Az egyszer használható jelszavak, jelszó listák nyújtanak megoldást erre a problémára. Ilyenkor a felhasználó nem egy, hanem egy egész sorozat jelszót kap, melynek minden elemét pontosan egyszer használhatja hitelesítésre. Ilyen azonosítási rendszer például az S/Key vagy újabb nevén OTP (One-Time-Password, RFC1938).

Vannak olyan azonosító eljárások, amik jelszó jellegűek (tehát a felhasználónak be kell gépelnie egy karakter-sorozatot), de azt egy hardver eszközzel, ún. tokennel kell kiszámolni, a számítógép által feltett kérdésre válaszul. (challenge-response).

Minden jelszó jellegű azonosításról elmondható, hogy az azonosító adat viszonylag alacsony entrópiájú, egy jó jelszó kb. 40-45 bitnyi adatot tartalmaz, viszont ilyenkor már a felhasználóra jelentős teher hárul. Egy jó jelszó számokat, kis és nagy betűket, valamint írásjeleket is tartalmaz: "qo3C";Zg".

2.2. Nem jelszó jellegű hitelesítések

Azok a módszerek tartoznak ide, amikor a felhasználó nem közvetlenül gépeli be az őt azonosító karaktersorozatot, hanem a számítógépe, illetve a számítógéphez csatolt egyéb perifériák az ő parancsára, automatikusan végzik a hitelesítést. Ezen módszerek előnye, hogy az azonosító információ akár jóval több is lehet, mint az előzőleg említett 40-45 bit.

Ide tartoznak a digitális aláírást alkalmazó rendszerek, melyek a felhasználó titkos kulcsával írnak alá egy random bájtsorozatot, amit a két azonosító fél együtt határoz meg.

Hasonló módon ide tartozik a kerberos hitelesítési rendszer, amikor a felhasználó jelszóval azonosítja magát a központhoz (KDC), majd a szolgáltatások felé már csak az itt kapott ticketet használja fel.

2.3. Hibrid rendszerek

A fenti hitelesítő eljárások kombinálásával további lehetőségekhez jutunk, például:

- az X.509 tanúsítványunk titkos kulcsához biometrikus azonosítás útján férünk hozzá.
- a Kerberos TGT ticket-et akkor kapjuk meg, ha ezzel a titkos kulccsal azonosítjuk magunkat.
- ezek után minden szolgáltatáshoz a ticket segítségével azonosítjuk magunkat.

3. Authentikációs keretrendszerek

Minden azonosítás valamilyen eltárolt adat alapján történik. Hogy pontosan mit tárolunk és hogyan az nagyon gyakran az igényelt hitelesítési módszertől függ, ezért a kiszolgálókon futó alkalmazások igyekeznek ezeket a funkciókat egységes keretrendszerekre bízni, melyeket az adminisztrátor szabadon konfigurálhat, akár az alkalmazástól függetlenül.

3.1. PAM, Pluggable Authentication Modules

A PAM egy széles körben elterjedt, portábilis keretrendszer. Viszonylag egyszerű konfigurálni, viszont meglehetősen limitált, csak jelszó jellegű autentikációhoz használható, mivel működési modellje szerint közvetlenül a felhasználóval kommunikál.

Ezek miatt viszonylag nehezen integrálható ettől eltérő modellt alkalmazó programokba, mint például az SSH. Az SSH-PAM integráció jelszavakra, illetve a keyboard-interactive módon keresztül egyéb jelszó jellegű autentikációs módszerekkel használható, viszont nem alkalmazható a kulcs alapú, vagy egyéb nem jelszó jellegű módszerekkel, ezekhez az SSH privát adatbázist (`authorized_keys`) használ.

3.2. BSD auth

Kevésbé portábilis, főleg a BSD változatokon található meg. A PAM implementációban használt `shared` objektum helyett külső programokkal működik, így kevesebb problémát okozhat, mint a programmal egy címtérületen osztozó PAM modul.

A PAM-hoz hasonlóan jelszó jellegű hitelesítéshez használható.

3.3. SASL, Simple Authentication and Security Layer

Ez a keretrendszer alapvetően különbözik a PAM vagy BSD autentikációtól. Egyrészt definiál egy olyan meta-protokollt, mely könnyedén illeszthető az elterjedt protokollokhoz (AUTH parancs) és mely lehetővé teszi az egyszerű jelszó jellegű azonosítások mellett a komplexebb, nem jelszó jellegű azonosításokat is.

A SASL legelterjedtebb implementációja a Cyrus féle SASL2, ami backendként képes használni PAM-ot (jelszó jellegű autentikációkra), LDAP-ot (jelszavak plusz DIGEST-MD5) illetve Kerberos-t (GSSAPI-n keresztül), ezek mellett pedig rendelkezik saját adatbázissal is (sasldb).

A legtöbb internetes protokoll rendelkezik a SASL AUTH kiterjesztésével, és az elterjedt implementációk képesek is használni azt (FTP, SMTP, IMAP, POP3...)

4. Autentikációs protokollok

A felhasználókat azonosító információknak el kell jutniuk az azonosítást végző alkalmazásokhoz, ezért valamiféle protokollra lesz szükségünk. A legegyszerűbb, ha maga az alap protokoll nyújt erre megoldást. Ez azonban nagyon gyakran limitált, egyszerű felhasználó név/jelszó azonosítást tesz lehetővé.

Az alkalmazás protokollja természetesen kiegészíthető, egyrészt SASL támogatással, másrészt – főleg a SASL előtti időkből – egyedi megoldásokkal. Ez utóbbira példa a POP3 protokoll APOP kiegészítése.

Autentikációs információt hordozhat a kapcsolatot titkosító SSL/TLS protokoll, mely a titkosítás kezdetén egyik vagy mindkét felet azonosítja. Elképzelhető, hogy a TLS által nyújtott X.509 azonosítás után, további autentikációra már nincs szükség.

A fenti megoldások csak olyan helyzetben működnek, amikor egy kapcsolatot csak maga az alkalmazás hitelesít. Előfordul azonban olyan, hogy a szerveralkalmazás mellett további autentikációt követelünk meg a hálózati infrastruktúrában. Erre példa a HTTP proxyk beépített proxy autentikációja, ami a szervertől független, a proxyn végrehajtott autentikációt tesz lehetővé. Ez a megoldás sajnos protokoll specifikus, egy hasonló megoldás FTP-re, POP3-ra vagy IMAP-ra általánosan nem, vagy csak nehézkesen működne, mivel egy elterjedt protokollt kell módosítani.

5. Autentikáció a tűzfalon

Ahogy említettem, hitelesítésre szükség lehet a szervertől függetlenül, talán éppen a hálózat határán való átlépéskor. Mivel az elterjedt protokollok nem támogatják a HTTP-hez hasonló proxy módot és autentikációt, ezért más protokollt kell keresni a hitelesítéshez szükséges információk átadásához.

5.1. SOCKS

A SOCKS egy olyan keretrendszer, melyben a kliensek nem közvetlenül kommunikálnak a külvilággal, hanem egy SOCKS szerveren keresztül, mely a szabályainak megfelelő kapcsolatokat közvetíti. Az alkalmazások a SOCKS szerver jelenlétéről nagyon gyakran nem is tudnak, számukra ez transzparens.

Ilyen módon a SOCKS közvetlenül használható transzparens, alkalmazás-szintű tűzfalazáshoz, azaz közvetlen alternatívája lehetne a transzparens proxy alapú tűzfalaknak. Sajnos azonban jelenleg a SOCKS csak a kliens oldalon értelmezhető.

A SOCKS protokoll 5-ös verziója képes hitelesítést megkövetelni a kapcsolat kialakítása előtt, így alkalmazható általános, tűzfalon történő autentikációhoz. (Ezt a módszert alkalmazza például a Microsoft ISA szerver.)

5.2. TLS/IPSec VPN

A tűzfal és a kliens közötti kapcsolat a szerver-tűzfal közti kapcsolattól függetlenül titkosítható, a titkosítás kapcsán megtörtént autentikáció szintén felhasználható a tűzfal szabályrendszerében. (például Zorp Pssl proxy, vagy csomagszűrő szabályok).

5.3. Egyedi megoldások

Az egyes tűzfalak egyedi megoldásokat is nyújtanak az áthaladó forgalom azonosítására, például: CheckPoint session autentikáció, Zorp Satyr, Gauntlet CK-GW.

Ezen megoldások általában támogatják az egyszerű jelszó azonosítástól a token alapú azonosításon (RSA SecurID) keresztül a komplexebb, X.509 alapú azonosítást is, de létezik olyan termék, ami a GSSAPI segítségével Windowsos domain bejelentkezéshez kapcsolódó ticketet is képes ellenőrizni ilyen felállásban.

pkgsrc – az univerzális csomagkezelő

Süveg Gábor

Kivonat

Az előadás célja a NetBSD rendszer programtelepítési rendszerének a pkgsrc-nek a bemutatása. A pkgsrc a NetBSD több, mint 55 architektúrája mellett jelenleg nyolc Unix-szerű operációs rendszeren (AIX, *BSD, Darwin (MacOSX), Linux, Solaris) is elérhető.

Tartalomjegyzék

1. Mi a pkgsrc?	138
2. Miért a pkgsrc ?	138
2.1. Alkalmazások egyszerű és kényelmes telepítése	138
2.2. Egységes felépítés	138
2.3. Függőségek kezelése	138
2.4. Hordozhatóság	138
2.5. Fordítási opciók	139
2.6. Jogok	139
3. pkgsrc-wip	139
4. A pkgsrc használata	139
4.1. Telepítés CVS használatával (ajánlott)	139
4.2. Telepítés ftp használatával	140
4.3. A környezet telepítése	140
4.4. Első lépések	140

1. Mi a pkgsrc?

A pkgsrc (Package Source) eredetileg a NetBSD operációs rendszer alkalmazásainak telepítésére, kezelésére készült. Tervezésekor a NetBSD-ben megszokott hordozhatóság volt az egyik fontos szempont. Így az évek során a NetBSD mellett számos Unix-szerű operációs rendszeren vált elérhetővé. A pkgsrc életképességét is bizonyítja, hogy hasonló módszert alkalmaznak a többi szabad BSD (FreeBSD, OpenBSD) rendszerek ports néven, és a Gentoo Linux létrehozásának egyik célja a pkgsrc megvalósítása, melyet a Gentoóban portage-nak neveznek. A pkgsrc jelenleg több mint 5000 alkalmazást tartalmaz.

2. Miért a pkgsrc ?

Röviden szeretném a pkgsrc előnyeit, fontos tulajdonságait bemutatni.

2.1. Alkalmazások egyszerű és kényelmes telepítése

A pkgsrc – ahogy a neve is mutatja – alkalmazások gyűjteménye. Az alkalmazások telepítése során a program legfrissebb forráskódját, és a szükséges foltokat letölti az Internetről, ellenőrzi a méretüket, a fájlok sértetlenségét, és elkészíti a program futtatható változatát. Természetesen lehetőség van bináris csomagok készítésére, amelynek a telepítése rendkívül egyszerű.

2.2. Egységes felépítés

A telepített programok, dokumentáció, könyvtárak teljesen elválaszthatóak magától az operációs rendszertől. A telepített alkalmazások a /usr/local/pkg alatt találhatóak.

2.3. Függőségek kezelése

A csomagok függőségeit – beleértve a csomag frissítését – a rendszer automatikusan kezeli. A csomagok telepítésekor a rendszer automatikusan telepíti a csomag használatához szükséges alkalmazásokat.

2.4. Hordozhatóság

Ahogy a NetBSD, úgy a pkgsrc tervezésekor is a hordozhatóság volt a legfőbb szempont. Így a pkgsrc is rendkívül könnyen és gyorsan portolható más operációs rendszerre.

Az alábbi táblázat mutatja a támogatott operációs rendszereket és a hivatalos támogatás időpontját.

operációs rendszer	év
NetBSD	1997. augusztus
Solaris	1999. március
Linux	1999. június
Darwin (MacOSX)	2001. október
FreeBSD	2002. november
OpenBSD	2002. november
IRIX	2002. december
BSD/OS	2003. december
AIX	2003. december
Interix (MS Windows Services for UNIX)	2004. március

A fenti platformokon használható a pkgsrc. Így a csomagok kezelésére nem szükséges a különböző operációs rendszerek adta sokszor teljesen eltérő alkalmazások használata, különböző biztonsági figyelmeztetések követése.

2.5. Fordítási opciók

A telepítések előtt a rendszer ellenőrzi az elfogadott szoftver licenceket, programfordítási és optimalizáló paramétereket. A különböző adatok beállítása egyszerűen kezelhető a /etc/mk.cf fájlban.

2.6. Jogok

A teljes pkgsrc (kivéve a programok forráskódjai) szabadon hozzáférhető BSD licenc alatt, így szabadon testreszabható az igények szerint, átvihető bármely – még nem támogatott – Unix-szerű operációs rendszerre. Alkalmazható speciális környezetben is.

3. pkgsrc-wip

A pkgsrc-wip elsősorban azon pkgsrc fejlesztők számára létrejött fejlesztés, melynek tagjai elsősorban nem NetBSD fejlesztők. Célja a pkgsrc csomagok fejlesztése. A pkgsrc-wip jelenleg 900 csomagot tartalmaz.

4. A pkgsrc használata

A pkgsrc használatának részletes bemutatását az előadásomban szeretném megtenni, itt csupán rövid ízelítőt szeretnék adni. A pkgsrc elérhető CVS-ben, vagy letölthető tömörített formában.

4.1. Telepítés CVS használatával (ajánlott)

```
setenv CVSROOT anoncvs@anoncvs.NetBSD.org:/cvsroot
setenv CVS_RSH ssh
cd /usr
cvs checkout -P pkgsrc
```

4.2. Telepítés ftp használatával

```
cd /usr
wget ftp://ftp.netbsd.org/pub/NetBSD/packages/pkgsrc.tar.gz
tar -xvzpf pkgsrc.tar.gz -C /usr
```

4.3. A környezet telepítése

A használatához szükség van a pkgsrc környezet telepítéséhez, ezt csupán a NetBSD tartalmazza, a további rendszereken ezt nekünk kell megtenni. Az alábbi parancsok segítségével:

```
cd /usr/pkgsrc/bootstrap
sh bootstrap
```

4.4. Első lépések

A sikeres telepítés után már használhatjuk a pkgsrc-t. Az alapértelmezett könyvtárak az alábbiak:

```
a pkgsrc mappája: /usr/pkgsrc,
a telepített programok helye: /usr/pkg/,
a telepített programok adatbázisa: /var/db/pkg.
```

Fontos, hogy a telepítés előtt ellenőrizzük, hogy az operációs rendszerünk adatbázisát ne írjuk felül!

A programok telepítéséhez egyszerű példán mutatom be a szükséges lépéseket.

```
cd /usr/pkgsrc/graphics/gimp
bmake install clean
```

A (b)make kiadása után a rendszer letölti a GIMP legfrissebb változatát, ellenőrzi a szükséges programok meglétét, igény szerint telepíti azokat, majd elkészíti a program futtatható változatát.

A pkgsrc részletes használatát az előadásom közben szeretném bemutatni.

Szerverkonszolidáció UML-lel

Tomka Gergely

Kivonat

Egyetemünkön örökké jelen lévő, és egyre erősödő igény, hogy minden karnak, tanszéknek, intézetnek, helyi adminisztrátornak, egyébnek legyen *Saját Szervere*. Ez önmagában nem baj, sőt, pozitívum, hiszen örülünk annak, hogy az informatika betör az ilyen helyekre is, de számtalan problémát vet föl.

Tartalomjegyzék

1. Megoldandó nehézségek	142
2. UML megvalósítás	143
2.1. A host szerver	143
2.2. Az uml	143
2.3. Hálózat	144
3. Karbantartás és ellenőrzés	145
3.1. Fájlrendszer, durva hibák	145
3.2. Alkalmazások és a használat módjának ellenőrzése	145
3.3. Tömeges használat	145
3.4. Sokszorosítás	147
4. Tények és tapasztalatok	147
4.1. Teljesítmény	147
4.2. Megbízhatóság	147
5. Zárszó	148

1. Megoldandó nehézségek

A felsorolt jelenségek minden egyetemi rendszergazda számára ismerősek. Vagy azért, mert tevőlegesen részese az ilyen helyzetek kialakulásának, és nem érti mi a baja ezzel a központi informatikának, vagy mert a központi informatikán dolgozik, és naponta többször is a haját tépi a helyzet alakulása folytán. A felsorolás távolról sem teljes, igyekeztem csak azokat említeni, amelyeket az UML¹ bevezetése megold.

- Egy ilyen szórványszerver működhet olcsó, gagyi, megbízhatatlan hardveren. Ilyenkor lefagy, meg kell nézni mi a baja, kell keríteni bele gagyi hardverelemet, papírfecnikvel kiékelni a cpu ventilátort, poroltót tartani a közelében, és a többi. Nem szerencsés megoldás.
- A szórványszerver lehet „state of the art” is, ami csak első pillantásra előny. Egy komolyabb szerver igen sok áramot fogyaszt, és igen sok meleget termel, ezért úgy a huszadik után már komoly költség a légkondicionálás és a szünetmentesek sem olcsók. Az erkölcsi hatásról nem is beszélve – én fizikailag rosszul vagyok egy telivér IBM szervertől, ha nem csinál semmit.
- A szórványszerver szórványrendszergazdával jár. Ritka a hozzáértő, akire tetszőleges erőforrást rá lehet bízni, és elvből benne sem bízunk igazán. A jó rendszergazda paranoiáját semmi sem korlátozza, a központi rendszergazdát is csak a lustaság.
- Ellenőrzés és kontroll. Minél kényelmesebben és hatékonyabban lehessen információit gyűjteni, elosztani az erőforrásokat, ellenőrizni a felhasználókat, baj esetén megakadályozni az elharapózást. Ez sok kicsi, ismeretlen rootjelszavú rendszernél körülményes. Sőt.

E gondok megoldására, illetve enyhítésére számtalan megoldás létezik, jelen sorok írója ebből csak párat ismer, röviden jellemezzük őket, a teljesség kedvéért:

- „Apacs” jellegű virtual hostok: egyfelől igazán erőforrás takarékos, viszont nem minden szolgáltatás képes ilyenre, és ha a felhasználónak beleszólási jogot akarunk adni a konfigurációba, nincs könnyű dolgunk.
- Chroot jellegű környezetek: erőforrás takarékosak, de megvalósításuk körülményes, és az erőforrások (egyenlőtlen) elosztása körülményes, ha lehetséges egyáltalán.
- User Mode Linux: nem tökéletesen erőforrás takarékos megoldás, de teljes szabadságot ad a szórványrendszergazdának, az ellenőrzést és erőforrás-gazdálkodást gyerekszerűvé teszi, és nem utolsó sorban, nagyon aranyos.

Vegyünk hát egy nagy levegőt, és fussuk át röviden, hogyan kell megvalósítani egy ilyen rendszert.

¹User Mode Linux

2. UML megvalósítás

Két élesen elkülönülő fázisra osztható. A host, az a gép, ahol az uml-ek futnak, és maguk az uml-ek két külön történet, és szerencsére elég kevés közülük van egymáshoz. A továbbiakban simán uml-nek nevezem ezt az entitást, sőt, magyarul fogom ragozni is, mert így kényelmes. Valamint, az eddigi hűvös, józan filozófiai megfontolások helyébe a tőlem megszokott „érdekes” megoldások lépnek, előre is elnézést kérek azért, hogy nem tudok értelmesen programozni, és hogy nem használok netről készen letölthető dolgokat. Én már csak ilyen vagyok...

Az alább részletezett megoldás pár alapelve épül:

- Az uml-től nem várunk nagy teljesítményt.
- A szórványrendszergazda teljes szabadságot élvez az uml-en belül.
- A host szerver gazdája lusta.

2.1. A host szerver

Bármilyen, nekünk szimpatikus Linux rendszer lehet. Pár követelménynek meg kell felelnie, de egyik sem nehezen megvalósítható.

- TUN/TAP legyen a kernelben.
- 2.6.x kernel esetén kell egy egysoros patch, hogy működjön.
- Sok diszk.
- Sok RAM.
- Még diszk. Kell legyen jó nagy tmp könyvtár is.
- Uml-utilities csomag, Debian rendszeren. Máshol más lehet neve.

Jelen történetben ez egy IBM eServer xseries 335, két processzorral, 2 GB rammal, nem túl sok diszkkal. Tárfelületet egy storage rendszer biztosít majd, mikor ez aktuális lesz. Az operációs rendszer Debian GNU/Linux, 2.6.x kernellel. Teljesen és tökéletesen elégedettek vagyunk vele. Mondjuk a kereskedőtől működésképtelenül érkezett, de azért vagyunk, hogy az ilyen problémákat megoldjuk.

2.2. Az uml

2.4.x sorozatú kernelből készül, az uml honlapról letöltött patchel, egyszeri kernelfordítással.

```
# make menuconfig ARCH=um
# make Linux ARCH=um
```

Ezután keletkezik egy linux bináris, ezt igény szerint strip-pel kicsinyítjük le, és másoljuk oda, ahová jól esik. Ez egy futtatható bináris, ami elindít egy Linux kernelt. Haszna természetesen akkor van, ha megadunk neki egy fájlrendszert, amit ő root partícióként kezelhet. Lássunk pár parancssori opciót:

```
#/usr/local/bin/linux ubd0=rock.bin eth0=tuntap,tap6 \
umid=rock uml_dir=/var/lib/uml mem=128M con=pts
```

Az `ubd0` a root partíciót tartalmazó diszk-image. Elkészítése egyszerű, `dd`-vel egy megfelelő méretű image, `losetup`, `mkfs`, `mount`, `debootstrap`, `chroot`, utókonfig. Akinek ez a felsorolás nem mond semmit, forduljon hozzám bizalommal, nekem van egy nem tökéletes, de bőven elegendő 500 Mbyteos woody image-em, nagy örömmel közzéteszem. Bárhogy is csináljuk, az inittabban két dolgot „jó” átállítani. Az `alt-ctrl-del` hatása ne reboot legyen, hanem halt, ugyanis ez az egyetlen jel amit kényelmesen tudunk közölni az `uml`-lel kívülről, és irtsuk ki a `tty`-ket, úgyszincs rá szükségünk.

Az `eth0` adja meg az `uml`-ből `eth0` interfésznek látszó entitás külső megjelenését. Esetünkben ez egy ún. TUN/TAP eszköz, ami megfelelően beállítva olyan, mintha saját hálókártyája lenne a gépnek. Van még pár megoldás, és ez sem olyan egyszerű, később részletesebben is foglalkozom vele.

Az `umid` egy egyedi azonosítót ad az `uml`-nek, az eredetileg járó krix–krax helyett. Ezzel lehet a `ps` kimenetében megkülönböztetni az `uml`-eket, és ez lesz a vezérlőfájlokat tartalmazó könyvtárnak is a neve. Az `uml_dir` opcióval megadott könyvtárban hozzátatik létre a fent említett vezérlőkönyvtár.

A `mem` opciót mindenki kitalálja, szerintem. Ezt amúgy nem lefoglalja `fixen`, hanem a `tmp` könyvtárban hoz létre egy fájlt, és abba írogat. Így ha éppen van `ram` szabadon, akkor ott csücsül az `uml` ramja a `host` gép `cache`-ben, és ekképpen gyors, ha nem, akkor meg nem. Szakértők szerint, ha nem akarunk túl assú ramot, akkor ún. `tmpfs`-sel generált `tmp` könyvtárba helyeztessük ezeket a fájlokat. Ezt elhiszem, hogy igaz, de még nem értem, hogy miért.

A `con` opció adja meg, hogy hol keresse a konzolt. Eredetileg ez egy `xterminál`, de ez nem mutat jól egy szerveren, ezért ezt adjuk meg. Boot közben a `stdoutra` is ír, azt irányítsuk át valahová, a `dev/null`-ban mindig van hely. Aki szereti látni a konzolt, annak sok lehetősége van, élvezze mértékkel.

A leállítás sem bonyolult. Kell hozzá az `uml_mconsole` nevű kis program, és tudnunk kell, hogy hol van a kontroll fájl. Ez a fenti példában így néz ki:

```
#uml_mconsole /var/lib/uml/rock/mconsole
(/var/lib/um) cad
```

Itt rögtön szemléltetem azt is, hogyan kell leállítani a megfelelően kiképzett `uml`-t. Az `mconsole cad` parancsától az `uml` azt hiszi, `alt-ctrl-del`-t nyomtak neki, és elmegy aludni. Tucatnyi más parancs is van, lehet kihúzni/bedugni eszközöket, megállítani az `uml`-t (például mentés készítéséhez), le lehet lőni gondolkodás nélkül (konnektorkitépés szimulálásához).

2.3. Hálózat

Az `uml`-ek széles körű felhasználási lehetőségeihez mérten többféle hálózati interfész is lehet. Én kizárólag egyet használok itt és most, a TUN/TAP becenevűt. Ez némi script-írogatás árán közel tökéletesen szabad felhasználást tesz lehetővé az `uml`-en belülről. Ehhez természetesen jó sok mindent el kell hitetnünk a külvilággal. Például azt, hogy a `host` gépnek nem csak egy címe van, hogy a `host` gép egy hálókártyája több hálókártya, több MAC címmel, és természetesen a `host` gépen belül is el kell jutnia a megfelelő helyre a biteknek.

Be kell állítani egy `tun/tap` eszközt, és ezt a `tapN`-et kell megadnunk az `uml`-nek, mint a hálózat felé eső végét. Ezen felül kell még sok minden, routolni kell az `uml` nyilvános `ip` címére igyekvő csomagokat a `host` gép `tapN` interfésze felé, statikus `arp`-tábla bejegyzést is kell csinálni, stb. Erre nekem van egy scriptem, a függelékek között szerepel majd.

Az uml-ek tudnak csak egymással beszélgetni. Ennek legtöbb lehetőséggel kecsegtető módja az uml_switch nevű kis program. Megfelelő parancssori opcióval ebbe „beledughatjuk” az uml-eket, akik így látják egymást. Az uml_switch viselkedhet hubként és switchként is. Nekem erre most nincs szükségem, de nagyon kellemes kis „rongálható” kabinetet föl lehet építeni egy hostgépen futó sok uml-ből, melyek egy kijelölt speciális tűzfal-uml-en és egy tun/tap eszközön át kapcsolódnak a külvilághoz. Egy fájl egyszerűbb visszamásolni, mint egy tönkrement valós gépet újrainstallálni.

3. Karbantartás és ellenőrzés

3.1. Fájlrendszer, durva hibák

A rendszerünk egy önálló fájlban húzódik meg, amit blokkeszközként kezel. Ezért ha sérül, akkor ugyanúgy kell ellenőrizni is. Losetuppal blokkeszközzé varázsoljuk, fsckval ellenőrizzük. Ha például „rootjelszó-elfelejtés” esete forog fenn, akkor föl is mountoljuk, és kedvünkre hegeszthetjük. Természetesen ilyen műveletek előtt jó leállítani az uml-t. Esetleges betörés esetén hasznos, hogy az uml „standby” állapotba hozható, kimerevíthető. Eképpen tetten érhetjük a behatolót, átvizsgálhatjuk a memóriát, a fájlrendszert, és a behatolónak esélye sincs adatokat törölni, hiszen két órajelciklus között megtorpantunk.

Nagyon szeretem, hogy password recoveryhez nem kell odaszaladni a géphez, álldogálni a hideg és zajos szerverszobában, ne adj isten vészhelyzetben betaxizni éjszaka, hanem bárhonnan el tudom intézni.

3.2. Alkalmazások és a használat módjának ellenőrzése

Az uml minden futó processzhez új thread-et indít a hostgépen, és korrektül ki is tölti a megfelelő táblázatokat, ezért gond nélkül tudjuk követni, hogy a szórványrendszergazda éppen mit futtat.

```
# ps ax
31490 /usr/local/bin/linux (rock) [(tracing thread)]
...
31949 /usr/local/bin/linux (rock) [/sbin/syslogd]
...
6682 /usr/local/bin/linux (rock) [/usr/sbin/portsentry]
```

Látható, hogy ez a user nem érzi magát biztonságban. A lista nem teljes, ezért az nem látható, hogy a user még nem tökéletesen ura a technikának, például eszébe sem jutott az Apache futó processzeinek számát csökkenteni. Sebaj, erre jó az uml. Játszótér.

Természetesen az az egyszerű tény, hogy az uml minden hálózati forgalma átmegy a hostgépen, ráadásul szépen szétosztva uml-enként, roppant kényelmessé teszi a forgalom figyelését is. Sőt, az uml-nek van olyan opciója, hogy naplózza a szerveren belül történő dolgokat is, hogy az ssh feltörésével se kelljen bajlódni. Ezek a megfigyelési módok jogi kérdéseket is fölvetnek – ügyeljünk arra, hogy ártatlan játszadozásunk ne vonjon maga után börtönbüntetést. Ezt legegyszerűbben egy jól megfogalmazott házi-renddel lehet elérni. Nekünk ilyen még nincs.

3.3. Tömeges használat

Kedvenc parancssoros környezetünkről van szó, tehát minden automatizálható, nincsen ez másképp az uml kérdéskörrel sem. Az már rám jellemző betegség, hogy ezek

a scriptek nem oldanak meg minden megoldható problémát, de talán hasznos kiindulópont lehet a scriptírásilag kihívásokkal küszködőnek, és jó mulatság a témát ismerőknek.

A megoldás lelke egy felsorolás, minden uml információival:

```
agymosoda /var/lib/uml 192.168.0.3 192.188.242.118 128M
trinity /var/lib/uml 192.168.0.2 192.188.242.123 128M
able /var/lib/uml 192.168.0.4 192.188.242.119 128M
hok /var/lib/uml 192.168.0.5 192.188.242.116 128M
grable /var/lib/uml 192.168.0.6 192.188.242.115 128M
rock /var/lib/uml 192.168.0.7 192.188.242.114 128M
```

A 192.168.0.x cím az a tun/tap interfész belső címe, routolás végett. Ehhez tartozik indítóscript:

```
#!/bin/bash
grep $1 uml.list | (
# filenev, konyvtar, belso ip, kulso ip, memoria
read FILE DIR IIP OIP MEM
echo $IIP $OIP $DIR $FILE $MEM
echo "uml" $FILE "halozat cfg..."
TAP=`tunctl -b`
echo "ifcfg..."
ifconfig $TAP $IIP up
echo 1 > /proc/sys/net/ipv4/ip_forward
echo "route..."
route add -host $OIP dev $TAP
echo 1 > /proc/sys/net/ipv4/conf/$TAP/proxy_arp
echo "arp..."
arp -Ds $OIP eth0 pub
echo "ok."
echo $TAP > $DIR/$FILE.tap
echo "uml" $FILE "inditas..."
cd $DIR
export TMPDIR=/var/tmp
/home/tomka/linux ubd0=$FILE.bin eth0=tuntap,$TAP \
umid=$FILE uml_dir=$DIR mem=$MEM con=pts \
>/dev/null </dev/null &
echo "ok" )
```

Hát mit is mondjak? Kicsit küzdünk a bash hiányosságaival, de egyszerű mint a faék. Eltesszük a tun/tap eszköz nevét, hogy leállításkor törölhessük. Leállítás:

```
#!/bin/bash
uml_mconsole /var/lib/uml/$1/mconsole cad
echo "türelem"
sleep 10
tunctl -d `cat $1.tap`
rm $1.tap
echo $1, "rest in peace"
```

Aludni kell 10 másodpercet, hogy az összes processz/thread leálljon. Türelem kérése... Emiatt nem egyszerű leállítani a host gépet. Itt két ponton is tetten érhető lustaságom – egyfelől nem keresek megoldást arra, hogy a shutdown hosszabb ideig várjon leállásnál az elhalásra. Másrészt meg mert nem ellenőrzöm, hogy tényleg leálltak-e az uml-ek, csak várok kicsit. Hmmm... Sosem terveztem tökéletesnek lenni. Majd mikor először pórul járok a host gép halomra gyilkolása miatt, majd megcsinálom rendesen, ígérem.

3.4. Sokszorosítás

Egy mindig készenlétben tartott „szűz” rendszerrel ez nem bonyolult feladat:

- `cp default.bin rock.bin`
- Megfelelő méretre hozzuk, a példa kedvéért 1 Gbyte-tal növeljük:
 - `dd if=/dev/zero bs=1024 count=1000000 >> rock.bin`
 - `losetup /dev/loop0 rock.bin`
 - `ext2resize /dev/loop0`
- `mount /dev/loop0/ /mnt`
- Szerkesztendő fájlok:
 - `/etc/hosts`, a hostnév kedvéért
 - `/etc/hostname`, dettó
 - `/etc/network/interfaces`, ip-cím és társai
- `chroot .`, root és user jelszó változtatás
- `rm /etc/ssh/*key*`
- `dpkg-reconfigure ssh`, hogy egyedi ssh kulcsok legyenek
- `exit`
- `umount /mnt`
- `losetup -d /dev/loop0`
- `uml.list` szerkesztése, indítás

Ez nagyjából 4–5 perc, és főnökünk büszke lesz ránk.

4. Tények és tapasztalatok

4.1. Teljesítmény

Egyik szemem sír, a másik meg üveg. Vannak jó hírek, és rossz hírek. Kezdjük a rosszal.

Az uml a diszk I/O teljesítményt nagyjából harmadolja. Ha a hostgépen 100 mbyte/s sebességgel tudunk írni/olvasni, akkor az uml-ben 33 mbyte/sec lesz a maximális sebesség. Ez bizonyos szempontból kellemetlen, de így is gyorsabb, mint egy átlag hálózati kapcsolat. Mindenesetre a tanulság fontos – sok, nagy fájl kiszolgáló szerverre keressünk más megoldást, esetleg nézünk körül az SKA-patchek környékén.

A hálózattal már jobb a helyzet, erős gépen a 100 mbps-t kitöltötte, különösebb cpu-terhelés nélkül. A CPU-felhasználás esetén pedig egyenesen csodálatos a helyzet, a veszteség az UML miatt kevesebb mint 1%.

4.2. Megbízhatóság

Kiváló.

5. Zárszó

A rendszer nekem, és a potenciális felhasználóknak is elnyerte tetszését. Jelenleg ugyan csak hat darab fut, ezek közül az egyik napi több gigabájtos forgalmat bonyolít le (bulik képeivel), egy másikon közepesen komoly PHP/PSQL alkalmazás fut, hiba, zökkenő és fönnakadás nélkül. Tapasztalataim alapján jelen IBM 335 több tucat uml-t is elbír a hátán, ami hatalmas előny most, mikor szerverkonszolidációt végzünk. A különböző blade-szerverek világában elegáns megoldás rámutatni egy 1 unit magas szerkezetre, hogy „ott a szerverfarmunk”.

Egy új uml életbe léptetése villámgyors folyamat, és ezzel hihetetlen mennyiségű piros pontot lehet szerezni minden értelemben. Minőségileg új kapcsolatot jelent a felhasználókkal, ha 10 perccel az igény bejelentése után már egy tökéletes kis szerver zakatol, ami csak az övék. Arról nem is beszélve, hogy a „nagyarcú” számolgotásoknál az uml kétszer is számít – a hostgép gazdája mondhatja lezserül, hogy „én karbantartok egy nagyvasat, persze azon fut vagy 30 szerver”, de a szórványszerver felhasználója is mondhatja, hogy ő bizony karbantart egy szervert, ami kint van az Interneten, bizony. Mindenki jól jár.

Nekem külön öröm, hogy szabad szoftverekkel tudok olyan megoldást, és olyan új lehetőségeket nyújtani, melyeket más eszközzel csak összehasonlíthatatlanul drágábban és sokkal nagyobb energiáfordítással lehet.

Általában, tapasztalataim szerint az User Mode Linux használata itt és most telt kecskéket, és érett, kártevőmentes káposztákat eredményezett. Csak javasolni tudom használatát.

Gameserver üzemeltetés Linux alatt *

Vincze Gábor
Interware Internet Szolgáltató Rt.

<http://www.interware.hu>

Kivonat

Hasonlóan a tőlünk nyugatabbra, északabbra és délebbre elhelyezkedő országokhoz, kis hazánkban is egyre nagyobb iramban terjed az otthonokban is elérhető széles sávú internet-elérés, melynek használatával az internetes játékok lehetősége is megnyílik a felhasználók előtt. Cikkünkben arra keressük a választ, hogy milyen szempontokat kell figyelembe venni egy nagyobb létszámú játékos közösséget kiszolgáló, megfelelő minőséget nyújtó játékszerver indításánál.

Tartalomjegyzék

1. Bevezetés	150
2. Mit szeretnénk – mit lehet?	150
3. Építsünk szervert	151
3.1. Hardver	152
3.2. Szoftver – OS	152
3.3. Mohaa telepítése	152
4. Adminisztráció	152
5. További jótanácsok	153

* A szerzőnek a *Linuxvilág* 2004/11. számában megjelent cikke alapján

1. Bevezetés

Hasonlóan a tőlünk nyugatabbra, északabbra és délebbre elhelyezkedő országokhoz, kis hazánkban is egyre nagyobb iramban terjed az otthonokban is elérhető széles sávú internet elérése. Ez ad táptalajt az on-line játszható játékok rohamos elterjedésének. Egy analóg modemmel még nem volt nagy élmény játszogatni, bár a nyújtott 33.6/56 Kbit/sec-os átviteli sebesség még megfelelő lett volna, a hálózati csomagok a kelleténél lassabban értek célba, így a késleltetés élvezhetetlenné tette a játékot.

Cikkünk témája tehát az, hogy milyen szempontokat kell figyelembe venni egy kiszolgáló beindításánál annak, aki nagyobb létszámú játékos közösséget szeretne „boldoggá tenni” megfelelő minőségű szolgáltatással.

2. Mit szeretnénk – mit lehet?

Mivel rengeteg ilyen játék van, és egyet azért ki kell most választani, így a konkrét „alany” ezúttal a Medal Of Honor Allied Assault nevű, már régebben kiadott, és mára relatíve olcsón beszerezhető szoftver lesz.

Mielőtt kialakítjuk a környezetet, tisztáznunk kell a terveinket:

- Mekkora célközönséget szándékozunk kiszolgálni?
- Mennyi szabad erőforrásunk van erre a célra?
- Ellenőrzött szolgáltatást kívánunk-e nyújtani, vagy nem törődünk azzal, hogy a játékosok mit művelnek a szerveren „játék” címen?
- Milyen egyéb szolgáltatások fognak futni a kiszolgálón? (http, ftp, sql stb.)

Természetesen hangzik, hogy minél több felhasználót szeretnénk kiszolgálni, annál több erőforrásra lesz szükségünk, ez azonban nem lineárisan növekvő hardverigényt jelent.

A játékszerverek terhelhetőségét általában a rajta lévő szabad slotok (kihasználható játékos férőhelyek) szabályozásával adják meg, illetve az egy felhasználóra jutó sáv-szélesség meghatározásával. Ebben a pillanatban el is érkeztünk két meglehetősen sarkalatos problémához, amitől általában az átlagos rendszergazda visszahőköl és gyors ütemben távozik a helyszínről. ☹

Az egyik (nem leküzdhetetlen) akadályunk, hogy a szoftvereket eredetileg nem Linuxra írták, a Linuxos változatok általában nem kellőképpen optimalizáltak, nem használják ki az operációs rendszer hasznos tulajdonságait, hanem önálló kis világként kezelik magukat, akik el sem tudják képzelni magukról, hogy az általuk lefoglalt memóriaterületen tárolt információt pl. több hozzá hasonló folyamat (process) is hasznosítani tudná...

Sajnos ki lehet mondani, hogy a legtöbb – többnyire évekkel ezelőtt kiadott, de ma is kedvelt és nagy felhasználótábornak örvendő – játék sokkal kellemesebben elfut azon az operációs rendszeren, amire eredetileg fejlesztették.

Másik komoly problémánk, hogy nagyon nehezen tesztelhető és mérhető az ilyen jellegű folyamatok hardverigénye. Sajnálatos módon, például a top parancs kiadásával, nem vesszük észre, ha egy ilyen alkalmazásnak nem áll rendelkezésére elegendő mennyiségű rendszererőforrás.

Igen jó példa erre, hogy egy átlagos kiszolgálón (pl. egy Athlon XP 2000+, 512MB memóriával, 7200-as fordulatszámú UDMA/100-as diszkekkel felszerelt masinán) 10

db játékkiszolgálót is el tudunk helyezni és játékosokat behívni rá, anélkül, hogy a gép loadja megnőne. Ennek ellenére a játékosok képtelenek rajta játszani, mert az általuk generált terhelésküszöbök apró késleltetéseket képeznek, amelyek ugyan egy weboldal letöltésénél észre sem vehetők, még ezer egyidejű felhasználónál sem, ebben az esetben viszont a játszhatatlanság sötétlő veszélye fenyeget minket.

Felvetődik a kérdés, hogy mit tudunk tenni egy gyakorlatilag épkezláb módszerekkel nem mérhető folyamathalmaz erőforrásigényének kielégítésére.

3. Építsünk szervert

Először is több íratlan alapszabályt figyelembe véve hozzuk létre a szervert.

- Ha több kiszolgálót üzemeltetünk egy gépen, akkor érdemes a nulláról telepített, optimalizált (erről később még lesz szó) operációs rendszer használata.
- Ne futtassunk más, nagy sávszélesség- vagy processzorigényű alkalmazást a gépen! Egy egyszerű, néhány oldalt kiszolgáló webszerver még nem jelent problémát, egy adatbázis-lekérdezésekkel tarkított, sok felhasználós oldaltömeg viszont nem lesz ínyére a játékszerver(ek) folyamatainak.
- Tapasztalatok alapján, valamiért a Debian/GNU Linux vált be a legjobban, természetesen egyéb disztribúciókkal sincs komolyabb probléma.
- Ne próbáljunk otthon, egy kábelnetes, vagy DSL kapcsolat végén szerverünkkel örömet okozni az internet nagy közösségének, mert sikertelenek leszünk, tegyük szerverünket valamelyik hosting központba. Evvel sávszélesség-problémánk nemigen adódhat, mert egy kb. 20 főt kiszolgáló szerver, 2 Mbitnyi adattömegnél nem továbbít többet másodpercenként (újabb játékok netkódja ennél is hatékonyabb, akár dupla ennyi játékost is kényelmesen elfuttatnak ekkora sávszélességen).
- Védjük gépünket! Hasonlóan az egyéb on-line fórumokhoz, a konfliktusokat sajnos elég sokszor a közösség drasztikusabb, nagyobb devianciával megáldott felhasználói a szerverek kiiktatásával próbálják rendezni. Mindenképp használjunk tűzfalszoftvert – pl. Netfilter/iptables –, de kerüljük az integrált, nagy tudású, vírusszűrős, „ingyombingyom csillivilli” képességekkel ellátott tűzfalszoftverek ilyen környezetben való alkalmazását, mert növelhetik a szerver reakcióidejét.
- Ne egyszerre indítsunk be több kiszolgálót, hanem minden folyamat beiktatása után várjunk a felhasználók/játékosok visszajelzésére, (ezt ne felejtsük el tőlük kérni, ha maguktól nem nyilatkoznak) és tökéletes működés esetén indítsuk be következő kiszolgálófolyamatunkat.
- Annyi hardver, amennyit elbír a ház...
 - Memóriával és processzorral sose spóroljunk, pl. egy MOH:AA szerver, akár 100 MB – néha több – memóriát is hajlandó a magáévá tenni, ha pedig nem kap, akkor kíméletlenül elkezd swappelni, darál a diszk, áll a játék.
 - Processzor... Eddigi tesztjeink alapján egy min. 2 Ghz-es konfigurációval kezdjük neki, ez kb. 5 kiszolgálófolyamatot elvisz kezdetnek, de terheléstől függően többet is beindíthatunk rajta.

- Óvatosak legyünk, pl. egy Counter Strike-ből ahol elfut 2, MOH:AA-ból elfut 8 és egy Battlefield 1942-ből akár 12 is. Ezek a számok természetesen nem általánosak, de hangsúlyozom, hogy csak a felhasználók konkrét visszajelzésére adhatunk, mérni nem nagyon tudjuk e folyamatok erőforrásigényét.

Tekintsük röviden át, miből is áll össze a játékszerverünk, úgy hardver, mint szoftver oldalon.

3.1. Hardver

Hardvernek egy 3 GHz-es Pentium 4 (valamiért jobb eredményeket értünk el vele, mint AMD-s társaival), 1 GB memória, 2 db 80 GB-os 7200 RPM merevlemez, amit szoftveresen RAID-1-be szervezünk, a biztonság kedvéért, vagy 1–2 db 36 GB-os UW-3 SCSI merevlemez egy gyors vezérlővel, bár itt már spórolásabbak legyünk a helyel.

3.2. Szoftver – OS

Telepítsünk egy Debian Sarge disztribúciót, de kizárólag az alaprendszert tegyük fel, mindenféle ppp, pppoe és olyan csomagokat, amik nem szükségesek a rendszer működéséhez, távolítsunk el. Hozzuk létre a szoftveres RAID partíciókat, a /home szekciót külön partícióra tegyük és használjunk XFS fájlrendszert, ami sebességben pl. az ext3-hoz képest nagy előrelépés.

Magától értetődik, hogy szép, új, optimalizált kernelt kell használjunk, amit saját magunk fordítunk. Használjunk 2.6-os sorozatot, – a 2.4-eseket még bütykölni kellene, a 2.6-osban sok minden alapértelmezett, amit a 2.4-esben javítgatnánk –, mindeképpen 2.6.3-as feletti verziót, mert ezekben kevesebb biztonsági rést tártak eddig fel. Használjunk grsecurity patchet [1], ez növeli a biztonságot.

Hozzuk létre a felhasználót, akinek a nevében az alkalmazás futni fog, esetünkben legyen ez mondjuk *mohaa*.

3.3. Mohaa telepítése

Egyedi telepítésre van szükségünk, mert a mohaa-nak nincsen Linuxos szerver telepítője.

Töltsük le a mohaa szerver linuxos bináris állományait [2], majd telepítsünk egy Windowsos gépen egy mohaa *klienst*. Az egészet csomagoljuk össze, és másoljuk be a felhasználónk home-könyvtára alá.

Ahol a futtatható windowsos állományok vannak, oda másoljuk a Linuxos változatokat is, és a szerver inentől már futtatható, a megfelelő parancs kiadásával.

Ne feledjük, hogy a gameserver configfájlját a `main` könyvtárba kell bepottyantatnunk.

4. Adminisztráció

Mitől lesz könnyen adminisztrálható és 24/7 futtatható a szerverünk? A kérdésre jó válasz lehet a Screen nevű segédprogram igénybevétele.

Telepítsük a Screen-t rendszerünkre (Debian alatt csak egy `apt-get install screen` parancs kiadása szükséges, és a szoftver már rendelkezésünkre is áll). A Screen segítségével „elrakhatjuk” háttérbe az alkalmazásunkat, és kilépésünk után a fókusz

visszkapjuk, tehát ha valami gond van a szerverrel, vagy változtatásra van szükség, azt könnyedén megtehetjük, mintha ki sem léptünk volna a kiszolgálóról.

Már jó úton haladunk, de ne feledjük, hogy ezek az alkalmazások nem mindig stabilitásukról híresek, nem árt tehát egy shell scriptet írni, ami figyel, hogy fut-e még a processzünk és újraindítgatja, ha valami miatt egy kellemetlen crash áldozatává válik a folyamat.

Azt is megtehetjük, hogy a rendszer újraindítása után automatikusan elinduljanak játékszervereink. Arra ügyeljünk, hogy a Screen-t ilyenkor automatikus detachra kérjük fel, és ne felejtjük el elnevezgetni a screen processzeinket.

5. További jótanácsok

Tanulmányozzuk át a hivatkozásokban található oldalakat, rengeteg hasznos információt találunk, adminisztrációt segítő scripteket és ötleteket, a szoftverek által meg nem valósított funkciókra.

Egy példát említve, a MOH:AA nincs felkészítve arra, hogy *ban listát* tartson fent, ha kirúgunk egy embert szerverünkről, bármikor visszajöhet és ez nem szerencsés. Ilyenkor a rendszer `route` parancsával `route-oljuk` el a nem kívánt IP-t egy jó időre valami nem használt, vagy nem létező tartomány felé, így már nem valószínű, hogy vissza fog térni. Ügyeljünk arra, hogy a többség dinamikus ip-vel jön játszani, tehát akit ma letiltunk, az holnap nem biztos, hogy ugyanaz a személy lesz, érdemes ezt is scriptből kitisztogatni, ha nem szeretnénk szép lassan letiltogatni a legtöbb szolgáltató dinamikus ip kiosztásra fenntartott tartományait. ☺

Jó szerencsét és sok kitartást mindenkinek, aki erre a vállalkozásra adja a fejét. Sok fejfájással járó szórakozás ez, de a felhasználók/játékosok szemmel látható öröme minden energiát megér.

Hivatkozások

[1] <http://www.grsecurity.net/>

[2] <http://mohaa.hu/>

[3] <http://www.callofduty.hu/>

[4] <http://www.counter-strike.hu/>

[5] <http://www.mohadmin.com/>

(x)

Rendszerfelügyelet Linux környezetben

Hargitai Zsolt
Novell

Kivonat

Az előadás áttekinti a Linuxos és vegyes környezetek során felmerülő legfontosabb rendszerfelügyeleti igényeket és az erre létező megoldásokat. Bemutatásra kerülnek a legtöbb Linux alapú megoldásban, így a SUSE LINUX Enterprise Server 9-ben is meglévő funkciók felhasználók és jogosultságok kezelésére, erőforrások felügyeletére. Az előadás során ezek a szolgáltatások összehasonlítása kerülnek a Novell által kínált továbbfejlesztési lehetőségekkel.

- Felhasználói adminisztráció password file, NIS, OpenLDAP és Novell eDirectory használata esetén
- Jogosultság kezelés az ismert Linuxos fájlrendszerekben, valamint a Novell hamarosan Linuxon is elérhető NSS (Novell Storage Services) rendszerében
- Szerverenkénti adminisztráció a YaST használatával, címtár alapú felügyelet az iManager-rel
- Szoftverfrissítés a YOU (YaST Online Update) és a ZENworks Linux Management (korábban Red Carpet Enterprise) használatával

Tartalomjegyzék

1. A Novell által jelenleg forgalmazott Linux alapú termékek, megoldások	156
2. A SUSE LINUX Enterprise Server által nyújtott szolgáltatások	157
2.1. A termék legfontosabb jellemzői	157
2.2. A SLES9 rendszerfelügyeleti szolgáltatásai	157
3. További felhasználó adminisztrációs megoldások a Novell-től	159
3.1. eDirectory	159
3.2. Nsure Identity Manager (korábban DirXML)	160
4. További rendszerfelügyeleti megoldások a Novell-től	161
4.1. ZENworks Linux Management	161

A Novell-ről

A Novell szerint a Linux és a nyílt forráskódú rendszerek terjedése a vállalati megoldásokhoz szükséges technológiák fejlődésének egyik kulcsfontosságú eleme. A Novell termékeinek nagy részét már portolta Linux-ra és a SUSE, valamint a Ximian felvásárlásával számos értékes technológia és jelentős Linux-os fejlesztői és tanácsadási tudás került a Novell-hez. Így napjainkban a Linux rendszerekhez is elérhetővé vált a Novell által biztosított világméretű műszaki támogatás és képzés, valamint az iparágon belül vezető szintű hálózati és biztonsági szolgáltatás. Ez vonzó alternatívát kínál azon cégek számára, akik ki szeretnék használni a nyílt forráskód számos előnyét. A Novell a vállalati megoldások szerverektől az asztali gépekig terjedő teljes skáláját kínálja a Linux platformra.

1. A Novell által jelenleg forgalmazott Linux alapú termékek, megoldások

Szervermegoldások

- SUSE LINUX Enterprise Server
- Novell Nterprise Linux Services

Rendszerfelügyelet

- ZENworks Linux Management (korábban Red Carpet Enterprise)

Személyazonosság kezelés

- eDirectory
- Nsure Identity Manager (korábban DirXML)

Csoportmunka megoldás

- SUSE LINUX Openexchange Server
- GroupWise

Alkalmazásfejlesztés

- Novell exteNd

Felhasználói munkaállomás

- SUSE LINUX Desktop
- Ximian Desktop

Szolgáltatások

- Terméktámogatás
- Oktatás
- Jogi Védelmi Program

2. A SUSE LINUX Enterprise Server által nyújtott szolgáltatások

A SUSE LINUX Enterprise Server 9 (SLES9), amely mögött a Novell kiterjedt támogatási infrastruktúrája és partnerhálózata áll, egy biztonságos, megbízható platform nyílt forráskódú számítástechnika használatához a vállalaton belül. Az új 2.6-os Linux kernelre épülő SUSE LINUX Enterprise Server 9 páratlan teljesítményt és méretezhetőséget kínál, átfogó, nyílt forráskódú funkcionalitást, valamint hardverplatformok és szoftvercsomagok széles körének támogatását. A SUSE LINUX Enterprise Server 9 ezenfelül nyílt alkalmazásprogramozási felületeket (API-kat) és más fejlesztőeszközöket is tartalmaz, amellyel leegyszerűsíthető a Linux-integráció és a rendszerek testreszabása.

2.1. A termék legfontosabb jellemzői

- Iparágvezető teljesítmény és méretezhetőség
- Magas szintű rendelkezésre állás és megbízhatóság
- Páratlan biztonság
- Egyedi üzembe helyezési és rendszerfelügyeleti funkciók
- Átfogó, nyílt forráskódú funkcionalitás
- Rugalmas vállalati tárolási és virtualizációs funkciók
- Széleskörű platformtámogatás, egységes kódalap
- Fejlesztői termelékenységi
- Novell-háttértámogatás

2.2. A SLES9 rendszerfelügyeleti szolgáltatásai

A YaST integrált telepítési, beállítási és adminisztrációs megoldást kínál a SUSE LINUX rendszerekhez. A jelenleg GPL-licenc hatálya alá tartozó YaST a felügyeleti feladatok széles skáláját képes ellátni, és egységes felügyeletet kínál minden támogatott SUSE LINUX platformon. API-jai használatával a fejlesztők és a külső gyártók kényelmesen bővíthetik. Mivel a YaST a GPL-licenc alatt került kiadásra, az iparág minden résztvevője teljes mértékben alkalmazhatja anélkül, hogy a Novellal való versenyzés miatt kellene aggódnia. A Novell döntése, hogy nyílt forráskódúvá teszi a YaST-ot, megfelel a gyártóknak és vásárlóknak tett ígéretünknek, hogy támogatjuk a nyílt szabványokat – jelen esetben a rendszerfelügyelettel kapcsolatban –, hogy sokkal könnyebben felügyelhetővé tegyük a rendszereket és az alkalmazásokat. A SUSE YaST eszközökkel folytatott szerveradminisztráció többféle mód támogatásával bővült. A rendszerindítás lehetséges CD-ről, hajlékonylemezről, helyi merevlemezről vagy PXE használatával a hálózatról. Az AutoYaST lehetővé teszi a felügyelet nélküli és az automatikus telepítést is.

Új YaST modulok

A YaST számos új modullal rendelkezik a SUSE LINUX Enterprise Server 9-ben, például:

- A YaST új, levéltovábbító szerverek beállítására alkalmas eszközével az adminisztrátorok biztonságos IMAP- vagy POP-szolgáltatásokat nyújtó, kvótákat, hozzáférés-vezérlési listákat, névtereket, útválasztást, helyi levélkézbesítést, szerveroldali vírus- és spam-szűrést vagy más nagyvállalati szintű levélküldési funkciókat használó szervereket készíthetnek.
- VPN-konfigurációs kisegítő mind a kliens, mind a szerver beállításához. A VPN kompatibilis a Linux- és a Windows-kliensekkel is, és kiegészítő szoftverek használata nélkül beállítható.
- A Samba 3 eszközzel PDC és BDC rendszerek hozhatók létre a fájlok megosztására Windows (CIFS vagy SMB) hálózaton keresztül. Lehetővé teszi az LDAP és az smbpasswd hitelesítés használatát, a grafikus megosztás-kiválasztást és felügyeletet, a hozzáférés-vezérlési listák használatát, valamint kibővített attribútum-támogatást nyújt a nem natív fájlrendszerekhez.

Továbbfejlesztett YaST modulok

A következő YaST-konfigurációs eszközök kerültek a jelenlegi kiadásban továbbfejlesztésre vagy frissítésre. A változások a felhasználói felület javításától az új funkciók és képességek bevezetéséig terjednek:

- Minden hálózati konfigurációs eszköz fejlődött, így a DNS, DHCP, LDAP, NIS, postfix és TFTP is NFS és Samba hálózati fájlrendszer-beállítások
- A tanúsítványhatóság (Certification Authority) már automatikusan generálja az alapértelmezett tanúsítványokat a szerverek számára (pl. LDAP, Apache, postfix)
- Virtuális magánhálózat (VPN)
- Telepítési szerver
- rendszerindító szerver
- CD-készítés
- User-Mode Linux telepítési és virtualizációs beállítás
- Apache
- Wake on LAN (rendszerébresztés hálózatról)
- A nagy rendelkezésre állást biztosító eszközök kiterjesztése a Heartbeattal történő használathoz
- Frissítési szerver
- A felhasználói felügyeleti eszközök már támogatják a külső háttérrendszerek bővítőmoduljait (pl. IMAP vagy Samba)

Common Information Modell támogatás

A Common Information Model (CIM, közös információs modell) platformfüggetlen és technológiailag semleges módon szabványosítja a felügyeleti információk cseréjét. A CIM-technológiás szoftverek használata csökkenti a felügyeleti költségeket azáltal, hogy egyetlen API használható a heterogén környezetekben. Ahol más CIM-megközelítéseknel minden feladathoz egy új alkalmazásmódulra volna szükség, ott a SUSE LINUX Enterprise Server 9 lehetővé teszi mindössze egyetlen CIM-modul használatát a YaST eléréséhez, és ez azután egyetlen felületet kínál, amelyről minden konfigurációs feladat elvégezhető az operációs rendszerben.

3. További felhasználó adminisztrációs megoldások a Novell-től

A SUSE LINUX Enterprise Server 9 számos felhasználói adminisztrációt lehetővé tevő eszközzel rendelkezik. Támogatja a Unix/Linux világban megszokott password/group/shadow password alapú felhasználó azonosítást, valamint a NIS (Network Information Services, Yellow Pages) technológiát. Címtárként az OpenLDAP használatára is van lehetőség. Ezek a megoldások viszonylag kis felhasználó szám és homogén környezetek esetén megfelelő megoldást nyújtanak. A nagyobb felhasználószám és/vagy heterogén környezet esetén felmerülő igények kielégítésére a Novell címtárszolgáltatása az eDirectory, az ehhez tartozó szerep alapú felügyeletet kínáló menedzsment rendszer az iManager és a Novell metacímtára az Nsure Identity Manager (korábban DirXML) nyújt megoldást.

3.1. eDirectory

A címtárak segítségével egy biztonságos, mindent átfogó, egyszerűen használható hálózatot tudunk kiépíteni. A címtár alapú struktúra egyik előnye az, hogy az összes hálózati erőforrás (adatok, alkalmazások, nyomtatók, internet stb.) mindenki számára könnyen elérhetővé válik, és a felhasználóknak nem kell azzal törődni, hogy hol is vannak a szükséges erőforrások, vagy hogy éppen szabadok-e. Mindez láthatatlan lesz a számukra, és sokkal átláthatóbb lesz a rendszergazdák számára és így az egész rendszer könnyebben menedzselhetővé, és biztonságosabbá válik. A címtár alapú rendszerek legnagyobb előnye pedig az, hogy egyre több alkalmazást, funkciót, illetve szolgáltatást tudunk erre ráépítve viszonylag könnyen bevezetni, és így hálózatunk egyre intelligensebbé válik.

A Novell címtárszolgáltatása az eDirectory (korábban NDS) egy skálázható, biztonságos, kiforrott, többplatformos címtárszolgáltatás és használatával egy egységes informatikai infrastruktúrát tudunk kialakítani. Az eDirectory Linuxon történő használatával az összes Linuxos szolgáltatás (pl. login, Samba, telnet, ftp, Apache) felhasználói azonosítása LDAP alapon az eDirectory-ban történhet és ezzel a Linuxos rendszereket is integrálni tudjuk a vállalat többi szerverével egy központi vállalati címtár kialakításával.

Az eDirectory alkalmazásának legfontosabb előnyei:

- Egységes jogosultság kezelés
- Rugalmas és skálázható felépítés

- Hatékony rendszerfelügyelet
- Biztonsági szolgáltatások
- Szabványok kezelése
- Platformfüggetlenség
- Új szolgáltatások bevezetésének támogatása

3.2. Nsure Identity Manager (korábban DirXML)

A mai informatikai rendszerek túlnyomó többségét a sokszínűség jellemzi. Számos különböző hardver- és operációsrendszer-platformon futó, jellemzően elszigetelt alkalmazások szolgálják ki a különböző üzleti igényeket. A rendszerek – sokszor több generációjuk is – egymás mellett élnek, de az együttműködés szintje alacsony. A Novell metacím tár megoldása az eDirectory és az Nsure Identity Manager képes a különböző korokból származó informatikai rendszereket integrálni, az intézmény adatait és alkalmazásait szinkronban tartani. Az egyes – különböző gyártóktól származó – részrendszerek a felhasználók számára láthatatlanul, a háttérben szinkronizálják adataikat és ezek az egységes, integrált és szinkronizált adatok és alkalmazások a felhasználók számára személyre szabottan, biztonságosan, a szervezeten belülről és kívülről egyaránt, az év bármely napján elérhetők.

Az Nsure Identity Manager támogatott Linux platformon és képes a Linuxos felhasználó kezelés és a Linux szervereken futó alkalmazások integrálására a vállalat más szoftvereivel, adatbázisaival.

A felhasználók számára az alábbi előnyökkel jár egy ilyen rendszer kialakítása:

Adatintegráció. A vállalat teljes egészére vonatkozóan egységesek, naprakészek és pontosak az adatok.

Üzleti folyamatok támogatása. Gyorsabbá válik az új alkalmazottak munkába állása, az informatikai rendszer könnyebben és rugalmasabban követi a szervezeti változásokat.

Hozzáférés. A kívánt információ könnyen, rugalmasan és sokféle módon nyerhető ki a rendszerből, pl. egy webes felületen elérhető vállalati telefonkönyv könnyen kialakítható.

Biztonság. A rendszer biztonsága, lévén alapszolgáltatás, eleve magas szintű és kikerülhetetlen. Ugyancsak a biztonságot erősíti az, hogy a vállalattól eltávozó alkalmazottak hozzáférési jogosultsága az összes rendszerben egyszerre megszüntethető.

Bővíthetőség. Új, későbbiekben bevezetendő szolgáltatások igény szerint könnyen felhasználhatják a metacím tárban rendszerezett információt megkönnyítve ezzel az informatikai rendszerek fejlesztését.

Jogosultságok. Megoldást kínál az Nsure Identity Manager arra is, hogy meghatározott címtárakat, adatkezelőket – és így a hozzájuk tartozó szervezeti egységeket – felelőssé lehessen tenni bizonyos típusú adatokért. Ezeket az információkat így csak adott személyek módosíthatják és ezek a változások csak adott, előre definiált módon terjedhetnek tovább a rendszerben.

Alkalmazások integrálása. Az Nsure Identity Manager ún. meghajtó programokon keresztül kommunikál a különböző alkalmazásokkal. A Novell az Nsure Identity Manager-t számos előre megírt meghajtó programmal szállítja. (NDS, Active Directory, NIS, LDAP, JDBC (Oracle, DB2, MS SQL), Lotus Notes, Exchange, GroupWise, SAP, PeopleSoft stb.), amelyek testre szabása a szabványos XML nyelv segítségével történik.

Fejlesztési lehetőségek. A terméknek része egy szoftverfejlesztői készlet (SDK) is, amellyel egyedi meghajtó programok írhatók a különböző alkalmazásokhoz, adatbázisokhoz és címtárakhoz. E C++ és Java-meghajtó programok még a régi vagy egyedi alkalmazásokkal, adatbázisokkal is képesek együttműködni.

4. További rendszerfelügyeleti megoldások a Novell-től

A SUSE LINUX Enterprise Server 9 szolgáltatása a YOU (YaST Online Update) képes a szerverenkénti szoftverfrissítés megvalósítására. Nagyobb számú Linux alapú számítógép (kiszolgáló, vagy munkaállomás), illetve többféle Linux platform használata esetén a ZENworks Linux Management (korábban Ximian Red Carpet Enterprise) nyújt integrált felügyeleti megoldást.

4.1. ZENworks Linux Management

A ZENworks Linux Management vállalati szoftvercsomagot kínál Linux rendszerekhez, mely teljes körű szoftver elosztást, telepítést, visszaállítást, javítást és jelentéskészítést kínál. A gyors bevezetésnek, valamint a Linux disztribúciók és alkalmazások széles körű támogatásának köszönhetően a szervezetek a befektetés gyors megtérülését érhetik el.

A szoftvercsomagok csomagkészletekbe szervezhetők, melyek termékeket, termék-komponenseket vagy termékcsaládokat fednek le, és külön egységként kezelhetők az elosztás, telepítés, visszaállítás és jelentéskészítés során. Elvégzi a függőségek automatizált és intelligens analízisét, feloldja a csomagok között fellépő esetleges ütközéseket, képes állandó és pillanatnyi host-csoportokat kezelni, időzíti a frissítéseket. A kezelt szerverek és munkaállomások készletezésének támogatásával összegyűjthető a hálózati információ, így a hardver-, szoftver- és rendszerspecifikációk. A továbbfejlesztett vállalati szoftverelosztás gyorsítótár szervereket használ a csomagok lassú hálózatokon keresztül történő hatékony eljuttatásához.

(x)



i n v e n t

Cégünk mottója:

*Célunk, hogy ügyfeleink ne csak elégedettek legyenek,
de a mi szolgáltatásunk legyen a mérték.*

Ennek szellemében vállaljuk az alábbiak kivitelezését:

- Hálózat építés és karbantartás
- Egyedi szerver telepítések és megoldások
- Alkalmazásfejlesztés
- Konzultáció és tanácsadás
- Tűzfal telepítés és karbantartás
- Távfelügyelet és support

Az alábbi cégek teljes termékskálájához biztosítunk telepítési és karbantartási szolgáltatást:

Apple - Macintosh kliensek és szerverek
Balabit IT - Zorpx tűzfal

Weboldal: <http://www.lsc.hu/>
e-mail: info@lsc.hu
Telefon: 341-0457



[...MISSION CRITICAL LINUX - SZAKÉRTELEM FELSŐFOKON...]
A Mission Critical Linux Kft. 2000-ben alakult a Linux operációsrendszer magyarországi elterjedésének elősegítése, és a professzionális Linux-szolgáltatások nyújtása érdekében.

Működési területeink:

a Linux-alapú cluster technológia, a Linux-kompatibilis speciális hardvereszközök disztribúciója, a Linuxrendszer telepítések és mindezek technikai támogatása.

1027 Budapest, Frankel Leó út 11-13. Tel.: +36 (1) 438-4100
E-mail: info@mclx.hu • Honlap: www.mclx.hu



redhat®



Kiskapu Kft. – Linuxvilág magazin



Cégünk magyar és angol nyelvű számítástechnikai szakkönyvek, magazinok terjesztésével és kiadásával foglalkozik. Kiemelt figyelmet szentelünk a Linuxszal kapcsolatos kiadványoknak – e törekvésünk gyümölcseként született meg a Linuxvilág magazin is, amely az amerikai Linux Journal társlapjaként jelenik meg havonta.

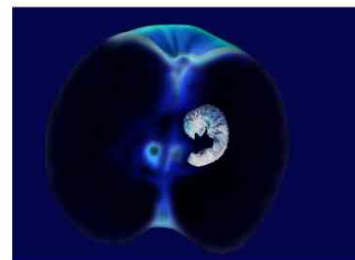
Címünk: 1081, Budapest, Népszínház utca 29.

Internetes könyvesboltunk:

www.kiskapu.hu

A Linuxvilág magazin honlapja:

www.linuxvilag.hu



Nem mindegy, mi jut be a szervezetébe!

Zorpx tűzfaltechnológia a valódi biztonságért

www.balabit.hu
© 2004 Balabit IT Biztonságtechnika



Novell®



• @ *matáv*



Interware

