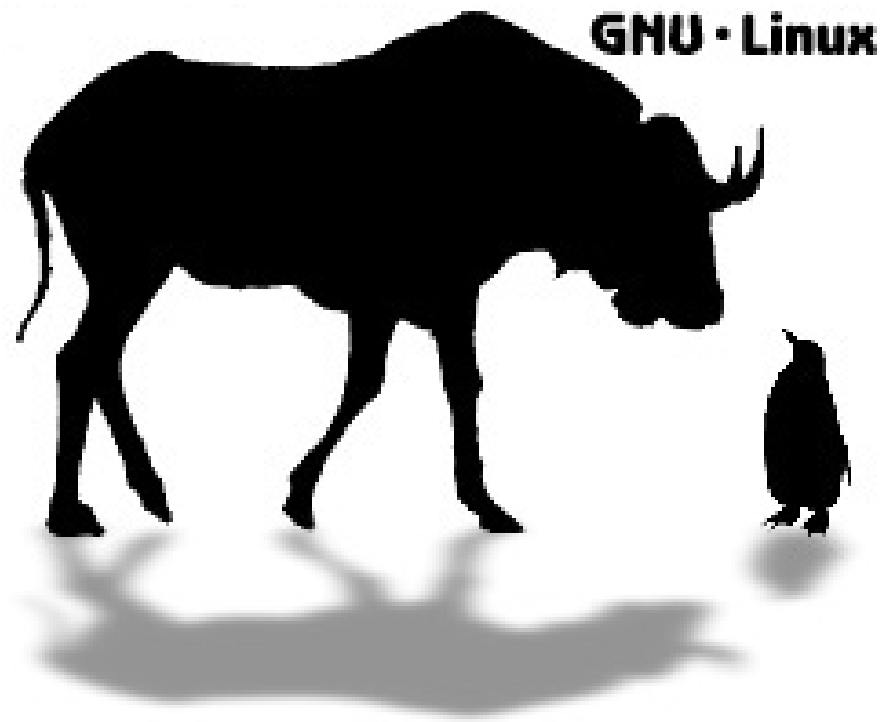
# GNU/Linux konferencia




Budapest 2001. szeptember 15.

A kiadvány tördelése a TEX 3.14159 verziójával készült,
GNU/Linux operációs rendszeren.
A TEX az American Mathematical Society bejegyzett védjegye.

Szerkesztette: Zelena 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*

# Tartalomjegyzék

**Deim Ágoston:**
**Mail szerverek védelme** **7**

**Fodor Orsolya:**
**Weboldalkészítés GNU/Linux alatt** **15**

**Kósa Attila:**
**A Linux vállalati alkalmazása** **17**

**Kósa Barna:**
**GNU/Linux fürtök a rendelkezésre állás és a teljesítmény növelésére** **29**

**Laky Norbert:**
**Vékony kliens technológia a Fővárosi Bíróságon** **39**

**Magosányi Árpád:**
**Nyílt forrású fejlesztési projectek dinamikája** **43**

**Mátó Péter:**
**Biztonságos aktív webkiszolgáló építése** **53**

**Németh László:**
**Unix, vagy nem Unix? A Unix filozófia jövője a Desktop rendszerek korában** **63**

**Papp Dániel – Pintér Zoltán:**
**Fürtözési megoldások – Linux-alapú cluster technológiák** **77**

**Pásztor Miklós:**
**Digitális aláírással kapcsolatos eszközök és veszélyek** **81**

**Scheidler Balázs:**
**Hálózati határvédelem eszközei** **91**

**Süveg Gábor:**
**GIMP, avagy grafika unixon** **97**

**Zahemszky Gábor:**
**A BSD világa** **101**

**Zámbó Marcell:**
**Biztonságos chroot kialakítása GNU/Linux Operációs rendszeren** **113**

# A konferencia támogatói

Fő támogató: **IBM Magyarország**

Támogatók:

- AVNET
- BalaBit
- HumanSoft
- Mission Critical Linux
- Polygon
- SuSE
- Telnet Magyarország

# Előszó

Üdvözöljük abból az alkalomból, hogy ismét elérkezett az LME már-már hagyományos szakmai konferenciája, s ezzel együtt elkészült ez a több, mint száz oldalas, a konferencia előadásait tartalmazó kiadvány.
Harmadik alkalommal szervezett egyesületünk szakmai konferenciát. Ez a mostani konferencia nem csak előadásai számát tekintve más, vagy nagyobb, mint a korábbiak, hanem reményeink szerint tartalmában is.
Konferenciánknak egy előadás erejéig vendége lesz Richard M. Stallman, továbbá –támogatónk jóvoltából– eddig nem, vagy csak keveset látott hardveren is találkozhatunk kedvenc „pingvines” rendszerünkkel.
Reméljük az előadások, workshop-ok, illetve kiállítások találkoznak az Ön igényeivel, és viszontlátjuk az LME jövőbeni rendezvényein.
Amennyiben kérdése, észrevétele, javaslata van, kérjük ossza azt meg velünk a konferencián az LME-standján, vagy a későbbiekben az *info@lme.linux.hu* e-mail címen.

A bevezető végén csak azt kívánhatjuk, hogy forgassák haszonnal kiadványunkat, és találkozzunk 2002-ben, egy újabb, nagyszabású Linux-os konferencián!

Tisztelettel,

A szervezők

# Mail szerverek védelme

Deim Ágoston

2001.08.29

**Kivonat**

Az előadás célkitűzése, hogy bemutassa a levelező szerverek ellen irányuló leggyakoribb támadások formáit és az azok ellen hozott intézkedéseket.
Az előadás elméleti része mutatja be a támadó technikákat és azok ellen hozható intézkedéseket. A gyakorlati rész célja, hogy bemutassa a védelem MTA-ra oszott szerepét. Az előadás gyakorlati részénél alkalmazott szerver a *postfix*, de az elméleti résznél ismertetett védelmi mechanizmusok alkalmazhatók más MTA-val is.

## Tartalomjegyzék

**1. Bevezetés** **8**

**2. Támadások az E-mail címek megszerzésére** **8**
2.1. E-mail címek begyűjtése . . . . . 8
2.2. POP3 és IMAP szerver problémái . . . . . 10
2.3. Címgyűjtés a webről és levelezőlistákról . . . . . 11

**3. Támadások egyéb célokból** **11**
3.1. Szerverek túlterhelése . . . . . 11

**4. Egyéb szolgáltatások hibái** **12**
4.1. Konfigurálási hibák . . . . . 12
4.2. Open Relay . . . . . 12

**5. Elméleti rész – összefoglalás** **13**

**6. Gyakorlati megoldások** **13**

**7. Gyakorlati rész – összefoglalás** **14**

# 1. Bevezetés

Az első kérdés, hogy mit tekintünk támadásnak. A levelező szerverek elleni támadások nem korlátozódnak a szokásos támadási formákra, melyeknek célja, hogy a megtámadott gépen rosszindulató programokat futassanak és kiemelt jogosultságokat szerezzenek azon, uralmat szerezve a rendszer felett. Mivel az elektronikus levelezés mára az üzleti élet egyik legfontosabb információközvetítő eszköze lett, ezért a levelek elfogása vagy meghamisítása is támadási formának minősül. A levelek eredetiségét és esetleges módosításának felfedezését biztosító eszközök bemutatása az előadásnak nem célja, mivel ez legnagyobb részt valamilyen titkosító algoritmus használatát feltételezi. Kitérünk azonban a veszélyek másik válfajára, a levelek eltérítésére illetve ellopására, megszerzésére.

# 2. Támadások az E-mail címek megszerzésére

A levelezőrendszerek többféle módon támadhatók, ezekre mutatok néhány példát, és természetesen az ezek elleni védekezés lehetőségét is, ahol lehet.

## 2.1. E-mail címek begyűjtése

Az első bemutatandó támadási forma az address harvesting, azaz az email címek begyűjtése. Miért tekintjük ezt támadási formának? Egyrészt mert ugródeszkát jelent más támadási formákhoz, másrészt a támadó előtt rögtön megnyit egy lehetőséget. Mindennapi levelzése során bizonyára mindenki találkozott már kéretlen hirdetéssel is postaládájában, mely egy új árura, kedvezményre vagy nagy nyereményre hívta fel a figyelmét. Ez az úgynevezett spam, hivatalos nevén UCE (Unsolicited Commercial E-mail). Bár nem közvetlenül veszélyeztetik felhasználóinkat vagy a rendszert mégis támadási formának tekinthető. Hiszen ha sokszor kapnak az általunk kiszolgált domainek illetve felhasználók kéretlen leveleket, akkor lehet, hogy továbbállnak és más cég szolgáltatásait veszik igénybe. A spam elleni védekezést egy másik pontban tárgyaljuk. Nézzük akkor tehát, hogy milyen módszerekkel lehet az MTA-n keresztül megszerezni a kiszolgált címeket.
A lehetőséget két SMTP parancs biztosítja támadó számára. Ez a két parancs a *VRFY* és az *EXPN*. Segítségükkel lehet végrehajtani az *address harvesting* nevű támadást. A *VRFY* segítségével a rendszeren létező felhasználók nevét tudjuk megszerezni. A kapcsolat a következőképpen néz ki:

```
ago@localhost:~[3]# telnet mail.domain.hu 25
Trying XXX.YYY.ZZZ.TTT...
Connected to mail.domain.hu.
Escape character is '^]'.
VR220 mail.domain.hu ESMTP Sendmail 8.9.3/8.8.7;
VRFY info
250 mail <info@mail.domain.hu>
```

Látszik tehát, hogy megkaptuk a felhasználó e-mail címét. Ha a válasz nemleges volt, tehát az e-mail cím nem létezik, akkor a válasz a következőképpen néz ki:

```
ago@localhost:~[3]# telnet mail.domain.hu 25
Trying XXX.YYY.ZZZ.TTT...
```

```
Connected to mail.domain.hu.
Escape character is '^]'.
  VR220 mail.domain.hu ESMTP Sendmail 8.9.3/8.8.7;
  VRFY ago
550 ago... User unknown
```

A helyzet azonban lehet még rosszabb is, ha az EXPN parancs használatát engedélyezzük. Ekkor ugyanis nem csak a rendszer felhasználóinak címéhez juthatnak hozzá, hanem az *alias* fájlban és a virtuális domainek felhasználóit tartalmazó fájlban létező címekhez is.
A fenti két parancs elég könnyen szkriptelhető. Gondoljunk csak bele milyen egyszerű olyan *Perl*-szkiptet írni, mely bejelentkezik a távoli gép 25-ös portjára és ott sorozatos lekérdezéseket hajt végre. Csak meg kell nézni, hogy milyen válaszokat ad vissza szerver és a kapott válaszok alapján a létező e-mail címeket el tudjuk tárolni egy fájlban. A próbált felhasználóneveket vehetjük fájlból is vagy *brute-force* módszerrel is felderíthetjük a távoli gépen érvényes e-mail címeket.
A megoldás tehát, hogy az *MTA* dokumentációjában keressük meg hogyan lehet letiltani a fenti parancsokat.
Ekkor egy elutasított kérésnek például így kell kinéznie:

```
[root@localhost /root]# telnet 0 25
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
220 localhost.localdomain ESMTP Postfix
HELO localhost
250 localhost.localdomain
VRFY ago
502 VRFY command is disabled
```

Sajnos van olyan módszer melyet nem lehet kiiktatni. E sajnálatos eredményt nem az *SMTP* protokoll gyengesége, hanem a levél fogadás folyamata teszi lehetővé. Ha a *telnet* parancs segítségével ismét nyomon követjük a kapcsolatot, akkor látni fogjuk, hogy hol a gyenge pont.

```
[root@localhost]# telnet 0 25
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
220 localhost.localdomain ESMTP Postfix
HELO localhost
250 localhost.localdomain
MAIL FROM: valaki@valahol.hu
250 Ok
RCPT TO: ago
250 Ok
```

Itt is elég könnyű elképzelni egy programot, melynek segítségével meg tudjuk szerezni a valós e-mail címeket. Csak a *250*-el kezdődő válaszokat kell figyelni, hogy a megadott mail címre ezt a választ kaptuk e. Ha igen, csak el kell tárolni egy fájlban vagy adatbázisban a neveket.

Ez ellen a módszer ellen nehéz védekezni, igazából csak lassítani tudjuk a támadót. Ez azonban elég lehet. Ha nagyon lassan kapja vissza a válaszokat és elunja a dolgot vagy már nem hatékony számára a módszer. A védelemnek tehát be kell vonnia egy külső programot a rendszerbe, mely a próbálkozásokat figyeli és túl sok megszakadt kapcsolat vagy csak a címzett megnevezéséig felépített kapcsolat észlelésekor adott ideig letiltja az adott IP-ről érkező kéréseket, így késleltetve a támadót. Ez az idő persze nem tarthat örökké, de ha minden kapcsolat felépüléséig elég ideig várakozni kényszerül a támadó, akkor egy idő után a címek megszerzése olyan lassú lesz, hogy már nem éri meg vele foglalkozni vagy megnő a lebukás veszélye - ami amúgy is fennáll, ha a támadott gép rendszergazdája rendszeresen olvassa a naplófájlokat. Ez tipikusan proxynak való feladat, bár néhány *MTA* a túl sok egy helyről érkező kapcsolatot képes lassítani, ha engedélyezzük ezt a funkciót.
Felmerülhet még egy olyan védekezési mód, melyhez ismét csak proxy kell. A támadó minden bejelentkezésére pozitív visszajelzést kap. Miért segít ez? Ha például néhány másodperc alatt mintegy 500 érvényes e-mail címet kap vissza, akkor talán elgondolkodik azon a támadó, hogy ebből nagyon nehéz lesz kiszűrni a valóban létező címeket. Sajnos ilyen proxyt még nem láttam, legalábbis GPL vagy BSD licenszűt.
Ha azonban eltekintünk attól a működési módtól, hogy az RCPT TO: után megadott megadott e-mail címet nem ellenőrizzük vissza rögtön, akkor még MTA szintjén is meg tudjuk ezt a támadást akadályozni. Ezt úgy lehet elérni, hogy rábírjuk az MTA-t, ne ellenőrizze le a kapcsolat elején az e-mail cím helyességét, hanem csak amikor már megpróbálja a felhasználó levelesládájába menteni a levelet vagy tovább akarja küldeni azt. A postfix, mivel moduláris, ezt könnyen elérhetővé teszi számunkra. Egyszerűen kikapcsoljuk a felhasználói nevek ellenőrzését a leveleket fogadó modulban, de ez majd működni fog a lokális levélközvetitő démonnál. Így minden levelet és címet elfogad a kapcsolat kezdetén, majd visszaküld egy hibaüzenetet a küldőnek, ha a cím nem létezik a rendszeren. Ez azonban adott esetben megnöveli a szerverre nehezedő terhelést. Ilyenkor választanunk kell, hogy a hatékonyságot vagy a biztonságot helyezzük előtérbe.

## 2.2. POP3 és IMAP szerver problémái

Másik gyenge pont lehet egy POP3 vagy IMAP szerver. Ezek akár a felhasználó leveleinek megszerzésére is alkalmasak a címek gyűjtésén kívűl. A címek megszerzésének problémája ismét az azonosításnál gyökerezik -hiszen az RCPT parancs is gyakorlatilag egy azonosítás-, mivel ezt mindenképpen engedélyeznünk kell. A támadás is hasonló, mint az előző példában, csak most a POP3 vagy IMAP szervert támadják. A támadó ír egy szkriptet, ami POP3 protokoll esetében a USER parancsot hívja csak meg és ha az pozitív választ eredményez, akkor máris van egy használható e-mail címe. A megoldás ismét csak egy, a sorozatosan hirtelen megszakított vagy csak bizonyos fázisig eljutó kapcsolatokat figyelő és az így viselkedő klienseket adott ideig letiltó proxy alkalmazása. Akárcsak az MTA-nál itt is alkalmazható lenne egy olyan proxy, mely minden név lekérdezésre pozitív választ ad vissza.
Van olyan támadás a POP3 és IMAP szerverek ellen, mely ellen nem védene még egy ilyen proxy sem és valóban csak a késleltetés nyújt időleges védelemet, hacsak nem alkalmazunk valamilyen hardveres, például CryptoCard-os azonosítást vagy egyéb trükköket. Ez az a támadás, mikor a felhasználó leveleihez akarnak hozzáférni, azokat akarják megszerezni. Ekkor valószínűleg tudják a felhasználó e-mail címét és a @ előtti nevet felhasználva próbálnak belépni a POP3 szerverre. Egy esetleges próbálkozás így néz ki:

```
[root@localhost /root]# telnet 0 110
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
+OK POP3 localhost.localdomain
USER ago
+OK User name accepted, password please
PASS rosszpasswd
-ERR Bad login
```

Ekkor -az említett hardveres megoldáson kívűl- csak a késleltetés segít. A kárt csökkenthetjük, ha minden üzleti vagy fontos magánlevél titkosítva van, de ezen módszerek tárgyalása nem tartozik az előadás tárgykörébe.

### 2.3. Címgyűjtés a webről és levelezőlistákról

A nem túl gyakori és nem is túl hatékony módszerek közé tartozik a címek weboldalakról történő legyűjtése. Itt a védelem abból állhat, hogy a *mailto:* HTML tag mögé nem a címet natívan írjuk, hanem HTML kóddal, például a @ tag helyett annak *HTML* kódját írjuk. Ez a védelem persze könnyen áttörhető, elég csak egy *sed* vagy *awk* szkriptet írni, ami lecseréli az összes tag-et az annak megfelelő karakterre. De ez a címgyűjtési forma annyira nem hatékony, hogy igazából ez nem számít.
Az e-mail címek egyik legnagyobb adatbázisa a levelezőlistákra feliratkozottak címe. Csábító lehet a nagyszámú cím megszerzése. Az egyetlen védekezési mód és egyben a leghatékonyabb, ha a lista adminisztrátora csak saját maga számára engedélyezi a feliratkozott személyek címének megtekintését. Értelemszerűen a címek megszerzése csak a listázását felhasználók számára is engedélyező rendszereken lehetséges.

## 3. Támadások egyéb célokból

Amikor már nem a címek megszerzése a cél, akkor egyéb támadási formák kerülnek előtérbe, ezek leggyakrabban a szolgáltatás ellehetlenítését tűzik ki célul. Ilyen lehet a szerver üzenetekkel való bombázása. Ez több célból is történhet. Az egyik ilyen a szerver túlterhelése.

### 3.1. Szerverek túlterhelése

A módszerek közül népszerűnek számít az úgynevezett mail bomba, amikor érvényes felhasználónak küldenek igen nagy mennyiségű levelet. Ennek célja, hogy az MTA leterhelje a rendszert. Ez ellen úgy lehet védekezni, hogy korlátozzuk az egy helyről érkezett kapcsolatok maximális számát és mesterséges késleltetést alkalmazunk. A postfix illetve a qmail rendelkezik ilyen képességekel.
A szerver túlterhelésének másik divatos módja, hogy a rendszert levelek fogadására alkalmatlanná tesszük. Ezt úgy lehet elérni a legbiztosabban, hogy folyamatosan, de nem sűrű egymásutánban megnyitja a támadó a kommunikációt és több megabyte méretű leveleket küld a címzetnek. Ha a rendszer rosszul van megtervezve és például nincs külön partíción a felhasználók leveleit tároló könyvtárstruktúra, előfordulhat, hogy a fájlrendszer telítődik. A megoldás tehát a helyes partícionálás. Mivel az sem kívántos állapot, hogy a rendszer ugyan megy, de egyik felhasználó sem kapja meg a leveleit

ezért kvótákat kell alkalmaznunk. Van olyan *LDA*, mely képes ezt lekezelni, de a leggyorsabb és legbiztosabb kvóta a fájlrendszeren alkalmazott.

# 4. Egyéb szolgáltatások hibái

Most azon módszerek következnek, melyek akkor válnak lehetővé, ha bizonyos szolgáltatásokat elérhetővé teszünk, mint például a webes levelezőfelület. Ha elérhető a webes felület és tudják a felhasználónevet -ami általában a mail cím @ előtti része- akkor ugyanúgy, mint a pop3 szerver esetén, itt is próbálkozhatnak brute-force módszerrel. Erre kitünően alkalmas például az *authforce* nevű program, mellyel felhasználóinkat is tesztelhetjük. Persze egy szkript írása itt sem lehetelen. Ha megvan a felhasználónév-jelszó páros és ismerjük a webes levelezőkliens viselkedését, akkor tudjuk automatizálni a levélküldést ezen keresztül. Viselkedés alatt a program változóit, hívásait értjük. Ez ellen a módszer ellen vagy proxyval vagy másfajta modullal lehet védekezni, mely adott számú próbálkozás után adott ideig letiltja a próbálkozást a forrás IP címről.
Több olyan lehetőség maradt fent, melyek ismertetésre kerülnek, többek között az open-relayek problémái, konfigurálási és protokoll hibák.

## 4.1. Konfigurálási hibák

Legelőször nézzük a konfigurálási hibákat, melyek összefüggnek a protokollokkal. Ha például SQL adatbázisból vesszük a felhasználóinkat, akkor ha az adatbázis távoli gépen van és nincs semmilyen titkosított csatornával körbevéve, akkor a felhasználónév és jelszó páros clear-textben jelenhet meg a hálózaton. Ugyanez vonatkozik az LDAP adatbázisokra, bár ott van beépített SSL3 támogatás. Viszont a protokollban nemrég találtak hibát, így alkalmazása megfontolandó. Ezek a hibák viszonylag könnyen kiküszöbölhetőek, ha az adatbázisok külön hálózatik kártyán, külön privát címtratományból választott címmel rendelkeznek és fizikailag is csak a kiszolgálóval állnak hálózati kapcsolatban. Egyéb módon például VPN segítségével és tűzfal szabályokkal is védhetőek ezek a szerverek.

## 4.2. Open Relay

Az elméleti rész végén ismerkedjünk meg az open relayek problémáival. Az open relay olyan nyitott levelezőszerver, mely engedi, hogy rajta keresztül a világon bárhová küldhessem a leveleimet. Egy gyakorlati példa. Ha én a domain.hu erős fantázianevű domain számára nyújtok levelezési szolgáltatást, akkor értelemszerűen csak a domain.hu címre érkező leveleket kell átengednem a védelmen és tárolnom vagy a felhasználó által kért címre továbbítanom. Ezek szerint a
tamado@gonosz.hu címről érkező leveleket nem engedhetem más címre, mint a domain.hu címre, ami viszont lokális. Ha a tamado@gonosz.hu rajtam keresztül nem csak a domain.hu címre tud írni, hanem például az valaki@aldozat.hu címre is, akkor nagy bajban vagyok, mert open relay a szerverem. Miért nagy baj ez? Először is, mert a támadó rendelkezésére álló címekre mind el tudja küldeni a levelet anélkül, hogy a saját szerverét -ha van neki- kockáztatná, másodszor pedig, mert hamarosan én sem tudok levelet küldeni sehova, ugyanis egy tiltólistára kerülök. A másik gond nem közvetlenül rám vonatkozik, de rám is hatással lehet. Nézzük miről van szó. Képzeljük el, hogy a szerveremen keresztül akárhová lehet levelet küldeni, tehát open relay. A támadó kiválaszt egy adott domain-t, mondjuk az aldozat.hu-t. Majd az áldozat.hu felhasználóinak

nevében -amit mondjuk VRFY parancs segítségével szerzett meg, mert az aldozat.hu rendszergazdája sem tud rendesen mail szervert konfigurálni- leveleket küld létező illetve többségében nem létező e-mail címekre. Melyiknek mi értelme van? Egyrészt a legtöbb felhasználó nem fog rájönni, hogy a levél nem az aldozat.hu felhasználóitól jött és mondjuk feljelenti őket vagy őket jelentik be open relaynek. Ez természetesen csak a megérkezett, tehát valós e-mail címekre érkező levelekre vonatkozott. A direkt nem létező címre küldött levelek vissza fognak pattani, és az aldozat.hu felhasználóinak jön egy jelentés, hogy az általuk küldött levél címzett ismeretlensége folytán nem kézbesíthető. Ha ez elég nagy mennyiségben fordul elő, akkor egyrészt fennáll a fájlrendszer megtelítése, ha kvótát alkalmazunk ez ellen akkor pedig a felhasználó postafiókja telik meg és ez esetben sem tudja a számára fontos leveleket fogadni. Az érvényes címzettek fiókját is meg lehet telíteni és hasonló lesz az eredmény. Ebből talán látszik, hogy nem biztos, hogy a felhasználóink túl boldogok lesznek és vagy más szolgáltató vagy más rendszergazda után néznek.

# 5. Elméleti rész – összefoglalás

Remélem sikerült meggyőznöm mindenkit a dokumentáció és a protokollok ismeretének szükségességéről. Néhány egyszerű szabályt betartva és a levelezésben részt vevő alkalmazások helyes megválasztásával a hibák és támadási pontok elkerülhetőek. Az előadás második része a gyakorlati alkalmazást vizsgálja meg, a postfix *MTA* használatát feltételezve. A gyakorlati rész nem egy teljes konfigurációt, hanem az ismertetett pontok egy részének elkerülését segíti elő.

# 6. Gyakorlati megoldások

Nézzük a gyakorlati alkalmazást. Ez feltételezi, hogy tudjuk egyedül is teelpíteni az alaklamzást és nem vagyunk lusták elolvasni a dokumentációt. A postfixet chroot környezetben való futtatásra fogjuk beállítani, ezzel is növelve a biztonságot. Ehhez nincs másra szükségünk, mint egy feltelepített postfix-re és a master.cf fájl ismeretére. A master.cf egyik oszlopa a *chroot* nevet viseli. Amely érték itt a *y* betű értékét veszi fel, az a chroot jailen belül indul. Mivel a chroot környezetben a postfix programjai számára a rendszer függvénykönyvtárai nem elérhetőek ezért azokat a fájlokat, melyek szükségesek a működéshez, be kell másolnunk a megfelelő helyre. A postfix forráskódja mellett megtalálható egy LINUX2 nevezetű szkript is. Ez elvégzi a szükséges munkát helyettünk. A Debian disztribúcióban található csomag már chroot környezetben való futtatásra lett felkészítve. Ne feledkezzünk el az indítószkript módosításáról sem. A LINUX2 szkript mindenesetre nagy segítséget nyújt.
Ha már telepítettük a postfixet, akkor a main.cf-ben a következő sort állítsuk be:

```
disable_vrfy_command = yes
```

Ezzel megakadályozzuk a címek felderítését a szerveren. Ahhoz, hogy korlátozzuk a küldhető és fogadható levelek méretét, a következő sort használjuk:

```
message_size_limit = 10240000
```

Ez a beállított alapérték. Bármikor megváltoztathatjuk a méretet, amit bájtban kell megadni.

Ami szintén hasznos lehetőség a postfix-ben, az a fejléc és a levéltörzs átvizsgálása. Ez nem klasszikus vírusírtó, de sokat tud segíteni az internetes férgek korában. A fejléc és a törzs vitzsgálatát úgyszintén a main.cf fájlban tudjuk beállítani:

```
header_checks = regexp:/etc/postfix/header
body_checks = regexp:/etc/postfix/body
```

A fájlok tartalma regurális kifejezéseket tartalmaz, de használhatóak Perl reguláris kifejezések is. Ekkor a *regexp* helyett *pcre*-t kell írni a konfigurációs fájlba. A szűréshez meg kell ismernünk a reguláris kifejezéseken kívül az e-mailek és a MIME kódolást sajátosságait is.
Az open relayek kiszűréséhez pedig a

```
maps_rbl_domains = blackholes.mail-abuse.org
```

sort állítsuk be. Ha akarunk más szervert is megadhatunk persze. Azonban előfordulhat olyan eset is, hogy be akarunk engedni valakit egy letiltott szerverről vagy éppenhogy ki akarunk tiltani egy, a spammerek listáján (még) nem szereplő szervert. Ehhez a postfix többszintű smtp szintű szűrését és a klasszikus *access* fájlt használhatjuk. Állítsuk be a main.cf-ben (Természetesen egy sorba írva):

```
smtpd_sender_restrictions =
           check_sender_access /etc/postfix/access
```

Ez már a küldő azonosításánál megfogja vagy éppen engedélyezi a leveleket a küldőtől. Az access fájl formátuma a következő:

```
artatlan@open.relay ACCEPT
bunos@domain.nev REJECT
@open.relay REJECT
```

A fájl formátum pontos megismeréséhez olvassuk el az access(5) man lapot.

# 7. Gyakorlati rész – összefoglalás

A gyakorlati rész nem a teljességre törekedett, hanem a *postfix*-et használó rendszergazdák segítésére törekedett.
Mindegyik mail szerverhez elég gondos dokumentáció tartozik, nézzünk utána az elméleti részben ismertetett problémák megoldásának, ha arra lehetőséget ad a szoftver.

# Weboldalkészítés GNU/Linux alatt

Fodor Orsolya <fodorsi@axelero.hu>

2001.08.29.

**Kivonat**

Az előadásom célja: a kliensoldali weboldal-fejlesztés folyamatának és a szükséges eszközök és szabványok bemutatása, kifejezetten GNU/Linux környezetben.
A hallgatóság megismerkedhet a weboldal kialakításhoz szükséges eszközök használatának alapjaival (pl. HTML-szerkesztőkkel,grafikai programokkal, az Office-csomagok beépített lehetőségeivel, hasznos segédprogramokkal), az alkalmazott (megvalósított) szabványokkal és az előadás végén betekintést kaphat a további lehetőségeket és tanulnivalókat illetően.
Mivel a rendelkezésre álló 45+15 perc csak a figyelemfelkeltésre, és a témába bevezetésre elegendő, az előadás végén összefoglalom, hogy egy webfejlesztőnek milyen programozási illetve szkriptnyelvi, grafikai és egyéb ismereteket kell elsajátítani ahhoz, hogy profivá válhasson, illetve hogy megtalálja a leendő feladatainak legmegfelelőbb programozási nyelvet.

## Tartalomjegyzék

**1. A honlaptervezés folyamata (áttekintés) 16**

**2. Hol tartunk most? 16**

**3. Hogyan tovább? (áttekintés) 16**

# 1. A honlaptervezés folyamata (áttekintés)

- Tervezési szempontok
- Anyaggyűjtés, a rendelkezésre álló források elektronikus feldolgozása (pl. fényképek szkennelése, szövegbevitel, fájlkonverziók stb.)
- Grafikai tervezés (pl. képek megrajzolása, feldolgozása, méretezése stb.)
- Kódolás, HTML-szerkesztők és hasznos segédprogramok használata
- Tesztelés több böngészőben (Netscape/Mozilla, Lynx, Opera, Konqeror stb.)
- Publikálás (pl. ftp-vel vagy weben keresztül)

Az előadás közben a hallgatókkal közösen megtervezünk és elkészítünk egy egyszerű céges honlapot. A grafikai tervezésnél kapcsolódunk Süveg Gábor: GIMP című előadásához.

# 2. Hol tartunk most?

- fontosabb támogatott szabványok az egyes fejlesztőeszközökben és böngészőkben
- magyar nyelv használata

# 3. Hogyan tovább? (áttekintés)

- kliensoldali ismeretek (pl. HTML, CSS, JavaScript)
- speciális lehetőségek és új technológiák (pl. WAP, Flash, XML, Java, streaming media)
- szerveroldali ismeretek (pl. Perl/CGI, PHP stb.)
- adatbáziskezelési alapismeretek (pl. MySQL, PostgreSQL)
- (web)szerver-üzemeltetési ismeretek (pl. Apache)

A web gyors változékonyságánál fogva nehéz lenne itt url-ek listáját, web-es olvasnivalót ajánlani, hiszen ami ma létezik, az holnap már nem biztos, ami ma érdekes és új, holnapra elavulhat. Emiatt azt a megoldást választottam, hogy az előadás témájával kapcsolatos, rendszeresen frissített listája megtalálható a *http://fodorsi.ini.hu* oldalon.

# A Linux vállalati alkalmazása

Kósa Attila

2001.08.29.

**Kivonat**

Tavaly is volt szerencsém ugyanebben a témában előadni, ezért most igyekeztem olyan anyagot összeállítani, amelyben nem szerepelnek az elmúlt évben már ismertetett cégek. Örömmel jelenthetem, hogy sikerült. Azért örülök neki, mert ez egyben azt is jelenti, hogy kis hazánkban is egyre jobban terjed a Linux.

## Tartalomjegyzék

**1. Fővárosi Oktatástechnológiai Központ** **18**

**2. Index.hu Informatikai Rt.** **18**

**3. Integrity Kft.** **19**

**4. Interware Kft.** **19**

**5. Mecsekfűszért Rt.** **20**

**6. Média 6 Rádió Szeged** **20**

**7. MT-Telecom Kft.** **21**

**8. Pécsi Tudományegyetem Állam- és Jogtudományi Kar** **21**

**9. Pécsi Tudományegyetem Linux Konzultációs Központja** **23**

**10. Prím Communications & Média Rt.** **24**

**11. Vas Megyei Bíróság** **25**

# 1. Fővárosi Oktatástechnológiai Központ

A használt hardver egy HP NetServer LH3, egy darab Pentium II-es, 350 MHz-es processzorral, 256 MB RAM-mal felszerelve. Az operációs rendszer számára egy 4 GB-os SCSI-UW merevlemez biztosítja a helyet, az adatok pedig 5 darab 9 GB-os SCSI-UW merevlemezre vannak bízva, amelyek RAID-5-be vannak szervezve. A hálózati kapcsolatot egy 3Com 3c509B-TX hálózati kártya biztosítja. Slackware disztribúción a következő feladatokat oldották meg:

- belső fájlszerver – SaMBa,
- webszerver – Roxen Challenger,
- levelezőszerver – sendmail,
- proxy-szerver – squid.

Ezek mellett még apróbb munkái is akadnak a gépnek: DNS-szerver, nyomtatószerver, tűzfal, dialup-szerver. Az általa kiszolgált felhasználók száma körülbelül 30.

# 2. Index.hu Informatikai Rt.

A *http://index.hu/* címen található webszerver tulajdonképpen 4 gépből áll jelenleg. 4 darab Compaq dl360-as gépből alkottak clustert, melyek mindegyike a következő alkatrészekből épül fel: két darab Pentium III-as, 800 MHz-es processzor, 512 MB RAM és gigabites hálózati kártya. Merevlemez nincs egyik gépben sem! Operációs rendszerüket és a szükséges programokat a fájlszerver megosztott meghajtóiról töltik be.

A fájlszerver egy Compaq ml530-as gép, egy darab Pentium III-as 933 MHz-es Xeon processzorral, 1 GB RAM-mal, 8 darab 18 GB merevlemezzel és gigabites hálózati kártyával. A merevlemezek megoszlása a következő: 2x18 GB tükrözve az operációs rendszernek, és 6x18 GB RAID5-be szervezve az adatoknak. NFS-en keresztül szolgálja ki a webes felületet biztosító gépeket.
Egy adatbáziskezelő-szerver is van a „csapatban”, egy Intel sitka, két darab 550 MHz-es Xeon processzorral, 1 GB RAM-mal, 5 darab 9 GB-os merevlemezzel. A használt adatbázisok mérete is tekintélyes: körülbelül 4 GB egy-egy tábla mérete, a rekordszám 3 millió körül van. Az is érdekesség, hogy ez rácáfol arra a hiedelemre, hogy Intel platformon nem lehet 2 GB-nál nagyobb fájlokat létrehozni. Létezik megoldás erre a feladatra is.

A clusterre visszatérve: tudomásuk szerint Magyarországon az egyetlen fellelhető Cisco localdirectort ők használják. Terheléselosztást végez a cluster 4 gépe között. Attól függően, hogy megy-e az adott webszerver és mekkora a terhelése vagy a válaszideje, osztja ki a beérkező kéréseket. Így elvileg optimális a gépenkénti terhelés, és egy gép kiesése nem látszik a rendszer működésében. Amennyiben valamilyen okból kiesne egy gép a clusterből, pillanatok alatt ki tudnák cserélni, és egy floppy segítségével be is tud bootolni a fájlszerverről.

A webes felület előállítására és kiszolgálására Apache webszervert használnak php, perl, MySQL és Oracle kiegészítésekkel.

A felhasználók száma körülbelül napi 500.000. Ez nagyjából 0,8-1 GB adatmennyiségnek, és 3 millió találatnak (hit) felel meg.

A merevlemezek RAID-be szervezéséhez Mylex RAID vezérlőkártyákat használnak. Minden gépen Debian disztribúciót alkalmaznak.

Figyelemreméltó, hogy a régi rendszer bizonyos részei is Linuxokon futottak. A squid proxyk és az adatbáziskezelők, valamint az egyik webkiszolgáló is Linuxos (volt).

# 3. Integrity Kft.

A 2000. évi Középiskolai Felvételi Rendszer Internetes adatgyűjtő és feldolgozó szerveroldali moduljait SuSE Linux szervereken futtatta az Integrity Kft. Három független Internet-kapcsolattal rendelkező szerver fogadta a több, mint ezer iskolából érkező adatokat, valamint egy Java-alapú adatbázisalkalmazást futtató szerver dolgozta fel a 400.000 rekordnyi adatot. Minden szerveren Linux futott, a gépek IBM Netfinity 3000-es és 3500-as szerverek voltak. A rendszer hibátlanul teljesített, ezért – a tervek szerint – a jövőben is Linuxon fog futni.

Egy másik feladatuk az INI aldoménátirányítási és regisztrációs szerver, a legnépszerűbb hazai ilyen alkalmazás fő kiszolgálója egy IBM Netfinity 5500-as gép, 512 MB RAM-mal. Ezen a gépen – Linux alatt – Java alkalmazások és SQL adatbáziskezelő fut. A Netfinity 5500-as magas rendelkezésre állást biztosító megoldásait a Linux a hotswap PCI buszok kivételével támogatja, idén azonban várhatóan ez a támogatás is megjelenik.

Egy nemrég indult project keretében közérdekű – elsősorban államigazgatási vonatkozású – információkat tartalmazó honlapok és adatbázis-alkalmazások indulnak Linux szervereken. A sorozat első tagja a *www.kozjegyzo.hu*, további tagjai a közeljövőben jelennek majd meg. A terv az, hogy az év végére az Interneten át el lehessen érni a közjegyzők, a hivatalok, bíróságok és ügyvédek adatait.

Általában nem túl erős gépeket alkalmaznak, főként IBM Netfinity 3000-es, 3500-as gépeket, legerősebb gépük a fent említett Netfinity 5500-as. Inkább több kisebb gépre igyekeznek szétosztani a feladatot – ez is a megbízhatóságot és a stabilitást növeli. Adatbázis és alkalmazáskiszolgálási célokra szinte kizárólag Linuxot használnak. Az Integrity több, mint 10 Linux-alapú gépe, mint webes alkalmazáskiszolgáló, adatbáziskezelő, Internet kiszolgáló működik. Tűzfal, proxy-szerver, fájlszerver és munkaállomás céljára is alkalmaznak Linuxot a következő szoftverekkel: adatbáziskezelés – PostgreSQL, Interbase, MySQL; fájlszerver – SaMBa, nfs; proxy-szerver – squid; tűzfal – TIS; levelezési listák – mailman; webszerver – Apache (Roxen Challenger csak tesztelésre), Deneb (saját fejlesztésű, Java-ban készült). Vannak saját fejlesztésű alkalmazásaik, melyeket Java-ban, Perl-ben, PHP-ben és C-ben írtak.

Különlegesebb területeken is alkalmaznak Linuxot: van egy beléptető rendszerük, amellyel egy vagy több Linuxos gép, akár több ezer kártyás, PIN-kódos, ujjlenyomatolvasós helyet képes vezérelni. A Linuxot megbízhatósága és kis hardverigénye miatt választották ki. A szoftver C nyelven (vezérlés) és PHP-ban (webfelület) íródott. Folyamatban vannak különböző mérő és kapcsoló fejlesztések, melyeket Linux rendszerre alapoztak.

# 4. Interware Kft.

Compaq Proliant illetve AlphaServer DS20, ES40-es gépeket használnak Debian Linux alatt. Legnagyobb gépük egy Alphaserver ES40, 1 GB RAM-mal és 36 GB merevle-

mezzel. Ezen jelenleg (átmenetileg) csak körülbelül 5.500 felhasználó levelezése fut, de komolyabban is igénybe kívánják venni a közeljövőben. Ugyanis gyakorlatilag sokszorosan túl van méretezve mind processzor, mind memória oldalról. Az átlag szerver Compaq Proliant DL380, 256 MB RAM-mal, körülbelül 36 GB merevlemezzel. A Linuxra bízott feladatok a következők: proxy-szerver – squid, webszerver – Roxen Challenger, levelezőszerver – exim + Courier + IMHO, hálózatfelügyelet – snmpd + mrtg, felhasználók menedzselése – MySQL adatbáziskezelőn keresztül történik, DNS szolgáltatás – bind. Körülbelül 5.500 felhasználót szolgálnak ki a Linuxok. Nagyon sok saját fejlesztésű webes szoftvert használnak, melyeknek az előállítását megkönnyítette a Linux alatt elérhető programozási nyelvek és eszközök bőséges tárháza. Az összes szerver fontosabb adatainak biztonsági mentése jelenleg külön szerverre, RAID5-be kötött merevlemezekre történik hálózaton keresztül (rsync segítségével), de tervezik DLT vásárlását, és ha sikerül, akkor átállnak a szalagra történő mentésre.

# 5. Mecsekfűszért Rt.

Egy Epox KP6-BS alaplapban lévő 2 darab 550 MHz-es Pentium III-as processzor, 2 darab Promise Ultra66 IDE vezérlővel, 256 MB RAM hajtja azt a gépet, amelyben 2 darab 6,4 GB-os Western Digital merevlemez helyezkedik el RAID1-be kötve a rendszer számára, és további 2 darab 20 GB-os IBM merevlemez szintén RAID1-be kötve az adatoknak. SuSE Linux operációs rendszer gondoskodik a következő programok futtatásáról: főkönyvi rendszer – Micro Focus Application Server; nagykereskedelmi rendszer – Sea-Change 2 (iBCS2-vel futtatva); Cobol fejlesztői rendszer – Micro Focus Object Cobol Developer Suite. Az eddigieket körülbelül 50 felhasználó használja. Mostanában pedig a StarOffice szerver funkcióit próbálgatják, körülbelül 5-7 felhasználó számára. Kisebb alrendszerek futnak még a rendszeren – Dataflex for Intel Unix segítségével. A programok 200-600 MB-os fájlokkal dolgoznak. Fájlrendszerként reiserfs-t használnak.

# 6. Média 6 Rádió Szeged

Két telephelyet (Szeged és Szentes) köt össze két Suse Linux, amelyek 166 MHz-es Pentium processzorral, 32 MB RAM-mal és 2 GB-os merevlemezzel felszerelt számítógépen futnak. Mindkét gépen van webkiszolgáló – Apache, levelezőkiszolgáló – qmail, ftpkiszolgáló – proftpd és ssh a távoli eléréshez. A két telephely között a biztonságos adatforgalom biztosításához vpn-t alakítottak ki. Szegeden körülbelül 40 felhasználója van a Linuxos szervernek, Szentesen 10. A gépek 1,5 éve üzemelnek, a kiesés maximum 10 óra volt ez idő alatt. Ennyi is csak a kernelfrissítések miatt, illetve kapcsolati problémák miatt jött össze.

Az üzenetrögzítő feladatát is egy Suse Linuxra osztották. Ebben a gépben egy 133 MHz-es Pentium processzor ketyeg, 32 MB RAM-mal és 2 GB merevlemezzel megtámogatva. Csak szoftveres megoldást nem találtak erre a munkára, ezért a következőt csinálták: a vgetty call programjához készítettek egy parancsfájlt, ami egy hangkártyán (egy GUS MAX-on) lejátsza az üdvözlő üzenetet, majd felvételre kapcsol, és 30 másodpercig rögzít. A modem kimenete rá van kötve a hangkártya bemenetére, a hangkártya kimenete pedig egy olyan áramkörhöz csatlakozik, ami ha a bemenetén jel van, akkor – egy relé segítségével – párhuzamosan egy trafón keresztül csatlakozik a telefonvonalra. Így egy broadcast minőségű üzenetrögzítőt sikerült készíteniük.

A rögzített üzenetek a SaMBa fájlkiszolgálón keresztül érhetők el a kívánságműsor szerkesztője számára. Biztonsági mentés nem készül ezekről az üzenetekről, miután lementek az adásban törlésre kerülnek.

# 7. MT-Telecom Kft.

3 darab Linuxos szerver van a cégnél, mind a három Debian Linuxot futtat.

1. Fájlszerver – 400 MHz-es Pentium II-es Celeron processzor, 128 MB RAM, 1 darab 4 GB-os és 2 darab 13 GB-os merevlemez, szoftveres RAID-del. A használt alkalmazások:
   - fájlszerver – SaMBa,
   - fax-szerver – Hylafax,
   - DNS-szerver – bind,
   - DHCP-szerver – DHCP.
2. Tűzfal – 400 MHz-es Pentium II-es Celeron processzor, 64 MB RAM. A használt alkalmazások:
   - proxy-szerver – squid,
   - pop3-proxy.

   Tervezik ftp-proxy és Socks használatát is.
3. Webszerver – 500 MHz-es Pentium II-es Celeron processzor, 128 MB RAM. A használt alkalmazások:
   - webszerver – Apache,
   - levelezőszerver – qmail + vpopmail.

   Tervezik ftp-szerver (proftpd) és Mapguide szerver kialakítását is.

Munkaállomásként is használnak Linuxot, irodai alkalmazások futtatására, valamint tesztelésre.

# 8. Pécsi Tudományegyetem Állam- és Jogtudományi Kar

5 darab Linux szervert üzemeltetnek:

1. Nyilvános webszerver
   - Celeron 366 MHz, 256 MB RAM, 2x15 GB merevlemez, 3C905B hálózati kártya;
   - Renegát I (Red Hat 6.1 alapú Linux);
   - Apache + stunnel;
   - wuftpd.

2. Maszkoló és router-szerver
    - Pentium 75 MHz, 32 MB RAM, 2x210 MB SCSI merevlemez, EEPRO-100, 3c905B és SMC-EZ hálózati kártyák, rack-kivitel (3U magas 19");
    - Red Hat 6.2 alapú Linux;
    - ipchains + ipportfwd + FWTK csomag;
    - bind (DNS-szerver).
3. CD-torony és nyomtató-szerver
    - Celeron 366 MHz, 128 MB RAM, 2x8 GB merevlemez, 3C905B hálózati kártya, 5 darab 8-szoros Sony SCSI CD-ROM;
    - Renegát II (Red Hat 6.1 alapú Linux);
    - SaMBa szerver;
    - NFS, YP-kliens.
4. CD-torony, HTTP proxy-szerver és X-alkalmazásszerver
    - Celeron 366 MHz, 128 MB RAM, 2x15 GB IDE merevlemez, 3C905B hálózati kártya, 4 darab 8-szoros Sony SCSI CD-ROM, 2 darab 40-szeres Plextor SCSI CD-ROM, 1 darab Sun DAT;
    - Renegát II (Red Hat 6.1 alapú Linux);
    - xdm X-szerver;
    - SaMBa szerver;
    - squid proxy-szerver;
    - NFS, YP-kliens;
    - grafikus alkalmazások:
        - qvwm, fvwm és kde ablakkezelők;
        - Star Office 5.1;
        - Netscape;
        - irc kliensek;
        - Complex jogtár (Kerszöv fejlesztésű);
5. levelezőszerver és fájlszerver
    - Pentium II 300 MHz, 256 MB RAM, 2x15 GB IDE merevlemez, 3C905B és SMC-EZ hálózati kártya;
    - Renegát II (Red Hat 6.1 alapú Linux);
    - sendmail;
    - Apache + stunnel (külső elérésre);
    - ssh;
    - NFS, YP-szerver;
    - SaMBa szerver;
    - APC UPS kezelés (APC hálózati szerver).

A hálózat: a szerverek között 100 Mbit/sec UTP hálózat van (3Com Superstack switchen keresztül). A kliensek felé többszörös csillagpont elrendezésű hálózat 100 Mbit/sec gerinc, és 10 Mbit/sec végponti sebességgel. Az internet-elérés üvegszálon történik, 10 Mbit/sec sebességgel. Az Internet továbbszolgáltatása 1 darab mikrohullámú brigde-en keresztül 2 Mbit/sec sebességgel, és 4 darab 56 Kbit/sec sebességű analóg modemen keresztül történik.

A kliensek:

- 80 darab Windows 95 és 98, Windows 3.11 az irodákban;
- 30 darab Windows 98 és Linux kliens a laborban;
- 12 darab X-terminál (NCD, Sun, DEC, vt2000 gyártmányok vegyesen) külön laborban;
- 10 darab vt320 soros terminál, valamint i386 terminálszerver, ez a folyosói levelezőrendszer.

A fentieket kiszolgáló személyzet 3 főből áll, 1 rendszergazda és 2 operátor. A felhasználók száma 1470, ebből körülbelül 200 aktív linuxos felhasználó van, a többi SaMBa illetve levelező felhasználó.

Rövidtávú fejlesztési tervek:

- hardver-oldalról
  - a belső szerverek és a tűzfal cseréje rack-kivitelű redundáns hardverre;
  - a modemes behívószerver bővítése 12 darab 56 Kbit/sec sebességű modemre.
- szoftver-oldalról
  - kettős – port és protokoll tűzfalrendszer;
  - SQL-alapú logolási rendszer, PostgreSQL és syslog segítségével.

# 9. Pécsi Tudományegyetem Linux Konzultációs Központja

Helyileg a Természettudományi Kar Informatika és Általános Technika Tanszéken található ez a központ. Három laborban kiszolgálói feladatokat látnak el Linuxok. Minden laborban 12 darab Sun Sparcstation X-terminál van, ezeket szolgálja ki a 3 gép, amelyek két 600 MHz-es Pentium III processzorral, 512 MB RAM-mal és 20 GB merevlemezzel vannak felszerelve. Mindhárom gép alkalmazás-kiszolgálói feladatokat lát el, tehát ezeken futnak a munkaállomásokon használt programok.

Fejlesztési terveik is vannak, a közeljövőben szeretnének beszerezni három Sun gépet 400 MHz-es processzorral és 512 MB RAM-mal. Az elképzelések szerint ezeken is Linux fog futni, fájlkiszolgálói és webkiszolgálói feladatokat ellátva. A tűzfal mögött ezen gépek egyikére várna a felhasználók nyilvántartása, a jelenlegi álláspont szerint NIS segítségével. Az elektronikus levelezés is ezekre a gépekre van tervezve, webes felületen, valamint pine segítségével. A távlati tervek között szerepel a 2001-es évben egy ötven fős labor létrehozása, és ECDL tanfolyamok tartása Linux alatt. A tananyag kidolgozása még hátravan, az akkreditáció még nem indult el.

A munkaállomásokat jelenleg alapfokú Linux-tanfolyamok és (február elejétől) rendszergazdai tanfolyamok tartására használják, valamint az oktatásban néhány dologra – például C nyelv oktatására. A jövőben szakprogramok (elektronika, térképészet, fizika, matematika) is fognak futni, a Star Office (vagy talán az Open Office) lesz az irodai programcsomag, és természetesen informatikus-oktatás is lesz ezeken a gépeken.

Az alkalmazott operációs rendszer neve Renegát II, amely a Red Hat Linux általuk készített magyarítása.

# 10. Prím Communications & Média Rt.

5 darab szervert üzemeltetnek, melyek közül a legkisebb egy Pentium II 300 MHz-es processzorral, 256 MB RAM-mal, SCSI merevlemezzel, a legnagyobb pedig két Pentium III 800 MHz-es processzort, 512 MB RAM-ot és 70 GB SCSI merevlemezt tartalmaz. Ez utóbbiak egy Intel vezérlő segítségével hardveresen RAID5-be vannak kötve.

Minden szerveren Debian GNU/Linux fut, és a következőket nyújtja:

- internetes szolgáltatások
  - dinamikus, adatbázis-alapú hír-rendszer, cikkadatbázissal és kapcsolódásokkal;
  - webmail rendszer (PrímPosta);
  - weboldal-generátor (weboldal.com);
  - fórum;
  - „internetes” folder (mappa), képek és más fájlok tárolására a szerverükön;
  - egyéb, adatbázis-alapú szolgáltatások;
  - Adserver – a reklámok kiszolgálását végző szoftver;
  - pop3 és levelezőszerver;
  - DNS-szerver;
  - rendszermonitorozás.
- a cégirodában
  - ip maszkolás;
  - proxy-szerver;
  - levelezőszerver.

A fentiek megoldására a következő főbb programokat alkalmazzák: exim, bind, Apache, PostgreSQL és PHP4. A biztonsági mentések az Amanda nevű szoftver segítségével történnek.

A felhasználók számát a webes szolgáltatás miatt nehéz pontosan megmondani. A Prím-rendszerben regisztrált felhasználók száma meghaladja a 80 ezret. A PrímPostát több, mint 10 ezer ember használja rendszeresen.

Gyakorlatilag az összes PHP szkript saját fejlesztésű, amelyekhez kiváló fejlesztőkörnyezetet biztosított a Linux az eszközeivel. Mivel internetes szerverekről van szó, már a munka legelején világossá vált, hogy a Linux megfelelő eszközöket tud nyújtani a kívánt célok eléréséhez. Az elsődleges cél tehát nem az volt, hogy ingyenes rendszert építsenek, hanem a használhatóságon volt a hangsúly.

# 11. Vas Megyei Bíróság

Az informatika a kilencvenes évek elején jelent meg a bíróságokon. 1998-ig – az Országos Igazságszolgáltatási Tanács megalakulásáig – a bírósági informatikai fejlesztést a megyei bíróságok többé-kevésbé önállóan végezték, általában a maradékelvet követve: volt fejlesztés, ha maradt rá pénz. (A szabályt erősítő kivétel a cégbíróságok országos hálózatának központi fejlesztése.)

1992 végétől a Vas Megyei Bíróságon – az azidőtájt megszokott DOS-NetWare hálózat helyett – SCO UNIX alapú hálózatot helyeztek üzembe. Azért döntöttek így, mert a DOS-os világ korlátai már kezdtek kirajzolódni, ugyanakkor megfigyelhetőek voltak az Internet kibontakozására utaló első jelek. Célszerűnek látszott egy eleve nagyobb lehetőségeket engedő, ugyanakkor komoly múlttal, referenciákkal rendelkező, de elérhető árú rendszer megismerése és megszokása. Természetesen azért követelmény volt a már meglévő szoftverek illeszthetősége is. Úgy ítélték meg, hogy választásukkal nem szűkítették le a mozgásterüket: mind a DOS mind a UNIX világa nyitott előttük.

A DOS-Windows 3.1 alapú klienseket eleinte SCO-specifikus program (PC-Interface) kapcsolta a szerverhez. Később, a kliensek számának növekedésével ingyenes (részben nyitott forráskódú) szoftverek használatára tértek át. 1996-ig üzemelt ez a rendszer, az SCO lényegében fájlszerver, nyomtatószerver és CD-szerver (Complex Jogtár) funkciót látott el. Közben egy hálózatbővítés eredményeként egy másik SCO szervert is üzembe helyeztek úgy, hogy a két szerver routerként is funkcionált, 3 koax ethernet szegmenst ellátva.

Linuxszal 1994-től kezdtek el foglalkozni, egy magánúton beszerzett Yggdrasil disztribúciót tartalmazó CD segítségével (0.99.13-as kernel, X11R5, Postgres 4.1 experimental ELF loader…). Eleinte kliensként próbálgatták, de hamar nyilvánvalóvá vált, hogy kezesebb, mint az SCO. 1996-ban jött el az ideje annak, hogy a szerverekre Linux kerüljön, már Slackware disztribúció, 1.2-es sorozatú kernellel. (Mindeközben a bíróság gazdasági hivatalán belül működött egy – az idő múlásával egyre „szürkülő” – NetWare hálózat 8 géppel. Ide UN*X nem tehette be a lábát…)

Gyökeres változás 1998 őszén következett be. Ekkor minden megyei bíróság (az addig informatikára jutott összegekhez képest) komoly központi forrásokhoz jutott, amelyet a (megye)székhelyi bíróság informatikai fejlesztésére kellett fordítania. Úgy döntöttek, hogy a maximális költséghatékonyság elérése érdekében alapvetően Linuxos rendszert alakítanak ki. Ekkorra már kezdett beindulni a „Linux-úthenger”, egyre több mértékadó folyóirat, cég, fórum volt kénytelen reagálni a Linux-jelenségre, ezáltal kezdték igazolva látni korábbi UNIX irányú döntésük helyességét. Végül a rendelkezésre álló 10 millió forintból a következőt sikerült kialakítani:

- 36 végpontos tisztán switchelt UTP hálózat (addig házilag barkácsolt koax ethernetet használtak mindenhol);
- 1 darab szerver (10 GB UWSCSI merevlemez, CD író, CD olvasó, 512 MB RAM, 24 GB DAT);
- 30 darab diskless (merevlemez nélküli) PC (K6-2/300, 64 MB RAM, boot eprom-os NIC, 15”-os monitor);
- 1 darab hálózati nyomtató (PostScript, A3/A4, 32 lap/perc, duplexer, 3000 lapos adagoló);
- 10 darab személyi lézernyomtató (8 lap/perc).

A topológiát úgy alakították ki, hogy a leíró/kezelő irodákba 1-1 nyomtatót közösen használnak az ott dolgozók. A bírói szobákban nincs nyomtató, de a hálózati nyomtató mindenki számára elérhető (elsősorban a jogszabályok nyomtatására a CD-s Jogtárból).

Szoftverként a SuSe disztribúcióját alkalmazták, két okból: egyrészt – megítélésük szerint – a leginkább támogatta az új videovezérlőket, másrészt a bevezetett ApplixWare irodai szoftver mellé ezt kapták.

A klienseket alapvetően kétféleképpen lehetne használni: X-terminálként, illetve X-es munkaállomásként, attól függően, hogy a felhasználói programok a szerveren vagy a kliensen futnak. Ők ez utóbbit választották: így egyrészt ki van használva a kliens processzorának teljesítménye, másrészt csökken a szerver terhelése, harmadrészt a lokális nyomtatás így nem jelent problémát. Minthogy még DOS-os szoftvereik is vannak, dosemu-t muszáj használniuk, ami szintén ilyen elrendezést kíván. Hátrányként el kell viselni a swap hiányát és a nagyobb LAN-sávszélesség-igényt.

A következő (több részből álló) nagyobb beruházásra 1999 őszén került sor. Ekkor már – a központosított közbeszerzés keretén belül – közvetlenül jutott a bíróság hardver- és szoftvereszközökhöz. Megkérték a szállító céget, hogy tesztelhessük az általuk felkínált gépet a működő rendszerben: kiderült, hogy az Intel alaplap (valószínűleg BIOS problémák miatt) nem volt hajlandó kezelni a boot-epromos hálózati kártyát. Így tehát az eredeti konfigurációt módosítani kellett: másik (olcsóbb) alaplapot kértek a gépekbe, merevlemez, floppy, CD és Windows nélkül, viszont gyorsabb processzorral, nagyobb memóriával, sőt – hogy kitöltsék a forintkeretet – még két plusz gépet és tíz lézernyomtatót. (Az eredeti rendelésben egyáltalán nem volt nyomtató.)

(Két érdekes történet ebből az időből: a szállító értékesítési menedzsere eleinte nem tudta elhinni, hogy nem kell minden géphez (diskless géphez!) OEM Linux...

Amikor kiderültek az alaplappal kapcsolatos problémák, nehéz volt rábeszélni a szállítót a cserére. Komolyan fölmerült bennünk, hogy ha nem megy a boot eprom, kérjünk floppy meghajtót a gépekbe, de nyílásával a gép belseje felé fordítva, egyszer s mindenkorra bennehagyva a boot-floppyt.)

Összesen 6 darab szervert is kaptak volna a vidéki bíróságok ellátására (1 nagyobbat, 5 kisebbet). Mivel csak két vidéki bíróságon van hálózat, az 5-ből 3-at „becserélteket” a szállítónál: helyette kérték a bíróság két szombathelyi épülete között mikrohullámú összeköttetés kiépítését. A routerek természetesen kiöregedett 486-osok, Linuxszal... A „nagy” szervert a megyei bíróságon állították munkába, az addigi szerver tartalékként szolgál illetve az archiválásokat végzi (naponta DAT-ra, nagyobb periódusok végén, szoftverfrissítés előtt CD-re is). Minthogy minden adat a szerveren van, a napi mentést egy lépésben, egyszerű scripttel lehet elvégezni.

Ezzel a beruházással lényegében Linux-alapon áll a bírósági informatikai rendszer. Egyúttal az egyre kínosabbá váló jogtisztasági problémát is megoldották. (Az ApplixWare mellett fokozatosan vezetik be a StarOffice-t.) Időközben saját erőből megvalósították az Internetelérést ISDN-en keresztül, sőt a körmendi és a sárvári bíróságok felé is van dedikált ISDN adatkapcsolat.

1999 végén kiépült a bíróságok országos hálózata is egy Phare projekt keretén belül: minden megyei bíróság kapott 16 kbit/s garantált sávszélességű Frame Relay kapcsolatot az OITH irányába, egy RS/6000 szervert (AIX + Apache + Samba + Oracle InterOffice (14 user)) a belső levelezés számára és 14 IBM PC-t Windows NT-vel. Mivel az installált levelező rendszer működése és kényelme némi kívánnivalót hagyott maga után, a 14 PC-t is inkább a lokális rendszerbe integrálták, kiiktatva az NT-ket. Az országos hálózat IP számaival való ütközés miatt a lokális rendszert egy (természetesen Linuxos) tűzfallal választották le az RS/6000-ről, úgy, hogy az azon még futó InterOffice szükség esetén elérhető legyen.

Jelenleg a szerveren a „szokásos” szerver szoftverek futnak: bootpd, nfs szerver, apache (belső használatra), squid, bind, qmail, postgresql, mars_nwe, stb. A közeljövőben fognak kapni vírusirtó szoftvert is, ennek szerepe még nem tisztázott... A szerver szolgáltatja a Jogtárat is, de nem CD-ről, hanem a merevlemezről a gyorsaság érdekében. Az 57 regisztrált felhasználó közül általában egyidejűleg kb. 30-35-öt szolgál ki, kellemesen alacsony (0.1 körüli) terheléssel. Folyamatosan történik a felhasználók oktatása az internetes technikákra, a Fővárosi Bíróság jóvoltából valamennyi felhasználó külön költség nélkül e-levelezhet. Különlegességként: a titkos anyagokat cfs (crypto file system) segítségével kódolt formában a szerveren tárolják. (Kódolás hiányában mobil adathordozót kéne használni és páncélszekrényben tárolni.)

A klienseken elsősorban szövegszerkesztés folyik, valamint a nagyrészt Clipper-alapú nyilvántartások vezetése. Ez utóbbiak továbbra is dosemu-ból érhetők el, de – egy-két szoftver esetén – mód van a kliens direkt DOS-os indítására is. Mindkét esetben a kliens gép egyúttal mars_nwe kliens is. Már van néhány programjuk, ami a Linuxos környezetre készült. Ezek egy része önálló fejlesztés, másik része meglévő DOS program helyettesítője illetve kiegészítője. (Ez utóbbi esetekben a .dbf formájú adatokat átteszik PostgreSQL alá, és úgy használják fel.)

Terveik között szerepel az LDAP bevezetése, pl. authentikáció céljára. Minthogy a diskless kliensek mind ugyanazt a passwd és shadow fájlt látják, az LDAP alkalmazása valójában nem sürgős, de szeretnének fölkészülni az egyéb LDAP-t igénylő szoftverek fogadására illetve készítésére. Tervezik további DOS alapú szoftverek kiváltását Linuxos illetve multiplatformos szoftverekkel, új programok készítését a bírósági ügyvitel még ellátatlan területeire, a vidéki bíróságok adatkapcsolatának kihasználását (e-mail, adatbázis elérés). Szeretnének áttérni a bootp-ről dhcp-re. A fejlesztések során elsősorban a Python-PostgreSQL párost használjuk. Tekintettel a roppant egyszerű és könnyen áttekinthető topológiára hálózatfelügyelő szoftvert egyelőre nem kívánunk használni.

A felhasználók többsége csak annyit tud a Linuxról, hogy más, mint a Windows. Korábban már többen használtak Windowst, az ő esetükben a grafikus felület használata nem jelentett problémát, viszont az alkalmazások „mássága” igen. Az első időkben előfordultak véletlen dokumentumtörlések, ilyenkor a felhasználók első reakciója sokszor az volt, hogy „rossz a gép”. Akiknek nem volt számítógépes tapasztalatuk, azoknál a legnagyobb problémát az egér kezelésének megtanulása és az írógépes múlt (az „1” és „l” karakterek, valamint a „0” és az „o” karakterek összekeverése, a „space” billentyűs szövegformázás) jelentette. Ők alapszintű oktatásban részesültek, és néhány nap alatt belejöttek. A „fortélyokat” pedig mindig egy-egy konkrét eset kapcsán tanulják meg. Korábban ugyanezek a problémák jelentkeztek a Windowsos környezetben is, úgy tűnik, hogy ezek a kérdések nem rendszerspecifikusak.

Magával az operációs rendszerrel csak az informatikusok találkoznak. Problémák abból szoktak adódni, hogy a .doc vagy .xls állományok bevitel után nem pontosan úgy néznek ki, mint egy Windowsos rendszerben. Ha nem lehet ettől eltekinteni, akkor általában más formában (például .rtf) kérik a küldőtől az adatokat. Nem okoz különösebb problémát a floppymeghajtók hiánya sem, mert kérésre bárkinek a floppyn szállított dokumentumait felteszik a rendszerre.

Az irodai rendszerként használt Applixware menürendszere magyarított, ezzel nincs gond. Szokatlan üzenet megjelenésekor (mindegy, hogy angol vagy magyar nyelvű), amikor a felhasználó bizonytalan a helyes válaszban, akkor inkább megkérdezi az informatikusokat.

# GNU/Linux fürtök a rendelkezésre állás és a teljesítmény növelésére

Kósa Barna <barna_kosa@hp.com>

2001.08.29.

**Kivonat**

A Linux még gyorsabb előretörését vállalati környezetben többek között két tényező hátráltatja: kevésbé ismertek a magas rendelkezésre állást biztosító fürt megoldások, és egyelőre a PC-s hardveren futó Linux nem tud versenyezni a nagy Unix szerverek skálázhatóságával. Mindkét problémára léteznek különböző, fürtözési technológiákra épülő megoldások. Ezek a technológiák már jól beváltak a Unix szerverek világában, és gyors ütemben honosodnak meg a GNU/Linux környezetben is.

## Tartalomjegyzék

**1. Bevezető** **30**

**2. Magas rendelkezésre állást biztosító fürtök** **30**
2.1. Megosztott diszkeket használó fürtök . . . . . 31
2.2. Megosztott elemek nélküli fürtök . . . . . 32

**3. Terheléskiegyenlítő fürtök** **33**
3.1. Virtuális szerverek . . . . . 33
3.2. Egyetlen nagy SMP gépnek látszó fürtök . . . . . 34
3.2.1. Beowulf fürtök . . . . . 35
3.2.2. Mosix fürtök . . . . . 36

**4. Párhuzamos feldolgozást biztosító fürtök** **36**
4.1. MPI alapú fürtök . . . . . 37
4.2. Parallel Virtual Machine fürt . . . . . 38

**5. Összefoglaló** **38**

# 1. Bevezető

A számítástechnikában a fürt (cluster) olyan, több gépből álló konfigurációt jelent, ami megsokszorozza az egyes gépek bizonyos tulajdonságát. A fürtözési technológiákat több szempont szerint is osztályozhatjuk:

- magas rendelkezésre állás (high availability),
- terhelés-kiegyenlítés (load balancing),
- párhuzamos feldolgozás (parallel processing).

A legtöbb fürtözési technológia egyszerre több kategóriába is besorolható.
A magas rendelkezésre állás elérésére két alapvető megoldás alkalmazható: teljesítmény-kiegyenlítő fürt (pl. web szerverek esetében) és a hagyományos high availability cluster, ami tartalék hardvert és közös elérésű diszkeket tartalmaz. Mindkét megoldásra számos nyílt forráskódú és kereskedelmi szoftver létezik. A magas rendelkezésre állást biztosító fürtök leggyakrabban adatbázis- és alkalmazásszervereket védenek. A Mosix és Beowulf megoldások olyan általános célú szerver fürtök kialakítását teszik lehetővé, amelyeken tetszőleges alkalmazásokat futtathatunk a terheléskiegyenlítés és a magas rendelkezésre állás minden előnyével.
A teljesítmény növelésére és a párhuzamos végrehajtás támogatására több nyílt forrású fürtözési megoldás létezik, például az MPI/LAM és a PVM. Ezek a megoldások nemcsak Linux alatt működnek, ezért heterogén környezetben is alkalmazni lehet őket. Ilyen módon igazi szuperszámítógépeket lehet létrehozni olcsó PC-s hardverből. A megfelelő szintű skálázhatóság előfeltétele a feladatok párhuzamosítása és a párhuzamos programozás. A párhuzamosítást főként a tudományos és technikai számítások területén lehet alkalmazni, de az általános vállalati környezetben is található jónéhány lehetőség.

# 2. Magas rendelkezésre állást biztosító fürtök

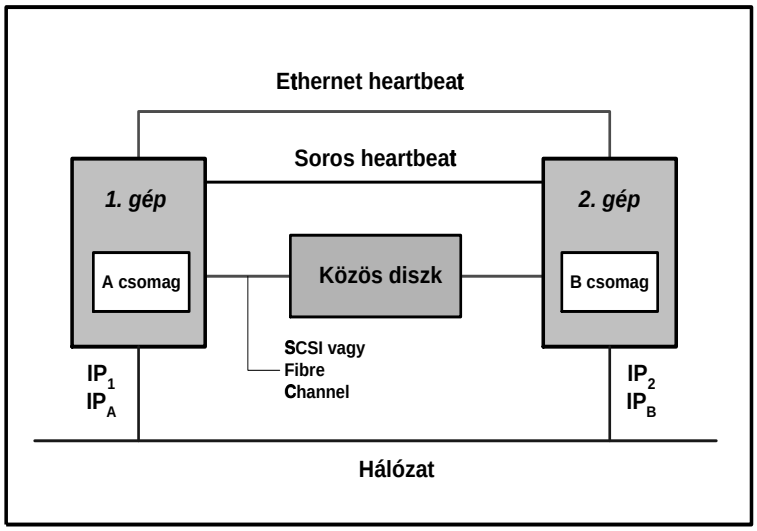
A magas rendelkezésre állást biztosító fürtök (high availability cluster) megfelelő hardver redundancia (több fizikai szerver) és szoftver megoldások alkalmazásával csökkentik az egyedi meghibásodási pontok (single point of failure) számát. A magas rendelkezésre állást biztosító fürtök elsősorban a hardverhibák ellen védenek, de bizonyos operációs rendszer- és alkalmazásszintű hibákat is ki tudnak küszöbölni. A magas rendelkezésre állást tovább lehet növelni a megfelelő környezet kialakításával (redundáns tápellátás és hálózat, környezeti feltételek).
A magas rendelkezésre állást biztosító fürtök működése az alkalmazások átkapcsolásán (failover) alapszik: a meghibásodott gépen futó alkalmazások átkapcsolnak a rendelkezésre álló tartalék gépekre. A magas rendelkezésre állást biztosító fürtöknek alapvetően két fajtája létezik :

- megosztott diszkeket használó (shared disk) fürtök,
- megosztott elemek nélküli (shared nothing) fürtök.

A magas rendelkezésre állást biztosító fürtök nem hibatűrő megoldások, mert meghibásodások esetén bizonyos szolgáltatások rövid időre megszakadhatnak. Hosszú távon viszont a rendelkezésre állás rendkívül magas értékeket érhet el.

## 2.1. Megosztott diszkeket használó fürtök

Egy tipikus magas rendelkezésre állást biztosító, megosztott diszkeket használó (shared disk) fürt konfigurációját az 1. ábra mutatja.




1. ábra

A fürt két gépbol áll és mindkettő aktív szerepet tölt be: alkalmazásokat futtat és kész átvenni a másik gép alkalmazásait. A fürt lényeges jellemzője, hogy a két gép egy közös SCSI buszon vagy Storage Area Network-ön keresztül éri el a megosztott diszkeket. A fürt megfelelő működését egy módosított kernel és démon processzek biztosítják. A fürt gépei bizonyos időközönként úgynevezett heartbeat jeleket küldenek egymásnak LAN-on (több hálózati kártyán is lehetséges) vagy soros vonalon keresztül. Amennyiben a bejövő heartbeat jelek az egyik gépen kimaradnak, a gép úgy veszi, hogy a másik gép nem működik és megkezdődik a fürt újraalakulási folyamata. Az újraalakult fürt már csak egyetlen gépből áll, amelyik átveszi a meghibásodott gép alkalmazásait. A meghibásodott gép újraindításkor automatikusan, vagy kézi vezérléssel csatlakozik a fürthöz. A fürt indítása, leállítása, átalakítása, az alkalmazás-csomagok indítása és leállítása egy adott gépen külön parancsokkal lehetséges.
A fürt megoldásnak helyesen kell kezelnie azokat a helyzeteket is, amikor a két gép között minden kommunikáció megszakad. Ebben az esetben egy speciális diszkterületet lehet használni a fürt átalakulási folyamatának vezérlésére.
A fürt gépein futó, magas rendelkezésre állást igénylő alkalmazásokat úgynevezett alkalmazáscsomagokba kell szervezni. Az alkalmazáscsomag minden olyan erőforrást tartalmaz, ami az alkalmazások működéséhez szükséges: fájlrendszerek, IP címek, futó programok (processzek). Minden egyes alkalmazáscsomaghoz hálózati kártyánként legalább egy mozgó IP cím tartozik. Ahhoz, hogy egy alkalmazást egy ilyen fürtön futtatni lehessen, meg kell felelnie néhány feltételnek:

- nem függhet a host névtől és semmilyen gép specifikus paramétertől (pl. MAC cím),
- az adatok a megosztott diszkterületen kell legyenek.

Minden alkalmazáscsomag számára létezik egy elsődleges gép és egy vagy több tartalék gép. Alapállapotban, amikor a fürt minden gépe működik, az alkalmazáscsomagok az elsődleges gépeken futnak. Amennyiben egy elsődleges gép meghibásodik vagy leáll, a rajta futó alkalmazáscsomagok automatikusan átkerülnek egy tartalék gépre. Az alkalmazáscsomagok fontosságának függvényében a tartalék gépen futó alkalmazásokat le lehet állítani, hogy megfelelő erőforrás (CPU, memória) jusson a kritikus alkalmazásoknak.
A fürt helyes működésének elengedhetetlen feltétele a megosztott diszkek biztonságos kezelése. Ezt nagyban segítheti a Logical Volume Manager (LVM) használata. A megosztott diszkeken ajánlott naplózó fájlrendszert használni (pl. Reiserfs, JFS, XFS). Ellentétben egyéb fürt megoldásokkal, a magas rendelkezésre állást biztosító, megosztott diszkekre épülő fürt megoldások esetében csak megfelelő, a hardver- és szoftvergyártók által minősített konfigurációk működnek helyesen. Ezért általában csak a jó minoségű, drága hardverkonfigurációk támogatottak. A fürt kialakítása és üzemeltetése nagymértékben egyszerűsödik, ha megegyezik a gépek hardver- és szoftverkonfigurációja. A gépek operációs rendszere és patch szintje azonos kell legyen.
A leggyakoribbak a két gépből álló fürtök, de a fürtszoftvertől függően a gépek száma ennél nagyobb is lehet.
A legismertebb Linux fürtszoftverek a már jól bevált Unix megoldások portolt változatai, például Hewlett-Packard MC/ServiceGuard, Silicon Graphics FailSafe, High-Availability.Com RSF-1, de nagyon népszerű a Mission Critical Linux Convolo Cluster megoldása is (*http://www.missioncriticallinux.com/*).
A shared disk fürtök magas ára és komplexitása miatt csak olyan alkalmazások esetében érdemes használni őket, amikor nem lehetséges több azonos szerver egyidejű működtetése (pl. web szerverek) vagy az adatok replikációja (pl. DNS szerverek). A leggyakoribb példa a shared disk fürtök által védett alkalmazásokra a különböző adatbázisszerverek. Érdemes megjegyezni, hogy az Oracle Parallel Server fut és támogatott is több GNU/Linux disztribúción is (például SuSE *http://www.suse.com/*).
A shared disk fürtök által az alkalmazások számára biztosított rendelkezésre állás nagymértékben függ az alkalmazásoktól. A meghibásodott gép észlelése és a fürt újraalakulása általában kevesebb, mint egy perc, de az alkalmazások újraindulása egy hiba után ennél jóval több is lehet. Megfelelő hardverkonfiguráció esetén (redundáns tápegységek, szünetmentes táp, RAID diszkrendszerek) Linux fürtökkel is elérhető 99,99%-os rendelkezésre állás.
Bár a shared disk fürtök elsődleges feladata a magas rendelkezésre állás biztosítása, az alkalmazáscsomagok kézi vagy automatikus mozgatásával el lehet érni egy bizonyos fokú terhelésmegosztást a gépek között.

## 2.2. Megosztott elemek nélküli fürtök

A megosztott elemek nélküli fürt a közös diszkeket használó fürt egy sajátos, egyszerűsített és ezért jóval olcsóbb változatának tekinthető. Mivel nincs megosztott diszkterület, jóval egyszerűbb a fürt felépítése és működése. A fürtöt alkotó gépek száma is jóval nagyobb lehet, mint a közös diszkeket használó fürt esetében.
A fürt gépei egymástól teljesen függetlenül működnek és egymás tartalékait képezik. Ezért az alkalmazások és a hozzájuk tartozó adatok mind az elsődleges, mind a tartalék gépeken meg kell legyenek. Amennyiben az adatok dinamikusan változnak, gondoskodni kell az adatok szinkronizációjáról (pl. *rsync* a fájlok szinkronizálására).
Hasonlóan a megosztott diszkeket használó fürtökhöz, a megosztott elemek nélküli fürtök gépei is heartbeat jelekkel kommunikálnak egymással, és virtuális (mozgó) IP

címek tartoznak az alkalmazásokhoz. Mivel nincsenek megosztott diszkek, a gépek közötti távolság tetszőlegesen nagy lehet. Így lehetoség nyílik katasztrófatűrő megoldások kialakítására.
A Linux High-Availability Project *http://linux-ha.org/* weboldalán részletes információ található a projekt keretében kifejlesztett megoldásokról (heartbeat, fake). Számos linuxos cég indított saját projektet ( RedHat Piranha *http://ha.redhat.com/*, VA Linux UltraMonkey *http://ultramonkey.sourceforge.net/* ) ilyen típusú fürtmegoldások kifejlesztésére.
A gyakorlatban a megosztott elemek nélküli fürtmegoldásokba beépítik a dinamikus terhelésmegosztást is, és ezek a megoldások az alkalmazások széles skáláját képesek futtatni (pl. fájl-, nyomtató-, web, ftp, mail szerverek).

# 3. Terheléskiegyenlítő fürtök

A terheléskiegyenlítő fürtök többnyire egyszerre oldják meg a magas rendelkezésre állás és a skálázhatóság problémáját.
A legegyszerűbb terheléskiegyenlítő fürtöt a DNS segítségével lehet megvalósítani. Ha egy host névhez több IP címet rendelünk és a time-to-live paraméter értéket kellően alacsonyra vesszük, akkor a névfeloldás gyakorlatilag véletlenszerűen fogja visszaadni a host névhez tartozó IP címeket. Ez megfelel egy primitív terhelésmegosztásnak. A megoldás nagy hiányossága, hogy nem biztosít semmilyen magas rendelkezésre állást, és nem veszi figyelembe a szerverek tényleges terhelését. A valós alkalmazások igényeinek megfelelően olyan terheléskiegyenlítő fürtmegoldások jelentek meg, amelyek egyetlen gépként mutatkoznak az alkalmazások számára. Alapvetően kétfajta terheléskiegyenlítő fürtmegoldásról beszélhetünk:

- virtuális szerverek, amelyek egy virtuális IP cím mögött több valódi szervert jelentenek
- egyetlen nagy SMP gépnek látszó fürtök

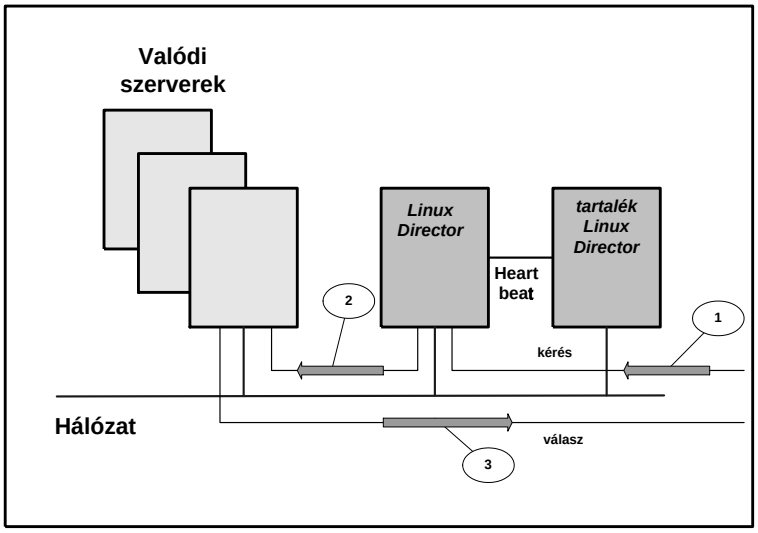
## 3.1. Virtuális szerverek

Ezt a fürttípust tekinthetjük egy ugyanarra a feladatra specializálódott szerverfarmnak, ami egyetlen szerverként jelenik meg a külső kapcsolatok számára. Az ilyen típusú fürt összes gépén azonos alkalmazások futnak.
A linuxos körökben legelterjedtebb terheléskiegyenlítő fürtmegoldás a Linux Virtual Server *http://www.linuxvirtualserver.org/*. Ez egyetlen virtuális IP cím mögé rejti a fürt valódi gépeit. A megoldás két részből áll: LinuxDirector és a valódi szerverek. A virtuális IP címre érkező kéréseket a LinuxDirector továbbítja valamelyik valódi gépnek. A virtuális szerver három módszert használhat az IP alapú terheléskiegyenlítésre:

- NAT
- IP Tunnelling
- Direct routing

Mindhárom módszer a Linux kernel módosítását igényli. A NAT módszer a teljes oda-vissza forgalmat a LinuxDirector-on keresztül valósítja meg. A másik két módszer esetében a virtuális szerverre érkező kéréseket a LinuxDirector szétosztja a valódi szerverek között, míg a válaszokat a valódi szerverek egyenesen a klienseknek küldik vissza.

Mivel a válaszok mérete nagyobb mint a kéréseké, ezek a megoldások jóval nagyobb átviteli sebességet tesznek lehetővé mint a NAT módszer.
A Linux Virtual Server több algoritmust is használhat a valódi szerverek terhelésének kiegyenlítésére. Az algoritmusok figyelembe veszik, hogy egy adott gép elérhető-e és hogy a gépen mekkora a terhelés. Így megvalósul a valódi szerverek magas rendelkezésre állása, mert a LinuxDirector csak a működő szerverek felé továbbítja a kéréseket. A terheléskiegyenlítés is működik, mert az újabb kérések a kevésbé terhelt szerverekhez érkeznek.
A Linux Virtual Server-nek a LinuxDirector az egyetlen egyedi meghibásodási pontja. Ezért ha egy teljesen HA megoldást akarunk, két LinuxDirector-t egy magas rendelkezésre állást biztosító fürtbe kell kapcsolni. Egy ilyen megoldással például egy rendkívül magas rendelkezésreállású és teljesítményű webszerver-farm alakítható ki. Mivel az összes valódi szerver ugyanazt a tartalmat szolgáltatja, a megoldást ki kell egészíteni egy hibatűrő fájlszolgáltatással. Erre a célra a Coda *http://www.coda.cs.cmu.edu/* elosztott és hibatűrő fájlrendszer az egyik legjobb megoldás.




2. ábra

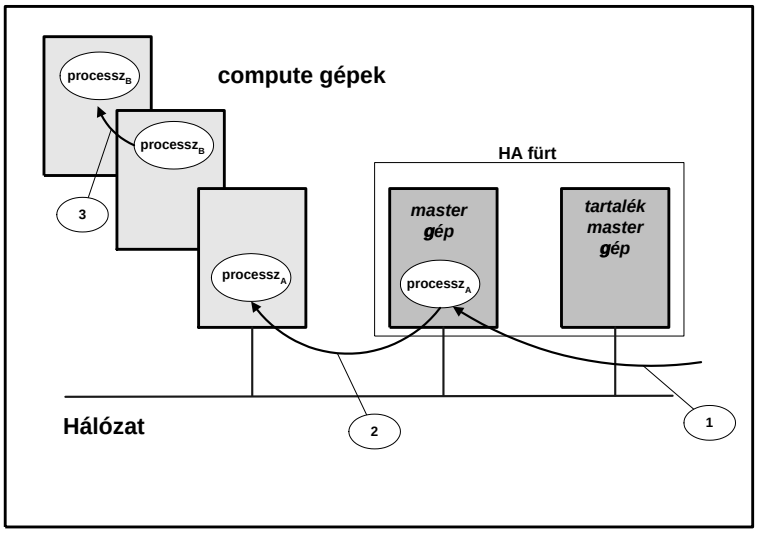
A 2. ábrán egyedi meghibásodási pont nélküli virtuális szerver fürt látható. A kliensről érkező kéréseket a LinixDirector fogadja és osztja szét a megfelelő valódi szervernek. A valódi szerver a választ közvetlenül küldi vissza a kliensnek.

## 3.2. Egyetlen nagy SMP gépnek látszó fürtök

Linux környezetben két olyan megoldás is létezik, amelyik egy több gépből álló fürtöt egyetlen nagy, SMP architektúrájú gépnek mutat a felhasználók és alkalmazások számára:
a Mosix *http://www.mosix.cs.huji.ac.il/* és a Beowulf *http://www.beowulf.org/*. A két megoldás működési elve nagyon hasonló és közel áll egy többprocesszoros gépéhez. Ezért az alkalmazások módosítás nélkül ki tudják használni a fürtmegoldás nyújtotta előnyöket. A CPU teljesítmény rendkívüli nagy skálázhatósága miatt ezeket a fürtöket a High Performance Computing kategóriába is sorolhatjuk.

### 3.2.1. Beowulf fürtök

A NASA berkeiben indult projektet jelenleg a Scyld Computing Corporation: *http://www.scyld.com/* folytatja. A Beowulf fürt legújabb változata nagymértékben egyszerűsítette a fürt installációját, működését és menedzsmentjét. A Beowulf terminológiát használva egy Beowulf fürt egy master gépből és több compute gépből áll. A compute gépek a master gépről bootolnak és a programok kódja a master gépen van tárolva. A compute gépek a saját merevlemezüket csak swap területként és adatok tárolására használják. Így garantált, hogy ugyanaz az operációs rendszer és az alkalmazások verziója a fürt minden egyes gépén.
A *Bproc*, a Linux kernel processzmenedzsment-bővítése, biztosítja, hogy a Beowulf fürt egyetlen rendszer képét mutassa. A *Bproc* lehetővé teszi, hogy a fürt gépein futó processzeket a master gépről lehessen látni és felügyelni. A processzek a master gépen indulnak és a Bproc segítségével átkerülnek egy megfelelő compute gépre. A processzek közötti szülő-gyerek kapcsolat és a job control érvényes marad a migrált processzekre. A Scyld Beowulf számos olyan szoftver komponenst tartalmaz, amivel hatékonyan lehet kezelni a fürt és a rajta futó alkalmazások működését. Ezek között érdemes megemlíteni az MPI és PVM Beowulf fürtre optimalizált változatát. Ezek a szoftver komponensek alkalmassá teszik a Beowulf fürtöt a hagyományos és a párhuzamos programok hatékony futtatására.
A Beowulf fürt muködésében a master gép jelenti az egyetlen egyedi meghibásodási pontot. A Beowulf következő verziója már több master gépet is tartalmazhat majd, és ezekből ki lehet alakítani egy magas rendelkezésre állást biztosító fürtöt.




3. ábra

A 3. ábra egy olyan Beowulf fürtöt mutat be, amelyikben nincsenek egyedi meghibásodási pontok. A processzeket a master gépen kell elindítani (közvetlen bejelentkezés vagy *remote shell*) majd a processz átkerül egy megfelelő compute gépre.

### 3.2.2. Mosix fürtök

A Mosix szoftvercsomagot a jeruzsálemi Hebrew University-n fejlesztették ki. A Beowulf-hoz hasonlóan a Mosix is módosítja a Linux kernelt. Mosix környezetben kétfajta gép létezik: szerver és munkaállomás. A szerverek mindig részei a fürtnek, a munkaállomások nem mindig. A nagysebességű hálózatba kötött gépekből nagyon rugalmasan több fürtkonfigurációt is ki lehet alakítani:

- server-pool - csak a kijelölt szerverek alkotják a fürtöt, a munkaállomások nem. Processzeket csak úgy lehet indítani, hogy bejelentkezünk egy szerverre.
- adatprive-pool - a munkaállomások beléphetnek a fürtbe illetve kiléphetnek a fürtből.
- half-duplex pool - a munkaállomások processzeket küldhetnek a szerverekből álló fürtre, de nem lépnek be.

A Mosix fürt működése teljesen transzparens az alkalmazások számára. Így a Mosix fürt lehetővé teszi tetszőleges szekvenciális vagy párhuzamos alkalmazás futtatását, függetlenül attól, hogy a processzek melyik gépen futnak és hogy más felhasználók mit csinálnak.
A Mosix által módosított kernel a gépek terhelése és a gépeken rendelkezésre álló erőforrások függvényében transzparens módon mozgatja a processzeket a gépek között. Ezáltal nagymértékben megnő a rendszer általános teljesítménye. Egy processz röviddel a létrehozása után átkerül a fürt legmegfelelőbb gépére. Ezután a módosított ütemező gondoskodik a processz mozgatásáról és rendszerterhelés optimalizálásáról.
A MOSIX ütemezője egy komplex algoritmust használ, ami figyelembe veszi a gépeken rendelkezésre álló erőforrásokat (CPU sebessége, memória mennyisége). A futó processzeket a QPS grafikus program segítségével lehet felügyelni.
A Mosix egy saját elosztott MFS fájlrendszert használ. Így a fürt összes gépe ugyanazt a fájlrendszert látja. Az MFS nagy előnye, hogy biztosítja az MFS fájlrendszer cache konzisztenciáját a gépek között. Az MFS megfelel a Direct File System Access (DFSA) szabványnak.

# 4. Párhuzamos feldolgozást biztosító fürtök

A párhuzamos feldolgozás (parallel processing) a modern számítástechnika egyik speciális területe. A nagy feladatokat sok apró hasonló feladatra bontva és párhuzamosan végrehajtva nagymértékben csökkenthető a programok futási idejét. A párhuzamos feldolgozás főként a rendkívül nagy számítási kapacitást igénylő tudományos feladatokra jellemző. Ezért ezt a területet High Performance Computing (HPC) névvel is jellemzik.
A párhuzamos feldolgozást nagymértékben támogatja a Massively Parallel Processors (MPP) architektúra. Az elosztott számítástechnika és a fürtözési technikák fejlődése lehetővé tette, hogy az olcsó PC hardverre épülő fürtök teljesítményben felvegyék a versenyt a náluk nagyságrendekkel drágább MPP vagy speciális architektúrájú szuperszámítógépekkel.
A párhuzamos feldolgozás során az egymással kooperáló feladatok adatokat kell cseréljenek egymás között. Az adatcserére számos modellt dolgoztak ki, ezek közül az egyik legnépszerűbb az üzenetátadás (message passing). Az üzenetátadás modell nagy

előnye, hogy egyaránt alkalmazható SMP vagy MPP architektúrájú gépeken vagy egy fürt gépein futó feladatok között. Az üzenetátadás során a küldő és fogadó oldal együttműködik az adatok küldésében. Az adatokat explicit módon el kell küldeni az egyik oldalon és fogadni kell a másikon.
Üzenetátadásos mechanizmust használva rendkívül nagy, több száz gépből álló fürtöket is létre lehet hozni. A fürt teljesítményét viszont csak megfelelő módon megírt programokkal lehetséges kihasználni. Nem minden feladat alkalmas az ilyen nagyfokú párhozamos működésre.
A két legelterjedtebb üzenetátadás modellre épülő megoldás a Message Passing Interface (MPI) *http://www-unix.mcs.anl.gov/mpi/indexold.html* és a Parallel Virtual Machine (PVM) *http://www.epm.ornl.gov/pvm/pvm_home.html*. Mindkét megoldás nyílt forráskódú. A GNU/Linux az egyik legkedveltebb platform az MPI és PVM fürtök megvalósítására.

## 4.1. MPI alapú fürtök

A Message Passing Interface tulajdonképpen nem egy termék, hanem egy üzenetátadási könyvtárspecifikáció (API) a párhuzamos programok támogatására. Jelenleg C, C++ és a Fortran 77 nyelvekhez létezik MPI specifikáció. A legutolsó specifikáció, az MPI-2 152 függvényt tartalmaz. Az alábbi kód a jól ismert Hello World C program MPI változata.

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char **argv)

{
MPI_Init(&argc, &argv);
printf("Hello world\n");
MPI_Finalize();
return 0;
}
```

A példából jól látható, hogy a hagyományos kódot nagyon egyszerű MPI formára átírni. Az igazi problémát a feladat párhuzamosítása jelenti. Az MPI specifikációnak megfelelő programokat speciális parancsok segítségével kell lefordítani (*mpic*, *mpicc*, *mpif*). A legismertebb nyílt forrású MPI implementáció az *mpich*, amely megtalálható a *http://www-unix.mcs.anl.gov/mpi/mpich/* URL-en.
Önmagában az *mpich* nem elég egy fürt működtetésére. Ezért született meg a Local Area Multicomputer (LAM) *http://www.lam-mpi.org/*, mint egy MPI fejlesztő és futtató környezet. A LAM lehetővé teszi, hogy hálózatba kötött heterogén gépekből (különböző CPU és operációs rendszer, MPP, SMP gépek, PC-k) párhuzamos feladatok futtatására alkalmas fürtöt hozzunk létre. A LAM nem alkalmas általános célú (nem MPI) programok futtatására.
A LAM fürt indítása a *lamboot* paranccsal történik. Induláskor a konfigurációs fájlban megadott gépeken elindul egy LAM démon processz. Ezután az *mpirun* paranccsal lehet az MPI specifikáció szerint megírt programokat a fürtön elindítani. Az *mpirun*

parancs a LAM démonokon keresztül indítja el a futtatandó programokat. A végén a *wipe* paranccsal a LAM fürtöt le kell állítani.
A LAM lehetővé tesz bizonyos fajta hibatűrést, de nem nevezhető magas rendelkezésre állást biztosító fürtnek.
A Beowulf tartalmaz egy optimalizált *mpirun* programot. Így a Beowulf fürt egyaránt használható hagyományos és MPI programok futtatására.

### 4.2. Parallel Virtual Machine fürt

A Parallel Virtual Machine egy üzenetátadásos modellre épülő, heterogén architektúrájú gépekből álló fürt. A használt üzenetküldési mechanizmus hasonlít az MPI-hez. A PVM rendszer két részből áll. Az első a fürt minden elemén futó *pvmd* démon, ami az alkalmazások számára szükséges virtuális gépet valósítja meg. Egy fürtön belül több felhasználó is létrehozhat egymást átfedő virtuális gépeket. Miután kialakult a virtuális gép, az alkalmazások a fürt bármelyik gépéről indíthatók. A PVM rendszer másik részét a PVM interfész könyvtárak képezik. Hasonlóan az MPI programokhoz, a PVM programok fordítását is egy speciális *make* paranccsal kell végezni (*aimk*).
A PVM fürt felügyeletét a *pvm* parancssoros vagy az *xpvm* grafikus programmal lehet végezni.
A Scyld Beofulw fürt tartalmaz egy optimalizált PVM-et. Így a Beowulf egy nagyon sokoldalú és rendkívül nagy mértékben skálázható fürt, ami gyakorlatilag minden fajta alkalmazás futtatására alkalmassá teszi.

## 5. Összefoglaló

Az x86-os processzorok egyre nagyobb teljesítménye, a már rendelkezésre álló IA64 architektúra és a nagyszámú nyílt forráskódú vagy kereskedelmi fürt szoftver egyre vonzóbbá teszik a GNU/Linux alapú fürtöket. Gyakorlatilag minden fürt típusra létezik ilyen megoldás. Így rendelkezésre állásban és skálázhatóságban a GNU/Linux készen áll a vállalati környezetek meghódítására. Bár GNU/Linux fürt még nem szerepel a TOP500 Supercomputer *http://www.top500.org/* listán, számos fürt installáció alapján ez a platform biztosítja a legjobb teljesítmény/ár tényezőt.

# Vékony kliens technológia a Fővárosi Bíróságon

Laky Norbert <lakyn@birosag.hu>

2001.08.30.

**Kivonat**

A XVIII-XIX. Kerületi Bíróság új épületbe költözött 2000 őszén. A beruházás kapcsán zöld mezős beruházásra nyílt lehetőségünk. Ennek során –elemezve az elvárásokat és a trendeket– vékony kliens technológia bevezetése mellet döntöttünk.

## Tartalomjegyzék

**1. Bevezetés** **40**

**2. Tervezés – trendek** **40**
2.1. Tervezés – konklúzió . . . . . 40
2.2. Tervezés – Mi az, ami elérhető? . . . . . 40

**3. Vékony kliensek – lehetőségek** **41**
3.1. A Sun Hot Desk technológiája . . . . . 41

**4. Megvalósítás – Eszközök, tapasztalatok** **42**

# 1. Bevezetés

A XVIII-XIX. Kerületi Bíróság új épületbe költözött 2000 őszén. A beruházás kapcsán zöld mezős beruházásra nyílt lehetőségünk. A bíróságok életében a zöld mezős beruházás két szempontból is izgalmas:

- Ilyenkor lehetőséget kap az informatika, hogy a fantáziája szárnyaljon és valami újat alkosson.
- Ritkán adódik egy bíróság életében hasonló lehetőség, tehát értékállót kell teremteni.

# 2. Tervezés – trendek

A beruházás megtervezésekor megfigyeltük a bírósági informatikában észlelhető trendeket. Az alábbi rendszereket, elvárásokat elemeztük:

1. **A Cégbírósági rendszer:** A Cégbírósági rendszer (SCO Unix alapú PC szerverek, DEC VT terminálok (97db), Soros kábelhálózat, Puritán alfanumerikus felület, Modern háromrétegű alkalmazás) A legfontosabb tapasztalatunk, az erkölcsi amortizáció = fizikai amortizáció, illetve, hogy nincsenek kényszerek! Semmi sem kényszerit bennünket a rendszerelemek folyamatos cserélgetésére. A felhasználók barátságosabb felhasználói felületre vágynak. Grafikus felhasználói felület igény!
2. **PC-s hálózatok** (NetPC): Novell szerverek / Zenworks, IBM AIX szerverek / SAMBA, MS Windows kliensek – NetPc-szerű alkalmazás! A régi PC-s op. rendszerek alkalmatlanok, az új PC-s op.rendszerek modern, nagy kapacitású PC-t igényelnek. A vékony hálózati megoldáshoz vastag kliensek szükségesek!
3. **Szabadszoftver alapú** vékony-kliens megoldások (lsd. ETB konferencia)
4. **A Fővárosi Bíróság elnökének elvárásai:** Folyamatos rendelkezésre állás, informatikai biztonság, az erkölcsi amortizáció ideje közelítse meg a fizikai amortizáció idejét, ergonómiai elvárások, elégedett felhasználók, olcsó üzemeltetés. Kompatibilis irodai programcsomag, Adatbázisok (Jogtár, Lajstrom, . . . ), E-learning a szakmai kollégiumoknak.

## 2.1. Tervezés – konklúzió

A tapasztalatok elemzése során az alábbiak derültek ki:

- Minden számítástechnika alkalmazásunk a vékony technológiák irányába halad.
- A felhasználók 90% nem igényel vastag klienst!

## 2.2. Tervezés – Mi az, ami elérhető?

A következő kérdés az volt, milyenek a jelenleg elérhető gyári vékony megoldások? Több alternatíva is kínálkozik, ezek:

- Network Computer (NC)

- Windows Terminálok (WBT) (Speciális NC)
- SunRay technológia

# 3. Vékony kliensek – lehetőségek

Az NC nem egyértelmű sikertörténet. Úgy gondolom azért, mert hiányzik az NC-khez való versenyképes irodai programcsomag, illetve teljesen új nem tradicionális technológiát képvisel.

> „A világi bölcsesség arra tanít, hogy kevésbé árt a hírnévnek ha valaki konvencionális módon megbukik, mintha nem konvencionális módon ér el sikert.” (Keynes).

Ha ezt kizárjuk két választásunk marad:

- WBT: konvencionális MS környezet,
- Sunray: konvencionális SUN környezet (Xwindow, CDE, Gnome, StarfOffice, SmartCard alapú azonosítás).

Mi a SUN megoldása a SunRay technológia mellett döntöttünk, a szabad és kompakt irodai környezet miatt.

## 3.1. A Sun Hot Desk technológiája

A SunRay készülékek a Hot Desk technológiára épülnek, ami a gazdaságosabb, megbízhatóbb és biztonságosabb számítástechnikai környezet irányába történő fejlődés mérföldkövének számít. Ebben az architektúrában csak az marad az asztali gépben, amire a felhasználói felület megvalósításához szükség van, azaz a billentyűzet, egér és hang, mint bemenet, illetve képernyő és hang, mint kimenet. Minden feldolgozás a háttérben, egy vagy több megosztott központi szerveren történik. Mindaz, ami korábban a felhasználók munkaállomásán futott - legyen az a grafikus felület, felhasználói programok, levelezőprogram - most a szerveren fut. Mivel a programok a szerveren futnak, lehetővé teszi, hogy a felhasználó a munkacsoporton belül bármelyik SunRay berendezésről elérhesse az éppen nyitott applikációs környezetét. A kimenet és a bemenet átirányítása révén a felhasználói környezet pillanatok alatt "áthelyezhető" egyik SunRay-ről a másikra.
A Sun Solaris operációs rendszeren futó alkalmazások nagy többsége módosítás nélkül képes futni a SunRay szerveren, mégpedig a virtuális X11 device driver-eken keresztül, amelyek megvalósítják a szokásos ki- és bemeneti eszközök emulációját, és alacsonyszintű parancsok segítségével valósítja meg a távoli eszközön keresztül a felhasználói felületet. Mivel a Sun Ray készülékeken nem tárolunk adatokat, így a berendezés meghibásodása nem okoz adatvesztést, sőt még a felhasználói felület, a nyitott alkalmazások elvesztését sem, mert azokhoz egy másik (vagy cserélt) Sun Ray eszközről azonnal hozzá lehet férni. Mivel a Sun Ray nem vesz részt a programok futtatásában, így upgrade-re és egyedi telepítésre sincs szükség, amikor a felhasználók új alkalmazást vesznek használatba. A Hot Desk architektúra segítségével ki lehet használni, hogy a legtöbb felhasználó erőforrásigénye viszonylag magas csúcsok mellett átlagosan alacsony értéket mutat. A rendszer erőforrásainak centralizálásával és az erőforrások megosztott használatával lényeges költségmegtakarítás érhető el úgy, hogy

a felhasználók egy nagy teljesítményű, jobban kihasznált rendszerben dolgoznak. Az erőforrások megosztásából fakadó nyereséget mint pótlólagos, a redundaciát növelő teljesítményt költséghatékony módon visszaillesztheti a rendszerbe mind tükrözés, mind pedig melegtartalék formájában. (forrás: http://www.sun.hu)

# 4. Megvalósítás – Eszközök, tapasztalatok

**A Kispesti Bíróság rendszerének paraméterei:** Cat5e hálózat 100Mbit/sec kapcsolókkal, Sun E250 szerver 2Gigabyte memóriával, 55 db Sunray1 kliens - minden bíró, minden leíró, 5 tárgyaló, 3 db hálózati nyomtató, Openwin – Gnome grafikus környezet, StarOffice, CD-Jogtár, webes felületű hozzáférés a Lajstrom adatokhoz (ügyviteli alkalmazás), interoperábilitás a régi PC-kel és Netware szerverrel. Linux alkalmazás és fájl szerver.

**Tapasztalatok:** A beruházási költségek konvencionálisak, hiszen teljesen összevethetők a normál PC-s beruházásokkal (tapasztalatunk szerint a vékony megoldás jelentősen olcsóbb), az igazi nyereség azonban az üzemeltetési költségek csökkenéséből, a rendszerből következő szabadságból és a felhasználók elégedettségéből adódik.

# Nyílt forrású fejlesztési projectek dinamikája

Magosányi Árpád <mag@lme.linux.hu>

2001.08.24.

**Kivonat**

A nyílt forrású fejlesztési projectek eredményessége elsősorban a mögöttük meghúzódó bazár modellnek köszönhetik eredményességüket. Az előadás bemutatja a nyílt forrású projectek legfontosabb tulajdonságait, azok sikerkritériumait és a szokott buktatókat. Az előadást bőven illusztrálják életből vett példák is, küztük a Linux-felhasználók Magyarországi Egyesületéé is, ami noha nem szoftverfejlesztési project, azokkal sok működésbeli hasonlóságot mutat.

## Tartalomjegyzék

**1. Bevezetés** **44**

**2. Project, project management** **44**
2.1. A project definíciója . . . . . 44
2.2. Nyílt forrású projectek . . . . . 44
2.3. A projectmenedzsment céljai . . . . . 45

**3. Az élet, a világmindenség, meg a projectmanagement** **46**
3.1. A hacker kultúra, avagy a kiindulási projectkörnyezet . . . . . 47
3.2. Technikai kérdések . . . . . 47
3.2.1. Tecnikai kérdések és a flamewar . . . . . 47
3.2.2. Okos struktúrák és buta kód . . . . . 48
3.2.3. A jó ötletek . . . . . 48
3.2.4. A szintaktikus cukor . . . . . 49
3.3. Motivációs kérdések . . . . . 49
3.3.1. A kritikus tömeg . . . . . 49
3.3.2. A felhasználók fontossága . . . . . 49
3.3.3. Release early, release often . . . . . 50
3.3.4. A személyes probléma varázsa . . . . . 50
3.3.5. A hozzáállás . . . . . 50
3.3.6. A jótékony diktátor . . . . . 50
3.3.7. Ha lelépsz, tedd szépen . . . . . 51

**4. Epilóg** **51**

# 1. Bevezetés

Ebben az előadásban azt próbálom meg körüljárni, hogy hogyan és mitől működnek illetve nem működnek azok a tevékenységek, amik a lelkesedésből végzett közös munkára építenek. Ezt a tevékenységformát és a mögötte meghúzódó mechanizmusokat a nyílt forrású szoftverfejeztési projecteken, mint egy informatikus által talán legjobban ismert ilyen tevékenységen keresztül fogjuk vizsgálni.

Ennek a kalandozásnak a nem titkolt célja annak a kérdésnek a boncolgatása, hogy az LME[11] által végzett munkában a leszűrt tanulságok hogyan alkalmazhatóak, de nem fogunk elsiklani olyan kérdések mellett sem, amik a nem puszta lelkesedésből végzett szoftverfejlesztési projectek számára mutathatnak új utakat.

A félreértések elkerülése végett szeretném kijelenteni hogy nem tartom magam még amatőr szociológusnak sem, nemhogy a téma avatott ismerőjének. Ha az előadásnak csak annyi eredménye lesz, hogy a téma iránti érdeklődést felkelti, elértem a célomat.

# 2. Project, project management

## 2.1. A project definíciója

A project definíciója általában az alábbi fontos szempontokat szokta tartalmazni:

**Meghatározott erőforrásokkal operál** A project számára elérhető erőforrások általában végesek, azokat optimálisan kell elosztani. Ezen nincs is mit magyarázni.

**Meghatározot célja van** A projecteknek meghatározott céljuk, sikerkritériumaik vannak. A project sikerességét ezeknek a sikerkritériumoknak az elérése jelzi.

**Határidők vannak** A projectnek meghatározott idő alatt kell a célját elérnie.

A fentieken kívül szokás a project definíciójához hozzávenni annak méretére vonatkozó megkötéseket: azt szokás mondani, hogy egy ”rendes” project legalább néhány hónapig tart és több szervezeti egységet ölel át. Ezt azért kötik ki, mert a projectmanagement módszertanok által megkövetelt overhead csak ekkora méreteknél kezd kifizetődő lenni. A mi szempontunkból ennek azért nincsen jelentősége, mert mint látni fogjuk, a nyílt forrású szoftverprojectek egy kicsit más tulajdonságokkal rendelkeznek.

## 2.2. Nyílt forrású projectek

Először definiáljuk, hogy jelen tárgyalás szempontjából mit tekintünk nyílt forrású projectnek:

*Nyílt forrású projectnek azokat a szoftverfejlesztési projecteket tekintjük, amikor a fejlesztést önkéntes módon, lelkesedésből végzik, és a project terméke szabad szoftver.*

Lássuk, mik is az ilyen projectek legfontosabb tulajdonságai:

**Lelkesedésből csinálják** A project résztvevői lelkesedésből, és/vagy saját problémájuk megoldásának kedvéért vesznek benne részt.

**A határidők nem annyira lényegesek** Mivel pénzt nem kapnak érte, az eredmény sem annyira számonkérhető. Ennek egyik megjelenési formája az, hogy a határidők nem túl lényegesek.

**Nincs vége** A nyílt forrású szoftverek folyamatosan fejlődnek. Tehát a fejlesztésnek nincs vége, csak szakaszai vannak.

Természetesen a nyílt forrású szoftvert készítő projectek közül nagyon sok nem tekinthető a fenti definíció szerint nyílt forrású projectnek, gondoljunk csak azokra a szoftverekre, amiket egy-egy cég alkalmazottai a fizetésükért írnak. Ez csupán azt jelenti, hogy a „Nyílt forrású project" elnevezés nem elég pontos. De mivel definiálva van az itt alkalmazott jelentése, remélem ez nagyobb félreértés forrása nem lesz.

## 2.3. A projectmenedzsment céljai

A projectmenedzsment az a szervezési tevékenység, amit azért végeznek, hogy a project sikerkritériumai teljesüljenek. A projectmenedzsmentet valamilyen módszertan alapján szokás végezni. Ez a módszertan a nem nyílt forrású projecteknél általában formális, könyvből megtanulható. Ilyen módszertanok a SUMMIT-D[8] vagy a PRINCE[7]. A nyílt forrású projecteknek leírt módszertana nincsen (hacsak ESR írásait[1][3][4] nem tekintjük annak), azt főként hagyomány és szokások útján lehet megtanulni. Gyakorlatilag sikerkritériumok sincsenek, egy project sikerességét csupán az elkészült szoftver felhasználótáborának mérete alapján lehet megbecsülni.
A projectmenedzsment céljai általában az alábbiak:

**Célok definíciója, és a résztvevők irányban tartása** Egy hagyományos projectnél ez a feladat egyértelmű: a vezetés megadja a célokat, és a sikerkritériumokat, a projectvezetés ezt lebontja részfeladatokra, amiket kioszt. Egy nyílt forrású projectnél a helyzet egy kicsit más: az elérendő célokat maguk a project résztvevői definiálják, aszerint hogy számukra mik azok a problémák, amikre megoldást várnak a projecttől. Ebből következik viszont, hogy a résztvevők irányban tartása szinte nem is feladat: a project céljai nagyon jól illeszkednek a résztvevők egyéni céljaira.

Itt a feladat a célok definíciója helyett a megfelelő mérnöki döntések meghozatala szokott lenni. Ez a munka első ránézésre, sőt másodikra sem tűnik projectmenedzsernek való feladatnak, de két fontos tényt vegyünk figyelembe: Egyrészt a nyílt forrású szoftvereknek az egyetlen sikerkritériuma a felhasználók száma, ami főként a szoftver használhatóságától függ. Egy nyílt forrású projectnél nem történhet meg az, ami a hagyományos projecteknél sajnos igen gyakori; a project eredménye technikai szempontból elfogadhatatlan, de a projectmanagement a vezetés felé képes a projectet mint sikereset feltüntetni. Tehát a technikai eredményesség a project túlélése szempontjából rendkívül kritikus, úgy a felhasználók, mint a résztvevők megtartása miatt. Másrészt a project vezetője a nyílt forrású szoftvereknél mindig elsősorban mérnök, és csak másodsorban projectmenedzser, így az ő érdeklődése is főként a technikai kérdésekre irányul.

**Figyelni hogy a fontos részletek ne sikkadjanak el** A project módszertanok egyik legfontosabb eleme éppen az, hogy a követendő feladatok és szempontok részletesen föl vannak sorolva, így a projectvezető tud puskázni hogy mikre kell figyelnie. A nyílt forrású projecteknél ez a feladat sokkal heurisztikusabbn működik: A gyors fejlesztési ciklusok miatt egy-egy funkció az elkészülte után csaknem

azonnal ki van téve a publikus szemlének, így az esetleges problémák gyorsan felszínre kerülnek.

**Az emberek motiválása az unalmas munkára** A hagyományos projecteknél ezt a problémát prémiumokkal és az emberek ismételt kérlelésével szokás megoldani (a kérlelést addig kell folytatni, amíg unalmasabbá nem válik mint a munka elvégzése maga). A nyílt forrású projecteknél kétféle hajtóerő van (a kérlelés persze rövid távon itt is beválik, de hosszú távon nem feltétlenül kifizetődő). Az egyik az, hogy a résztvevők sokfélesége miatt gyakran lehet találni olyan embert, aki szmára az adott munka nem unalmas; akár azért, mert az elvégzendő munkát jelképező problémát nagyon érzi, akár azért, mert az ő készségeinek pontosan az adott munka a megfelelő szintű kihívás. A másik ok az, hogy a nyílt forrású projectek ajándék-kultúrán (gift culture) alapulnak, ahol a legfontosabb „fizetőeszköz" a reputáció. Ez azt jelenti, hogy az, akinek a kompetenciájába tartozik egy adott feladat elvégzése, annak elhanyagolásáért a saját reputációjával fizet. Ez sok esetben nagyobb hajtóerőt jelent az anyagi javaknál.

**Munkabeosztás** A munka beosztásáról a résztvevők irányban tartásánál már beszéltünk. A nyílt forrású projecteknél a kiemelendő szempont az, hogy a munkabeosztás egy önszabályzó, a célok kijelölésével harmonizáló folyamat. Legjobban talán a Debian projectnél látszik ez: az egyes csomagokat azok pátyolgatják, akik az adott témát legjobban ismerik, és a funkcionalitásra a legnagyobb szükségük van.

**Az erőforrások megszerzése, megtartása** A hagyományos projectmenedzseri munkának fontos része a project elején az erőforrások biztosítása, azt követően pedig ezeknek az erőforrásoknak a megtartásáért vívott reménytelen küzdelem:) A nyílt forrású projecteknél a kép egyszerűbb, noha a feladat nem kevésbé bonyolult. Itt ugyanis egyetlen erőforrás van, ami lényeges; az emberi tudás. A project vezetőjének tehát a feladata az hogy megfelelő mennyiségű embert motiváljon résztvételre a projektben.

# 3. Az élet, a világmindenség, meg a projectmanagement

Mint a fentiekből is látható, a nyílt forrású projectek vezetésének két alapvető problémája a résztvevők motiválása és a technikai döntések meghozatala.

Továbbmenve, a technikai döntések meghozatalának mikéntje is a résztvevők motiválásának egyik nagyon fontos eleme.

Ez akkor válik igazán nyilvánvalóvá, ha áttekintjük ESR „Cathedral and the Bazaar"[1] című írását. Az írásban ESR a sikeres projectmenedzsment 19 alapelvét fektette le. Ezek közül 9 direkt módon a project résztvevőinek motiválásával foglalkozik. A maradék 10 technikai jellegű alapelv közül is többnek van nem tehnikai jellegű mondanivalója. Ha ehhez hozzávesszük azt a rendkívül fontos szerepet amit a technikai diszkusszió a résztvevők motiválásában játszik, máris látható az ábra.

A továbbiakban ezt fogjuk boncolgatni, méghozzá három lépésben. Először a hacker kultúrában kevésbé járatosak kedvéért áttekintjük azt a környezetet, amiben a nyílt forrású projectek léteznek, majd sorra vesszük a technikai majd a motivációval kapcsolatos kérdéseket, ESR alapelveit használva szamárvezetőként.

## 3.1. A hacker kultúra, avagy a kiindulási projectkörnyezet

A hacker kultúra, ahogyan ESR a ”Homesteading the Noosphere” című írásában[4] kimutatta, alapvetően egy ajándék-kultúra (Gift culture), ami azt jelenti, hogy az egyén társadalmi pozíciója attól függ, hogy a közösségnek mit tud adni. A hacker kultúra által figyelembe vett javak jórészt virtuálisak: szoftver, dokumentáció és tudás.

Ennek a kultúrának az éltető eleme az Internet. Az internet kommunikációs lehetőségei azok, amik lehetővé tették olyan méretű szoftverfejlesztési projectek elindítását, mint a Linux Kernel vagy a Debian project. A hackerek és az internet között egyfajta szimbiózis lelhető fel. Egyrészt a hackerek közötti kommunikáció alkalmazkodott az internet adta lehetőségekhez: a szigorú írásbeliség, a metakommunikáció korlátozott lehetőségei mind befolyásolják a kommunikációs stílust. Másfelől az internetet a hackerek alkották, és formálták a saját igényeiknek megfelelőre.

A legfontosabb kommunikációs csatornák az email, levelezési listák, az nntp news, és az IRC. Az internet mint tárolóeszköz és „groupware médium” is szerepel. Erre a célra weboldalakat és ftp site-okat használunk.

A hacker kultúra legtöbb eleme a szoftverfejlesztéssel valamilyen módon kapcsolatba hozható. Ezek között lehetne említeni rengeteg tradíciót. Talán legfontosabbak ezek közül a mi tárgyalásunk szempontjából a projectekkel kapcsolatos tabuk:

**Nem forkolunk** A projectek „forkolása” az amikor egy szoftverből olyan verziót hoznak létre, ami a szoftver aktuális fejlesztőinek céljaitól eltérő módon működik. Ez a fejlesztői bázis szakadásával járó, fájdalmas folyamat. Mivel a folyamat végén mindkét változat bázisa gyengébb lesz, ilyet csak akkor csinálnak ha feltétlenül muszály, és ilenkor is mindenki magyarázkodik meg minden.

**Kooperálunk a fő fejlesztővel** A funkcionális változtatások disztribúciója a fő fejlesztő tudta nélkül gorombaságnak számít.

**A jóváírásokat meghagyjuk** Valakinek a nevét kitörölni a hozzájárulók listájáról az érintett kérése nélkül a legfelháborítóbb dolog amit tenni lehet.

Az első két tabu célja egyértelműen a működő fejlesztői bázis fenntartása, míg a harmadik tabu mutatja a reputáció fontosságát.

A hackerek történetéről részletesebben a „A Brief History of Hackerdom” című írásban[2] olvashatunk, és a Hacker-howto[5] és a Loginakata[6] nyújthat bepillantást ebbe a kultúrába.

## 3.2. Technikai kérdések

Mivel a technikai kérdések önmagukban nem túl érdekesek, ezért nem fogom mindet érinteni amik a Cathedral and the Bazaar írásban szerepelnek, csak azokat, amelyek szociológiai szempontból tanulsággal szolgálnak. Mielőtt ezekre rátérnék, a technikai kérdésekről való egyeztetés fontosságát boncolgatom

### 3.2.1. Tecnikai kérdések és a flamewar

A technikai kérdésekkel kapcsolatos egyeztetés a project életében két szempontból fontos: Egyrészt a jó technikai döntések alapozzák meg a készülő szoftver minőségét és az elkészítéshez szükséges munka mennyiségét, másrészt a szoftverről szóló diszkusszió az ami életben tartja a projectet, mivel ez az egyik olyan terület amivel a project tagjainak érdeklődését fel lehet kelteni és fenntartani.

Érdekes megfigyelni azt, hogy ez a diszkusszió igen gyakran fűtött hangulatú vitákban csúcsosodik ki. Ez olyan szinten igaz, hogy az ehhez nem szokott szemlélő számára riasztó módon hathat az a mód és hevesség ahogyan az egyes kérdések a levelező listán megvitatásra kerülnek. Ez alól a szabály alól úgy tűnik nincs olyan sikeres nyílt forrású project ami kivétel. Az LME levelezőlistáját és az egyesületben végzett munkát megfigyelve pedig kiderül, hogy ezeknek a „flamewar"-oknak rendkívül erős motivációs hatásuk van. Egy-egy ilyen után két-három hétig csak úgy buzog az élet.

Sajnos a flamewaroknak visszatartó erejük is van: esett meg hogy egy nyílt szoftvert fejlesztő cég azért nem készített nyílt fejlesztői listát mert az azon várható flamewart túlságosan erős zavaró tényezőnek gondolták. Ugyanígy látható, hogy többen éppen ezek miatt a viták miatt távolodtak el az Egyesülettől.

A fenti két effektust figyelembe véve az tűnik ésszerűnek, hogy a levelezőlistán folyamatos, de lehetőleg szélsőségmentes forgalmat tartunk fönn. A forgalom katalizálására az Egyesületnél a Keddlog, a fejlesztési projecteknél a release notes-ok és különböző státuszriportok szolgálnak, míg a moderálásra a lista illemtantól kezdve a moderátori eszközök és a spamdb alkalmasak. Abban úgy tűnik egyetértés van, hogy emberi moderátor beiktatása az esetek nagy részében nem a legalkalmasabb megoldás: nem csak azért mert az emberi közreműködés sok erőforrás, hanem az általa bevezetett késleltetés néha túl lassúvá teszi a kommunikációt.

### 3.2.2. Okos struktúrák és buta kód

ESR egyik alapelve azt mondja: *„Az okos adatstruktúrák és a buta kód sokkal jobb, mint fordítva"*.

Ennek a mondásnak a technikai alapja az, hogy ha az struktúrák okosak, akkor „elvégzik azt a munkát", amit egyéb esetben a kódnak kellene végeznie. Ha belegondolunk, az egész digitális technika arra épül hogy az anyagot olyan okos struktúrákba hajtogatják, amitől az a buta elektron pont úgy görbül hogy mindenféle okos számításokat lehessen vele végezni.

Ez a mondás az emberi munkára is igaz. Ha a struktúrák (itt nyugodtan gondoljunk szervezeti struktúrára éppúgy mint egy weboldal felépítésére) okosan vannak felépítve, akkor a dolgokat működtető emberek sokka nagyobb hatást tudnak kifejteni. Ennek egyik oka az, hogy aki csinál valamit, az ezt effektívebben tudja tenni, a másik pedig az hogy olyanok is „odaférnek" a munkához, akiknek egy rosszabb struktúra esetén nem lenne hozzá elég affinitásuk. Fontos azt tudni, hogy az okos struktúra nem feltétlenül bonyolult, sőt a jól bevált „KISS" alapelv szerint (Keep It Simple, Stupid), a legegyszerűbb struktúrák a legjobbak.

A szervezeti struktúra okos felépítésére irányuló erőfeszítésekre az egyesületnél a csoportok kialakítása, az azóta megszüntetett választmány, és a Vének Tanácsa volt a példa. Sajnos ezek közül összesen a csoportok voltak azok amikre egyelőre nem lehet azt mondani hogy hamvába holt ötlet.

Az okosan struktúrált munkakörnyezetre irányuló erőfeszítés a Fordítás Segítő Rendszer (FSR)[12], amelynek célja az, hogy a fordítási munkákat koordinálja, egyszerűbbé tegye a fordítók munkáját, és lehetővé tegye a bekapcsolódást azoknak is, akik egyébként csak kevés idővel rendelkeznek erre a célra.

### 3.2.3. A jó ötletek

A következő számbavett alapelv így szól: *„Felismerni a felhasználók jó ötleteit majdnem olyan jó, mintha neked lennének jó ötleteid. Néha még jobb is"*.

Én tovább mennék. A felhasználók jó ötleteit felismerni mindig jobb, mert ez azzal jár, hogy a felhasználó úgy érzi, hogy „foglalkozva van vele”. Ezen túl a szoftver a valós felhasználói igényeket fogja kielégíteni.

Ha elvonatkoztatunk a szoftverfejlesztéstől, a helyzet még egy kicsit csiszolódik. Ugyanis egy nem annyira jó ötlet, ami egy projecttagtól ered, gyakran nagyságrendekkel hatásosabb mint egy nagyon jó, ami nem. Ugyanis ha valakinek van egy ötlete, könnyen rávehető arra hogy valósítsa is meg.

#### 3.2.4. A szintaktikus cukor

Az utolsónak citált technikai alapelv: *Ha a nyelved sehol sincs a Turing-teljeshez, a szintaktikus cukor a barátod lehet.*

Azt hiszem ezt a mondást a hacker slangben kevésbé járatosak számára nem árt értelmezni. ESR itt a konfigurációs állomány szintaxisáról beszél. Azt mondja, hogy ha ez a szintaxis egyszerű, akkor kedves ötlet lehet olyanra formálni, ami az angol nyelvhez közelebb áll. Ezzel két legyet lehet ütni egy csapásra. Az egyik ami ESR-nek úgy tűnik nem jutott eszébe, és a mi diszkussziónk számára sem túl lényeges, az hogy az így bevitt redundancia egyszerűsítheti a konfigurációs hibák felismerését. A lényegesebb elem az hogy így a konfiguráció közelebb fog állni a felhasználó szívéhez, és így újabb elégedett felhasználókra teszünk szert.

### 3.3. Motivációs kérdések

A motiváció nagyon fontos dolog. A felhasználók elégedettségétől függ ugyanis azok száma. A felhasználók számától függ az hogy mennyire lesz tesztelve a program, és az is hogy mennyi lesz az aktív fejlesztő. Ugyanis az aktív feljesztők és az összfelhasználószám aránya egy adott programtípusnál nagyjából konstansnak tekinthető (most tekintsünk el a programozási stílusnak és a fejlesztőkkel való kapcsolattartás stílusának a fejlesztők számát befolyásoló hatásától). Ez abból is látszik, hogy ESR alapelveinek majdnem a fele ezzel foglalkozik. Lássuk őket:

#### 3.3.1. A kritikus tömeg

*Ha elég nagy a bétateszteri és a fejlesztői bázis, szinte minden probléma gyorsan be lesz határolvam és nyilvánvaló lesz valakinek a javítás.*

Ez a szabály egy rendkívül fontos fogalomhoz, a kritikus tömeg fogalmához kapcsolódik. A kritikus tömeg az a felhasználómennyiség, ami fölött egy fejlesztés önfenntartóvá válik. A felhasználók számával ugyanis egyenesen arányos a bétateszterek és a fejlesztők száma, és létezik olyan fejlesztői szám, ami elegendő egyszoftver folyamatos fejlődéséhez. A sikeres szoftverprojectek tulajdonsága az, hogy elérték a kritikus tömeget. Továbbmenve, a legsikeresebb szoftverproject, a Linux kernel immár azzal a problémával küzd, hogy túl sok a fejlesztő, és a fejlesztői ötlet, és ez a project koordinációjában okoz zavarokat, amik persze visszahatnak a minőségre is.

Az egyesület életében ez úgy fogalmazódik meg, hogy ha sok a tag, akkor várhatóan több lesz közülük aktív (az aktív tagok aránya kb tíz százalékra tehető), és így többet tudunk tenni a céljainkért is.

#### 3.3.2. A felhasználók fontossága

*A felhasználók fejlesztőként való kezelése a legegyszerűbb útja a gyors fejlesztésnek és az effektív hibakeresésnek.*

*Ha a bétateszterekkkel úgy bánsz, mint a legfontosabb erőforrásoddal, ők őgy válaszolnak, hogy a legfontosabb erőforrásoddá válnak.*

Ez a két alapelv nagyjából ugyanazt mondja: a felhasználói és a bétateszteri kört meg kell nagyon becsülni, és akkor nagyon sok terhet levesznek a fejlesztők válláról. Ez az alapelv a nem nyílt forrású projectek számára is megszívlelendő.

Az egyesületnél a potenciális és valós tagság az, ami ezt a figyelmet megérdemelné, sajnos eddig nem sikerült igazán jó megoldást találni a megszólításukra.

### 3.3.3. Release early, release often

*Tedd közzé hamar. Tedd közzé gyakran. És figyelj a felhasználóidra.*

Ez talán ESR leggyakrabban idézett mondása. A nyílt forrású fejlesztések egyik mozgatórugója az evolucionális fejlődés, aminek sebessége legjobban a generációváltás sebességétől függ. Ha gyakran jönnek ki új verziók, akkor a hibák hamar napfényre kerülnek, a fejlesztési irányváltások simák lehetnek, és a felhasználói/fejlesztői tábor folyamatos izgalomban tartható.

Főként a folytonos izgalomban tartás és az irányváltások simasága az ami megszívlelendő a nem szoftverprojecteknél.

### 3.3.4. A személyes probléma varázsa

*Minden jó szoftver úgy kezdődik hogy a fejlesztőnek személyes problémája támad.*

Ez az alapelv a személyes érintettség fontosságát emeli ki. Ahhoz hogy valaki bekerüljön egy nyílt forrású projectbe, szinte kötelező hogy valamiért személyes leszámolnivalója legyen egy olyan problémával, ami az adott szoftverhez kapcsolódik.

A nem szoftverfejlesztési projecteknél a levonható tanulság az, hogy a project által képviselt ügyet a résztvevők vagy potenciális résztvevők személyes ügyévé kell tenni. Ez az egyesületnél talán a Linuxxal kapcsolatos ismeretekre való éhség kiaknázásával lenne legjobban elérhető, és lehet hogy a motivációs problémák pontosan onnan fakadnak, hogy az egyesület tagjai által végzett tevékenységek nagy része a PR és a rendezvényszervezés témakörébe esnek, amihez az átlag informatikusnak nincs túlzott affinitása. Pedig székpakolás közben lehet a memóriamenedzsment rejtelmeiről és a hálózati határvédelem kérdéseiről beszélgetni:)

### 3.3.5. A hozzáállás

*Ha megfelelő a hozzáállásod, az érdekes problémák megtalálnak.*

*Hogy megoldhass egy érdekes problémát, keress egyet.*

Ez az alapelv inkább a projectek résztvevőinek szól, mint azoknak, akik irányítják azokat. Egyszerűen arról szól, hogy a világ tele van érdekes tennivalókkal, és ha az ember nem ilyed meg azoktól, és keresi őket, rengeteget tanulhat.

### 3.3.6. A jótékony diktátor

*Ha a fejlesztés vezetőjének van egy médiuma ami legalább olyan jó, mint az internet, és tudja hogyan kell kényszer nélkül vezetni, a sok fej mindenképpen jobb mint egy*

Ez az alapelv azt mondja, hogy a jó vezető nem erővel, hanem meggyőzéssel veszi rá a segítőit a megfelelő lépések megtételére. Azt hiszem egy lelkesedésből vállalt feladatnál nem kell ecsetelni ennek a jelentőségét, és a formális felépítésű szervezeteknél is nyilvánvaló az ilyen vezetési stílus előnye.

### 3.3.7. Ha lelépsz, tedd szépen

*Ha elveszted az érdeklődésed a program iránt, az utolsó feladatod átani azt egy hozzáértő örökösnek.*

Ez a mondás önmagáért beszél. Láthatjuk azt a folyamatot, ahogyan Linus a stabil verzióval kapcsolatos ügyeket átadja Alan Coxnak. Az egyesület életében pedig megfigyelhetjük, hogy az elnökök miután egy év alatt tele lesz mindenük az egyesülettel, javasolnak egy olyan elnökséget, akikből kinézik hogy tovább tudják azt vinni a hátukon.

# 4. Epilóg

Az itt felvillantott kérdések a nyílt forrású projectek vezetésének csak néhány aspektusát fedték le. Technikai kérdésekről direkt nem esett túl sok szó, nem szóltam a problémafeloldás mechanizmusairól (ezt a problémakört ESR a „Homesteading the Noosphere" című írásában alaposan felboncolja), és szó sem esett arról a lehetőségről, hogy esetleg nyílt forrású projectekhez is lehetne készíteni módszertant a SUMMIT-D vagy a PRINCE példájára. A cél csupán annyi volt, hogy egy kicsit többet megtudjunk az LME működéséről a nyílt forrású projectek példáján keresztül.

# Hivatkozások

[1] Eric S. Raymond (ESR): The Cathedral and the Bazaar
(http://tuxedo.org/ esr/writings/cathedral-bazaar/cathedral-bazaar/)

[2] Eric S. Raymond (ESR): A Brief History of Hackerdom
(http://tuxedo.org/ esr/writings/cathedral-bazaar/hacker-history/)

[3] Eric S. Raymond (ESR): The Magic Cauldron
(http://tuxedo.org/ esr/writings/cathedral-bazaar/magic-cauldron/)

[4] Eric S. Raymond (ESR): Homesteading the Noosphere
(http://tuxedo.org/ esr/writings/cathedral-bazaar/homesteading/)

[5] Eric S. Raymond (ESR): Hogyan váljunk hackerré? (Kovács Emese fordítása)
(http://lme.linux.hu/forditas/hacker-howto.html)

[6] Eric S. Raymond (ESR): Loginakata
(http://lme.linux.hu/forditas/loginataka.html)

[7] Prince2, site index page.
(http://www.prince2.com/)

[8] Summit home page:
(http://www.pwcglobal.co.uk/gx/eng/about/svcs/summit/smmthome_allsallgxeng.html)

[9] Debian home page
(http://www.debian.org/)

[10] Kernel Traffic
(http://www.zork.net/)

[11] Linux-felhasználók Magyarországi Egyesülete
(http://www.lme.hu/)

[12] Fordítás Segítő Rendszer (FSR)
(http://fsr.zope.hu/)

# Biztonságos aktív webkiszolgáló építése

Mátó Péter

2001. szeptember 3.

Köszönet Borbély Zoltánnak és Kiss Gabriellának

## Tartalomjegyzék

**1. Kinek szól ez az előadás? 54**

**2. Az operációs rendszer biztonsági kérdései 54**

**3. Megfelelő eszközök kiválasztása 56**

**4. Rendszer tervezése gyakorlati szemszögből 56**

**5. Az aktív tartalom fejlesztése 58**

**6. A kiszolgáló telepítése, beállításai 58**

**7. Üzemeltetési kérdések 59**

**8. Már létező rendszer védelme 60**

**9. Idegen szavak jegyzéke 60**

# 1. Kinek szól ez az előadás?

Ezen előadás elsősorban azoknak szól, akik felelősek, vagy azok lehetnek egy webszerver biztonságáért. Az **aktív** webkiszolgálók védelméről lesz szó. Miért?

Az Internet leggyakrabban használt nyilvános rendszerei a webkiszolgálók. Egy cég vagy intézmény virtuális életének kirakatai, melyek a legtöbb támadásnak vannak kitéve. Ennek több oka is van. Ma még viszonylag ritka jelenség az irányított adatrablás, ahol a csibészek kifejezetten egy adott cég adataira pályáznak. A weboldalak lecserélése (angol nyelvterületen „deface") viszont látványos, rontható vele egy szervezet megítélése, nyilvánossága miatt el lehet vele dicsekedni. Nem utolsósorban a web alapú szolgáltatások gyakran házilag barkácsolt kiszolgálóval működnek, így nem ritka a könnyen áthatolható védelem. Az operációs rendszere gyakran nem célszerűen, szokásjog alapján van kiválasztva, a rajta futó szerverprogram is gyakran hibás. Ha a kiszolgáló tartalmaz aktív lapokat is, az aktivátor általában házilag fejlesztett, biztonsági szempontból nem megfelelően átgondolt, így sokszor lehetővé teszi a rendszerbe való bejutást.

Mivel támadásnak van kitéve ésszerű biztonsági rendszereknél alkalmazott módszertan szerint terveni és felépíteni. Egy általánosan használt rendszerben talán megbocsátható, ha hibák lépnek fel, egy biztonsági rendszernél azonban ez végzetes lehet. Érdemes tehát olyan megoldásokra törekedni, amelyek a lehető legkevésbé támadhatóak. Ha nem is adunk a rendszernek tökéletes biztonságot, de elérhetjük azt, hogy olyan nagy energiára legyen szükség a rendszer sikeres támadásához, hogy az már ne érje meg.

# 2. Az operációs rendszer biztonsági kérdései

Az operációs rendszer beállításai közvetetten ugyan, de erősen hatással vannak a biztonságra. A biztonság a rendszer fizikai felépítésénél kezdődik. Most nyilván mindenki arra gondol: „Tudom, tudom. A biztonság része az üzembiztonság is." Ebben az esetben azonban sajnos nem ez a helyzet. Mivel a Linux kernelbe kerülő meghajtók fejlesztői nem szükségképpen biztonsági szakemberek, azokban is lehet hiba. Célszerű megbízható, gyakran használt eszközöket használni a rendszer felépítésénél, így kisebb valószínűsége van a kártyameghajtó hibájából való behatolásnak. Szükség esetén az operációs rendszer használt része auditálható, de ebben az esetben a szükséges munkaigény igen nagy.

Biztonsági szempontból a legfontosabb tényező a telepítés. A disztribúciók területén a hiedelmek ellenére nincs csodaszer. Vannak biztonságosabb alapbeállításokkal rendelkezők, de minden Linux disztribúció átalakítható biztonságossá kisebb-nagyobb energia-befektetéssel. A rendszer telepítése közben mindig oda kell figyelni arra, hogy kizárólag a feltétlenül szükséges binárisok kerüljenek a rendszerre. Minden egyes plusz program növeli az esélyét annak, hogy hibás program lesz a rendszeren. Szélsőséges esetben akár az is megoldás lehet, hogy egy már élő rendszeren egy könyvtárba telepítjük a szükséges csomagokat, és abból valamilyen automatizált eljárással generáljuk a szerver állománykészletét. Így a frissítés jóval nehézkesebb, de a csibészek dolgát jócskán megnehezíthetjük. Figyelni kell a rendszer felállásakor elinduló szolgáltatásokra is. Minden újabb szolgáltatás tovább növeli a kompromittálódás veszélyét. Legyünk óvatosak. Bizonyos disztribúciók (ilyen például a Debian) hajlamosak egy korábban leállított szolgáltatást frissítés esetén újra indítandóvá tenni, ezzel kellemetlen meglepetéseket okozva a rendszer fenntartójának.

A korábban már említett meghajtóhibák elkerülésére és a hatékonyabb működés érdekében célszerű egyedi fordítású kernelt használni. Ha olyan igény merül fel, amelyet a hivatalos Linux kernel egyelőre nem támogat (például IPSec VPN[5]), akkor a kernelbe az adott patch is beépíthető, de a nem hivatalos kiegészítések használatát célszerű kerülni. Ezen kiegészítések ugyanis okkal nem kerülnek a hivatalos kernelbe, ami pedig a leggyakrabban teszteletlenség, vagy valamely komponenssel való összeférhetetlenség. Elérhetők a kernel biztonsági lehetőségeit bővítő kiegészítések is (például OW [1], HAP [2], LIDS [3] vagy a grsecurity [4]), ezek használata abban az esetben célszerű, ha azok forrása hiteles. A kernelbe érdemes befordítani a csomagszűrő támogatást is, amely segítségével egy erős nulladik védelmi réteg alakítható ki a rendszer részére. Ez lehetővé teszi a felesleges szolgáltatások tiltását (így nem okoz gondot egy esetleg véletlenül újraindult szolgáltatás), illetve az engedélyezett szolgáltatások elérhetőségének korlátozását (ssh engedélyezése csak az arra kijelölt gépekről). További lehetőség a 2.4-es kernelsorozatnál a DoS és DDoS támadások elhárítása. Ez nem védi meg a rendszert az elérhetetlenségtől, hisz a DDoS éppen arra épül, hogy a sok haszontalan kérés kiszolgálása közben a rendszer vagy összeroppan, vagy a hasznos kérések nem lesznek kiszolgálva. Mivel azonban korlátozható a segítségével a felépült kapcsolatok száma, így a rendszert meg lehet védeni az összeomlástól, és a támadás befejeztével a rendszer tovább üzemel.

A rendszeren lévő setuid és setgid állományok számát a lehető legkisebbre kell korlátozni. Az esetek többségében a sudo parancs elegendő, de bizonyos esetekben elérhető akár az is, hogy ne legyen egyetlen ilyen állomány sem. Ügyelni kell arra, hogy a rendszeren az állományok hozzáférési jogosultságai a megfelelőek legyenek. Ez annyit jelent, hogy olyan állományok ne legyenek mindenki által olvashatóak vagy írhatóak, amelyek általános elérésére nincs szükség. Különös figyelmet kell fordítanunk a hálózati démonokat futtató felhasználók állomány-hozzáférési jogaira. A legjobb megoldás az egyes rendszerek különválasztása. Ez a legegyszerűbben a kernel által nyújtott chroot hívással valósítható meg. Ugyan felmerülhetnek vele szemben biztonsági problémák, de ezek a kernel némi módosításával elfogadhatóan biztonságossá tehetők [2]. Jóval biztosabb megoldás valamilyen számítógép emulátor használata. Ilyenek: user-mode Linux [6], VMWare [7] vagy az Argante [8]. Ebben az esetben az egyetlen gépen futtatandó szolgáltatások nagy biztonsággal szeparálhatók. Ezen megoldások részletes ismertetése meghaladja előadásom kereteit.

Ha a rendszer lehetővé teszi, be kell határolnunk az egyes felhasználók erőforrás-felhasználását, így elkerülhető, hogy valamely hálózati kiszolgáló túl sok erőforrást használjon fel. Kérdéses azonban, hogy hogyan reagál az adott kiszolgálóprogram az erőforrásigény elutasítására. Ezt minden rendszernél tesztekkel lehet eldönteni. Ha azonban kiderül egy kiszolgálóról, hogy leáll abban az esetben, ha nem kap elegendő memóriát, akkor is érdemes határokat bevezetni. Így ugyanis elkerülhető, hogy egy távoli, elérhetetlen gépteremben lévő rendszer néhány órára, vagy akár végleg elérhetetlenné váljon. A másik véglet, hogy a gép jóval nagyobb terhelést képes elviselni, mint azt a kernelben beállított értékek lehetővé teszik (például a nyitott FD-k maximális száma kevésnek bizonyulhat egy többprocesszoros gépen). Ebben az esetben szükség lehet a kernel kisebb módosítására is. Az alapértelmezett értékek magasabbra állításával a rendszer működése stabilizálható.

# 3. Megfelelő eszközök kiválasztása

A rendszer megvalósításához célszerű olyan eszközöket kiválasztani, amelyek tervezésénél a biztonság szempontjait is figyelembe vették, helyesen implementálták, és megfelelően leteszteltek. Ezen kitételek megítélése nem egyszerű feladat, az aktív tartalom fejlesztésénél bővebben szólok róla. Általános ökölszabályok: minél gyakrabban használt a szerverprogram, annál valószínűbb, hogy az esetleges hibákat kiszűrték belőle. Ha egy szerverprogram fejlesztésénél egy elismert biztonsági guru is tényked ik, akkor nagy az esélye, hogy a rendszer megbízható. Ha az adott programban korábban több súlyos hiba került nyilvánosságra, akkor annak használatától célszerű eltekinteni. Ennél többet általában nem lehet állítani egyetlen programról sem. Ma még szinte egyetlen program sem létezik, amely minősítetten biztonságos. A CC [9] mára szabvánnyá vált, a biztonságos fejlesztés azonban túl nagy plusz erőforrás-ráfordítást igényel. Ezt az erőforrást szívesebben használja mindenki a funkcionalitás továbbfejlesztésére, és valljuk be: a biztonságos fejlesztés és a folyamatos ellenőrzés meglehetősen unalmas.

Az aktív tartalommal bíró webszolgáltató valamilyen adatokat tesz elérhetővé. Tehát a webszerver mellett szükséges valamilyen adattárolási módszer is. Saját fejlesztés esetén a szükségesre fokozható a rendszer biztonsága, de ez általában komoly erőforrásokat igényel, különösen akkor, ha SQL nyelvet is szeretnénk használni. Saját adatformátum esetén nem használhatóak a mások által fejlesztett kiegészítő eszközök sem. Mindent egybevetve valamely általánosan használt SQL szerver használata a legésszerűbb.

A dinamikus lapoknak két alapvető típusa van: a kliens oldalon futó (Java, JavaScript, Flash ...) és a szerver oldalon futó. A kliens oldalon futó tartalom a szerver szempontjából statikus, arra semmilyen közvetlen veszélyt nem jelent, így ezzel itt nem foglalkozunk. A szerver oldalon futó dinamikus tartalom azonban komoly gondok forrása lehet.

Az aktivátor kétféleképp futhat: a webszerveren belül vagy az operációs rendszer egy különálló programjaként (CGI). Mindkét típus lehet valódi bináris (C, C++ stb.), vagy interpreter által értelmezett script. Mivel a rendszerek alatt lévő vasak elég erősek (általában a vonal a szűk keresztmetszet), és a scriptek fejlesztése és módosítása sokkal egyszerűbb és olcsóbb, így általában az utóbbit részesítik előnyben. Sok olyan scriptnyelvet ismerünk, amely megfelelő aktív tartalom fejlesztésére, azonban ma vitathatatlanul a legnépszerűbb a php. Az aktivátor nyelvének kiválasztásánál elsődleges szempontnak kellene lennie a biztonságnak és a kijelölt feladatnak, ma azonban inkább az a jellemző, hogy szokásjog alapján választanak nyelvet.

# 4. Rendszer tervezése gyakorlati szemszögből

Tételezzük fel, hogy rendszerünk viszonylag kis mennyiségű adatot fog tárolni (legfeljebb néhány száz MByte), ezért bátran használhatjuk az egyik legnépszerűbb nyílt forrású SQL szervert, a PostgreSQL-t. Az igényeinket tökéletesen kielégíti, akár bonyolultabb lekérdezések is könnyen összeállíthatók vele, és megfelelő hangolással elegendően gyors is. Mivel a leendő rendszer meglehetősen nagy terhelésnek lesz kitéve, és mert a jail nem ad tökéletes leválasztást, ezért célszerű a webszervert és az adatszolgáltatót külön gépre telepíteni, és akár több adatszervert vagy webszervert is használni. A két rendszer közé a jó reakcióidő és a viszonylag nagy adatmozgás miatt célszerű nagyobb sávszélességű vonalat tervezni. Ha a két rendszer fizikailag egymás mellé tehető, akkor a legolcsóbb egy 100Mbs sebességű ethernet kapcsolat.

A processzor és a hálózat túlterhelésének elkerülésére az adatszolgáltatón futtatandó lekérdezéseket a fejlesztőkkel optimalizáltatni kell. A fejlesztőkkel egyeztetve a virtuális tervezők arra a megállapításra jutottak, hogy megfelelő ellenőrzés mellett a php4 megfelelő fejlesztőeszköz lesz. Beállításait a biztonságot szem előtt tartva kell elvégezni (a php sok olyan lehetőséget ad, melyek a programozást kényelmetlenebbé, a rendszert biztonságosabbá teszik).

Az alkalmazás egyik legkritikusabb pontja a rendszert elérők azonosításának és jogosultságainak kérdése. Vegyünk egy egyszerű példarendszert négy felhasználói szinttel:

- a webszervert futtató felhasználó (u0)
- a rendszer karbantartói és automatizmusai (u1)
- a viszonteladók (u2)
- a névtelen felhasználók (u3)

és a rendszerben jelen lévő adatokat fontosság szerint osszuk négy szintre (’s’ jelöli az állományrendszerben elérhető adatokat, ’d’ pedig az adatbázisban megtalálhatóakat):

- a honlap grafikai és szöveges elemei, a bevezető flash animáció (s1)
- a termékek adatai (d1)
- a rendeléslista (d2)
- a raktári rendszer adatai (d3)

Miután meghatároztuk, hogy milyen felhasználói szintek, és milyen adatok vannak, már csak a hozzáférési jogokat kell megfelelően kiosztani. Az alábbi táblázatok két alapvetően különböző álláspontot szemléltetnek.

| | u0 | u1 | u2 | u3 |
|---|---|---|---|---|
| s1: | r | rw | r | r |
| d1: | rw | rw | r | r |
| d2: | rw | rw | rw | w |
| d3: | rw | rw | r | - |

1. táblázat. Hibás jogosultsági tábla

| | u0 | u1 | u2 | u3 |
|---|---|---|---|---|
| s1: | r | rw | r | r |
| d1: | - | rw | r | r |
| d2: | - | rw | rw | w |
| d3: | - | rw | r | - |

2. táblázat. Helyes jogosultsági tábla

Vizsgáljuk meg, mi a probléma az első jogosultságrendszerrel. Ha a rendszer felhasználóinak adatokat kell módosítani vagy lekérdezni az adatbázisból, akkor oda valakinek hozzáférési jogosultságra van szüksége. Gyakori hibás hozzáállás az, hogy

a webszerveren futó aktív tartalom minden adathoz minden hozzáférési jogot birtokol, és ő dönti el, hogy melyik felhasználónak mit tesz meg. Ez két problémát vet fel: amennyiben a rendszert meg lehet téveszteni, vagyis nem megfelelő az azonosítási rendszer, akkor a csibész más felhasználóként tud a rendszerhez férni. A komolyabb probléma az, hogy akár a webszerverben, akár az aktivátorban van hiba, a csibészek rossz esetben hozzá tudnak férni minden adathoz. Mi lehet a megfelelő megoldás? Az aktivátornak nem kell ismernie az adatbázis eléréséhez szükséges authentikációs tokent, hiszen neki nem kell elérnie azt. Mikor a felhasználó azonosítja magát, valamilyen formában megadhatja azt (jelszó esetén például bekérhető), így biztosított az adatbázishoz való kapcsolódás. Ebben az esetben a felhasználó semmilyen módon nem tudja rászedni az aktivátort illetéktelen műveletre, hiszen annak nincs joga hozzá, az authentikáció az adatbázis szinten (is) lezajlik, és ott történik az authorizáció. A felhasználó biztonságos azonosítására érdemes egy egyszerű CA összeállítása (openssl segítségével egyszerűen megoldható), és a jogosult felhasználóknak kiadni egy-egy digitális azonosítót.

# 5. Az aktív tartalom fejlesztése

Az aktív webkiszolgáló megfelelő biztosításához elengedhetetlen az aktivátor körültekintő tervezése és fejlesztése. Ha az adott feladat látszólag nem is indokolja a komoly energiabefektetést, akkor is érdemes szem előtt tartani az esetleges feltörésből adódó várható anyagi és erkölcsi károkat. Biztonságos rendszer fejlesztésében járatlan fejlesztőknek javaslom az adott nyelv biztonsági funkcióiról szóló leírások és egyéb dokumentációk alapos átolvasását. A gyakrabban használt nyelvekről az Interneten elérhető olyan összefoglaló, amely az esetleges biztonsági kockázatokkal foglalkozik. Komolyabb rendszer tervezésének megkezdése előtt érdemes legalább átolvasni a CC [9] kapcsolódó részeit. Ha a rendszer fejlesztését nem is tökéletesen CC szerint folytatják, akkor is rávilágít olyan kérdésekre, amelyek biztonsági területen tapasztalatlan fejlesztők fejében fel sem merültek volna. A tervezésnél figyelembe kell venni, hogy a rendszer várhatóan mekkora támadásnak lesz kitéve, és ez alapján kell megtervezni a védelmi rendszert. A tervben meg kell határozni, hogy ki milyen információkhoz férhet hozzá és milyen azonosítást használ. A rendszer hibatűréséről sem szabad megfeledkezni.

A hibakezelés megtervezésénél érzékeny adatok kezelése esetén érdemesebb a biztonsági rendszereknél használt elveket felhasználni. Ismeretlen hiba esetén a rendszer inkább álljon le, mint hibásan működjön tovább. Ilyen esetben érdemes a rendszerbe valamilyen önellenőrző folyamatot is beépíteni, amely képes érzékelni, ha az adatbázisba idegen adatok jutnak. Javaslom az erős titkosítási eljárásokat használatát, mint tudjuk, az Internet nem túl biztonságos. Soha nem szabad megfelelő teszt nélkül egy ilyen rendszert használni. Nem szorosan vett biztonsági kérdés, de igen fontos, hogy a rendszer tervezési és fejlesztési dokumentációját minden esetben el kell készíteni vagy készíttetni a későbbi továbbfejlesztés lehetővé tétele érdekében.

# 6. A kiszolgáló telepítése, beállításai

A korábban kiválasztott rendszer telepítése meglehetősen egyszerű. Vannak ugyan disztribúciós különbségek, nagy általánosságban elmondható azonban, hogy a webszerver tartalmazni fog minden olyan funkciót, amelyet a terjesztők fontosnak ítéltek.

Ezen a beállítások módosításával sem lehet megnyugtatóan változtatni, mivel az ilyen finomhangolást csak fordítás közben lehet elvégezni. Javaslom tehát a kiválasztott eszköz forrásának letöltését, és a valóban használt opciók beállítása után annak újrafordítását. Amennyiben a rendszer lehetővé teszi, fordításra praktikus a StackGuard [10] használata, amely bizonyos támadások kivitelezését alaposan megnehezíti (nem teszi lehetlenné!). Ha elkészült a futtatandó szerverprogram, elő kell állítanunk a megfelelő futási környezetet a számára. Ez alapesetben az operációs rendszerből áll, melynek beállításairól korábban szóltunk. Ha nagyobb biztonságra törekszünk, a programnak érdemes egy külön futtató környezetet összeállítani (jail vagy sandbox). Ez hatalmasat dob a rendszer biztonságán, hiszen az adott alrendszer kompromittálódása esetén a csibészek jóval nehezebben, vagy egyáltalán nem tudnak a többi alrendszerek adataihoz férkőzni. Itt is figyelmeztetnem kell azonban mindenkit, a jail nem csodaszer. Hibás kivitelezése vagy kernelhiba esetén elérhetőek a külső adatok. Helyes felépítése túlmutat előadásom keretein [11], a megfelelő eljárásokat Zámbó Marcell előadásában részletesen ismerteti. Amennyiben a rendszerünk futásra kész, akkor már csak a beállítások vannak hátra.

Ha valaki a cím láttán abban reménykedett, hogy itt kész mintákat kap, sajnos ki kell ábrándítsam. Véleményem szerint egy működő konfigurációs példa gyakran rossz irányba viszi a lusta vagy elfoglalt rendszerépítőt. Megtörténhet, hogy a példa által használt beállítások maradnak az éles rendszeren is. Mivel azonban a dolgunk egy valóban biztonságos kiszolgáló építése, így át kell olvasni a teljes rendszer dokumentációt, és mindenkinek egyedi igényei szerint kell beállítania a rendszerét. Ha például a webszerver moduláris felépítésű, akkor kizárólag azokat a modulokat szabad betölteni, amelyek valóban használtak. Így elkerülhetők a nem használt, de hibás modulokon keresztüli támadások. A rendszer biztonsági funkcióit kitüntetett figyelemmel kell átvizsgálni és beállítani (például a php futási hibák kiírása tiltható). A rendszer naplóit érdemes egy távoli rendszerre is továbbítani és archiválni, így a rendszer esetleges kompromittálódása esetén a mentett naplókból rekonstruálható a betörés.

# 7. Üzemeltetési kérdések

Miután a rendszer üzemképessé vált, már csak arról kell gondoskodnunk, hogy szerverünknek nagy megbízhatósággal üzemeljen kell. Ennek megkönnyítése érdekében álljon itt néhány jótanács.

Minden alrendszert (csomagot) érdemes a lehető legutóbbi stabil változaton tartani, mivel az új verzióban kijavíthattak olyan biztonsági hibákat is, amelyeket még nem publikáltak. Az általunk választott rendszerek biztonsági listáit és a BugTraq-et [12] mindenképpen érdemes olvasni, mivel itt lehet legkorábban értesülni a napfényre került biztonsági hibákról. Azok a hibák, amelyeket a csibészek találnak meg, és nem publikálnak, sajnos folyamatos veszélyt jelentenek a rendszereinkre. Itt legfeljebb annyit tehetünk, hogy az eddigieknek megfelelően megnehezítjük a dolgukat, és reménykedünk. Ne áltassunk magunkat azzal, hogy a mi rendszerünk sérthetetlen.

A kiszolgálás zökkenőmentessége miatt a rendszer nem adat részéről készítsünk minden frissítés előtt teljes mentést, vagy egy tesztrendszeren próbáljuk meg a frissítés utáni állapotot, és csak ezután kezdjünk az éles rendszer frissítésébe. Határozzuk meg, hogy milyen gyors a rendszeren lévő adatok változási üteme, és készítsünk mentési tervet. A mentési tervet tartsuk be. Tesztrendszeren kísérletezzünk a mentésből való visszaállással, így tökéletesíthetjük a mentési rendszert, és összeállíthatunk egy katasztrófatervet.

A rendszeren lévő állományok módosításának az ellenőrzésére használjunk valamilyen digitális ujjlenyomat-ellenőrző eszközt (például Tripwire). Ezzel, ha nem is biztosan, de viszonylag nagy biztonsággal eldönthető, hogy a rendszer állományai sérültek-e. Ha igen, megállapítható, hogy mi, és a hibás állományok visszaállíthatóak. Valamely konfigurációs állomány vagy bináris változása esetén azonban sürgősen ki kell deríteni, hogy hogyan hatolhatott be az illető a rendszerbe. Ekkor nagy segítségünkre lesznek a naplóállományok.

Egy közepesen terhelt rendszer is olyan mennyiségű naplóállományt állít elő, amelyet egy embernek naponta átnézni vagy lehetetlen, vagy nagyon unalmas. A naplók előszűrésére a legegyszerűbb megoldás a klasszikus „Artificial Ignorance" módszer, amely a már ismert sorokat kiszűri, így a naplóelemzőhöz csak a különös tartalmú sorok jutnak el. Ez ugyan adatvesztés, mivel a naplók mennyiségéből illetve frekvenciájából is lehet következtetéseket levonni, de itt ez most nem számít.

Ha a csibészek a rendszerbe jutva lecserélik a weblapot, arról célszerű elsőként értesülni. Érdemes olyan rendszert telepíteni egy távoli gépre, amely időnként ellenőrzi a főbb oldalak állapotát, és azok változása esetén értesítést küld a rendszergazdának. Ez statikus tartalom esetén nagyon egyszerű, hiszen az adott lapok helyes tartalmát lemásolva az ellenőrzés pofonegyszerűen elvégezhető. A dinamikus tartalmú lapok esetében kissé összetettebb a helyzet. Ilyen esetben érdemes az aktivátor tervezésénél figyelembe venni az ellenőrzés igényét, és olyan vízjellel látni el a generált oldalakat, amely ellenőrizhető, de az esetleges támadók számára nehezen érzékelhető. Egy nagyon egyszerű megoldás lehet például, hogy valamely HTML tag minden esetben egy előre meghatározott módon kerül az oldalra. Ez a megoldás természetesen nem véd az adatbázisba vitt hibás adatok általi deface ellen, de arra is kitalálható megoldás.

# 8. Már létező rendszer védelme

Azok a már létező rendszerek, melyek elkészítésénél a fenti szempontok valamelyikét nem vették figyelembe, nagyon nehezen védhetők. Le lehet írni, hogy mi a normális működés, ezt tűzfallal meg lehet próbálni kikényszeríteni, de a rendszer védelme nem lesz tökéletes. Általánosságban elmondható, hogy az ilyen rendszert részben vagy egészben újra kell tervezni és fejleszteni. Ezért érdemes a project elejétől figyelembe venni a biztonság szempontjait.

# 9. Idegen szavak jegyzéke

**kompromittálódott szerver** Olyan szerver, amelynek a biztonságával kapcsolatban kérdés merült fel. Ha a valódi behatolás nem is bizonyítható, a rendszert érdemes újratelepíteni vagy erősen szemmel tartani. Szükség esetén a rendszer egyes kiszolgáló programjait is érdemes lecserélni. Mivel a biztonságtechnikával foglalkozó emberek szakmájukból adódóan paranoiások, így a kompromittálódott szerver gyakorlatilag a feltört rendszer szinonimája.

**deface** A weblap rosszindulatú lecserélése valamely nem odaillő tartalommal. A csibészek ezzel bizonyítják a világnak, hogy az adott rendszer fölött sikerült átvenniük a hatalmat.

**aktivátor** A szerveroldali dinamikus tartalom meghajtóprogramjainak összefoglaló neve. Mivel ez saját kifejezés, így a szakirodalomban nem található meg. Az

alábbi szószerkezetet helyettesíti: „a program, amely a weblapot dinamikussá teszi”.

**certificate** Digitális azonosító. Egy aláírt digitális kulcsból és néhány egyéb adatból álló struktúra, amely segítségével az ssl-re épülő protokollok használatánál a két kommunikáló fél megbízhatóan azonosíthatja a másik oldalt.

**authentikáció** Azonosítás. Minimális szintű védelmi rendszer is authentikálja a felhasználóját mielőtt az bármilyen erőforrást igénybe vehet. Az egyik legegyszerűbb formája a jelszavas azonosítás, bonyolultabbak például az S-Key, CCard, SmartCard, Certificate stb.

**authorizáció** Meghatalmazás, felhatalmazás. Ha a rendszer megkülönböztet valamilyen hozzáférési szinteket, akkor a felhasználót annak azonosítása után a rendszer felhatalmazza bizonyos erőforrások használatára. Ezek lehetnek állomány- vagy objektum-hozzáférési jogosultságok, beléptető rendszer esetén akár egy ajtó kinyitása.

**sandbox** $\Rightarrow$ jail

**jail** Olyan futtató környezet, melyben csak egy adott program vagy programcsoport futásához szükséges állományok vannak. Használata lehetővé teszi az egyes hálózati démonok szeparálását. Ha megfelelő hozzáértéssel állítják össze, és a kernel is megfelelő (gyakran kissé módosítani kell), akkor nem lehet kitörni belőle.

# Hivatkozások

[1] Az OpenWall project honlapja: *http://www.openwall.com*

[2] A HAP honlapja:
*http://www.theaimsgroup.com/∼hlein/hap-linux*

[3] Linux Intrusion Detection System *http://www.lids.org*

[4] grsecurity project honlapja *http://www.getrewted.net*

[5] A FreeSWAN project honlapja *http://www.freeswan.org*

[6] User Mode Linux project
*http://user-mode-linux.sourceforge.net*

[7] A VMWare honlapja *http://www.vmware.com*

[8] Az Argante project honlapja *http://www.argante.org*

[9] Common Criteria project *http://csrc.nist.gov/cc*

[10] Immunix project: StackGuard *http://www.immunix.org*

[11] Zámbó Marcell, Biztonságos chroot környezet kialakítása
*III. GNU/Linux Konferencia kiadványa*, LME, [Bp., 2001.]

[12] BugTraq levelezőlista *http://www.securityfocus.com*

# Unix, vagy nem Unix?
# A Unix filozófia jövője a Desktop rendszerek korában

Németh László <nemethl@mailbox.hu>

2001.08.30.

**Kivonat**

A Unix filozófia része az átfogó (szöveges) fájlkezelés, a héj, a csőbe köthető segédprogramok, a reguláris kifejezések, az eszközfájlok, és a többi. Kérdés, hogy azok a lehetőségek, amelyeket ezek az eszközök biztosítanak, hogyan állják meg a helyüket a csillogó-villogó grafikus felületek, illetve alkalmazások korában?

## Tartalomjegyzék

**1. A Unix sikere** **64**
- 1.1. Kedvező ár/teljesítmény arány . . . 64
- 1.2. Hordozhatóság . . . 64
- 1.3. Programozási környezet . . . 65

**2. Unix-e a Linux?** **65**

**3. A Unix parancsnyelv** **65**

**4. Példák** **66**
- 4.1. Egyszerű példák . . . 66
- 4.2. KWIC . . . 66
- 4.3. Megjegyezhető véletlen jelszavak . . . 71
- 4.4. Web-tools . . . 72

**5. Összefoglalás** **75**

# 1. A Unix sikere

A Unix operációs rendszer meglepően sikeresnek mondhatók az informatikában: több, mint 30 éve működik ugyanazon alapelvek alapján. Ha a siker okait firtatjuk, a következőket kell kiemelnünk: a Unix egy olcsó, rendkívüli mértékben hordozható és programozható általános célú operációs rendszer.

## 1.1. Kedvező ár/teljesítmény arány

A gazdaságos, költségtakarékos, stb. jelzők is az olcsóság pozitív megfogalmazását jelentik. A Unix assembly változata szükségből egy már kifutóban lévő géptípusra, egy DEC PDP-7-esre készült, mivel a Ken Thompson szerezte Space Travel játék egy menete 75 dollárnyi CPU időt vesztegetett el a modernebb, és leterheltebb GE-635-ösön[5]. A PDP-7-es gép kezdetben kiválóan megfelelt a játékra és egy kisméretű interaktív operációs rendszer írására. Pár évvel később a korszerűbb PDP-11-en olyan valódi időosztásos rendszer készült el, ami tizedére faragta le egy terminál létesítésének és üzemeltetésének költségeit [6], lehetővé téve, hogy az egyetemi oktatásban, a nagy és drága számítógépközpontok árnyékában is természetessé váljék az informatika.

## 1.2. Hordozhatóság

Az időosztást tekintve a Unix nem az első operációs rendszer, hiszen Kemény János és Thomas Kurtz 1963-ban készítette el a Dartmouth Time-Sharing System (DTSS) elődjét egy GE-235-ösön, ami 1964-ben a szintén általuk kidolgozott (és nem interpretált, hanem fordított) BASIC nyelv segítségével a hallgatók rendelkezésére állt. ’65-ben már a környékbeli középiskolákba is terminálok kerültek, ’66-ban pedig egy ajándék GE-635-ösön kidolgozásra került a DTTS, ami kb. 1 MIPS ($10^6$ művelet/másodperc) sebesség mellett egyidejűleg 200 felhasználót tudott kiszolgálni [1].

Míg a BASIC nyelv változataiban ma is él (Visual Basic), a DTSS-ről ez nem mondható el, a legtöbb assembly-ben megvalósított rendszerhez hasonlóan.

1973-ban Ken Thompson Dennis Ritchie segítségével C nyelvre ültette át a Unix-ot, ezzel elkészítve az első hordozható forráskódú operációs rendszert. A C nyelv Dennis Ritchie műve. A C gépfüggetlen és mégis gépközeli programírásra szolgál mind a mai napig (Ritchie és Thompson megunta a programok átvitelével járó bonyodalmakat).

Rosszmájúak szerint a Unix igazi sikerének oka ez [7], illetve, hogy az AT&T az akkori monopoltörvények miatt nem csinálhatott nagy üzletet a Unix-ból: az oktatás számára ingyenesen hozzáférhető tette a forráskódot, egyébként 99 dollárért lehetett hozzájutni. Ken Thompson egyetemi kurzusokon sorról sorra ismertette az operációs rendszer forráskódját. Az, hogy ezt megtehette jócskán köszönhető a C nyelvű megvalósításnak! A Unix nem csak hordozhatóvá vált, hanem érthetővé is a széles szakma számára. Thompson-t olyan tanítványok hallgatták, mint Bill Joy (BSD Unix és a Java specifikáció társszerzője, az Internetet megalapozó BSD Unix-os TCP/IP implementáció, valamint a C-shell, NFS, vi, stb. szerzője). A Berkeley egyetem forradalmi BSD Unix-változata is kezdetben a gyenge BSD engedéllyel jelent meg, ami lehetővé tette a forráskód szabad felhasználását [6].

### 1.3. Programozási környezet

A Unix nem rendelkezik BASIC-szerű programnyelvvel, mint a DTSS, vagy mint az első mikroszámítógépek (a Python jóval későbbi fejlesztés). Nem jellemzi az assembly használata sem, mégis egyedülállóan gazdag programozási környezet. A Basic helyett a héjat, az assembly helyett a C nyelvet találjuk meg egy Unix-ban.

Brian Kernighan-t idézve a héj (héj) nem interaktív parancsértelmező, hanem valójában egy programnyelv [8]. Magasabbszintű a BASIC-nél is, hiszen utasításai sokszor külön programok: a Unix operációs rendszer segédprogramjai. Az érdemi munka ezen a programozási nyelven keresztül folyik, ha egy terminál, vagy terminál-emulációs program előtt ülünk. Kettős arca van: hiszen teljes mértékben interaktív lehet, soronként végrehajtva parancsainkat, vagy pedig ha ezeket egy állományba (héjprogram=héj forráskód) rögzítjük, akkor képes egyszerre végrehajtani azt.

A rendszer használata szorosan összeforrt a programozással. Tévedés azonban azt hinni, hogy ezért csak programozók képesek használni. Guido van Rossum, a Python nyelv készítőjének függőben lévő Computer Programming for Everybody (CP4E) projektje gyakorlatilag már több évtizede megvalósult, pontosabban megvalósulhatott volna a Unix héjjal. Ami hiányzik még a rendszerből, és ami igazából meggátolja, hogy mindenki programozójává váljon egy Unix rendszernek, hogy nincs a kezdők számára kellően dokumentálva a rendszer. Magyar nyelven [8] a legteljesebb bevezető a héj használatába.

## 2. Unix-e a Linux?

Mivel végig Unix-ról beszélünk, felmerül a kérdés, hogy a Linux mennyiben tekinthető Unix-nak. Linus Torvalds szerint a Linux Unix-szerű, mivel a hordozhatóság miatt van egy Unix-kompatibilis felülete, valamint kialakítható rajta Unix környezet a héj-, és a Unix segédprogramokkal, de nem Unix-változat, mivel a Linux rendszermag nem tartalmaz BSD Unix-ból örökölt kódot [2].

A Linux terjesztések jelenleg nagymértékben a Unix-os oldalát támogatják a Linuxnak, alapértelmezett segédprogramoknak tekintve a GNU Unix segédprogramokat (*bash* héj, *textutils*, *fileutils*, *sh-utils*, *findutils*, stb.)

Kapcsolódva az előző szakasz záró megjegyzéséhez, szomorú, hogy a Linux erről az oldaláról kevéssé dokumentált. A Linux terjesztések alapértelmezett héjához, a *bash*-hoz készült HOGYAN elképesztően szegényes, a *bash* kézikönyv pedig túl tömör, hogy bárki egyszerűen megtanulja az alapján a héj használatát.

## 3. A Unix parancsnyelv

A Unix parancsnyelv, másnéven héj közel egyidős a Unix-szal. A Thompson héj, az eredeti */bin/sh* még a Multics-ból vett át ötleteket, de olyan eredeti megoldásokkal bővült, mint a csövek, valamint a szabályos kifejezések felhasználása a héjból meghívható segédprogramoknál.

A cső nem teljesen Unix találmány: a DTSS kommunikációs fájlai hasonló célt szolgáltak. Még régebbről ismert a számítástechnikában a korutinok fogalma, amit gyakran megfeleltetnek a csőhálózatba kötött programoknak. Ami egyedülálló, az az egyszerű szintaxis, ahogy ez megvalósítható a Unixban. Továbbá az egyszerűen kezelhető fájlrendszer, az átirányítások, és a héj helyettesítései maximálisan kibővítik a héj és a csövek felhasználási területét.

# 4. Példák

A következő példák célja, hogy a Unix filozófia erejét alátámasszák. Az első példa szokásos Unix parancsokat mutat be. A második példa egy klasszikus számítástechnikai feladat, a Parnas-féle KWIC indexkészítés, amelyet egy egysoros héjprogrammal is megvalósítunk. A harmadik megjegyezhető véletlen jelszavak generálását mutatja be. Végül kibővítjük a Unix segédprogram-készletét egy angol értelmező kéziszótárral, és egy élőnyelvi fordítóprogrammal.

## 4.1. Egyszerű példák

Készítsünk lemezképet egy floppylemezről:

```
$ cat /dev/fd0 > lemezkép
```

Megtehettük volna ezt a *cp* paranccsal is, de így természetesebb.
Másoljuk az alkönyvtárakban lévő *txt* állományokat egy helyre:

```
$ cp */*.txt /tmp
```

Konvertáljuk a könyvtárban lévő állományok nevét kisbetűsre:

```
$ for i in *; do mv $i $(echo -n $i | tr A-Z a-z); done
```

Konvertáljuk a könyvtárban lévő összes BMP állományt feleakkora szélességű és negyedakkora magasságú JPG állományra:

```
$ for i in *.bmp
do convert -geometry 50%:25% $i $(basename $i bmp)gif
done
```

## 4.2. KWIC

A KWIC (Keyword in Context) mutatók a könyvtárlátogatók hasznos segédeszközei, mivel nem csak azt mutatják meg, hogy a keresett tárgyszó milyen műben fordul elő, hanem azt is, hogy milyen szövegkörnyezetben. Dave Parnas 1972-es cikke [3] a KWIC egy olyan változatát veszi példának, ami pl. könyvcímek keresését könnyíti meg akkor, ha csak az egyik címben előforduló szót ismerjük. A példafeladat lényege a következő: adott címeknek a sorozata, minden cím külön sorban kerül elhelyezésre. Minden egyes sor szavait körkörösen eltolva (vagyis az első szavakat hátra küldve) további sorokat képezünk. Egy *n*-szavas címből tehát *n-1* további sort kapunk, végül az összes sort ábécé szerint rendezzük. Parnas emlékezetes kijelentése szerint bármely gyakorlott programozó két hét alatt megbirkózik ezzel a problémával. Azóta persze sokat fejlődött a világ, pl. C++ nyelven 80-90 soros programmal megoldható a probléma [4], ha az adott nyelvnek megfelelő módon programozunk.

#### Megoldás *awk*-kal

Tekintsük a Unix lehetőségeit. Első esetben az *awk*, mint jól programozható sorfeldolgozó jön számításba. A következő *circ.awk* állományban tárolt program a szabványos bemenetének minden sorára elvégzi a körkörös eltolás-sorozatot, és az eredményt kiírja a szabványos kimenetére:

```
{
    for(i=1; i<=NF; i++) {
        text = $i
        for (j=i+1; j<=NF; j++) {
            text = text " " $j
        }
        text = text ","
        for (j=1; j<i; j++) {
            text = text " " $j
        }
        print text
    }
}
```

A program teszteléséhez futtatjuk az állományt az *awk*-kal.

```
$ awk -f circ.awk
a b c
[Ctrl-D]
a b c,
b c, a
c, a b
```

A példában a bemenet most a terminál, de persze ha valóban használnánk ezt a programot, gyakoribb lenne az ilyen formában történő meghívás:

```
$ cat cim.txt | awk -f circ.awk > cim.kwic
```

A KWIC pobléma megoldásához már csak az ábécésorrendbe történő rendezés hiányzik: egyszerűen csak átadjuk a kimenetet a *sort* programnak:

```
$ awk -f circ.awk | sort
c b a
[Ctrl-D]
a, b c
b a, c
c b a,
```

Ha a programunk a tesztek során működőképesnek bizonyult, el is helyezhetjük egy futtatható állományba:

```
#!/bin/sh
# Parnas-féle KWIC I. megoldás
awk '
{
    for(i=1; i<=NF; i++) {
        text = $i
        for (j=i+1; j<=NF; j++) {
            text = text " " $j
        }
        text = text ","
```

```
        for (j=1; j<i; j++) {
            text = text " " $j
        }
        print text
    }
}' | sort | uniq
```

A forrásban szereplő *uniq* nem csinál mást, mint egy sort hagy csak meg az ismétlődő sorokból.

**Megoldás *awk* nélkül**

Az *awk* hatékony, de a héj mellett megint egy új nyelvet jelent, ami ráadásul inkább a C-re emlékeztet, mint a héjra. Keressünk egy másik, bizonyos szempontból jobb megoldást.

A héj rendelkezik egy olyan utasítással, ami nagyban megkönnyíti a körkörös eltolás kivitelezését. Ez a *shift*, ami az aktuális héjprogram paramétereit tolja el balra (szám paramétert is elfogad az utasítás, de erre most nem lesz szükségünk). Készíthetünk egy *circ* nevű héjprogramot, ami a paramétereit körkörösen eltolja:

```
#!/bin/sh
for i
do
    echo $*, $j
    j="$j $1"
    shift
done
```

Például:

```
$ circ a b c
a b c,
b c, a
c, a b
```

Most már csak össze kell raknunk a megfelelő csővezetéket, ami a Parnas-féle KWIC probléma előzőhöz hasonló megoldását jelenti. Ehhez felhasználjuk a *xargs* programot, ami egy szövegfájl sorait paraméterként adja át a megadott programnak. A *-l1* kapcsoló jelentése, hogy egyszerre csak egy sorból készítsen paraméterlistát:

```
$ echo A kutya meg a mája | xargs -l1 circ | sort
A kutya meg a mája,
a mája, A kutya meg
kutya meg a mája, A
mája, A kutya meg a
meg a mája, A kutya
```

Ez a megoldás 10–20-szor lassabb, mint az előző *awk* program, mivel a bemenet minden egyes sorára meghívunk egy alhéjat. Ami miatt mi mégis ezt preferáljuk, az az, hogy kicsi és egyszerű, valamint kellően rugalmas ahhoz, hogy könnyedén kiszűrjük például a KWIC listát feleslegesen duzzasztó névelőket és kötőszókat. Íme a módosított körkörös eltoló, a *circ2*:

```
#!/bin/sh
for i
do
    case $1 in
        [aA]|[aA]z|[eE]gy|és|vagy);;
        *) echo $*, $j;;
    esac
    j="$j $1"
    shift
done
```

```
$ echo A kutya meg a mája | xargs -l1 circ2 | sort
kutya meg a mája, A
mája, A kutya meg a
meg a mája, A kutya
```

Mindez héjprogrammal (visszatérve a *circ*-hez):

```
#!/bin/sh
# Parnas-féle KWIC II. megoldás
xargs -l1 circ | sort | uniq
```

Ha egy héjprogramba szeretnénk összegezni programjainkat, akkor a kézikönyv fellapozása után a következő megoldást kapjuk:

```
#!/bin/sh
# Parnas-féle KWIC III. megoldás
xargs -l1 sh -c '
for i
do
    echo $*, $j
    j="$j $1"
    shift
done' $0 | sort | uniq
```

Ez a megoldás bonyolultabb, mint az előző, mivel egy új kapcsolót kell megjegyeznünk (az *sh -c* kapcsolója), valamint azt, hogy a héj a *-c* kapcsoló megadása esetén az *$0* paramétert (ez rendszerint a héjprogram indítási nevét tartalmazó változó) tőlünk várja. Mi ide *$0*-át írtunk, továbbadva a KWIC héjprogramunk nevét az alhéjnak.

Miért bonyolítottuk el KWIC programunkat? Azért, hogy egy állományunk legyen kettő helyett, valamint hogy egy sorba írhassuk a Parnas-féle KWIC probléma megoldását (itt most nem látszik, de megszámolható, hogy pontosan 80 karakteres a program):

```
#!/bin/sh
xargs -l1 sh -c 'for i; do
    echo $*, $j; j="$j $1"; shift; done' x |sort
```

A tömör egysoros implementáció ellenére a II. megoldás a legegyszerűbb megoldás. Egészítsük ki ezt egy szabványos *-h*, illetve *–help* kapcsolóval:

```
#!/bin/sh
# Parnas-féle KWIC kapcsolókkal
case $# in
0) xargs -l1 circ | sort | uniq ;;
*)  case $1 in
    -h|--help)
    echo "Parnas-féle KWIC, a bemenet minden sorát
          körkörösen eltoló, és rendező program." ;;
    *) echo "Hibás paraméter" >/dev/stderr ;;
    esac
esac
```

Látható, hogy két egymásba ágyazott *case* szerkezettel készítettünk barátságosabb héjprogramot. A *$#* visszaadja a héjprogram paramétereinek a számát. Ha ez nem nulla, akkor megvizsgáljuk, hogy az első paraméter *-h*, vagy *–help*-e, és ebben az esetben jelenítünk meg leírást. Egyébként pedig a szabványos hibakimeneten jelenítünk meg egy hibaüzenetet.

**Túl szép a menyasszony**

Ha tesztállományokon futtatjuk az *awk*, és pusztán a héjjal megvalósított KWIC programokat, hamar rájövünk, hogy az utóbbi nem tökéletes. A fordított perjelek eltűnnek az adatállományból, az első dupla és az egyszeres idézőjel pedig a feldolgozás megszakadását eredményezi. A magyar nyelvben ez nem jelentene problémát, de az angolban az aposztróf könnyedén előfordulhat a címekben.

A kézikönyv fellapozásával kiderül, hogy ez a jelenség a *xargs* programhoz köthető, és ugyan kikapcsolható, de akkor a soronkénti feldolgozás nem működik (az egész bemenő állományt egy sorként van kezelve, hacsak a sorok nem nulla ASCII kódú karakterre végződnek). Közben előkerült egy másik probléma is a kézikönyvből: ha egy sor szóköz, vagy tabulátor karakterrel végződik, akkor a következő sort is hozzáveszi a *xargs*.

Ha a kellemetlen, de tipikus (a héj védőkaraktereivel kapcsolatos) problémát meg akarjuk oldani, a *sed* szűrőhöz kell fordulnunk. Egy *sed* paranccsal levédjük a fordított perjelet, és a két idézőjelet, úgy, hogy egy további fordított perjelet helyezünk eléjük. Egy másik *sed* paranccsal pedig a sorvégi szóközöket, illetve tabulátorokat töröljük, elintézve a *xargs* másik turpisságát:

```
#!/bin/sh
# awk kompatibilis KWIC héjprogram
sed "s/\(['"'\"]\)/\\\1/g
     s/[  ]*$//' | xargs -l1 circ | sort | uniq
```

A *sed* paraméteréül megadott, szabályos kifejezést tartalmazó mintát (sajnos) trükkös módon kellett levédeni: az egyszeres idézőjelet dupla idézőjel között, a duplát egyszeres idézőjelek között adhatjuk meg.

A példában a prototípus- és a valódi programkészítés határán mozogtunk a héjjal: *awk* és *sed* nélkül is jó prototípust alkottunk, de a minden körülmények közötti helyes működéshez szükség van a *sed*-re, vagy az *awk*-ra is.

## 4.3. Megjegyezhető véletlen jelszavak

Fontos biztonsági szempont a megfelelő jelszavak használata. A következő példa megjegyezhető véletlen jelszavak előállítását és kezelését taglalja.

A Linux két olyan eszközfájllal is rendelkezik, amiből véletlen karaktersorozatot olvashatunk ki. A */dev/random* a megbízhatóbb, mivel teljes mértékben egy entrópiatáron alapul. Az entrópiatárat a Linux rendszermag kezeli az eszközmeghajtók által detektált zaj alapján. Jellemzője, hogy gyorsan kiürül, így előfordul, hogy sokáig várnak a hozzá kapcsolódó folyamatok a karakterekre. A másik eszközfájl, a */dev/urandom* úgy oldja meg a várakoztatás problémáját, hogy az entrópiatár kiürülése után egy álvéletlenszám-generátorral szolgáltatja a véletlen karaktereket. A példában az utóbbit fogjuk használni.

Az első feladat, hogy a vezérlő, és kiterjesztett ASCII karaktereket is tartalmazó karakterfolyamot megszűrjük:

```
$ tr -cd a-zA-Z0-9 < /dev/urandom
FstdW4FiPJDxpLen1sgkQt...[Ctrl-C]
```

A *tr* szűrőt többnyire karakterkonverzióra (*tr*anslate) használjuk, de a *-d* kapcsolóval a megadott karakterek törlésére is lehetőség van vele. A *-c* kapcsolóval a megadott karaktertartomány komplementerébe eső karakterek törölhetők, tehát gyakorlatilag egy olyan szűrőt készítettünk, ami az angol ábécé betűit, és számokat szolgáltat véletlenszerűen.

Ennyi (végtelen) karakterre nincs szükség a jelszóhoz. Az első nyolcat lecsippenthetjük a *dd* szűrő megfelelő paraméterezésével:

```
$ tr -cd a-zA-Z0-9 < /dev/urandom | dd ibs=8 count=1
KcjESXBf8+0 records in
0+1 records out
```

Az *ibs*-t (input buffer size) 8 bájtra állítjuk, és ebből egyet olvasunk csak be (*count=1*), majd a csővezeték befejezi működését.

A hibakimenetet érdemes átirányítani az univerzális adatnyelőbe:

```
$ tr -cd a-zA-Z0-9 < /dev/urandom |
    dd ibs=1 count=8 2>/dev/null
```

Látszólag minden kimenet eltűnt, mivel hiányzik a sorvége karakter. Egy *echo*-val pótolható.

```
$ tr -cd a-zA-Z0-9 < /dev/urandom |
    dd ibs=1 count=8 2>/dev/null; echo
lsgkCwTl
```

A fenti jelszavak véletlenek, de nem megjegyezhetők. A következő *flex* forrás egy olyan lexikai elemzőt állít elő, ami magas, vagy mély hangrendű, váltakozva magán-, illetve mássalhangzót (a magyar többjegyűeket is) tartalmazó szöveget szűr ki a bemenetéből:

```
/* megjegyezhető szűrő */
%option noyywrap
%s MASSALMAGAS MASSALMELY MAGAN MAGAS MELY
MINTA [bcdfghjklmnprstvz]|"cs"|"gy"|"ny"|"sz"|"ty"|"zs"
```

```
%%
<INITIAL>{MINTA} { printf("%s",yytext); BEGIN(MAGAN); }
<MASSALMAGAS>{MINTA} { printf("%s",yytext); BEGIN(MAGAS); }
<MASSALMELY>{MINTA} { printf("%s",yytext); BEGIN(MELY); }
<MAGAN,MAGAS>[ei] { printf("%s",yytext); BEGIN(MASSALMAGAS); }
<MAGAN,MELY>[aou] { printf("%s",yytext); BEGIN(MASSALMELY); }
.
```

Fordítsuk le, és futtassuk:

```
$ flex jelszo.flex
$ gcc -o jelszo -lfl lex.yy.c
$ tr -cd a-zA-Z0-9 < /dev/urandom | jelszo |
    dd ibs=1 count=10 2>/dev/null; echo
napodanoho
```

A *tr*-re azért van szükségünk még elsősorban, hogy nagyobb valószínűséggel kerüljenek többjegyű mássalhangzók a generált jelszavakba.

Készítsünk az előző parancsból egy héjprogramot *jel* néven, és generáljunk le 10 darab megjegyezhető véletlen jelszót:

```
$ for i in ‘seq 10‘; do jel; done
jemezidisi
repimepepi
tifijipepe
hucamugajo
kahujakoto
ridedejive
bucaponugu
suganurugo
jujajufuto
pojojobahu
```

A jelszavak irónikusak, és nem igazán megjegyezhetők még, de a *flex* kellően flexibilis ahhoz, hogy a magyar nyelv nyelvtani szabályait jobban figyelembe vevő megoldást készítsünk.

A következő lépés – ami itt már nem kerül megvalósításra – egy olyan héjprogram elkészítése, ami nem tárolja a jelszavakat, hanem a megadott azonosítójú felhasználóknak azonnal módosítja a jelszavát (*passwd –stdin*), és kinyomtatja külön oldalra (*pr -f*) a jelszavakat.[1]

## 4.4. Web-tools

A következő két héjprogram nem csinál mást, mint hálózati szolgáltatásokhoz fér hozzá, és az eredményt beilleszti a Unix környezetbe. Gyakorlatilag a szegény ember Unix-os XML-RPC/Soap implementációinak is tekinthetjük.

[1] Zárt kétrétegű indigós leporelló – ami magába rejti a kinyomtatott jelszót – kellően megnyugtatja a felhasználókat.

**Merriam-Webster szótár**

A következő héjprogram a Merriam-Webster szótár hálózati változatához a *links* (vagy a *lynx*) karakteres böngésző segítségével fér hozzá.

```
#!/bin/bash
# Merriam-Webster Dictionary
case $# in
0) echo -e "Merriam-Webster Dictionary
   Usage: mw keyword" ; exit;
esac

url=http://www.m-w.com/cgi-bin/dictionary
links -source $url?book=Dictionary\&va=$1 |
grep "^<b>" > /tmp/webster.$$
links -dump /tmp/webster.$$
rm /tmp/webster.$$
```

**Babelfish (SysTran) fordító**

A következő héjprogram az előzőhőz hasonló megoldással a SysTran fordítóhoz fér hozzá, kihasználva az Altavista ingyenes szolgáltatását.

```
#!/bin/bash
i="Babelfish (SysTran) Translator filter
human text translating from stdin to stdout
Usage: fish [option] < textfile
Options:
-c English to Chinese
-f English to French
-g English to German
-i English to Italian
-j English to Japan
-k English to Korean
-p English to Portuguese
-s English to Spanish
-ce Chinese to English
-fe French to English
-ge German to English
-ie Italian to English
-je Japan to English
-ke Korean to English
-pe Portuguese to English
-se Spanish to English
-fg French to German
-gf German to French"

lp=en_es

case $1 in
```

```
-h) echo -e "$i" ; exit ;;
-c) lp=en_zh ;;
-f) lp=en_fr ;;
-g) lp=en_de ;;
-i) lp=en_it ;;
-j) lp=en_ja ;;
-k) lp=en_ko ;;
-p) lp=en_pt ;;
-s) lp=en_es ;;
-ce) lp=zh_en ;;
-fe) lp=fr_en ;;
-ge) lp=de_en ;;
-ie) lp=it_en ;;
-je) lp=ja_en ;;
-ke) lp=ko_en ;;
-pe) lp=pt_en ;;
-se) lp=es_en ;;
-fg) lp=fr_de ;;
-gf) lp=de_fr ;;
esac

tr " \n<>\&\"" "++++++" > /tmp/fish.$$
url='http://babelfish.altavista.com/tr?doit=done\&urltext='
links -source $url"$(</tmp/fish.$$)"\&lp=$lp |
sed -n '/<td bgcolor="*white"*>/,/<\/td>/p
/<textarea[^>]*name="q">/,/^<\/textarea>/p' |
sed 's/<\/*[^>]*>//g'
rm -f /tmp/fish.$$
```

Teszteljük le a programjaink működését:

```
$ webster keyboard
   1 : a bank of keys on a musical instrument (as a piano) that usually
   consists of seven white and five raised black keys to the octave
   2 : an assemblage of systematically arranged keys by which a machine or
   device is operated
   3 : a board on which keys for locks are hung
$ webster keyboard | fish -i
    1:  una banca dei tasti su uno strumento musicale (come piano) che
consiste solitamente di sette bianchi e di cinque ha sollevato i tasti
neri all' ottava 2:  un raduno dei tasti sistematicamente organizzati
da cui una macchina o un dispositivo funzionata 3:  una scheda su
cui imposta per le serrature appesa
$ webster keyboard | fish -i | fish -ie
    1:  a bank of the keys on a musical instrument (like plan) that it
usually consists of seven white men and of five has raised the black
keys to the eighth 2:  a gathering of the systematically organized
keys from which a machine or a device it is worked 3:  a card on which
tax for the locks it is hung
```

Az eszközök működnek, az olaszból visszafordítás pedig kellően humoros.

# 5. Összefoglalás

A Unix parancsnyelve verhetetlen a mindennapos feladatok automatizálásában, a csöveknek és egyéb koncepcióinak köszönhetően egyedülállóan robusztus. A gyakorlatban prototípus-készítés szempontjából sem megvetendő, hiszen a prototípus gyakran tökéletes programnak bizonyul, és nem szükséges C, vagy egyéb programnyelvekhez fordulni.

# Hivatkozások

[1] A. Colvin: The DTSS Time-Sharing System,
*http://www.dartmouth.edu/ carlp/dtss.html*

[2] L. Torvalds: The Linux Edge, in Open Sources: Voices from the Open Source Revolution
*http://www.oreilly.com/catalog/opensources/book/toc.html*

[3] D. Parnas: *On the criteria to be used in decomposing systems into modules* (Communications of the ACM, 15(12):1053-8, dec. 1972), *http://www.acm.org/classics/may96/*

[4] KWIC Page C++: *http://www.infosys.tuwien.ac.at/Staff/gt/KWIC/*

[5] Dennis Ritchie: *Early Unix history and evolution*,
*http://cm.bell-labs.com/cm/cs/who/dmr/hist.html*
Klasszikus cikk a Unix első pár évéről.

[6] Andrew Leonard: *BSD Unix: Power to the people, from the code*, Salon Free Software Project,
*http://www.salon.com/tech/fsp/2000/05/16/chapter_2_part_one/*
Ígéretes böngészhető könyv rendkívül sok háttérinformációval a szabadszoftver-mozgalomról.

[7] Eric S. Raymond: *Unix conspiracy* in Jargon Files,
*http://www.tuxedo.org/ esr/jargon/html/entry/Unix-conspiracy.html*

[8] B. Kernighan, R. Pike: *A Unix operációs rendszer*, 5. burokprogramozás, Műszaki Könyvkiadó, Budapest, 1994
A Unix programozás témájában még mindig alapmű.

# Fürtözési megoldások – Linux-alapú cluster technológiák

Papp Dániel – Pintér Zoltán

2001. szeptember 3.

**Kivonat**

Az utóbbi időben Linux alatt is megjelentek olyan fürtözéses megoldások, melyek korábban csak nagygépes rendszerekben voltak elérhetők. Ez a fajta technológia az elmúlt években dinamikusan fejlődött.
Linux alatt többféle fürtözéses megoldásról beszélhetünk:

- Nagy teljesítményű tudományos-technikai fürtözés. Itt a cél az alkalmazás skálázhatóságának növelése.
- Hálózati terhelésmegosztó fürtözés. Ennek célja a nagyméterű hálózati terhelés megosztása.
- Magas rendelkezésreállást nyújtó fürtözés, üzleti alkalmazások részére. Célja az alkalmazás rendelkezésreállásának növelése.
- SSI (Single System Image) fürtözés. Ez a rendszererőforrások optimális kihasználását, skálázhatóságát célozza meg.

## Tartalomjegyzék

**1. Tudományos-technikai fürtözés (pl. Beowulf-cluster) 78**
1.1. Tulajdonságok . . . . . 78
1.2. Technikai jellemzők . . . . . 78

**2. Terhelésmegosztó fürtözés (pl. Linux Virtual Server) 78**
2.1. Tulajdonságok . . . . . 78
2.2. Technikai jellemzők . . . . . 78

**3. Nagy rendelkezésreállást (HA - High availability) biztosító fürtözés (pl. Kimberlite, Convolo) 79**
3.1. Tulajdonságok . . . . . 79
3.2. Technikai jellemzők . . . . . 79

**4. SSI (Single System Image) fürt (pl. Mosix) 79**
4.1. Tulajdonságok . . . . . 79
4.2. Technikai jellemzők . . . . . 79

**5. HA-fürtök fejlődési szakaszai 80**

**6. Összegzés 80**

**7. Hivatkozások 80**

# 1. Tudományos-technikai fürtözés (pl. Beowulf-cluster)

## 1.1. Tulajdonságok

- Jól párhuzamosítható alkalmazás futtatása több, vagy akár több száz szerveren
- A felhasználói programnak kell gondoskodnia a párhuzamosításról
- Elsődlegesen tudományos alkalmazások számára jelent megfelelő megoldást
- Üzleti alkalmazások számára általában nem megfelelő

## 1.2. Technikai jellemzők

- A Beowulf szoftver a párhuzamosan megírt alkalmazás számára biztosít osztott környezetet
- Az adatokat vagy az ütemező processz vagy egy NFS-szerver szolgáltatja
- Az alkalmazás vezérlésére általában külön munkaállomás szolgál
- Az egyes szerverek kiesése csökkenti a számítási kapacitást

# 2. Terhelésmegosztó fürtözés (pl. Linux Virtual Server)

## 2.1. Tulajdonságok

- Nagy-skálázhatóságú web szerverek építésére szolgál
- A bejövő hálózati kérések generálta terhelés megosztására szolgál egy web-farmon belül
- Linux Virtual Server (LVS): a legnépszerűbb, nyílt forráskódú ilyen jellegű implementáció
- Nagy teljesítmény, megfelelő skálázhatóság, magas rendelkezésreállás, kifejezetten web-szerverek részére

## 2.2. Technikai jellemzők

- A bejövő kéréseket a terhelésmegosztó fogadja, ez irányítja tovább a megfelelő web-szerver felé
- A válaszokat egyből az ügyfél kapja meg
- Skálázhatóság és nagy rendelkezésreállás
- Redundáns terhelésmegosztók
- Menet közben kivehető-berakható web-szerverek
- Általában nincs közösen használt diszk terület
- Viszonylag statikus web-tartalom esetén használható

# 3. Nagy rendelkezésreállást (HA - High availability) biztosító fürtözés (pl. Kimberlite, Convolo)

## 3.1. Tulajdonságok

- Cél: Több számítógép és periféria összekapcsolása olyan módon, hogy azok egyetlen rendszerként látsszanak és ez a rendszer működőképes maradjon akkor is, ha valamelyik komponens meghibásodik.
- NSPF - No Single Point of Failure
- Előtörténet: Korábban gyártófüggő megoldások születtek, az adott gyártó hardveréhez kapcsolódóan (pl. VMS, TruCluster, SGI Failsafe, IBM Phoenix, NT Wolfpack)
- Linux alapú megoldások: PC-alkatrészekből, jóval alacsonyabb áron megépíthetők

## 3.2. Technikai jellemzők

- Általában két szerver van összekötve
- Közösen használt SCSI vagy fibre diszk alrendszer
- Heartbeat csatornák, quorum eszköz (device)
- A másik HA-tag lekapcsolásának lehetősége
- Szolgáltatások (service) definiálásának lehetősége

# 4. SSI (Single System Image) fürt (pl. Mosix)

## 4.1. Tulajdonságok

- A nagy rendelkezésreállást és a skálázhatóságot célozza meg
- Optimális rendszer erőforrás kihasználás
- Ideális esetben transzparens az alkalmazások számára, nem igényel alkalmazás módosítást

## 4.2. Technikai jellemzők

- beépül a kernelbe
- egyetlen rendszernek látszik
- közös fájl és processztábla
- automatikus processz relokálás

# 5. HA-fürtök fejlődési szakaszai

**Átkapcsolásos fürtök (Failover Cluster)** Egy adott alkalmazás egyszerre csak egy tagon futhat. A fürt tagjai figyelik egymás állapotát és hiba esetén átveszik a másikon futó alkalmazásokat.

**Párhuzamos fürtök (Parallel Cluster)** Az alkalmazás párhuzamosan futhat mindegyik fürt tagon. Ehhez az alkalmazás módosítása szükséges.

**Egyetlen rendszernek látszó fürt (Single System Image)** Egyetlen rendszernek látszik (Egységesített PID- és fájlrendszer kezelés). Általában nem igényel alkalmazás módosítást.

# 6. Összegzés

A Linux-alapú cluster megoldások segítségével kisebb méretű cégek, alkalmazások is részesülhetnek a cluster technológia nyújtotta előnyökből. A Beowulffal szuperszámítógép teljesítményt hozatunk ki rendszerünkből, az LVS-sel web-farmokat építhetünk, a HA-clusterek révén pedig megfelelő megbízhatóságú platformot nyújthatunk akár nagyvállalati alkalmazások számára is.

# 7. Hivatkozások

*www.beowulf.org*
*www.linuxvirtualserver.org*
*www.linux-ha.org/LinuxFailSafe*
*www.missioncriticallinux.com*
*ultramonkey.sourceforge.net*
*www.steeleye.com*
*www.redhat.com*
*www.turbolinux.com*
*www.resonate.com*
*www.sistina.com*
*www.mosix.org*

# Digitális aláírással kapcsolatos eszközök és veszélyek

Pásztor Miklós
Pázmány Péter Katolikus Egyetem
pasztor@ppke.hu

2001. szeptember 3.

## Tartalomjegyzék

**1. Mi köze a GNU/Linux-nak a témához? 82**

**2. Hogyan is működik a digitális aláírás? 82**
2.1. A digitális aláírás és a kézirásos aláírás közti különbségek . . . . . 82
2.2. Elég-e a digitális aláírás és a titkosítás? . . . . . 83
2.3. A digitális kulcsok kezelésének problémái . . . . . 83

**3. Az X.509 alapú technológia 84**
3.1. Névtévesztés . . . . . 84
3.2. Mit garantálhat egy X.509 tanúsítvány egyáltalán? . . . . . 85

**4. A CA-k felelősségvállalása 85**

**5. Miért veszélyes ha 1 személy - 1 név - 1 kulcs? 86**
5.1. Magánszféránk fenyegetettsége . . . . . 86
5.2. Implementációs hibák . . . . . 87

**6. GNU/Linux eszközök 87**
6.1. PGP/GPG . . . . . 87
6.2. SSH/OpenSSH . . . . . 88
6.3. OpenSSL . . . . . 88
6.4. Újabb kezdeményezések . . . . . 88

**7. Összfoglalás 89**

# 1. Mi köze a GNU/Linux-nak a témához?

A múlt arra tanít, hogy éppen a biztonsággal kapcsolatos eszközökbe építenek be gyakran hátsó ajtót. Sok esetet lehet olvasni arról, hogy pl. egy kormány egy nagy tiszteletben álló másik államból titkosító eszközt vásárol, amiről utóbb kiderül, hogy az eladó vagy annak egy megbízottja számára kémkedik. Az ilyen esetek elkerülésének egyik módja a nyílt forráskódú programok, GNU/Linux eszközök használata. Persze ez annyit használ, mint egy zár amit az ajtónkra teszünk: bölcs dolog, hogy ott van, de önmagában nem garantálja, hogy nem jönnek be.
Egy másik kapcsolódási pont a GNU/Linux szellemiség: az a gyakorlat, hogy *gondolkodunk*, szeretünk a dolgok belsejébe látni, keressük a célszerű és egyszerű megoldásokat, nem vagyunk konzumidióták. A digitális aláírás lehetőségeinek felismeréséhez, veszélyeinek elkerüléséhez éppen ilyen tulajdonságok szükségesek.

# 2. Hogyan is működik a digitális aláírás?

A nyilvános kulcsú titkosítás elvén alapul. Hagyományos titkosítási eljárásnál egyetlen kulcsot kell ismernünk az üzenet kódolásához és dekódolásához. Nyilvános kulcsú titkosításnál minden egyes felhasználóhoz két kulcs tartozik: egy titkos, és egy nyilvános. A titkos és a nyilvános kulcs szerepe szimmetrikus. Ha *N* jelöli a nyilvános kulcs alkalmazását, *T* a titkos kulcsét, és *x* egy kódolandó információ, akkor

$$N(T(x))=x \text{ és } T(N(x))=x$$

Minden felhasználónak generálnia kell a maga részére egy nyilvános/titkos kulcs párt. Ezután a nyilvános kulcsot minél szélesebb körben ismertté kell tenni, a titkos kulcsra értelemszerűen vigyázni kell.

Bárki, aki titkosított üzenetet akar küldeni nem kell mást tegyen, mint a fogadó nyilvános kulcsával kódolnia kell az üzenetet. A nyilvános kulcs ismerete nem segít abban, hogy a titkos kulcsot megfejtsük, ezért ha egy üzenetet valaki nyilvános kulcsával kódoltunk, akkor már magunk sem tudjuk azt visszafejteni, csakis a fogadó.

Ha hitelesíteni akarunk egy üzenetet, akkor pedig a saját titkos kulcsunkat használjuk. Az üzenetből képezünk, egy az üzenetnél jóval rövidebb számot, amit az üzenet ellenőrző összegének, „ujjlenyomatának” is nevezhetünk. Ezt a számot kódoljuk azután a saját titkos kulcsunkkal. A fogadó ezt csakis a mi nyilvános kulcsunkkal tudja „kinyitni” és így biztos lehet abban, hogy az üzenetet valóban mi küldtük. Az üzenet ilyen esetben nincs feltétlenül kódolva, de mivel az egész üzenet ujjlenyomatát tartalmazza az aláírásunk, az üzeneten végrehajtott minden változtatás, egyetlen vesszőcske beszúrása vagy elhagyása is kiderül a fogadó oldalon. Ilyen módon - hasonlóan ahhoz mint amikor aláírunk valamit - a hitelesítéssel nem csak azt garantálhatjuk, hogy kitől származik az üzenet, hanem azt is, hogy az pontosan ugyan az. Ez alkalomadtán - akár csak az aláírás - arra is alkalmas, hogy valamit a fejünkre olvassanak, mint általunk elismert, vállalt dolgot.

## 2.1. A digitális aláírás és a kézirásos aláírás közti különbségek

Kézírással készült aláírása csak személynek lehet, és az többnyire egyfajta. Digitális aláírása (kulcspárja) viszont akárhány lehet minden embernek. Sőt nem csak emberek

hanem számítógépek, programok, szerverek és kliensek, hálózati interfészek vagy mobil telefonok is saját kulcspárral készíthetnek digitális aláírást. Valójában ez a dolog sokkal gyakoribb mint a személyek által készített digitális aláírás.

Egy személy által készített digitális aláírás jobban hasonlít egy pecséthez: egy eszköz kell hozzá, ami az aláíró tulajdonában van. Ahhoz, azonban hogy egy személy egy digitális aláírását ne lehessen hamisítani, a titkos kulcsát kell védeni úgy, mint egy pecsétet. Csakhogy ez sokkal nehezebb: ha egy pecsétet ellopnak, az nem marad ott. A titkos kulcs azonban elektronikus információ, másolható. Egy személy titkos kulcsa rendszerint valamilyen jelszóval védett. A jelszót illetéktelenek kileshetik, megtudhatják. A számítógéphez hozzáférhetnek stb. Súlyosbítja a helyzetet, hogy a legtöbben - még azok is akiktől ez elvárható lenne -, nagyon keveset értenek abból, hogy mi és hogyan történik.

## 2.2. Elég-e a digitális aláírás és a titkosítás?

A Nagy Vállalat digitális aláírásával elektronikus levél jön a titkosítva X-hez, X nyilvános kulcsa segítségével titkosítva, amely nagy összegű megrendelést tartalmaz. A kulcsok használatával semmi visszaélés nem történt. Biztos-e, hogy nincs turpisság?

*Nem!* Lehet, hogy a megrendelés nem is X-nek, hanem Y-nak szólt, X eredetileg Y nyilvános kulcsával titkosította, de pl. Y egy alkalmazottja lefülelte az üzenetet, mely így szólt:

> „Március 20-i árajánlatuknak megfelelően megrendelünk Önöktől 1000 darab lekvárzsibbasztó gépet."

Az imposztor aztán X nyilvános kulcsával újra titkosította az üzenetet, melyen ott volt a Nagy Vállalat hiteles digitális aláírása, és így továbbította X-nek.

Ez a visszaélés az „alattomos továbbítás". A 2001-es Usenix konferencián Don Davis egy egész előadást szentelt ennek a témának. Persze az alattomos továbbítás csak akkor működik, ha az üzenet szövegéből nem derül ki, hogy kinek szól. De ha megszólítás van az üzenetben, az még nem garancia: ha Iluska azt írja egy aláírt titkosított levélben, hogy „Jancsi szeretlek", akkor Jancsi pl. viccből alattomos továbbküldéssel becsaphatja ezzel a levéllel egy névrokonát.

## 2.3. A digitális kulcsok kezelésének problémái

Egy nyilvános kulcsú titkosítás elvén készült digitális aláírás azt garantálja, hogy egy üzenet olyan forrásból származik, ahol a kulcspár titkos része az üzenet keletkezésekor rendelkezésre állt. Ha a nyilvános és titkos kulcsok kezelése megfelelő, akkor ez jó okot ad arra, hogy feltételezzük: az üzenet egy bizonyos személytől, eszköztől, bizonyos tisztség viselőjétől, vagy bizonyos dologra felhatalmazott egyéntől származik. Jó okunk van azonban kételkedni, ha a nyilvános és titkos kulcs keletkezése, tárolása, továbbítása, használata nem megfelelő. A legnagyobb probléma az, hogy a nem megfelelő kezelést könnyű látni, nehéz azonban a megfelelőt elérni, arra kritériumokat felállítani, szabványokat definiálni.

A digitális aláírásokról szóló cikkeket, és különösen a hitelesítésszolgáltatók (Certification Authorityk) reklámait olvasva az a benyomásunk támadhat, mintha ezen szolgáltatásának magas színvonala, az ott bevezetett szigorú ügymenet és kiváló technika garantálná a digitális aláírás biztonságát. Nem lehet eléggé hangsúlyozni, hogy ez súlyos tévedés: a digitális aláírás biztonsága egy lánc, és olyan erős, mint a leggyengébb

láncszeme. Lehet bármilyen kitűnő a tanúsítvány, bármilyen brilliáns a matematikai eljárás, bármilyen sok bites a kulcs amit használunk, ha a digitális aláíráshoz tartozó titkos kulcs illetéktelenek használhatják, mit sem ér. A illetéktelen felhasználó nem feltétlenül ember, *lehet progam is*. Éppen ezért nem szabad fontos digitális aláírást készíteni, vagy ellenőrizni olyan gépen, mely hálózatba van kötve, vagy pl. Word is van rajta telepítve. Egy vírus, vagy hálózati betörés veszélyt jelent nem csak az aláíró, hanem *az aláírást ellenőrző gépen is.* Vélhető, hogy az ilyen vírus, vagy behatoló nem fedi fel magát olyan gyorsan, mint pl. egy web szerver esetén: lehetséges, hogy akár hónapokig lappang. Egyébként is várható, hogy csak késlekedve jelentkezik a hatása: megjelenik valahol egy irat, amit a mi nevünkben írtak alá és kicsit más mint amit mi aláírni véltünk, más összegű számlát fizettünk ki, vagy nyújtottunk be, vagy valamilyen erőforrást a nevünkben használtak. Illetve valakiről tévesen azt hisszük, hogy valamit aláírt, valamilyen számlát kifizetett, vagy benyújtott stb.

# 3. Az X.509 alapú technológia

Amikor digitális aláírásról beszélünk, a legtöbben az X.509 szabványú tanúsítvánnyal (certificate) ellátott kulcsokkal történő aláírásra gondolnak. Az X.500 a 80-as évekből származó CCITT szabvány, mely globális névtár létrehozását célozza. Ennek egy folyománya az X.509, melyben nevekhez (DN, Distinguished Name) nyilvános kulcsokat rendelünk. Az X.509 koncepció egyik fő eleme a hitelesítő szervezet (CA, Certification Authority), amely tanúsítja, hogy egy bizonyos nyilvános kulcs egy bizonyos névhez tartozik. A hitelesítő szervezetek (CA-k) hierarchiát alkothatnak. X.509 szerint működnek pl. a TLS (SSL) web szerverek. Egy-egy web szerverhez tartozó kulcsot lehet hitelesíteni valamely hitelesítő szervezetnél, de nem szükséges. Az X.509 elképzelésekben szerepel egy legfelső szintű (root) CA, amely más hitelesítőket hitelesít, akik újabbakat egészen addig, amíg az éppen használt tanúsítványhoz nem jutunk.

## 3.1. Névtévesztés

Az X.509 alapú digitális aláírások egyik pillére a név. Minden X.509 tanúsítvány egy név-kulcs megfelelést tanúsít. A gyakorlat azt mutatja, hogy a nevek használata sok zavart okoz, sem nem szükséges, sem nem elégséges egy-egy alkalommal. Ha valaki az adóbevallását akarja digitálisan benyújtani, akkor nem elég tudnunk, hogy Kovács János, aki Jászberényben a Thököly utcában lakik: nincs kizárva, hogy két Kovács János is lakik ugyanabban a házban, pl. apa és fia. Még az sem elég, ha azt tudjuk, hogy az illető a Jászberényi Hűtőgépgyárban dolgozik: lehet, hogy mindketten ott dolgoznak. Az sincs kizárva, hogy éppen egy családban nagy a kísértés, hogy az egyik Kovács a másik kulcsával visszaéljen... Nem egyszer előfordul, hogy pl. több Bornemissza van egy vállalatnál, és a bornemissza@valahol alakú címre küldött levelek rendszeresen eltévednek. Ha egy magánlevelet így rosszul címzünk, általában kisebb bosszankodással megússzuk. Ha azonban digitális aláírásban tévedünk, akkor nagy kár érhet.

### Kell-e valóban név?

Láttuk, hogy a név alapján megkülönböztetni kulcsokat (tanúsítványokat) nem elégséges. Nem is szükséges. Számos példa van rá a hagyományos világban, hogy elfogadunk egy aláírást, de valójában nem érdekel, hogy ki is írta alá, nem is fontos a neve. Egy érettségi bizonyítványon például ott van az érettségi elnök, az iskola igazgatójának

az aláírása, de a legtöbbször nem is tudjuk elolvasni. Vitás esetben az iskola tudja csak ellenőrizni, hogy abban az évben kik is voltak az illetékesek, és az aláírás valódi-e. Hasonló a helyzet pl. a patikában, ahol a orvos által aláírt receptet mutatunk be: fontos az aláírás, de az orvos neve nem. Lehetséges, hogy a fenti két Kovács János, apa és fia Jászberényben mind a ketten orvosok, és ha el is tudjuk a recepten olvasni az aláírásukat, maga a név nem szolgál valódi információval. Ezért van, hogy a recepteket az aláírás önmagában még nem hitelesíti.

### 3.2. Mit garantálhat egy X.509 tanúsítvány egyáltalán?

Egy tanúsítvány csak azt garantálhatja, hogy - a fenti példánál maradva -, egy bizonyos kulcsnak köze volt egy bizonyos pillanatban az egyik Kovács Jánoshoz. Azt hogy a kulcs titkos párja valaha is János birtokában volt, csak akkor valószínűsíthető, ha a kiállító meggyőződött róla, hogy alá tud írni valamit a megfelelő kulccsal. Egyes tanúsítvány kibocsátók folytatnak is ilyen gyakorlatot: a kulcs tulajdonosát felszólítják, hogy - akár a tanúsító előtt -, írjanak alá valamit. Hogy a kulcs milyen eszközök, vagy személyek birtokában van, volt vagy lesz még, hogy János egy későbbi aláíráskor pl. nem öntudatlanul használta a kulcsot arra ilyenkor sincs biztosíték.

Egy tanúsítvány nem feltétlenül garantálja, hogy valaki egy intézményt képvisel, még akkor sem, ha tanúsítvány kibocsátója ezt is tanúsítani akarja. 2001 januárjában egy nagy szoftvergyártó vállalat munkatársaként kapott valaki egy tanúsítványt az egyik legismertebb CA-tól, amit arra használhatott fel, hogy hibás, vírusos programokat terjesszen mint a cég „javításait”. Lásd:

http://www.teleconnect.com/article/IWK20010322S0007, vagy

http://www.microsoft.com/technet/security/bulletin/MS01-017.asp

Ez az eset a CA filozófia alapelvét érinti: egész egyszerűen azt bizonyítja, hogy *nem szabad megbíznunk a tanúsítványban.* A valós életben az emberek azonosságának megállapításánál egész más módszereket használunk. Ha arról akarunk meggyőződni, hogy pl. Kovács János valóban az, aki általános iskolában mellettünk ült, egy X.509 tanúsítvánnyal semmire sem megyünk, de egy személyes beszélgetés pillanatok alatt eredményre vezethet. Tanúsítvány helyett is megfelelőbb lehet egy telefonbeszélgetés, egy névjegy amin rajta van a nyilvános kulcs ujjlenyomata, vagy akár egy könyv vagy újság ami az ujjlenyomatot közli.

## 4. A CA-k felelősségvállalása

A tanúsítvány hitelesítő szervezetek (CA-k) tisztában vannak a korlátaikkal. Ezért reklámaikban és ügymenet leírásukban (CPS, Certification Practices Statement) azt hangsúlyozzák, hogy milyen szigorúan őrzött helyiségekben dolgoznak, milyen szigorú az adminisztráció, stb. Óvakodnak azonban a digitális aláírásokért felelősséget vállalni, különösen anyagi felelősséget.

### Félrevezető szavak

Ahogy fentebb már szó volt róla, valójában a digitális aláírás szó is félrevezető: talán helyesebb lenne digitális pecsétet mondani. De nem csak ez a szóhasználat kelt zavart, túlzott várakozást. A „tanúsítvány”, „tanúsítvány kibocsátó” szó azt sugallja, mintha

egy digitálisan aláírt dokumentumról a kibocsátó tanúsítaná, hogy hiteles, valóban az írta alá, aki/ami szerepel az aláírásban. Akik értik, akik végiggondolták, hogy erről szó sincs, azok is könnyen megfeledkezhetnek róla a szóhasználat miatt. Sajnálatos, hogy a törvény is „minősített digitális aláírásról" beszél, holott nyilvánvaló, hogy nem az aláírás minősített, hanem a tanúsítványkibocsátó, aki a digitális aláírásnál használt kulcsot aláírta.

### A központosítás hátrányai

Az X.509 tanúsítványok világában a hitelesítők (CA-k) láncolatot alkotnak. Egy tanúsítvány kibocsátójának tanúsítványát egy másik, magasabb szinten levő hitelesítő írja alá, akinek tanúsítványát egy másik és így tovább, egy önmagát hitelesítő legfelsőbb szintű hitelesítő központig. A feltételezés az, hogy ez a legfelső root CA egy mindenki által támogatott és elfogadott intézmény. Az élet azonban számtalanszor bizonyította, hogy az ilyen szerkezet igen sérülékeny: egyetlen emberi hiba, netán visszaélés, egyetlen ügyintézési figyelmetlenség, vagy programhiba óriási következményekkel járhat. Egy központosított struktúra hátránya, hogy a hitelesítők visszaélhetnek lehetőségeikkel: nem csak a szolgáltatás árát srófolhatják magasra, *visszaélhetnek* pl. azzal, amit egy-egy *kulcs használatával kapcsolatos információk jelenthetnek.* A visszavont tanúsítványok listájában (Revocation List) való keresgélés naplófájlja információt szolgáltathat például egy cég, vagy egy személy vásárlási szokásairól.

A központosított szervezés hátránya, hogy a hierarchia magas fokán álló hitelesítők titkos kulcsa különösen csábító célpont lehet a bűnözők számára. Ez abba az irányba hat, hogy az ilyen kulcsot gyakran váltsák. A központi szerep miatt azonban a kulcs váltása igen nehézkes: nem csak a „gyerek" kulcsokat, hanem esetenként programokat, program konfigurációkat is érint.

Egy központosított modell csak akkor működhet jól, ha tökéletesen megbízhatunk a központban és nem csak a szándékait, hanem a *szervezettségét, ügyintézését*, technikai felszerelését illetően is. Az életben az *osztott, több lábon álló*, változatos rendszerek bizonyulnak jobbnak és hatékonyabbnak.

## 5. Miért veszélyes ha 1 személy - 1 név - 1 kulcs?

Az X.509 elképzelésben minden személyhez (szervezethez, számítógéphez, stb.) tartozik egy egyedi név, és ehhez egy-egy kulcs. Ha ezt a gondolatot összevetjük az Alkotmánybíróság személyi számmal kapcsolatos döntésével, azonnal láthatjuk, hogy nem elfogadható személyiségvédelmi, jogi szempontból sem, hogy ezt a nevet, illetve kulcsot kelljen használni minden egyes elektronikus aláíráskor.

A valós életben is többféle azonosítónk, igazolványunk van. Megszoktuk, hogy különböző társaságokban más és más nevünk is lehet. Ha digitális azonosítót kívánunk használni, annak más súlya lesz az útlevelünknél, és a bélyeggyűjtő szakkör tagsági igazolványnál. Ezért természetes igény, hogy különböző célokra és különböző neveken lehessen digitális aláírásra szolgáló kulcsunk, tanúsítványunk.

### 5.1. Magánszféránk fenyegetettsége

Az a gyakorlat, ami a digitális aláírás használatának infrastruktúrájára kialakulni látszik a már említetteken kívül is több veszélyt jelent magánszféránkra. És nem csak

az Orwell regényéből ismert rém fenyeget (kulcsunk használatának nyomonkövetése, adatgyűjtés).

Nem egy hitelesítő hatóság maga generálja és osztja is a kulcsot. Ez azt jelenti, hogy eleve a kibocsátó birtokában van a titkos kulcs, eleve lehetősége van arra, hogy megszemélyesítse a kulcs tulajdonost. Egyes esetekben (pl. egy-egy munkahelyen) ez megengedhető lehet, a legtöbb esetben azonban elfogadhatatlan.

A magyarországi X.509 PKI-vel kapcsolatos tervekben felmerül, hogy az összes hitelesítő szervezet, azok minden telephelye használjon egy közös tanúsítvány adatbázist. Egy ilyen adatbázis - különösen más adatokkal kombinálva - óriási információszerző forrás lehet magánszemélyekről, intézményekről, azok szokásairól stb. Már ma is sok intézmény használ hasonlót - pl. internetről gyűjtött adatokat - elsősorban marketing céllal. Az ilyen cél is kifogásolható, de semmi biztosíték sincs arra, hogy a felhasználás csak marketing célokra korlátozódik.

Van olyan törekvés is, hogy a generált kulcs titkos részét valamilyen biztos helyen, letétben kell tartani (key escrow). Ez szintén indokolt lehet egy-egy munkahelyen, a legtöbb esetben azonban ez is elfogadhatatlan veszéllyel jár. Érdemes különbséget tenni a titkosításra és a digitális aláírásra szolgáló kulcsok közt. A munkehelyen érdemes a tisztséghez tartozó kulcsokat használni titkosításra, és a személyhez tartozó kulcsokat digitális aláírásra. A tisztséghez tartozó kulcsokat van értelme letétbe helyezni, több ember számára hozzáférhetővé tenni, azok titkos részéről is biztonsági másolatot fenntartani, a személyhez tartozó, digitális aláírásra szolgáló kulcs titkos része viszont maradhat teljesen a magánember használatában és tulajdonában.

Illetéktelen használat nem csak úgy fordulhat elő, hogy valaki rossz szándékkal birtokolja kulcsunkat, vagy pl. a hitelesítő hatóságnál valaki visszaél a lehetőségeivel. Gondolnunk kell arra, hogy hiába bízunk meg például a főnökünkben, annak is lehet utódja, barátja, a barátjának is barátja stb. Ha valaki kérné a titkos kulcsunkat, ajánlatos úgy viselkednünk, mint az öreg miniszterelnök, akiről az ismert anekdota szól. Az újságírók hiába ácsorogtak a kapu előtt, ahol a kormány zárt tárgyalást folytatott, semmit sem tudtak meg. Az utoljára távozó miniszterelnököt megállítja két kitartó fiatal újságíró. A miniszterelnök kérdésére - „Uraim, tudnak Önök titkot tartani?” - a két fiatalember buzgón válaszolja: - „Hát hogyne!” Az öreg válasza: - „Én is.”

### 5.2. Implementációs hibák

Amint látjuk az X.509 szerint használt digitális aláírással sok elvi baj is van. Még több azonban az implementációkkal. Az egyik legsúlyosabb, hogy nem kezelik a tanúsítvány visszavonást, sem a statikus CRL kezelést, sem a dinamikus OCSP, RFC2560 szerintit. Éppen ezért a fent említett szoftver gyártóhoz tartozó tanúsítványt hiába vonta vissza a hitelesítő, a vírusos szoftverek úgy terjedhetnek, hogy a kliensek hitelesített tanúsítványúnak látják.

## 6. GNU/Linux eszközök

Szabadon terjeszthető digitális aláíró eszközök már régen használatban vannak.

### 6.1. PGP/GPG

Éppen 10 éve annak, hogy az első forráskódban is elérhető nyilvános kulcsú titkosítást és digitális aláírást lehetővé tevő eszköz a PGP (Pretty Good Privacy) kezdett elterjedni.

Elsősorban személyes dokumentumok, levelek digitális aláírására szánta alkotója, Phil Zimmermann. A PGP aláírás sok éve rutinszerűen használatos éles területeken: pl. GNU szoftverek hitelesítésére és a CERT-ek (Cumputer Emergency Response Team) kommunikációjában.

A PGP formátumot azóta internet RFC (RFC2440) szabványosította, és több implementáció született. A legjelentősebb és legteljesebb ezek közül az GPG, a GNU Privacy Guard. Nem csak kulcsokat, hanem kulcsok hierarchiáját kezelhetjük segítségével. Az alkulcsokat azután különböző célokra használhatjuk. A klasszikus pgp-ben az egyik probléma az volt, hogy ha valaki elvesztette a titkos kulcsát vagy elfelejtette a jelmondatot, akkor nem volt mód arra, hogy a kulcsát visszavonja. Az OpenPGP és a GPG módot ad arra, hogy kijelöljünk olyan személyt pontosabban kulcsot ami ilyen esetben segít.

PGP/GPG kulcsot bárki generálhat, használhat. Egy nyilvános kulcson akárhány aláírás lehet. A kölcsönösen aláírt kulcsok hálót, a PGP *bizalmi hálót* alkotják. Így egy nyilvános kulcs hitelességét akár több tucat aláírás tanúsíthatja, és minden felhasználó maga döntheti el, hogy milyen feltételekkel fogad el hitelesnek egy addig nem látott nyilvános kulcsot.

## 6.2. SSH/OpenSSH

Az SSH elsősorban távoli számítógépekbe való bejelentkezésre szolgáló eszköz. Nyilvános/titkos kulcspár tartozik felhasználókhoz, és számítógépekhez is. SSH-val a szerver és kliens számítógépeket is digitális aláírás segítségével azonosítjuk. Ez módot ad arra, hogy finoman (és pl. az IP címnél biztonságosabban) szabályozzuk, hogy mely gépekről szabad bizonyos tevékenységeket végezni, és hogy megakadályozzuk, hogy egy imposztor lecserélje a szervert amin dolgozunk, vagy lehallgatás céljából közbeiktassa a sajátját. Nagyszerű eszköz az ssh-agent: lehetővé teszi, hogy egyszer azonosítsuk magunkat pl. reggel a saját pc-nken, és aztán a többi bejelentkezést mind ő végzi el úgy, hogy a szerverektől kapott kihívást (challange) a mi digitális aláírásunkkal küldi vissza. Ezt a módot nevezik SSO-nak (Single Sign On). Mindez évek óta rutinszerűen használt dolog GNU/Linux platformon.

## 6.3. OpenSSL

Az SSL (Secure Socket Layer) a TCP kapcsolatok titkosítására és digitális aláírásra szolgáló protokoll, a Netscape fejlesztése. Internet RFC is vált belőle (RFC2246), ahol TLS-nek (Transport Layer Security) nevezik. Elsősorban arra használatos, hogy web szerverek azonosítsák magukat a böngészőknél X.509 tanúsítvánnyal, és a forgalmat a böngésző és a szerver közt titkosítva bonyolítsák. Az OpenSSL alkalmas arra, hogy SSL-t használjunk bármilyen programban, például a nyílt forráskódú web szerverekben. Az OpenSSL csomag segítségével tanúsítványokat is készíthetünk: használhatjuk az X.509 technológiát k*ülső tanúsító hatóság (CA) nélkül*. Amint arról fentebb már szó volt egy CA tanúsítványa nem teszi biztonságosabbá a web szervert, és a kliensek sem feltétlenül nyernek vele, ha a szerverünk nyilvános kulcsát CA-val aláiratjuk.

## 6.4. Újabb kezdeményezések

A világban számos kezdeményezés folyik, mely túl akar lépni az X.509 korlátain, hátrányain, elsősorban a kulcs kezelés területén. Az SPKI/SDSI (Simple Public Key

Infrastructure/Simple Distributed Security Infrastructure, RFC2693), vagy a KeyNote Trust Management (RFC2704).

Az SPKI nem csak egyszerű, hanem általános is: speciális esetként tartalmazza az X.509, PGP, SSH szerinti kulcs kezelést is. Mód van arra, hogy egy SPKI kulcs által kapott jogokat, azoknak egy részét tovább delegáljuk egy másik kulcs tulajdonosnak. A delegálás finoman szabályozható: pl. megmonthatjuk, hogy meddig érvényes, vagy hogy az újabb továbbdelegálásra is felhatalmazást adunk-e.

# 7. Összfoglalás

Az X.509-en alapuló PKI egy-egy jól körülhatárolt - akár széles - körben megfelelő lehet (pl. katonaság, hivatalok) de a polgári életben, és az interneten való általános használata több mint problematikus. Szerencsére több alternatíva létezik: ezek egy része már régen bevált, évek óta használatos digitális aláírásra és titkosításra (PGP, SSH). Néhány újabb pedig éppen a kulcs kezelés területén jelent előrelépést (SPKI/SDSI). A digitális aláírásról szóló törvényt mostanában fogadta el az országgyűlés. Ezért igen aktuális ma ez a téma. Beszélni, gondolkodni róla, értve és figyelmesen használni, illetve készülni a használatára különösen fontos: nem csak az a veszély fenyeget, hogy hitelesnek hiszünk valamit, ami nem az, hanem az is, hogy személyek és intézmények nagy összegű beruházásokkal vélnek olyasmit elérni, amihez inkább józan ész, figyelem, tanulás és hozzáértés kell.

A magyarországi törvény - más országok törvényeihez hasonlóan -, nem kötelezi el magát semmilyen technológia, vagy infrastruktúra mellett, bár több ponton egyértelmű utalás van az X.509 PKI-ra. Mindazonáltal szabatosan, és általánosan a fogalmaz, és a törvény szerint jogilag hatályosak lesznek más digitális aláírások is. Nyilvánvaló, hogy sem technikailag, sem jogilag nem zárult le a digitális aláírás alkalmazásának folyamata. A GNU/Linux eszközök és különösen az a szellem amit ezek képviselnek nyilvánvalóan sokat lendítenek azon, hogy technikailag jobb, anyagilag kedvezőbb, kevésbé veszélyes és nem utolsó sorban emberségesebb legyen az, ami kialakul.

## A témához kapcsolódó néhány hálózati hely

Ellison C. & Schneier B.: Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure, Computer Security Journal, v 16, n 1, 2000
*http://www.counterpane.com/pki-risks.html*

The KeyNote Trust-Management System Version 2 RFC2704
*ftp://ftp.sztaki.hu/pub/nic/rfc/rfc2704.txt.Z*

Ellison C.: Naming and Certificates, Proc. Computers, Freedom & Privacy 2000, at *http://www.cfp2000.org/papers/ellison.pdf*

Simple Public Key Infrastructure (SPKI), RFC2693
*ftp://ftp.sztaki.hu/pub/nic/rfc/rfc2693.txt.Z*

X.509
Internet Public Key Infrastructure: Online Certificate Status Protocol - OCSP, RFC2560 *ftp://ftp.sztaki.hu/pub/nic/rfc/rfc2560.txt.Z*

Rivest R.L. & Lampson B. (1996): SDSI – A Simple Distributed Security Infrastructure, 15 Sep 1996, *http://theory.lcs.mit.edu/~rivest/sdsi10.html*

OpenPGP Message Format, RFC2440
*ftp://ftp.sztaki.hu/pub/nic/rfc/rfc2440.txt.Z*

PGP kulcs szerverek:
*http://www.rubin.ch/pgp/keyserver.en.html*

SPKI-hoz kapcsolódó web lap:
*http://world.std.com/~cme/html/spki.html*

Alattomos továbbításról szóló cikk:
*http://world.std.com/~dtd/sign_encrypt/sign_encrypt7.html*

Az X.509 modell részletes kritikája:
*http://www.anu.edu.au/people/Roger.Clarke/II/PKIMisFit.html*

# Hálózati határvédelem eszközei

Scheidler Balázs <bazsi@balabit.hu>

2001.08.14.

**Kivonat**

Napjainkban sajnos egyre kevésbé „biztonságos” hely a Net. Néhány évvel ezelőtt az Internet még félig-meddig bizalmi alapon működött, ma már mindenki gyanakvással figyel a „külvilágba”. Gyakran hallani tizenévesek „akcióiról”, mely során nagynevű cégek hálózataiba hatolnak be látszólag különösebb probléma nélkül. E csínyek során súlyos milliók kerülhetnek és kerülnek veszélybe.
Az írott és íratlan sajtó persze igyekszik nagy lufit varázsolni mindenből, a sikeres behatolások oka azonban leginkább a nem létező vagy elégtelen, rosszul kivitelezett vagy rosszul üzemeltetett hálózati határvédelem.
Előadásomban igyekszem bemutatni a rendelkezésre álló védelmi technológiákat, és azt hogy az általunk fejlesztett Zorp tűzfalrendszer hogyan illeszkedik ebbe a képbe.

## Tartalomjegyzék

**1. Bevezetés** **92**

**2. Bastion hoszt** **92**

**3. Csomagszűrő router** **92**
3.1. Egyszerű router . . . . . 93
3.2. Router és a NAT . . . . . 93
3.3. Csomagszűrés . . . . . 93

**4. Stateful Packet Filter - Állapottartó csomagszűrő** **94**

**5. Alkalmazásszintű proxy tűzfalak** **94**

**6. Zorp** **95**

**7. Zorp a jövőben** **95**

# 1. Bevezetés

Napjainkban egyre természetesebb, hogy egy cég internet-kapcsolattal rendelkezik, és valamilyen módon felhasználja az Internet szolgáltatásait, ilyen módon beolvasztva azt saját folyamataiba.

Így azonban saját informatikai infrastruktúrája kapcsolatba kerül az Internettel, mely kapcsolatot a cég (és az internet) védelme érdekében szabályozni kell. A hálózati határvédelem feladata, hogy különböző védelmi szintű zónákat elválasszon egymástól olymódon, hogy köztük a kommunikáció áramlása szabályozott körülmények között, de biztosított legyen.

A határvédelem legfontosabb feladatait így összegezhetjük:

- a helyi Informatikai Biztonsági Szabályzatnak megfelelő szabályrendszer betartatása.
- információáramlás biztosítása, a bájtsorozat folyamatos értelmezésével és ellenőrzésével.
- ismeretlen és tiltott elemek áthaladásának megakadályozása, ilyen módon védve a két zónát egymástól.

Nézzük, hogy ezeket a feladatokat hogy oldották meg az idők folyamán.

# 2. Bastion hoszt

Bastion hoszt neve magyarra nagyon nehezen fordítható le úgy, hogy tükrözze az eredeti jelentést. A bastion angolul megerősített területet jelent, a bastion host kifejezés pedig egy olyan megerősített számítógép, mely egy védett belső hálózat számára a hálózati erőforrások igénybevételét lehetővé teszi.

A bastion hoszt egy szabványos munkaállomás, két hálózati interfésszel, melyek közül az egyik a belső-, a másik pedig a külső hálózat felé figyel (a külső és belső elnevezés kizárólag a szemléltetést szolgálja, egyszerűen két zónáról van szó).

A felhasználók a saját munkaállomásaikról úgy tudják igénybevenni a hálózat szolgáltatásait, hogy bejelentkeznek a bastion hosztra (akár az X Window akár terminál emulátor használatával), majd onnan indítják a hálózati kliensprogramot (pl: böngészőt).

A bastion hoszt használata egyrészt kényelmetlen, mert az adatok nem közvetlenül a mi gépünkre érkeznek, másrészt nem is nyújt elegendő biztonságot, mivel a munkaállomásokra írt szoftvereknél nem ez az elsődleges szempont.

Természetesen proaktív konfigurációval a veszély csökkenthető, de nem szüntethető meg (pl: ha lehet, akkor szolgáltatások csak a belső interfészen figyelnek, külsőn nem).

Bastion hoszt alkalmazásakor a belső hálózaton lévő számítógépek IP címeinek kívülről nem kell címezhetőnek lennie, hiszen minden kérés a külvilág felé a bastion hoszt külső IP-jéről indul.

# 3. Csomagszűrő router

Nézzük meg, hogy egy csomagszűrő nélküli router milyen módon valósítja meg két alhálózat összekapcsolását.

## 3.1. Egyszerű router

A routernek, hasonlóan a bastion hoszthoz, legalább két hálózati interfésze van, két különböző kívülről címezhető címtartományban. A belső gépek a külső hálózatnak szóló csomagokat a routerhez továbbítják, ami - saját routing táblázata alapján - gondoskodik a csomag további útjáról. A csomag forráscíme - alapesetben - továbbításkor nem változik.

A válaszcsomagok célcíme a feladó gép IP címét tartalmazza. Hasonlóan az előző esethez, a válaszcsomagok is megérkeznek a routerhez, ami a belső hálózati interfészén keresztül továbbítja azt megfelelő belső számítógéphez.

A router a csomagokat IP-szinten továbbítja, azaz nem foglalkozik az IP feletti TCP, vagy UDP protokollok fejlécével, azokat kizárólag a kommunikáló felek (a belső, és a külső számítógép) dolgozzák fel. A védendő számítógép teljesen kiszolgáltatott helyzetben van, hiszen a teljes hálózati protokoll stackje elérhető, és akár az operációs rendszerben, akár az azon futó alkalmazásban lehet olyan hiba, melyet a külső támadó kihasználhat.

Ez a felállás semmiféle védelmet nem nyújt, hiszen a két fél közvetlen kapcsolatban áll egymással, a router nem dönt csomagok továbbításáról vagy eldobásáról.

## 3.2. Router és a NAT

A védekezés hatékonyabbá tételében logikus lépés az, hogy a belső számítógépeknek kívülről nem címezhető, hanem belső címeket osztunk ki. Így a belső gépek kívülről nem érhetőek el közvetlenül.

Belülről kezdeményezett kapcsolatok esetén a router a kimenő csomagok forráscímét és portját átírja, a válaszokat pedig a belső hálózatra továbbítja. Ezt a technikát hívják Network Address Translation-nek vagy NAT-nak. A NAT egy speciális esete, amikor 1:N leképezést végzünk, ezt hívják masquerading-nek is.

Fontos megjegyeznünk azt, hogy a csomagok a routeren mindkét irányba - a cím átírásán kívül - változatlanul továbbítódnak, azaz az eredeti IP opciók, és fragmentek változatlanul „átmennek”.

## 3.3. Csomagszűrés

Mint ahogy a nevében is benne van, a csomagszűrés lehetővé teszi bizonyos csomagok eldobását ill. átengedését a csomag fejlécében lévő információk alapján.

Ez általában egy szabályhalmazt jelent, ahol minden szabály egy feltételt, valamint egy akciót jelöl. Ha a beérkező csomag megfelel a feltételnek, azon alkalmazni kell a megadott akciót, mely lehet például a csomag átengedése, eldobása vagy visszautasítása. A feltételben kiköthetjük a csomag forrás valamint cél IP címét, az IP feletti protokoll azonosítóját, valamint TCP és UDP protokollok esetén a portszámot.

Csomagszűrő router alkalmazása esetén tehát lehetőségünk van bizonyos csomagokat átengedni vagy eldobni az alkalmazott címtartományok, valamint portszámok alapján, viszont ha egy csomag engedélyezett, akkor címátíráson kívűl a csomag változatlanul halad át.

Számos operációs rendszerben fordult elő olyan hiba, mely az IP fejlécek rossz értelmezéséből adódott, a csomagszűrő router már ezekkel szemben is tehetetlen, nem beszélve a csomagok adatrészében átmenő adatokról.

További probléma a kizárólag portszám alapján történő szűrésben az, hogy nem lehet ellenőrizni, hogy egy adott porton tényleg az adott felsőbb szintű protokoll halad át (pl. 80-as port esetén HTTP).

# 4. Stateful Packet Filter - Állapottartó csomagszűrő

A stateful packet filter a csomagszűrő router gondolatát fejleszti tovább, annyiban, hogy képes két áthaladt csomag között állapotot tartani, és összefüggéseket figyelni. Az SPF tűzfalak ún. connection tracking technikát használnak, melynek segítségével képesek a csomagokat kapcsolathoz rendelni, így a kapcsolaton kívüli csomagok automatikusan eldobhatók.

Az SPF már nem csak a csomag fejléce-, hanem tartalma alapján is hoz döntéseket, azaz az egyes protokoll-elemeket is feldolgozza. SMTP protokoll esetén például észreveheti, ha a protokollnak nem megfelelő kérés megy át a csatornán, és ennek kapcsán képes lépéseket tenni (pl. megszakítani a kapcsolatot - ilyen például a Cisco PIX tűzfala).

Az SPF-ek legnagyobb problémája az, hogy felépítéséből következően az adatcsatornákat csomagok sorozataként kezeli, nem pedig bájtfolyamként, ezért:

- a csomagok fejlécei módosítás nélkül kerülhetnek a klienshez, ami problémát okozhat az IP stacknek.
- amennyiben a tűzfal a másképp értelmezi a fragmentált IP csomagokat, úgy elképzelhető, hogy mást "lát" a tűzfal, és mást a kliens.
- nehéz az adatfolyamban olyan változtatásokat eszközölni, melyek átlépik a csomaghatárt (például tartalomszűrés, újrakódolás vagy kódolás levétele, stb).

További probléma jelentkezik, ha a feldolgozandó protokoll bonyolult. Ez könnyen állapottér-robbanáshoz vezethet, ami átláthatatlanná teheti a protokoll ellenőrzését.

Összességében: az állapottartó csomagszűrő valamivel nagyobb biztonságot képes nyújtani egy egyszerű csomagszűrőnél.

# 5. Alkalmazásszintű proxy tűzfalak

A proxy tűzfalak felépítésükben közelebb állnak a bastion hosztokhoz, mint a stateful packet filterekhez, ugyanis a proxy tűzfal alapvetően nem csomagok továbbításával foglalkozik. A proxy tűzfal egy dedikált számítógép, speciálisan előkészített, de szabványos operációs rendszerrel (BSD, Linux) és a szolgáltatások közvetítését végző proxy szoftverrel.

A tűzfal szoftver minden támogatott protokollra tartalmaz egy ún. proxy programot, mely a belső hálózati csatolón hallgatózik, kapcsolat érkezése esetén pedig kapcsolatot kezdeményez a külső oldalon, majd a kéréseket-válaszokat ellenőrzés után közvetíti. Fontos megjegyeznünk, hogy két teljesen független kapcsolatról van szó, a két zóna között csak az adatrész mozog, az IP/TCP fejléc teljesen újraépül.

Láthatjuk, hogy a legtöbb lehetősége a proxy tűzfalaknak van: ha akarják képesek teljes mértékben feldolgozni egy kérést, ami akár több MB is lehet, majd, ha az megfelel a helyi biztonsági szabályzatnak, tovább engedni azt. Extrém esetben képesek lehetnek arra is, hogy egy protokollt konvertáljanak egy másikra (pl. imap->pop3). Ezt a problémát SPF alkalmazása esetén sokkal nehezebb megvalósítani.

# 6. Zorp

A Zorp egy alkalmazás szintű proxy tűzfal, azonban számos újítást tartalmaz a szokványos proxy tűzfalakhoz képest:

- Modularitás: a Zorp a protokollok felépítéséhez hasonlóan rekurzív, azaz képes lekezelni azt, ha egy protokoll beágyazott alprotokollt tartalmaz és a beágyazott protokollt is ellenőrizni kívánjuk.
- Kriptoproxy: a Zorp SSL proxyja képes a titkosítást eltávolítani az SSL-t alkalmazó protokollokról, így az ilyen adatok is ellenőrizhetőek (HTTPS, POP3S, IMAPS), valamint képes arra is, hogy az egyik oldalon cleartextben érkező folyamot SSL-be csomagolva küldje tovább a szerver felé.
- Elemzés mélysége: a proxyk fejlesztése során törekedtünk arra, hogy a protokollokat a lehető legrészletesebben elemezzük, így biztosítva a kommunikáló felek kölcsönös biztonságát.
- Konfiguráció: a konfiguráció leírásának nyelve egy programozási nyelv, a Python. Ennek segítségével a proxyk működése finomhangolható, szinte minden protokoll-eseményre saját eseménykezelőt írhatunk, mellyel beavatkozhatunk a proxyk működésébe. Így például a HTTP proxy URL szűrését megvalósíthatjuk tisztán Pythonban néhány sor megírásával.

A Zorp fejlesztése jelenleg két ágon zajlik: egyrészt a Zorp GPL alatti változata, mely szabad szoftverként letölthető és használható, másrészt a Zorp Professional, mely termékként kapható. A GPL-es változat teljes értékű, a különbség elsősorban a támogatott protokollok számában, illetve a csomagolás minőségében van.

A tervezett 1.0 kiadás a következő proxykat tartalmazza majd: HTTP, FTP, FINGER, WHOIS, SSL, POP3, IMAP, NNTP, PRINTER, TELNET.

# 7. Zorp a jövőben

A Zorp fejlesztése a továbbiakban a következők szerint zajlik:

- További protokollok támogatása
- Menedzsment eszközök
- Naplófeldolgozó eszközök
- Grafikus felület

# GIMP, avagy grafika unixon

Süveg Gábor <gsuveg@sgsystem.com>

2001.08.30.

**Kivonat**

A Unix rendszereket elsősorban kiszolgálóként ismerhettük meg. Az elmúlt évek, évtizedek során a kedvenc rendszerünk grafikai képességei is gyorsan fejlődtek. Ma már nyugodtan elmondhatjuk, hogy a unix rendszereken a tőlük megszokott stabilitás, megbízhatóság megtartása mellett, jól használhatjuk grafikai feladatokra. Az előadásom első részének célja áttekintést adni a szabad unix rendszereken használható grafikai lehetőségekről. Röviden átekintjük a fontosabb programokat, a fejlődési irányokat. Az előadás második részében a GIMP program bemutatása során az ismert és a kevésbé ismert lehetőségeit mutatom be.

## Tartalomjegyzék

**1. Bevezető** **98**

**2. A számítógépes képalkotás rövid áttekintése** **98**
2.1. Mely grafikai programokat használhatjuk linux alatt ? . . . . . 98
2.1.1. Pixelgrafikai programok . . . . . 98
2.1.2. Vektorgrafikai programok . . . . . 99
2.1.3. 3D rajzprogramok . . . . . 99

**3. A GIMP** **99**
3.1. Rövid bemutató . . . . . 99
3.2. A GIMP tulajdonságai . . . . . 100

# 1. Bevezető

A Unix rendszereket elsősorban kiszolgálóként ismerjük, a háttérben dolgozó sokszor monitor nélkül hónapokon keresztül leállítás nélkül megbízhatóan működő, külsőre nem a legszebb gépként. Szerencsére ez a helyzet az elmúlt évek gyors hardver fejlődésének és a unix terjedésének köszönhetően egyre szebb grafikus felületek (Gnome, KDE, Enlightenment stb.), és alkalmazói programok használhatók alatta. Mára már elmondhatjuk, hogy a unix felsorakozott a 'desktop' rendszerek közé (is). Természetesen ez még nem jelenti azt, hogy a szabad UNIX-ok megelőzik a professzionális grafikai rendszereket. De szerencsére vannak olyan területek, ahol a szabad unix rendszerünk már jobban megállja a helyét, mint a méregdrága eddig ismert grafikai operációs rendszerek. Az előadás célja nem a többi operációs rendszer és a grafikai alkalmazás véleményezése, hanem elsősorban a unixban rejlő lehetőségek kihasználásának bemutatása.

# 2. A számítógépes képalkotás rövid áttekintése

A számítógépes képalkotás nagyon rövid múltra tekint vissza, sőt az amúgy is rövid számítógépes történelem legrövidebb fejezete is a grafikáé. Az első számítógépek még nem rendelkeztek monitorral sem, majd a igények, és a technikai fejlődés eredményeképpen jutottunk el a mai, grafikailag rendkívül fejlett számítógépekhez. Ez a fejlődés rendkívül gyors, egy év múlva valószínüleg nevetve fogunk beszélni a mostani gépek grafikai képességein.

A számítógépes képábrázolást többféleképpen csoportosíthatjuk, de egyik kategória sem megfelelő. Nagyon nehéz a különféle művészeti irányokat, technikákat vagy módszereket kategorizálni, osztályozni.

Nézzük meg milyen osztályozások vannak: vektorgrafika-pixelgrafika, 2D-3D. A további osztályokat is felállíthatnánk, amiknek meg lenne mind, mind a maga előnye. De az előadásnak nem ez a célja.

## 2.1. Mely grafikai programokat használhatjuk linux alatt ?

### 2.1.1. Pixelgrafikai programok

(rövid felsorolás a teljesség igénye nélkül)

1. GIMP - (szabad unixon) a legelterjedtebb szabad képszerkesztő program. Az előadás második részében részletesen foglalkozom vele.

2. KDraw - nagyon igéretes program. A KDE csomag része, de még fejlesztés alatt áll.

3. OpenDraw (StarDraw) - az OpenOffice csomag része. A program elsősorban irodai feladarokra készült, igy a képek szerkesztésére nem annyira ajánlott

4. XPaint - egyszerű rajz program.

5. XV - az egyik unix-os örökzöld sajnos nem teljesen ingyenes.

#### 2.1.2. Vektorgrafikai programok

1. XFig - egyszerübb vektoros rajzok készítése
2. Killustrator - a KDE csomag része.
3. GYVE (Yellow Vector Editor) - erősen fejlesztés alatt
4. Gill - Gnome Illustrator (korábban SVG)
5. Sketch - igéretes vektoros rajzprogram

#### 2.1.3. 3D rajzprogramok

1. Blender - jól, sokrétüen használható program, (sajnos a forráskód nem szabad)
2. Moonlight - 3D rajzporgram, erősen fejlesztés alatt
3. PovRay - kicsit kilóg a sorból, de sokak által használt program
3. Maya - a Hollywoody stúdiókban is használt program, nemrég jelent meg a linuxos változata (nem ingyenes).

A felsorolásbol is látszik, hogy a grafika unixon még nagyon fiatal. Kevés jól használható programot találhatunk. A felsorolás korántsem teljes, sok hasznos, és igéretes program kimaradt a listából. Ebből a sorból kilóg a GIMP a tulajdonságaival.

# 3. A GIMP

A GIMP megjelenése óta szinte minden összehasonlításban az első helyet foglalja el a linuxos képszerkesztő programok között. Tudása, használhatósága eléri, néha meg is előzi a digitális képfeldolgozásban és grafikában eddig ismert, drága programokat. Vajon miért ilyen sikeres a GIMP? Mi a titka a sikerének? Hogyan lehet vele dolgozni? Hogyan lehet a GIMP rejtett tartalékait kihasználni? Ezekre próbálok nagyon rövid bevezetést tenni az előadás során.

## 3.1. Rövid bemutató

A GIMP a GNU Image Manipulation Program angol szavakból alkotott mozaikszó, jelentése pedig GNU Képszerkesztő program. 1995 nyarán a fejlesztést Spencer Kimball és Peter Mattis kezdte a Berkeley Egyetemen. Fél évvel később a korai próbaváltozat megjelent az Interneten és ezzel elkezdődött a nagy kaland. Ez a GIMP-változat még a Motif könyvtárat használta, mely abban az időben még kereskedelmi termék volt. Ez azonban megakadályozta a linuxos felhasználók között a széles körű elterjedését. Ezért 1996. júliusában megszületett a GIMP Toolkit (GTK), a GIMP eszközkészlet. Ekkor csatlakozott sok fejlesztő a munkához, segítve a tervezést, a fejlesztést és az ellenőrzést. Hosszú munka után 1997. februárjában megjelent a 0.99 változat. Ez már támogatta a rétegeket (layer) és számos bővítmény készült hozzá. Végül 1998. május 19-én megjelent a hosszadalmas munka gyümölcseként az első megbízható változat: a GIMP 1.0. A GIMP célja elsősorban a számítógépen felhasznált képek szerkesztése, rajzolása, módosítása volt. Mára elsősorban a webes felületre szánt grafikák elkészítésének kiváló eszközévé vált, de használják programfejlesztés során a program grafikai megjelenésének tervezésére is. Ma már minden szabadon terjeszthető Linux-rendszer része, de elérhető Win32 és OS/2 alatt is, sőt MacOSX alatt is.

## 3.2. A GIMP tulajdonságai

- Teljes rajzeszközkészlet: ecset, festékszóró, ceruza, toll, másolás, elmosás.
- Hatékony memóriagazdálkodás eredményeként a szerkeszthető kép méretét a szabad háttértár korlátozza!
- Kezeli a rétegeket (layer) és csatornákat (chanel).
- Rendkívül hatékony színátmenet és színkeverés.
- A GIMP belső szolgáltatásainak elérése a Script-Fu nyelven keresztül.
- Többlépcsős Visszavonás/Visszaállítás (ennek csupán a szabad háttértár szab korlátot).
- Korlátlan számú, egy időben megnyitható kép.
- Animációk készítése.
- Többek között e képformátumokat támogatja: gif, jpg, png, xpm, tiff, tga, mpeg, ps, pdf, pcx, bmp, psd. Ezek a képek betölthetők, menthetők, konvertálhatók a GIMP-pel.
- Kijelölés eszközök közül négyzet, ellipszis, szabad, Bezier-görbe, egybefüggő területek kijelölés használata.
- Több, mint száz beépített bővítmény, valamint a beépített script-fu segítségével korlátlanul fejleszthető.
- Egyéni ecsetek és kitöltőminták készítése.
- Digitalizáló tábla támogatása.
- Hatékony a betűkezelés.
- Transform eszközök: elforgatás, kicsinyítés, nagyítás, torzítás.
- Lapolvasó támogatás, képernyőkép mentése.

A rövid áttekintés után készítünk grafikákat a GIMP segítségével. Az idő rövidsége miatt a program rejtett, kevésbé ismert tulajdonságait mutatom be. A program segitsegevel elkeszitjuk egy weboldal grafikai munkáit. Megtervezzük az oldalt, készítünk plasztikus feliratot és többféle módon készítük gombokat.
Zárásként a GIMP programozásából, a 'script-fu' programozásról mutatok példákat.

# A BSD világa

Zahemszky Gábor <ZGabor@BSD.HU>
Magyar BSD Egyesület

2001.08.31.

**Kivonat**

A BSD világáról, filozófiájáról szól majd az előadás, sok érdekességgel a különböző licenszekről (XFree86, Perl Apache...), a BSD-rendszerek használatáról és használóiról.

## Tartalomjegyzék

**1. Bevezetés** **102**

**2. Az általánosságok után nézzünk néhány konkrétumot e rendszerekről!** **105**
2.1. Fájlrendszerek . . . 105
2.2. RAID, és vidéke . . . 106
2.3. Hálózat . . . 106
2.4. NAT . . . 107
2.5. Csomagszűrés . . . 107
2.6. Forgalomkorlátozás, -szabályozás . . . 108
2.7. Szoftver Portolhatóság . . . 108
2.8. Emulációk . . . 108
2.9. Csomagkezelés . . . 109
2.10. Kernel . . . 110
2.11. Naprakész BSD-k . . . 110

**3. BSD-k használata a világban és itthon** **111**
3.1. BSD-k elérhetősége . . . 111

# 1. Bevezetés

Mi is az a BSD? Első ránézésre valami rondaság, hisz ha egyáltalán képes ezt a nevet valaki megjegyezni (már ez se szokott sikerülni, van akik többszöri nekifutásra is csak BDS-nek hívják - mint pl. az APEH), akkor is csak az jut eszébe, hogy: valami, ami nagyon hasonlít az LSD-re (ami lehet, hogy jó, de nem biztos, hogy legális), és még ennél is sokkal jobban emlékeztet a BSA-ra (ami lehet, hogy legális, de biztos, hogy nem jó).

Szóval BSD. Maga a név egy rövidítés, a Berkeley Software Distribution (azaz Berkeley-i Szoftver Terjesztés) rövidítése. Persze, mint minden rövidítés, és azok magyarázata, ez se ér túl sokat. Így maradjunk annyiban: ez egy UNIX-verzió elnevezése. Ebbe persze nem megyek bele, hogy most a BSD-k valóban UNIX-ok, vagy nem, jogilag nem, gyakorlatilag igen. Ezen a konferencián elhangzik egy előadás, amelynek címe (UNIX vagy nem UNIX?) azt sejteti, hogy ugyanez a probléma fennáll a Linux esetében is. (Sajnos a tervezett időbeosztás ismeretében már sose fogom megtudni, kivéve, ha az előadó velem ellentétben gyönyörű és használható előadásanyagot küldött a szervezőknek a konferenciakiadvány megjelentetéséhez.) Én az egyszerűség kedvéért UNIX-nak fogom nevezni. Ha egy kicsit megvizsgálnánk a BSD múltját (amihez pl. a Linuxvilág c. lap II/1-es számában találhatunk egy kis segítséget), akkor sok minden kiderülne erről az oprendszerről, amely 1977-ben (vagy a körül) jelent meg, és az elmúlt nagyjából 25 évben jó pár hasznos funkció itt jelent meg először. Ez a UNIX-verzió jó néhány másik rendszer születéséhez járult hozzá. Az ismertebbek, ABC sorrendben: AIX - az IBM-től, HP-UX a HP-től, SunOS a SUN-tól, és Tru64 UNIX (másnéven Digital UNIX azaz DEC OSF/1 azaz Ultrix a Compaq-tól, korábban Digital-tól). Ezekről az évekről is kell(ene) beszélni, de figyeljünk egyelőre a jelenre: manapság a BSD a legtöbb hozzáértő számára semmit nem jelent.

Jelenleg a következő BSD verziók ismertek - megint ABC-rendben: BSD/OS, FreeBSD, NetBSD, OpenBSD. Ezek közül az első kissé kilóg a sorból, ez egy cég kereskedelmi terméke, melyet Intel platformra, elsősorban hálózati kiszolgálónak, valamint tűzfalakhoz ajánlanak. (A cég neve csak azért marad ki, mert az utóbbi időben a tulajdonoson kívül a neve is igen sűrűn változott.) (Találkozhatunk egyéb nevekkel: 386BSD - ez tulajdonképp már nem létezik; Trusted BSD, ami egy biztonsagi dolgokra kihegyezett BSD, a stabil dolgok rendszeresen kerulnek bele a FreeBSD-be.)

- A FreeBSD, amit én rendszeresen használok, egy eredetileg Intel platformra fejlesztett BSD változat, a 386BSD továbbfejlesztéséből alakult ki. Ez az elérhető BSD rendszerek közül talán a legjobb PC-s hardvertámogatással bír. Mint neve is jelzi, szabad. Forrása is, a használható bináris is ingyenesen elérhető - letölthető a hálóról, megvehető CD-n, és lemásolható (legálisan) ismerősöktől.

- A NetBSD egy olyan BSD változat, amelynél egyetlen jól megfogható cél létezik: „AZ´´ operációs rendszer akar lenni. Legfőbb jellemzője, hogy mindenféle hardveren képes működni. (Sajnos ez nem igaz, pl. a jelen levő AS400-on nem megy, de a felhasználók zöme által elérhető vasakat - PC, Amiga, Atari, Mac, Sun-ok, régi HP-k, Alpha-k, különböző palmtopok - támogatja.)

- Végül az OpenBSD. Eredetileg a NetBSD fejlesztői csapatából vált ki néhány ember, és a legbiztonságosabb operációs rendszer kifejlesztését tűzték ki célul. Alapként az akkori NetBSD-t vették, és azóta igen sokat változtattak rajta, elsősorban biztonsággal kapcsolatos dolgokat. (Volt a fejlesztésben egy igen hosszú

időszak, amikor gyakorlatilag az új funkciók beépítése megállt, és a meglevő kód igen erőteljes auditáláson ment keresztül.)

Ebből a bevezetésből avatatlan kívülálló azt szűri le, hogy akkor ez egy eléggé kusza rendszer, hisz van legalább 4 különböző verzió belőle, eléggé eltérő célokkal. (Ez a 4 azonban nem is sok, ha összevetjük a különböző Linux-terjesztések - disztribúciók - számával.)

A BSD rendszerek azonban közös alapból fejlődtek, tehát nagyon sok hasonló (nemegyszer megegyező) részt találhatunk. Az igazság az, hogy néhány, az egyik, vagy másik rendszerhez elkötelezett fejlesztőt kivéve, a fejlesztők jelentős része nem csak az egyik vagy másik, hanem 2-3 BSD fejlesztésében vesz részt. Ezek a részek aztán nyilván ugyanolyanok, ugyanúgy működnek, és a beállításuk is megegyezik.

Természetesen, mint a szoftveriparban mindenütt, az egyik fejlesztő(csapat) által kitalált, megvalósított dolgokat itt is ellopják a másiktól. Illetve nem lopják, ugyanis nincs rá szükség. A BSD-k egyik igen komoly eltérése a GNU/Linux világtól ugyanis a szoftverekhez tartozó engedély (licensz). Míg a GNU/Linux rendszereken használt programok többsége a GPL-t (vagy LGPL-t) használja, addig a BSD rendszereken használt programok zöme Berkeley-engedély alá tartozik. Ez rendszerint komoly vitákat eredményez, ugyanis a Berkeley-engedélynek van több, sokak - feltételezett - szabadságérzetét sértő kitétel. Ezek közül néhány:

- ezen programokat lehet módosítani anélkül, hogy a módosítást kötelező lenni kiadni
- nem kötelező a forráskódot átadni
- pénzért lehet árulni.

Ezt sokan félreértik, és a Linux- és egyéb rendszereket használók egy része néhány apróságot nem lát (az elvakult Linux hívők pedig nem hajlandók látni).

- Egy modern irodai Linux rendszer gyakorlatilag nem létezik a grafikus felület nélkül (és most nem a GNOME/KDE/??? rendszerekre, hanem az alatta levő X Window System rétegre gondolok). Ennek legsűrűbben használt változata - az XFree86 - nem GPL-t használ. Akkor tehát ne használjuk az X-et? Legtöbb Linuxos rendszer szerves részét képezi a Perl, mely az un. Artistic License (művészi engedély?) alapján terjeszthető. Akkor tehát ne használjuk a perl-t és az ezen alapuló dolgokat? (Megjegyzendő, hogy a szerző lehetővé tette programjának GPL-alapú terjesztését is.) De található sok Linux rendszeren Apache, (n)vi, és sok egyéb program, mely nem GPL-t használ. Tehát ezeket se kellene használnunk? Ezek a programok az eltérő jogi szabályozás ellenére nem rosszak. (Arról nem is beszélve, hogy még a legelvakultabb Nyílt Forráskód-hívők többsége is a Netscape, illetve az Opera böngészőt használja, és jóval kevesebben vannak a Konqueror illetve a Mozilla rendszeres - és kizárólagos - használói.)
- Senki nem kötelez minket arra, hogy BSD-alapú rendszerünkön csak BSD-engedélyes programokat használjunk, vagy fejlesszünk. Ezek a rendszerek is használnak olyan programokat, melyekhez tartozó engedély sokszor szigorúbb.
- És a talán legfontosabb: a felhasználók zömét nem különösebben érdeklik a Copyright-problémák. Ha venni kell a programot, veszi, ha lopni lehet, akkor lopja; ha nem kell lopni, mert legálisan juthat hozzá pénz nélkül (vagy legalábbis

jóval kevesebb pénzért) akkor legálisan gyűjti be; a lényeg: működjön, és azt és úgy csinálja, ahogy azt kell, és amire a felhasználója utasítja, ne pedig valami mást, kisujjból kiszopva. (Szerencsére(?) egyre többen vannak olyanok is, akik egyes cégek irreális üzletpolitikája, bizonyos termékek időnként a használatot is megnehezítő instabilitása, illetve a számítógépes világot elárasztó levélszemét és vírusáradat elleni védekezés okán szeretnének váltani.)

Térjünk vissza a BSD-khez. A továbbiakban (mivel csak eléggé rég, és eléggé keveset használtam) a kereskedelmi BSD-t, a BSD/OS-t nem tárgyalom. A BSD-k világa olyan világ, amely bizonyos területeken hasonlít, másokban eltér a Linux-világtól.

**Hasonlít abban, hogy:**

- legnagyobb részben szabad szoftver
- többségében megszállott őrültek fejlesztik
- több különböző, ámde eléggé hasonló változata létezik
- jól használható szerver funkciók ellátására, és lassan de biztosan halad abba az irányba, hogy használható alternatíva legyen a hozzá-nem-értők számára is a jelenleg elterjedt rendszerekkel szemben

**Eltérnek:**

- a legtöbb BSD rendszernél kimondva, kimondatlanul a szerver kategóriát, és sokkal kevésbé az otthoni játszó, internetező, filmet néző, zenét hallgató felhasználót célozzák meg (ezt talán kevésbé félreérthetően valaki egyszer így fogalmazta meg: az Adaptec RAID-vezérlő meghajtóprogramja fontosabb, mint az Opti931-es hangkártyáé).
- különböző BSD rendszerek vannak ugyan, de nincs többféle (pl.) NetBSD-terjesztés: ha valaki az 1.5.2-es NetBSD verziót használja, az eléggé konkrét dolgot takar, ha nem kezd a rendszergazda őrült rendszerátszervezésbe, akkor szinte mindegy, hogy mely vasat választottuk a telepítéshez, ugyanazt ugyanott fogjuk találni. (Ellentétben a legtöbb – eleve egymástól is különböző Linux-terjesztéstől –, amelyeknél a többé-kevésbé általánosnak tekinthető dolgokból általában a telepítés közben el kell dönteni, hogy:
  - melyik vi-hasonmást telepítsük (nvi/elvis/vim/...)
  - melyik levelezőszerver programot telepítsük (sendmail/smail/qmail/postfix/exim/...)
  - melyik karakteres levelező klienst telepítsük – természetesen teljesen feleslegesen, lévén a többség grafikus felületen dolgozik (mailx/pine/elm/mutt/mh/nmh/...)
  - melyik hírolvasó programot telepítsük, melyik grafikus levelező klienst telepítsük – és í. t.)
- ezeknél komolyabb eltérés: a BSD rendszerek fejlesztése a szoftverfejlesztésben eddig eléggé jól bevált központosított módszert követi. Általában jól elhatárolt területek léteznek, és az ezekhez értő személyek csoportja koordináltan dolgozik

ezen. Nincsen olyan egyértelműen azonosítható személy, mint Linus Torvalds, aki az egész felett őrködik, és akinek szava tulajdonképpen az egyetlen döntő szó a tekintetben, hogy valami igen, vagy nem: a fejlesztők közös döntése határozza meg, hogy valami a rendszer szerves részét képezi-e, vagy marad becserkészendő rendszerfolt (patch) valahol a Hálón.

- a rendszer szerves egész. Az én rendszerem nem Debian 2.2 (Potato) 2.2.19-es kernellel, a libc6-nak a 2.1.3-10-es verziójával, valamint a . . . -val . Nem, az én rendszerem FreeBSD 4.3-RELEASE, ha esetleg nagyon akarom, akkor a szükséges biztonsági foltokkal. És ezt így a BSD-sek többsége érti, és tudja, mit hol keressen.

- a csomagkezelés (a kereskedelmi BSD/OS-t kivéve) egységes és rugalmas. Nem vagy .tar.gz, vagy .rpm, vagy .deb, a hozzá tartozó segédprogramokkal. Egyetlen formátum létezik, ami egyébiránt .tar.gz, de „természetesen´´ a csomagkezelés mint olyan használatához szükséges funkciók megtalálhatók. (Látszólag nem sokban térnek el az egyes Linux-rendszerek rpm-jei, de legalábbis a linux- és linux-kezdő levelezőlisták, illetve az Index fórumain olvasható hozzászólások - valamint némi saját tapasztalat alapján – ez a csekély eltérés elég sok felhasználó – és persze néhány rendszergazda – életét meg tudja nehezíteni. És ha szerencsénk van, akkor szép lassan előáll ugyanez a deb formával is.)

- a legtöbb Linux-rendszerrel szemben, a rendszer beállításához (kernel fordítás, szoftverek működésének beállítása, egyebek) egyetlen jól használható, ámde kicsit nehezen tanulható „konfiguráló´´ eszközt lehet csak használni. Neve: „vi´´. Más nincs! A BSD rendszerekben az alaptelepítés (tehát ha extra programokat nem teszek föl) nem ad joe-t, emacs-ot, ae-t, nedit-et, és főleg nem ad adott terjesztésben (és csak ott!) használható konfiguráló eszközt. Erősebb idegzetűek esetleg megpróbálkozhatnak az ed-vel – vagy telepítsék föl az igényelt plusz szövegszerkesztőt, ami valószínűleg előrefordított csomagként megtalálható.

# 2. Az általánosságok után nézzünk néhány konkrétumot e rendszerekről!

## 2.1. Fájlrendszerek

A BSD-kben gyakorlatilag egyetlen fájlrendszer létezik, ezt a szakirodalomban FFS-nek, UFS-nek nevezik. Ez megegyezik a legtöbb kereskedelmi UNIX-rendszerben megtalálható fájlrendszerrel, akkor is, ha azokban nevük, esetleg belső szerkezetük kisebb-nagyobb változtatáson átesett. Ezt a formátumot közel 20 éve fejlesztették ki, és az eltelt idő a garancia arra, hogy a fejlesztés során elkövetett programozói hibák eltűntek. Stabil, megbízható darab, melynek fejlesztésekor az adatbiztonság a teljesítménynél is fontosabb volt. Ezt elsősorban a (rendszer számára) fontos adatok szinkron diszkre írásával oldja meg. Ebből természetszerűleg következik, hogy alapállapotban teljesítménye elmaradhat az általánosan elterjedt ext2 fájlrendszert használó Linuxokkal szemben, mert az ext2 aszinkron írásmódot használ a metaadatok írásakor is – ami nagyobb sebességet, viszont sérülékenyebb rendszert eredményez. A sebesség természetesen fontos, ezért a következő lehetőségek adottak azoknak, akik vágya a minél nagyobb fájlrendszersebesség elérése.

Stabilan egyelőre csak a NetBSD-nek része, de létezik egy LFS-nek (Log Structured File System) nevezett, nevéből látható, naplózó fájlrendszer. A szekvenciális diszkre írás miatt az olvasás és írás is gyorsul, míg a naplózás miatt a megbízhatóság is megfelelő.

Aki szeret kockáztatni, az BSD alatt is beállíthatja az aszinkron írást – azaz ugyanazt, ami ext2-n alapértelmezés

Végül, jelenleg mindegyik BSD-nek része egy viszonylag új mechanizmus, amit soft-updates-nek neveznek. Nagyon leegyszerűsítve a dolgot, a rendszer a diszkre írandó adatok között függőségi fát épít fel, és ez alapján dönti el, hogy mely diszkterületet kell kiírni. Ennek segítségével megmarad a szinkron írásmód stabilitása, és majdnem az aszinkron írás sebességét kapjuk. Ez már ugyanaz a stabilitás, mint amit a kereskedelmi UNIX-rendszerekben – néha pénzért, néha szériatartozékként – adott különböző naplózónak (journaling) nevezett fájlrendszerek (GNU/Linux rendszeren a ReiserFS, az ext3, az SGI-féle XFS, illetve az IBM-féle JFS) nyújtanak, egyetlen hátulütője a rendszer nem-szabályszerű leállása utáni diszkellenőrzés lassúsága. (Ezt naplózó fájlrendszerek esetén azzal kerülik meg, hogy a napló alapján, csak a legutolsó módosításokat ellenőrzi a rendszer.) Ez természetesen komoly hátrány, ezért egy újabb eszköz jelent meg mostanában: a rendszerindítás utáni, háttérben történő fájlrendszer-ellenőrzés (background fsck). Azaz miután a rendszer elindul, egy-két pillanat múlva már használható a fájlrendszer, a különböző inkonzisztenciákat a normál működés közben javítja a rendszer.

Az UFS fájlrendszerben a fájlokhoz tartozik néhány speciális jelző. A chflags szolgáltatás hasonlít az ext2-fájlrendszerben megtalálható, chattr paranccsal állítható dolgokhoz. Beállítható egy fájlra például, hogy ne lehessen törölni, vagy ne lehessen semmilyen módon változtatni. Amiben eltér az ext2-s megvalósítástól, hogy ezekből a lehetőségekből kettő van: egy, a közönséges felhasználók (nyilván a fájl tulajdonosa) által állítható, és egy, csak a rendszeradminisztrátor által állítható példány. Ez utóbbinak van egy igen érdekes tulajdonsága: a kernel speciális beállítása mellett ez utóbbi jelzők törlése még a rendszergazdának sem engedélyezett.

## 2.2. RAID, és vidéke

Természetesen a BSD-k is támogatják a különböző gyártók különböző RAID-funkciókat nyújtó eszközeit, a nevesebbeket mindenképp. Ezen túlmenően, mindegyik BSD-nek része egy vagy több különböző szoftveres megoldás. Az első ilyen megvalósítás neve CCD (Concatenated Disk Drives). Ez mindhárom szabad BSD-rendszernek része, a RAID0-nak és a RAID1-nek megfelelő szolgáltatásokat lehet vele elérni. Fejlesztése tulajdonképpen áll, sajnos bizonyos karbantartó funkciók hiányoznak. NetBSD és OpenBSD alatt egy RAIDFrame-nek nevezett rendszer érhető el, amely RAID0-tól RAID5-ig mindent nyújt (és ezek egymással is kombinálhatók), természetesen hotspare támogatással. Ennek FreeBSD-re az elmúlt hónapban készült el az átvitele (portolása), az írás pillanatában még nem szerves része a rendszernek, csak foltozással lehet elérni. A FreeBSD része egy Vinum nevű eszköz, tulajdonképpen ugyanezekkel a funkciókkal, csak másvalakinek egy másik fejlesztése. (Erről homályos információk hallhatók, miszerint portolták a másik két BSD-be is, de ott nem rendszertartozékok.)

## 2.3. Hálózat

Mindhárom szabad BSD „A´´ referencia TCP/IP megvalósítást használja, mely az elmúlt évtizedek alatt eléggé kiforrott, gyerekbetegségeit kinövő programrendszer lett.

Természetesen mind az IPv4, mind az IPv6 használható, Magyarország első, kísérleti IPv6-os hálózatát is BSD-s gépek alkották. Szerencsére mindhárom rendszer ugyanazt az IPv6-os megvalósítást építette rendszerébe, így ezek között csekély eltérés található. A BSD-k szerves részét képezi az IPSEC támogatás (IPv4-en is). Használható IPX/SPX, AppleTalk, valamint NetBSD és OpenBSD alatt néhány manapság kevésbé elterjedten használt hálózati protokoll is, pl. Xerox NS.

A SLIP, CSLIP és elsősorban a PPP ugyanígy rendszertartozék. Míg a SLIP megvalósításban nincs különbség, addig OpenBSD-n és FreeBSD-n egy ppp-nek nevezett program (ez egy speciális „tun´´ névre hallgató eszköz segítségével, felhasználói programból intézi a PPP protokollt), míg NetBSD-n és FreeBSD-n egy pppd-nek nevezett, kernel szinten működő szoftver használható. Két megjegyzés: FreeBSD-n ezek szerint két különböző megvalósítás is létezik (ez még néhány egyéb dologgal így van); ez utóbbi pppd megegyezik a Linux-rendszerekben található pppd programmal (mind a kernel-szintű, mind a felhasználói szintű része ugyanaz).

Az IP, IPX és AppleTalk támogatáson kívül az egyéb oprendszerek fájl- és nyomtatómegosztási protokollját is támogatják, a Samba, a NetAtalk, a MaRS (valószínűleg sokak által ismert) programok segítségével.

## 2.4. NAT

A korábbi Linux-kernelekben masquerading-nak nevezett, a hivatalos dokumentációkban NAT (Network Address Translation – kb. Hálózati Címfordítás) névre hallgató funkció is elérhető. FreeBSD-n, NetBSD-n az alaprendszer része Darren Reed ipfilter nevű programcsomagja, melynek egyik funkciója a címfordítás. Ez az utolsó OpenBSD verzióból (tulajdonképpen) személyes okokból hivatalosan kikerült, de a fejlesztője változatlanul karbantartja az OpenBSD specifikus részeket, így gyakorlatilag ott is használható. OpenBSD-s fejlesztők gőzerővel dolgoznak egy pf-nek nevezett, az ipfilterrel (legalább parancssori beállításszinten, egyes hírek szerint kernel programozói felület-szinten) kompatibilis megvalósításon, amely valószínűleg az ipfilter funkcióit átveszi. FreeBSD-n ezen kívül létezik egy speciális rendszermechanizmus, amit „divert socket´´-nek neveznek (eltérített hálózati csatlakozás), és ennek, a csomagszűrőnek, és egy „natd´´-nek nevezett felhasználói programnak a segítségével is megoldható a NAT. (Újabb funkció, melynek FreeBSD alatt több különböző megvalósítása létezik.)

## 2.5. Csomagszűrés

Az előbb említettek szinte szóról-szóra igazak, ugyanis az ipfilter (és az OpenBSD-s pf) elsődlegesen csomagszűrők. Azaz NetBSD-n, FreeBSD-n rendszertartozék, míg OpenBSD-hez külső csomag. OpenBSD-ben pf néven egy másik megvalósítás érhető el hivatalosan. FreeBSD-n szokás szerint létezik még egy eszköz, ezt „ipfw´´-nek hívják (és az 1.x-es Linux-kernelbeli Alan Cox-féle csomagszűrő ezen alapult. BSD -> Linux, és nem viszont :-) Azóta persze sokat változott.) Ezek mindegyike, azaz az ipfilter is, a pf is, és most már jó ideje (még ha sok FreeBSD felhasználóban nem is tudatosult idáig) az ipfw is un. SPF, azaz állapottartó csomagszűrő (Stateful Packet Filter). Ennek lényege, hogy a ki- be- irányuló csomagokat nem önmagában értelmezi a rendszer (ami a csomagszűrők alapvető viselkedési módja), hanem az esetlegesen korábban már áthaladt csomagokban található információkat is figyelembe veszi. Ennek segítségével bizonyos szükséges hálózati forgalom lényegesen biztonságosabbá tehető.

Az újabb Linux-kernelekben (2.4-től már a stabilban is) ennek (az SPF-nek) egy némileg továbbfejlesztett változata található kapcsolatkövetés (connection tracking) néven. (Bővebben lásd pl. az egyszer már emlegetett Linuxvilág-ban, az I/1-es számtól sok ezer részen át futó sorozatot. Esetleg az itteni előadások ...)

## 2.6. Forgalomkorlátozás, -szabályozás

A Linux-világban „Traffic shaping´´ néven emlegetett lehetőség alapban csak FreeBSD alatt található meg, neve „dummynet´´. (Eredetileg hálózati teszt-eszköznek készült, innen a lehetetlen elnevezés.) A korábban említett ipfw segítségével állítható be. Ez látszólag ahhoz vezet, hogy akinek mind csomagszűrésre, mind forgalomkorlátozásra szüksége van, az az ipfw-t használja, ezzel szemben az ipfw és az ipfilter szépen megférnek egymás mellett, így akinek kell, megteheti, hogy a kiforrottabbnak tartott ipfilterrel szűr, és a dummynet segítségével fogja vissza a forgalmat.

## 2.7. Szoftver Portolhatóság

Mivel a BSD elég régóta létező rendszer, ezért a Linux-előtti időkben megírt programok tekintélyes hányada probléma nélkül, vagy kisebb „kozmetikázás´´ után fordítható és használható. Az utóbbi évek erőteljes Linux-centrikussága a kezdő (és sajnos nem csak a kezdő) úgymond UNIX-programozók (helyesebben Linux-programozók) között kissé nehezítette a dolgot, de szerencsére ez is javul. Egyre többen ismerik fel, hogy léteznek Linuxon kívül is egyéb rendszerek, és a szabad szoftver közösségben sem feltétlenül hátrány, ha az ember nem zárja ki a potenciális felhasználók egy részét. Ráadásul a BSD rendszereknél is fontos a különböző (értelmes) szabványok betartása, így mindhárom BSD tartja magát pl. a legelterjedtebbnek tekintett POSIX-szabványhoz. Mivel a BSD-s tábor sem csekély, itt is vannak olyanok, akik ha találkoznak egy számukra fontos vagy érdekes programmal, akkor ha tudnak, segítenek azt az általuk preferált BSD-n is futtathatóvá tenni (ezzel gyakorlatilag a másik két BSD is hozzájut); illetve zárt kódú program esetén a cégből vagy a kódot, vagy egy BSD-(ke)n futó binárist kinyerni.

## 2.8. Emulációk

Mivel a BSD-k egyelőre a kereskedelmi szoftvereket gyártó cégek számára amolyan mostohagyerekek (szerencsére egyre több kivétel van), nagyon fontos az adott vason futó egyéb operációs rendszerek által futtatható alkalmazások BSD alatti futtathatósága. Intel/AMD/... alapú PC-n ez mindhárom BSD-n ugyanazt jelenti: elsősorban a mindenki által nagyrabecsült M$ cég W* nevű szoftverére írt programok, illetve (szégyen-gyalázat, hogy egy mondatba kerülnek) Linux-ra írt alkalmazások futtathatóságát. Erre ugyanazokat a lehetőségeket nyújtják. Egyrészt a feltehetően sokak által ismert WINE-emulátor elérhető BSD-ken is, így szinte minden windows-os program, amit a WINE képes futtatni, fut a BSD-ken.

Másrészt mindhárom BSD-nek létezik Linux-emulátora (szakszerűen talán jobb lenne szimulátornak nevezni, sőt még ezt is lehetne ragozni), és ennek a kódnak a segítségével Linux-alkalmazások is futtathatók. Sőt, a hab a tortán, az egyelőre zárt kódú VMWARE segítségével Linux-emuláción fut VMWARE, abban esetleg Windows 2000 – vagy Linux – vagy más. (És mivel a Linux-emulátor kernel-szintű megvalósítás, nem nagyon tapasztalható az emulátorokra jellemző lassulás.) NetBSD-n és FreeBSD-n létezik némileg korlátozott képességű SCO UNIX, és elég jól használható Solaris

emuláció is, OpenBSD-n sajnos nem tudom. A nem PC-platformokon az adott vas eredeti gyártója által fejlesztett oprendszer alkalmazásainak futtatása többé-kevésbé megoldott (pl. Alpha-kon Tru64-es alkalmazásoké.)

Jelenleg a FreeBSD az egyetlen, mely több processzor használatát támogatja. A hamarosan megjelenő 5.0-s sorozat egyik leglényegesebb változtatása az ehhez kapcsolódó rendszerrészekben található. Természetesen mind a NetBSD, mind az OpenBSD fejlesztők célkitűzései között szerepel.

## 2.9. Csomagkezelés

Ez a BSD-rendszerek egyik nagy erőssége. FreeBSD és OpenBSD alatt az elnevezések megegyeznek, míg a NetBSD egy némileg eltérő (talán még logikusabb) elnevezést használ. A rendszer szerves részét képező programokon kívül kétféle, azonnal (vagy minimális erőfeszítéssel) elérhető szoftver létezik. Az elsőt csomagnak (package) hívják. Ez gyakorlatilag a különböző Linux-terjesztések rpm vagy deb csomagjaival megegyező kategória. Ez .tar.gz formájú, amelyben valamely program futásához szükséges fájlok találhatók, a futtatható, a segédprogramok, a csak hozzá tartozó könyvtárfájlok, a dokumentáció, stb. A csomagfájlban az adatok a rendszerbeli katalógusstruktúrájuknak megfelelő formában találhatók. Ezen kívül speciális nevű adatfájlokban néhány, a csomag kezelésével kapcsolatos információ: a telepítés előtt, vagy utána végrehajtandó műveletek; mely más csomagok megléte szükséges; illetve milyen fájlok tartoznak a csomaghoz – ez az információ például a csomag eltávolításához szükséges. Ha egy csomagot föltelepítek, akkor a működéséhez szükséges egyéb csomagokat is automatikusan fölrakja a csomagkezelő. A csomagkezelő helyben, vagy URL-lel adott, ftp protokollal elérhető csomagot képes telepíteni.

A másik lehetőség FreeBSD-ben és OpenBSD-ben port, NetBSD-ben pedig a sokkal logikusabb csomagforrás (package source, pkgsrc) névre hallgat. (NetBSD-n portnak a rendszer különböző vasakon futó verzióit nevezik – azaz van ARM-port, meg Alpha-port, de az mgetty elérhető csomag, és csomagforrás formájában is.)

Ez egy adott programnál egy kicsi, normális esetben csak 5-10 fájlt tartalmazó katalógusstruktúra. Az itt található fájlok tartalmazzák a program felépítéséhez szükséges – esetleg – letöltendő fájlok nevét, forrásának helyét, a letöltés módját – ftp/http, és a forrásprogramból a végrehajtható előállításához szükséges utasításokat. Ha valamilyen számunkra szükséges program előfordított bináris csomagként nem létezik, akkor telepítése összesen két utasítást igényel:

```
# cd /usr/ports/audio/mpg123
# make install
```

Ez először letölti a forrást (illetve ha a forrás már rendelkezésre áll, és megfelelő helyen van – ez FreeBSD alatt pl. a /usr/ports/distfiles könyvtár –, akkor a letöltés elmarad), a letöltés után kicsomagolja egy átmeneti könyvtárba, a program fordításához esetleg szükséges javításokat automatikusan elvégzi, lefuttatja a ./configure-t (és javít a végeredményen), ha erre szükség van, lefordítja a leírófájlokban szereplő beállításoknak megfelelően, föltelepíti a rendszerbe, elkészít egy csomagleírót (hogy később a csomagkezelő parancsokkal lehessen kezelni), és kész. Mivel így a fordítás szemete a rendszerben marad, általában e helyett a

```
# make install clean
```

javasolt. Ez takarít is maga után. Ha valakit zavar, hogy a letöltés, a foltozás, a fordítás rendszergazdai jogokkal történik, akkor lehet:

```
$ cd /usr/ports/audio/mpg123
$ make all
$ su(do)
# make install clean
```

formában is. További lehetőségként lehet csak a letöltésig, vagy csak a foltozásig menni a:

```
$ make fetch
$ make patch
```

parancsokal. Ez utóbbi elsősorban olyankor javasolt, ha valaki saját beállításokkal szeretné fordítani, nem pedig azokkal, melyeket az adott csomag elkészítője jónak lát.

Természetesen a leírókban a portok esetén is megtalálhatók a különböző függőségek, így egy csomag előállítása minden esetben föltelepíti a program használatához szükséges egyéb szoftvereket. Mint az várható, ezeket a függőségeket a rendszer számon tartja, így a csomagkezelő parancsok szabályos használatakor nem állhat elő az, hogy leszedünk egy feleslegesnek ítélt programot, amelyre egy másiknak szüksége van. (FreeBSD alatt van két furcsa csomag: KDE és GNOME, ezek olyan csomagok, melyek egyedül csomagfüggőségeket tartalmaznak, így egyszerűsítve ezeknek a rendszereknek a teljes és konzisztens telepítését.)

## 2.10. Kernel

A BSD rendszerek kernelje monolitikus. Hasonlóan a Sun, Compaq, HP – vagy éppen Linux – rendszerekhez, BSD alatt is lehet un. modulokat használni. Ezt NetBSD és OpenBSD alatt az LKM (Loadable Kernel Module), míg FreeBSD alatt az ennél egy fokkal többet nyújtó KLD (Kernel Loadable Driver?) névre hallgató szolgáltatás nyújtja.

A rendszer bizonyos paraméterei a kernel újrafordításával, mások az un. sysctl mechanizmus segítségével, futás közben változtathatók. Ez utóbbi Linux-alatt is létezik, de BSD-k alatt nem a /proc matatását jelenti a sysctl.

Létezik egy szolgáltatás, amely legalábbis név alapján szintén ismerős lehet: úgy hívják, security level (védelmi szint). Ez egy kernel változón kersztül szabályozható (természetesen sysctl-lel), ezt a változót még a rendszergazda is csak növelheti, csökkenteni nincs joga. Ennek a szintnek a beállításai sok, a rendszer biztonságára veszélyes műveletet tiltanak. Adott szint fölött, a rendszergazdának nincs joga a fájl-jelzőket törölni (chflags), nincs joga a csomagszűrő szabályrendszerén változtatni, a diszk perifériákhoz közvetlenül hozzányúlni, és hasonlók. Ezen lehetőségek féktelen használatával elő lehet állítani igen biztonságos, de főként kezelhetetlenül nehézkes rendszereket – ezeket feltörni aztán a valódi hozzáértő számára szinte kihívás lesz, de kétségtelen, hogy napjaink átlag cracker-je valószínűleg csak bambán néz.

## 2.11. Naprakész BSD-k

Mivel a BSD-k eléggé szorosan összetartozónak tekintik a kernelt, a rendszerkönyvtárakat és a különböző felhasználói programokat, ezért a különböző új verziók elérésére is kialakult egy viszonylag egységes mechanizmus. Nincs olyan, hogy izgatottan várjuk a legújabb kernelt, esetleg az új libutils csomagot, és csak ezeket frissítjük. Az egész rendszer CVS (Concurrent Versioning System) fában tárolódik, és gyakorlatilag

az egész rendszert egyszerre lehet csak frissíteni. Ebből következően nem sok esély van arra, hogy frissítjük egy adott funkció kernelbeli részét, de megfeledkezünk a felhasználói rétegről. A BSD-knél – hasonlóan a Linux-kernelhez – párhuzamosan két fejlesztési ág létezik. A fejlesztői ág (Current) tartalmazza az újdonságokat, ez viharos tempóban szokott változni, időnként teljesen használhatatlan – gyakorlatilag csak akkor javasolt használni, ha valakinek van rá külön élete. A stabil változat lassabban halad, ide a fejlesztői ágban már stabilnak minősülő dolgok kerülnek, pár hetes, hónapos tesztelés után. A stabil ágat egy adott pillanatban befagyasztják, ilyenkor kizárólag a már benne levő dolgok javítása kerül bele, majd útnak engedik az ekkor release-nek nevezett változatot. Ez általában CD-n megvásárolható, illetve CD-image formájában letölthető. Néha kisebb méretű részekre szétszedett formában is hozzá lehet jutni.

# 3. BSD-k használata a világban és itthon

*The Matrix*

*Walnut Creek CDROM/Wind River – a világ legnagyobb forgalmú ftp-szervere*

*FTPSEARCH*

*Yahoo!*

*IPv6 magyar háló*

*FTP.FSN.HU – Magyarország valószínűleg legnagyobb szabadszoftver szervere*

*c3 (freemail)*

*Datanet*

*Matáv*

*Axelero (MatávNET)*

## 3.1. BSD-k elérhetősége

*WWW.FreeBSD.ORG*

*WWW.NetBSD.ORG*

*WWW.OpenBSD.ORG*

. . . és ezek WWW.országrövidítés.xxxBSD.ORG -formája: pl.

*WWW.HU.FreeBSD.ORG*

*WWW.xxxBSD.HU*

*WWW.BSD.HU*

# Biztonságos chroot kialakítása GNU/Linux Operációs rendszeren

Zámbó Marcell

2001.08.30.

**Kivonat**

Az előadás célja részletesen bemutatni egy, a Unix rendszereken rendelkezésre álló chroot rendszerhívás, ráépülő alkalmazás, illetve a hozzátartozó környezet biztonságos kialakítását, használatát.

## Tartalomjegyzék

**1. Mi az a chroot? 114**

**2. A chroot alkalmazás biztonságos használata 114**

**3. Chroot környezet kialakítása 114**

**4. Szükséges könyvtárszerkezet kialakítása 115**

**5. Alkalmazások, rendszerkönyvtárak (libek) 115**

**6. Jogosultságok, mount opciók 115**

**7. Chrootok fajtái 116**

**8. Külső környezet 116**

**9. Mi ellen nem véd a chroot? 117**

**10. Egyéb megoldások 117**

# 1. Mi az a chroot?

A mindennapokban a chroot általában valamilyen népszerűbb nevén jelenik meg, például: jail, sandbox, subsystem és így tovább. Chroot alatt általában három dolgot szoktunk érteni:

- - a chroot rendszerhívást [1]
- - a chroot alkalmazást [2]
- - illetve a chroot környezetet.

A chroot rendszerhívás feladata a gyökérkönyvtár megváltoztatása, a chroot alkalmazás pedig ehhez a feladathoz nyújt egy egyszerű felületet. Ha nem programozói szemmel vizsgáljuk a kérdést, gyakorlatilag az első kettő egynek tekinthető.

Eredetileg az init használta a gyökérkönyvtár beállítására, ez a feladata ma már megszűnt, inkább mint lehetséges biztonsági kiegészítés tölti be feladatát.

A chroot alkalmazás legalább egy paramétert vár, ez az új gyökérkönyvtár elérési útja, második paraméterként a futtatandó parancsot várja, amennyiben ez nincs megadva, úgy a SHELL környezeti változóban található beállítást próbálja meg futtatni, ha nincs SHELL környezeti változó, ebben az esetben a /bin/sh-t kísérli meg futtatni. A chroot rendszerhíváshoz, így a chroot alkalmazásához is rendszergazdai jogosultság (de legalább cap_chroot [3]) szükséges.

A chroot által megvalósított védelem alapja a filerendszer szintű leválasztás: a futtatni kívánt alkalmazás, alrendszer körül egy kellően kicsiny, minimális környezetet lehet kialakítani, amely nem tartalmaz felesleges alkalmazásokat, hibalehetőségeket.

# 2. A chroot alkalmazás biztonságos használata

```
cd /new chroot
. /bin/ls
```

Fontos: A gyökérkönyvtár váltása előtt célszerű az aktuális könyvtárt az új gyökérre (vagy az alá) állítani, ellenkező esetben elég egyszerű módon ki lehet jutni a régi gyökérbe. (chroot break [4] )

# 3. Chroot környezet kialakítása

- - ldd [5] Megmutatja az ELF formátumú alkalmazások rendszerkönyvtárakhoz (libekhez) való függését, pl.: ldd /bin/bash
- - ld.so [6] Dinamikus fordító/betöltő alkalmazás, elsődleges feladata a szükséges rendszerkönyvtárak betöltése a futtatni kívánt alkalmazás számára. Ezen funkciója mellett azonban az ldd-hez hasonló feladatra is használható. Pl.: /lib/ld-linux.so.2 –list /bin/bash
- - strace [7] Segítségével nyomon követhető az adott alkalmazás működése vagy éppen nem működése, hasznos eszköz lehet ismeretlen alkalmazás jailesitésekor.

# 4. Szükséges könyvtárszerkezet kialakítása

A chroot környezetekről általában elmondható, hogy a következő könyvtárakat nagy valószínűséggel tartalmazzák: /bin /etc /lib /tmp /var, illetve ez alatt /var/run és így tovább. A megvalósítandó környezettől függően szükségessé, illetve feleslegessé válhatnak könyvtárak, ezt az adott cél szabja meg. Ha az alkalmazás dokumentációjában említett könyvtárakat létrehozzuk a jailben, nagy valószínűséggel ezt a problémát elkerülhetjük. Amennyiben az alkalmazás mégsem működik tökéletesen, úgy a strace segítségével általában kideríthető a hiba oka. Pl.: strace -o log -f /usr/bin/apache ...

# 5. Alkalmazások, rendszerkönyvtárak (libek)

Érdemes odafigyelni az alkalmazás által igényelt állományokra, rendszerkönyvtárakra, esetleges kiegészítő alkalmazásokra, sok esetben a használni kívánt eszköz képessége meghaladja az általunk vele szemben támasztott igényeket. Például mi csak egy egyszerű statikus webszervert szeretnénk, de az apache ennél jóval többet tud. Tapasztaltabb unixosok megpróbálhatják az igényeknek megfelelőre szabni az alkalmazás tudásszintjét, vagy ha tökéletessé tenni nem is tudják, legalább a felesleges funkciókat eltávolíthatják az alkalmazásból. (./configure [8] –without-... –disable-..., vagy vi main.c :)

Egy alkalmazás függőségeinek megismerésére alkalmas eszközök a fentebb bemutatott ldd és ld-linux.so.

Fontos: sem az ldd, sem az ld-linux.so nem mutat meg bizonyos függőségeket, ezek a függőségek pedig az ú.n. nss (Name Service Switch) rendszerkönyvtárak, ennek oka, hogy ezekre nem az alkalmazás hivatkozik, hanem maga a glibc [9].

# 6. Jogosultságok, mount opciók

Általában olyan alkalmazásokat helyezünk jailbe, melyek biztonságos működésében nem bízunk túlságosan, ezért is célszerű minden eszközzel növelni a jailen belüli biztonságot. Néhány ötlet ennek megvalósítására:

- Chmod [10]: - Általában sem a könyvtáraknak, a jailen belüli alkalmazásoknak, rendszerkönyvtáraknak (libeknek) nem kell olvashatónak, listázhatónak lennie. - Az irható file-ok, könyvtárak ne legyenek olvashatóak.
- Chattr [11]: - A jailen belül a csak olvasható/futtatható állományokra célszerű az immutable flaget feltenni. - Az írható állományok törlésének megnehezítésére jól használható az append only flag.
- Mount [12] opciók:

  - A chroot környezeten belül ne legyen set[ug]id állomány, az esetlegesen odakerülő set[ug]id alkalmazások futtatását a nosuid mount opcióval meggátolhatjuk.

  - A jailben eszközfile-t nem célszerű tartani, mivel azok segítségével igen könnyen ki lehet törni az elzárt környezetből, az esetlegesen odakerülő eszközfile-ok használatát a nodev opcióval hiúsíthatjuk meg.

Ahhoz, hogy ezen opciókat használhassuk, a chroot környezetet külön partícióra kell elhelyezni. Egy nem csak jailben követhető egyszerű szabály alkalmazásával tovább fokozható a rendszer sérthetetlensége: ahonnan fut alkalmazás, oda nem lehet írni, ahova lehet írni, onnan nem futhat alkalmazás. Íme:

```
mount /dev/hdc1 -o nodev,nosuid,ro /jail/exec
mount /dev/hdc2 -o nodev,nosuid,noexec,rw /jail/write
```

- Fontos: a noexec opció elég könnyen megkerülhető az ld-linux.so használatával, pl: /lib/ld-linux.so /jail/write/bash
- Érdekesség: chroot környezet készítésekor érdemes elgondolkodni a statikusan fordított alkalmazások használatán. (gcc [13])

# 7. Chrootok fajtái

Általában kétféle chroot környezetet alkalmaznak jelenleg. Az első, amikor az alkalmazás a chroot környezeten kívülről indul, majd adott feladatok elvégzése után a jailen belül tevékenykedik (szép magyarsággal bechrootol).

A második forma, amikor az alkalmazás már eleve a chroot környezetben kezdi meg működését. Mindkét esetnek megvannak az előnyei, illetve hátrányai. Az első esetben a futtatott alkalmazás és annak indító/beállító állományai nincsenek a jailben, így a támadó nem tudja azokat megszerezni, azonban a chroot művelet elvégzésének sikeressége az alkalmazástól függ. A második esetben az alkalmazásnak nincsen szüksége a chroot elvégzéséhez, azonban az indításához, valamint futtatásához szükséges valamennyi állomány megtalálható a jailben.

# 8. Külső környezet

A chroot környezet kialakításához hozzátartozik annak elhelyezése valamely működő rendszeren. Ez is legalább akkora körültekintést igényel, mint az előző feladatok. Néhány ezen feladatok közül.

- Syslog [14]. Ha tudni szeretnénk, mi történik a jailen belül, valamilyen formában ki kell onnan juttatni a naplóbejegyzéseket, semmi esetre se válasszuk a jailen belüli file-ba történő naplózást, mivel a jail esetleges támadhatósága esetén a napló törölhető/módosítható lesz. Célszerűbb megoldás a syslogd unix socketen vagy hálozaton keresztüli használata. (Esetleg egy chrootolt syslogd :)
- Ulimitek [15]. Ha nem szeretnénk, hogy a jailben futó alkalmazás teljes egészében magával ránthassa rendszerünket, úgy célszerű a PAM által nyújtott erőforráskorlátozást alkalmazni [limits.conf]. (Érdekes kérdés a jailen belül használt uid kérdése is, azaz hogy a jailen belül használt uid létezzen-e a külső rendszeren.)
- Indító/leállító [16] scriptek. Ezek módosítása általában nem túl nehéz feladat, azonban érdemes több időt szánni rájuk. (Nézzük csak meg egy chrootolt alkalmazás környezeti változóit:

  Pl.: cat /proc/[chroot-olt-alkamazás-pid]/environ)

## 9. Mi ellen nem véd a chroot?

Sajnos számtalan ilyen dolog van, de ezen problémák egyéb megoldással jórészt kiszűrhetőek, illetve kockázatuk csökkenthető. Két csoportra bontva, ha rootként (sic!), vagy ha normál felhasználóként fut az alkalmazás. Az első esetben mondhatni semmitől sem véd, amennyiben az alkalmazás rávehető valamilyen módon külső kód futtatására, úgy nagyon nehéz azt megvédeni bármitől is. Normál felhasználóként futó alkalmazás esetén jó eséllyel megvédi rendszerünket a külső támadástól, itt rendszer alatt ne a chrootot értsük, hanem az azt befogadó hostot. Néhány veszély [17], ami ellen tényleg nem véd: Stack exec, ptrace, shm, localhoston figyelő alkalmazások elérhetősége, erőforrások. . .

## 10. Egyéb megoldások

Egyéb megoldások/kiterjesztések a chroot biztonságának növelésére:

- - hap [18] patch
- - capability tools: lcap [19], lids [20]

## Hivatkozások

[1] man 2 chroot

[2] man 8 chroot

[3] *ftp://ftp.guardian.no/pub/free/linux/capabilities/capfaq.txt*

[4] chroot break:
*http://www.bpfh.net/simes/computing/chroot-break.html*

[5] man 1 ldd

[6] man 8 ld.so

[7] man 1 strace

[8] *http://www.gnu.org/software/autoconf/autoconf.html*

[9] *http://www.gnu.org/software/glibc/*

[10] : man 1 chmod

[11] : man 1 chattr

[12] man 5 fstab

[13] *http://gcc.gnu.org*

[14] man 8 syslogd

[15] *http://www.kernel.org/pub/linux/libs/pam/*, man limits.conf

[16] man start-stop-daemon

[17] *http://www.theaimsgroup.com/~hlein/hap-linux/*

[18] *http://www.theaimsgroup.com/~hlein/hap-linux/*

[19] *http://www.netcom.com/~spoon/lcap/*

[20] *http://medusa.fornax.sk/*

Jegyzetek

Jegyzetek