# Szabad Szoftver Konferencia és Kiállítás 2009

*követő kiadvány*

fsf.hu
alapítvány

A konferencia rendezője az:


Fő támogató:


Kiemelt támogató:


Kiemelt médiatámogató:


Ezüst fokozatú támogatók:




Támogató:


Médiatámogatók:


# Szabad Szoftver Konferencia és Kiállítás 2009.

(követő kiadvány)

Budapest, 2009. október 31.

A kiadvány tördelése a TEX 3.141592 verziójával készült,
GNU/Linux operációs rendszeren.
A TEX az American Mathematical Society bejegyzett védjegye.

Szerkesztette: *Zelena Endre*
Lektorálás: *Kiss Gabriella*

FSF.hu Alapítvány
URL: *http://www.fsf.hu/*
E-mail: *fsf@fsf.hu*

ISBN 978-963-06-8276-3

# Tartalomjegyzék

**Bártházi András: Redis, a memcached gyilkos** **7**

**Bodnár Csaba – Sörös József: Hyppolit – otthonautomatizálás Linux-alapon** **17**

**Farkas Szilveszter: A Launchpad.net fejlesztői platform** **29**

**Höltzl Péter: Naplóelemzés – syslog-ng OSE** **35**

**Kadlecsik József: iptables és ipset** **45**

**Kis Gergely: Google Android fejlesztői szemmel** **57**

**Mátó Péter: Adatbiztonság és üzembiztonság mindenkinek? Egy frappáns válasz: uDS** **75**

**Papp-Varga Zsuzsanna: A GeoGebra alkalmazása a matematikaoktatásban az általános iskolától az egyetemig** **85**

**Papp Zsolt: Linux vékonykliens megoldások** **91**

**Sütő János: Egy irtó jó spamszűrő** **99**

**Szalai Kálmán: OpenOffice.org projekt ma és holnap** **107**

**Szántó Iván: Kernel Virtual Machine és Virtualizáció** **119**

**Szőke Sándor: LyX – Újdonságok, a hatékony szövegszerkesztés érdekében** **135**

**Torma László: Új technológiák az Ubuntuban** **149**

**Trencséni Márton – Gazsó Attila: Nyílt forráskódú elosztott rendszerek** **161**

**Trencsényi József: Független fejlesztés menete nyílt forráskódú szoftverekkel** **173**

**Vajna Miklós: Verziókezelés a Git használatával** **179**

# Előadások

# Redis, a memcached gyilkos

Bártházi András

**Kivonat**

Az előadásom célja, hogy bemutassa a Redis nevű[1], memória alapú key-value adatbázis lehetőségeit, illetve hogy miben tér el más projektektől. Számos hasonló projekt létezik, melyekkel összehasonlítani nem fér bele sem ennek a dokumentumnak, sem az előadásom kereteibe, mindazonáltal megpróbálok képet adni arról is, hogy milyen egyéb lehetőségek vannak.

Először a key-value adatbázisokról ejtek szót, illetve arról beszélek, hogy milyen gondolatiság van ennek a filozófiának újbóli fellángolása mögött, miért is aktuális ez a téma manapság, miért célszerű megismernünk minél több lehetőséget ezirányban.

Az előadás másik felében pedig a Redis lehetőségeit mutatom be a funkcióitól a backupolási lehetőségekig, s szó lesz majd egy esettanulmány keretében arról is, hogy milyen gondolkodásmóddal tudjuk hatékony módon kihasználni a Redis, illetve a hasonló projektek lehetőségeit.

# Tartalomjegyzék

**1. Redis** **8**
- 1.1. Hasonló projektek . . . . . 9
- 1.2. Támogatott nyelvek . . . . . 9
- 1.3. Parancsok . . . . . 10
- 1.4. Backup . . . . . 13
- 1.5. Replikáció . . . . . 13
- 1.6. Ha az adatbázis nem fér be a memóriába . . . . . 14

**2. Esettanulmány: egy Twitter klón** **14**

**3. Összefoglalás** **16**

[1] http://code.google.com/p/redis/

# 1. Redis

Az előadásom címében a Redist memcached gyilkosnak neveztem, mivel számos tekintetben többet nyújt annál úgy, hogy megtartja annak előnyeit is. Alább össze is hasonlítom majd a két projektet, de először arról is essen szó, hogy mi is a Redis.

A Redis egy key-value adattárolást megvalósító, az adatokat memóriában tároló, de perzisztensen háttértárra írni képes, az értékek tekintetében nem csak egyszerű sztringeket, de összetettebb listákat, halmazokat atomi műveletekkel is támogatottan megvalósító adatbázis, mely replikációra is képes.

Rendkívül dinamikusan fejlődő projektről van szó, mely pár hónap alatt jutott el az 1.0-s kiadásig úgy, hogy már korábban is stabilan használható megoldást kínált. Hamarosan várható az 1.1-es kiadás is, mely további újdonságokkal, lehetőségekkel szolgál. A projekt igen jól dokumentált, számos programozási nyelven érhető el kliens szoftver hozzá.

A key-value adattárolás mivoltát az előzőekben már tisztáztam, de nézzük, hogy a több funkció mit is jelent. A Redis a memóriában tárolja az összes adatot, amiből egyenesen következik az is, hogy maximum annyi információt tudunk segítségével letárolni, amennyi belefér a szerver memóriájába. Mivel járulékos információk is letárolásra kerülnek, ezért figyelembe kell venni, hogy 1 GB memóriába nem tudunk 1 GB-nyi adatot letárolni, csak kevesebbet. A pontos overhead jelentősen függ attól, hogy milyen adatokkal dolgozunk, a Redis FAQ-jában olvasható példa szerint rossz esetben akár 84%-os veszteséget is jelenthetnek a különböző indexek (ez egy elég elrettentő szám, ennél sokkal jobb százalékok is kihozhatóak, mivel duplikált értékeket képes felismerni a Redis, illetve az értékeket tömöríteni is tudja).

A Redis képes az adatokat a háttértárra is kiírni, ezáltal perzisztens adattárolást valósít meg. Az adatok kiírásának folyamata jól hangolható, megadhatunk intervallumot (pl. 5 percenként), írások számát (pl. minden 10.000 írás) is, de mindezeket vegyesen is be lehet állítani. Adatkiírást kezdeményezni lehet menet közben is bármikor. Az adatok kiírása közben a szerver kiszolgálja a kéréseket, nem áll meg, de a háttértárra kiírt adathalmaz konzisztens marad, s a kiírás kezdeti pillanatának állapotát tartalmazza.

Az összetett adattárolást illetően nem csak sztringek tárolhatóak el a Redis adatbázisában értékként, hanem támogatva vannak a listák és a halmazok is. Ez azt jelenti, hogy listákhoz, halmazokhoz különböző műveletek segítségével atomi módon tudunk például hozzáadni vagy lekérdezni elemeket, s ezekhez különböző parancsokat is kapunk (lásd később). További adatszerkezetek is tervbe vannak véve a Redis következő verziójába.

Ami a replikációt illeti, a Redis több szerver között is fenn tud tartani konzisztens adatállapotot. A replikációhoz szükséges szinkronizálást önállóan, beavatkozás nélkül képes beindítani, illetve fenntartani.

A Redis ezeket a funkciókat nagyon gyorsan képes kivitelezni, másodpercenként százezer feletti írásműveletet, illetve százezerhez közeli olvasási műveletet lehet megvalósítani a segítségével egy átlagos szerveren. Ami a konfigurálást, telepítést illeti, nagyon gyorsan, szinte nulla konfigurációval üzembe állítható a szerver szinte fapados módon, és egyből tesztelhető, kipróbálható. A parancsai egyszerűek, s gyorsan tanulhatóak, s egy kiváló, stabil eszközt ismerhetünk meg egy rövid ismerkedés során.

## 1.1. Hasonló projektek

Ami a memcached-del történő összehasonlítást illeti, az eddig leírtakból talán látható, hogy miben nyújt a Redis többet a Memcached szolgáltatásainál. Legfőképpen az összetett adatszerkezetek, illetve a perzisztens adattárolás lehetnek azok a funkciók, melyek miatt érdemes lehet váltani, de persze az igényektől függ, hogy ez szükséges-e. A Memcached-del szemben a Redis segítségével nem csak cache megoldásokat, hanem akár "igazi" adattárolást is megvalósíthatunk, mely egészen más felhasználási módokat is lehetővé tesz a Memcached lehetőségeinek megtartása mellett.

Létezik egy MemcacheDB nevű projekt is, mely valójában csak nevében és protokolljában rokon a Memcacheddel, hiszen a netes interfész megvalósítását annak protokolljával kompatibilisen valósították meg, az adattárolása azonban nem a memóriában történik, hanem a BerkleyDB projektet használja erre a célra. Tekintettel arra, hogy írás műveleteknél itt minden alkalommal háttértárra írási művelet valósul meg (az mondjuk nagyon gyorsan), illetve hogy a BDB is csak egyszerű adatszerkezeteket támogat, ezért a Redis ezzel a projekttel szemben is előnyösebb választás lehet.

## 1.2. Támogatott nyelvek

A Redishez rövid pályafutása alatt a közösség jóvoltából (illetve a jól dokumentált protokolljának köszönhetően) számos programozási nyelven készült kliens, illetve további nyelvekhez viszonylag egyszerűen implementálható kliens megvalósítás.

Ruby, Python, PHP, Erlang, Tcl, Perl, Lua, Java és Scala nyelvek alkotják a hivatalos listát, az egyetlen hiányzó ismertebb nyelvcsalád a .Net vonal. Egyes nyelvek alá (Ruby, Perl...) nem csak a parancsokat megvalósító, de komplexebb rutinkönyvtárak is rendelkezésre állnak.

## 1.3. Parancsok

A Redis parancsait négy csoportba sorolnám, vannak az alap-, a lista és a halmaz műveleteket megvalósítóak, továbbá az egyéb, legfőképpen adminisztrációs célúak. Persze ez elég önkényes besorolás, a projekt wiki oldalán specifikusabb kategorizálással találkozhatunk. Íme a fontosabb műveletek.

**Alapműveletek**

- SET kulcs érték: beállít egy új értéket a kulcs kulcshoz
- GET kulcs, MGET kulcs1, kulcs2, kulcs3...: egy, illetve több kulcs értéke kérdezhető le
- EXISTS kulcs: definiált-e adott kulcsú elem?
- INCR kulcs, DECR kulcs: az adott kulcs numerikus értékének növelése, illetve csökkentése 1-gyel
- INCRBY kulcs érték, DECRBY kulcs érték: mint az előző, de konkrét értékkel növelés, illetve csökkentés
- DEL kulcs: adott kulcsú elem törlése az adatbázisból
- KEYS minta: a "minta" mintával kezdődő kulcsok nevének lekérdezése
- RANDOMKEY: egy véletlenszerű elem értékének lekérdezése
- RENAME régikulcs újkulcs: kulcs átnevezése
- EXPIRE kulcs másodperc: az adott kulcshoz lejárati idő társítás

A SET, GET, INCR/DECR műveletek sztring értékű kulcsok esetében használhatóak. Értékadás jellegű műveleteknél, ha egy kulcshoz eddig nem volt érték letárolva, akkor a Redis nem reklamál, hanem létrehozza az adott kulcsot, ez a tapasztalatok alapján így praktikus művelet.

Az EXPIRE művelet viszonylag egyedi módon került megvalósításra, ugyanis amennyiben egy adott kulcshoz lejárati időt társítunk, legközelebbi olvasást és írást is tartalmazó műveletkor (például INCR) a korábbi érték megsemmisül, avagy hiába nem járt még le a kulcs, de nulla értékről indul ezeknél a műveleteknél. Ez a replikáció konzisztensen tartása miatt működik így.

**Listaműveletek**

- RPUSH kulcs érték, LPUSH kulcs érték: az adott kulcsú listába új elemet szúr érték értékkel, jobb oldalra, illetve bal oldalra
- LLEN kulcs: lista hosszának lekérdezése
- LRANGE kulcs kezdet vég: lista egy adott szakaszának lekérdezése
- LTRIM kulcs kezdet vég: lista csonkolása csak a megadott szakasz megtartásával
- LINDEX kulcs index: adott indexű elem lekérdezése a listából
- LSET kulcs index: adott indexű elem értékének módosítása a listában
- LREM kulcs darab érték: adott értékű, maximum "darab" darab elem eltávolítása a listából
- LPOP kulcs, RPOP kulcs: listából egy elem kiemelése bal, illetve jobb oldalról

A lista pozíciók (kezdet, vég) megadásakor negatív számok is használhatóak, a "-2" érték hátulról a második elemet jelöli. A halmaz műveletek igen érdekes felhasználási lehetőségeket rejtenek:

- SADD kulcs érték, SREM kulcs érték: adott értékű elem halmazhoz adása, törlése
- SPOP kulcs: egy véletlenszerű elem kivétele a listából
- SMEMBERS kulcs: halmaz elemeinek a lekérdezése
- SMOVE kulcs1 kulcs2 érték: az adott érték egyik halmazból a másikba történő mozgatása
- SCARD kulcs: adott halmaz tagjainak száma
- SISMEMBER kulcs érték: adott érték eleme a halmaznak?
- SINTER kulcs1 kulcs2 kulcs3...: halmazok metszetének lekérdezése
- SINTERSTORE célkulcs kulcs1 kulcs2 kulcs3...: halmazok metszetének lekérdezése és letárolása a "célkulcs"-ú halmazba
- SUNION kulcs1 kulcs2 kulcs3...: halmazok összegének lekérdezése

- SUNIONSTORE célkulcs kulcs1 kulcs2 kulcs3...: halmazok összegének lekérdezése, és letárolása a "célkulcs"-ú halmazba
- SDIFF kulcs1 kulcs2 kulcs3...: halmazok különbségének lekérdezése
- SDIFFSTORE célkulcs kulcs1 kulcs2 kulcs3...: halmazok különbségének lekérdezése és letárolása a "célkulcs"-ú halmazba

Egy halmazban egy adott érték csak egyszer szerepelhet, ha megpróbálunk hozzáadni egy már szereplő értékű elemet a halmazhoz, akkor a művelet "sikertelen" lesz (hozzáadáskor válaszul megkapjuk az információt, hogy mennyi elem lett sikeresen hozzáadva a halmazhoz). Egy elég összetett lehetőségeket kínáló, a rendezés gondolatára felépített művelet is elérhető, mely listákkal és halmazokkal is működik:

- SORT kulcs
- SORT kulcs DESC
- SORT kulcs LIMIT 0 10 ALPHA DESC
- SORT kulcs BY suly_*
- SORT kulcs BY suly_* GET object_*

A művelet adott kulcs elemeit képes rendezni (növekvő, csökkenő sorrendbe, numerikusan és alfanumerikusan), s limitálható az eredményhalmaz is a LIMIT kifejezés hozzáadásával. A BY kulcsszó segítségével bár a kulcs elemei lesznek rendezve, de nem a saját értékük szerint, hanem a BY által megadott prefixhez hozzáfűzött értékük szerinti kulcs értékét véve. Ha a GET kulcsszót is megadjuk, akkor válaszként nem az adott kulcs elemeit fogjuk visszakapni, hanem a GET által megadott prefixhez hozzáfűzött értékük szerinti kulcsok értékét.

A SORT funkció nem tud csodát művelni, ami a sebességet illeti, gyorsrendezés az algoritmusa. Alfanumerikus rendezéskor a LC_COLLATE környezeti változó értékét veszi figyelembe.

**További érdekesebb műveletek**

- DBSIZE: adatbáziselemek számának lekérdezése
- TYPE kulcs: kulcs típusának (sztring, lista, halmaz) lekérdezése
- SAVE, BGSAVE, LASTSAVE: háttértárra mentéssel kapcsolatos műveletek

Nem mutattam be az összes műveletet, de a Redis által biztosított lehetőségek talán jól láthatóak ezen felsorolás után. Egy átgondolt, biztos alapokon álló fejlesztésről van szó, minden parancs esetén use-case indokolja annak meglétét, illetve csak olyan parancsok állnak rendelkezésre, melyek legtöbb esetben gyorsak, vagy melyek nem feltétlenül gyorsak, de a funkcionalitáshoz sokat hozzátesznek.

Nem vettem bele a felsorolásba, de a Redis több adatbázist is támogat, melyeket sorszámokkal lehet elérni, s alapból 16 áll rendelkezésre. Ezek leginkább névtérként működnek, ezek között a névterek között is lehet kulcsokat mozgatni igény szerint. Használatuk helyett inkább több Redis szerver indítása (különböző portokon) lehet javasolt, tekintettel arra, hogy így az adatbázismentések is elkülönülnek, hogy többprocesszoros szerveren a Redis egyszálas futtatási környezete így jobban megoszlik a processzorok között, egy esetleges blokkoló művelet csak egy adott Redis szervert tart fel, s mert talán egyszerűbb a nyilvántartása egy portnak, mint egy Redis adatbázis sorszámnak.

## 1.4. Backup

A Redis perzisztens módon háttértárra, konkrétan egy darab fájlba írja adatbázisának tartalmát a konfigurációban meghatározott időpontokban, események bekövetkeztekor. Ez a fájl a memóriatartalom egy speciális, gyorsan kiírható, illetve gyorsan beolvasható lenyomata, más célokra nem használható - nem valami egyszerű fájlformátum, hanem tömény bináris fájl.

A kimentett tartalom backupolása igen egyszerű: csak le kell másolni a fájlt. Ez akár menet közben is történhet, a szerver leállítása nélkül, még akkor is, ha éppen egy háttértárra mentési folyamat fut éppen. A Redis egy átmeneti fájlt hoz létre, és a rename nevű parancs segítségével írja felül a korábbi fájlt, így biztosítva azt, hogy az átnevezés csak akkor történik meg, ha a fájlba írás véget ért, és sikeres volt. A művelet atominak számít, tehát vagy sikeresen megtörténik a régebbi fájl felülírása, vagy sikertelen lesz a művelet, s a korábbi fájl nem sérül. Backupolni ezen kívül egy master-slave replikáció beállításával is lehet egy slave adatbázison, így még az I/O műveletekkel sem terhelve a fő szervert.

## 1.5. Replikáció

A Redis replikációja egy elég egyszerűen konfigurálható, master-slave felállású replikáció, mely lehetővé teszi több slave használatát is, illetve a slave-ek láncként felfűzve (a slave is masterként viselkedik, hozzá további slave csatlakozik) gráfot is alkothatnak.

Replikáció közben a master szerver nem áll le, így egy slave kiesésekor vagy beállásakor (szinkronizáció esetén) a master továbbra is kiszolgálja a kéréseket.

Ezzel szemben a slave oldalán a replikáció beindítása, a master állapotának felvétele közben nem végez műveleteket, ezzel is biztosítva az adatok konzisztenciáját.

## 1.6. Ha az adatbázis nem fér be a memóriába

Az adatbázisnak a Redis esetén be kell férnie a memóriába. Ennek megkerüléséhez a Redisnek nem célja segítséget adni, bár már felmerült egy olyan jövőbeni fejlesztési irány a levelezőlistán (egészen más okokból), hogy a Redis adattárolását megvalósító layer esetlegesen cserélhető lesz. Mindenesetre ha az adatbázisunk nem fér el a memóriában, akkor vagy nem tudjuk a Redist használni, vagy valamilyen adattárolási stratégiát bevezetve meg kell kerülnünk a kérdést.

Egy lehetőség, hogy a Redist csak cache-ként használjuk azokhoz az adatokhoz, melyekről tudjuk, hogy jelentős memóriát igényelnének. Ekkor ezekhez az információkhoz lejárati időt rendelünk az EXPIRE segítségével, így ha fogyóban a memória, a Redis fel fogja ezeket a területeket szabadítani, illetve maguktól is felszabadulnak egy idő után. Az írást mind a Redisbe, mind pedig a nagyobb kapacitású másik adatbázisba elvégezzük, s először megpróbáljuk a Redisből kiolvasni az információt, ha ott nem létezik, akkor fordulunk a másik háttértárhoz. Ezt a felállás mi sikeresen használjuk.

# 2. Esettanulmány: egy Twitter klón

A Redishez szinte az indulása óta elérhető egy PHP-ben írt Twitter klón forrása, mely nagyon egyszerű forráskódjával, illetve az alkalmazott stratégiákkal jól szemlélteti a Redis lehetőségeit, illetve azt a gondolkodásmódot, melyet el kell sajátítanunk, ha key-value adatbázist szeretnénk használni fejlesztéseinkhez.

Erre a gondolkodásmódra SQL múlttal nem is olyan egyszerű átállni, mert ellentmond minden ott megtanult metodikának, többek között a normalizálás szabályainak. Mindazonáltal ha végiggondoljuk, hogy mi történik egy relációs adatbázisban a háttérben, látni fogjuk, hogy gyakorlatilag kevés a különbség ami a letárolt adatokat illeti, az indexek létrehozásával és redundáns tárolásával egy relációs adatbázis is hasonlóképpen működik mint amire szükség van key-value adatbázisok esetén.

A Redis wikiben található kis tutorialt érdemes végigolvasni, mivel olyan alapokkal kezd, hogy mi is jelent a key-value adattárolás a gyakorlatban, illetve mik azok az atomi műveleteket. Ebből a tutorialból emeltem ki pár gondolatot, melyek megmutatják, hogy hogyan lehet felépíteni egy összetett adatbázist. A Twitter klón nem csak üzenetek küldését teszi lehetővé, hanem egy teljes(ebb) klónról van szó, mely a regisztrációt, a felhasználók személyes oldalát, ismerőseinek kezelését is tartalmazza.

Ez a Twitter-klón a felhasználókat számmal azonosítja, melynek előállításához egy `"global:nextUserId"` kulcsú elemet használ. Új felhasználó létrehozásakor a következő parancsokat futtatja le a rendszer:

```
INCR global:nextUserId (pl: 1000)
SET uid:1000:username felhasznalonev
SET uid:1000:password jelszo
```

Ebből a három sorból már rengeteget lehet tanulni. A kulcsok kapcsán, mint látható, speciális elnevezéseket célszerű használni, "névtereket" létrehozva a kettőspontok használatával. Míg egy key-value adatbázis alapvetően nem strukturált, ezzel a trükkel hierarchiát vezethetünk be. A felhasználói -azonosító legyártásához szükséges változót a globális névtérbe, a felhasználó azonosítójához köthető információkat az "uid" névtérbe helyeztük.

Vegyük észre a párhuzamot az adatbázis táblákkal is, hasonló a helyzet, mintha egy "uid" táblában két oszlopot hoznánk létre, az 1000-es id-jű felhasználónak két oszlopa: a "username" és "password" állnak rendelkezésre. Most egy fontos design patternt tanultunk meg a key-value adatbázisokat illetően.

Mivel csak kulcs szerint érhetőek el az adatbázisunk értékei, jelenleg nem tudjuk lekérdezni azt, hogy a "felhasznalonev" felhasználóhoz milyen azonosító, milyen jelszó kapcsolódik, bejelentkezéskor, illetve a felhasználó saját oldalának kiszolgálásakor azonban erre az információra szükségünk van. Mint látható, az adatbázis tervezését jóval alaposabban végig kell gondolnunk, mint egy relációs adatbázis esetén, legfőképpen azt kell megvizsgálnunk, hogy milyen módon szeretnénk a későbbiekben hozzáférni az adatbázisban szereplő információkhoz.

A felhasználónévből felhasználói azonosító létrehozását egy új kulcs létrehozásával tudjuk megtenni, relációs adatbázisok esetén erre egy index szolgált volna:

```
SET username:felhasznalonev:uid 1000
```

A Twitternél minden felhasználónak vannak követői, illetve követik is felhasználók. Itt is hasonló stratégiát kell alkalmaznunk, mint az előbb, ugyanis mind a két irányból szeretnénk lekérdezni az információt a szolgáltatás működtetésekor (kiket követ, kik követik a felhasználót), ezért két kulcsot fogunk használni felhasználónként erre a célra. Ami az adatszerkezetet illeti, a halmazok tökéletes választást jelentenek majd:

```
uid:1000:followers
uid:1000:following
```

Egy nagyon fontos adatstruktúra még hátra van, a felhasználó legfrissebb hozzászólásait szeretnénk megjeleníteni az oldalán, mondjuk visszanézhetően maximum ezret. A hozzászólásokat egy globális névtérben fogjuk eltárolni, s a felhasználókhoz hasonlóan azonosítókkal látjuk el, s hogy lekérdezhető legyen, egy adott felhasználónak milyen hozzászólásai voltak, egy listát vetünk be:

`uid:1000:posts`

Amikor a felhasználó hozzászól, akkor a globális lista adminisztrációja mellett ebbe a listába is felvesszük a hozzászólás azonosítóját (LPUSH), majd a listát az ezredik elem után csonkoljuk (LTRIM), hogy ne nőjön a végtelenségig. A lista "lapozhatóan" az LRANGE parancs segítségével érhető el. A Twitter klónt bemutató wiki oldalon ennél sokkal több gondolat olvasható, de ezzel a kivonattal talán sikerült megmutatnom pár design patternt, melyek jól használhatóvá teszik a key-value adatbázisokat, illetve megmutatják azon gyengeségeit, melyek erősségként is felfoghatóak.

# 3. Összefoglalás

A fentiekben olvasható bemutatóból, illetve reményeim szerint az előadásomból kiderül: egy nagyon jól használható adatbázis szervert ismerhetünk meg a Redis személyében, ha úgy döntünk, kipróbáljuk, mit tud. Sok olyan, gyorsan tanulható, egyszerűen használható megoldást kínál sok problémára, melyek webes alkalmazásfejlesztéskor kifejezetten hasznosak tudnak lenni.

Mi a Miner.hu projektünk keretében, illetve iWiW alkalmazásokhoz használjuk háttéradatbázisként a Redist. Az előbbihez nagyrészt cache megoldásként, a friss bejegyzések memóriában tárolására, tematikus rovatainknál pedig a friss bejegyzéseket listákban tartjuk nyilván. Eddigi tapasztalataink azt mutatják, hogy egy terhelhető, stabil, jól használható eszközt vezettünk be, melyet másnak is szívesen ajánlunk. Előadásom után szívesen válaszolok kérdésekre, illetve a blogomon[2] is közzé lesz téve előadásom, illetve ezen dokumentum. Twitteren lehet követni csiripelésemet, felhasználói azonosítóm `ba78`.

[2] `http://webakademia.hu/`

# Hyppolit – otthonautomatizálás Linux-alapon

Bodnár Csaba – Sörös József

**Kivonat**

Előadásunk a Hyppolit otthonautomatizálási szoftver felépítését és működési logikáját ismerteti. A rendszer magja, a H-core bemutatása után rátérünk az eddig elkészült modulok ismertetésére, majd néhány példán keresztül bemutatjuk a háztartásunkban lévő eszközök vezérlésének – kvázi végtelen – lehetőségeit.

## Tartalomjegyzék

**1. Áttekintés** **18**

**2. A H-core** **18**
2.1. Esemény-broker az egyes modulok között . . . . . 19
2.2. Modulok kezelése, állapotának figyelése . . . . . 20

**3. Modulok** **20**
3.1. I/O-modulok . . . . . 20
3.2. Logikai modulok . . . . . 21
3.3. Emberi kapcsolattartás modulok . . . . . 24
3.4. Egyéb modulok . . . . . 25

**4. Fejlesztési irányok** **25**

**5. Felhasználási lehetőségek** **25**
5.1. Műszaki felügyelet, vagyonvédelem . . . . . 26
5.2. Otthonautomatizálás . . . . . 26

# 1. Áttekintés

A Hyppolit együttműködő egységgé olvasztja össze egy lakás védelmi rendszerét és elektromos háztartási berendezéseit, szem előtt tartva a készülékek működtetésének és használatának egyszerűsítését, a biztonság, a kényelem és az irányíthatóság összehangolásának lehetőségeit. Segítségével valóra válhat a teljes biztonságtechnika, a háztartási berendezések, a szórakoztató elektronika, a számítástechnika és a lakóhelyéhez tartozó világítás, a hűtés-fűtés, a páratartalom-szabályozás, a kaputelefon és telefonrendszer, a redőny- és kapuvezérlés, az uszodatechnika, az öntözőrendszer, a szauna, a pezsgőfürdő és az egyéb kényelmi berendezések működésének ellenőrzése és irányítása. Az alkalmazott modell és az azon alapuló műszaki megoldás lehetővé teszi bonyolult döntési logikák megvalósítását gyártóktól független eszközök felhasználásával, segítve az új, eddig még nem implementált funkciók villámgyors létrehozását. A felhasználó önállóan is megvalósíthatja elképzeléseit egy egyszerű script nyelv programozásával.

A kezelőfelület lehet egyszerű nyomógomb, mobil telefonkészülék, iPhone vagy számítógép – akár az Interneten keresztül elérhető módon. Így az ellenőrzés és az irányítás abból a pontból történhet, ahol a felhasználó éppen tartózkodik. Továbbá nem csak a telepített funkciók beállításait változtathatja igénye szerint (pl. feltekeri a fűtés hőmérsékletét), hanem kedvtelésből, szabad idejében önállóan is programozhatja rendszert.

A rendszer alapvetően az alábbi feladatköröket valósítja meg:

- rossz szándékú események megelőzése vagy értesítés ezekről megfigyelő, riasztó és védekező funkcionalitás megvalósításával (otthonvédelem)
- otthoni háztartási berendezések vezérlése, működésük ellenőrzése (otthonautomatizálás)

# 2. A H-core

A Hyppolit nem egy program, hanem programok összessége, melynek célja egy épület vezérlése. Az egyik alapvető koncepció a tervezéskor az volt, hogy minél stabilabb és konfigurálhatóbb legyen. Ennek köszönhető, hogy teljesen moduláris, minden feladat elvégzésére külön modul való, amit bármikor le lehet cserélni vagy frissíteni. A Hyppolit központjában az ún. H-core áll. Ehhez modulokon keresztül kapcsolódnak a rendszer különböző komponensei. Maga a core limitált funkciókészlettel rendelkezik. Még olyan alapvető funkció, mint a hálózati kommunikáció más gépeken futó Hyppolit-modulokkal is modulként van megvalósítva. A tervezők szándéka szerint komplett alrendszerek több kisebb modul összekapcsolásával valósíthatók meg.

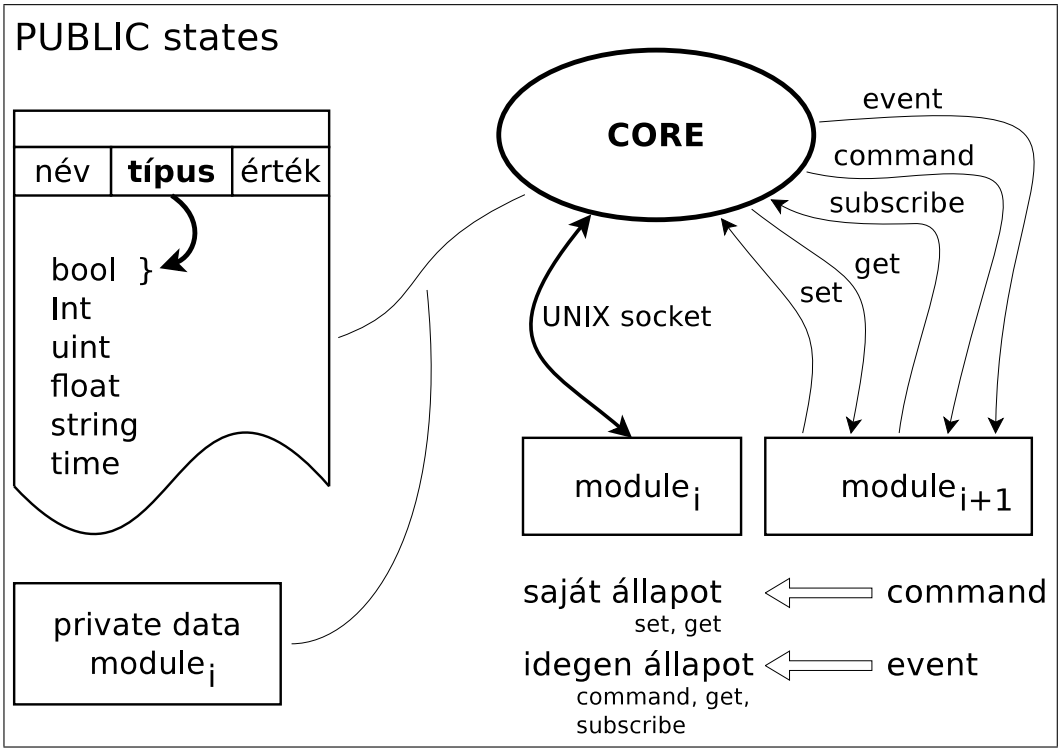
A H-core funkciói:

- Esemény-broker az egyes modulok között
- Modulok kezelése, állapotának figyelése

A rendszer tervezésekor lefektetett alapelvek rugalmas implementációt és későbbi bővíthetőséget tesznek lehetővé.

## 2.1. Esemény-broker az egyes modulok között

A core működésének lényege központilag kezelt változók menedzselése. Az egyes modulok létrehozhatnak saját nyilvános változókat, illetve beregisztrálhatnak a más modulok által létrehozott változókban bekövetkező változásokra. Ilyenkor a core értesíti őket a változásokról. A modulok kiadhatnak parancsokat is más modulok változóival kapcsolatosan. Minden modulnak lehet privát adatterülete is, amit más modul nem lát. Ebből a core-ban tud egy másolatot tárolni amit a core a modul esetleges újraindítása esetén vissza tud állítani, illetve a rendszer leállásakor fájlba tud tárolni, és újraindításkor vissza tud tölteni.



## 2.2. Modulok kezelése, állapotának figyelése

A core elindítja a modulokat, ha pedig azok meghalnának, akkor újraindítja őket. A modulokat bizonyos időközönként pingeli, ellenőrizve ezzel, hogy a modul tényleg fut-e. Ha a core adott időn belül nem kap választ a modultól, akkor újraindítja azt, így biztosítva az állandó rendelkezésre állást.

# 3. Modulok

A modulok csoportosítása:

- I/O-modulok
- logikai modulok
- emberi kapcsolattartást szolgáló modulok
- egyéb modulok

Minden modul csak egyfajta funkciót lát el, és a modulokat összekapcsolva hozunk létre bonyolultabb alrendszereket, vezérlő logikákat. Ez egyszerűbb fejlesztést és hibakeresést, rugalmasabb rendszert eredményez.

Az alábbiakban már elkészült modulokat ismertetünk:

## 3.1. I/O-modulok

Az I/O modulok biztosítják a hardverrel való direkt kommunikációt. A külvilágból érkező jeleket core változókba teszik be, illetve változókra érkező parancsokat hajtanak végre. Tevékenységük csak erre irányul, minden más funkciót logikai modulokban valósítunk meg.

### Esprit

A modul a Paradox Esprit riasztók soros vonalon való illesztését teszi lehetővé. A modul a soros vonalon keresztül olvassa a riasztóból érkező adatokat és ezt kiértékeli.

### La Crosse

A modul a La Crosse gyártmányú 23xx típusú meteorológiai állomások rendszerhez való illesztését végzi. Indításra kiolvassa az időjárásjellemzőket, és változókba rakja azokat.

#### MHS

Az mhs modul az MHS-2-ben fut, és lehetővé teszi, hogy a hardver bemeneteit, kimeneteit és a ledjeit vezérelni lehessen. Az external modullal együtt célszerű használni.

#### MOXA

A MOXA modul a MOXA cég ioLogic termékcsaládjának moduljait kezeli.

#### Sky

A sky programhoz tartozó modul, ami a Nap és a Hold mozgásához kapcsolódó csillagászati változók kalkulálását végzi (pl. a Nap látószöge). Ennek az a haszna, hogy amikor a Nap lemegy, akkor célszerű lehet leereszteni a redőnyöket, amikor felkel, akkor pedig felhúzni.

#### States

A states modul célja állapotváltoztatás és parancsküldés parancssorból. Le lehet kérdezni egy állapot értékét is, valamint ki lehet listázni az összes állapotot. A modulnak a parancssori párja a statesctl.

#### System

A system modul tulajdonképpen a states modul ellentéte. Események és parancsok érkezésekor lefuttat egy programot a gépen. Ennek gyakorlati haszna lehet például, hogy a számítógépen futó tv-alkalmazást lehet villanykapcsolóról vezérelni, újra lehet olvastatni a hyppolit konfigurációját, és persze egyéb elborult dolgokra is lehet használni.

#### TIxx3

A modul az ACT TI213 típusú X10 protokollvezérlő kezelését végzi.

## 3.2. Logikai modulok

A logikai modulok nincsenek direkt kapcsolatban a külvilággal, I/O igényüket az I/O-modulok szolgálják ki.

### Alarm

A mozgásérzékelőket kezelő modul, ami riasztásokra is képes. A mozgásérzékelőket csoportokba kell helyezni, ezután a csoportok külön-külön élesíthetőek.

### Average

Több változó értékeiből csúszóátlagot készít egy általa létrehozott változóba. Az átlagképzés mélysége a konfigurációban adható meg.

### Control

A control (vezérlő) modul klasszikus vezérlő funkciót képes megvalósítani. Egy bemeneti változó értékét a célérték megadott toleranciájú környezetébe igyekszik vezérelni. Ennek lehetséges felhasználása például nyáron a redőnyökkel történő fényerőszabályzás.

### CMQ

Command queue modul feladata, hogy egy változó elé fűzve szabályozza a változóra kimenő parancsok továbbítását.

### Cron

A cron modul periodikus időzítésre használatos.

### Delay

A Delay modul parancsokat ad ki változókra, és közéjük késleltetéseket tud berakni.

### Expr

Kifejezéseket és feltételeket kiértékelő modul. Egy kifejezés állapotokból és konstansokból állhat, amik közé matematikai műveleteket és relációkat lehet tenni. Jelenleg lebegőpontos számokkal és szövegekkel tud műveleteket végezni.

### Forward

A bejövő eseményeket és parancsokat továbbítja akár több helyre is, esemény vagy parancs formában. Az értéket megőrzi. Segítségével így állapotokat lehet átnevezni, ami akkor lehet hasznos, ha sok számmal azonosított kimenet van, aminek szöveges név is adható a modul segítségével.

**FSM**

Finite State Machine module. Lehetővé teszi, hogy egy véges állapotú automatát definiáljunk. Az automata különböző állapotokból épül fel. Minden egyes állapotban képes különböző parancsokat fogadni, aminek hatására egy másik állapotba megy, közben más parancsokat küldhet a többi állapotnak.

**Lua**

A Lua modul célja, hogy a Hyppolit rendszert lua nyelvű szkriptekkel lehessen bővíteni. A szkriptek hozzáférnek mindenhez, amihez egy sima modul is: lekérdezhetnek állapotokat, várhatnak eseményekre, állíthatnak be saját állapotokat, fogadhatnak ezekre érkező parancsokat.

**Mem**

Egy úgynevezett memóriamodul, ami létrehoz állapotokat, majd ha az állapotra parancs érkezik, akkor az állapot értékét a parancs tartalmára változtatja meg.

**Sched**

Scheduler module. Lehetővé teszi, hogy időzítve parancsokat hajthassunk végre. A konfigurációban definiálhatunk schedule group-okat. Minden group tartalmazhat schedule set-eket. Egy időben egy set van érvényben. Induláskor a konfigban megadott default set lesz érvényben. Váltani parancs küldésével lehet.

**Seq**

Sequencer module. Lehetővé teszi, hogy relatív időzített parancsokat hajthassunk végre meghatározott sorrendben. A konfigurációban definiálhatunk schedule group-okat. Minden group tartalmazhat sequencer set-eket. Egy időben egy set van érvényben. Induláskor a konfigban megadott default set lesz aktív. Váltani parancs küldésével lehet.

**Shutter**

Redőnyvezérlő modul. A redőnyt automatikusan kikapcsolja, ha le- vagy felért. Mindig nyilvántartja, hogy a redőny éppen hol jár, emiatt akár pozicionálni is lehet.

### Switch

A switch modul célja a kapcsolók megnyomásának feldolgozása. Egy kapcsoló alapvetően ki van kapcsolva, majd valaki megnyomja őket, ekkor végig be vannak kapcsolva, majd végül elengedésre kerülnek, amikor megint ki lesznek kapcsolva. Minden hagyományos kapcsoló a gombnyomás ideje alatt rövidre zárja az áramkört. Ez alól egyedül a mozgásérzékelők képeznek kivételt, hiszen azok éppen akkor vannak bekapcsolva, amikor nincs mozgás, hiszen ha valaki elvágja a kábelt, akkor megszakad az áramkör.

A modult többféleképpen be lehet állítani. Meg lehet adni, hogy a gombnyomáskor, vagy az elengedéskor kapcsoljon. Ha elengedéskor kapcsol, akkor adott a gombnyomás időtartalma, így különbséget lehet tenni a gombnyomások között időtartam alapján.

### Thermo

A hűtést és a fűtést vezérlő modul. A lakásokra jellemző, hogy egy vagy több fűtéskör található bennük, amelyeken tetszőleges számú fűtőtestek találhatóak. Egy körön egy időben nyilván csak fűteni vagy hűteni lehet, de ha a radiátorok külön vezérelhetőek, és minden szobában van hőmérő, akkor szobánként különböző hőmérsékletet lehet beállítani. Tehát egy szobához tartozik egy hőmérséklet és valahány fűtőtest. Egy fűtőtest csak és kizárólag egy szobához tartozhat.

## 3.3. Emberi kapcsolattartás modulok

### Config

A Hyppolit konfigurációsállomány-olvasó modulja, aminek segítségével az összes modul megkapja a beállításait.

### Log

A log modul felelős a többi modul logjainak kezeléséért. A modulok minden üzenetet a log állapotra küldenek, amit a H-core továbbít a log modulnak, az pedig a log konfigurációtól függően kezeli őket (pl. fájban tárolja).

### UI (User Interface)

A Hyppolithoz tartozó felhasználói felületek célja a különböző eszközök állapotának lekérdezése, valamint egy esetleges beavatkozás elvégzése. Többféle felhasználói felület létezik, mindegyikhez különböző menürendszer definiálható. A felületek önálló alkalmazások, amik ezen a modulon keresztül csatlakoznak. Az

ui modulon keresztül mindent el lehet végezni, amit a states modulon keresztül is. Azonban annyi különbség akad, hogy az ui modulon keresztül csak azokhoz az állapotokhoz lehet hozzáférni, amelyek a konfigurációban meg vannak adva, továbbá csak meghatározott parancsokat lehet küldeni.

Az UI-modulhoz különböző felhasználói felületek kapcsolódhatnak:

- hamcli: egy parancssori kliens
- hamwww: egy webes kliens
- backend: a további webes kliensek kiszolgálója
- hammobil: egy mobil telefonos kliens

## 3.4. Egyéb modulok

**At**

Belső ütemező modul.

**External**

Az external programnak a modul párja. Lehetővé teszi, hogy egy távoli gépen futtassunk egy modult. A távoli gépen kell futnia az external programnak. A modul csak egy távoli géphez képes csatlakozni, több külső modul esetén több példányban kell lefuttatni. A programnak nincs konfigurációja, mindent a parancssorból kap meg.

# 4. Fejlesztési irányok

A közeljövőben a rendszert az alábbi irányokba tervezzük továbbfejleszteni:

- Magas rendelkezésre állás: osztott core, redundáns elemek
- Wireless megoldások támogatásának bővítése: zigbee, zwave
- Szórakoztató elektronikai funkcionalitás

# 5. Felhasználási lehetőségek

Az alábbiakban különböző felhasználási lehetőségeket villantunk fel a teljesség igénye nélkül. Ezek egy része már megvalósított funkcionalitás, másik része elvi szinten „benne van” a rendszerben, a megfelelő irányban történő továbbfejlesztéssel megvalósítható.

## 5.1. Műszaki felügyelet, vagyonvédelem

A felhasználó mobiltelefonja vagy számítógépe segítségével rendszeres időközönként vagy szúrópróbaszerűen betekinthet otthona helyiségeibe, megnézheti és döntése szerint módosíthatja a beépített szabályozó által biztosított belső hőmérsékletet, ellenőrizheti a páratartalmat, a füst- és tűzjelzők állapotát, de még a fürdőszoba padlójának nedvességét is.

A bejáratot megfigyelő videókamera képét a rendszer a felhasználó televíziója, számítógépe képernyőjére, mobiltelefonja kijelzőjére juttathatja, így a felhasználó azonnal láthatja, ha valaki feltűnik bejárata előtt. Távollétében a rendszer ilyen eseményről képüzenetet küldhet mobiltelefonjára, így azonnal láthatja, ki érkezett. Ismeretlen vagy gyanús látogató érkezése esetén mobiltelefonjáról megszólaltathatja kaputelefonját, úgy beszélhet látogatójával, mintha otthon lenne. Az idegent akkor is megszólíthatja a kaputelefonon át, ha ő nem jelentkezik be, hanem csak a bejárat előtt van. Hívatlan vendégét fogadhatja a lakásból hallatszó kutyaugatással.

Vagyonvédelmi szempontból fontos lehet az otthonlét szimulálása. A kívánt időszakban az automatika a világítás változtatásával, a szórakoztató elektronikai berendezések a felhasználó szokásainak megfelelő vezérlésével a jelenlét látszatát keltve riaszthatja el a betörőket.

Megoldható az időszakos zárfeloldás, beléptetés. Hetente a szokott időpontban a takarítónő a megbeszélt kód, ujjlenyomat, RFID eszköz stb. segítségével bejuthat a lakásba, máskor nem. Belépése és tevékenysége WEB kamerával figyelhető. Az is megoldható, hogy bejárónőjét a kaputelefonon való bejelentkezésekor a felhasználó mobiltelefonjáról küldött paranccsal ereszthesse be bárhonnan és bármikor. A naplózó funkció segítséget nyújt események visszakövetésére.

Lehetőség van eseményfüggő riasztásra. A műszaki felügyelet beállított paramétereinek határérték-túllépéséről a rendszer tetszőlegesen választható módon értesíthet, pl. SMS-ben küldi a felhasználó mobiltelefonjára az esemény pontos és értelmes közlésével. (Pl. hőmérséklet indokolatlanul alacsony, baj lehet a kazánnal.)

Az otthonvédelem céljából telepített berendezések riasztásait a rendszer SMS-ben továbbíthatja akár a rendőrségre is. A riasztás észlelésekor automatikusan elindulhat a WEB térfigyelő rendszer, amely az észlelt képeket rögzíti és továbbítja.

## 5.2. Otthonautomatizálás

A ház külső és belső világítása az igények változása szerint tervezhető és működtethető. Példa: egyik helyiségből a másikba jutva villanykapcsolója két gyors egymás utáni billentésével a világítás felkapcsolására az elhagyott helyiség világítása lekapcsolódik, ha az infraérzékelő nem talál ott senkit. Azon helyiségekben,

ahol nem érzékel mozgást az érzékelő, lekapcsolja a világítást. Ha a felhasználó mégis ott tartózkodik, karja felemelését követően újra bekapcsol a világítás.

A lakásban elhelyezett hagyományos villanykapcsolók funkcionalitása nem csak az adott helyiség világításának ki- és bekapcsolására terjedhet ki, hanem a kapcsolók teljesen más funkciót is elláthatnak. Ha például a beteg családtag az ágya melletti villanykapcsolót hosszan lenyomja, a hálószobában hívójel hallatszik. A beteg felépülése után a funkció átprogramozással megszüntethető.

Beállítható, hogy az éjszakai világítás fényereje szemkímélő módon érje el a maximumot, mivel rendszerünk tudja, hogy éjjel van. Hangulatvilágítások, partyfényüzemmód akár mobiltelefonról is kapcsolható.

A redőnyök mozgatása időzítés vagy egyéb vizsgált és tetszőlegesen beállítható paraméterek szerint történhet. Például a beállított sötétedési szint elérésekor a rendszer leereszti a redőnyt, felhúzza, ha reggel 7 óra van hétköznap, vagy ha meteorológiai érzékelő erős északi szelet jelez, de akkor is, ha a külső hőmérséklet –5 Celsius fok alá süllyed.

A belső hőmérséklet adott szinten tartása a fűtő- és hűtőrendszer és az árnyékolások optimális és gazdaságos működtetésével oldható meg. A helyiségek belső hőmérsékletének mérésével, a meteorológiai érzékelők által mért külső hőmérséklet figyelembe vételével, az ablakok napsütöttségének szabályozásával, a szél élénkségével és irányával együttesen, a hűtő- és fűtőberendezések működésének szabályozásával és a redőnyök mozgatásával lehet elérni a kívánt hatást. Lehetőség van arra is, hogy a konyha mindig hűvösebb legyen, a redőny csak akkor záródjon, ha az érzékelő szerint senki sincs a helyiségben. Ha a felhasználó a szokásos időpontnál előbb érkezik haza, autójából mobiltelefon vagy iphone használatával vezérelheti fűtését.

A pince és az éléskamra hőmérsékletét nyári időszakban a külső, a pincében és a kamrában mért hőmérséklettől és páratartalomtól függően önműködően üzembe lépő levegőcserével szabályozhatja a rendszer.

A felhasználó vezérelhető háztartási berendezéseit előre programozottan használhatja, elindíthatja a bekészített mosást, mosogatást mobiljáról, hogy a program hazaérkezésekor érjen véget. Tetszés szerinti időpontban elindíthatja robotporszívóját, ha értesül anyósa látogatási szándékáról. Lehetőség van a konnektorok vezérlésére, így távolról bekapcsolhatja nem vezérelhető eszközeit, de ellenőrizheti azt is, lekapcsolta-e a vasalót, a hajsütővasat. A rendszer időjárásfigyelő és -előrejelző berendezéssel integráltan működik, így a kert locsolása szabályozható: a rendszer a hőmérséklet, a napsütés és az adott időszakon belül lehullott csapadék mennyiségének mérési adatait felhasználva a beállított locsolási időpontban a locsolás időtartamát automata módon szabályozhatja, eldöntheti, szükséges-e reggel és este is locsolni.

A felhasználó felügyelheti és optimalizálhatja energiafogyasztását. Háztartási eszközeinek üzemét a legalacsonyabb tarifákhoz igazíthatja. A fűtést csak akkor

kell bekapcsolni, amikor tényleg szükség van rá. Ha a felhasználó a szokásostól eltérő időpontban érkezik haza, mobiltelefonja vagy iphone használatával vezérelheti fűtését, a kerti locsolást, szórakoztatóelektronikai vagy háztartási eszközeit.

A fogyasztásmérők állása rendszeresen ellenőrizhető a rendszer megfelelő kijelzőin (nem kell pl. lemászni a vízmérő aknájába a vízfogyasztás leolvasásához), a mérési eredmények tárolhatóak, és grafikonokon is megjeleníthetők. Így a felhasználó észreveheti a mérőberendezések meghibásodását, de még arról is kaphat automatikus figyelmeztetést, ha a víz-, gáz- vagy elektromosenergia-fogyasztása a szokásos érték fölé emelkedett, hirtelen megnövekedett – csőtörés, gázszivárgás vagy részleges elektromos zárlat miatt.

A felhasználó WEB-kamera és mikrofon segítségével megfigyelheti csecsemőjét, kisgyermekét a konyhából, főzés közben is. A babaágyba helyezett nedvességérzékelő jelzi a váratlan „túlcsordulást”, de figyelheti újszülöttje légzését is. Kisgyermeke szobájában hatástalaníthatja a konnektorokat, ilyen módon is megvédheti őt a balesettől.

# A Launchpad.net fejlesztői platform

Farkas Szilveszter

**Kivonat**

A Launchpad.net-et sokan csak úgy ismerik, mint az Ubuntu fejlesztését támogató webes rendszert, holott sokkal több annál. Mi sem bizonyítja ezt jobban, mint hogy egyre több ismert szabad szoftver projekt választja hátteréül (pl. Zope, MySQL, Inkscape).

## Tartalomjegyzék

**1. Történet** **30**

**2. Hibakövető rendszer** **30**

**3. Forráskódtároló** **31**

**4. Fordítások** **31**

**5. Kérdések és válaszok** **32**

**6. Specifikációk** **32**

**7. Bináris csomagok** **33**

**8. Általános lehetőségek** **33**

**9. Nyílt adatok** **34**

**10. További információk** **34**

# 1. Történet

A szolgáltatás akkor vált szélesebb körben ismertté, amikor 2006 januárjában az Ubuntu disztribúció addigi Bugzilla alapú hibakövető rendszerének hibajegyeit migrálták az akkoriban még Malone kódnéven futó Launchpad komponensbe. Akkor még teljesen zárt volt a rendszer, azonban a szabad szoftveres közösség hosszú éveken át tartó jogos kritikái után 2008. július 22-én az OSCON konferencián Mark Shuttleworth (az Ubuntu Linux és a Launchpad fejlesztését is támogató Canonical alapítója) bejelentette, hogy egy éven belül megnyitják a platform forráskódját két komponens kivételével. Ezt az ígéretet a legutolsó lehetséges időpontban, 2009. július 21-én valóra is váltották, ráadásul az előzetes tervekkel ellentétben a teljes kódbázist megnyitották a GNU Affero GPL 3-as verziójú licence alatt.

# 2. Hibakövető rendszer

A hagyományos funkciókon kívül, mint például státuszinformáció, duplikátumok követése vagy hozzászólás, magáénak mondhat néhány igazán különleges tulajdonságot is a Launchpad „bug tracker” komponense. Az egyik ilyen annak a lehetősége, hogy egy hibát több különböző projekthez is hozzárendelhetünk, projektalapon szétválasztva a hiba státuszának és az egyéb metaadatoknak a kezelését. Ez nagyon hasznos tud lenni olyan esetekben, amikor egy végfelhasználói alkalmazásban talált hibáról kiderül, hogy a program által használt valamelyik függvénykönyvtár (vagy interpreter vagy akár a rendszermag maga) okozza a problémát. Ha az alsóbb rétegben található alkotóelem hibáját kijavították, akkor arról a közös hibajegyen keresztül értesülnek a többiek (pl. az eredeti alkalmazás fejlesztői), így ők is megtehetik a szükséges lépéseket erre a munkára építve.

Az előbbiekben vázolt folyamat csak akkor működhet kifogástalanul, ha mindegyik projekt aktívan használja a Launchpad szolgáltatását. Természetesen ez a valóságban nem így van, hiszen rengeteg projektnek saját hibakövető rendszere van. Erre is van megoldás, amely alapesetben egy távoli rendszer hibajegyének figyelését jelenti (pl. lezárja a hibát Launchpadben, ha azt a távolban megtették).

A külső hibakezelő karbantartóinak segítségével tovább javítható a kommunikáció egészen a kétirányú szinkronizációig, elsősorban a hozzászólások szintjén. Erre egy nagyon jó példa a következő: a felhasználó egy disztribúció használata során talált hibát nagyobb valószínűséggel jelenti be az adott terjesztés hibakövető rendszerében, mint az adott komponens (pl. GNOME) fejlesztőinél. A disztribúció készítői össze tudják kapcsolni a felhasználót és a fejlesztőt úgy, hogy a gazdaprojekt rendszerében található hibajegyet hozzárendelik a Launchpadben lévő megfelelőjéhez. Ezután a fejlesztő minden hozzászólása és státuszmódosí-

tása, amelyet a saját rendszerében végez, látszódni fog a Launchpaden, illetve a felhasználó további megjegyzései pedig a gazdaprojektnél. Mindehhez csak egy bővítményt kell telepíteni a távoli hibakövető rendszerre. Jelenleg ily módon támogatottak a Bugzilla és a Trac rendszerek.

# 3. Forráskódtároló

Ezen komponens magját a Bazaar elosztott verziókezelő rendszer adja, amely önmagában megérne egy előadást. Általánosságban elmondható a hasonló verziókövetőkről, hogy az elágazást („branch”) és a beolvasztást („merge”) segítik leginkább – nyílt forráskódú rendszereknél ezek a leggyakrabban használt műveletek.

Jó hír, hogy azok is kihasználhatják a Launchpad által nyújtott előnyöket, akik a kódjukat nem a fent említett rendszer segítségével kezelik: létező CVS, vagy Subversion tárolókat lehet egy az egyben importálni teljes kódtörténettel. A verziókezelők piacán tapasztalható éles verseny hatására pedig már kísérleti jelleggel támogatott a Git tárolók importálása, illetve már fejlesztés alatt áll egy olyan modul, amellyel a Mercurialben (hg) tárolt forráskódokat lehet a Launchpad-Bazaar párossal kezelni.

Rendkívül könnyen és gyorsan lehet egy-egy projekt fejlesztésébe beszállni: egyszerűen csak egy új ágat kell létrehozni, amelybe rögzíthetők a változtatások. Ezután egy már meglévő hibajegyhez lehet kapcsolni az ágat, mint az adott probléma megoldását, vagy akár kérhető a kód beolvasztása a fejlesztőktől („merge request”). Ilyenkor egy külön erre a célra kialakított felületen vizsgálják az adott módosítást („code review”). A már említett lehetőségeken kívül még egy kényelmes felületű kódböngésző is segíti a munkát (érdekes módon ez az eszköz eredetileg is szabad szoftver volt – a Loggerhead névre rákeresve bárki telepíthet a saját Bazaar tárolóihoz is kódböngészőt).

# 4. Fordítások

A Rosetta kódnevű modul elsődleges célja indulásakor az Ubuntuban hivatalosan támogatott szoftverek weben keresztüli fordításának segítése volt. Azóta már bármely regisztrált projekt használhatja saját fordítóközösségének felvirágoztatásához. Az alapvető koncepció az, hogy bárki tehet javaslatokat a különböző szoftverek felületén megjelenő szövegek anyanyelvi megfelelőjére. Innentől pedig a projekt vezetőire van bízva, hogy az adott fordítás rögtön elfogadásra kerül (nyílt rendszer), vagy bizonyos megbízható emberek jóváhagyása szükségeltetik ehhez (részben korlátozott). Lehetőség van arra is, hogy kizárólag előre kijelölt emberek javasolhassanak fordításokat, de ennek nyílt forráskódú szoftverek esetén nincs

sok értelme.

A webes felületen az adott kifejezés fordításakor segítséget is kapunk, nevezetesen ha már korábban előfordult az adott szóösszetétel valamely szabad szoftverben, és annak van anyanyelvi megfelelője, akkor egy kattintással be tudjuk azt illeszteni. Ezáltal biztosítható a projekteken belüli, illetve azok közötti konzisztens nyelvezet.

További lehetőség, hogy a fordításokat tartalmazó fájlokat letölthetjük, így azonnal tesztelni tudjuk a változtatásainkat. Elnézést, az „azonnal” kicsit túlzás, hiszen egyelőre úgy működik a dolog, hogy kérvényeznünk kell a letöltést, és csak bizonyos (terheléstől függ, de jellemzően néhány perc) idő után kapjuk meg e-mailben a fájlra mutató hivatkozást.

Sokan teljesen jogosan bírálják a Launchpad fordító szolgáltatását amiatt, hogy részben a fent is említett folyamatból kifolyólag elég nehézkes a fordítások visszajuttatása azon gazdaprojektekhez, amelyek a saját rendszereiket használják a fordítások kezeléséhez. Ezt kétféle módon is szeretnék orvosolni a jövőben: egyrészt teljes állásban alkalmaz a fejlesztést támogató cég egy fordítói közösségekért felelős embert (elsősorban az Ubuntu disztribúcióhoz kapcsolódva), továbbá a következő nagy kiadási ciklus legfontosabb céljaként tűzte ki ennek a rendkívül fontos együttműködési lehetőségnek a fejlesztését.

# 5. Kérdések és válaszok

Az Answers nevű modul működése rendkívül egyszerű: ha engedélyezve van a projekt vezetői által, akkor a felhasználók kérdéseket tehetnek fel, amelyekre bárki válaszolhat, egyfajta közösségi támogatást nyújtva. Lehetőség van a kérdések állapotának követésére is (hasonlóan a hibakezelőhöz), de leghasznosabb funkciója az, hogy a gyakran feltett kérdések alapján egy tudásbázist tudunk építeni. Érdekes lehet még, hogy a felhasználók saját nyelvükön is feltehetnek kérdéseket abban a reményben, hogy olvassák azt honfitárasaik, akik szívesen segítenek a problémák megoldásában.

# 6. Specifikációk

A Launchpad nem a hagyományos értelemben vett, több oldalas specifikációkat támogatja, hanem az ún. „blueprint”-eket. Ezek leginkább különböző feladatok pár mondatos leírásai vagy egyszerű ötletek, amelyek kidolgozásra, megvalósításra várnak. Mélyen beépül a rendszer különböző részeibe, így például hibajegyeket és kód ágakat tudunk hozzárendelni egy-egy specifikációhoz.

# 7. Bináris csomagok

Ezen modul nevéből (Soyuz) arra lehet következtetni, hogy a rendszer lelkéről van szó. Nagyot nem is tévednénk, hiszen hosszú ideig csak az Ubuntu, illetve néhány leszármazott disztribúció élvezhette a modul csomagkezelési képességeit. Manapság is így van ez, azzal az apró különbséggel, hogy most már bárki közzétehet bármilyen szoftvert olyan formában, hogy azt bárki egyszerűen telepíteni tudja Ubuntu alapú rendszerére.

A gyakorlatban ez úgy néz ki, hogy felhasználók és csapatok is létre tudnak hozni szoftvertárolókat („repository”), ahová bármilyen szoftver forrását feltölthetik a csomagolási információkkal kiegészítve. Ezekből egy szerverfarm segítségével belátható időn belül (terheléstől függően, saját tapasztalatom alapján átlagosan 10-15 perc alatt) bináris csomagok készülnek több architektúrára (jelenleg támogatottak: 32 bites PC, 64 bites PC, LPIA mobil platform) és akár több Ubuntu verzióra is.

Ilyen egyedi tárolókat többféleképpen is lehet hasznosítani. Nagyszerű lehetőség olyan új szoftverek terjesztésére, amelyek a legfrissebb stabil Ubuntu kiadásban sem érhetők még el (erre jó példa az Ubuntu 9.04 kiadása után megjelent Ubuntu One szolgáltatás béta kliense), vagy csak régebbi változat áll rendelkezésre a hivatalos forrásokból. Többen olyan tárlókat tartanak karban, amelyek bizonyos alkalmazások legfrissebb buildjét („nightly”) tartalmazzák (pl. Chromium böngésző).

# 8. Általános lehetőségek

A Launchpad mindent átfogóan három különböző egységet használ: felhasználók, csapatok és projektek.

A felhasználókról annyit érdemes tudni, hogy bárki regisztrálhat a rendszerbe. Motivációs céllal vezették be a karmát: az oldalon végzett bármilyen tevékenységért (fordítási javaslattétel, javaslat jóváhagyása, hibajelentés, specifikáció módosítása stb.) karmapontot kapunk, amelynek mértéke változó (kevésbé népszerű cselekvésért több pont jár). Igazából nemcsak a pontozás miatt hasznos ez az egész, hanem azért is, mert így részletes képet kaphatunk egy felhasználó tevékenységéről.

A csapatoknak a következők közül kell típust választaniuk: moderált (bárki jelentkezhet, de adminisztrátori jóváhagyással lehet csak csatlakozni), nyílt (bárki szabadon csatlakozhat) vagy korlátozott (csak az adminisztrátor adhat hozzá új csapattagokat). Ha valakinek van egy csapata, akkor igényelhet hozzá levelező listát (gyakorlatilag a Mailmant integrálták a Launchpadbe). A projektek a szokásos adatok tárolása mellett biztosítják a különböző, párhuzamos fejlesztési ágak,

de igazából a teljes kiadási folyamat kezelését. Jelenleg ez a következő fogalmak köré csoportosul: „series” (fejlesztési ág, pl. stabil, fejlesztés alatti, még támogatott), „milestone” (mérföldkövek egy-egy kiadás fejlesztése során) és „release” (tényleges kiadás). Legutóbbihoz tárhelyet is kapunk, ahová feltölthetjük a forráskódot tartalmazó csomagfájlt.

A Launchpad OpenID szolgáltatóként is működik (fogyasztóként egyelőre még nem), tehát a felhasználói fiókunk segítségével bármikor regisztrálhatunk, majd bejelentkezhetünk olyan szolgáltatásokba, amelyek elfogadják az OpenID-t, mint hitelesítési formát.

# 9. Nyílt adatok

Még a forráskód megnyitása előtt elindult a törekvés a szabadon hozzáférhető adatok biztosítására. Régebben a legegyszerűbb feladatok automatizálásához is olyan programokat kellett írni, amelyek a webalkalmazás által megjelenített oldalak forrásából gyűjtötték az információkat. Ennek az állapotnak az API megjelenése vetett véget, igaz, egyelőre csak részben, mivel béta fázisban van a fejlesztése, és nem támogat teljes körűen minden komponenst.

Python programozók előnyben vannak, hiszen egy remek kis modul érhető el launchpadlib néven LGPL licenc alatt, amellyel objektumorientáltan lehet a rendelkezésre álló erőforrásokat kezelni.

# 10. További információk

- Launchpad.net kezdőlap: `https://launchpad.net/`
- Körséta: `https://launchpad.net/+tour/index`
- Súgó (felhasználói dokumentáció): `https://help.launchpad.net/`
- Fejlesztői dokumentáció: `https://dev.launchpad.net/`

# Naplóelemzés – syslog-ng OSE

Höltzl Péter

**Kivonat**

Az informatikai rendszerek üzemeltetésének kiemelkedően fontos feladata naplók begyűjtése és elemzése. Előadásomban a syslog-ng OSE segítségével megvalósítható napló gyűjtési módszereket, protokollokat, előfeldolgozási valamint üzenet-osztályozási lehetőségeket szeretném bemutatni.

## Tartalomjegyzék

**1. Bevezető** **36**
1.1. A syslog-ng konfiguráció alapjai . . . . . 36

**2. Központi naplószerver és napló elemzés syslog-ng alapokon** **39**
2.1. A gyűjtés . . . . . 39
2.2. Formátum konverzió . . . . . 40
2.3. Klasszifikáció . . . . . 42
2.4. Tárolás és feldolgozás . . . . . 43

# 1. Bevezető

Az informatikai rendszerek üzemeltetésének kiemelkedően fontos feladata naplók begyűjtése és elemzése. A rendszer biztonsága érdekében a beérkezett naplókat rendszeresen értékelni kell, mely komoly kihívást jelent az üzemeltetők számára. Előadásomban a syslog-ng OSE segítségével megvalósítható napló gyűjtési módszereket, protokollokat, előfeldolgozási valamint üzenet-osztályozási lehetőségeket szeretném bemutatni. Célom egy könnyen használható, kizárólag GNU/GPL alapokra épülő naplóelemzési eljárás bemutatása életből vett példák segítségével.

## 1.1. A syslog-ng konfiguráció alapjai

**Source**: Nevesített definíciós objektumok, melyek meghatározzák a naplóüzenetek forrásait. A syslog-ng folyamatosan olvassa a beérkező üzeneteket, melyek a forrásokon jelennek meg.

**Source driver**: Azok a konkrét syslog-ng modulok, melyek meghatározott módokon képesek naplókat olvasni. A syslog-ng számos naplófogadási módot ismer (`internal`, `file`, `unix-socket`, `unix-stream`, `pipe`, `syslog`, `tcp/udp` és `ipv4/ipv6`, `program` stb.).

Példa napló fájl felolvasására:

```
source s_apache {
    file("/path/to/apache/var/log/apache/access.log");
};
```

**Destination**: Nevesített definíciós objektumok, melyek meghatározzák a napló tároló helyeinket. A syslog-ng a beérkezett naplókat a cél bufferébe helyezi el további tárolás céljából.

**Destination driver**: Azok a konkrét syslog-ng modulok, melyek meghatározott módokon képesek naplókat tárolni/továbbküldeni. A driverek a destination objektumok bufferét olvassák. A syslog-ng számos napló írási módot támogat (`file`, `unix-socket`, `unix-stream`, `pipe`, `syslog`, `tcp/udp` és `ipv4/ipv6`, `program` stb.).

Példa napló tárolásra `tcp` csatornán:

```
destination d_apache {
    tcp("1.2.3.4" port (514));
};
```

**Log path**: Nem nevesített definíciós objektum, mely összerendeli a forrásokat és a célokat, tehát egyfajta logroutingot végez:

```
log {
    source(s_apache);
    destination(d_apache);
};
```

Megjegyzések:

- Egyetlen source vagy destination objektum tartalmazhat több modult is. Lehetőleg viszont egy forrást egyszerre csak egy modullal olvassunk!
- A definíciós objektumok így használhatóak csoportosításra.
- Egy logpath tartalmazhat több forrást vagy célt is. Elágazások is lehetségesek.

**Filter**: Olyan nevesített definíciós objektum, melyek szűrő feltételeket határoznak meg. A syslog-ng számos szűrő funkciót képes alkalmazni a felismert protokoll elemekre (severity, priority, host, program és message rész).
Példa egy szűrő kifejezésre:

```
filter f_filter {
    program(postfix);
};
```

Példa egy szűrő használata a log pathban:

```
log {
    source(s_all);
    filter(f_filter);
    destination(d_mail_log);
};
```

Amennyiben ugyanazt a log forrást több célhoz rendeljük, a bejövő üzeneteinket megtöbbszörözhetjük:

```
log {
    source(s_apache);
    destination(d_apache_1);
};

log {
    source(s_apache);
    destination(d_apache_2);
};
```

A logpathban ezért flag-ek segítségével definiálható, hogy mely beérkező üzeneteket kezelje. A logpathok a következő flageket használják:

- final: a feldolgozott (pl. a szűrőknek megfelelő) üzeneteket ne küldje tovább.
- fallback: csak a fel nem dolgozott (pl. a szűrők által kiszűrt) üzeneteket küldje tovább.
- catchall: az összes forrásra beérkező minden üzenetet dolgozzon fel, a logpathok flagjeit figyelmen kívül hagyva.

```
log {
    source(s_apache);
    filter(f_filter_1);
    destination(d_apache_2);
};

log {
    source(s_apache);
    destination(d_apache_2);
    flags(fallback);
};
```

A logpathok összevonhatóak, de a flagek csak globáliasan értelmezettek:

```
log {
    source(s_apache);
    filter(f_filter_1);
    destination(d_apache_2);
    filter(f_filter_2);
    destination(d_apache_2);
};
```

Illetve a logpath-ok ún. subpathokat is tartalmazhatnak, amelyben mindkét subpath bemenete a destination d_apache_2 tartalma lesz:

```
log {
    source(s_apache);
    filter(f_filter_1);
    destination(d_apache_2);
    log {
        filter(f_filter_2);
        destination(d_apache_2);
    };
    log {
        filter(f_filter_3);
        destination(d_apache_2);
    };
};
```

Megjegyzések:

- A flagek csak a logpathok között érvényesek, a sub pathokra nem
- Egy hibásan konfigurált syslog-ng-vel akár üzenetvesztés is lehetséges

# 2. Központi naplószerver és napló elemzés syslog-ng alapokon

Példánkban egy elképzelt hálózat üzeneteit gyűjtjük össze syslog-ng OSE 3.0 segítségével. A rendszer Debian GNU/Linux operációs rendszert futtat, mely szolgálatatásait `chroot`-olt környezetben futó alkalmazások szolgálják ki. Az így szeparált alkalamazások naplóit a `chroot` környezeten kívül kell tárolni. A naplózó alrendszer a következő funkciókkal rendelkezik:

- Gyűjtés
- Formátum konverzió
- Klasszifikáció
- Tárolás és feldolgozás

A következő részekben ezeket a funkciókat részletezzük.

## 2.1. A gyűjtés

Első lépesben fel kell deríteni, hogy a telepített szerver alkalmazásai hová loggolnak. Alapvetően a `/dev/log`-ot használják a standard linux alkalmazások. Chrootolt környezetek esetében fel kell szedni az ott futó alkalmazások üzeneteit, tehát az ott található `/dev/log`-jait is.[1] Példa:

```
source s_all {
    internal();
    unix-stream("/dev/log");
    unix-stream("/path/to/chroots/chroot1/dev/log");
    filel("/proc/kmsg" program_override("kernel: "));
    pipe("/var/log/acpid" program_override("acpid: "));
};
```

Egyes alkalmazások nem a standard eszközöket használjak, hanem minden esetben napló fájlokat szeretnének írni. Ennek oka többféle lehet:

[1] Chrootok esetében nincs szükség a megfelelő log eszközök létrehozására mknod parancs segítségével, mivel azt a syslog-ng induláskor létrehozza.

- Nem tudják vagy nem akarják betartani az RFC-t
- A `/dev/log` eszközt ilyenkor egy másik alkalmazás olvassa fel. Egyes alkalmazások azonban nem szolgálnak ki klienseket, ha nem képesek napló üzenetet írni. Ezért ha a napló gyűjtő megállna, az az alkalmazás leállását is okozza, ami új kockázati tényezőt jelent.

Megoldás:

- Named pipe-ok használata (’`mknod pipe.log p`’) ezeken a helyeken.
- Fájl olvasása[2]

A szerver hálózaton kereszül szintén fogad napló üzeneteket. Ehhez szerver oldalon beállítjuk az üzenet fogadást. Szerverünk támogatja az RFC3164 valamint az RFC5424-es RFC5425 felett. Ehhez a következő forrást kell beállítanunk:

```
source s_net {
    #RFC3165
    tcp(ip(0.0.0.0) port(514)
        tls(
        ca_dir(/opt/syslog_ng/etc/ssl/)
        key_file(/opt/syslog_ng/etc/ssl/server.key)
        cert_file(/opt/syslog_ng/etc/ssl/server.crt)
        peer_verify(requred-trusted))
    );
    #RFC5424
    syslog(ip(0.0.0.0) port(6514)
        transport("tls")
        tls (
        ca_dir('/opt/syslog_ng/etc/syslog-ng/keys/ca.d/')
        key_file('/opt/syslog_ng/etc/ssl/server.key')
        cert_file('/opt/syslog_ng/etc/ssl/server.crt')
        peer-verify(required-trusted)
        )
    );
};
```

## 2.2. Formátum konverzió

A nem szabványos syslog üzeneteket egységes syslog formára kell hozni. Amennyiben a telepített alkalmazásokat szeretnénk naplózni, akkor a Debian `dpkg` parancsa által készített üzeneteket kell összegyűjteni. A `dpkg` nem tartja be az RFC-t és a következő üzeneteket készíti:

[2]Ebben az esetben, pl. egy chroot környezet esetében külön figyelmet kell fordítani a naplók forgatására (lást. a logrotate helyes konfigurációja)

```
2009-07-02 10:18:23 install libjinglexmllite0.3-0 0.3.12-3ubuntu1
```

Az ilyen beérkezett üzeneteket a syslog-ng parser funkciója segítségével a megfelelő formátumra hozhatjuk.

**Parser**: olyan syslog-ng modul, mely a beérkezett üzenetrészeket valamilyen modul segítségével értelmezi és az értelmezett részből makrókat készít, melyeket felhasználva saját sablonokkal a kívánt alakra hozhatjuk üzeneteinket.
A parsereket a megfelelő nevesített parser definíciós objektum segítségével lehet konfigurálni.

**CSV-Parser**: a bejövő struktúrált üzeneteket képes CSV (comma-separated value) értelmezni és az egyes oszlopokra saját makrókat állítani:

```
parser p_dpkg {
    csv-parser(columns("DPKGMSG.HOUR", "DPKGMSG.MSG")
    delimiters(" ")
    flags(escape-none,greedy)
    template("${MSG}"));
};
```

A parser elkészíti a számunkra fontos `${DPKGMSG.MSG}` makrót, amit a megfelelő template segítségével standard syslog formátumra hozhatunk. Ebben a syslog-ng MAKRÓ-k és TEMPLATE-ek nyújtanak segítséget.

**Template**: olyan syslog-ng modul, mely a beérkezett üzenetek formázását definiálja (sablon).

**Makró**: olyan eszköz, mely a syslog-ng által feldolgozott üzenet elemeket és azok formátumát tartalmazza

```
template t_dpkg {
    template("$R_ISODATE $HOST $PROGRAM ${DPKGMSG.MSG}\n");
    template-escape(no);
};
```

Majd a beérkezett üzeneteket a syslog-ba írjuk:

```
destination d_dpkg_messages {
    file("/var/log/messages" template(t_dpkg));
};
log {
    source(s_dpkg);
    parser(p_dpkg);
    destination(d_dpkg_messages);
};
```

## 2.3. Klasszifikáció

A beérkezett naplóüzenetek elemzése az üzemeltetés során keletkező üzenetek rendszeres áttekintése miatt fontos:

- Hibakezelés, az üzemszerű működési anomáliák felderítése és a lehető leggyorsabb reakció miatt.
- Rossz szándékú tevékenységek felderítése.

A naplóüzenetek áttekintésére számos alkalmazás használható. Cél:

- Megfelelés a biztonsági szabályzatnak (IBSZ).
- Megfelelés az auditnak vagy a szabályzásnak.
- Hibakeresés elősegítése.
- Nyomok rögzítése.
- Válasz a biztonsági eseményekre.

Jelen előadás célja a hibakeresés és a biztonsági események megtalálásának elősegítése. A cél eléréséhez a naplóüzenetek osztályozását minta adatbázissal valósítjuk meg. A beérkező összes üzenetet egy klasszifikációs folyamat után szétválasztjuk, majd a kívánt kategória üzeneteit rendszeresen ellenőrizzük. A fel nem ismert üzenetek (amire semmilyen minta nem illeszkedik) az "unknown" kategóriát kapják, melynek segítségével a minta adatbázis folyamatosan bővíthető.
A **db-parser**: a beérkező üzeneteket egy minta adatbázis alapján osztályokba sorolja, melyekre egy `${.classifier.class}` MAKRÓ segítségével hivatkozhatunk. A minta adatbázis egy XML fa, mely a következő képpen néz ki:

```
<patterndb version='1' pub_date='2009-10-31'>
  <program name='PRG'>
    <pattern>PRG</pattern>
      <rule id='RULE-ID' class='CLASS'>
        <pattern>message pattern</pattern>
      </rule>
  <program name='sshd'>
</patterndb>
```

Az üzenetre a minta akkor illeszkedik, ha:

- A program pattern illeszkedik a PROGRAM mezőre
- A rule-ban található minta illeszkedik a MESSAGE mezőre

A rule-ban szereplő üzenet változó elemeiben a következő elemek használhatóak:

- `@STRING::@` egyszeű string
- `@QSTRING::@` idézett string
- `@ESTRING::@` bármely string egy stop karakterig
- `@NUMBER::@` szám
- `@IP::@` bármely IPv4 vagy IPv6
- `@IPv4::@` bármely IPv4
- `@IPv6::@` bármely IPv6

Egy példa minta az SSH üzenetek naplóira:

```
<program name='sshd'>
 <pattern>sshd</pattern>
  <rule id='ssh-1' class='system'>
   <pattern>Accepted publickey for @STRING@ from @IPv4@ port @NUMBER@ ssh2</pattern>
  </rule>
  <rule id='ssh-2' class='violation'>
   <pattern>Did not receive identification string from @IPv4@</pattern>
  </rule> <rule id='ssh-3' class='violation'>
   <pattern>reverse mapping checking getaddrinfo for @ESTRING:: @ [@IPv4@] failed - POSSIBLE BREAK-IN ATTEMPT!</pattern>
  </rule>
  <rule id='ssh-4' class='violation'>
   <pattern>Bad protocol version identification @QSTRING::'@ from @IPv4@</pattern>
  </rule>
</program>
```

A pattern-db használata a syslog-ng konfigurációban:

```
parser p_logcheck {
    db-parser(file("/opt/syslog\_ng/var/db/classes.db"));
};
destination d_logcheck {
    file("/var/log/logcheck/$HOST_${.classifier.class}.log" owner("root") group("adm") perm(0640));
};
log {
    source(s_all);
    source(s_net)
    parser(p_logcheck);
    destination(d_logcheck);
};
```

## 2.4. Tárolás és feldolgozás

Az előbbi példában a beérkezett üzeneteket különálló napló állományokba mentettük aszerint, hogy milyen hostról és milyen osztályokba soroltuk őket. Egy átlagos rendszeren érdemes a következő osztályokat használni, bár az itt felsoroltak meglehetősen szubjektív osztályok:

- **System**: a rendszerrel kapcsolatos üzenetek (pl. a `cron` üzenetei)

- **Error**: az egyes alkalmazásokkal kapcsolatos hibaüzenetek (pl. diszk telítődés)
- **Violation**: valamilyen szabályzat sértést tartalmazó üzenetek (pl. jelszó elrontás)
- **Unknown**: a már említett, még nem felismert üzenetek, melyekből mintákat kell készíteni a pattern db számára

Az így elkészített konfigurációval a syslog-ng a `/var/log/logcheck` könyvtárban `violation.log`, `error.log`, `system.log` valamint `unknown.log` napló fájlokat hoz létre. Végül egy egyszerű script segítségével a kívánt időközönként postázzuk a szervert üzemeltető rendszergazdának.[3]

[3]Linux alatt az utolsó módosítás után beérkezett sorokat a `logtail` nevű alkalmazás segítségével lehet egyszerűen leválogatni.

# iptables és ipset

Kadlecsik József
<kadlec@blackhole.kfki.hu>

**Kivonat**

Az **iptables**[1] nyújtotta rugalmasságot könnyen tudjuk ötvözni kiváló teljesítménnyel, ha tűzfal szabályainkat megfűszerezzük egy kis **ipset**[2]-el. Az előadásban a „miért” alapozást néhány „hogyan” recept követi, a gyakorlat felől bemutatva a megoldási lehetőségeket.

# Tartalomjegyzék

**1. Linearitás** **46**

**2. Halmazok** **48**

**3. ipset** **48**
3.1. map típusok . . . . . 49
3.2. hash típusok . . . . . 49
3.3. tree típusok . . . . . 50
3.4. setlist típus . . . . . 50

**4. Példák ipset használatára** **50**
4.1. daemonshield és ipset . . . . . 50
4.2. DNS és ipset . . . . . 51

**5. iptables és ipset** **53**

**6. Összefoglalás** **55**

# 1. Linearitás

A Linux/netfilter alapú tűzfalak építésénél az **iptables** az az eszköz, amellyel hálózatbiztonsági szabályzatunkat a tűzfalat megvalósító számítógép számára utasításkészletté fordítjuk le. A biztonsági szabályzat gyakorlati értelmezése során a következő **iptables** elemekkel fejezzük ki azt:

- a funkcionális alrendszereknek (*raw*, *mangle*, *nat*, *filter*) megfelelő táblázatok;
- a táblázat alapértelmezett (táblázattól függően: PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING), valamint az adott táblázat számára általunk definiált szabályláncok;
- a láncokban a szabályok;
- a szabályokban a csomagegyezések (*match*) és az azokat kielégítő csomagokra vonatkozó döntés (*target*);
- a szabályok kiértékelését optimalizálhatjuk „jump” és „goto” döntésekkel.

A netfilternek minden egyes csomagnál meg kell vizsgálnia, hogy a definiált táblázatokban mely szabálynál talál egyezést, és az milyen döntést jelent az adott csomagra nézve. Egy-egy táblázat kiértékelése során a láncokban a szabályok logikai VAGY kapcsolatokként értelmezendők („a csomag egyezik-e a 0. szabállyal, vagy az 1. szabállyal, vagy ...”), míg az egyes szabályokban az egyezések ÉS kapcsolatokat jelentenek („a csomag kielégíti-e a szabály 0. egyezését, és az 1. egyezését, és ...”).

A netfilter a kiértékelés során mind a láncokban (szabályok léptetése), mind a szabályokban (egyezések léptetése) lineáris feldogozást végez. Ez természetesen a teljesítményre jelentős befolyással van: ha magas a szabályok száma, akkor a legutolsó szabállyal egyező csomagnak „hiábavalóan” be kell járnia az összes megelőző szabályt/láncot.

A láncok közti VAGY kapcsolat miatt nem diszjunkt egyezések esetén a sorrendiség érdekes és igen fontos kérdés. Ha nincs más egyezési feltétel *és* a szabályok végén a döntések eltérőek, akkor például csak a

```
-A lanc -s 192.168.0.1    -j ...
-A lanc -s 192.168.0.0/24 -j ...
```

sorrendnek van értelme, a fordítottjának nincsen. Viszont ha a a döntés mindkét szabályban ugyanaz, akkor az első szabály fölösleges, elegendő a második.

Diszjunkt egyezések esetén a sorrendiség tisztán logikai szempontból érdektelen, a

```
-A lanc -s 192.168.0.1    -j ...
-A lanc -s 192.168.1.0/24 -j ...
```

szabályokat írhatjuk felcserélt sorrendben – de *teljesítmény* szempontból a felcserélés akár jelentős eltérést is okozhat.

Egy lényeges megállapítást tehetünk a fentiek alapján: ha a szabályok sorrendiségének nem lenne a teljesítményre ható következménye, akkor könnyedén tehetnénk tetszőleges, akár grafikus, akár karakteres, egy pszeudo-nyelvet interpretáló interfészt az **iptables** fölé, amelynél csak a szabályokat kellene specifikálnunk

```
ha IP cím <...>
   és célport <...>
   és ...
akkor ELDOB|BEENGED|...
```

és a szabályoknak a tényleges megvalósítását, láncokba, táblázatokba való fűzését az interfész végezné el. Vegyük észre, hogy ez mennyire közel van egy biztonsági szabályzat tételes felsorolásához:

```
<...> szerver
   <...> szolgáltatást
   ....
nyújt
```

*Ha* nem kellene a teljesítmény miatt aggódnunk, a szabálygyártást egészen magas szinten automatizálhatnánk, egyszerűsíthetnénk, tehetnénk kényelmessé.

Természetesen ezen a szinten is létezik megoldás a teljesítményt részben figyelembe vevő egyszerűsítésre. A gyakorlatból tudjuk, hogy a magas szabályszám tipikusan a magas szerver/kliens számból és az azokra vonatkozó egyedi szabályokból fakad. Készíthetünk olyan interfészt, amelyik az összes szerver/kliens szabály ismeretében optimalizált láncrendszert generál, például:

```
-N 192/8
-N 192.168/16
-N 192.168.0/24
-A -s 192.0.0.0/8 -j 192/8
-A 192/8 -s 192.168.0.0/16 -j 192.168/16
-A 192.168/16 -s 192.168.0.0/24 -j 192.168.0/24
-A 192.168.0/24 -s 192.168.0.1 -j LOG-ACCEPT
....
```

Ha az összes IP címet fel kellene sorolnuk a 192.0.0.0/8 hálózatból, akkor ezzel bármely IP címet maximálisan 768 szabályon keresztül érnénk el a legrosszabb esetbeni 16 777 216 szabály helyett. Az interfész optimalizálható, hogy csak akkor hozzon létre alláncot, ha arra szükség van (azaz a nem használt blokkok kihagyhatók).

Egy ilyen interfész elkészítése szép feladat! Nem igazán elegáns a rengeteg lánc, és nem elhanyagolható tényező, hogy a tűzfal teljesítménye még így is romlik $\sim$20%-al[3].

# 2. Halmazok

Ha folytatjuk az előző fejezet feltevését – a magas szabályszám oka a magas kliens/szerverre vonatkozó egyedi szabály –, és még kikötjük azt is, hogy ezeknél a szabályoknál a döntés (*target*) ugyanaz, akkor a problémát átfogalmazhatjuk egy egyszerű halmazelméleti kérdéssé: adott csomag benne van-e, kielégíti-e a szabályok által meghatározott feltételek halmazát?

Fogalmazzuk meg a halmazok használatának pontos peremfeltételeit: ha

- a szabályok ugyanazon típusú egyezést vizsgálnak, és
- a szabályoknál a döntés ugyanaz,

akkor egy halmazra való egyezés keresésére redukálhatjuk az előző fejezet végén megadott komplex szabálylláncban való keresést:

```
-A -m set --match-set halmaz,src -j LOG-ACCEPT
```

Vegyük észre, mivel a szabályoknál a döntés ugyanaz, ezért a halmaz elemei közti keresésben nincs sorrendiségi kérdés.

Szó szerint szabályok és láncok ezreit, tízezreit tudjuk helyettesíteni egyetlen **iptables** szabállyal – ezt nyújtja nekünk az **ipset**.

# 3. ipset

Az **ipset** a halmazokban való egyezés keresését teszi lehetővé számunkra **iptables** szabályokból, ami nagyfokú szabályegyszerűsödést, egyszerűsítést eredményez.

Az **ipset** esetében különböző dimenziójú halmazokról beszélhetünk, ahol a dimenzión a halmazbeli elemek paramétereinek a számát értjük. Jelenleg a következő dimenziókat (típusokat) támogatja a rendszer:

1. IP cím: ipmap, iphash, iptree, iptreemap

2. network cím (azonos méretű hálózatok): ipmap, iphash
3. network cím (eltérő méretű hálózatok): nethash
4. IP cím, (forrás) MAC cím: macipmap
5. portszám: portmap
6. IP cím, portszám: ipporthash
7. IP cím, portszám, IP cím: ipportiphash
8. IP cím, portszám, network cím (eltérő méretű hálózatok): ipportnethash
9. halmaznév: setlist

A különböző típusok nevei értelemszerűen „kódolják” a tárolható elemek paramétereit és dimenzióit (*ip*, *net*, *port*, *mac*), valamint a tárolási módszert (*map*, *hash*, *tree*).

## 3.1. map típusok

A *map* tárolási módszer egy bináris tömbben keres indexelt bit egyezést. A típus a leggyorsabb keresést adja az összes közül, azonban használatának peremfeltétele, hogy a tárolandó IP címeknek maximálisan egy /16-os blokkból kell származni. (Port esetén az összes lehetséges port tárolható ☺.)

## 3.2. hash típusok

A *hash* típus – nevéből következően – egy hash tömbben tárolja az elemeket. Az ütközések feloldására a láncolt elemtárolás helyett újra hasheléssel keres szabad pozíciót a tömbben. Ez azt jelenti, hogy próbálkozások számával (`--probes` paraméter) azt is beállítjuk, hogy egyezés keresésekor hányszor ismételjen egy keresést! Vagyis ha pl. a 192.168.1.1 IP cím **nincs** az adott tömbben, és a `--probes` paraméter az alapértelmezett 8, akkor az algoritmus nyolccszor próbálkozik hasheléssel, és csak ez után állapítja meg, hogy az IP cím nem található az adott halmazban.

A *hash* típusok (adott határértékig) újraméretezik önmagukat, amikor betelnek, és újab elemeket akarunk hozzájuk adni.

### 3.3. tree típusok

A *tree* típus egy fix, négyes mélységű fában tárolja és keresi az IP címeket. Sebességben ezért (alapértelmezett beállításoknál) gyorsabb, mint a hash, de lassabb, mint a map típus. Különlegessége, hogy támogat elemenkénti timeout paramétert, így adott idő után automatikusan törlődő IP cím tárolásra is képes.

### 3.4. setlist típus

Az *setlist* típusnál a halmazok unióját tudjuk megvalósítani: az elemei más típusú halmazok lehetnek.

## 4. Példák ipset használatára

A leghatékonyabb konkrét példákon, a gyakorlat szempontjából megmutatni az **ipset** használatát.

### 4.1. daemonshield és ipset

A *daemonshield*[4] egy Python-ban írt naplófigyelő program, amellyel támadók automatikusan kitilthatók. A csomag a kitiltásra **iptables** és null-routing megoldásokat tartalmaz, de könnyedén bővíthetjük az **ipset** használatával. Mindössze egy kis plugin scriptre van szükségünk, amely induláskor létrehozza a szükséges **ipset** halmazt és segédszabályokat, majd leálláskor mindezeket eltávolítja, a támadók IP címeit pedig hozzáadja/törli a halmazból. Adjuk meg ennek a script-nek az elérési útvonalát a *daemonshield* konfigurációs fáljban az *iptablescmd* paraméter értékeként.

Következzék egy lehetséges változata a script-nek:

```
#!/bin/sh

# Útvonalak a binárisokhoz
iptables="/usr/sbin/iptables"
ipset="/usr/local/sbin/ipset"
# Halmaz és létrehozásának paraméterei
setname="daemonshield"
create="iphash --probes 2"

case "$1" in
  init)
    # Inicializálás
    $iptables -N $2-log
```

```
    $iptables -A $2-log -m limit --limit 1/m \
      -j LOG --log-level info --log-prefix 'Shielded: '
    $iptables -A $2-log -j DROP
    $ipset -N $setname $create
    $iptables -A $2 -m set --match-set $setname src \
      -j $2-log
    ;;
  cleanup)
    # Leállás
    $iptables -D $2 -m set --match-set $setname src \
      -j $2-log
    $iptables -F $2-log
    $iptables -X $2-log
    $ipset -X $setname
    ;;
  add|del)
    if [ $1 = "add" ]; then
      opt="-A"
    else
      opt="-D"
    fi
    # Támadó hozzáadása/törlése a halmazból
    $ipset $opt $setname $4
    ;;
  *)
    echo "Usage: $0 init|cleanup <inchain> <outchain>"
    echo "       $0 add|del <inchain> <outchain> IP"
    exit 1
    ;;
esac
```

## 4.2. DNS és ipset

Ha olyan a biztonsági szabályzat, hogy csak a DNS-ben regisztrált (kliens) gépek férhetnek hozzá az Internethez, ekkor is segíthet az **ipset**. Létrehozunk egy halmazt a kliensek számára, majd ezt rendszeresen frissítjük cron-ból. A frissítés során legegyszerűbb az **ipset** azon tulajdonságára támaszkodnunk, hogy egy halmaz könnyedén felcserélhető egy másik halmazzal anélkül, hogy az a halmazra hivatkozó, azt használó **iptables**-t megzavarná.

A cron a scriptünket „refresh” argumentummal hívja meg; az erre illeszkedő script részlete a következő:

```
case "$1" in
```

```
....
  refresh)
    # DNS-ből a halmaz elemeinek a generálása
    /etc/fw/refresh_set
    # Ha van változás, betöltjük az új halmazt
    # és felcseréljük az aktívval, majd töröljük
    # az ideiglenes halmazt
    if [ "‘diff /etc/fw/clients \
                /etc/fw/clients.temp‘" ]; then
      ipset -R < /etc/fw/clients.temp
      ipset -W clients clients.temp
      ipset -X clients.temp
      mv /etc/fw/clients.temp /etc/fw/clients
    fi
    ;;
....
```

Az */etc/fw/refresh_set* script lekérdezi a DNS szerverünktől a teljes táblázatot, majd elmenti az IP címeket az **ipset** ún *restore* kompatibilis formátumában:

```
#!/usr/bin/perl

my $nameserver = ’name.server.ip’;
my @zones = qw(zone1.hu zone2.hu);
my($name, $ip);
my %ip;
foreach my $zone (@zones) {
  open(DIG, "/usr/bin/dig \@$nameserver -t axfr $zone|");
  while (<DIG>) {
    if (/^(\S+)\s+\d+\s+IN\s+A\s+(\S+)/) {
      $ip{$2} = $1;
    }
  close(DIG);
}
open(OUT, ">/etc/fw/clients.temp");
print OUT <<TXT;
# DO NOT EDIT: GENERATED FILE
-N clients.temp ipmap --network 192.168.0.0/16
TXT
foreach (sort keys %ip) {
  print OUT "# $ip{$_}\n-A clients.temp $_\n";
}
print OUT "COMMIT\n";
close(OUT);
```

Az egyszerűség kedvéért ez egy Perl script – nagyon könnyen alakítható, bővíthető.

Talán szükségtelen hangúlyozni, hogy a *clients* halmazra hivatkozó **iptables** szabály akár tetszőlegesen komplex szabályegyüttes része lehet. A szabályokat, a halmazra hivatkozó szabályt a halmaz módosítása, kicserélése semmilyen szempontból nem érinti, nem zavarja meg.

# 5. iptables és ipset

Megtehetjük azt is, hogy az **iptables** mellett, amennyire csak lehet, **ipset**-re támaszkodunk. Az előnye az, hogy minimális szabálykészletünk marad, amelyet rendkívül könnyű átlátni, megérteni. Ráadásul ezek a szabályok statikusak abban az értelemben, hogy – a hálózati tartományon belül – a kliensek, szerverek listája tetszőlegesen módosítható.

A tűzfal indítása két fázisból áll: a halmazok definiálásából és feltöltéséből, majd az **iptables** szabályaink betöltéséből:

```
#!/bin/sh

load_sets() {
    # Definiáljuk a halmazainkat
    ipset -N clients$1 ipmap --network 192.168.0.0/16
    for x in tcp-servers udp-servers; do
      ipset -N $x$1 ipporthash --network 192.168.0.0/16
    done
    # Listafájlokból feltöltjük ezeket
    for x in clients$1 tcp-servers$1 udp-servers$1; do
      (awk "\$1 != \"#\" {print \"-A $x\", \$1}" \
         < /etc/fw/$x; echo COMMIT) | ipset -R
    done
}

case "$1" in
  start)
    # Betöltjük a halmazokat:
    load_sets
    # Stateful tűzfal:
    iptables -A FORWARD \
      -m state --state ESTABLISHED,RELATED -j ACCEPT
    # Naplózó szabályok:
    for x in drop accep; do
      action=‘echo $x | tr a-z A-Z‘
```

```
      iptables -N log-$x
      iptables -A log-$x -j LOG --log-prefix "$action: "
      iptables -A log-$x -j $action
    done
    # Kliensek szabálya
    iptables -A FORWARD -i lan -m state --state NEW \
      -m set --match-set clients src -j log-accept
    # Szerverek szabálya, TCP
    iptables -A FORWARD -i wan -m state --state NEW -p tcp\
      -m set --match-set tcp-servers dst,dst -j log-accept
    # Szerverek szabálya, UDP
    iptables -A FORWARD -i wan -m state --state NEW -p udp\
      -m set --match-set udp-servers dst,dst -j log-accept
    # Minden más tiltott
    iptables -A FORWARD -j log-drop
    ;;
....
```

A tűzfal leállításakor fordított sorrendben kell lebontanunk az elemeket: először töröljük a szabályokat, majd a halmazokat:

```
....
  stop)
    # Kiürítjük, majd töröljük a láncokat
    iptables -F
    iptables -X
    # Hasonlóan, kiürítjük, majd töröljük a halmazokat
    ipset -F
    ipset -X
    ;;
....
```

Scriptünkhöz könnyen hozzáadhatunk egy *refresh* kapcsolót, amellyel a kliensek vagy szerverek változása esetén *frissíthetjük* a szabályainkat:

```
....
  refresh)
    # A klienseket/szervereket ideiglenes
    # halmazokba töltjük
    load_sets .temp
    # Felcseréljük a halmazok tartalmát,
    # majd töröljük az ideiglenes halmazokat
    for x in clients tcp-servers udp-servers; do
      ipset -W $x $x.temp
      ipset -X $x.temp
```

```
    done
    ;;
....
```

A tényleges listák meglehetősen egyszerűek: a klienseké pusztán egy IP cím lista

```
# /etc/fw/clients fájl: IP cím soronként
192.168.1.23
...
```

míg a szervereké `IP cím,portszám` páros, szóköz nélkül:

```
# /etc/fw/tcp-servers fájl: IP cím,portszám soronként
192.168.1.88,80
...
```

Amint látjuk, minimális számú **iptables** szabály ténylegesen hatalmas, több ezer szabályt fejezhet ki **ipset** segítségével.

Nincs akadálya annak sem, hogy az első fejezetben említett teoretikus **iptables** szabálygeneráló interfészt általánosítsuk – és egyszerűsítsük – az **ipset** háttérbeli használatával ☺.

# 6. Összefoglalás

Az elméleti alapozás után gyakorlati példákon kereszül mutattuk be az **ipset** használatát a netfilter/iptables alapú tűzfalak egyszerűbb és nagyobb teljesítményű felépítése érdekében.

# Hivatkozások


[1] http://www.netfilter.org

[2] http://ipset.netfilter.org

[3] http://people.netfilter.org/kadlec/nftest.pdf

[4] http://daemonshield.sourceforge.net

# Google Android fejlesztői szemmel

Kis Gergely

**Kivonat**

Szinte mindenki hallott már a Google mobiltelefon platformjáról, az Androidról. Nagy várakozás előzte meg a bejelentését, és az izgalom csak nőtt arra a hírre, hogy a Google nyílt forráskódúvá kívánja tenni. Az első, fejlesztőknek szóló előzetes kiadás óvatos optimizmusra adott okot, de 2008 szeptember végéig nem tudhattuk, hogy mit kapunk az 1.0-ás verzióban.

Azt kell mondjam, hogy az Android 1.0 felülmúlta a várakozásaimat. Egy nagy halom használhatatlan forráskód helyett egy jól megtervezett, azonnal használható, nyílt forráskódú platformot tettek elérhetővé, amely azóta is folyamatosan fejlődik. Nagyon jó fejlesztői közösség alakult ki mind a hivatalos projekt keretein belül, mind pedig a különböző külső fejlesztőcsoportoknál.

A továbbiakban áttekintem az Android platform lehetőségeit az alkalmazásfejlesztés szemszögéből. Bemutatok egy példaprogramot, amely jól illusztrálja a platformban rejlő lehetőségeket. Ezután bemutatom a Hundroid néven most szerveződő Magyar Android Közösséget. Végül arra is kitérek, hogy mit várhatunk az Androidtól mint a Google által szponzorált nyílt forráskódú projekttől.

# Tartalomjegyzék

**1. Mi az Android? 59**
1.1. Hivatalosan Androiddal szállított eszközök . . . . . 59
1.2. Nem hivatalos eszközök . . . . . 60

**2. Az Android architektúrája 61**

**3. Fejlesztőkörnyezet – gyakorlati példa 64**

**4. Az Android 1.6 (Donut) újdonságai 68**

**5. Nyílt forráskódú az Android? 69**

**6. Hundroid – Magyar Android Közösség 72**

**7. Összegzés 73**

# 1. Mi az Android?

Az Android fejlesztői szemmel nézve egy szoftverplatform, amit mobil eszközökhöz fejlesztettek ki. Tekintsük át, hogy milyen fő tulajdonságai teszik egyedivé:

- **Nyílt forráskódú:** Sok más nyílt forráskódúnak titulált rendszerrel szemben az Androidból valóban használható rendszer építhető kizárólag nyílt forráskódú komponensek felhasználásával.
- **Linux kernelre épül:** Azonban mégsem egy átlagos beágyazott Linux rendszerről beszélhetünk. Az Android architektúrája, mint később látni fogjuk, jelentős különbségeket mutat a GNU/Linux rendszerekkel szemben.
- **Java nyelven írhatók rá alkalmazások:** A Java nyelv egy egyszerű, széles körben elterjedt és emiatt nagyon jól „felszerszámozott” nyelv – az Eclipse és egyéb fejlesztőkörnyezetek kezdők számára is nagyon leegyszerűsítik a fejlesztést.

Az Android, bár jelenleg elsősorban mobiltelefonokon használják, de alkalmas arra, hogy tabletek, set-top-boxok vagy car-pc-k operációs rendszere legyen.

Joggal merülhet fel a kérdés az olvasóban, hogy jelenleg hogyan juthat hozzá Android rendszert futtató készülékhez. Itt két csoportot különböztethetünk meg.

## 1.1. Hivatalosan Androiddal szállított eszközök

Már több gyártó is forgalmaz Android alapú mobiltelefont. A legtöbb eszköz jelenleg a HTC kínálatában található, de a többi gyártó (pl. Samsung, Motorola, Huawei, LG…stb.) is gyorsan zárkózik fel. Ki kell emelni két eszközt, a *HTC Dream*-et (T-Mobile G1) és a *HTC Sapphire*-t (HTC Magic néven forgalmazzák). Ezeknek az eszközöknek megvásárolható a fejlesztői változata ADP1 és ION néven. Miért érdekes ez?

Amikor megvásárolunk egy mobiltelefont, akkor nem kapjuk meg azt a szabadságot, amit például egy általános célú PC-nél megszokhattunk: nem cserélhetjük le szabadon az operációs rendszert, nem rendelkezünk rendszergazdai jogosultságokkal. Kizárólag a gyártók által kiadott hivatalos frissítéseket tölthetjük fel az eszközünkre.

Ez azt eredményezné, hogy bár az Android forráskódjából építhetünk működő saját verziót, azt hivatalos hardveren nem tudnánk kipróbálni. Erre a problémára nyújtanak megoldást a fejlesztői eszközök: ezekre bármilyen saját magunk által épített szoftvert telepíthetünk. Bár néhány eszközmeghajtó csak zárt kódként érhető el (pl. GPS, WLAN vagy a GSM/UMTS alrendszer eléréséhez szükséges modulok), a rendszer jelentős részét szabadon módosíthatjuk.

Természetesen egy rendszer sem tökéletes. Szoftverhibákat kihasználva a jelenleg piacon kapható Android készülékek jelentős része fejlesztői készülékké alakítható, így megnyitva az utat a nem hivatalos szoftverkiadások előtt mindenki számára. Ezek általában több funkciót tartalmaznak, mint a hivatalos verziók, ám mivel a telepítésük (a fejlesztői telefonok kivételével) a garancia elvesztését is jelentheti, ezért mindenki csak saját felelősségre próbálkozzon.

Egy új jövevény a piacon az Archos Androidot futtató tablet termékei. Két szempontból is úttörőnek tekinthetők:

- Az első olyan termék a piacon, amely nem mobiltelefon.
- Az Archos az első olyan gyártó, amely a nyílt forráskódú Android platformra építette a termékét, így a Google saját, zárt kódú alkalmazásai nem találhatók meg rajta, így nem érhető el az alkalmazások fő forrása, az Android Market sem.

Talán ezek miatt is vegyes ezeknek az eszközöknek a fogadtatása. Szerencsére a nyílt forráskód lehetővé teszi, hogy a gyermekbetegségeket gyorsan orvosolják. A legtöbb bírálat amiatt érte az Archost, hogy nem tették nyilvánvalóvá: a tabletek nem fogják elérni az Android Marketen található alkalmazásokat. Szerencsére alternatív lehetőségek, mint például a SlideME vagy az AndAppStore ezeknek az eszközöknek a felhasználói számára is megoldást nyújthatnak.

## 1.2. Nem hivatalos eszközök

A nyílt forráskódú platform erejét mutatja, hogy hetekkel az Android 1.0 megjelenése után elkészült az első futtatható változat az OpenMoko FreeRunner készülékekre. Ez különösen azért tekinthető nagy eredménynek, mivel az Android platform eredetileg egy újabb ARM utasításkészlet-verziót, az ARMv5TE-t támogatja, míg a FreeRunnerben csupán ARMv4T változatot futtatni képes processzor található. A platform jól használhatóságát mutatja, hogy néhány nap alatt sikerült portolni a rendszert a régebbi processzorokhoz.

Az azóta eltelt egy év alatt a nyílt forráskódú Android platform már támogatja a MIPS és x86 architektúrákat is, így a beágyazott eszközök egy jelentős részén futtatható. A LiveAndroid kiadás segítségével akár az otthoni PC-nket is ideiglenesen Android rendszerré alakíthatjuk.

Az OpenMoko FreeRunneren kívül más konkrét hardverplatformokra is portolták már az Androidot. Külön figyelmet érdemel a TI OMAP platform, amely már az újabb Cortex-A8 -as ARM mag köré épül, és olcsó fejlesztői verzióként elérhető BeagleBoard néven. Erre épül az Always Innovating cég TouchBook terméke, amely így képes az Android futtatására is.

Általánosságban elmondható, hogy bármely olyan Linuxot futtató beágyazott eszköz, amelyben legalább 100-128MB RAM és 4-500 Mhz órajelű processzor található, alkalmas az Android platform futtatására. Amennyiben nyílt forráskódú meghajtó programok rendelkezésre állnak az eszköz speciális hardverelemeihez, teljes értékű Android rendszer építhető.

# 2. Az Android architektúrája

Ebben a részben áttekintjük az Android architektúrájának különböző elemeit, különös figyelmet fordítva a más Linux alapú rendszerekhez képest felfedezhető különbségekre.

A rendszer alsó szintjén természetesen a Linux kernelt találjuk, néhány Android specifikus folttal kiegészítve. Ezek a teljesség igénye nélkül:

- **Binder:** az Android komponensrendszerének kernel oldali kommunikációs modulja
- **Android Power Management:** speciális, mobiltelefonok igényeihez kifejlesztett erőforráskezelési funkciók, mint például a „wake lock”, amivel egy alkalmazás ébren tartja az egyébként felhasználói események hiányában automatikusan elalvó a rendszert.
- **Android Logging:** speciális naplózó alrendszer.
- **Low Memory Killer:** A Linux kernel OOM killer komponenséhez hasonló megoldás. Lényege, hogy a felhasználó számára fontos folyamatokat (az éppen képernyőn látható alkalmazásokat) tekinti a legfontosabbnak a rendszer szempontjából, Így csak olyan alkalmazásokat állít le, amelyek a felhasználó számára éppen nem bírnak jelentőséggel. Az Android platform életciklus-menedzsmentje lehetővé teszi, hogy szükség esetén ezek a folyamatok újrainduljanak, így a felhasználó ebből a zsonglőrködésből semmit nem vesz észre.

A rendszer következő szintjén találhatók a különböző függvénykönyvtárak. Itt is jelentős különbségeket találunk egy hagyományos GNU/Linux rendszerhez képest.

- **Libc:** A megszokott glibc / uclibc vagy újabban eglibc használata helyett az Android tervezői egy saját, minimalista implementáció mellett döntöttek. A BSD gyökerekkel is rendelkező könyvtár a Bionic nevet kapta. Csupán azokat a funkciókat tartalmazza, amelyek az Android számára szükségesek.

- **SGL (Skia):** Az Android Skia 2D grafikus könyvtárat használja a hagyományosan használt alternatívák helyett, mint például a Cairo.

- **OpenGL ES:** Az Android tartalmaz egy OpenGL ES implementációt is, amellyel a 3D grafikai igényeket tudja kiszolgálni

- **OpenCORE:** Az Android multimédiás képességei a Packet Video által kifejlesztett OpenCORE könyvtárra épülnek. A legfontosabb kodekeket tartalmazza a nyílt forráskódú változat, de lehetőség van hardvergyorsítást használó kodekek integrálására is.

- **Surface Manager:** Az Android egy saját, könnyűsúlyú 2D / 3D kompozitor könyvtárat használ a grafikus felület kezeléséhez.

Mindezek mellett természetesen az Android is sok ismert, nyílt forráskódú könyvtárat használ, mint például: WebKit, FreeType, Dbus vagy SqLite.

Az alkalmazások számára a Dalvik virtuális gép biztosít futtatókörnyezetet. Ez egy regiszteralapú virtuális gép, amely korlátozott erőforrásokkal rendelkező eszközökre lett optimalizálva. Néhány érdekesebb tulajdonsága:

- **Saját bájtkód formátum** (DEX) a memóriahasználat optimalizálásához. Ez a formátum olyan tömör, mint a jar fájlokban tömörítetten tárolt Java bájtkód fájlok. Ez lehetővé teszi, hogy a programkódot közvetlenül több folyamat is a memóriába mappelje.

- **Zygote folyamat:** Előre betölti a rendszer indulásakor a közös osztálykönyvtárakat, és új folyamatok indításakor a fork() rendszerhívással közvetlenül átadja azokat az új folyamatok részére, további memóriafelhasználás és betöltési műveletek nélkül.

- **Szemétgyűjtő adatok a heaptől elkülönítve:** Így több lap maradhat megosztva a folyamatok között, és kevesebb memóriát igényel.

Az Android saját folyamatmodellt nyújt az alkalmazások részére. Az egyes alkalmazások külön-külön felhasználói és csoportazonosítóval futnak (hasonlóan, ahogy a Linux szervereken külön azonosítóval fut például a webszerver vagy a levelezőszerver). Ezzel azonnal biztosítva van, hogy a különböző alkalmazások csak korlátozott mértékben tudják egymás futását befolyásolni, még véletlenül se tudja egy alkalmazás a másik adatait törölni. Az alkalmazások közötti adatcserét az alkalmazás keretrendszer biztosítja.

Az alkalmazás keretrendszer a következő 4 elemre épül:

- **Activity:** Egy activity általában egy interakciós egységnek – egy képernyőnek – felel meg egy alkalmazásban, amellyel a felhasználó egy konkrét feladatot tud elvégezni. Az activity nézeteket (*View* – más rendszereken Widgeteknek is nevezik) használ a felhasználói felület megjelenítésére. Az egyes nézetek elrendezését Layoutok segítségével adhatjuk meg. Layoutok előre definiálhatók deklaratívan erőforrásfájlokban, vagy programkódból is létrehozhatók.

- **Service:** Egy alkalmazás háttérben futó szolgáltatásokat is definiálhat. Ezek a szolgáltatások akkor is futhatnak, amikor az alkalmazás felhasználói felülete nem látható. Tipikus példa a médialejátszó szolgáltatás, amely akkor is fut, amikor a programot lezártuk, és már egy másik alkalmazást használunk. Ezeket a szolgáltatásokat egy másik alkalmazás – másik folyamat – is igénybe veheti.

- **Broadcast Receiver:** Egy alkalmazásnak gyakran szüksége lehet arra, hogy különböző rendszereseményekre reagáljon. Ilyen esemény lehet például egy SMS üzenet érkezése vagy a telep töltöttségének kritikus szintje. Az egyes alkalmazások saját eseményeket is definiálhatnak. Egy Broadcast Receiver segítségével lehetőség van arra, hogy egy alkalmazás értesüljön ezekről az eseményekről, és reagáljon rá. A Broadcast Receiverek önmagukban nem rendelkezhetnek felhasználói felülettel, de indíthatnak új Activity-t, vagy küldhetnek felhasználói értesítést a Notification Framework felhasználásával, amely a képernyő tetején található értesítési sorban jelenik meg.

- **Content Provider:** Adatokat tesz elérhetővé más alkalmazások számára. Ezzel a megoldással nem kell pontosan ismerni, hogy melyik alkalmazás tudja a kívánt adatokat biztosítani, csupán a megfelelő kérést kell elküldeni a rendszerhez, amely megkeresi a megfelelő alkalmazást, és közvetíti a választ a kérdező felé.

- **Intent:** Az Android rendszer egyik legérdekesebb funkciója, hogy a komponensek (Activity-k) aktiválása Intentek segítségével történik. Egy Intent egy üzenet, amellyel az éppen aktív alkalmazás egy felhasználói szándékot közvetít a rendszer felé. A rendszer ezt a szándékot értelmezi, és megkeresi az azt legjobban teljesítő alkalmazást, és elindítja azt. Így ha egy alkalmazásból szükségünk van egy felhasználó által kiválasztott névjegy adataira, akkor nem kell nekünk megírnunk a névjegyeket megjelenítő Activity-t, hanem egyszerűen közöljük a rendszerrel ezt az igényünket. A rendszer meg fogja találni a címtáralkalmazás megfelelő névjegykiválasztó Activity-jét és megjeleníti azt. Miután a felhasználó választott, ismét a mi alkalmazásunk kerül előtérbe, és a rendszertől megkapjuk a kiválasztott névjegy adatait is.

Ez a rendszer lehetővé teszi, hogy a rendszer bármely komponense cserélhető legyen, akár a hívások kezelése vagy a főképernyő is. (Mindkét lehetőségre van már működő példa.)

Egy speciális alkalmazástípus az App Widget, amely mini-alkalmazásként a főképernyőn jelenhet meg. Különlegessége, hogy az alkalmazás külön szabályozhatja a felhasználói felület frissítési gyakoriságát. Így egy olyan Widget, ami csak 10 percenként módosítja a felületét, jóval kevesebb energiát fog felhasználni.

# 3. Fejlesztőkörnyezet – gyakorlati példa

Az előadás illusztrálásához kifejlesztettem egy alkalmazást, a PreziMoze-ot. Ez lehetővé teszi, hogy egy szerveroldali alkalmazás segítségével az Androidot futtató telefonunkról irányítsuk az OpenOffice.org prezentációt.

A szoftver kezelőfelülete puritán, a prezentáció vezérlésére két gomb szolgál: az előző, illetve következő diára ugrás. A két gomb a teljes képernyőt elfoglalja, így az előadás közben anélkül kezelhetjük, hogy a telefon kijelzőjére kellene néznünk.

A kommunikáció a gazdagéppel Bluetooth segítségével történik. Az Android alkalmazások fejlesztésében kicsit jártasabb olvasó számára ez meglepő lehet: a jelenlegi Android SDK verziók egyik legnagyobb hiányossága a Bluetooth programozói felületek hiánya. Hogyan oldhatjuk meg ezt a problémát?

A hagyományosan Java nyelven írt alkalmazások mellett lehetőség van az egyes programok natív kódrészekkel történő kiegészítésére a JNI szabvány felhasználásával. Megfelelő Android kompatibilis, natív könyvtárakat két módon készíthetünk.

- **Az Android NDK segítségével:** A Google által hivatalosan támogatott, jövőbeni verziókkal garantáltan kompatibilis megoldás. Csupán néhány könyvtár érhető el, az 1.6-os verzióban: Libc, liblog, libm, libz, OpenGL ES és minimális C++ támogatás.
- **A nyílt forráskódú platform felhasználásával:** Így lehetőség van nem publikus API-k felhasználására azon az áron, hogy újabb platformverziókkal vagy akár különböző gyártók készülékeivel nem lesz kompatibilis az alkalmazás.

Az Android a Bluez Bluetooth protokollmegvalósítást használja, így lehetőség van az erre épülő Java könyvtárak Androidhoz portolására. Szerencsére ezt

a BlueCove könyvtár[1] fejlesztői meg is tették, így könnyedén lefordíthatjuk Android platformra.

Ezután a következő feladat magának az alkalmazásnak az elkészítése. Az Android SDK nagyon kényelmes környezetet biztosít:

- Windows, Linux és Mac OS X támogatás
- Jól működő emulátor, ha valódi hardver nem áll rendelkezésre
- Közvetlen hozzáférés a hardverhez és az emulátorhoz az adb eszköz segítségével (shell elérés, fájlrendszer elérés, szoftvertelepítés. . . stb.)
- Integrált Eclipse támogatás (Android projekt, erőforrásfájlok kényelmes szerkesztése és automatikus fordítása, integrált debugger szolgáltatás mind emulátoron, mind pedig valódi eszközökön, natív könyvtárak beillesztésének támogatása)

Miután telepítettük az SDK-t a developer.android.com oldalról, és feltelepítettük az Eclipse integrációt is, be kell állítanunk az Eclipse-et, hogy megtalálja az Android SDK-t. Ezután létrehozhatjuk első Android projektünket.

A példa alkalmazás kifejlesztése 3 részre osztható:

- Az elmentett eszközök megtekintését és újra felhasználását lehetővé tevő Activity kifejlesztése. Ez az Activity indul el, amikor az alkalmazást megnyitjuk.
- A környezetünkben található Bluetooth eszközök felderítését és a megfelelő eszköz kiválasztását lehetővé tevő Activity kifejlesztése.
- A prezentációt vezérlő Activity kifejlesztése.

A példaprogram teljes forráskódja elérhető nyílt forráskódú licenc szerint a `http://www.mattakis.hu/konferenciak/fsf-2009` címről. Itt csak néhány jellemző elemet mutatok be.

[1] `http://code.google.com/p/bluecove`

A prezentációs nézetet leíró erőforrásfájl tartalma:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10px">

        <Button
            android:text="@string/PrevSlide"
            android:id="@+id/PrevSlide"
            android:layout_width="0px"
            android:layout_weight="1"
            android:layout_height="fill_parent"
            android:padding="15dip"/>

        <Button
            android:text="@string/NextSlide"
            android:id="@+id/NextSlide"
            android:layout_width="0px"
            android:layout_weight="1"
            android:layout_height="fill_parent"
            android:padding="15dip"
            android:layout_alignParentRight="true"
            />

</LinearLayout>
```

Mindkét gomb felirata egy – egy szöveg-erőforrás azonosítóval van megadva, így a felület honosítása a megfelelő szövegek lefordítására egyszerűsödik. A gombok azonosítóival pedig a programkódból hivatkozhatunk rájuk, és viselkedést rendelhetünk hozzájuk. A következő dia behívását végző kódrészlet:

```
nextSlide = (Button) findViewById(R.id.NextSlide);
nextSlide.setOnClickListener(new OnClickListener() {

    public void onClick(View v) {
        nextSlide();
    }
});
```

Itt láthatjuk, hogy az Androidban az egyes erőforrásokra az $R$ osztály segítségével hivatkozhatunk. Ezt az osztályt az Eclipse integráció állítja elő számunkra automatikusan, parancssori, illetve automatizált fordításnál pedig az aapt parancs segítségével generálhatjuk.

Fontos elem az alkalmazások fejlesztésénél az Activity-k életciklusainak megfelelő kezelése. Élete során az Activity a következő állapotokat veheti fel:

- Az Activity indításakor az `onCreate()` metódus hívódik meg. A leállításakor az `onDestroy()` metódus. E két metódushívás között zajlik az Activity teljes életciklusa.

- Az `onStart()` metódus akkor hívódik, amikor az Activity látható válik a felhasználó számára, noha nem feltétlenül van az előtérben, és így nem az

adott Activity-vel dolgozik a felhasználó. Az onStop() metódus akkor hívódik, ha az Activity elrejtésre kerül a felhasználó elől. Az `onStart()` és az `onStop()` metódusok többször is meghívódhatnak az Activity élete során.

- Az `onResume()` metódus akkor hívódik meg, amikor egy adott Activity előtérbe kerül, és a felhasználó azzal az Activity-vel végez interakciót. Az `onPause()` metódus akkor hívódik meg, ha az Activity háttérbe kerül. Ezek a metódusok igen gyakran meghívódhatnak, ezért a futásidejüknek lehetőség szerint rövidnek kell lenniük.

Végül az alkalmazásomban található komponensek és a felhasználni kívánt jogosultság az AndroidManifest.xml fájlban kerülnek definiálásra. A telepítéskor az itt található adatok alapján adja meg a rendszer a megfelelő jogosultságokat, illetve regisztrálja az Activity-ket, hogy az Intentek segítségével később megtalálhatók legyenek. A PreziMote alkalmazás esetében a fájl tartalma:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.mattakis.prezimote"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Main"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".DeviceSearch"
                  android:label="@string/device_search">
        </activity>
        <activity android:name=".PreziControl"
                  android:label="@string/prezi_control">
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />

<uses-permission android:name="android.permission.BLUETOOTH"></uses-permission>
<uses-permission
          android:name="android.permission.BLUETOOTH_ADMIN"></uses-permission>
</manifest>
```

Az alkalmazás tesztelését végezhetjük közvetlenül a célhardveren: egyszerűen az Eclipse-ben a debug indítást választva a rendszer automatikusan telepíti az alkalmazásunkat, elindítja és csatlakoztatja a debuggert. A definiált töréspontokon a futás felfüggesztésre kerül, így a szokott módon vizsgálhatjuk a program állapotát, vagy léptethetjük előre.

Remélem, sikerült kedvet csinálni az alkalmazásfejlesztés kipróbálásához. Ha valaki a fejlesztés során bármilyen problémába ütközik, a Hundroid levelezési listáján kaphat magyar nyelvű segítséget.

# 4. Az Android 1.6 (Donut) újdonságai

Szeptemberben jelent meg az Android platform legújabb változata. Röviden bemutatok néhány érdekesebb funkciót az újdonságok közül: Újdonságok felhasználók számára:

- **Gyorskereső (Quick Search Box):** Az új, áttervezett kereső keretrendszer lehetővé teszi, hogy a felhasználó bármit gyorsan megtaláljon a telefonján. A kereső több forrást is felhasznál: böngésző könyvjelzők és történet, címjegyzék és az egész web – közvetlenül a főképernyőről. A rendszer folyamatosan tanulja, hogy melyek a fontosabb információk a felhasználó kattintásai alapján. A fejlesztők könnyedén új adatforrásokat adhatnak a keresőhöz.

- **Kamera, videófelvevő és galéria:** Az áttervezett felület integrált felhasználói élményt nyújt: a felhasználó könnyedén válthat a fényképezés és a videókészítés között. A galériában már lehetőség van több fotó törlésére egyszerre. A kamera teljesítménye is javul az 1.6-os verzióban: az alkalmazás 39%-kal gyorsabban indul, és egy fénykép készítése 28%-kal gyorsabb.

- **Virtuális magánhálózat (VPN) támogatás:** Lehetővé válik a következő típusú VPN hálózatok használata: L2TP/IPSEC VPN előre megosztott kulccsal vagy tanúsítvánnyal, csak L2TP alapú VPN, PPTP alapú VPN.

- **Akkumulátor felhasználás jelző:** Az eszköz segítségével a felhasználó ellenőrizheti, hogy mennyi energiát használ egy-egy alkalmazás, és törölheti a túl „éhes” alkalmazásokat.

- **Android Market frissítések:** Az új felület lehetővé teszi a könnyebb navigációt (Legjobb ingyenes, legjobb kereskedelmi és az Új alkalmazások) . Ezen kívül a készítők által feltöltött képernyőképeket is meg tudja jeleníteni.

- **Akadálymentesítés támogatása:** Az új akadálymentesítés keretrendszerhez a felhasználók alkalmazásokat tölthetnek le a Marketről, amelyek segíthetik a fogyatékkal élő felhasználók számára az eszközök felhasználását (pl. hangvezérlés, képernyőolvasó. . . stb.).

Újdonságok fejlesztők számára:

- **Új kereső keretrendszer:** A fejlesztők kereshetővé tehetik az alkalmazásaikat, és találatokat adhatnak vissza a felhasználók kereséseire. A felhasználó beállíthatja, hogy mely alkalmazásoktól vár találatokat.

- **Szövegfelolvasó (text-to-speech) motor:** A platformba integrált felolvasó motor („Pico”) lehetővé teszi, hogy bármely alkalmazás szöveget olvasson fel a kiválasztott nyelvnek megfelelő kiejtéssel. A támogatott nyelvek: angol (amerikai és brit akcentus), francia, olasz, német és spanyol. A T-Mobile G1 és Dream (ADP1) felhasználóknak az Android Marketről kell majd letölteniük az egyes nyelvekhez tartozó hangadatokat a telefonok korlátozott belső memóriája miatt.

- **Gesztusok támogatása:** Az új gesztus keretrendszer lehetővé teszi, hogy az alkalmazások készítői gesztusokat hozzanak létre, tároljanak és ismerjenek fel, így az alkalmazások jobb felhasználói élményt nyújthatnak. A GestureBuilder eszköz lehetővé teszi, hogy a fejlesztők előre elkészített gesztusokat csomagoljanak az alkalmazásaik mellé.

- **Új akadálymentesítés keretrendszer:** A fejlesztők új „akadálymentesítés beépülőmodulokat” fejleszthetnek, mint például hangjelzés adása, ha egy új ablak jelenik meg, vibrálás a listák tetején vagy szöveg felolvasása a képernyőről. Kiterjesztett támogatás különböző képernyősűrűségek és felbontások számára: Az új verzió lehetővé teszi, hogy az alkalmazások különböző méretű képernyőkön is megfelelő módon jelenhessenek meg. A fejlesztők a támogatott képernyőméreteket is megadhatják.

- CDMA telefonhálózatok támogatása

- **Új OpenCore verzió (multimédia keretrendszer):** Az 1.6-os verzióba az OpenCore 2-es változata került, mely a következő újdonságokat tartalmazza: OpenMAX enkóderek támogatása, további audio kodekek támogatása az AuthorEngine-ben, továbbfejlesztett bufferelés támogatás lehetővé teszi a dekóderek által lefoglalt osztott pufferek használatát.

- 2.6.29-es Linux kernel

# 5. Nyílt forráskódú az Android?

Az Android platform megjelenésekor voltak néhányan, akik meglehetősen hangosan adtak hangot azon véleményüknek, hogy az Android valójában nem is nyílt forráskódú, és ez az egész csak egy átverés. „J2ME új köntösben”.

A valóság ezzel szemben az, hogy az Android elérte azt, amit az OpenMoko próbált tenni: létrehozott egy nyílt forráskódú mobiltelefon platformot, és azt széles körben elterjesztette. Mindezt úgy, hogy a nyílt forráskód gondolatát összeegyeztethetővé tette a mobiltelefongyártók és a mobilszolgáltatók érdekeivel. Te-

kintsük át, hogy ezt hogyan érhette el (a Google által belefektetett jelentős tőke mellett).

Első feladatunk a „szoftverplatform” szó definiálása:

- Jól dokumentált programozási felületek (API-k) rendszere, amelyekkel alkalmazások fejleszthetők.
- Előregyártott komponensek, amelyek egyrészt megvalósítják a definiált API-kat, másrészt egy alaprendszert nyújtanak a kifejlesztett alkalmazások futtatásához. Sok esetben néhány alapalkalmazás is a platform részét képezi.

Értelemszerűen nyílt forráskódú szoftverplatformról akkor beszélhetünk, ha a felületek és a komponensek megvalósítása nyílt forráskódú licenc szerint érhető el.

Egy termékfejlesztéssel foglalkozó gyártó (pl. mobiltelefongyártó) egy adott szoftverplatformra építheti a termékét. Általában ebbe beletartozik az adott hardverkörnyezethez illesztés, további alkalmazások kifejlesztése, illetve a teljes termék viselkedésének finomhangolása. Az a szoftverplatform licencétől függ, hogy ezek a további fejlesztések szintén nyílt forráskódúak lesznek-e.

A fentiekből láthatjuk, hogy az Android platform nyílt forráskódúsága nem jelenti azt, hogy az arra épülő termékek minden komponense is nyílt forráskódú. Ez egyébként a GNU/Linux platformra is igaz: vannak kereskedelmi Linux terjesztések, melyek jelentős, zárt forráskódú komponenseket tartalmaznak.

Mit jelent ez számunkra?

- A nyílt forráskódú platform lehetővé teszi, hogy a saját magunk által fejlesztett hardverre portoljuk az Androidot.
- Lehetőséget kapunk arra, hogy új funkciókkal bővítsük a platformot, amely aztán minden Androidra épülő eszközben megtalálható lesz.
- A készülékgyártók támogatásától függően lehetőséget kapunk arra, hogy az általunk bővített platformot az eszközeiken futtassuk. Jelenleg ilyenek a HTC fejlesztői mobiltelefonjai.

Ezt nyújtja számunkra az Android Open Source Project. Természetesen ez a helyzet sem tökéletes. Néhány példa:

- A HTC telefonokon történő futtatáshoz szükséges bináris komponensek csak azután elérhetők, miután a megfelelő szoftververziót a HTC hivatalosan is kiadta. Emiatt a platform 1.6 (Donut) verzióját sokáig csak emulátoron lehetett tesztelni, amíg a hivatalos kiadás meg nem érkezett.

- Bizonyos fejlesztés alatt álló komponensek üzleti titkot képeznek, így nem jelenhetnek meg a nyílt forráskódú fában, amíg maga a termék meg nem jelenik.

- A Google fejlesztők nyílt forráskódú platformra fordítható erőforrásai végesek – bár gyakran erőn felül teljesítenek.

Ezekkel a korlátozásokkal együtt is az Android az egyik legnyíltabb, beágyazott rendszereket célzó platform. A jelenlegi jogszabályi és üzleti környezetet figyelembe véve sokkal nyíltabb megoldás csak jelentős korlátokkal képzelhető el – amely esetben a keletkező termékek nem lennének piacképesek. Mindezt láthattuk az OpenMoko példáján.

És el is jutottunk az Android üzleti oldalához, az Open Handset Alliance-hez. Ebben a szövetségben, néhány versenytárs kivételtől eltekintve (pl. Microsoft és Nokia) a mobil eszközökben érdekelt piac összes szereplője megtalálható.

Az Android platform túlnyomó része Apache 2.0 licenc alatt terjeszthető. Ez ugyanúgy támogatja a zárt kódú felhasználást, mint a BSD licenc, ugyanakkor a szabadalmi problémákra is megpróbál védelmet nyújtani. A kernel ugyan GPLv2 licencű, de az egy jól körülhatárolható komponense a rendszernek, így elfogadható volt minden szereplő számára. Ezen kívül csupán néhány GPL/LGPL v2 licencű komponens található a rendszerben, mindegyik jól elkülöníthető a rendszer többi elemétől.

Az Android platform fejlesztését jelenleg a Google vezeti a saját ütemterve szerint, és a hivatalos kiadásokat is ők készítik. A jelenlegi folyamat szerint a Google saját belső forrásfájában készíti el az új kiadás fejlesztői branchát, ahol a fejlesztés jelentős része zajlik. Ezt a branchot bizonyos időközönként a zárt kódú komponensek kivételével szinkronizálják a nyílt forráskódú tárolókba az android.git.kernel.org-on. Itt történik meg az új fejlesztések beolvasztása a master fejlesztői ágba, ahova a külső fejlesztések is kerülnek.

Látható, hogy a jelenlegi rendszerben csak korlátozott lehetőség van a nyílt forráskódú fejlesztések visszajuttatására a hivatalos kiadásokba. Gyakorlatilag a Google fejlesztőinek kézzel kell beleolvasztani a szükséges patcheket a belső fejlesztői fába.

A tervek szerint a jövőben módosulni fog a folyamat, és a Google az egyes verziók fejlesztésének kezdetekor a nyílt forráskódú platform master ágából fog kiindulni – így a külső fejlesztők elfogadott hozzájárulásai is bekerülhetnek.

A fentiekből jól látszik, hogy az Android platform jól egyensúlyozik az üzleti igények és egy nyílt forrású platformmal szemben támasztott közösségi elvárások között.

A platform lehetőségei és a fejlesztőket támogató funkciók kiadásról kiadásra gazdagodnak. A hivatalos Android Open Source Project-en (AOSP) kívül jelen-

tős szerepe van a közösségben a nem hivatalos buildek (vagy ROM-ok) készítőinek. Ők egyrészt a hivatalos, készülékgyártók által kiadott verziókra, másrészt az AOSP nem hivatalos továbbfejlesztéseire épített verziókat tesznek elérhetővé a fejlesztői készülékek tulajdonosai számára.

Ilyen kiadásokban megjelentek már olyan funkciók, amelyek technikai vagy jogi okok miatt még nem kerülhettek bele a hivatalos változatokba, például:

- Multitouch támogatás
- Alkalmazások futtatása SD kártyáról

Összességében kijelenthető, hogy az Android platform körül egy élő, aktív nyílt forráskódú közösség szerveződött.

# 6. Hundroid – Magyar Android Közösség

Azzal a céllal hoztuk létre a **Hundroid**ot[2], hogy lehetőséget adjon egy magyar nyelvű Android közösség létrejöttére.

A Hundroid infrastruktúra a következő szolgáltatásokat nyújtja:

- Közösen karbantartott Wiki jellegű weboldal
- Közösen szerkesztett Hundroid Blog
- Magyar nyelvű Android levelezési lista
- Online fordítórendszer a Honosítás projekthez
- Automatizált build rendszer a Honosítás projekthez – magyar nyelvű szoftververziók előállításához

Jelenleg első nagyobb projektünkön dolgozunk, ami a teljes nyílt forráskódú Android platform honosítása lesz. A munka koordinálása a levelezési listánkon folyik. A fordításba bárki bekapcsolódhat a `http://translate.hundroid.com/` címen. A fordításokat Apache 2.0 licenc szerint tesszük elérhetővé a hivatalos Android Open Source Project (AOSP) tárolóiban.

Ezért minden fordítónak regisztrálnia kell az AOSP-nél (`http://r.android.com`), és ott elektronikusan ki kell töltenie a Contributor License Agreement-et, hogy elfogadhassuk a hozzájárulását.

Minden érdeklődőt, támogatót szívesen látunk.

[2] `http://www.hundroid.com/`

# 7. Összegzés

Az Android egy jól használható, modern, nyílt forráskódú szoftverplatform. Segítségével kényelmesen fejleszthetünk alkalmazásokat mobiltelefonok millióira. Ha úgy tartja kedvünk, akkor a rendszert – nyílt forráskódú volta miatt – új hardverekre portolhatjuk, új felhasználási területeket találhatunk.

A platform folyamatosan fejlődik, és már most meghatározó szereplője a mobil eszközök piacának. A platform piaci részesedése minden bizonnyal továbbra is folyamatosan növekedni fog a következő évek során, ahogy egyre több gyártó jelenik meg Androidra épülő termékekkel.

Az üzleti megfontolások mellett fejlesztőként nagyon kényelmes használni. A nyílt forráskódú platform előnye, hogy az esetleges korlátok jelentős része megkerülhető (ahogy azt a példaprogramban is bemutattuk), illetve idővel a hivatalos platformba is bejuttatható a szükséges továbbfejlesztés.

# A szerzőről

Kis Gergely a MattaKis Consulting egyik alapítója és cégvezetője. Mobiltelefonoktól a J2EE vállalati alkalmazásokig számos projektben volt fejlesztő, architekt és projektvezető. A Budapesti Műszaki Egyetem műszaki informatika karán diplomázott, ezt követően 3 évig az Irányítástechnika és Informatika Tanszéken a Beágyazott Linux alkalmazások fejlesztése tantárgy oktatója. 12 éve Linux felhasználó és fejlesztő. Aktív támogatója a nyílt forráskódnak, korábban a Linux-felhasználók Magyarországi Egyesületében. A Hundroid – Magyar Android Közösség alapítója. Feleségével Dunakeszin él.

Elérhetőség: `gergely.kis@mattakis.com`, `http://www.mattakis.com/`

# Adatbiztonság és üzembiztonság mindenkinek? Egy frappáns válasz: uDS

Mátó Péter
Andrews Kft.

**Kivonat**

Mindenki szeretné biztonságban tudni az adatait, ezt azonban nem egyszerű megoldani. Lehet rendszeresen menteni, de a titkosítás megoldása még akkor is nehézkes, ha nem felejtjük el a mentést épp a legfontosabb munka közepén. Nagyjából egy éve kezdtem el egy biztonsággal és üzembiztonsággal foglalkozó szabad szoftver projektet, uDS néven. Kezdetben ez csak egy egyszerű shell script volt, mely összefogja a Linux kernel loop, RAID és crypt lehetőségeit, ma már egy kényelmes és jól használható megoldás, melyet mindenkinek csak ajánlani tudok. Előadásomban bemutatom a rendszer részeit és működését, lehetséges felhasználási területeit.

## Tartalomjegyzék

**1. A működési elv** **76**

**2. Blokkeszközök a Linuxban** **78**

**3. Az USB eszközökről** **78**

**4. A széf fájl létrehozása és a loop eszköz** **79**

**5. A RAID létrehozása** **80**

**6. Blokkeszközök titkosítása, a LUKS** **81**

**7. Fájlrendszer választás** **82**

**8. Felhasználási javaslatok** **82**

**9. Továbbfejlesztési lehetőségek** **83**

# Miért?

*Kedves Naplóm!*
*A mai nap úgy indult, mint a többi. Hullásan ébredtem, kávé, reggeli, irány a munka. Délután előadok az ELTE-n, még azt is át kell nézni. Beértem, ekkor jött a hideg zuhany: nem indult az a nyomorult notebook. Megint elszállt a diszk. Miért, uram, miért? Miért vered szerencsétlen szolgád? Naná, hogy csak azon volt meg az előadás anyagom. Kezdhetek gondolkodni, hogy beteget jelentsek, vagy inkább próbáljam meg a lehetetlent, álljak neki új diszket beszerezni, telepíteni, majd újra megcsinálni az előadást... Persze Lilo szokás szerint elhülyéskedte:*

*– Miért nincs a gépedben RAID?*
*– Miért, miért? Mert nem fér bele két diszk.*
*– Miért nem csinálsz akkor USB pendrive-okból?*
*– Hülye!*

Hmmm... Nem is olyan rossz ötlet!

Így kezdődött. Akkor a legnagyobb pendrive 64M-s volt, ami elég szánalmas. Már akkor is az volt. Néhány nagyobb doksi, és tele is lett. Pár éve már bármekkora pendrive vagy USB-s diszk kapható, és a problémám továbbra is megvolt: egy notebookon az adataim nincsenek biztonságban. Nincsenek, mert tönkremehet, ellophatják. Ha tönkremegy? Az adatoknak annyi. A Kürt kissé borsos áron dolgozik, nem nekem való. Ha ellopják? Még rosszabb. Nemcsak az a baj, hogy nekem nincs már meg, hanem az is, hogy másnak meg igen. Nem jó. Ha védekezni akarok a diszkhiba ellen? Mentek. Az utolsó diszkhiba után még gyakran, aztán egyre ritkábban. Végül nagyon ritkán. Megaztán a legnagyobb meló közepén csak olyan precíz emberek csinálnak mentést, mint Forest Gump. És igen, nyilván akkor megy tönkre a diszk. Így megy ez. Na jó, megcsináljuk a mentést. De azt is el lehet hagyni. El is szokták. Akkor megint ott tartunk, hogy nekünk már nincs meg, másnak viszont igen. Szinte minden megoldás nehézkes és veszélyes. Tiszta orosz rulett.

Mi lenne, ha lenne egy olyan program, ami automatikusan, rendszeresen megcsinálná a mentéseket, titkosítva, így az se lenne baj, ha ellopják, az se gond, ha szokás szerint tönkremegy a diszk... Na, épp ilyen az uDS. És a legjobb, hogy mindenki szabadon használhatja.

# 1. A működési elv

Az egész elgondolás mögött az az alapelv áll, hogy egy notebookban lévő eszköz és egy külső adattároló (például USB pendrive) között építek egy RAID1, azaz tükrözött tömböt, és azon tárolom az adatokat. De ha már lehetőség van rá, akkor

a tükrön lévő fájlrendszer legyen titkosított. Ezeket az adattárolókat úgy hívom, hogy széf. Mi ennek a miskulanciának az eredménye?

- Nem kell többé törődnöm a mentéssel és a titkosítással, az uDS megcsinálja helyettem.
- Az adataimat úgy használhatom több gépen, hogy minden gépen lehet egy-egy mentett példány az adatokból, a legfrissebb szinkronizálásáról a rendszer gondoskodik. És nem kell hozzá semmilyen internetes jócég, ingyenes tárhellyel.
- Tönkremegy a diszk vagy az USB pendrive (de csak az egyik)? A másikon még megvannak az adatok, szóval minden rendben.
- Elhagyom az USB kütyüt? Nincs gond, a notebook diszkjén még ott vannak az adatok.
- Ellopják a külső eszközt vagy a notebook-ot (de csak az egyiket)? Nem baj, a másik még nálam van, a lopó pedig nem tud mit kezdeni a nála lévő, zagyva adathalmazzal a jelszó nélkül.
- Ha időnként lecserélem az USB cuccot, és elteszem emlékbe, akkor az archiválást is megoldottam.

Mekkora fejlesztésre volt szükség ennek a létrehozásához? A válasz meglepő. Minimálisra. Csak használni kellett, amit a Linux rendszer nyújt. OK, sok napot töltöttem teszteléssel, próbálgatással, doksik és forráskódok olvasásával és a kollégákkal való beszélgetéssel, ötleteléssel. De maga a végeredmény meglehetősen egyszerű. Röviden összefoglalva ennyi lenne egy széf létrehozása:

```
# fdisk /dev/sdc
# dd if=/dev/zero of=~/uds/cfs.img bs=1 count=1 seek=7718490k
# losetup -f --show ~/uds/cfs.img
# mdadm --create /dev/md42 --verbose \
   --assume-clean --homehost=nowheree \
   --bitmap=internal --name=cfs \
   --level=mirror --raid-devices=2 \
   --write-mostly --write-behind /dev/sdc1 /dev/loop0
# dd if=/dev/urandom of=/dev/md42 bs=1M
# cryptsetup luksFormat /dev/md42
# cryptsetup luksOpen /dev/md42 cfs
# mkfs.ext2 -L cfs -m 0 /dev/mapper/cfs
# mkdir ~/uds/cfs
# mount /dev/mapper/cfs ~/uds/cfs
```

Jó, rendben, de hogy működik ez? Az, aki meg érti, hogy mi van fent leírva, azt kérdezheti, hogy miért éppen így? Leírom.

## 2. Blokkeszközök a Linuxban

A Linuxban az adattárolókat blokkeszközökön keresztül lehet elérni. Bővebb információért olvass el valami kezdő Unix könyvet. A Linuxban a Legóhoz hasonlóan egymásba lehet építgetni ezeket a blokkeszközöket. Különböző származásúakat is. Csinálhatok egy IDE, egy SATA és egy USB diszkből egy RAID tömböt, annak részeit titkosíthatom, majd a titkosított eszközökön LVM-et hozhatok létre, és annak részeiből RAID-et hozhatok létre és így tovább. Mondjuk nem szoktak ilyet csinálni, de a rendszer nagyon rugalmas.

## 3. Az USB eszközökről

Az USB eszköz használata meglehetősen tipikusnak mondható. Létrehozok rajta két partíciót, egy SFS (kódja: 42) és egy Linux (kódja: 83) típusút. Az SFS azért jó, mert ebben a galaxisban senki nem használja semmire, tehát lehet rá számítani, hogy az összes oprendszer békén fogja hagyni. Ez lesz a széf eszköz. A másik, az USB méretének nagyjából a 3%-a egy sima ext2 szerviz partíció, amin az uds bináris és a konfig egy másolata is megtalálható. Ezt lehet akár átmeneti adattárolónak is használni. A második partícióra két okból van szükség. Az egyik nyilvánvaló: ha valahol beteszem az eszközt, akkor elindíthatom a széfet az USB-n lévő bináris segítségével (persze néhány csomagnak fenn kell lennie a gépen). A másik ok nem ennyire nyilvánvaló, de nagyon praktikus: az USB eszközök gyártói néha trükköznek az eszközök méretével. Ha veszünk egy 8G feliratú pendrive-ot, nem tudhatjuk, hogy a G giga- vagy gigibájt. Ha időnként szeretném cserélni az eszközt, akkor nagyon nem mindegy, hogy az új eszközt hozzá tudom-e építeni a már működő széfemhez, vagy külön fel kell építenem, és át kell másolnom rá az adatokat. A 3% biztonsági tartalék. Számításaim szerint így a trükközés ellenére is be lehet üzemelni egy másik USB eszközt egy már létező széfhez.

Miért is szeretném időnként cserélni az USB eszközt? Az egyik megfejtés egyszerű: mert így három vagy négyhavonta kapok egy-egy archív USB eszközt, és ha előfordul az az aljasság, hogy ittas állapotban az USB diszkkel együtt a Dunába esik a notebook-om is, akkor még van valami használható állapotom. A másik érvem sokkal fontosabb: az USB eszközök flash memórián tárolják az adatokat, amely érzékeny az írásra, bizonyos idő után elromlik. Ettől sokkal pontosabbat nem lehet mondani. Ádám kollégám tesztjei szerint a legolcsóbb USB pendrive egy bizonyos egy megás része átírható volt 10 000 000-szor sérülés nélkül, bár papíron csak egymilliót kellene kibírnia. Ennek Ádám szerint az lehet az oka, hogy a CF és SSD eszközökben kötelező valamilyen írás-terítő (wear leveling) algoritmust beépíteni, és a távol-keletiek akkor inkább egy gyártósoron elintézik az USB eszközök vezérlőit is. Hát... Mittudomén. Mindenki döntse el maga. Egy biztos:

az USB eszközökön én még nem láttam precíz élettartamra vonatkozó feliratot, maximum a garanciaidőből lehet erre következtetni. Én nem szeretek veszélyesen élni, inkább időnként cserélgetem.

# 4. A széf fájl létrehozása és a loop eszköz

A notebook-ban lévő adattárolót kétféleképpen lehet felhasználni egy RAID létrehozásánál. Az egyik, hogy van a diszkjén egy megfelelő méretű partíció. Vagy a diszkjén lévő LVM-en. Ebből az utóbbi nem mindenkinek van a notebookján, sőt Lilon kívül szerintem senkinek nincs az ismert univerzumban. Az előbbi pedig annyira kényelmetlenül tekergethető (adatokat lement, újra partícionál, adatokat visszatölt, adatokat a mentés helyén megsemmisít), hogy már inkább szót se vesztegessünk rá. Valami más megoldás kell.

Bizonyára többen ismerik a trükköt, hogy CD képfájlt (gyk.: iso-t) lemezre írás nélkül is lehet csatolni, használni. Van ugyanis a mount parancsnak egy "-o loop" lehetősége, amivel ún. hurokeszközként csatolható egy fájlrendszer. Ez egy többlépéses eljárás automatizált változata, mely a valóságban így néz ki: a mezei fájlt rendeljük hozzá egy speciális emulált blokkeszközhöz a losetup parancs segítségével, majd azt a blokkeszközt használjuk diszkként. Ha tehát lenne egy normál fájl, azt is tudnám diszkként használni. Pont ezt csinálom.

Kell tehát egy fájl. Akkora kell, hogy épp jó legyen az USB-n lévő titkos partíció mellé egy RAID eszközbe. Ennek méretét a /dev/partitions fájlból lehet legegyszerűbben kiolvasni. Ez a partíciók méretét kilobájtokban adja meg. A fájl létrehozására több mód is van.

Első út: a dd parancs segítségével normál módon, valahogy így:

```
# dd if=/dev/zero of=~/uds/cfs.img bs=1k count=7718490
```

Tudom, hogy a blokkméret nagyon kicsi, ami lassítja a végrehajtást, de nem ez a legnagyobb gond vele. Az igazi baj az, hogy ha például a /dev/zero eszközből töltöm fel a fájlt, akkor teleírja a diszk területet nullákkal, ami nagyon sok idő és felesleges helyfoglalás. (Később látni fogjuk, hogy a nagyobb biztonság érdekében majd még egyszer fel kell töltenünk az egész fájlt, most még felesleges.) Az igaz, hogy ez a lehető leginkább kézenfekvő módszer, de messze nem a legszerencsésebb. A Linux ext(2|3|4) fájlrendszerei képesek lyukas fájlokat (sparse files) kezelni. A dd parancs segítségével ilyet is létre lehet hozni, mégpedig így:

```
# dd if=/dev/zero of=~/uds/cfs.img bs=1k count=1 seek=7718490k
```

Ez egy szempillantás alatt lefut, mert csak annyit csinál, hogy egy darab egy kilobájtos blokkot átmásol, majd odébb irányítja a fájlban a mutatót a kívánt végére, így a fájl mérete a seek paraméterben megadott lesz. Az így létrehozott 10G

méretűnek látszó fájl a fájlrendszeren valójában csak pár megát foglal (sparse fájlok kezelésére van a fájlkezelő eszközökben a -s paraméter, a valódi méret az ls -ls paranccsal ellenőrizhető), és csak akkor nő a valódi helyfoglalása, amikor feltöltjük adatokkal. Megjegyzem, hogy ez a technika mostanában vált egyre fontosabbá, ahogy a virtualizáció egyre komolyabb méreteket öltött, és a virtuális diszkek esetén a lyukas fájlok használata egyszerűen képes kiváltani az automatikusan növekvő méretű speciális formátumú virtuális diszk fájlokat. Hátránya, hogy nem minden állományrendszer támogatja.

Ahhoz, hogy ezt a fájlt blokkos eszközként használhassuk, a losetup parancsra lesz szükségünk. A következő parancs:

```
# losetup -f --show ~/uds/cfs.img
```

Az előbb létrehozott állományt hozzárendeli egy speciális hurokeszközhöz (loop device), és kiírja a felhasznált eszköz nevét. Mostantól tehát van a notebook diszkjén egy blokkeszközként használható tárolónk.

# 5. A RAID létrehozása

A tükröt nagyon egyszerű létrehozni az mdadm parancs segítségével:

```
# mdadm --create /dev/md42 --verbose \
   --name=cfs --level=mirror --raid-devices=2 \
   /dev/sdc1 /dev/loop0
```

Ennek az egyszerű módszernek azonban van néhány kellemetlen tulajdonsága. Vegyük sorra. Ha nem adok meg gépet az mdadm-nek a tömb létrehozásakor, akkor automatikusan az adott gép nevét írja be a tömb adatai közé. Velem előfordult, hogy a széfet egy ugyanolyan nevű gépbe betéve (épp új gépet telepítettem) a rendszer automatikusan hozzákapcsolta egy md eszközhöz, de még véletlenül sem ahhoz, amihez én szerettem volna. Így indítás előtt mindig le kellett állítani a másik md eszközt, és elindítani azt, amit én akartam. Ennek elkerülésére beállítom a tömb otthonát a –homehost paraméterrel valami olyanra, ami feltehetőleg nem lesz sehol gépnév. Ezért helyesírási hibás a név. A következő probléma a létrehozáskori szinkron. Mivel nekünk nincs semmilyen adat a két diszken, ezért teljesen felesleges idő és energia szinkronizálni őket. Ezért használom a –assume-clean paramétert.

Ha a széfet egy másik gépen használom, majd újra bedugom a saját gépembe, akkor az USB-n lévő RAID tag eltér majd a loop eszköztől, és ezért újra kell szinkronizálni őket. A teljes szinkron (tehát minden adatot bájtról-bájtra újraírni) nem kevés idő. Ennek elkerülésére használható az ún. bitmap, amely egy nyilvántartás arról, hogy mely szektorok vannak szinkronban. Leegyszerűsítve, mikor az

összes RAID tag szinkronba kerül, akkor a bitmapbe 0 kerül beírásra, ha valamelyik nincs szinkronban, például mert egy másik gépnél használom, és a loop tag nincs is jelen, akkor minden átírt szektorhoz 1 íródik. Mikor újra összeépítem a két tömbelemet, akkor a bitmap alapján csak a változások szinkronizálódnak, ami hihetetlenül meggyorsítja a szinkron folyamatát.

Van még egy probléma. Az USB tag sokkal lassabban olvasható (és írható, de ez most nem számít), mint a loop. Ezért jobb lenne, ha az USB-t csak akkor használná a rendszer írásra, ha nagyon muszáj. Erre használható a –write-mostly, a –write-behind pedig arra utasítja a rendszert, hogy bizonyos mértékig nézze el az eszköznek, ha lemarad az írással. Így tehát a tömb létrehozása a következő módon történik:

```
# mdadm --create /dev/md42 --verbose \
   --assume-clean --homehost=nowheree \
   --bitmap=internal --name=cfs \
   --level=mirror --raid-devices=2 \
   --write-mostly --write-behind /dev/sdc1 /dev/loop0
```

# 6. Blokkeszközök titkosítása, a LUKS

Amennyiben nagyobb biztonságot akarunk, akkor a bizonytalan tartalmú md eszközt először is fel kell tölteni véletlen adatokkal:

```
# dd if=/dev/urandom of=/dev/md42 bs=64M
```

Vigyázat! Ez nagyon sokáig fog tartani. Órákig. Próbaképpen csináltam egy 1G-s loop fájlt, feltöltöttem az urandom-ból, ez két és fél percig tartott. Ha nem lenne gyors gépem, akkor még sokkal tovább tartott volna (a pszeudo-random generálás nagyon CPU igényes). Ha tehát van egy külső 300G-s wincsink, akkor szánjunk rá jó pár órát. Lassúság tekintetében az USB írási sebessége életre-halálra szóló harcot vív az urandom eszköz olvasási sebességével. És most semmilyen ’lyukas fájl’ trükk nem menti meg a hátsónkat, ezt végig kell várni.

A Linux a dm_crypt segítségével lehetőséget ad a blokkeszközökön lévő adatok titkosítására. A korábban létrehozott RAID tömb blokkeszköze, a /dev/md42 könnyedén titkosítható a cryptsetup parancs segítségével. Az egyszerű módszer használata esetén beállítunk egy jelszót az eszköznek, utána létrehozzuk rajta a fájlrendszert, csatoljuk és használjuk azt, így később a jelszó ismerete nélkül nem lehet a rajta lévő adatokat megszerezni. Ez valahogy így néz ki:

```
# cryptsetup create cfs /dev/md42
```

Ezzel egy nagy baj van. Ha egyszer valaki beállított egy jelszót egy eszközre, akkor azt már nem lehet megváltoztatni. Mivel mi hosszan szeretnénk használni a széfet, így valami jobb megoldást kellene választani. Kézenfekvő, de nem túl

közismert megoldás a LUKS (Linux Unified Key Setup) kiegészítés használata. Ennek az az előnye, hogy a titkosított partícióra kerül egy szabványos adatterület, mely lehetővé teszi a kulcsok cseréjét, valamint több kulcs használatát is. Így készíthető akár egy biztonsági kulcs, ami nem megjegyezhető, de el lehet helyezni egy valódi páncélszekrényben, és gond esetén használható. A LUKS használata csak egy kicsit bonyolultabb, valahogy így néz ki:

```
# cryptsetup luksFormat /dev/md42
# cryptsetup luksOpen /dev/md42 cfs
```

# 7. Fájlrendszer választás

A széfre ext2 fájlrendszert választottam. Korábban azt írtam, hogy az USB flash tárolási technológiája miatt gondok lehetnek a sok átírással, de a gyakorlatban ezt a problémát nem sikerült igazolni (lásd az USB alfejezetben). A speciális flashre készült, írásterítő algoritmust is tartalmazó fájlrendszerek teljesen feleslegesen lassítanák a rendszerünket, és ráadásul számos hiányosságuk is van az ext fájlrendszerekhez képest. Tehát ext2. Azért nem ext3, mert a notebookokban van akkumulátor, tehát az áramkimaradástól nemigen kell tartani, én már vagy öt éve nem láttam nem szándékos fsck-t. Így a naplózó fájlrendszer használata hely és időpocsékolás. És végül, de első sorban, a naplófájl (journal) nagyon gyorsan frissül, sokszor íródik. Ha még sincs írásterítés az USB eszközön, akkor hamar kipurcantja az eszközt.

Az ext2-vel kapcsolatban csak annyi speciális beállítást célszerű tenni, hogy a létrehozáskor (mkfs.ext2) nullázzuk a rootnak fenntartott helyet (-m 0), és a csatoláskor (mount) úgy kell beállítanunk, hogy a felesleges írások számát minimalizáljuk. Ennek érdekében a hozzáférési idő állandó írogatásáról jó lebeszélni a rendszert, ezt részlegesen a relatime opció használatával tehetjük meg, de ez csak annyit csinál, hogy ha frissíti a módosítás idejét, akkor nem frissíti a hozzáférését. Ha nem akarjuk használni a hozzáférési időt (amire jó néhány éves Linux gyakorlatom alatt csak néhány nagyon speciális esetben volt szükség a napi használatban, a felhasználók többsége még csak le sem tudja kérdezni), akkor használjuk a noatime opciót.

# 8. Felhasználási javaslatok

Az uDS tipikus felhasználása a munka könyvtár (nem a home!) tárolása egy külső USB és egy belső loop eszközből álló titkosított RAID tükrön. Erre általában elegendő szokott lenni az a 8-16G-ás USB, ami még normális áron beszerezhető.

Érdemes még megfontolni a rendszer és a felhasználó fontosabb konfigurációs állományainak, levelező mappáinak és a böngésző fontosabb adattárainak a széfen tárolását. A rendszer fontos konfigurációs állományait én időnként egy kis shell szkript segítségével lementem. Ha a felhasználó adatainak védelmére akarja valaki használni, akkor nincs más dolga, mint átmásolni az adott, védendő fájlokat és könyvtárakat a széfre, majd szoft linkeket létrehozni hozzájuk. Ekkor azonban figyelni kell arra, hogy a széf bekapcsolása nélkül az adott program nem fog, vagy nem fog helyesen működni. A Firefox böngésző például hibaüzenet nélkül kilép, ha a /.mozilla egy link, de a könyvtár nem létezik. Én erre azt szoktam csinálni, hogy a széfen van egy könyvtár, abban vannak a konfigok, és ezt a könyvtárat létrehozom nem becsatolt állapotban is a csatolási pont alatt, a minimális konfigurációs fájlokkal.

# 9. Továbbfejlesztési lehetőségek

A rendszert néhányan már rendszeresen használjuk, de van még fejleszteni való. A következő lehetőségek merültek fel, mint továbbfejlesztési irányok:

- automatikus szinkron javítása,
- a grafikus felület tökéletesítése, státuszsor használata, deszktop integráció,
- többfelhasználós, többjelszavas üzemmód a LUKS segítségével,
- teljesen felhasználói jogú üzemmód kidolgozása,
- smartcard vagy USB token integrációja,
- dokumentáció,
- fittség, világuralom.

# Elérhetőségek, hivatkozások

A program aktuális változata letölthető a `http://www.fixme.hu/uds` címről.

Ha valakinek valami javaslata van a fent leírtakkal kapcsolatban, az kérem, írja meg a `mato.peter@andrews.hu` e-mail címre.

# Hivatkozások

[1] USB flash drive, Wikipedia: http://en.wikipedia.org/wiki/USB_flash_drive

[2] ext2 filesystem, Wikipedia: http://en.wikipedia.org/wiki/Ext2

[3] Howto use Cryptsetup with LUKS support:
http://feraga.com/library/howto_use_cryptsetup_with_luks_support_0

[4] Sparse files – what, why, and how:
http://administratosphere.wordpress.com/2008/05/23/sparse-files-what-why-and-how/

# A GeoGebra alkalmazása a matematikaoktatásban az általános iskolától az egyetemig

Papp-Varga Zsuzsanna

**Kivonat**

A GeoGebra egy olyan világszerte 190 országban ismert és elismert, nyílt forráskódú (GNU GPL v2 licensz alatti), platform független, magyar fordításban is elérhető dinamikus matematikai program, amely szinte minden korosztály oktatásában alkalmazható. Témájában kapcsolódik a geometriához, az algebrához, az analízishez és a statisztikához is. A GeoGebra egyrészt egy dinamikus geometriai program, másrészt pedig egy computer algebrai rendszer.

A GeoGebra egyik legfontosabb előnye, hogy segítségével különböző absztrakt fogalmakat szemléltethetünk a diákok számára. A szoftverrel készült segédanyagok lehetőséget adnak a diákoknak, hogy meglássák és megfogalmazzák a különböző reprezentációk közötti összefüggéseket. Nem utolsó sorban pedig a GeoGebra lehetőséget ad a diákok számára a kísérletezgetésre, a felfedező tanulásra is.

## Tartalomjegyzék

**1. A GeoGebra program rövid bemutatása** **86**

**2. Első lépések a GeoGebra program használatában** **86**
2.1. A GeoGebra felhasználói felülete . . . . . 86
2.2. Az alapfunkciók ismertetése egy példán keresztül . . . . . 87

**3. A GeoGebra oktatásban való alkalmazásnak lehetőségei** **88**

**4. A GeoGebra közösség** **90**

# 1. A GeoGebra program rövid bemutatása

A GeoGebra egy olyan dinamikus matematikai program melyet készítője Markus Hohenwarter eredetileg középiskolai oktatási segédletnek szánt, de azóta már szinte minden korosztály oktatásában sikerrel alkalmazzák. Világszerte 190 országban ismerik és 46 nyelvre fordították le. Az évek során számtalan nemzetközi díjjal jutalmazták. Sikerét többek között annak köszönheti, hogy open-source és tetszőleges Java futtatására alkalmas platformon telepíthető. Legfontosabb előnye azonban talán mégis az, hogy használatát, az alap funkcióinak működését szinte bárki pár óra alatt el tudja sajátítani.

A GeoGebra témájában kapcsolódik a geometriához, az algebrához, az analízishez és a beépített táblázatkezelőnek köszönhetően már a statisztikához is. A GeoGebra egyrészt egy dinamikus geometriai program, másrészt pedig egy computer algebrai rendszer. Talán legfontosabb tulajdonsága, hogy összekapcsolja az objektumok különböző reprezentációit, azok geometriai megjelenését és algebrai leírását.

Az, hogy a GeoGebra egy dinamikus szerkesztő rendszer, azt jelenti, hogy a felhasználó tulajdonképpen kap egy virtuális szerkesztőkészletet a kezébe. A papíron végzett szerkesztésektől eltérően viszont a programban a kiinduló objektumok (pontok, függvények, stb.) szabadon mozgathatók úgy, hogy a tőlük függő objektumok a geometriai kapcsolatok alapján velük együtt mozognak.

A GeoGebra másrészt egy computer algebrai rendszer, amiben az objektumok algebrai úton adhatók meg (pontok koordinátáikkal, egyenesek egyenleteikkel, függvények képletükkel, stb.). Az objektumokkal különböző számítások is végezhetők, például meghatározható a függvények deriváltja és integrálja is.

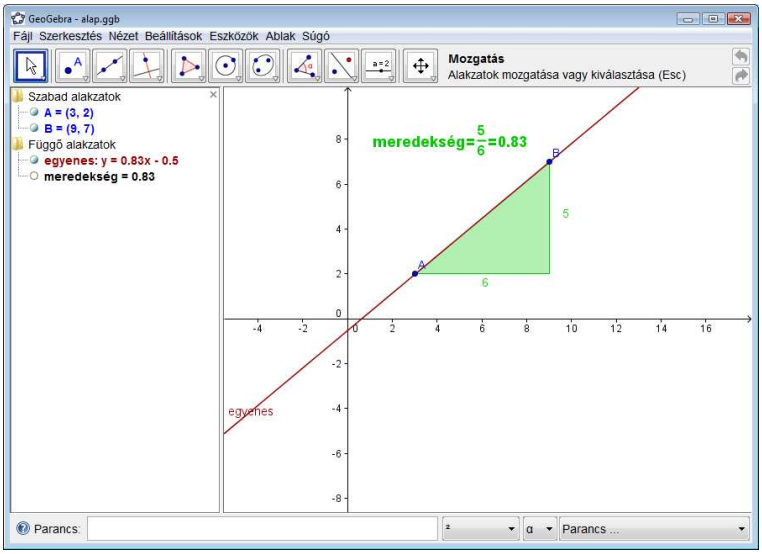
A program legújabb 3.2-es verziójában a geometria és az algebra ablakon kívül már egy táblázatkezelő is megtalálható, melynek segítségével további távlatok nyílnak a GeoGebra használatában, mint például a már említett statisztikai témakörben való alkalmazási lehetőségek.

# 2. Első lépések a GeoGebra program használatában

## 2.1. A GeoGebra felhasználói felülete

Az ablak tetején található a menü, melyben az olyan alapfunkciókat, mint mentés és megnyitás, valamint a szerkesztés egészét érintő beállításokat, mint például a nézet testre szabása tudjuk elérni. A menüsor alatti gombok segítségével tudunk szerkeszteni. Egy-egy gombbal egy egész funkciócsoportból tudunk választani, ha a gomb jobb alsó sarkában lévő kis nyílra kattintunk. Az ablak bal oldalán található az algebra ablak, jobb oldalán pedig a geometria ablak vagy más néven a

rajzlap. Az algebra ablakban láthatjuk az objektumok algebrai leírását, a rajzlapon pedig geometriai reprezentációjukat. Az ablak alján található parancssor segítségével pedig különböző alakzatokat definiálhatunk, számításokat végezhetünk (1. ábra).




1. ábra. A GeoGebra felhasználói felülete

## 2.2. Az alapfunkciók ismertetése egy példán keresztül

Az elkészítendő feladat a háromszög körülírható körének megszerkesztése. Geometria példáról lévén szó, nincs szükség koordinátákra, ezért első lépésként érdemes a Nézet menüben kikapcsolni az algebra ablak és a tengelyek megjelenítését.

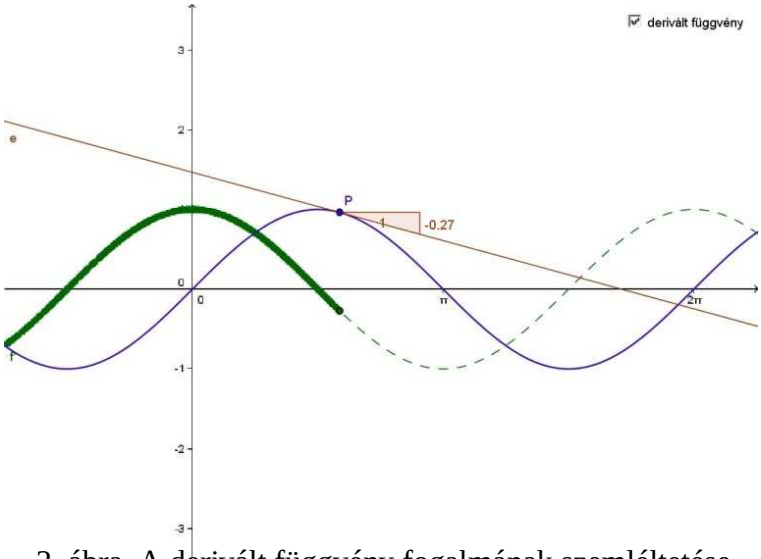
Először fel kell vennünk a háromszöget a Sokszög szerkesztési funkció segítségével úgy, hogy a rajzlapon kattintással kijelöljük a három csúcspont helyét, majd az első csúcspontra kattintunk. Ezt követően meg kell szerkeszteni az oldalfelező merőlegeseket, amit többféleképpen is megtehetünk. Egyik lehetőség, hogy a Szakaszfelező gomb kiválasztása után rákattintunk a megfelelő oldalra, vagy két csúcsra, a másik lehetőség pedig, hogy beírjuk a parancssorba a Szakaszfelező parancsot, melynek szögletes zárójelben megadott paramétere az egyik oldal, vagy két csúcs. A kör középpontját úgy tudjuk meghatározni, hogy a Két alakzat metszéspontja funkció kiválasztása után két oldalfelezőre kattintunk. Utolsó lépésként a Kör középponttal és kerületi ponttal gomb kiválasztását követően pedig megadjuk az előbb megszerkesztett pontot és az egyik csúcspontot.

Fontos, hogy ha mozgatni akarjuk az objektumokat, akkor mindig a gombsor baloldalán lévő nyílra kell kattintanunk (Mozgatás funkció). Bármely hibás

lépésünket vissza tudjuk vonni a Szerkesztés menü Visszavonás funkciójával. Az objektumok összes tulajdonságát pedig be tudjuk állítani a Szerkesztés menü Tulajdonságok pontjával megnyíló ablakban.

# 3. A GeoGebra oktatásban való alkalmazásnak lehetőségei

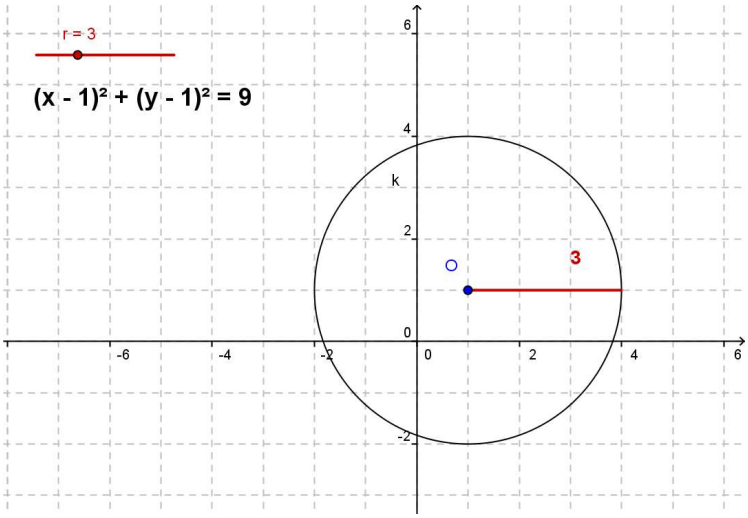
A szoftver segítségével különböző műveleteket és absztrakt fogalmakat szemléltethetünk a diákok számára, oly módon, ahogy hagyományos eszközökkel csak nehezen, vagy egyáltalán nem lehetséges. Például általános iskolában szemléltethetjük az egész számok összeadását, a törtek szorzását, középiskolában a lineáris függvények paramétereinek jelentését, vagy egyetemen a derivált függvény fogalmát (2. ábra).




2. ábra. A derivált függvény fogalmának szemléltetése

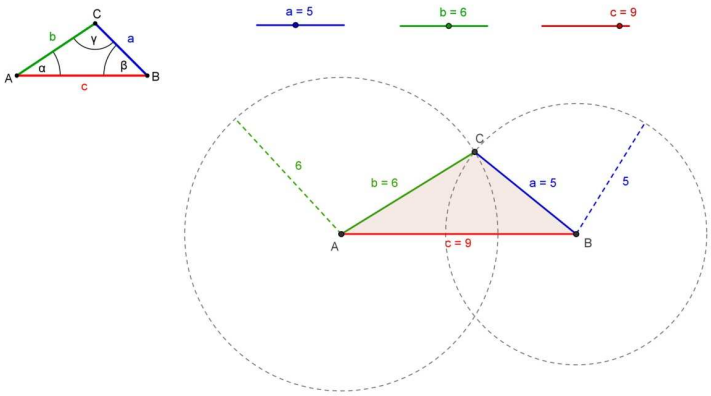
A GeoGebra továbbá lehetőséget ad a diákoknak, hogy meglássák és megfogalmazzák a különböző matematikai reprezentációk közötti összefüggéseket. Például, középiskolában a kör egyenlete és képe közötti (3. ábra), a felsőoktatásban pedig a komplex számok algebrai és geometriai reprezentációja közötti összefüggéseket.

A GeoGebra segítségével a diákoknak lehetőségük nyílik a felfedező tanulásra. Például megvizsgálhatják a kép importálási funkció segítségével, hogy van-e egy virágnak szimmetriatengelye vagy kipróbálhatják, hogy milyen oldalhosszak ese-




3. ábra. A kör egyenlete

tén szerkeszthető meg egy háromszög (4. ábra) vagy akár azt, hogy hogyan változik az alsó összeg értéke, ha növeljük a felosztások számát.




4. ábra. A háromszög szerkeszthetőségének vizsgálata

A GeoGebra a tárgyi feltételek és a módszertani célok függvényében több módon is alkalmazható az oktatásban. Egy számítógép és egy projektor segítségével szemléltethetünk az egész osztálynak, ami interaktív táblát használva talán még könnyebben követhető a diákoknak. Ha lehetőségünk nyílik gépteremben dolgozni, akkor minden tanuló önállóan (vagy párban/csoportban) dolgozhat, s akár mindenkinek különböző interaktív feladatlapot készíthetünk. Egy nem elhanyagolható lehetőség – főleg azok számára, akiknek az előbbiek nem megvalósíthatók -, hogy a GeoGebrával készült dinamikus segédanyagok, interaktív feladatlapok könnyedén publikálhatók az interneten is. Fontos megemlíteni, hogy a különböző helyzetekben alkalmazott segédanyagoknak más és más feltételeknek kell megfelelniük, amiről a hatékonyság érdekében nem szabad elfeledkezni.

# 4. A GeoGebra közösség

A GeoGebra weboldalnak (`http://www.geogebra.org/`) havonta 400.000 látogatója van. A főoldalon megtalálható a programmal kapcsolatos összes fontosabb információ. A wiki oldalon (`http://www.geogebra.org/wiki`) pedig több ezer felhasználó által feltöltött segédanyag érhető el, amiket bárki szabadon alkalmazhat az oktatási munkájában. Amennyiben valakinek kérdése van, egy regisztrációt követően a fórumban (`http://www.geogebra.org/forum`) teheti azt fel.

A GeoGebra csapat, felismerve az igényeket, 2007 decemberében létrehozta az International GeoGebra Institute-ot (`http://www.geogebra.org/IGI`). Az intézet három fő célkitűzése:

- továbbképzések szervezése és folyamatos segítségnyújtás tanárok részére,
- segédanyag- és szoftverfejlesztés, valamint
- módszertani kutatások megtervezése és koordinálása.

Az IGI megalapítását követően több lokális GeoGebra intézet is alakult, hogy helyi szinten valósítsák meg az IGI célkitűzéseit. A Magyar GeoGebra Intézet megalapítása is a közeljövőben várható.

# Linux vékonykliens megoldások

Papp Zsolt

**Kivonat**

A vékonykliens egy olyan számítógép, amely hálózati operációs rendszer futtatására képes, és nem rendelkezik olyan, a felhasználó által módosítható adat tárolására alkalmas belső tároló eszközzel, amely hosszútávú adattárolásra képes. Nem rendelkezik felesleges hardverkomponensekkel, és amelyekkel rendelkezik, azok is csak akkora teljesítményűek, amekkorára a felhasználási környezetében éppen szükség van. Ez ténylegesen töredéke a megszokott asztali munkaállomások teljesítményének, mivel az adatok feldolgozása nem ezen a gépen történik, hanem egy sokkal erősebb kiszolgálón, és a vékonykliensen csak a felhasználói felület fut. Sokrétűsége miatt egy Linux elindítható egy belső kis méretű és kapacitású flash alapú meghajtóról vagy akár hálózatról is, azaz ideális szoftverkörnyezetet nyújthat ilyen eszközökhöz.

## Tartalomjegyzék

**1. Bevezetés** **92**

**2. Vékonykliens megoldások** **92**
2.1. X Display Manager Control Protocol . . . . . 93
2.2. FreeNX . . . . . 94
2.3. KIWI-LTSP . . . . . 94
2.4. SLETC . . . . . 95

**3. Hibridkliens** **95**

**4. SUSE Studio** **96**

**5. Összefoglalás** **98**

# 1. Bevezetés

A Linux a flexibilitásának köszönhetően egyre elterjedtebb platform. A kiszolgálópiacon már nagyon régóta jelen van, a beágyazott rendszerek körében is igen közkedvelt, és az asztali munkaállomásokat is régóta ostromolja. Ez utóbbi piacon eddig kicsit kevesebb sikerrel.

A vékonykliens egy olyan számítógép, amely hálózati operációs rendszer futtatására képes, és nem rendelkezik olyan, a felhasználó által módosítható adat tárolására alkalmas belső tároló eszközzel, amely hosszútávú adattárolásra képes. Nem rendelkezik felesleges hardverkomponensekkel, és amelyekkel rendelkezik, azok is csak akkora teljesítményűek, amekkorára a felhasználási környezetében éppen szükség van. Ez ténylegesen töredéke a megszokott asztali munkaállomások teljesítményének, mivel az adatok feldolgozása nem ezen a gépen történik, hanem egy sokkal erősebb kiszolgálón, és a vékonykliensen csak a felhasználói felület fut.

Sokrétűsége miatt egy Linux elindítható egy belső kis méretű és kapacitású flash alapú meghajtóról vagy akár hálózatról is. Minden feltétel adott, hogy megfelelő alap legyen egy vékonykliens környezet kialakításához.

# 2. Vékonykliens megoldások

A vékonykliens környezet többnyire kis fogyasztású egységesített hardverparkot és központi szoftvermenedzsmentet jelent. A felhasználók alkalmazásai a kiszolgálóoldalon, nagy teljesítményű gépeken futnak, amelyek több felhasználó egyidejű kiszolgálására képesek. Ezért a kliensoldalon többnyire elegendő egy alacsony fogyasztású eszköz, billentyűzet, monitor, esetleg hangszóró.

A vékonykliensnek számtalan előnye van, ami miatt napjaink közkedvelt megoldása, és egyre több asztali környezetet vált ki kisebb és nagyobb vállalatoknál is. Ilyen előnyök például a következők:

- alacsonyabb adminisztrációs költségek,
- alacsonyabb hardverköltségek,
- kisebb energiafogyasztás,
- egyszerűbb a szervizelés,
- egyszerűbb a rendszer biztonságos üzemeltetése.

A Linux Terminal Server Project[1] olyan szolgáltatásokkal egészíti ki a Linux disztribúciókat, amelyekkel azok képesek lesznek kiszolgálóként működni egy vékonykliens környezetben. Régebben ez egy telepíthető szoftvercsomag volt, jelenleg inkább csak egy koncepció vagy ajánlás, amit az egyes disztribútorok saját eszközeiket felhasználva valósítanak meg, de az alapjai minden disztribúcióban nagyon hasonlóak. Pont ezért egy vékonykliens környezet kialakításához több megoldás közül is választhatunk.

Egy vékonykliens környezet működéséhez alapvetően kiszolgálóra és vékonykliensre van szükség. A vékonykliensen egy pehelysúlyú Linux-rendszer fut, amely képes kezelni a perifériákat, és futtatja a felhasználói felületet, amely ebben az esetben X11. A kiszolgálóoldal sokkal összetettebb, általában az alábbi szolgáltatásokat kell futtatnia:

- DHCP
- TFTP
- NFS/NBD/AoE
- XDMCP

A hálózatról induló vékonykliensek támogatják a PXE indítást, amihez a DHCP és TFTP szolgáltatások szükségesek. Az induló operációs rendszer a gyökérkötetet NFS-megosztáson keresztül csatolhatja fel, esetleg Network Block Device-ot vagy ATA over Ethernet protokollt használhat. Ezekről a szolgáltatásokról itt most nem esik szó, mert sokkal fontosabb az, hogyan áll elő a vékonykliensek által futtatott operációs rendszer, illetve az, hogy milyen módon kerül megvalósításra a képernyő átvitele.

## 2.1. X Display Manager Control Protocol

Az XDMCP az X11 saját protokollja, amelyet egy grafikus terminál használhat távoli képernyő átvételére. A protokollnak több elterjedt implementációja van, például az eredeti XDM vagy a két nagy asztali környezethez tartozó bejelentkezéskezelő (display manager), a GDM és a KDM.

Az XDMCP protokoll ugyan támogat hitelesítést az X-kiszolgáló felé, de a kommunikációs csatorna nincs titkosítva, ezért nem ajánlatos olyan környezetben használni, ahol a hálózat lehetőséget ad a lehallgatásra. Ilyenkor a legjobb megoldás, hogyha egy SSH-alagúton keresztül érik el a felhasználók a grafikus munkamenetet.

[1] LTSP – `http://www.ltsp.org`

További hátránya ennek a rendszernek az, hogy a billentyűzeten, egéren és a képernyőn kívül más osztályú perifériát nem képes továbbítani. Tehát hogyha a vékonykliensen a gazdag tartalommal ellátott weboldalak hangjait kell megszólaltatni vagy az USB-meghajtónkat kell felcsatolni, akkor ezekhez valamilyen kiegészítő szolgáltatásra van szükség.

## 2.2. FreeNX

A FreeNX az NX (NoMachine) technológia egyik ingyenes és nyílt forrású implementációja, amely szintén a távoli grafikus munkamenetet képes megvalósítani. Arra optimalizálták, hogy alacsony sávszélességű hálózaton is képes legyen jó teljesítményt nyújtani a natív X protokollhoz képest, emiatt alacsonyabb átviteli sebességgel rendelkező kapcsolaton keresztül is használható marad. A távoli kapcsolatokat SSH-alagúton vezeti keresztül, ezzel növelve az ezen keresztül áthaladó adatok biztonságát.

Az NX a kapcsolatok tömörítésére az DXPC által használt eljárást fejlesztették tovább, hogy minimalizálják a kapcsolatok sávszélességét. A protokoll ezen kívül nagy mértékben használ gyorsítótárat, hogy a grafikus munkamenet sokkal alacsonyabb válaszidőket produkáljon, ez nagy mértékben képes fokozni a felhasználói élményt. Egy már korábban megjelenített és eltárolt ablakrészletet a rendszer sokkal gyorsabban képes újra megjeleníteni.

Az NX protokollt alapvetően az X11 optimalizálására tervezték, ennek ellenére képes RDP, illetve VNC protokollt is támogatni, gyorsítani.

Azon felül, hogy lehetővé teszi a lassú kapcsolattal rendelkező felhasználók számára a grafikus képernyő átvételét, az NX támogatja a munkamenet felfüggesztését és folytatását is. A felfüggesztett munkamenetben a grafikus alkalmazások tovább futnak, és később a felhasználó bármikor újra csatlakozhat a korábban felfüggesztett munkamenethez.

Az NX protokollt GPL-licenc védi, de a technológia megalkotója, a NoMachine kereskedelmi termékeket épít rá.

## 2.3. KIWI-LTSP

Az openSUSE megoldása az úgynevezett KIWI-LTSP, amely a KIWI rendszerképkezelő alkalmazást integrálva kibővített szolgáltatásokat nyújt. A KIWI a következő újdonságokat nyújtja az LTSP5 funkcióin felül:

- boot és kliens rendszerképek létrehozása;
- előre elkészített rendszerképeket tartalmaz, a felhasználóknak nem kell sajátot készíteni;

- a KIWI különböző típusú rendszerképeket tud létrehozni, például live CD-t, vagy USB-meghajtóról induló rendszert, melyek képesek bármilyen LTSP5 rendszerű környezetben működni;
- a kiwi-ltsp-script minden eszközt tartalmaz a rendszerképek elkészítéséhez és a szükséges szolgáltatások beállításához;
- az NFS és az NBD protokollon kívül támogatja az AoE protokollt is.

A KIWI integrációjával új irányból lehet megközelíteni a vékonykliens rendszereket. Ahogyan minden rendszernél, úgy itt is szükség van a szokásos szolgáltatásokra (PXE, dhcp, tftp stb), amelyek beállításában segítséget nyújt a kiwi-ltsp-script. Ami az újdonság, az a vékonyklienseken futó operációs rendszer telepítésének, karbantartásának és kezelésének a módja. A KIWI képes egy sablon alapján telepíteni egy operációs rendszert különböző formátumokban, ami azt jelenti, hogy a vékonyklienseken futó operációs rendszerből képes akár live CD-t, USB-meghajtóról induló változatot készíteni, vagy lehetőség van merevlemezre történő telepítésre is.

További információ és letöltési lehetőség a `http://en.opensuse.org/Ltsp` címen.

## 2.4. SLETC

A SUSE Linux Enterprise Thin Client egy SUSE Linux Enterprise Desktop 11 és KIWI-LTSP alapú megoldás, amely elérhetővé teszi a Linux alapú vékonykliens rendszereket a nagyvállalatok számára. A megoldás hasonló funkciókat nyújt, mint a KIWI-LTSP, azzal a különbséggel, hogy a Novell támogatást nyújt hozzá. További információ a `http://www.novell.com/products/thinclient/` címen érhető el.

# 3. Hibridkliens

A kezdeti lelkesedés a vékonykliensek irányába megtorpant, illetve tovább szegmentálódott, amikor megjelentek a netbook, illetve nettop elnevezésű eszközök. Ezek az eszközök teljes értékű számítógépeknek felelnek meg, áruk és fogyasztásuk azonban sokkal jobban közelít a vékonykliensekéhez, mint a megszokott asztali munkaállomásokéhoz vagy hordozható számítógépekéhez. A hibridkliens ezen két típusú megoldás közötti szakadékot igyekszik kitölteni.

Ezzel egybeestek olyan gazdasági hatások, amelyeek hatására a felhasználók árérzékenysége megnőtt, elkezdtek terjedni a böngésző alapú alkalmazások, és

elindult az energiatakarékos adatközpont-koncepció, amely a GreenIT és a virtualizáció megoldásaiban csúcsosodott ki.

Egy vékonykliens rendszer kialakításánál a munkaállomáson lévő teljesítmény a kiszolgálókon összpontosul. Ez azt jelenti, hogy az adatközpontok teljesítményigénye, áramfelvétele, hűtési igénye és elfoglalt területének nagysága ugrásszerűen megnőtt. Cserébe valóban teljesen központi felügyelet oldható meg, hiszen a vékonykliensek számottevő feladatot nem látnak a beviteli eszközök, a képernyő és a hálózatkezelésén kívül.

A hibridkliens koncepciója az, hogy a felhasználónál megjelenő szolgáltatásokat – legyen az operációs rendszer, vagy alkalmazás – olyan módon kerüljenek megosztásra, hogy az felesleges költségeket ne okozzon. Érdemes elgondolni, hogy mekkora teljesítményre van szükség, ha 100 konkurens felhasználó egyszerre kezd el böngészni az interneten Firefoxot használatával. Ha ezek a felhasználók olyan tartalmat is meg kívánnak tekinteni, amely processzorigényes (pl. flash), akkor az adatközpont költségei indokolatlanul megugorhatnak, még azelőtt, hogy a klasszikus alkalmazások futtatása szóba került volna.

Logikusnak látszik, hogy egyfajta ésszerű teljesítmény-egyensúly kialakítása az ideális a „mindent az adatközpontba”, illetve a másik véglet, a „mindent a munkaállomásra” koncepció mellett. A hibridkliens elgondolásnál az operációs rendszer, a hitelesítés, a böngésző, a multimédiatámogatás, a perifériák kezelése, esetlegesen az irodai alkalmazások is a hibridkliensre kerülnek, amíg az adatközpontba kerül az azonosítás, az adattárolás, az egyedi alkalmazások és azok felügyelete.

A hibridkliensre tervezett szolgáltatásokat egy netbook számítógép gond nélkül képes elfuttatni, mi több az alacsony teljesítményű munkaállomások is újra hadrendbe foghatók ezzel a megoldással. A hibrid technológia annyira rugalmas, hogy bármikor bármilyen alkalmazást lehetséges a munkaállomás oldalról a kiszolgálóoldalra helyezni és vissza. Ezzel szemben egy vékonykliens környezetnél egyetlen irány van, hiszen a vékonykliensek fizikailag képtelenek többre, mint amire tervezték őket.

A rugalmasság nem csak abban merül ki, hogy a szoftverkomponensek szabadon mozgathatók, de olyan hardvereszköz is helyet kaphat ebben az infrastruktúrában, amely nem rendelkezik merevlemezzel, de CD/DVD-meghajtóval vagy USB-csatolóval igen. Ezekben az esetben az operációs rendszer és annak komponensei CD/DVD-ről vagy pendrive-ról is betöltődhetnek.

# 4. SUSE Studio

Gondot okozhat egy hibridkliensre telepítendő operációs rendszer kialakítása, főleg akkor, ha számtalan megvalósítást kell egyszerre bevetni: lehet, hogy CD/DVD-

ről, pendrive-ról, vagy merevlemezről kell indítani az operációs rendszert és a hibridkliens architektúrában meghatározott további alkalmazásokat. Ez hatalmas munkát jelent, még akkor is, ha közös kódbázisról lehet dolgozni, hiszen egy teljes disztribúciókészítő környezetet kell üzemben tartani.

Erre a problémára lehet válasz a SUSE Studio, amely egy egyszerű, gyors, böngészőn keresztül használható appliance-készítő rendszer. Akár egy virtuális gépet is kezelhetünk vele a TestDrive nevű funkción keresztül.

Az appliance kifejezés egy előre beállított szolgáltatást jelent egy operációs rendszerrel összecsomagolva. Ezt a csomagot egy fizikai kiszolgálóra másolva teljesen beállított, működőképes rendszert kapunk. Az appliance típusa lehet virtuális is, ekkor a csomagot egy virtualizációs környezetbe kell másolni. Ezeket a speciális appliance-eket hívjuk virtual appliance-nek. Egyre sűrűbben lehet találkozni letölthető appliance-ekkel, például egy nehezen telepíthető vagy beállítható szolgáltatás esetében.

Az appliance előnyei:

- könnyen futtatható,
- kicsi méret,
- nem tartalmaz felesleges szoftverösszetevőket,
- könnyű karbantartani.

A SUSE Studio esetében a fejlesztők a könnyen kezelhetőséget tartották szem előtt, amikor az alkalmazás készült. Az első belépés után a sablonok használatával nagyon gyorsan létre lehet hozni egy appliance-et. A webes felületen az appliance-ek teljes körű beállítása elvégezhető:

- tárolók és csomagok telepítése és eltávolítása;
- nyelvi beállítások, indítási opciók, adatbázis szolgáltatás beállítása, háttértárkezelés és egyéb kinézettel kapcsolatos beállítások, például háttérkép beállítása, indítási háttérkép stb.;
- egyéb fájok elhelyezése az appliance-ben telepítés után;
- az appliance formátumának kiválasztása: lemezkép, live CD/DVD, Xen vagy VMware lemezkép;
- a TestDrive funkció segítségével az appliance elindítható, kipróbálható és módosítható, és ez is csak egy böngészőt igényel;

- a TestDrive során eszközölt változtatások integrálhatóak az appliance-be, akár egyenként is;
- az elkészült appliance ezután letölthető és használható.

A lemezkép típusú appliance-ot kiajánlhatjuk AoE vagy NBD protokollon, amit a vékonykliens hálózaton keresztül elindíthat.

A SUSE Studio jelenleg egy interneten keresztül elérhető szolgáltatás, amely mindenki számára szabadon hozzáférhető, és hamarosan elindul egy ehhez kapcsolódó szolgáltatás, ahol az elkészített rendszerképeket lehet cserélni továbbfejleszteni, módosítani stb. Másik fejlesztési irány a stand-alone SUSE Studio, amikor ezt a jelenleg szolgáltatásként működő rendszert a nagyvállalat izoláltan, saját infrastruktúrán belül képes használni.

# 5. Összefoglalás

A klasszikus vékonykliens megoldások után rendkívül izgalmas fordulatot eredményezett az, hogy a Linux ennyire sokoldalúan képes alkalmazkodni a gazdasági és piaci igényekhez, megreformálva és újragondolva a lehetőségeket. Olyan megoldások fognak a közeljövőben megjelenni, amelyek ötvözik a szolgáltatás alapú technológiákat, a virtualizációt, és nagyfokú testre szabhatóságot biztosítanak az informatikai infrastruktúrán belül.

# Egy irtó jó spamszűrő

Sütő János

**Kivonat**

Már évek óta fejlesztem (és használom is!) a clapf spamszűrőt, és már szinte el is felejtettem, hogy mi a spam. A cikkben bemutatom a program jellemzőit, funkcióit, néhány számadattal szemléltetem a képességeit. Kitérek arra is, hogyan illeszthető be a legkülönfélébb környezetbe, továbbá ismertetek néhány teljesítményhangolási tippet, végül röviden vázolom a program jövőjét.

## Tartalomjegyzék

**1. Miért érdemes kipróbálni? 100**

**2. Nagy teljesítmény 100**

**3. Egyszerű használat 101**

**4. A távirányító is szériatartozék 101**

**5. Házirend 102**

**6. Akció közben 102**

6.1. Kis cég . . . 103
6.2. Nagyobb cég . . . 103
6.3. Még nagyobb cég . . . 103
6.4. Microsoft környezetben . . . 103
6.5. Nagyvállalati konfiguráció . . . 104

**7. Teljesítményhangolás 105**

# 1. Miért érdemes kipróbálni?

Ha csak egy okot mondhatnék, hogy miért érdemes statisztikai spamszűrőt használni, akkor a pontosságot nevezném meg. Amíg a legtöbb kereskedelmi termék 95-99%-os pontosságot ígér, addig a clapf akár egy nagyságrenddel jobbat képes nyújtani. A hivatalos marketingszövegem alapján – megfelelő tanítás után – a fals pozitív hibák (**H**am **S**trike **R**ate, HSR) aránya nem több 0,1%-nál, míg a spam felismerés (**S**pam **H**it **R**ate, SHR) könnyen eléri a 99,5%-ot.

Idén augusztusban 2619 jó levélből 2 regisztráció-visszaigazoló levél akadt fenn a szűrőn, ami $100 \cdot 2/2619 = 0,07\%$-os HSR-t jelent. 1440 spamet is kaptam, amelyek közül 12 csúszott át, így a spam felismerés aránya $100 \cdot 1428/1440 = 99,17\%$. Ez elmarad ugyan a hirdetett 99,5%-tól, de meg tudom magyarázni. Az augusztusi gyengébb eredmény annak a következménye, hogy kikapcsoltam az RBL listákat. Arra voltam kíváncsi, hogy a statisztikai elv önmagában mire képes, és néhány levelet pluszban meg kellett tanulnia. Szeptemberre azonban beérett az eredmény, és a clapf rekordot döntött: 1506 spamet kaptam, és csak 1 csúszott át, így a spam felismerés aránya $100 \cdot 1505/1506 = 99.93\%$. A 2913 jó levél mindegyikét helyesen kategorizálta, azaz a fals pozitív hibaarány 0% volt, az összesített pontosság értéke pedig $100 \cdot (1505 + 2913)/(1506 + 2913) = 99.97\%$.

Ilyen pontosságot azonban csak a felhasználók bevonásával lehet elérni, és ebben a tanuló szűrők a legjobbak: a felhasználó ízlésének és a spammerek trükkjeinek ki-, illetve felismerésében.

# 2. Nagy teljesítmény

A program C-ben írt, optimalizálva fordított, így gyorsabb és kisebb az erőforrásigénye, mint az interpreter nyelveken készített megoldásoké. Aki nagyobb méretű levelezés felett őrködik, bizonyára beleütközött már a népszerű SpamAssassin teljesítményproblémáiba. A clapf sokkal gyorsabban dolgozik, és kevésbé terheli a gépet.

Az alábbi tesztkörnyezetben hasonlítottam össze az amavis és a clapf teljesítményét. Egy Windows-os PC-n futtatott VMware alatt (512 MB memóriát rendelve a VM-hez) a spamszűrők fogadták SMTP protokollal a leveleket, amelyeket feldolgozás után a localhoston futó postfixnek továbbítottak. A spamszűrőkre egy SSH tunnelen keresztül küldtem 10 szálon keresztül összesen 9700 levelet (fele jó levél, fele spam), és mértem az áteresztőképességet, hogy meddig tart a levelek továbbítása. Mind az amavis, mind a clapf csak spamszűrést végeztek, vírusellenőrzést ezúttal nem, illetve az amavis csak heurisztikus vizsgálatot végzett, tehát nem fordult feketelistákhoz, és a statisztikai modulját sem használta.

A clapf átlagosan 525 ms alatt végzett egy levéllel, az amavis pedig 3,43 má-

sodperc alatt, ami 6-szoros különbséget jelent a clapf javára. Ha mind az amavis, mind pedig a clapf adataiból levonjuk azt az időt, amíg a levelet fogadta, illetve továbbította SMTP-n (kb. 375, illetve 105 ms), akkor azt kapjuk, hogy a clapf átlagosan 45 ms alatt dolgozott fel egy levelet, míg az amavis 2,95 másodpercig. A clapf tehát kb. 65-ször gyorsabb. Végül ha a szokásos telepítést vesszük alapul, ahol a postfix fogadja a leveleket, és a localhoston gyakorlatilag néhány ms alatt átadja a leveleket a tartalomszűrőnek, akkor a clapf kb. 20-szor gyorsabb a gyakorlatban az amavisnál. A clapf teljesítménye kb. 61K levél óránként, azaz 1,5 millió levél naponta.

Ha a clapf spamszűrőt egy dedikált gépre telepítettem, akkor másodpercenként 55 levelet volt képes feldolgozni egy kommersz Core2 Duo-s desktop PC-n, ami napi 4,75 millió levelet jelent.

Hogy jobban szemléltessem ezeket a számokat, vegyük összehasonlításként a freemail levelezését, amely 2007-ben napi 18 millió levelet dolgozott fel több, mint 60 darab (egy fotó alapján) HP DL 145-ös géppel, amiből 18 csak a spamszűréssel foglalkozott. Ezt a terhelést ki lehet(ett) volna szolgálni 4 db Core2 Duo-s PC-vel vagy 12 db Windows-os desktop PC-vel is, ha azok VMware alatt clapf spamszűrőt futtatnak.

# 3. Egyszerű használat

A tervezés során kiemelt szerepet kapott a könnyű használhatóság és a majdnem zéró adminisztrációs igény. A telepítés és a kezdeti tanítás után a clapf minimális karbantartást igényel. Az volt a cél, hogy akár a nagymamám is könnyedén használhassa. A felhasználóknak tehát nem kell az adminisztrátorokat zaklatni, mindent képesek önmaguk elintézni, pl. a karantén kezelését, jelszócserét, fehérlista beállításokat és a tanítást. Ez utóbbi egyébként nagyon egyszerű: a fals pozitív hibákat a ham@domain, míg a fals negatív hibákat a spam@domain címre kell továbbítani mellékletként.

# 4. A távirányító is szériatartozék

Nem csak a TV-hez, de a clapfhoz is jár távirányító. A webui (=web user interface) egy opcionális kiegészítő, ahol egy böngészőből, néhány kattintással lehet elvégezni az adminisztratív műveleteket, pl. felhasználók, domainek, email címek és házirend csoportok kezelését. A felhasználók bejelentkezés után megtekinthetik a karanténjukat (mindenki szigorúan a sajátját, kivéve ha adminisztrátor), módosíthatják a fehérlistájukat, megnézhetik, illetve elengedhetnek leveleket a karanténból, sőt egy röptében generált képen a ham/spam arányt is nyomon követhetik.

Az adminisztrátorokat érdekelheti az egyes levelek sorsa. Ez könnyen megoldható néhány 'grep' paranccsal. A webui azonban egy AJAX-os táblázat segítségével megmutatja egy levél útvonalát a rendszerben. Bizonyos mezőkre húzva az egeret, részletes információk jelennek meg egy ballonban, pl. queue azonosítók, message-id-k,stb.

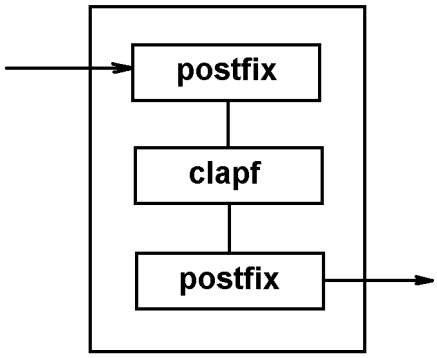
A webui az MVC (model-view-controller) módszer alapján készült. Ez azt jelenti, hogy ha valakinek nem tetszik a kezdetleges HTML dizájneri képességem, akkor lecserélheti az alapértelmezett témát. A webui használatához mindössze egy PHP-t futtatni képes webkiszolgálóra van szükség.

# 5. Házirend

A 0.4-es verziótól a clapf képes a beállításokat akár felhasználónként is testre szabni. Ez a gyakorlatban azt jelenti, hogy a konfigurációs beállításokat felül lehet definiálni, és a program viselkedése megváltoztatható. Megadhatjuk pl. hogy alkalmazzon-e egyáltalán spamszűrést, hol húzzuk meg a spam határt, használjon-e feketelistákat, jelölje-e meg a levél tárgy sorát, szükségünk van-e karanténra. Természetesen az is megoldható, ha az egyik felhasználó nem akar találkozni a spam levelekkel, míg a másik a saját gépén akarja egy külön mappába gyűjteni a kéretlen leveleket. A házirendek segítségével nagyon rugalmas konfigurációkat készíthetünk.

# 6. Akció közben

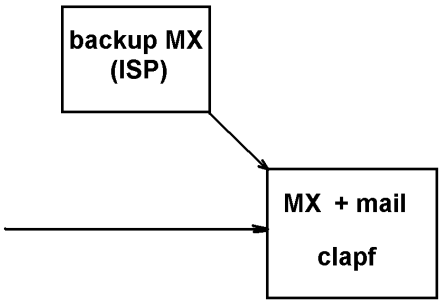
A szokásos szendvics felállásban (1. ábra) az MTA fogadja a leveleket, amelyet továbbít a clapfnak, amely, miután elvégezte a dolgát, visszaadja az MTA-nak.




1. ábra. A clapf spamszűrő kapcsolata az MTA-val

## 6.1. Kis cég

A legegyszerűbb felállásban a vállalkozásnak egyetlen levelezőszervere van. A tartalék MX-et a szolgáltató nyújtja, amelyről a levelek a cég gépére érkeznek (2. ábra). Mivel a clapf sok kereskedelmi megoldásnál hatékonyabb, ezért a spamszűrést a cég gépén oldjuk meg. Kérhetjük azt is a szolgáltatótól, hogy egyáltalán ne szűrje a mi leveleinket.




2. ábra. Minimális konfiguráció

## 6.2. Nagyobb cég

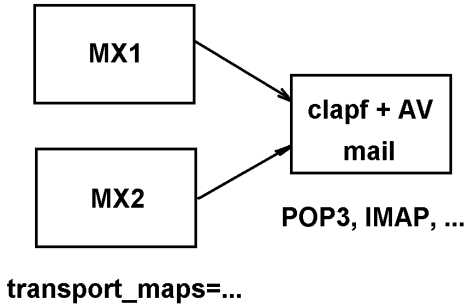
Ha komolyabb levelezést folytat a cég, akkor érdemes 2 MX szervert használni. A 3. ábrán látható konfigurációban az MX1 és MX2 nevű gépek csak fogadják a leveleket, majd továbbítják a mögöttük lévő mailszerverre, ahol a clapf is fut. Ebben a felállásban az MX-ek (relatíve) olcsóak lehetnek, mert az erőforrás igényes tartalomszűrést nem azok végzik. Ha pedig a belső mailszerver leállna, akkor az MX-ek átmenetileg tárolják a leveleket.

## 6.3. Még nagyobb cég

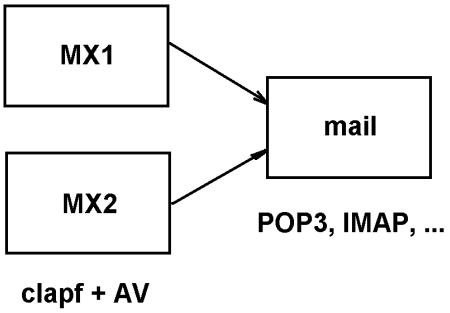
Nagyobb levélforgalomnál érdemes a tartalomszűrést az MX-ekre tenni, és a szűrés I/O intenzív terhelését megosztani közöttük (4. ábra). Ebben az esetben azonban már nagyobb teljesítményű gépeket kell használnunk erre a célra, viszont a mögöttük levő mailszerver terhelése csökken.

## 6.4. Microsoft környezetben

Gyakori eset, hogy egy kis- vagy közepes cégnél adott a Microsoft platform, jellemzően az Exchange és Active Directory duóval. Noha ezeken is lehet spamet




3. ábra. Nagyobb cég levelezése

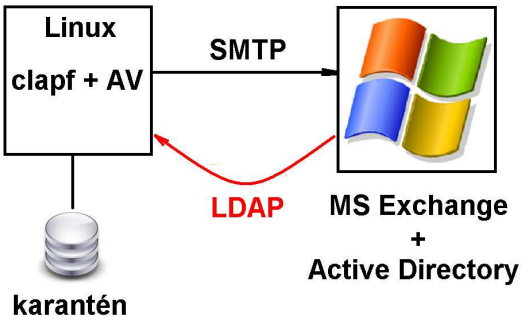



4. ábra. Több gépen is futhat a spamszűrő

szűrni, azonban ez nem olcsó mulatság, mivel a nyílt forrású termékek itt ritkaságszámba mennek. A megoldás azonban egyszerű: telepítsük a clapf spamszűrőt egy Linuxot futtató gépre (ez akár egy virtuális gép is lehet), majd irányítsuk rá a bejövő leveleket. A clapf menedzsment felületén mindössze néhány kattintással importálhatjuk LDAP protokollal az érvényes email címeket. Emellett a linuxos gép a karantén szerepét is elláthatja, így az Exchange-re már csak a tiszta levelek érkeznek meg (5. ábra). Ez a konfiguráció nagy mértékben csökkenti az Exchange szerver terhelését, illetve leveszi róla a nem létező címzettek kezelésének nyűgjét. A felhasználók számára a működés teljesen transzparens, észre sem veszik, hacsak azt nem, hogy eltűnt a spam.

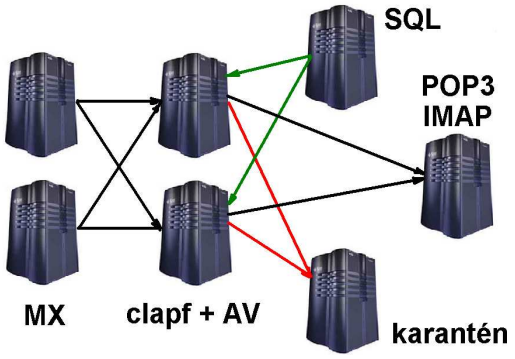
## 6.5. Nagyvállalati konfiguráció

Nagyobb környezetben, mondjuk egy szolgáltatónál nem egy vagy két géppel oldják meg a levelezést, hiszen gondolni kell pl. a redundanciára, illetve az egyes funkciók, szolgáltatások megfelelő szétválasztására. A 6. ábrán látható példában




5. ábra. Windows-os környezetbe is integrálható

2 MX fogadja a leveleket, majd továbbítják 2 tartalomszűrőnek, amelyeken a clapf és egy támogatott antivírus szoftver (pl. clamav) fut. Ezek a jó leveleket azonnal továbbítják a mailszerverre, ahonnan a felhasználók POP3, IMAP4, webmail vagy más módon tölthetik le. Végül egy dedikált gépen gyűlnek a spamek - ha úgy állítjuk be -, mert a clapf képes a spameket más SMTP szerverre továbbítani, mint a jó leveleket. Ez természetesen akár felhasználónként állítható a házirend szabályok (policy group) segítségével. A clapf preferáltan SQL-ben tárolja az email címeket, domaineket, a felhasználónkénti beállításokat és a tokeneket. Egy adott méret felett érdemes az SQL szervert is egy külön gépre tenni.



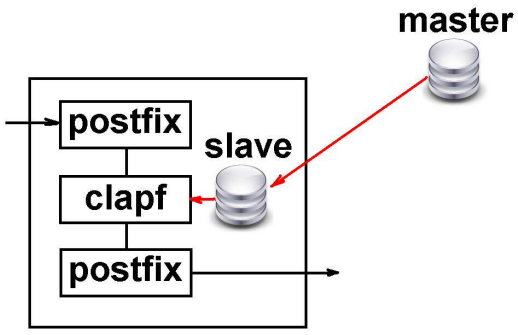

6. ábra. Elosztott rendszert is építhetünk a clapf spamszűrővel

# 7. Teljesítményhangolás

Bár a clapf az alapértelmezett konfigurációval is elfogadható teljesítményt ad, a beállítások hangolásával még nagyobb áteresztőképességet érhetünk el. Az aláb-

biakban néhány tippet mutatok be.

- Vegyük le 1-re a naplózás szintjét, hacsak nem éppen hibát keresünk.
- Tegyük az ideiglenes állományokat ramdiszkre, így a clapf gyorsabban képes beolvasni a levelet.
- Kapcsoljuk ki az RBL vagy SURBL/URIBL lekérdezéseket, mivel a DNS válaszok ideje számottevő lehet, és így visszafogják a clapf teljesítményét. Ha nem tudunk RBL listák nélkül élni, akkor az egyes gépek közös DNS cache-t használjanak.
- Kapcsoljuk ki a tokenek időbélyegének frissítését, vagy irányítsuk azokat memcached démonba, ahonnan késleltetett, illetve kötegelt (batch) feldolgozással oldhatjuk meg a feladatot. Fontos, hogy ha kikapcsoljuk ezt a funkciót, akkor ne futtassuk a törlő szkripteket.
- Ha kisebb teljesítményű gépünk van (pl. virtualizált környezet), hagyjuk el a szendvics első elemét.
- Ha több gépen futtatjuk a spamszűrőt, akkor használjunk azokon MySQL replikákat (7. ábra). A spamszűrőt úgy is be lehet állítani, hogy a replika adatbázist csak olvassa, amelyet ebben az esetben ennek megfelelően lehet optimalizálni.




7. ábra. MySQL replikák használata

- A legnagyobb teljesítménynövekedést MySQL hangolással érhetjük el. Tegyünk elég memóriát az SQL kiszolgálót futtató gépbe, és tuningoljuk okosan a különféle pufferek, illetve paraméterek értékét. Tegyük az SQL adatokat külön diszkre vagy külön gépre. Az SQL szervert helyesen beállítva, gyakorlatilag minden kérést memóriából képes kiszolgálni.

# OpenOffice.org projekt ma és holnap

Szalai Kálmán

**Kivonat**

Az OpenOffice.org egy nyílt forráskódú, ingyenes, nagy tudású irodai programcsomag, amely kiválóan alkalmas szövegszerkesztési, táblázatkezelési, bemutatókészítési, adatbázis-kezelési és rajzolási feladatok megvalósítására Windows, Linux, Solaris és Mac OS X operációs rendszerek alatt. Ezzel a tulajdonságával helyettesíteni képes olyan költséges megoldásokat, mint amilyen a Microsoft Office vagy Corel WordPerfect Office.

A programcsomag a kor igényeinek megfelelő eszköz, amelynek működése nagymértékben hasonlít a kereskedelmi forgalomban lévő hasonló célú programcsomagokhoz. De azokkal ellentétben az OpenOffice.org teljesen ingyenes és szabadon használható tetszőleges számú számítógépen – bármilyen célra – magánszemélyek, közintézmények és vállalkozások esetében egyaránt.

# Tartalomjegyzék

**1. Az OpenOffice.org története** **109**
- 1.1. A legkisebb testvér szerencsét próbál . . . 109
- 1.2. Kifogták a csillagot . . . 109
- 1.3. Megszületik az OpenOffice.org programcsomag . . . 110
- 1.4. Az OpenOffice.org fejlődése . . . 110

**2. Az OpenOffice.org fájlformátumai** **112**
- 2.1. OpenDocument fájlformátum . . . 112
- 2.2. További támogatott fájlformátumok . . . 112

**3. Az OpenOffice.org programjai** **113**
- 3.1. OpenOffice.org Writer . . . 113
- 3.2. OpenOffice.org Calc . . . 113
- 3.3. OpenOffice.org Impress . . . 113
- 3.4. OpenOffice.org Draw . . . 113
- 3.5. OpenOffice.org Base . . . 114

3.6. OpenOffice.org Chart . . . . . 114
3.7. OpenOffice.org Math . . . . . 114
3.8. OpenOffice.org keretrendszer . . . . . 114
3.9. OpenOffice.org Gyorsindító . . . . . 114
3.10. Makrórögzítő és a StarBasic . . . . . 115

**4. Kapcsolat a külvilággal** **115**

**5. Kiterjesztések** **115**

**6. Az OpenOffice.org napjainkban** **116**

**7. Mindenki lehet hozzájáruló** **116**

# 1. Az OpenOffice.org története

## 1.1. A legkisebb testvér szerencsét próbál

Az OpenOffice.org első kiadásának alapjaiul szolgáló StarOffice alkalmazáscsomagot – egy német cég – a StarDivision kezdte el fejleszteni még 1986-ban. A lüneburgi születésű Marco Börries 16 évesen határozta el, hogy egyedi fejlesztésű irodai programcsomagot készít. A még ebben az évben megjelenő termék mindössze egyetlen komponenst, a szövegszerkesztésre szakosodott StarWriter komponenst tartalmazta, és DOS környezetben futott.

1993-ban elkészült a termék Windows-os verziója, melyet egy évvel később az OS/2-es és a Macintosh-os operációs rendszerekre szánt változatok követtek. A StarOffice nevet 1995-ben vette fel az irodai programcsomag, amely ekkorra már több jelentős összetevőt is tartalmazott: szövegszerkesztőt (StarWriter), rajzprogramot (StarImage), táblázatkezelőt (StarCalc), grafikonkészítőt (StarChart) és egy vektoros rajzolóprogramot (StarDraw).

A StarOffice 3.1 jelzésű verziója 1996-ban jelent meg. Ez a kiadás már egy internet böngészőt és egy HTML-szerkesztőt is tartalmazott az eddig meglévő komponensek mellett. A csomag 1997-ben vált teljes irodai csomaggá, amikor bemutatókészítő (StarImpress) és adatbázis-kezelő (StarBase) is került a StarOffice irodai programcsomagba. Ezek a változatok a német piacon nagy népszerűségnek örvendtek, így ezzel párhuzamosan néhány további nyelven is elérhetővé vált az alkalmazás.

## 1.2. Kifogták a csillagot

A StarDivision cég története 1999-ben ért véget, amikor a Sun Microsystems 73,5 millió dollárért felvásárolta a céget.

Az elsődleges ok, amiért a Sun megvásárolta a StarDivisiont, hogy irodai programcsomaggal lássa el az akkor alkalmazásában álló negyvenkétezer alkalmazottját, akiknek Unix-os munkaállomások mellett Windows-os laptopokkal is rendelkeztek. Ilyen körülmények között olcsóbb volt megvásárolni egy olyan céget, amely Solaris és Linux környezetbe is képes irodai programcsomagot készíteni, mintsem megvásárolni a Microsofttól a 42 000 munkaállomáshoz szükséges licencet.

Nem sokkal a megvásárlás után a Sun – személyes használatra – ingyenesen letölthetővé tette a StarOffice 5.2-es verzióját, hogy így próbálja meg növelni a termék piaci részesedését. A későbbi változatok már ismét fizetős, kereskedelmi termékekként kerültek a felhasználókhoz, miközben a fejlesztési modell gyökeresen átalakult, s a szabad szoftverekre jellemző fejlesztési eljárást vezetett be a Sun.

## 1.3. Megszületik az OpenOffice.org programcsomag

2000. október 13-án a Sun OpenOffice.org néven – az LGPL és a SISSL licenc alapján – szabaddá tette a StarOffice forráskódját. Ezt a hatalmas lépést rengeteg előkészítő munka előzte meg, mert több – zárt forráskódú – licencelt termékektől való függőséggel le kellett számolni. Persze az így – OpenOffice.org néven – kiadott kód inkább a fejlesztőknek szólt még, akik egyre nagyobb lelkesedéssel vetették rá magukat az új lehetőségre, hogy a nyílt forrás alapjain irodai programcsomagot fejlesszenek. Az OpenOffice.org körülbelül másfél év alatt érte el az első nagy mérföldkövet: az 1.0-s verzió 2002. május elsején jelent meg. Az azt követő 1.0.x verziók elsősorban hibajavításokkal szolgáltak. Jelentősebb fejlődés 2003 őszén történt, amikor megjelent az OpenOffice.org 1.1-es kiadása, amelyet ismét hibajavító verziók követtek.

Bár eredetileg a StarOffice forrásából született az OpenOffice.org, azóta az ellentettjére fordult a helyzet. A StarOffice 6.1 megjelenése előtt a korábbi verziók az OpenOffice.org egyes stabil fázisaiból indultak ki. A két, párhuzamosan karbantartott vonal többletmunkát és sok gondot jelentett a fejlesztőknek, ezért a StarOffice 6.1 illetve OpenOffice.org 1.1-es kiadása óta azonos forráskódból készül mindkét programcsomag.

Különbséget a hozzáadott adatok, programok és szolgáltatások jelentenek. A StarOffice-hoz kereskedelmi helyesírás-ellenőrző, professzionális szintű kép- (ClipArt) és sablongyűjtemény jár, emellett a termék minőségbiztosítását a Sun végzi, míg az OpenOffice minőségbiztosítását a közösség tagjai látják el. A StarOffice az OpenOffice.org programcsomaghoz képes még további hozzáadott értéket tud felmutatni, úgy mint az Adabas D adatbázis-kezelő, a licencelt betűtípusok, a teljes dokumentáció és a hivatalos támogatás.

## 1.4. Az OpenOffice.org fejlődése

A következő nagy lépcső, a 2.0-s verzió fejlesztése 2003 elején kezdődött a következő célkitűzésekkel:

- Nagyobb fokú együttműködés a Microsoft Office programmal
- Nagyobb teljesítmény és kisebb memóriafoglalás
- Jobb integráció az operációs rendszerekkel
- Könnyebben használható és átláthatóbb adatbázis-kezelőfelület: táblák, jelentések, űrlapok készítéséhez és beépített SQL adatbázis-motor (Base)
- Fejlettebb felhasználói felület

2005. szeptember 2-án a Sun bejelentette, hogy az SISSL licencét megszünteti. Ennek hatására az OpenOffice.org Közösségi Tanács bejelentette, hogy attól kezdve a kettős licenc megszűnik, és a szoftvercsomag csupán az LGPL licenc alatt jelenik meg.

2005. október 20-án megjelent az OpenOffice 2.0, amely az első – külföldi beszámolók alapján szélesebb körben – sikereket elérő kiadás volt. A közel két év fejlesztési eredményeit tartalmazó 2.0-s verzió folyatásaként a sorozatból is több kiadás jelent meg, melyek javításokat és kisebb-nagyobb újdonságokat hoztak.

Eleinte 3 havonta jelent meg egy ilyen, hibajavításokat és új funkciókat is tartalmazó alverzió. Azonban ez túl gyors ütemnek bizonyult, az újdonságokat nem volt idő megjelenés előtt alaposan kitesztelni. Ezért a 2.2-es verzió után a fejlesztők áttértek a fél éves ciklusra, azaz például az OpenOffice.org 2.3 verziója fél évvel a 2.2 verzió után jelent meg 2007 szeptemberében. Félidőben beiktattak egy kizárólag javításokat tartalmazó 2.2.1-es verziót. A menetrend azóta is változatlan, a kiadások hozzávetőlegesen három havonta érkeznek, amelyekből minden második új szolgáltatásokat, továbbfejlesztéseket is nyújt. Ezek közül némelyik kiadás egészen komoly frissítéseket is tartalmazott, mint például az alapoktól újraírt Chart modul vagy a Hunspell helyesírási modul.

A napjainkban megjelenő új verziók is erőteljes fejlődést mutatnak. Ráadásul a 3.x verziók fejlődése nem csak a kínált szolgáltatások terén mérhető le, hanem a külalak is csiszolódik a felhasználók igényeihez, és a szoftver sebessége is javul, hála a kapcsolódó kutatásoknak és fejlesztéseknek. Az OpenOffice.org 3.0-s verziója 2008 októberében jelent meg, új szolgáltatásokkal bővülve, többek között az Office Open XML dokumentumok importálása, az új ODF 1.2 formátum támogatása, a megújult megjegyzések a Writerben, a Megoldó megjelenése a Calcban, a fejlettebb VBA makró támogatás és a natívan futó binárisok Mac OS X operációs rendszerekhez.

Bő fél évre rá, 2009 májusában jelent meg az OpenOffice.org 3.1, amely a következő lényegesebb újdonságokat hozta: élsimítással továbbfejlesztett megjelenés, dokumentumba illesztett grafikus elemek egyszerűbb mozgatása, továbbfejlesztett fájlzárolás, nagyítás/kicsinyítés-csúszka minden alkalmazás állapotsorában, makróalkalmazások a Base programban és javított teljesítmény.

A következő kiadás a még ebben az évben megjelenő 3.2-es verzió lesz, amelynek újdonságait az előadásom keretein belül fogom részletesen bemutatni.

# 2. Az OpenOffice.org fájlformátumai

## 2.1. OpenDocument fájlformátum

Az OpenOffice.org az ISO/IEC által ISO/IEC 26300:2006 néven szabványosított OpenDocument fájlformátumot használja alapértelmezetten. Az OpenDocument vagy ODF (eredetileg OASIS Open Document Format for Office Applications) egy nyílt fájlformátum-szabvány irodai programcsomagok dokumentumainak, úgy mint szöveges dokumentumok, táblázatok, adatbázisok és bemutatók tárolására és cseréjére.

A széles iparági támogatottságot jól jelzi, hogy a szabványt az OASIS ipari konzorcium készítette, amelynek tagja minden jelentősebb informatikai vállalat. Az OpenDocument fájlformátum elkészítéséhez az OpenOffice.org XML-alapú formátuma szolgált alapul. Az OpenDocument az első olyan fájlformátum-szabvány, amit az irodai programcsomagok számára készített egy független, elismert szabványosító szervezet. A szabvány szabadon, jogdíjak nélkül felhasználható, ezzel életképes alternatívája a piaci versenyt gátló zárt vagy jogdíj ellenében felhasználható formátumoknak.

A fentiekkel összhangban számos termék és internetes szolgáltatás kezdte el támogatni az OpenDocument fájlformátumot, és számos fejlesztő építette bele termékeibe az OpenDocument fájlformátum támogatását. A teljesség igénye nélkül néhány alkalmazás, amely érti az OpenDocument fájlformátumot:

Abiword, Adobe Buzzword, Google Docs, IBM Lotus Symphony, IBM Lotus Notes, KOffice, Microsoft Office, Mobile Office, WordPerfect Office, Apple Quick Look, Gnumeric, phpMyAdmin, OmegaT+, IBM WebSphere, Scribus, Inkscape, Google, Beagle, Google Desktop Search, Apple Spotlight, Copernic Desktop Search, BlackBerry Enterprise Server.

## 2.2. További támogatott fájlformátumok

Az OpenOffice.org természetesen teljes körűen támogatja a StarOffice-ban régebben alkalmazott dokumentumformátumokat is, és jó minőségben kezeli a versenytársak egyedi fájlformátumait is. Az OpenOffice.org a Microsoft Office és más irodai csomagok (például: WordPerfect, Lotus 1-2-3, Microsoft Works), valamint a Rich Text Format állományok szinte minden formátumát és verzióját olvassa, és bizonyos esetekben írja is. Az OpenOffice.org ezen tulajdonsága a felhasználók többségének alapvetően fontos. Az OpenOffice.org képes megnyitni a Microsoft Office korábbi változataival készült dokumentumokat, valamint olyan sérült Microsoft Office állományokat is, amelyekkel az aktuális Microsoft Office változat már nem birkózik meg.

# 3. Az OpenOffice.org programjai

Az OpenOffice.org a kínált programok tekintetében egy kisebb csomag, mint amilyen volt a StarOffice az 5-ös sorozat idejében. Mivel az OpenOffice.org forrása immáron közös alap a StarOffice forrásával, ezért a felhasználók számára kínált programok megegyeznek a két irodai programcsomagban. Az OpenOffice.org a következő programokból áll:

## 3.1. OpenOffice.org Writer

A Writer az OpenOffice.org szövegszerkesztője: bármire használható az egyoldalas levelektől kezdve a beágyazott ábrákat, kereszthivatkozásokat, tartalomjegyzéket, tárgymutatót és irodalomjegyzéket tartalmazó teljes könyvig bezáróan.

Az automatikus kiegészítés, az automatikus formázás, az írás közben működő helyesírás-ellenőrzés a legnehezebb feladatot is könnyen elvégezhetővé teszi. A Writer tudása elég ahhoz, hogy kiadványszerkesztési feladatokat is végezzenek vele, például többhasábos hírlevél vagy brossúra készítését.

## 3.2. OpenOffice.org Calc

A Calc egy könnyen használható táblázatkezelő program, ami megkönnyít a munkát. Legyen szó egyszerű táblázatokról vagy éppen bonyolult adatelemzésekről, a Calc program megbízható társ lesz. Az adatok – akár külső forrásból is – könnyedén emelhetőek be a Calc programba, hogy ezek alapján hatékonyan lehessen számolni, elemezni és adatokat megjeleníteni. A haladó táblázatkezelő funkciók és döntéshozó eszközök miatt bonyolult adatelemzések bemutatásához is alkalmas.

## 3.3. OpenOffice.org Impress

Az Impress gyors és egyszerű módot kínál hatásos multimédia prezentációk készítésére. A programmal elkészített prezentációk igazán ki fognak tűnni különleges effektusaikkal, animációikkal és színvonalas rajzos illusztrációik révén. Az elkészített prezentációk könnyedén exportálhatók Flash vagy weboldal formátumú előadásba is.

## 3.4. OpenOffice.org Draw

A Draw segítségével bármilyen illusztráció könnyedén készíthető el az egyszerű diagramoktól kezdve a háromdimenziós illusztrációkon át a különleges effektusokig. A professzionális hatás érdekében a Draw programban készült képek be-

ágyazhatók az OpenOffice.org szövegszerkesztőjében, táblázatkezelőjében vagy bemutatókészítőjében megnyitott dokumentumokba is.

## 3.5. OpenOffice.org Base

A Base komponens lehetővé teszi adatbázisok adatainak manipulálását az OpenOffice.org barátságos felületén keresztül. Az program alkalmas táblák, űrlapok, lekérdezések és jelentések létrehozására és módosítására, és a felhasznált adatok akár külső adatbázis, akár a Base beépített HSQL adatbázismotorja használatával kezelhetők.

## 3.6. OpenOffice.org Chart

A Chart kiváló módja annak, hogy a Calcban elkészített táblázatok eredményeit az olvasó számára is látványos formában jelenítsük meg. A számtalan beépített diagram közül biztosan lesz olyan, amely az adatokat a legjobb módon szemlélteti, így elősegítve az ábrázolt kimutatások és trendek megértését.

## 3.7. OpenOffice.org Math

A Math összetett matematikai képletek készítésére és szerkesztésére szolgáló eszköz. A szerkesztő használatával a nagy komplexitású képletek bevitele is leegyszerűsödik. A képletek beilleszthetők más OpenOffice.org dokumentumokba.

## 3.8. OpenOffice.org keretrendszer

Ez a komponens önállóan nem érthető el, mégis minden alkalmazásban megjelenik. A keretrendszer lényege, hogy az OpenOffice.org irodai programcsomag minden eleme egy közös alapon nyuszik, amely így egységességet és azonos funkciókat kínál a rá épülő alkalmazások számára. Ez az egységesség vonatkozik egyrészt az alkalmazások megjelenésére, a fájlkezelés univerzális módjára, másrészt a kínált szolgáltatások alkalmazásokon átívelő lehetőségeire. Erre a programok közötti szoros integrációra példa a bárhonnan elérhető PDF exportálás és a dokumentumok e-mailben történő küldésének lehetősége, az egységes és minőségi nyelvhelyességi eszközök vagy a különféle eszközök és eszköztárak megegyező működési és elérési módja.

## 3.9. OpenOffice.org Gyorsindító

A Gyorsindító egy kis segédprogram Windows és Linux operációs rendszerekre, amely a számítógép bekapcsolásakor indul el, és betölti az OpenOffice.org alap-

vető részeit, lehetővé téve, hogy a csomag alkalmazásai gyorsabban indulhassanak el. Az OpenOffice.org-hoz hasonló komplex szoftverek gyorsabb indítására ez az egyik lehetséges jó megoldás.

### 3.10. Makrórögzítő és a StarBasic

Az OpenOffice.org rendelkezik makrórögzítő funkcióval is, amely a Writerben és a Calcban használható, a felhasználó tevékenységének rögzítésére és visszajátszására való eszköz. Ennek eredménye egy a saját StarBasic nyelvében készült programkód, ami tovább szerkeszthető. A StarBasic egy önálló programozási nyelv, így készíthetünk teljesen egyedi programokat is az irodai munka megkönnyítésére. Az OpenOffice.org a StarBasiben történő fejlesztések támogatásához egy beépített fejlesztői környezetet is biztosít. A StarBasic nem kompatibilis a Microsoft Officeban alkalmazott Microsoft Visual Basic for Applications (VBA) nyelvvel, de az OpenOffice.org 3.0-s verziójába is beépítették a Microsoft VBA makrók futtatásának képességét, amely jelenleg nem teljes még, de a folyamatos fejlesztéseknek köszönhetően igen gyors ütemben gyarapodik tudása.

## 4. Kapcsolat a külvilággal

Az OpenOffice.org nemcsak egy sokoldalú irodai programcsomag, hanem egy alkalmazásfejlesztő platform is. Az egyre bővülő, ám stabil API-n keresztül az OpenOffice.org szinte minden funkciója elérhető külső programból. Az OpenOffice.org egyik nagy előnye, hogy széleskörűen támogatja külső modulok írását, amik igénybe veszik az OpenOffice.org szolgáltatásait, vagy az OpenOffice.orgban végzik el a megadott műveleteket. Ezen kötések létrehozásával az OpenOffice.org kiválóan testre szabható, és a lehetőségeknek csak a fantázia szab határt. Az OpenOffice.org programozható a már említett StarBasic beépített makrónyelven, valamint az alábbi nyelvek egyikén: C, C++, Python, Java, JavaScipt és BeanShell.

## 5. Kiterjesztések

A Mozilla Firefox által meghonosított kiterjesztések technológiáját az OpenOffice.org is átvette. A könnyen telepíthető és frissíthető kiterjesztések segítségével egyszerűen adhatunk új funkciókat az OpenOffice.org irodai programcsomaghoz, kezdve az OpenOffice.org által támogatott programnyelveken megírt programoktól, funkcionális kiegészítésektől, a nyelvi ellenőrző komponenseken át egészen

a felhasználók számára hasznos sablonokat és képtárakat tartalmazó kiterjesztésekig.

Nem csoda, hogy a kiterjesztések az OpenOffice.org legnépszerűbb és a közösség számára leginkább értékelhető területe, amely segítségével a hozzájárulók száma tovább emelkedhet úgy, hogy a közösség számára is hasznos kiterjesztések látnak napvilágot. A Firefoxot – többek között – a kiterjesztései tették naggyá, s úgy tűnik, az OpenOffice.org irodai programcsomag is ebbe az irányba halad.

# 6. Az OpenOffice.org napjainkban

Jóllehet az OpenOffice.org egy nyílt forráskódú termék, amit a közösség készít a közösség számára, azért a Sun továbbra is vezető szerepet játszik a termék fejlesztési irányának meghatározásában, stratégiai döntésekben és a fejlesztés megvalósításában is. Ennek legfőbb okai, hogy a számos hozzájáruló mellett továbbra is a Sun adja a legtöbb teljes munkaidőben alkalmazott fejlesztőt, fejlesztői erőforrást és az infrastruktúrát a projekt számára. Noha az OpenOffice.org továbbra is a Sun által koordinált közösségi projekt, azért a projekt széles közösségi hozzájáruló csoport támogatását is élvezi, sok céges háttérrel támogatott fejlesztést is végeznek olyan cégek közreműködésével, mint amilyen a Novell, a RedFlag 2000, a Red Hat, az IBM vagy éppen a Google. A számok tükrében kifejezve a főállású fejlesztők száma meghaladja a százat, és legalább ötszáz rendszeres hozzájárulót tartanak számon a projekttel kapcsolatban. Az OpenOffice.org több mint 750 000 regisztrált taggal dicsekedhet világszerte. Az OpenOffice.org – elsősorban külföldön – egyre nagyobb iparági elfogadottságnak örvend az állami és a privát szektorban is. Ezért alapvető cél, hogy az OpenOffice.org irodai programcsomag magyarországi jelenlétét megerősítsük, és felhívjuk a figyelmet erre a nagyszerű alternatívára, egyszersmind bővítsük a hazai hozzájárulók körét, akik a magyar igényeknek megfelelő tartalommal és szolgáltatásokkal képesek ellátni a felhasználókat.

# 7. Mindenki lehet hozzájáruló

Az OpenOffice.org-hoz hasonló közösségi projektek legnagyobb előnye, hogy mindenki tudásához és lehetőségeihez mérten tehet hozzá a közösség értékeihez. A hozzájárulás módját szinte nem korlátozza semmi, de célszerű a hazai OpenOffice.org levelezőlistán vagy fórumon egyeztetni a többi közreműködővel. A legtöbb tevékenység felbontható részfeladatokra, így könnyedén dolgozhat az adott feladaton több hozzájáruló, s az egyén szakértelmének fejlődésével összhangban egyre testhezállóbb, nagyobb kihívást jelentő munkákkal találkozhat.

Pár terület, ahol az új felhasználók hozzájárulásait várjuk:

- OpenOffice.org használata és megismerése
- OpenOffice.org bemutatása közönség előtt
- Marketing tevékenységek
- Fórumokon és levelezőlistákon önkéntes segítségnyújtás
  - `http://user.services.openoffice.org/hu/forum/`
- Sablonok és kiegészítők közzététele
- Dokumentáció készítése és fordítása
- Magyar verzió hibáinak jelentése (magyarul)
  - `http://code.google.com/p/openscope/issues/list`
- Hibajelentések készítése (angolul)
  - `http://qa.openoffice.org/issue_handling/pre_submission.html`
- Programozói tudás birtokában:
  - hibajavítások,
  - kiterjesztések,
  - és továbbfejlesztések készítése

# Kernel Virtual Machine és Virtualizáció

Szántó Iván
ULX Open Source Consulting & Distribution

**Kivonat**

A virtualizációnak sokféle definíciója létezik. Ebben az anyagban virtualizáció alatt azt a technológiát értjük, amely absztrahálja a hardvererőforrásokat abból a célból, hogy ugyanazon a hardveren egyszerre több operációs rendszer példánya futhasson.

Mivel az operációs rendszer azzal a céllal készül, hogy a hardvert kezelje, ezért az összes olyan hardvererőforrást absztrahálni kell a virtualizált operációs rendszer számára, amit az adott operációs rendszer kezelni kíván. Amikor ezt az absztrahálást szoftver végzi, akkor emulációról beszélünk. Alapesetben az összes hardvererőforrást emulálni kell: a processzort, a memóriát és az I/O eszközöket. Az ettől az alapesettől való eltérésekre később kitérünk.

A virtualizáció célja, azaz az, hogy mit akarunk tulajdonképp elérni a virtualizációval, igen sokféle lehet: szerverkonszolidáció, a szerverek erőforrásainak jobb kihasználása, energiatakarékosság, új technológiák kihasználása, új gépek gyors provizionálása, költségek csökkentése, legacy szoftverek életciklusának megnövelése, hardverhibákkal szembeni tűrőképesség növelése.

# Tartalomjegyzék

**1. Történelem és jelenkor** **121**

**2. Processzorvirtualizáció** **121**
2.1. Teljes virtualizáció (full virtualizáció) . . . . . 122
2.2. Hardveres támogatás (hardware assist) . . . . . 124

**3. Memóriavirtualizáció** **124**

**4. I/O Virtualizáció** **125**
4.1. Absztrakció: elvonatkoztatás a valóságtól . . . . . 125
4.2. Virtualizált meghajtók . . . . . 126

**5. Erőforrások felosztása és összegzése** **127**
5.1. Processzor és memória . . . . . 127
5.2. Hálózat . . . . . 128
5.3. Háttértár . . . . . 128

**6. KVM architektúra** **129**
6.1. Processzorvirtualizáció . . . . . 129
6.2. Memóriavirtualizáció . . . . . 130
6.3. I/O virtualizáció . . . . . 131
6.4. KVM desktop virtualizáció . . . . . 131

**7. KVM Menedzsment eszközök** **132**
7.1. Tapasztalatok éles környezetben . . . . . 133

# 1. Történelem és jelenkor

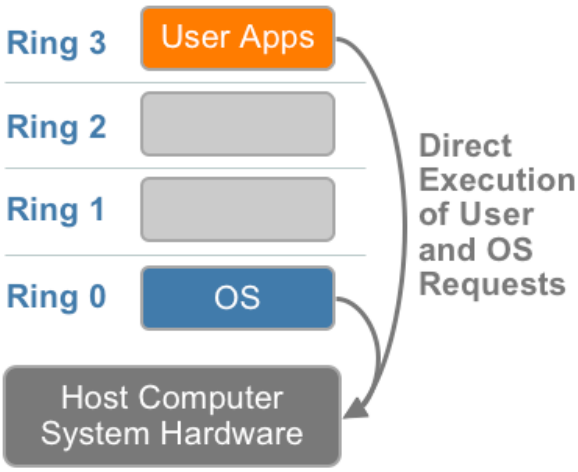
A virtualizáció először az 1960-as években valósult meg az IBM M44/44X rendszerben. Ebben a kísérleti rendszerben az M44-es hardveren 44X-es virtuális gépek (VM, virtual machine) futottak. Az absztrahálás szoftver- és hardverelemek segítségével történt. Ebből fejlesztették ki a később széles körben elterjedt CP-67/CMS rendszert, amely S/360-as mainframe gépeken futott. Ebben a rendszerben a CP (Control Program) teljes és paravirtualizációs technológiák felhasználásával biztosította a virtualizációs környezetet. Itt minden felhasználó számára külön egyfelhasználós virtuális gépet lehetett létrehozni, amelyeken a CMS operációs rendszer futott. A CMS-en minden olyan szoftver fut, amely az S/360-as gépeken fut, ez a rendszer kényelmes feloszthatóságát tette lehetővé. Továbbá a CMS-t úgy tervezték, hogy ne használjon sok erőforrást, ezért az alatta levő hardvert hatékonyan használta ki.

A virtualizációval kapcsolatban gyakran felmerülő hypervisor kifejezés is a mainframe-es időkből ered, mégpedig a supervisor fokozásából. A mainframeeken a kernel supervisor módban futott, ezért a kernelt supervisornak is nevezték. A rendszerhívásokat, amelyeken keresztül a felhasználói folyamatok a kernelt elérik, supervisor hívásoknak hívták. Ennek mintájára a virtualizációt megvalósító szoftvert hypervisornak, azokat a rendszerhívásokat, amelyeken keresztül a virtualizált operációs rendszerek a hypervisort elérik, hypervisor hívásoknak nevezték.

Viszont itt koncentráljunk inkább az x86 architektúrára, mivel jelenleg ez az egyik legelterjedtebb hardverarchitektúra, viszonylag olcsón elérhető, ezért szinte mindenhol ezt használják, és de-facto szabvánnyá vált, valamint éppen ez ütötte ki a nyeregből többek közt a fent említett cég virtualizációs megoldásait is. Ezen az architektúrán a VMware hozott létre először széles körben elterjedt virtualizációs megoldást 1999-ben. A továbbiakban az x86-os architektúrára alapozva bemutatjuk a fent említett hardvererőforrások konkrét virtualizálási megoldásait. Először a legfontosabb erőforrással, a processzorral foglalkozunk.

# 2. Processzorvirtualizáció

Mielőtt a processzorvirtualizáció típusaira rátérnénk, induljunk ki abból, hogy a hardvert hogyan érik el a szoftverek virtualizáció nélkül. Virtualizáció nélküli esetben az operációs rendszer tartja a hardvert kontroll alatt. Ennek korrekt megvalósításához hardvertámogatás szükséges, amely x86 architektúrán úgy működik, hogy a processzor négyféle módban tud futni. Ezt a négyféle módot gyűrűknek nevezik és számokkal jelölik (0-3). Ezeket a gyűrűket azért vezették be, hogy az operációs rendszert megvédjék a felhasználói folyamatok hibás működésétől vagy biztonsági réseitől.

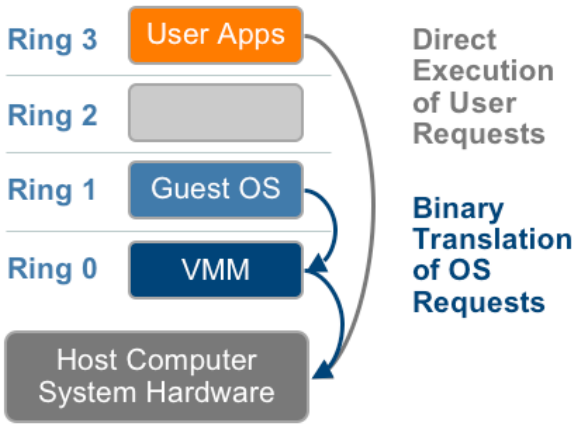



A 0-ás gyűrű a privilegizált módnak felel meg. Bizonyos utasításokat, amelyeket privilegizált utasításoknak (privileged instruction) nevezünk, csak ebben a módban lehet kiadni. Virtualizálás nélküli esetben a kernel privilegizált módban fut, a felhasználói folyamatok pedig user módban a 3-as gyűrűn. A 3-as gyűrűn futó programok elszállása, lefagyása vagy másfajta helytelen működése a többi gyűrűn futó programok működőképességét nem befolyásolja.

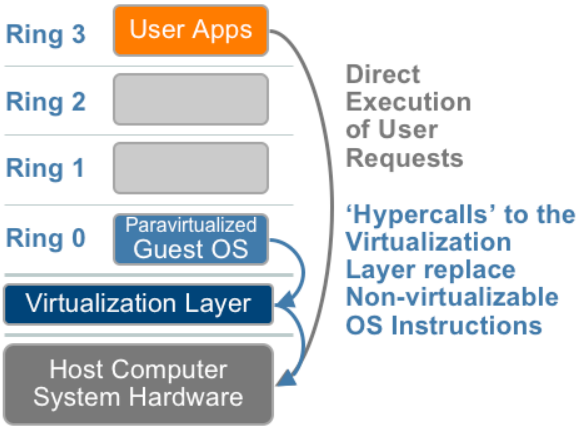
A fentiekből következik, hogy a privilegizált utasításokat csak az operációs rendszer adhatja ki, és a felhasználói folyamatok a hardvert kizárólag rendszerhívásokon keresztül tudják elérni. Technológiailag háromféle processzorvirtualizációról beszélhetünk.

## 2.1. Teljes virtualizáció (full virtualizáció)

Teljes virtualizáció esetén a virtualizálni kívánt operációs rendszer módosítás nélkül fut. Ezt a virtualizációs szoftver bináris fordítással (BT, binary translation) a következőképp teszi lehetővé. A virtualizációs szoftver figyeli a rajta futó operációs rendszert, és a privilegizált utasításokat binárisan lefordítja olyan hívásokra, amelyek nem közvetlenül hajtódnak végre, hanem az emulált hardvert hívják. A nem privilegizált utasításokat a virtualizációs szoftver átadja a processzornak közvetlen végrehajtásra.

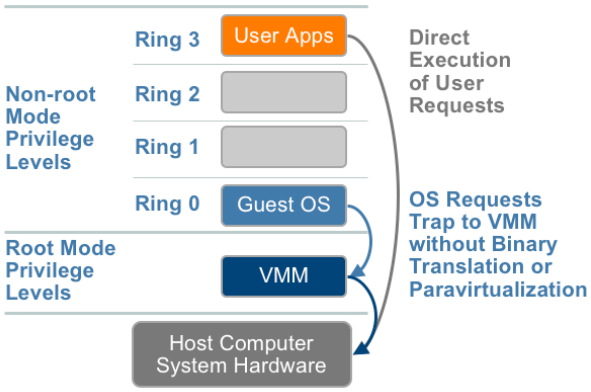



A fentiekből következik, hogy a full virtualizáció hátránya, hogy a szoftver lassabban fut, a szoftveres bináris fordítás overhead miatt, mint virtualizáció nélkül, előnye viszont, hogy az operációs rendszer változtatás nélkül átvihető virtualizált környezetbe. Paravirtualizáció A bináris fordítás overheadjének az eltüntetésére vezették be a paravirtualizációt. Ehhez a virtualizálni kívánt operációs rendszert módosítani kell: a privilegizált utasításokat és egyéb hardverhozzáféréseket ki kell cserélni a hardveremulációt megvalósító szoftver hypervisor hívásaira. Ezzel a bináris fordítás szükségtelenné válik: a paravirtualizált futtatásra felkészített operációs rendszert a virtualizációs szoftverkörnyezet a processzoron külön figyelés nélkül, közvetlenül tudja futtatni, és így lényegesen jobb teljesítményt tud elérni a teljes virtualizációnál. Ami a dolog előnye, az a hátránya is: az operációs rendszert át kell írni, és ez csak akkor tehető meg, ha a forrás rendelkezésre áll, pl. nyílt forráskódú operációs rendszereknél.



## 2.2. Hardveres támogatás (hardware assist)

Amikor maga a hardver támogatja a virtualizációt, akkor a bináris fordításra szintén nincs szükség, sőt az operációs rendszert sem kell átírni. Ez a megoldás az előző kétféle módszer előnyeit egyesíti magában: a jó teljesítményt, valamint azt, hogy a szoftver változatlanul hagyható

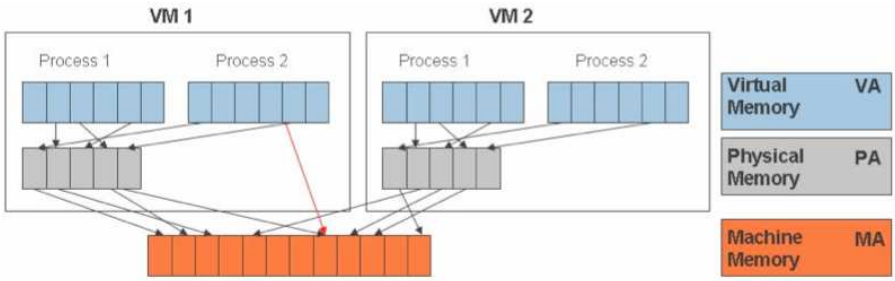



A hardvertámogatást x86 architektúrák esetén a processzor (Intel VT-X vagy AMD-V) és az alaplap együtt nyújtja, ezt a BIOS-ban is be kell kapcsolni.

# 3. Memóriavirtualizáció

Az x86 architektúrában megtalálható lapozótáblák (page tables) feladata, hogy a felhasználói folyamatok által látott virtuális memóriát a gépben található fizikai memóriába képezzék le. Virtualizáció nélküli esetben a lapozótáblák használatával valósítják meg az operációs rendszerek azokat az ismert lehetőségeket is, hogy a felhasználói folyamatok a fizikai memóriánál lényegesen nagyobb virtuális tárterületet is láthatnak, melynek nem szükséges egy időben a fizikai memóriában lennie, hanem ennek egy része kilapozható (page out) a háttértárra. Továbbá ha a felhasználói folyamat olyan lapra hivatkozik, amely nincs épp benn a fizikai memóriában, akkor laphiba (page fault) generálódik, mely az operációs rendszer hívását eredményezi, amely gondoskodik arról, hogy a hivatkozott lap bekerüljön a fizikai memóriába.

A lapozótáblák tehát egy indirekciós réteget képeznek a felhasználói folyamatok által kezelt virtuális memória lapjai és a gép fizikai memória lapjai között.




Az operációs rendszer virtualizációja esetén egy további indirekciós rétegre van szükség, amely a virtualizált operációs rendszer által kezelt memória lapjait a gép fizikai memóriájának lapjaira képzi le. Ezt az indirekciós réteget árnyék laptábláknak (shadow page tables) nevezik, és a jelenleg elérhető hardverek ehhez is nyújtanak támogatást.

# 4. I/O Virtualizáció

Az I/O virtualizálását két szempontból vizsgálhatjuk. Az egyik szempont, hogy mit virtualizálunk, a másik pedig, hogy hogyan.

## 4.1. Absztrakció: elvonatkoztatás a valóságtól

Vegyük először azt, hogy mit virtualizálunk. Itt azt kell eldönteni, hogy a hardveremulációt mennyire vonatkoztatjuk el a valóságtól, azaz mennyire választjuk absztraktra, illetve mennyire részletesen akarjuk kidolgozni az emulált hardvert. Általános célú virtualizáció esetén az absztraktabb eszközöket választjuk. Ilyenkor általában olyan eszközt emulálunk, mint egy virtuális grafikus kártya, egy hálózati kártya vagy egy diszk. Ennek előnye, hogy néhány virtualizált eszközzel a kívánt funkcionalitás lefedhető és működőképessé tehető, feltéve, hogy a felhasználás szempontjából valóban elegendő ennek a háromféle eszköznek az emulációja.

Ha ennél részletesebben szükséges kidolgozni az emulált hardvert, akkor a diszk elérés helyett akár különböző SCSI és optikai csatolókártyákat, különböző hálózati kártyákat, hangkártyákat, USB és PCI busz elérést is lehetővé tehetünk. Ennél alacsonyabb szintre általános célú virtualizáció esetén nem nagyon megy az ember, de ha épp a hardvert tervezi, akkor akár még részletesebben is ki lehet dolgozni: a kártya egyes hardverelemeinek és azok időzítéseinek az emulációja lehetővé teszi a meghajtó programmal való optimális együttműködés kidolgozását.

## 4.2. Virtualizált meghajtók

Arra, hogy az I/O hardvert hogyan virtualizálunk, szintén többféle szintű megoldás létezik. Teljes virtualizáció esetén az operációs rendszert változatlanul hagyjuk. Ezt csak akkor lehet megtenni, ha a virtualizációs szoftver olyan hardverkörnyezetet tud emulálni, amelynek a kezeléséhez megfelelő meghajtóprogramok a virtualizálni kívánt operációs rendszer(ek)ben rendelkezésre állnak. Ahogy a processzorvirtualizációnál is láttuk, a teljes virtualizáció igen erőforrásigényes feladat, tehát ennek a megoldásnak hátránya a szoftveres overhead, előnye pedig, hogy az operációs rendszer változtatás nélkül tud működni.

A paravirtualizált operációs rendszerek esetén, mint a processzorvirtualizációnál láttuk, a privilegizált utasításokat hypervisor hívásokra cserélik le. Így a virtualizációs környezet megteremtésekor szabadon rendelkezhetünk az I/O virtualizáció mindkét oldaláról, az emulált hardverről és az ezt hívó meghajtóprogramokról is. Ennek a megoldásnak előnye, hogy sokkal jobb teljesítményt lehet elérni, hátránya pedig, hogy az operációs rendszert át kell alakítani.

Egy köztes megoldás az, hogy csak azokat a meghajtóprogramokat alakítjuk át, amelyek az adott felhasználás esetén a teljesítmény szempontjából kritikusak. Megfelelő profile-ozás után ezek kiválogathatók (pl. hálózati és diszkmeghajtók). Az operációs rendszer eredeti formájában is képes marad funkcionálisan megfelelő működésre, de a kiválogatott meghajtóprogramokat paravirtualizálva a teljesítmény a paravirtualizált operációs rendszer szintjére hozható. A teljesítmény tovább fokozható a virtualizáció elhagyásával, úgynevezett pass-through (áteresztő) meghajtóprogramok használatával, amikor a hardver egyes elemeihez egy-egy virtuális gép operációs rendszerének adunk kizárólagos hozzáférést.

A virtualizáció használatának I/O esetén is megvannak az előnyei, amelyek a pass-through meghajtóprogramok esetén nem vagy csak korlátozottan jelentkeznek:

- Jól ismert hardver emulációja: ha jól választjuk meg azt a hardverelemet, amit emulálunk, akkor az operációs rendszerek jó eséllyel rendelkeznek ehhez való meghajtóprogrammal.
- Hardverfüggetlenség VM oldalon: ha a virtualizációs réteg elfedi a virtualizált operációs rendszer elől a valódi hardvert, akkor attól függetlenül működik, hogy milyen vasat teszünk alá.
- I/O eszközök megoszthatók a virtuális gépek között.
- Szeparáció, biztonság: a hypervisor gondoskodik arról, hogy az egyes virtuális gépek egymástól és a hálózat többi részétől el legyenek szeparálva, és

csak rajta keresztül érhessék el egymást. Ez biztosítja, hogy ha az egyik virtuális gépre bejut egy támadó, akkor ezzel az ugyanazon a vason futó többi virtuális gépre automatikusan még nem jutott be.

- Migráció lehetősége: ha a virtuális gépek egy emulált hardvert érnek el a meghajtóprogramokon keresztül, akkor pontosan ugyanazt a hardvert lehet egy másik gépen emulálni, ami lehetővé teszi a migrációt. Pass-through meghajtóprogramok esetén ez nem mindig biztosítható, gondoljunk pl. a hálózati kártyák MAC címére vagy az optikai csatolókártyák WWID-jére.

# 5. Erőforrások felosztása és összegzése

A virtualizáció használatának egyik motiváló oka, hogy gyakran előfordul az az eset, hogy a rendelkezésre álló szerverek jó része nincs kihasználva. Ezt a problémát a virtualizáció a rendelkezésre álló erőforrások felosztásával oldja meg.

## 5.1. Processzor és memória

A processzor és memória felosztására jó megoldás a virtualizáció, amit a virtualizációs hoszt felől lehet adminisztrálni. Abban az esetben, ha több alkalmazás között kívánjuk felosztani egy gép memória- és processzor-erőforrásait, de nincs szükség a futtatókörnyezetek olyan éles elválasztására, mint virtualizáció esetén, akkor az operációs rendszer multiprocesszing, illetve. multitasking funkciója is használható erre a feladatra. Ebben az esetben az operációs rendszer felől lehet adminisztrálni a processzor és a memória felosztását.

Ha a memória méretét és a processzorok számát növelni akarjuk, kétféle lehetőségünk van: vagy nagyobb gépet veszünk, amibe több memória és processzor fér (scale-up), vagy több gépet állítunk össze úgy, hogy közösen végezzék el azt a munkát, amit eddig csak egy gép végzett el (scale-out). A nagyobb gépre többféle architektúra létezik: SMP és NUMA. Az SMP (symmetric multiprocessing) esetében a processzorok egyforma sebességgel tudnak hozzáférni a közös memóriához, a NUMA (non-uniform memory access) esetén viszont bizonyos processzorok bizonyos memóriaterületeket gyorsabban érnek el, mint másokat. Több gép esetén is megkülönböztethetünk shared disk és shared nothing architektúrákat, attól függően, hogy van-e közösen használt diszkterület, továbbá aktív-passzív, aktív-aktív és terheléselosztó klasztereket. A különböző hardverarchitektúrákhoz különböző szoftverek és különböző teljesítmény és skálázhatósági tulajdonságok tartoznak, de ezeket itt ennél részletesebben nem tekintjük át, mert ez nem kapcsolódik szorosan a virtualizációhoz.

## 5.2. Hálózat

Egy gépen belüli hálózati erőforrások felosztását, a processzorhoz és a memóriához hasonlóan, operációs rendszerre és virtualizációs technológiára is lehet bízni. Azt, hogy egy adott felhasználói folyamat vagy egy virtuális gép milyen arányban részesülhessen pl. a rendelkezésre álló teljes sávszélességből, az operációs rendszer oldaláról traffic-shaping eszközökkel lehet beállítani.

Az egy gépen belül található hálózati csatolókártyák sávszélességét összegezni lehet egyrészt összekötéssel (bonding), másrészt ha külön címe van a csatolókártyáknak, mivel külön hálózatokba vannak bekötve, akkor ismét a traffic-shapinget lehet segítségül hívni, de lehet alkalmazás-szintű terheléselosztót is használni (például a squid tud ilyet).

## 5.3. Háttértár

Az elérhető háttértár felosztásának tradicionális megoldása a diszkek particionálása, amely a diszk eltérő operációs rendszerek közötti megosztását is lehetővé teszi. Ma már a logikai kötetkezelő is gyakran használt megoldás a diszkterület felosztására, és ez jóval rugalmasabb a particionálásnál, mert egyrészt a logikai kötetek menet közbeni átméretezését is megengedi, másrészt a diszkterület felosztásán kívül több diszkterület összegzésére is megoldást kínál. Virtualizáció esetén a virtualizációs hoszt felől az egyes virtuális gépek számára ki lehet osztani diszk particiót, logikai kötetet, de akár egy fájlt is, és a hypervisor ezek bármelyikét diszkként tudja prezentálni a virtuális gép felé.

A diszkterületek összegzésére való megoldások közül vegyük előre a logikai kötetkezelőt, amelyet, ahogy az előbb láttuk, a diszkterületek felosztására is lehet használni. Linuxban a jelenleg használt logikai kötetkezelő az LVM2, amely a device-mapper kernelmeghajtóit használja a fizikai eszközök logikaivá való leképzésére. A device-mapper a kernelben tartja az éppen érvényes leképzési táblákat, és a menet közbeni átméretezést ezeknek a tábláknak az átírásával oldja meg. Amikor az átméretezést shared disk klaszteres környezetben akarjuk megtenni, akkor a leképzési táblák szinkronban tartásáról külön gondoskodni kell. Erre fejlesztették ki (és fejlesztik tovább most is) a CLVM-et (cluster LVM). Az LVM2, a device-mapper és a CLVM fejlesztését a Red Hat fejlesztőkkel és infrastruktúrával is támogatja.

Amennyiben olyan terheléselosztó klaszterre van szükség, amelyben több gép egyidejűleg kell, hogy írjon-olvasson egy közös fájlrendszert, akkor nem jók a hagyományos fájlrendszerek, hanem olyan klaszteres fájlrendszerre van szükség, amelyet e célra terveztek. Erre való például a GFS fájlrendszer, amelynek szellemi jogtulajdonosát, a Sistina Software nevű céget a Red Hat megvásárolta, majd a kódot nyílt forráskódként, GPL alatt adta ki.

A shared disk architektúra esetén a diszkhez gyakran több út is vezet, és ezeket az operációs rendszernek kell lekezelni. Itt a több eszköz elfedésén túl az egyes utak közötti redundancia kihasználása a cél, ami jelentheti az egyik út kiesése esetén a másikra való automatikus átkapcsolást (failover), és a több út által biztosított nagyobb sávszélesség kihasználását (load-balancing). Erre a feladatra a device-mapper-multipath egy rugalmas, gyártófüggetlen, nyílt forráskódú megoldás.

Több, különálló diszkalrendszer használata esetén felmerül a tükrözés kérdése:a feladat az egyik diszkalrendszerben tárolt adatok tükrözése a másik diszkalrendszerre. Ez is megoldható a device-mapper egyik moduljával, a dm-mirror-ral. Klaszteres környezetben, amikor több gép egyidejűleg kell, hogy írja-olvassa ugyanazt a diszkalrendszerek között tükrözött tárterületet, akkor még külön koordinálni kell, hogy melyik diszkblokkhoz melyik gépnek van írási-hozzáférési joga, és erre a koordinálásra készítették a dm-mirror-ra épülő cmirror modult.

A fent felsorolt eszközökön kívül a diszkek összegzésére valók a RAID eszközök, amelyek lehetnek hardveres vagy szoftveres eszközök, és a szoftveres RAID eszközök között is vannak tulajdonosi és nyílt forráskódú szoftverek, például a Linux Software RAID.

# 6. KVM architektúra

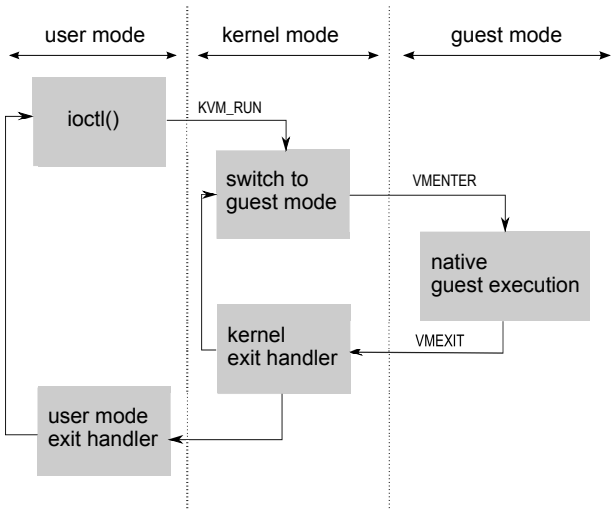
A KVM a Kernel-based Virtual Machine rövidítése. A kvm modul 2006 óta része a Linux kernelnek, és tudni kell róla, hogy nyílt forráskódú (máshogy nem is kerülhetett volna be a kernelbe), valamint csak hardveres támogatással működik (Intel VT-X vagy AMD-V). A KVM-et a Qumranet nevű cég fejlesztette ki. A Qumranetet 2008-ban megvette a Red Hat, azóta tehát a Red Hat a fő szponzora és karbantartója a KVM-nek.

A KVM alapötlete onnan származik, hogy a virtualizációhoz szükséges infrastruktúraelemek nagy része rendelkezésre áll a Linux kernelben: ütemező, memóriakezelő, a platform (x86) kezelése, eszközkezelők (device diver). Egy virtualizációs megoldás kifejlesztésekor tehát nagy lépéselőnyt jelent, ha ezekre a dolgokra kiforrott, karbantartott szoftverek állnak rendelkezésre, és a kifejezetten virtualizáció-specifikus dolgokra lehet koncentrálni.

## 6.1. Processzorvirtualizáció

A Linuxban a felhasználói folyamatok kétféle módban futnak: user mód és kernel mód. Ahogy azt a processzorvirtualizációról szóló általános fejezetben láttuk, a hardverelléréshez a user módból rendszerhívással lehet eljutni kernel módba, hogy a kernel kiadhassa a megfelelő privilegizált utasításokat, és elérje a hardvert. A

KVM virtualizáció esetében a user és kernel módon kívül bejön még egy harmadik futási mód is: a guest mód.




A virtuális gép operációs rendszere alapesetben guest módban fut. Ha I/O-ra van szükség, ha laphiba generálódik, vagy ha a virtualizációs hoszt ütemezője úgy dönt, hogy eljött az idő (preempció), akkor a vezérlés guest módból átkerül a Linux kernelbe. Ekkor a kernel eldönti, hogy melyik eset áll fönn. Amennyiben I/O-ra van szükség, akkor a vezérlést továbbítja a user módban futó QEMU felé. Ha laphiba generálódott, vagy preempció történt, akkor azokat a kernel lekezeli. Ha a QEMU kapta meg a vezérlést, akkor az kezeli a virtuális (emulált) hardvert, majd visszaadja a vezérlést a kernelnek, hogy az léptesse tovább a processzort guest módba.

A jelenlegi megvalósításban egy virtuális CPU egy Linux thread formájában fut, amelyet a Linux ütemező ütemez.

## 6.2. Memóriavirtualizáció

Mivel a KVM alatti virtuális gépek memóriáját a hosztgépen futó operációs rendszer virtuális memóriaként kezeli, ezért a virtuális gépek memóriájára is érvényesek a következő tulajdonságok, amelyek a Linux által kezelt memóriára is:

- lapozható, azaz nem használt lapjai kiírhatók a háttértárra,
- megosztható a processzek (virtuális gépek) között,
- használható hozzá large page, ezáltal elegendő a kisebb page table használata,

- használható hozzá diszk fájl,
- COW (Copy-on-Write),
- NUMA támogatás van hozzá.

Most nézzük meg a virtuális gépek közötti memóriamegosztást kicsit részletesebben. A memóriamegosztáshoz készült egy kernel modul: KSM (Kernel SamePage Merging). Működése a következő: a kernel végigpásztázza a memóriát, és az egyforma lapokat „egybeolvasztja” úgy, hogy az egyforma lapok közül csak egy maradjon. Amint az egyik gép meg akar változtatni egy olyan lapot, amely más géphez is tartozik, akkor a változtatás előtt kap ez a gép egy saját másolatot az adott lapról, és attól kezdve ezen a másolaton dolgozik. Ezáltal jelentős memóriamegtakarítások érhetők el, például abban az esetben, ha ugyanazon operációs rendszer ugyanazon verziójából futtatunk több példányt.

## 6.3. I/O virtualizáció

A QEMU szoftver végzi a hardver emulációját. Ez egy teljes I/O virtualizációt megvalósító szoftver, többféle hardverarchitektúrát tud emulálni az x86-on kívül. Linuxhoz és Windows-hoz rendelkezésre állnak paravirtualizált driverek: virtiok, amelyek a QEMU módosított változatában érhetők el.

A legújabb Red Hat által kiadott kernelben I/O pass-trough is elérhető. Teljes virtualizáció esetén a KVM az első, amellyel elérhető, hogy a virtualizációs környezet PCI pass-through segítségével dedikáljon hardvereszközt egy virtualizált gép számára. Az SR-IOV támogatással megoldható PCI eszközök megosztása több virtualizált gép között.

## 6.4. KVM desktop virtualizáció

A Red Hat a Qumranet akvizíciója után rendelkezik a desktop virtualizáció korábbi eszközeinél hatékonyabb eszközzel, a SPICE-szal, mely egy távoli renderelési protokoll. Ennek főbb jellemzői a következők:

- 30+ fps video,
- flash támogatás,
- kétirányú audio és videó támogatása, mely lehetővé teszi a telefonálást illetve videókonferenciákat,
- multi-monitor támogatás (4 vagy több monitort is),

- USB 1.1 és 2.0 támogatás,
- Adaptív távoli renderelés, amely azt jelenti, hogy a szoftver a rendelkezésre álló erőforrások vizsgálata alapján autonóm módon kiválasztja, hogy a kliens vagy a szervergépben levő videókártya (GPU) tud gyorsabban renderelni, és utána a gyorsabbal végezteti a munkát.

# 7. KVM Menedzsment eszközök

Minden virtualizációs platform annyit ér, amennyi menedzsment eszköz van hozzá. A ma elérhető menedzsment eszközök a következők:

- A libvirt virtualizációs rétegre épülő eszközök.
- Virt-manager. A libvirtre épülő GUI, amellyel listázni lehet a virtuális gépeket egy adott hosztgépen, új virtuális gépet lehet létrehozni, meglévőt lehet módosítani, indítani, leállítani stb.
- Virsh: A libvirtre épülő CLI, amelyből a főbb adminisztrálási feladatok elvégezhetők.
- A libvirt-et különböző nyelvekből lehet használni: C, python, perl, java, ruby.
- További lehetőségek: virt-v2v, virt-top, virt-ps, virt-uname, virt-dmesg, libguestfs.

A Red Hat dolgozik továbbá egy virtualizációs szoftver platformon, a Red Hat Enterprise Virtualization-ön. A platform egy központi menedzsment eszközből (RHEV-M Virtualization Management Tool) és a KVM virtualizációt futtató virtualizációs hosztokból áll. Ez a megoldás képes több hosztgépet egyszerre, egységesen kezelni. Definiálni lehet benne a következőket:

- data centert, clustert, hypervisort,
- sokféle adattárolót (NFS, IO, ISCSI, FC),
- ISO könyvtárat (csak NFS-en).

Ezzel a szoftverrel meg lehet adni, hogy bizonyos virtuális gépek automatikusan migrálódjanak egy másik virtualizációs hosztgépre. Mindez úgy működik, hogy választani lehet egyformán elosztott és energiatakarékos üzemmód (policy) között.

Az egyformán elosztott üzemmód célja, hogy ne legyen olyan hoszt, amelyiknek a processzora túl van terhelve. Ebben az üzemmódban be lehet állítani a processzor maximális terhelési szintjét. Ha egy hoszton a terhelés egy szintén előre beállítható ideig meghaladja a beállított maximális terhelési szintet, akkor arról a hosztról a szoftver elkezdi elmigrálni a virtuális gépeket.

Az energiatakarékos üzemmód célja, hogy csak a szükséges mennyiségű hoszt üzemeljen a megadott terhelési határok között. Ebben az üzemmódban a processzor maximális terhelési szintjén kívül a minimális terhelési szintet is be lehet állítani. Ha egy hoszton a terhelés egy szintén előre beállítható ideig alulmarad a beállított minimális terhelési szinten, akkor arról a hosztról a szoftver elkezdi elmigrálni a virtuális gépeket, olyan hosztra, ahol a terhelés a beállított minimális és maximális terhelési szint között van. Ha az összes hoszton kicsi a terhelés, nem kezd el migrálni, csak akkor, ha valamelyik hoszton a minimális szint fölé megy a terhelés, ebben az esetbenerre hosztra kezd migrálni.

## 7.1. Tapasztalatok éles környezetben

Munkám részeként az ULX-nél Red Hat alapú nagyvállalati datacenterek építésében és támogatásában veszek részt. Ennek keretében egyre gyakrabban kerül sor virtualizációs megoldások bevezetésére is. Az utóbbi időben a KVM alapú RHEV megoldással elég kedvező tapasztalatokat szereztünk.

Azt mondják, hogy egy technológia pontosan annyira jó, amennyire értenek hozzá, és amilyen jó eszközök léteznek a kezelésére. A számos RHEV tool közül kettőről szólnék részletesebben, mert éles környezetekben ennek volt a felhasználók oldaláról a legnagyobb visszhangja.

A sablon (template) toollal egy létező virtuális gép paramétereit és diszken található képét lehet menteni sablonként. Az így készült sablont le lehet másolni, és ezzel új gépeket lehet készíteni. Ez hasonlít a Windows rendszergazdák körében kedvelt klónozási funkcióhoz, de annyiban több annál, hogy ha a sablonról készült másolatok diszken található képe változik, akkor a memóriavirtualizációnál már említett COW elvet felhasználva csak a megváltozott blokkokról készült másolat foglal helyet a diszken. Így jelentős megtakarítást lehet elérni a háttértár felhasználásában is, de időt lehet megspórolni a másolat készítésekor is. A pillanatfelvétel (snapshot) tool segítségével egy virtuális gépről egy kritikusnak ígérkező változtatás előtt olyan felvételt lehet készíteni, amelyre adott esetben, ha a változtatás után a gép valamiért nem használható, gombnyomásra vissza lehet állni.

# LyX – Újdonságok, a hatékony szövegszerkesztés érdekében

Szőke Sándor

**Kivonat**

A LyX egy olyan program, ami a LaTeX-re épülve kihasználja annak minden előnyét. Ezáltal adva a kezünkbe egy hatékony dokumentum készítő/szerkesztő segédeszközt. A program gyors fejlődésének köszönhetően, időről-időre sok, jól használható, új funkció kerül beépítésre. Ezen újdonságok segítségével, valamint a LaTeX rugalmasságának köszönhetően, a LyX használata egyszerűsödik.

A program legnagyobb előnye, talán a legnagyobb hátránya is egyben. A használata során szakítanunk kell az eddigi megszokásokkal és át kell térnünk egy másfajta szemléletre, aminek során a művünk kerül a középpontba. Ha ezt elfogadjuk, egy nagyszerű eszköz lesz a kezünkben.

## Tartalomjegyzék

**1. Bevezetés** **137**

**2. Történelmi háttér** **137**
2.1. Elődök vászna . . . . . 137
2.2. Korszakváltás . . . . . 138
2.3. Alakhű - WYSIWYG . . . . . 138
2.4. Logikai megjelenés – WYSIWYM . . . . . 139
2.5. TeX, LaTeX . . . . . 139
2.6. LyX . . . . . 141

**3. Használat** **141**
3.1. Dokumentáció, segítség . . . . . 142
3.2. Felépítés . . . . . 142
3.3. Szerkesztés elve . . . . . 142

**4. Funkciók** **142**
4.1. Újdonságok . . . . . 144
4.2. Javított funkciók . . . . . 146

**5. Közreműködés** **147**

# 1. Bevezetés

Dokumentumunk elkészítéséhez megannyi szövegszerkesztő közül választhatunk. A legjobbat keresve találhatunk szolgáltatásokban gazdagot, gyönyörű grafikával rendelkezőt, könnyen és nehezen/komplikáltan használhatót is. Azonban olyat, amelyik gyönyörű kimenet elkészítésére képes és nagymértékben támogatja a munkánkat, keveset találunk.

Már biztosan mindenki belefutott abba a kellemetlen helyzetbe, hogy az elkészített dokumentuma két különböző nyomtatón *valamiért* máshogy jelent meg, máshol voltak a laptörések, máshogy törtek a sorok, stb. Ahhoz, hogy megérthessük miért is történhetett ez így, és hogyan tudunk rajta változtatni, érdemes visszatérnünk a gyökerekhez.

# 2. Történelmi háttér

## 2.1. Elődök vászna

A szerzőknek nem volt mindig olyan jó dolguk, mint ma. Kezdetben kézzel kellett mindent elkészíteni. Az első hordozó a kőtábla volt, illetve ahol nem volt elérhető, ott a napon kiszárított, majd kiégetett agyagtábla. Ezt váltotta fel a papirusz, mely az ókori Egyiptomból indult hódító útjára. A papiruszt, az ember kb. négy és félezer éve használja gondolatainak megörökítésére. A papirusznád[1] növény szárából és beléből készítették, és egészen a X.-XI. századig használatban volt.

Ezt követte a papír, mely Kínából származik. Sajnos a feltalálásának pontos körülményei máig tisztázatlanok. Az biztos, hogy sokat köszönhet[2] HO-TI császárnak, aki 105-ben rendelte el a papírkészítés országos elterjesztését. A papír, mint árú elsőként kereskedők útján jutott el messzi tájakra (Hajrabad közelében találtak olyan papírdarabokat is, amelyek egyik oldalán kínai írásjelek voltak, azaz félig használt papírral is kereskedtek). Sajnos a papírkészítés módját a kínaiak sokáig titkolták, ezért Európába csak kb. 1000 évvel később kalandos úton került. A X. században jelentek meg az első papírmalmok az ibériai-félszigeten.

A papír kora először a kézzel írott kódex-ekkel kezdődött, melyet a szerzetesek másoltak. Ekkor egy-egy mű elkészítése, másolása, rendkívül sok időt vett igénybe. Történhetett ez azért, mert a kódexek sok díszítést kaptak, és a másolók törekedtek minden egyes motívum pontos másolására.

A nyomtatás korának kezdete Európában[3] JOHANNES GUTENBERG nevéhez

[1] Cyperus papyrus

[2] CAJ-LUN tökéletesítette a papírkészítés módját

[3] 363-ban jelent meg az első pekingi újság

köthető [1]. Aki 1450-1455 között készítette el 42 soros kézi festéssel gazdagon díszített bibliáját. A könyvnyomtatás, a következő 100 évben egész Európában elterjedt. Ennek köszönhetően az írásbeliség és szellemi élet fejlődése új irányt vett. A diákok már nyomtatott ABC-s könyvből tanulhattak. Vándor nyomdászok oda is elvitték a könyvnyomtatást, ahol még nem volt állandó nyomdász, tőlük lehetett metszett betűformákat is vásárolni. Persze a nyomtatás elterjedésével, létrejött egy új fogalom is a: *cenzúra*, de az egy másik történet...

A papír történetéről részletesen, az [2]-ban olvashatunk.

## 2.2. Korszakváltás

A szerzőknek a művüket először *kézzel* kellett papírra leírniuk, majd ezt követhette a körülményes nyomtatás. A papírra írást forradalmasította a írógép gyártásának megjelenése. Számos európai feltaláló kísérletezett megfelelő készülékkel, az első említésre méltó a dán MALLING HANSEN által készített „Író labda„, amit 1870-ben talált fel.

Az első – mai értelemben vett írógép – a: *Sholes & Glidden írógép* [3, 4] volt, melyet 1874-ben találtak fel. Mindössze 5000 darabot adtak el belőle, de ezzel egy iparágat keltettek életre. Az utóbbi mellett, volt még egy nagy haszna: egyedi dokumentumok, levelek, hivatalos iratok létrehozásának lehetősége, jól olvasható formában. Az írógépek gyorsan elterjedtek az élet szinte minden területén.

Egészen az 1980-as évekig, meg is tudták tartani egyeduralkodó szerepüket. Azonban a számítógépek fejlődése, és a személyi számítógépek gyors elterjedésének hatására az írógépek jelentősége megszűnni látszik. Az írógép feltalálása óta eltelt 100 év erős hatást fejtett ki, és rányomta a bélyegét, az írógépek funkcióját átvevő számítógépprogramokra. Számos – jelenleg is használt – programunk hűen követi az írógépeknél megszokott technikákat.

## 2.3. Alakhű - WYSIWYG

Az írógépekkel való munka sok figyelmet követelt. Figyelni kellett mikor jön a sor vége, hogy időben el tudjuk választani a szavakat. Táblázatok készítésekor, azok tartalmát előre meg kellett tervezni. *Pontosan* ki kellett számolni, hova mi kerül. Ezt a szemléletet vette figyelembe az összes számítógépes szövegszerkesztő program. A képernyőn azt igyekezték megjeleníteni, ami a papírra kerül. Ezt hívjuk ALAKHű megjelenítésnek, vagyis ahogy a Wikipédia [5] fogalmaz: „Azt Látod, Amit Kapsz, Hűen” (az angol megfelelője „What You See Is What You Get”, azaz az „amit látsz, azt kapod”). Egy az USA-ba kivándorolt magyar CHARLES SIMONYI, BUTLER LAWSON-nal közösen fejlesztette ki az első ALAKHű szövegszerkesztőt az 1970-es évek végén.

Előnyei: könnyen használható; a pozícionáláshoz a SZÓKÖZ, ENTER billentyűk és a különböző betűméretek jól használhatóak; a megjelenítés minden részlete át-/beállítható; tartalmilag könnyen áttekinthető.

Hátrányai: nehéz eldönteni valamiről, hogy az pl. biztosan középen van-e; az eredmény, két különböző számítógépen, nem minden esetben *teljesen* azonos; a dokumentum elkészítése közben, annak külalakját folyamatosan alakítgatnunk kell.

## 2.4. Logikai megjelenés – WYSIWYM

Amint láthatjuk, az előbbi esetben inkább a megjelenés dominál, mintsem a dokumentum tartalma. Egy szövegszerkesztőnek, a szöveggel kell törődnie, és segíteni a szerzőt az alkotásban. Segítenie kell az egyes szövegrészek bevitelét, azok pozíciójának megváltoztatását, valamint a bevitt szövegek funkciójának meghatározását. Ezekkel a munkát nagymértékben fel tudjuk gyorsítani, valamint le tudjuk azt egyszerűsíteni. A WYSIWYM mozaikszó jelentése: „What You See Is What You Mean", azaz „azt kapja, amire gondol".

A dokumentumunknak, csak a *formázó* parancsokat szabad tartalmaznia, ezzel biztosíthatjuk, mind az egységes megjelenést, mind a könnyű szerkesztést. A nyomtatott kimenet elkészítését, – a dokumentumban elhelyezett formázó parancsok alapján – megfelelő szabályrendszerek figyelembevételével (elválasztás, tördelés, képek kezelése, stb.) egy számítógépes programnak kell elvégeznie. Ezzel tudjuk biztosítja, azt hogy eltérő számítógépen is azonos eredményt kapjunk.

Nos ez így szép és jó, de létezik egyáltalán ilyen rendszer? Igen és úgy hívják LaTeX!

## 2.5. TeX, LaTeX

Hogy miért is született a rendszer? Mert DONALD ERVIN KNUTH egyik művének a tipográfiája nagyon csúnyára sikerült, a kiadójának új szövegszedő rendszerével (KNUTH műve: *„The art of Computer Programming", volume 2, 1977*). Nos KNUTH emiatt merült bele a tipográfia rejtelmeibe.

Feltehetjük a kérdést mi is tulajdonképpen ez a TeX? A válasz egyszerű: a szövegszedés lemodellezése matematikai algoritmusokkal. A helyzet azonban egy kicsit bonyolultabb: KNUTH a szövegszedést elemi algoritmusokra bontotta és azokhoz *egyszerűen* paraméterezhető parancsokat rendelt. Persze mindezt, úgy vitte véghez, hogy kitalált hozzá egy nagyon jól testreszabható makrónyelvet, amivel szinte mindent meg tudott csinálni. Készített hozzá betűkészletet, mivel az még nem létezett és ezen elemekből felépítette a TeX-et.Ha jól belegondolunk, ez azt jelenti, hogy egy dokumentum valójában egy program, ami létrehozza a kívánt dokumentumot. Mindezt persze kusza parancsok tömkelegével.

A helyzet szerencsére nem reménytelen! Felismerve a rendszer jó programozhatóságát, LASLIE LAMPORT elkészített egy makrócsomagot a TEX-hez, aminek a LATEX nevet adta, ezzel *nagymértékben* megkönnyítette egy dokumentum elkészítését. A LATEX makrócsomagot használva a TEX segítségével nagyon gyorsan és nagyon jó minőségben vált lehetővé dokumentumok elkészítése. Mivel a szöveget az ALAKHŰ megjelenésnek megfelelően formázni nem kell, csak beírni és pár parancsot felhasználni, a fájl akár egy sima szöveges fájl is lehet, ezért komolyabb szerkesztőprogramra nincs szükség. Ez a tény magában hordozza a platform függetlenséget is.

A TEX életének főbb eseményei:

- 1977: kutatások elkezdése
- 1980: A TUG elindulása (lásd. [6])
- 1982: TEX 0. verziójának kiadása
- 1983: TEX 1. verziójának kiadása
- 1984: KNUTH: „The TEX Book„ kiadása
- 1984: a MetaFont v0 megjelenése
- 1984: LASLIE LAMPORT kiadja a LATEX v2.06a-t
- 1985: A „Computer Modern„ betűkészlet elkészülése
- 1985: LATEX v2.09 kiadása
- 1986: TEX 2. verziójának kiadása
- 1986: KNUTH: „The Metafont Book„ kiadása
- 1989: TEX támogatja a 8 bites karaktereket
- 1990: TEX 3. verziójának kiadása
- 1994: A LATEX3 csapat kiadja a LATEX $2_\varepsilon$-t
- 1994: LAMPORT: „A Document Preparation System„ könyv kiadása
- 2004: a LATEX2 kiadásának 10. évfordulója a TEX hivatalosan is támogatja az UTF-8-at, eTEX, PDFTEX
- 2008: XeTEX átírása MacOSX-ről linux-ra (TEXLive) és Windows-ra (W32TEX)

## 2.6. LyX

Egy LaTeX dokumentum szerkesztése közben időnként ki kell adnunk bizonyos parancsokat. Ami önmagában nem is baj, de vannak esetek, amikor egy parancs lezárójel nem megfelelő helyre kerülése, teljesen más kimenetet eredményez, mint amire számítunk. Ezen esetek elkerülése végett fogott MATTHIAS ETTRICH a program elkészítésébe, valamikor 1995 körül egy egyetemi projektként. Az a MATTHIAS ETTRICH, akiről a KDE projekt kapcsán hallhattunk.

A program történelme főbb pontokban:

- 1995. szeptember 2.: Lyrix v1.05, az első hivatkozott publikus kiadás
- 1996. január 7.: 0.8.6 verzió, új menüvel és beállító fájl támogatással
- 1996. október 30.: 0.10.7 verzió, mely tartalmaz képletszerkesztőt, ábrák és táblázatok támogatása, nemzetközi billentyűzet-kiosztások támogatása, változtatható gyorsbillentyű összerendelés, valamint végtelen visszavonást, stb.
- 1998. február 10.: 0.12.0 verzió, mely tartalmaz honosítás támogatását, javított interaktív táblázat szerkesztést, ISO8859-x javítást, új dokumentum osztályokat, szebb kinézetet, stb.
- 1999. február 1.: az 1.0 verzió megjelenése, mely tartalmaz verziókezelést, többrészes dokumentum kezelést, stb.
- 2002. május 29.: az 1.2 verzió megjelenése, mely tartalmaz új grafikai alrendszer, ERT mód, stb.
- 2003. február 7.: az 1.3 verzió megjelenése, Qt támogatásmegjelenése, azonnali előnézet, stb.
- 2006. március 8.: az 1.4 verzió megjelenése, változatok kezelése, korrektúra, extra eszköztárak, szószámlálás, stb.
- 2007. július 27.: az 1.5 verzió megjelenése, Unikód használata, CJK támogatás, többszörös nézet, vázlatszerkesztő, forrás nézet, stb.
- 2008. november 10.: az 1.6 verzió megjelenése

# 3. Használat

A programmal való ismerkedést érdemes a beépített dokumentációba való betekintéssel kezdeni. Szépen lassan haladva, mivel a szokatlan szemlélet miatt, bizonyos dolgok másképpen működnek. A programba nem rejtettek, sem repülőgép szimulátort, sem flippert, szóval csak írással és kisegítő funkciókkal fogunk találkozni.

## 3.1. Dokumentáció, segítség

A program használata közben felmerülhetnek bennünk kérdések, valamint lehetnek problémáink. A programhoz már sok-sok dokumentáció tartozik, ezek a SEGÍTSÉG menüpont alatt érhetőek el. Egyenlőre jobbára angol nyelven, de van már magyarra lefordított közöttük. Az aktuális helyzetről a magyar honlapon [7] tájékozódhatunk. Amennyiben ezekből nem találunk a kérdésünkre megoldást, írhatunk e-mail-t a levelezési listára angol nyelven [8] (jelenleg nincs magyar nyelvű lista) vagy a dokumentumok fordítóinak. Érdemes megnéznünk a program wiki oldalát [9] is.

## 3.2. Felépítés

A program felhasználói felülete 4 fő részből áll:

1. Szerkesztőablak – itt jelenik meg a szerkesztés alatt álló dokumentumunk
2. Menük – a program funkcióinak nagy része helyzetérzékeny menükön keresztül érhető el
3. Eszköztárak – melyek nagy része szintén lehet helyzetérzékeny
4. Dialógusablakok – nagy része nyitva tartható a munka során, illetve dokkolható a munkaablakhoz

## 3.3. Szerkesztés elve

Ahogy azt a 2.4 szakaszban le van írva, a szerkesztés során a bevitt szöveg funkcióját minden esetben meg kell adni. Ezt alapesetben a fájl menü alatt található legördülő választódobozból történő választással tudjuk megtenni. Két fontos dolog van, ami említésre szorul és sokaknak bosszúságot fog okozni: a szavakat 1 db SZÓKÖZ választja el, a bekezdéseket pedig 1 db pedig ENTER. Tehát a megszokott formázás a SZÓKÖZ és ENTER billentyűzettel egyszerűen nem működik!

# 4. Funkciók

A program sok LaTeX kiegészítést támogat, de sajnos minden elkészült LaTeX csomag támogatására nem szabad számítani, már csak számuk miatt sem (lásd. [6]).

A LyX, az elmúlt 14 évben nagyon sokat fejlődött, ennek köszönhetően számos olyan funkciót tartalmaz melyek feladata a szerző támogatása. Az összeset felsorolni terjedelmi okoknál fogva lehetetlen, de nézzük a legfontosabbakat.

Általános jellemzők: unikód támogatása, többrészes dokumentumok, tárgymutató, vágólap használat, nyomtatás, URL-ek, könyvjelzők, kitűnő online navigáció a dokumentumban, hatalmas méretű dokumentumok támogatása

Kereszthivatkozások: *mindig* helyesen szereplő kereszthivatkozások, amelyek szövegáthelyezés esetén is működnek, különböző dokumentumok között is.

Felsorolások: számozott felsorolás esetében, a számozással nem kell törődni, még több szint esetében sem, elem áthelyezése, szintek közötti mozgatása egyszerű

Tartalomjegyzék: *mindig aktuális* tartalomjegyzék létrehozása

Képek: sokféleképpen manipulálhatjuk a felhasznált ábrákat, valamint ábracsoportok támogatása

Lábjegyzet, széljegyzet: automatikusan számozott lábjegyzetek

Képletszerkesztő: a létező legjobb megjelenésben készíthetünk matematikai formulákat, ide érte a mátrixokat és többsoros egyenleteket, egyenletrendszereket is. A képernyőn ALAKHűen követhetjük végig a képlet létrejöttét. A bevitel gyorsításához begépelhetünk LaTeX parancsokat is, az ábrázolás azoknak megfelelően fog alakulni.

Táblázatok: támogatja a többoldalas LaTeX táblázatokat, akár oldalankénti fejléccel és lábléccel

BibTeX adatbázisok: a programmal készíthetünk hivatkozásokat BibTeX adatbázisok használatával

Úsztatások: táblázatok és képek úsztatásának lehetősége, valamint körbefuttatás

Import-export: különböző formátumok között átalakítás, beépített és külső programok felhasználásával

Helyesírás-ellenőrzés: használható programok ispell, aspell valamint egy kis trükkel a hunspell is

Honosítás: teljesen magyar kezelőfelülettel rendelkezik, támogatja a magyar karaktereket

Változatok: egy dokumentum felhasználása több célra. Pl. több csoportos dolgozatlapok tartalmazhatják a megoldásokat is, akár adatbázis szerűen (feladatszámozás mindig jó lesz). Nyomtatás előtt állítjuk be melyik változatot szeretnénk kinyomtatni

Korrektúra: szerző-lektor vagy főnök-beosztott munkafolyamat támogatása. Minden egyes változást megjelöl a dokumentumban, azokat egyesével vagy az összeset elfogadhatjuk, illetve elutasíthatjuk.

Verziókezelés: cvs, svn támogatása. A program automatikusan detektálja, ha egy tárolóból kivett fájlt nyitunk meg és képes a változások azonnali érvényesítésére, amennyiben van jogunk a tárolóhoz.

## 4.1. Újdonságok

A program minden egyes *új* verziójában számos újdonság jelenik meg. Persze vannak olyan újdonságok, amelyek becsúsznak egy-egy hibajavító kiadásba. Nézzük sorban mi minden számít újdonságnak az aktuális 1.6 verzióban.

### Munkaterületek

Több nyitott dokumentum között fülek segítségével válthatunk. Egy dokumentumhoz tartozó fület el is rejthetünk, ez hasznos lehet, sok aldokumentummal rendelkező dokumentum szerkesztésekor. Használhatunk több főablakot is, ekkor kiválaszthatjuk melyik ablakban melyik dokumentum jelenjen meg.

### Nézet felosztása

Egy munkaterületen belül a nézetet lehetőségünk van felosztani vízszintesen vagy függőlegesen. Ezáltal a dokumentumunk két részét egyszerre láthatjuk és szerkeszthetjük is.

### Helyi menük

A szerkesztőablakban bárhova kattintva jobb egérgombbal, egy helyi helyzetérzékeny menü fog megjelenni. Bizonyos elemek tulajdonságait így, a megnyitásuk nélkül is változtathatjuk.

### Teljesképernyő

Kis képernyővel rendelkező számítógépen is jól használhatóvá teszi a programot, mivel az képes a teljes képernyőt a dokumentum szerkesztésére hasznosítani. A funkció, a megszokott F11 billentyűkombinációval kapcsolható be vagy ki.

### Munkamenet kezelés

Egyes ablakok képesek megjegyezni korábbi helyüket és méretüket. Ezáltal az ismételt megnyitásuk hatására, a korábbi pozíciójukon fognak megjelenni. Ezen felül, bizonyos ablakok megjegyzik a beállításaikat is (pl. a bekezdés, vagy szöveg stílus ablakok). Valamint a programból való kilépéskor és ismételt elindításának hatására a megnyitott dokumentumok visszaállításra kerülnek (amennyiben az lehetséges), a kurzor a kilépéskori pozícióba kerül, a munka ugyanott folytatható ahol befejeztük.

### Gyorsbillentyűk

A programban a legsűrűbben használt funkciókhoz gyorsbillentyűket rendeltek, ezeket igény szerint testreszabhatjuk. Eddig egy szövegfájlt kellett szerkesztenünk ezen célból, de mostantól a program beállításai között van egy dialógusablak, ahol az egyes funkciókhoz rendelt, gyorsbillentyűk módosíthatók, törölhetőek, esetleg új összerendelés hozható létre.

### Betét buboréksúgók

Becsukott betétek tartalmát csak akkor tudjuk meg, ha kinyitjuk őket. Hogy ne kelljen mindig nyitni-csukni a betéteket, az egérrel a betét fölé állva, egy buboréksúgóban megjelenik azok tartalma, bizonyos esetekben egy-egy hasznos tanáccsal kiegészítve azt.

### hyperref támogatás

A PDF és DVI fájlokban, működő hivatkozásokat (kereszthivatkozások, URL-ek) hozhatunk létre a segítségével. Beállíthatjuk a PDF fejléc információkat, automatikus PDF könyvjelzők létrehozását, linkek megjelenését, csak úgy mint az indítási beállításokat (teljesképernyő, nagyítás).

### Automatikus kiegészítés

Bonyolult, hosszú kifejezések használata során nagy segítségünkre lehet ez a funkció. A dokumentumban található összes szót indexeli és a beírás során egyezőség esetén ajánlatokat tesz, amelyeket megjelenít. Ezzel nagymértékben gyorsíthatja a munkát, csökkentheti az elgépelések számát, illetve a helyesírás-ellenőrzésre fordítandó időt. A funkció működése jól beállítható és testreszabható.

### Szimbólumok

A LaTeX szimbólumok használatának megkönnyítése érdekébe, az aktuális kódolás bármelyik szimbólumához, gyorsan hozzáférhetünk. Ezáltal nem kell észben tartani minden használni kívánt szimbólum LaTeX parancsát.

### Képlet makrók

A matematikát oktatók számára nyújt segítséget ez a funkció. Célja: ismétlődő mintával rendelkező matematikai kifejezések létrehozásának gyorsítása. A makrókat bárhol definiálhatjuk a dokumentumban, egy külön fájlban makrógyüjteményt is létrehozhatunk. A makróknak maximum 9 kötelező és opcionális paramétere

lehet, melyek rendelkezhetnek alapértékekkel is. Egy makró módosításakor, a dokumentumban található összes előfordulás módosításra kerül. Egy aldokumentummal való munka során, a makrók akkor is jól működnek, ha azok a fődokumentumban vannak definiálva.

## 4.2. Javított funkciók

### Környezet választás

A program talán egyik legsűrűbben használt funkciója. Segítségével, tudjuk meghatározni a bevitel alatt álló szöveg funkcióját. Egy-egy dokumentumosztály esetén számos különböző környezet létezhet. A közülük történő választást segíti, a funkció szerinti csoportosítás, ABC sorrendbe rendezés vagy mindkettő egyszerre. Gyorsbillentyűje: Meta-P SZÓKÖZ vagy Alt-P SZÓKÖZ.

### URL és hiperhivatkozás

Az URL betét át lett alakítva, mindössze egy URL címet tartalmazhat. Egy hiperhivatkozás esetében viszont meg tudjuk adni, hogy a cím milyen protokoll alapján kerüljön értelmezésre. Választhatunk www, e-mail és fájl között.

### Úsztatások

Beállíthatóvá vált az úsztatások túllógása vízszintesen, a margóra való tekintettel. Szintén beállítható az úsztatásban szereplő sorok száma is. Ezzel egy valóban működő úsztatás jött létre. A táblázatok úsztatása szintén újdonság. Lehetővé vált az úsztatások elforgatása, valamint azok képesek több hasábot is átfogni. Ide tartozik még az úsztatásokba ágyazott ábrák, táblázatok, stb. támogatásának felülvizsgálata is. Mostantól ezek használata során is élvezhetjük a LyX által nyújtott szolgáltatásokat (címkék, címek, rövidcímek).

### Navigátor

Egy méretes dokumentum szerkesztése során, annak bármelyik eleméhez gyorsan és könnyedén el kell tudnunk jutni. Ezt a célt segíti a Navigátor ablak. A dokumentumban található szakaszok, képek, hivatkozások, jegyzetek, lábjegyzetek, változatok listáját tartalmazza. A szerkesztőmezőben bármelyikre kattintva, a kurzor a választott elemre ugrik. Így gyorsan és könnyedén mozoghatunk a dokumentumon belül. Hasonló funkciót találunk a menüben a NAVIGÁCIÓ menüpont alatt is.

# 5. Közreműködés

A program fejlesztői, bármilyen hozzájárulást szívesen fogadnak. Legyen az a programfejlesztésében való közreműködés, a program dokumentálása, esetleg annak honosítása. Ez vonatkozik a hibajelentésekre is. Amennyiben bárhol, bármiben valamilyen hibát vesz észre, kérjük jelezze azt a program készítőinek a honlapokon [7, 10] megadott elérhetőségeken.

A program használatához sok sikert kívánunk!

# Hivatkozások


[1] *Gutenberg, a könyvnyomtatás feltalálója*,
`http://www.oszk.hu/hun/kiallit/virtualis/nyomdak/nyomtatott_konyv_index_hu.htm`

[2] Kalmár Péter: *A kétezer éves papír*, 1980.
`http://mek.oszk.hu/01200/01208/01208.htm`

[3] *The Classical typewriter page*,
`http://site.xavier.edu/polt/typewriters/tw-history.html`

[4] *The First typewriter*,
`http://home.earthlink.net/~dcrehr/firsttw.html`

[5] `http://hu.wikipedia.org/wiki/WYSIWYG`

[6] TUG – TEX Users Group: `http://tug.org/`

[7] A LYX magyar honlapja: `http://www.lyx.hu/`

[8] LYX angol levelezõlista: `http://www.lyx.org/WebHu.MailingLists`

[9] A LYX Wiki: `http://wiki.lyx.org/`

[10] A LYX honlapja: `http://www.lyx.org/`

# Új technológiák az Ubuntuban

Torma László

## Tartalomjegyzék

**1. Bevezetés** **150**

**2. Ubuntu netbookon** **151**

**3. Az Ubuntu szoftverközpont** **153**

**4. Social from the Start** **154**

**5. Ubuntu One** **156**

**6. Konklúzió** **158**

# 1. Bevezetés

Az Ubuntu első kiadása nem is olyan régen, 2004. október 20-án jelent meg. A mindössze öt éves GNU/Linux disztribúció azonban az elmúlt időszakban óriási népszerűségre tett szert, így mára egyértelműen az egyik legmeghatározóbb szereplővé nőtte ki magát a szabad szoftverek világában. Az Ubuntu fél évente jelentkezik új kiadással, ez pedig lehetővé teszi a rendkívül gyors fejlődést. Míg a disztribúcióval napi szinten foglalkozók számára sokszor már egy kettővel korábbi kiadás (vagyis olyan, amit mondjuk 7-8 hónapja még aktívan használt) is szinte történelminek tűnik, addig a rendszer mögött álló fejlesztéseket kevésbé ismerők számára sokszor már szinte követhetetlen az új szolgáltatások és funkciók megjelenése. Ez pedig különösen igaz a 2009-es évre, amikor minden eddiginél több új érdekes fejlesztés indult be az Ubuntu körül. A 2009. október 29-én megjelent Ubuntu 9.10 (fejlesztői kódnevén a Karmic Koala) az Ubuntu eddigi történetének talán legizgalmasabb kiadása, hiszen rengeteg olyan új technológia mutatkozik meg benne, ami az elkövetkező időszakban meghatározó lehet.

Az új fejlesztések jelentős része már most is kiforrott megoldást kínál, így minden esély megvan arra, hogy a felhasználók hamar felfedezzék és megkedveljék ezeket, és néhány hónap múlva már el se tudják képzelni nélküle az életüket. Léteznek olyan projektet is azonban az Ubuntu körül, amelyeknek egyelőre csak az alapjait tették le, és a bennük rejlő valódi potenciál majd csak egy későbbi kiadásban mutatkozik meg igazán. A következőkben mindkettőre láthatunk példákat. Ennek az írásnak az a célja, hogy bemutassa az Ubuntuban megjelenő új technológiákat. Ezen belül is elsődlegesen az egyéni felhasználókat, vagyis az asztali számítógépet, notebookot és netbookot használókat érintő fejlesztéseket mutatja be. Ez természetesen nem azt jelenti, hogy ne lennének az Ubuntunak olyan fontos fejlesztései, amik kifejezetten a kiszolgálókat célozzák meg, azonban terjedelmi okokból ezek bemutatására most itt nem kerül már sor. Azért megragadnám viszont az alkalmat, hogy a rendszergazdák figyelmét felhívjam az Ubuntu Enterprise Cloud technológiára: ha komolyabb szerverparkot üzemeltetnek, vagy egyszerűen csak érdeklődnek a téma iránt, ennek mindenképpen érdemes utánanézniük.

A következőkben tehát azt mutatja be ez az írás, hogy milyen új fejlesztések jelennek meg az Ubuntuban, és hogyan befolyásolják ezek majd a felhasználók mindennapjait és szokásait. Az Ubuntu ugyanis megváltoztatja azt, ahogy az alkalmazásokat, az adatainkat, a személyes kapcsolatainkat és a számítógépünket kezeljük. Ez elsőre egyszerre meglehetősen közhelyesnek és indokolatlanul hatásvadásznak hangozhat, ha viszont gondolatban megfosztjuk a felesleges pátosztól, akkor tökéletesen leírja, hogy miről van szó: ügyes, jól célzott fejlesztésekkel az Ubuntu jelentős előrelépést nyújt azokon a tipikus területeken, amelyeket a leggyakrabban használunk. Így megkönnyíti az alkalmazások telepítését és eltávolí-

tását a kezdő felhasználók számára is, kényelmesebbé teszi az adataink tárolását, gépek közötti szinkronizálását és megosztását, segíti a különféle közösségi oldalakon való kommunikációt és kapcsolattartást, valamint barátságos, egyszerűen megtanulható és használható felületet nyújt az egyre nagyobb népszerűségnek örvendő netbookokra, vagyis a kisméretű, könnyen hordozható számítógépekre.

## 2. Ubuntu netbookon

Az Ubuntu mögött álló cég, a Canonical 2008 júniusában, a Tajvanban megrendezett Computex informatikai szakkiállításon jelentette be az Ubuntu első kifejezetten netbookokra szánt változatát, az Ubuntu Netbook Remixet. A rendszer a GNOME asztali környezetet használja, azonban az alap felületet látványosan átalakították, hogy kényelmesebbé tegyék a használatát a kis gépek viszonylag szűkösebb kijelzőjén. Ennek részeként átalakították a panelt, az indítómenü a munkaasztalon kapott helyet, és a legtöbb alkalmazás rögtön teljes képernyőn indul. A felületet az elmúlt egy évben folyamatosan finomították, nemrégiben pedig teljes megújuláson esett át, így az Ubuntu legújabb kiadásában, a Karmic Koalában már az új külsővel találkozhatunk. Ha valaki már látta a korábbi Ubuntu Netbook Remix felületet, valószínűleg elsőként az tűnik majd fel neki, hogy a korábbi három oszlopos felépítést az új kiadásban két oszlopos váltja. Míg a régebbi kiadásokban a Gnome panel „Helyek” menüjének megfelelőjét találhattuk a jobb oldalon, ez az új kiadásban a bal oldali menü egyik alpontja lett. Az új felépítésnek köszönhetően a felület így nem olyan zsúfolt, mint korábban, és már első ránézésre is jobban átlátható. Emellett újra is rajzolták az egyes elemeket, ettől pedig az egész rendszer modernebb benyomást kelt.

A Canonical a 2009-es Computexre is tartogatott izgalmas bejelentést a netbook tulajdonosok és az ilyen gép vásárlását tervezők számára: a 2009. június 2-án kiadott sajtóközleményben arról számolt be a cég, hogy a jövőben támogatni fogja a Moblin platformot. A Moblin egy kifejezetten kisméretű gépekre (Netbookokra, UMPC-kre) optimalizált platform, amit az Intel Atom processzorok köré épülő rendszerekre szánnak. A projektben olyan cégek és szervezetek vesznek részt, mint a Linux Foundation és az Intel. A Moblin egy meglehetősen sokrétű projekt, aminek mibenlétével kapcsolatban elég sok félreértéssel és tévhittel találkozhatunk. Talán a legpontosabb meghatározás az, hogy a Moblin projekt olyan, mobil gépeket célzó fejlesztések összessége, amelyek egy referencia disztribúcióban összegződnek. A Moblin ernyője alatt 27 alprojekt működik, amiből a nyílt forráskódnak köszönhetően bármelyik GNU/Linux disztribútor profitálhat. Ugyanakkor a Moblinnak létezik egy saját referencia disztribúciója is, aminek segítségével bárki kipróbálhatja a rendszert. Ez azonban nem egy végfelhasználóknak szánt változat, mindössze demonstrációs célokat szolgál. A Moblin pro-

jekt fejlesztéseire építve azonban az Ubuntu dolgozik egy saját változaton: ez az Ubuntu Moblin Remix.

A Moblin felület egyébként, bár ez első ránézésre egyáltalán nem nyilvánvaló, a GNOME technológiáira épül. Az alapot a GNOME Mobile platform biztosítja. A látványos kezelőfelület azzalFIXME a Clutter segítségével valósították meg, amire a GNOME 3.0-ban debütáló GNOME Shell is támaszkodik majd. Amikor elindítjuk a Moblint, nem egy szokásos desktop jelenik meg, hanem a my zone, vagyis a személyes zónánk. Itt láthatjuk a naptárunkat, a legfrissebb dokumentumainkat és a különféle webes szolgáltatások üzeneteit. Ez utóbbi jelenleg még csak a twittert és a last.fm-et támogatja, azonban a közeljövőben ez a kör várhatóan jelentősen bővülni fog.

A rendszer különféle szolgáltatásait a felső menüsor segítségével érhetjük el. Első helyen a fentebb bemutatott saját zónánkat találjuk. Ezt követi a status, ahol a twitterünket érhetjük el. Ettől eggyel jobbra találhatjuk a people elnevezésű szolgáltatást, ami valójában az Empathy azonnali üzenetküldő kliens motorjára, a Telepathy keretrendszerre épül, így ezen keresztül érhetjük el például a Google Talkot használó ismerőseinket. A negyedik menüpont az internet nevet viseli, és ennek segítségével a webböngészőt indíthatjuk. Ez egy klasszikus, Geckora (vagyis a Mozilla Firefox motorjára) épülő böngésző, azonban a felületét a rendszer többi részéhez igazították. A következő pont a felső menüsorban a media, ahol a képeinket, zenéinket, videóinkat találjuk. A hatodik pont a pasteboard, ahol a vágólapra kimásolt tartalmakat láthatjuk. Az utolsó előtti menüpont az applications, magyarul alkalmazások, ahonnan a programokat tudjuk indítani.

Végül az utolsó, nyolcadik pont a zones, vagyis a zónák. A Moblin platform egyik legfontosabb újítása, hogy ellentétben a klasszikus Linux desktopokkal, itt nem munkaasztalokat találunk, hanem zónákat. Amikor elindítunk egy alkalmazást, a rendszer automatikusan felajánlja a választási lehetőséget, hogy egy már meglévő zónán futtassuk, vagy hozzunk létre számára egy újat. Ha ez utóbbit választjuk, akkor automatikusan létrejön az új zóna, ha pedig bezárjuk, akkor az is megszűnik. Vagyis, ellentétben azzal, ahogy korábban megszokhattuk, nem adott számú munkaasztalunk van, hanem dinamikusan jönnek létre és tűnnek el, mindig az igényeinknek megfelelően. Ez rendkívül hasonló ahhoz, ahogy a tervek szerint a GNOME 3.0-ás kiadásában működik majd a GNOME Shell. Ez a módszer egyébként rendkívül logikus és kényelmes, és már rövid használat után is nagyon meg tudja szeretni az ember.

Az első, Moblinra épülő, kereskedelmi forgalomban kapható netbookot szeptember 23-án jelentette be a Dell: az Ubuntu Moblin Remixet futtató Dell Inspiron Mini 10v netbookok már másnap, szeptember 24-én megjelentek a gyártó kínálatában. A gépet ugyanakkor a gyártó egyelőre nem a széles közönségnek ajánlja elsősorban, hanem fejlesztőknek, tapasztalt GNU/Linux felhasználóknak és a technikai újdonságokra fogékony hozzáértőknek. Ennek megfelelően a rend-

szer elnevezése „Ubuntu Moblin Remix Developer Edition” lett, ezzel is utalva arra, hogy a kevésbé magabiztos felhasználóknak egyelőre inkább a már kiforrottabb Ubuntu változatokra épülő gépeket ajánlják.

Nemrégiben pedig egy újabb netbookokra szánt Ubuntu változatot jelentettek be: a Kubuntu Netbook Edition a Kubuntu 9.10 Karmic Koala Alpha 4 kiadásával együtt mutatkozott be. A rendszer speciálisan kialakított KDE4 asztali környezetet használ. A felület ugyanakkor nem egyszerűen a GNOME-alapú Ubuntu Netbook Remix KDE-re átültetett változata, ugyanis a Kubuntu fejlesztők meglehetősen sajátos utat követtek. A rendszer elindítása után a Newspaper elnevezésű főoldalon találjuk magunkat, ami valóban olyan, mintha egy újságot tartanánk a kezünkben: látjuk a legfrissebb, RSS-ben érkezett híreket, az aktuális és várható időjárást, a naptárat, a jegyzettömböt, sőt alul egy kockában még a kedvenc képregényünknek is jutott hely. A felül található menüből érhetjük el az alkalmazások indítására szolgáló, teljes képernyős menüt. A Kubuntu Netbook Edition első stabil kiadása október 29-én debütált a Karmic Koala megjelenésével, és természetesen bárki számára ingyenesen elérhető.

# 3. Az Ubuntu szoftverközpont

Az Ubuntu szoftverközpont, vagyis eredeti nevén az Ubuntu Software Center célja, hogy jelentősen megkönnyítse a csomagkezelést a kezdő felhasználók számára is. Bár egy kicsit tapasztaltabb felhasználó szemszögéből nézve úgy tűnhet, hogy a grafikus csomagkezelés nagyon egyszerű és barátságos Ubuntun, a kezdőket mégis igencsak össze tudja zavarni: a legegyszerűbb az Alkalmazások hozzáadása/eltávolítása nevet viselő eszköz, amiből azonban csak grafikus alkalmazásokat tudunk telepíteni. Más csomagok telepítésére ott a Synaptic. A frissítések kezelését jellemzően a Frissítéskezelő, vagyis az Update Manager végzi grafikus felületen. Külső tárolókat a Szoftverforrások nevű alkalmazással tudunk felvenni, más lehetőségek mellett. Egy-egy .deb csomag telepítésére ott a gdebi. A már nem szükséges csomagok eltávolítását pedig a Lomtalanító végzi. Ez nem csak a kezdő felhasználókat zavarhatja össze, a fejlesztők dolgát sem könnyíti meg, hogy ennyiféle eszközt kell folyamatosan karban tartani.

Ezért a jövőben ezeket egy egységes alkalmazással szeretnék kiváltani: ez lesz az Ubuntu szoftverközpont. Az új eszköz az Ubuntu 9.10-es kiadásában debütált, és jelen pillanatban a Hozzáadás/eltávolítás eszközt (gnome application installer) váltja ki, azonban a jövőre nézve ennél sokkal komolyabb tervei vannak az Ubuntu fejlesztőknek: a cél az, hogy a 2010 áprilisában megjelenő Ubuntu 10.04 LTS kiadástól (Lucid Lynx) egységes eszközként át tudja venni a Synaptic csomagkezelő, a GDebi csomagtelepítő, valamint a Szoftverforrások (Software Sources) feladatát. Az Ubuntu szoftverközpont előtt ugyan még komoly fejlesztések állnak,

de már most is jelentősen megkönnyíti a kezdő felhasználók életét: egyszerűen, átláthatóan telepíthetünk és távolíthatunk el alkalmazásokat, és az információkat is olyan formában tárja elénk, hogy az a kezdő Ubuntusok számára is könnyen áttekinthető legyen.

Bár az Ubuntu szoftverközpont a gnome application installert váltja az Ubuntu 9.10-es kiadásában, a két alkalmazás szolgáltatásaiban van némi eltérés. A korábbi alkalmazás ugyanis pontozta a programokat népszerűségük szerint (igaz, mivel egyszerűen csak a telepítések számát vette figyelembe, így az alaptelepítés részét képező csomagok mind 5 csillagos értékelést kaptak), míg az újban jelenleg nincs ilyen funkció. A gnome application installer a csomagok megjelenítésénél különféle szűrési módokra ad lehetőséget: minden alkalmazás, minden nyílt forráskódú alkalmazás, a Canonical által karban tartott alkalmazások, külső alkalmazások és csak a telepített alkalmazások. A szoftverközpont kissé másképp működik, hiszen a szűrőnek csak két opciója van: a Canonical által támogatott és minden alkalmazás. A telepített és elérhető alkalmazások között pedig a jobb oldalon található menü segítségével válthatunk.

Az Ubuntu szoftverközpont egyik legfőbb erénye, hogy a kezdők számára valószínűleg kicsit bonyolultnak tűnő és nem feltétlen egyértelmű szűrők helyett az új megoldás egyszerűen választja szét a telepített és telepíthető alkalmazásokat. A netbook felhasználók pedig valószínűleg nagyra fogják értékelni, hogy a meglehetősen zsúfolt és nehezen áttekinthető felületet egy jóval tisztább, könnyebben átlátható ablak váltja, ami még a legkisebb, 800x480 pixeles felbontású Eee PC képernyőjén is szépen elfér. Az egyes alkalmazásokkal kapcsolatos információkat pedig egyértelmű, jól áttekinthető formában tárja elénk. Ha pedig megvalósítják mindazon fejlesztéseket, amiket jövő áprilisra, az Ubuntu 10.04 LTS kiadásra terveznek, az sokkal egyszerűbbé teheti a csomagok kezelését a kezdő Ubuntu felhasználók számára is.

# 4. Social from the Start

Az egyik legizgalmasabb, azonban jelen pillanatban még meglehetősen korai stádiumban lévő projekt célja, hogy az Ubuntu már a kezdetektől fogva közösségi élményt nyújtson. A Social From The Start projektben központi szerepet tölt be a Gwibber mikroblog-kliens integrációja. A mikroblog egy olyan webes szolgáltatás, ahova pillanatnyi helyzetjelentéseket, üzeneteket küldhetünk. Az üzenetek hosszára vonatkozóan szigorú korlát van: általában 140 karakter. Ebből adódóan ezek az oldalak tipikusan arra valók, hogy gyorsan megosszuk másokkal, éppen mit csinálunk, leírjuk valamivel kapcsolatban a véleményünket egy velős mondatban, vagy feldobjunk egy érdekes linket. A mikroblog oldalak arra is lehetőséget biztosítanak, hogy megjelöljük azokat, akikre kíváncsiak vagyunk, és az

üzeneteiket egyetlen oldalon összegyűjtve olvasgathassuk. A Gwibber támogatja a legismertebb ilyen jellegű szolgáltatásokat, így például az egész divatot elindító Twittert, a nyílt forráskódú Identica-t, Brightkite-ot, a Flickr-t, a Facebookot, a FriendFeedet és sok mást.

A Social from the Start projekt célja nem egyszerűen az, hogy a Gwibber az alaptelepítés része legyen, hanem ennél többről van szó: szeretnék a rendszer szerves részévé tenni, így például integrálni a gnome-about-me (vagyis a felhasználó névjegyének) beállítási lehetőségeibe. Ehhez azonban a Gwibber működési mechanizmusát át kellett alakítani: a korábban önálló klienst szét kell választani felhasználói felületre és a háttérben futó szolgáltatásra, hogy egyszerűen tudják más alkalmazások is használni a lehetőségeit. A komponensek közötti kommunikáció így szabványos dbus üzenetekkel történhet. Az Ubuntu Karmic Koalában ennek első lépése már megtörtént, vagyis a Gwibber 2.0-ás szériájában különvált a grafikus kliens a háttérben futó szolgáltatástól. Ezzel megteremtődött a Social from the Start kezdeményezés alapja. A rendszerbe történő integráció ugyanakkor idő hiányában még nem valósult meg, így erre legkorábban csak a következő, várhatóan jövő áprilisban megjelenő Ubuntu 10.04-es kiadásában számíthatunk.

Bár az Ubuntu Karmic Koalának még nem lesz része alaptelepítésben a Gwibber 2.0, de ha használunk valamilyen mikroblog szolgáltatást, akkor érdemes lesz feltelepítenünk az alkalmazást. Különösen igaz ez akkor, ha nem csak egy ilyen szolgáltatást használunk, hanem van mondjuk Identica és Twitter regisztrációnk, de emellett Facebookot is használunk, és lehetőség szerint ezek mindegyikét szeretnénk folyamatosan követni is. A Gwibber új kiadásának ugyanis, a háttérben történt fejlesztések mellett az egyik legfőbb újdonsága, hogy sokkal jobban kezel párhuzamosan több szolgáltatást, mint a korábbi, 1.20-as verziók. A hozzáféréseinket akár fa struktúrába rendezve is láthatjuk, ez pedig megkönnyíti a különböző csatornák nyomon követését, és átláthatóbbá teszi a felületet.

A Gwibber 2.0 jelentős előrelépést jelent a korábbi kiadásokhoz képest, és egyben fontos fejlesztés lehet az Ubuntu jövője szempontjából. Bár a mikroblog még mindig újdonságnak számít, amivel csak most kezdenek megismerkedni a felhasználók, és egyelőre azt is nehéz megmondani, hogy mi lesz a fejlődés iránya, egy dolog azonban már most jól látható: a különböző forrásból érkező üzenetek nyomon követése egyre komolyabb feladatot jelent, különösen azok számára, akik kifejezetten aktív társadalmi életet élnek. Egy olyan GNU/Linux disztribúció esetében, ami a barátságosságot és az emberközeliséget helyezi középpontba, éppen ezért különösen fontos, hogy meg tudjon felelni ennek a kihívásnak. A Gwibber egy kiváló mikroblog-kliens, ami jelen pillanatban is jól használható, de ami igazán izgalmassá teszi, az a benne rejlő lehetőség: egy valódi közösségi élményt biztosító platform ígérete.

# 5. Ubuntu One

Az Ubuntu One egy Ubuntuba integrált cloud szolgáltatás, ami lehetővé teszi a fájlok tárolását, gépek közötti szinkronizálását és megosztását. Sőt ennél többre is képes, hiszen arra is lehetőségünk van, hogy az Ubuntu One segítségével szinkronizáljuk a gépeink között a Firefox könyvjelzőket, a Tomboy jegyzeteket vagy az Evolution levelezőkliensben tárolt címjegyzékünket. Ez pedig különösen kényelmes akkor, ha párhuzamosan több gépet is használunk (például egyet az irodában és egyet otthon, vagy mondjuk egy asztali gépet és egy netbookot). Az Ubuntu One része az Ubuntu 9.10-es kiadásának, és a Canonical 2 GByte ingyenes tárhelyet biztosít minden felhasználónak a kiszolgálón. Ha pedig ezt szűkösnek éreznénk, havi 10 USD előfizetési díjért 50 GByte helyhez juthatunk.

Az Ubuntu One első indítása és a hozzáférés engedélyezése után létrejön a saját mappánkban egy Ubuntu One elnevezésű könyvtár. Ezt a mappát egy háttérben futó démon folyamatosan figyeli, és ha bemásolunk ide valamit, akkor automatikusan szinkronizálja a szerverrel. A dolog a másik irányba is működik, vagyis ha feltöltünk valamit a szerverre (például a webes felületet használva vagy egy másik számítógépről), akkor pár másodpercen belül leszinkronizálja az általunk használt gépre. Az Ubuntu One arra is lehetőséget biztosít, hogy megosszunk egy-egy mappát az Ubuntu One-t használó ismerőseinkkel. Ennek két fokozata van: a mappa lehet csak olvasható a másik fél számára, vagy akár biztosíthatunk számára írási jogot. Ez utóbbival gyakorlatilag egy közösen használt mappa jön létre, aminek segítségével egyszerűen cserélhet fájlokat két fél, így megkönnyítve például a közös munkát.

Nem csak fájlokat tárolhatunk az Ubuntu One segítségével, hanem különféle alkalmazások adatait is. Ez a CouchDB segítségével történik. A CouchDB egy elosztott, hibatűrő adatbázis-technológia, amelynek nincsenek kötött sémái. A CouchDB adatbázis elemei a dokumentumok: mindegyiknek van egy egyedi azonosítója, és a dokumentumon belül nincs kötött forma. Egy dokumentumon belül lehetnek stringek, számok, dátumok vagy akár rendezett listák és asszociatív tömbök. A CouchDB, elosztott jellegéből adódóan, kiválóan kezeli a konfliktusokat, és így akkor sem kell adatvesztéstől tartanunk, ha ugyanazon az adaton egy időben két, egymással ütköző változtatás történik. Ezen tulajdonságai ideálissá teszik arra, hogy az Ubuntu One adatkezelésének hátteréül szolgáljon. A CouchDB az Erlang futtatókörnyezetre épül: ezt az Ericsson számítástudományi laboratóriumában fejlesztették ki valós idejű telekommunikációs alkalmazásra, így nagy hangsúlyt fektettek a megbízhatóságra és a folyamatos elérhetőségre, ami különösen alkalmassá teszi erre a feladatra.

A leggyakoribb kérdés, ami felmerül a felhasználókban, hogy mennyire biztonságos ez a szolgáltatás, mennyire kell aggódnunk az adataink miatt. A legfontosabb dolog, amit általában tudnunk kell az ilyen cloud technológián alapuló

megoldásokról, hogy a gyakorlatban ilyenkor az történik, hogy a fájljainkat, adatainkat felmásoljuk a szolgáltató számítógépére, így az kikerül a közvetlen ellenőrzésünk alól. Vagyis igazából teljesen mindegy, hogy egy Gmail fiókról van szó, amiben levélként küldtük haza magunknak a munkát az irodából, hogy majd otthon befejezzük, esetleg egy félig szerkesztett Google dokumentumról vagy akár a DropBoxban megosztott fájlokról, a lényeg ugyanaz marad: azt kell végiggondolnunk, hogy rá merjük-e bízni az adott fájl, dokumentum vagy adat tárolását arra a bizonyos külső félre. Ez pedig mindig személyes, bizalmi kérdés, amit az embernek magában kell meghoznia. Persze a megalapozott döntéshez nem árt tudni, hogy milyen módon tárolódnak ezek a bizonyos adatok.

Az Ubuntu One esetében a fájlok tárolása az Amazon S3 szolgáltatása segítségével történik, vagyis az adatok nem a Canonical saját szerverparkjában tárolódnak, hanem az egyik legnagyobb cloud computing szolgáltatónál, az Amazonnál. Ennek több előnye is van: egyrészt az Amazon rendelkezik már azzal a infrastruktúrával és tapasztalattal, ami egy ilyen típusú szolgáltatás biztonságos működtetéséhez szükséges. Másrészt pedig az itt tárolt óriási adatmennyiség is biztosítja, hogy ne lehessen csak úgy mazsolázni az adatainkban. Annál is inkább, mert az Ubuntu One esetében a tárolás sem egészen úgy történik, mint ahogy azt mondjuk egy tipikus FTP szervernél megszoktuk: a fájlok tartalma tömörítve található meg az Amazon S3-ban, míg a hozzá tartozó adatok, úgy mint a fájl neve, a tulajdonosa és más információk, egy külön adatbázisban tárolódnak. Vagyis még ha valaki sikeresen be is törne az Amazon S3-ra, akkor sem tudná megmondani, hogy melyik fájl tartozik hozzánk. A fájlok fel- és letöltése során sem kell aggódnunk, hiszen a szinkronizáció folyamán az adatforgalom titkosított csatornán zajlik. Mint látható, az Ubuntu One infrastruktúrája meglehetősen nagy biztonságot nyújt az adataink számára, és éppen a szeparált adatbázisból adódik, hogy még egy esetlegesen sikeres betörést végrehajtó külső fél sem tud könnyen hozzájutni az adatainkhoz. Az adatforgalmunkat megcsapoló, man-in-the-middle támadások ellen hatékony védelmet nyújt az, hogy a szinkronizáció titkosított csatornán történik. Ugyanez vonatkozik a webes felületre is, ahol már a bejelentkezés is https protokollon keresztül zajlik, így a jelszavunkat is biztonságban tudhatjuk. A szinkronizált Ubuntu One mappát tartalmazó gépeink azonosítása pedig egy egyedi tokennel történik.

Az Ubuntu One tipikusan az a szolgáltatás, ami nélkül egész jól megvoltunk sokáig, de ha egyszer elkezdi aktívan használni az ember, akkor néhány hét után már nagyon hiányozna, ha le kellene mondani róla. Különösen igaz ez akkor, ha párhuzamosan több gépet is használ. Ezen írás például több szakaszban készült, két különböző gépen, az adatcsere a gépek között pedig az Ubuntu One segítségével történt. A munkát pedig egy Tomboy jegyzetként elkészített vázlat segítette, ami szintén folyamatosan szinkronizálva volt a két gép között. Ez pedig jelentősen megkönnyítette a munkát, hiszen nem kellett arra figyelni, hogy melyik gépen melyik verzió található a dokumentumból vagy a hozzá tartozó jegyzetből, és nem

kellett külön időt vagy energiát fordítani az adatok ide-oda másolgatására. Nem csak felhasználók számára izgalmas platform az Ubuntu One, hanem a fejlesztők számára is: a CouchDB nyújtotta lehetőségeket kihasználva új szolgáltatásokat valósíthatnak meg az alkalmazásukban. Az Ubuntu felhasználói bázisa pedig elég nagy ahhoz, hogy érdemes legyen fejleszteni rá. Vagyis remélhetőleg hamarosan még több helyen élvezhetjük az Ubuntu One előnyeit.

# 6. Konklúzió

Az Ubuntu az elmúlt öt évben az egyik legfontosabb GNU/Linux disztribúcióvá vált, és ami ennél is fontosabb, hogy rendkívül nagy népszerűségre tudott szert tenni a felhasználók körében világszerte. Az Ubuntu az informatika iránt érdeklődők mellett hatásosan szólítja meg a laikus felhasználókat is, akik egyszerűen egy kényelmesen használható, barátságos operációs rendszert szeretnének a gépükre. Az új fejlesztések között pedig mindkét csoport megtalálhatja a számára vonzó szolgáltatást. A netbookok gyors ütemű térnyerésével egyre fontosabbá válnak ezek az eszközök: miközben 2008 júniusában még újdonságnak számított egy ilyen gépekre szabott Ubuntu változat, másfél évvel később már három alternatíva közül is választhatnak a felhasználók, az Ubuntu Netbook Remix pedig egy több kiadást megélt, kiforrott rendszerré vált, amivel olyan neves gyártók forgalmaznak gépeket, mint a Dell vagy a Toshiba. Az Ubuntu szoftverközpont valóban egyszerűsíti az alkalmazások telepítését és eltávolítását, így azok, akik eddig féltek a Synaptictól, és zavarosnak tartották a Hozzáadás/eltávolítás felületét, ezentúl magabiztosabban végezhetik el ezeket a műveleteket. A Social from the Start projekt azok számára lehet különösen vonzó, akik aktív társadalmi életet élnek az interneten, több közösségi oldalon is jelen vannak, és nem szeretnének lemaradni az izgalmas történésekről. Az Ubuntu One pedig rendkívül megkönnyíti a fájlok és adatok kezelését, és a már meglévő szolgáltatások mellett számos olyan potenciális lehetőséget is tartogat még, amik a jövőbeni fejlesztésekkel bontakozhatnak ki.

Az Ubuntu eddigi eredményei valóban elismerésre méltók, de ennél sokkal izgalmasabb a jövője. Az Ubuntu jelentős hatást gyakorolt arra a képre, amit az elmúlt öt évben a desktop GNU/Linux disztribúciókról gondoltunk, viszont az elkövetkezendő időszakban immáron arra gyakorolhat hatást, ahogy általában a desktop operációs rendszerekre tekintünk. A közelmúlt az online szolgáltatások előretöréséről szólt. Miközben azonban sokan azt jövendölték, hogy minden izgalmas dolog a webböngészőben fog történni, a gyakorlati tapasztalat azt mutatja, hogy az olyan tipikusan webes szolgáltatások kezelése, mint az Identica vagy a Twitter, sokkal kényelmesebb egy olyan, kifejezetten erre a célra kialakított alkalmazásból, mint a Gwibber. Bár az adatainkat online tárhelyre töltjük fel, mind-

ezt sokkal egyszerűbb egy folyamatosan szinkronizált mappa segítségével tenni, mintha a webböngészőben kattintgatva kellene csatolni és letölteni ugyanezeket a fájlokat. Egy, a böngésző nyolcadik fülén található webes jegyzetfüzetnél sokkal kényelmesebb, ha a panelről azonnal elérhetjük a Tomboy jegyzetelőnket, ami mellesleg ugyanúgy szinkronizálva van a gépeink között, de szükség esetén persze böngészőből is hozzáférhetünk. Miközben egyesek azt jövendölték, hogy az egész világot sikerül bezárni a böngészőablakba, az Ubuntu éppen az ellenkező irány létjogosultságát bizonyította: a világot ki lehet szabadítani az ablakból.

Az írásban szereplő témákkal kapcsolatban részletesebb cikkek a mogorvamormota.hu[1] weboldalon olvashatók. Ezekben további linkeket is talál, melyeket követve még több információhoz juthat.

[1] `http://mogorvamormota.hu/`

# Nyílt forráskódú elosztott rendszerek

Trencséni Márton, mtrencseni@scalien.com
Gazsó Attila, agazso@scalien.com

**Kivonat**

Az előadás tézise, hogy néhány éven belül nyílt forrású elosztott rendszerek fognak nagy skálájú, nagy megbízhatóságú webes háttérarchitektúrát szolgáltatni. Ezen rendszerek jelenleg a nagy Internetes cégek, mint Google, Amazon és Facebook által nyilvánosságra hozott architektúrák, mérnöki tapasztalatok, illetve forráskódok alapján készülnek. Az előadásban ismertetjük az iparág által felhalmozott tapasztalokat és tervezési pontokat, melyek segítségével jobban átláthatóak az elosztott rendszerek közötti különbségek, előnyök, hátrányok. Az előadás második felében a jelenleg is elérhető elosztott rendszereket ismertetjük, különös hangsúlyt fektetve a saját készítésű Keyspace kulcs-érték adatbázisunkra.

## Tartalomjegyzék

**1. Bevezetés** **162**

**2. Elosztott rendszerek alapelvei** **162**

**3. A Google architektúrája** **164**

**4. Amazon Dynamo** **165**

**5. Scalien Keyspace** **166**

**6. Más nyílt forráskódú elosztott rendszerek** **169**

**7. Konklúzió** **170**

# 1. Bevezetés

Ma már a legtöbb alkalmazás *web alkalmazás*, melyek az Interneten vagy belső hálózatokon futnak. A web alapú alkalmazások sajátossága, hogy a felhasználó adatait a szolgáltató rendszerein tárolják, és az alkalmazás futása során a szolgáltató rendszere is számításokat végez. A webes alkalmazások természetüknél fogva elosztott rendszerek, általában három különböző számítógépen fut a browser, az alkalmazásszerver és az adatbázisszerver. Egyre inkább igény van arra, hogy ezek az elosztott rendszerek, pontosabban az alkalmazás- és adatbázisréteg skálázhatók és/vagy hibatűrők legyenek, azaz minél több klienst ki tudjanak szolgálni, minél több adatot tudjanak tárolni, illetve egyes komponensek meghibásodása esetén a rendszer összeségében tovább üzemeljen.

Az előadásban bemutatandó *nyílt forráskódú* elosztott rendszereket néhány éve kezdték el fejleszteni, de egy-két kivételtől eltekintve de facto standard megoldások (mint pl. Mysql nyílt forrású adatbázisok terén) még nincsenek. Az előadás alaptézise, hogy néhány éven belül létezni fognak produkciós rendszerekben használható skálázható, hibatűrő, nyílt forráskódú rendszerek; az előadás ezeknek a rendszereknek a rövid történetével kezdődik, majd néhány, már használható rendszert mutat be, különös hangsúlyt fektetve a szerzők saját (Scalien Kft.) készítésű nagy megbízhatóságú kulcs-érték adatbázisára, a *Keyspace*-re.

Az előadás első részében általános elveket ismertetek melyek az elosztott rendszerek megértéséhez elengedhetetlenek: shared nothing architektúra, CAP háromszög, konzisztencia kérdések.

A legelső, nagyon nagy webes alkalmazásokat kiszolgáló elosztott rendszerek nagy Internetes cégeknél alakultak ki. Néhány rendzsernek cikkekben publikálták a rendszer működését, néhány rendszernek pedig kiadták a forráskódját is. A jelenleg fejlesztés alatt álló nyílt forráskódú projektek is ezen — komoly mérnöki tudást és tapasztalatokat képviselő — rendszerekből merítenek ötleteket és általános elveket, gyakran ezeket a rendszereket duplikálják. Ezért az előadás első részében *a Google Chubby, GFS, MapReduce, BigTable és az Amazon Dynamo* belső használatban lévő elosztott rendszereit ismertetem a fontosabb tervezési pontokra koncentrálva.

Az előadás második részében 1.x verziónál tartó, jelenleg is fejlesztés alatt álló, *saját fejlesztésű Keyspace* rendszert mutatom be, majd röviden összefoglalom az *Apache Hadoop* és a *Facebook Cassandra* projekteket.

# 2. Elosztott rendszerek alapelvei

A webes alkalmazások és az open-source világában az ún. *shared nothing* [1] elosztott architektúra dominál, ami lényegében azt jelenti, hogy a különálló szerve-

rek együttesen alkotnak egy elosztott rendszert, de nincsen szorosan csatolva (pl. hardveres vagy operációs rendszer szinten) a gépek memóriája (shared memory) vagy diskei (shared disk).

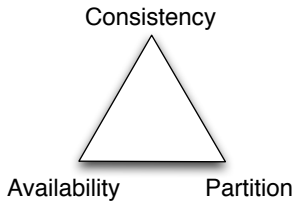
Az elosztott rendszereknél alapvető ökölszabály az ún. *CAP (consistency, availability, partition tolerance) tézis* [2], mely azt mondja ki, hogy a felsorolt három tulajdonság közül nem valósítható meg mindhárom (egyszerre shared nothing architektúrákban). A három fogalom tömör magyarázata:

1. *Konzisztencia:* az elosztott rendszerhez intézett egymást követő írás és olvasás műveletek esetén — melyeket potenciálisan más-más szerver szolgál ki — milyen garanciákat nyújt a rendszer arra, hogy az olvasás során az előzőleg beírt adatot viszontlátjuk.

2. *Rendelkezésre állás:* a rendszer képes kérések (írás és olvasás műveletek) kiszolgálására néhány szerver kiesése mellett is.

3. *Particíció tolerancia:* a rendszer hibatűrése, amennyiben a szervereket összekötő hálózat (hub, switch, router, kábel) meghibásodása esetén a rendszer kettő vagy több különálló hálózatra esik szét.

Két rövid példán keresztül ecseteljük, hogy a „CAP háromszögben”elhelyezett különböző rendszerek hogyan viselkedhetnek.




1. ábra. A CAP háromszög.

Első példaként képzeljünk el egy $n = 3$ szerverből álló rendszert, ahol induláskor mindhárom szerver szerint az `mtrencseni` felhasználó (az elosztott adatbázisban) tárolt születési dátuma `1881-04-24`, majd átjavítjuk `1981-04-24`-re, de a változás csak az 1. és 2. szerveren történik meg, és azok, mielőtt továbbítanák a változást a harmadikhoz, hiba folytán lekapcsolódnak. Újra lekérdezve a születési dátumot a még rendelkezésre álló szervertől a régi, elavult, „rossz” adatot kapjuk vissza. Egy ilyen esetet engedélyező rendszert *gyengén konzisztensnek* nevezünk. Ez a furcsa, de nagy arányban rendelkezésre álló és mindenféle particionálást magában foglaló működés előnyös, amikor egy „régi, elavult, kicsit rossz”

adat visszanyerése előnyösebb, mint hibával visszatérni. Ilyen, gyengén konzisztens rendszer például a később bemutatott Amazon Dynamo.

Második példaként vegyünk egy „többség alapú” rendszert, amelynél írás és olvasás műveletekhez a szerverek többsége rendelkezésre kell, hogy álljon. Egy ilyen rendszer csak akkor nyugtázza az írás műveletet, ha a szerverek többségén az adat kikerült a diszkre. A fenti $n = 3$ példánál maradva, egy írás műveletet akkor nyugtáz a rendszer, ha legalább két szerverre kikerült az új `1981-04-24` adat. Amennyiben két szerver kiesik, a rendszer nem tudja az olvasás műveletet kiszolgálni, mert ahhoz a szerverek többsége kell; viszont ha kettő rendelkezésre áll, akkor mindig vissza tudja adni a „jó, friss” értéket, hiszen az a rendelkezésre álló két gépből legalább az egyiken megtalálható. Az ilyen, erősebb garanciát biztosító rendszereket *erősen konzisztensnek* nevezünk. Ilyen rendszerekben a működéshez többség kell, egészséges állapotában nagyon hasonlít egy hagyományos, egyszerveres rendszerre. Az erős konzisztencia ára, hogy a szerverek többsége egy partícióban rendelkezésre kell, hogy álljon. Ilyen, erősen konzisztens rendszer például a Keyspace.

# 3. A Google architektúrája

A Google néhány évvel ezelőtt cikkek formájában nyilvánosságra hozta belső rendszerének leírását. A rendszert a Google keresőjére optimalizálták, azóta azonban teljesen más alkalmazások is futnak fölötte, pl. Google Mail és Google App Engine, ami az eredeti rendszer robosztus jellegét jelzi.

A Google architektúrája a következő elemekből épül fel:

1. Chubby: elosztott lock szerver [4].
2. Google File System (GFS): nagy teljesítményű elosztott file rendszer [5].
3. MapReduce: elosztott batch feldolgozó rendszer [6].
4. Bigtable: tábla alapú elosztott adatbázis [7].

A *Chubby* egy elosztott lock szerver, amelyet más szolgáltatások (pl. GFS vagy Bigtable) használnak jól ismert elosztott primitívként. Egy Chubby cella több tízezer másik szerveren futó elosztott rendszert szolgál ki, amelyek master választásra és konfigurációs metaadatok (pl. mely szerverek részei a rendszernek) megosztására használják a cellát. A Chubby egy erősen konzisztens rendszer, lényegében a többségi alapú Paxos [3] algoritmust valósítja meg, melyről később még lesz szó a Keyspace kapcsán.

A *Google File System (GFS)* a Google nagy teljesítményű elosztott filerendszere, melyet a keresőhöz szükséges nagy mennyiségű, szekvenciális íráshoz (pl.

weboldalak lementése), és kevesebb, véletlenszerű olvasáshoz (pl. keresésnél) optimalizáltak. Master-alapú filerendszer, ahol a master tárolja az összes metaadatot, és ún. chunkszerverek tárolják a chunkokra bontott fileokat 64MB-os blokkokban, replikálva. Érdekesség, hogy a master szerver az összes metaadatot memóriában tárolja, hogy a kliens kéréseket megfelelő sebességgel kiszolgálhassa, ami bizonyos korlátokat jelent a rendszerre nézve (metaadat mérete). Egy GFS rendszer néhány millió filet, petabyte mennyiségű adatot tárol. A metaadatok erősen konzisztensek, mivel a master szerveren keresztül történik a változtatásuk, míg az chunkok egyfajta „eventual consistency" („előbb-utóbb konzisztens lesz") modellt követnek, melyre az elosztott filerendszert használó alkalmazásnak fel kell készülnie.

Míg az eddig felsorolt rendszerek file vagy adatbázis rendszerek voltak, a *MapReduce* egy elosztott job-kezelő rendszer, melyet a Google a keresője alapjául szolgáló invertált index előállításához használ. A MapReduce lényege, hogy a feladatot a funkcionális nyelvekből ismert Map és Reduce lépésekre bontja, amelyeket a rendszer automatikusan szétoszt, és két fázisban végrehajta őket. A legegyszerűbb példa, ahogy weblapokban szavak előfordulását számolja ki: a Map lépésben egy weblapból kiszedi a $w$ szavakat, és $(w, darab)$ alakú kulcs-érték párokat ír ki; a Reduce lépésben a $w$ szót tartalmazó párokban található darabszámokat összeadja, így megkapjuk a szavak előfordulását egy adott mintában. A Map és Reduce lépések eloszthatóak, így nagy mennyiségű adatot lehet egyszerre, gyorsan feldolgozni. A Google esetében a MapReduce rendszer GFS vagy Bigtable fölött fut.

Az utolsó itt említett Google rendszer a tábla (igazából sor/oszlop) alapú Bigtable. A hagyományos relációs adatmodell helyett a Bigtable egy elosztott módon is implementálható, lényegében kulcs-érték adatmodellt kínál. A Bigtable adatokat `(sor, oszlop) -> adat` címzéssel kaphatja vissza a kliens, illetve egy plusz `verziót` is megadhat, amivel egy bizonyos adat régebbi verzióját kaphatja vissza, mert a Bigtable változtatás esetén automatikusan tárolja a régi verziókat is. A hozzáférés sor (és oszlop) szinten történik, és csak sor szintű módosítások végezhetők tranzakciósan. A Bigtable GFS fölött fut, és Chubby-t használ master kiválasztásra és metaadat tárolására.

# 4. Amazon Dynamo

Az Amazon több belső elosztott rendszert is üzemeltet: egy részük az `amazon.com` online boltot szolgálja ki, egy másik részük az Amazon Web Services (AWS) rendszert alkotják. Itt az online boltnál használt Dynamo rendszert mutatjuk be röviden egy 2007-ben kiadott cikk alapján [8].

A Dynamo egy gyengén konzisztens rendszer, melynek a célja, hogy minden

esetben kiszolgálja a kliens kéréseit — akkor is, ha nem tud teljesen friss adattal szolgálni valamilyen hiba miatt. Ezt az online bolt követeli meg, melynek mindig működnie kell („always-on experience”), ugyanis ha nem működik, akkor jól becsülhető, lényeges pénzügyi veszteséget szenved a cég. A rendszer kulcs-érték alapon működik, a kulcs-érték párok többszörösen replikálva vannak. A gyenge konzisztencia miatt ugyanazon adat több verziója is jelen lehet a rendszerben, ezért az adatokat a rendszer családfaszerűen verzióbélyegekkel látja el. Amennyiben egy adatnak több verziója van jelen, azt előbb-utóbb észleli a rendszer, és egy alkalmazásspecifikus, konfliktusfeloldó algoritmus újra előállítja a konzisztens elosztott állapotot. Ezért ezt a modellt *eventual consistency*-nek is hívják („előbb-utóbb konzisztens lesz”).

Például tegyük fel, hogy az `mtrencseni` vásárlónak két könyv van a kosarában, `A` és `B`. A vásárlás folyamata közben a felhasználó kosarát tároló szerverek hálózati hiba miatt lekapcsolódnak, ezért a rendszer nem éri el a kosár legutolsó állapotát, így a rendszer a kosarat üresnek jelzi a felhasználónak. A felhasználó érzékeli a hibát, és újra belerakja az `A` és `B` könyvet, majd kicsit később egy új `C` könyvet. Közben a hálózati hiba helyreáll, és a rendszer érzékeli, hogy a felhasználónak két különböző verziójú kosara van a rendszerben. Ilyenkor egy konfliktust feloldó, (kosár-)alkalmazásspecifikus algoritmus előállít egy új, konzisztens állapotot, pl. a kosarak unióját képzi.

# 5. Scalien Keyspace

A saját készítésű, kulcs-érték alapú *Keyspace adatbázis* az első nyílt forráskódú rendszer melyet bemutatunk. A Keyspace az eddig bemutatott rendszerek közül leginkább a Google Chubby rendszeréhez hasonlít. A Keyspace egy konzisztensen replikált adatbázis: replikált, mert az összes szerver ugyanazt az adatot tárolja; konzisztens, mert a CAP háromszögben a konzisztenciára helyezi a hangsúlyt (vs. „eventual consistency”), és garantálja, hogy sikeres írások után az olvasások tükrözik az írást, akármilyen hálózati vagy szerverhiba esetén is.

Hasonlóan a Chubby-hoz a Keyspace is a Paxos elosztott konszenzus algoritmust valósítja meg (mely egy többségi algoritmus). A Keyspace cellákat $n = 3$ konfigurációban futtatva pl. egy szerver $95\%$-os rendelkezésre állása $99.27\%$-ra javítható (ld. táblázat).

A rendszer lelkét alkotó elosztott algoritmus, a Paxos miatt a Keyspace minden praktikusan előálló hálózati vagy szerverhiba esetet kezel, és tovább üzemel, amennyiben a szerverek többsége rendelkezésre áll és kommunikál:

1. Szerverek leállnak és újraindulnak: a Keyspace programot futtató szerverek leállhatnak és újraindulhatnak, elveszítve a memóriában tárolt állapotot, de nem a diszkre kiírt adatokat.

| szerverek | többség | rendelkezésre állás |
|---|---|---|
| 1 | 1 | 95.00% |
| 2 | 2 | 90.25% |
| **3** | **2** | **99.27%** |
| 4 | 3 | 98.59% |
| 5 | 3 | 99.88% |
| .. | .. | .. |

1. táblázat. Rendelkezésre állás különböző méretű Keyspace cellák esetében.

2. Hálózati partíciók: hubok és routerek tönkremehetnek, ezért a hálózat átmenetileg részekre eshet.

3. Csomagveszteség, duplikáció és átrendeződés: operációs rendszerek hálózati stackje és routerek eldobhatnak és átrendezhetnek üzeneteket. A TCP-szerű protokollok garantálják ezen esetek kezelését, míg az UDP-szerűek nem. A Keyspace mindkét fajta hálózati protokoll fölött tud futni.

4. Hálózati késleltetések: terhelt helyi hálózatokon és WAN-okon (Internet) az üzenetek több másodperces késéssel érkezhetnek meg a címzetthez. A Keyspace minden esetben erős konzisztenciát nyújt.

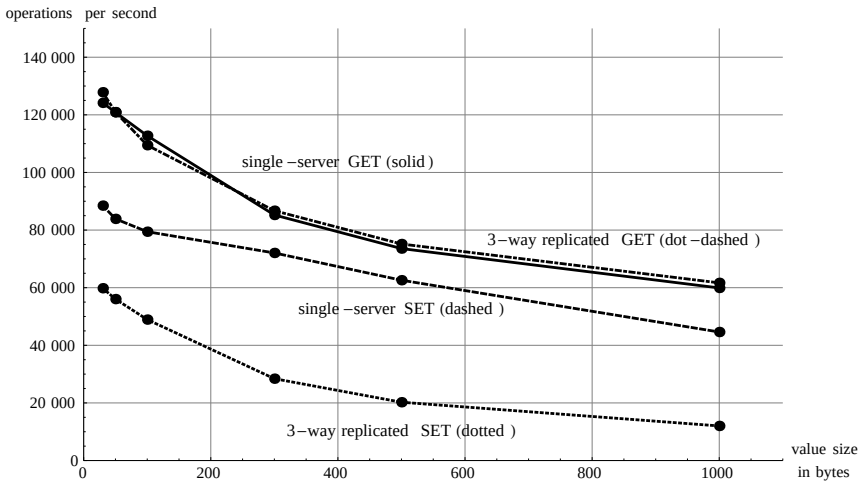
A Keyspace más kulcs-érték adatbázisokhoz képest viszonylag kiterjedt adathozzáférési API-val rendelkezik (ld. alább). Az olvasási műveleteknek létezik piszkos („dirty”) verziója is, mely semmilyen konzisztencia garanciát nem ad, viszont akár egyedülálló szerver is ki tudja szolgálni. A támogatott műveletek:

- `GET(key)`: visszaadja a `key`-hez tartozó értéket, ha létezik az adatbázisban.
- `SET(key, value)`: beállítja a `key` értékét, átírva az előző értéket, ha létezett az adatbázisban.
- `TEST-AND-SET(key, test, value)`: atomi módon átírja `key` értékét `value`-ra, ha a jelenlegi értéke `test`.
- `ADD(key, a)`: a `key` értéket számként értelmezi, és hozzáad `a`-t.
- `RENAME(key, newKey)`: átnevezi `key`-t `newKey`-re, megtartva az értékét.
- `DELETE(key)`: kitörli `key`-t és az értékét az adatbázisból.

- `REMOVE(key)`: kitörli `key`-t és az értékét az adatbázisból, visszaadja az értéket.
- `PRUNE(prefix)`: kitörli az összes kulcs-érték párt, amely `prefix`-szel kezdődik.
- `LIST-KEYS(prefix, startKey, count, next, forward)`: legfeljebb `count` kulcsot ad vissza, melyek `prefix`-szel kezdődnek, a `startKey` kulcstól indulva. Amennyiben a `startKey` kulcs nem létezik az adatbázisban, ABC szerint a következő kulcsnál kezdődik. Amennyiben `startKey` létezik, átugorható `next = true` beadásával. Ez webes „lapozott” oldalak előállításánál hasznos.
- `LIST-KEYVALUES(prefix, startKey, count, next, forward)`: ugyanaz, mint `LIST-KEYS`, de a kulcsokon kívül az értékeket is visszaadja.
- `COUNT(prefix, startKey, count, next, forward, forward)`: visszaadja a kulcsok számát, melyeket az ugyanezen paraméterekkel meghívott `LIST` adna vissza.
- `DIRTY-GET(key)`: mint az előző `GET`, de konzisztencia garanciák nélkül.
- `DIRTY-LIST-KEYS(prefix, startKey, count, next, forward)`: mint az előző `LIST-KEYS`, de konzisztencia garanciák nélkül.
- `DIRTY-LIST-KEYVALUES(prefix, startKey, count, next, forward)`: mint az előző `LIST-KEYVALUES`, de konzisztencia garanciák nélkül.
- `DIRTY-COUNT(prefix, startKey, count, next, forward)`: mint az előző `COUNT`, de konzisztencia garanciák nélkül.

A Keyspace adatbázist saját, nagy hatékonyságú protokollon, ill. adminisztációs és tesztelési célokból HTTP illetve HTTP+JSON API-n keresztül lehet elérni. A nagyhatékonyságú aszinkron architektúra miatt a Keyspace nagyszámú konkurens műveletet ki tud szolgálni (ld. köv. ábra).

A Keyspace letölthető a Scalien honlapjáról a `http://scalien.com` címen, az adatbázis és kliens library-k (C, PHP, Python) a nyílt BSD licensz alatt érhetők el.




2. ábra. Keyspace bulk adatátviteli sebességek.

# 6. Más nyílt forráskódú elosztott rendszerek

Alább a teljesség igénye nélkül felsoroljuk a fontosabb nyílt forráskódú elosztott rendszereket rövid leírással.

1. *Apache Hadoop.* Az Apache Foundation Java alapú projektje, mely az ismertetett Google architektúrát másolja. A *HDFS* a GFS, a *HBase* a Bigtable megfelelője, illetve tartalmaz *MapReduce* modult is. A Hadoopot eredetileg a Yahoo! cég fejlesztette ki, egyben a legnagyobb felhasználója is, és hasonló feladatokra használja, mint a Google saját rendszerét. A Hadoop rendszer viszonylag elterjedt és népszerű, pl. AWS-en „natív” módon lehet futtatni.

2. *Facebook Cassandra.* A Cassandra a Facebook Java alapú belső projekte, melyet az Amazon Dynamo egyik eredeti fejlesztője vezet, így sokban hasonlít ahhoz. A hangsúly a véletlen műveletek (vs. bulk írások vagy olvasások) kiszolgálásán van, könnyen skálázható, a Dynamohoz hasonlóan gyengén konzisztens („eventual consistency”). Az adatmodellt a Bigtable-től kölcsönzi: táblaszerű, de gyakorlatilag kulcs-érték alapú.

3. *Memcached.* A Memcached-et a Danga Interactive cégnél fejlesztették ki a LiveJournal szolgáltatásukhoz. A Memcached önmagában nem egy elosztott rendszer, csupán egy tisztán memóriában dolgozó, kulcs-érték alapú *cache*. A cachelés azonban annyira alapvető része egy nagy teljesítményű elosztott rendszernek, hogy ezt a viszonylag egyszerű szoftvert használják a

leggyakrabban (pl. Facebook rendszereiben is). Mivel a Memcached maga nem tud a többi szerveren futó másik Memcached példányokról, ezért az alkalmazás feladata a kulcsok szétosztása és nyilvántartása. Egy jól működő rendszerben a kérések nagy hányadát (pl. több mint 95%) kívánatos cacheből kiszolgálni, diszk hozzáférés nélkül.

# 7. Konklúzió

A webes alkalmazások és hálózatba kapcsolt eszközök terjedésével egyre több adattárolási és számítási kapacitásra van szükség a szolgáltatók oldalán, akiknek üzleti igényük, hogy szolgáltatásaik gyorsak, megbízhatóak és skálázhatóak legyenek. A szolgáltatók egy jelentős része kulturális és anyagi okokból kifolyólag nyílt forráskódú megoldásokat alkalmaz az adattárolási és feldolgozási feladatokra. Az előadás során megmutattuk, hogy a nagy Internetes cégek milyen belső megoldásokat használnak, azok milyen tulajdonsággal rendelkeznek, és ennek milyen következményei vannak (pl. konzisztencia). Végül bemutattunk néhány jelenleg is elérhető nyílt forráskódú elosztott rendszert, melyek a „nagyok” rendszerei alapján készülnek. Tézisünk szerint ezekből vagy hasonló rendszerekből fog kialakulni néhány éven belül egy *nyílt forráskódú elosztott stack*.

# Hivatkozások


[1] M. Stonebraker. *The Case for Shared Nothing*, Database Engineering, Volume 9, Number 1 (1986).

[2] E. Brewer. *Keynote Address*, Symposium on Principles of Distributed Computing (2000).

[3] L. Lamport, *Paxos Made Simple*, ACM SIGACT News 32, 4 (Dec. 2001), pp. 18-25.

[4] M. Burrows, *The Chubby Lock Service for Loosely-Coupled Distributed Systems*, OSDI ’06: Seventh Symposium on Operating System Design and Implementation.

[5] S. Ghemawat, H. Gobioff, S. Leung, *The Google File System*, 19th ACM Symposium on Operating Systems Principles (2003).

[6] J. Dean, S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, OSDI’04: Sixth Symposium on Operating System Design and Implementation (2004).

[7] F. Chang et al., *Bigtable: A Distributed Storage System for Structured Data*, OSDI'06: Seventh Symposium on Operating System Design and Implementation (2006).

[8] W. Vogels et al., *Dynamo: Amazon's Highly Available Key-value store*, SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (2007), pp. 205-220.

[9] M. Trencseni, A. Gazso, *Keyspace: A Consistently Replicated, Highly-Available Key-Value Store*, `http://scalien.com/whitepapers`.

# Független fejlesztés menete nyílt forráskódú szoftverekkel

Trencsényi József

## Tartalomjegyzék

**1. Áttekintés** **174**

**2. A Mahjong Zodiac fejlesztése és a felhasznált szoftverek** **175**
2.1. Code::Blocks . . . . . 175
2.2. GIMP . . . . . 175
2.3. SDL . . . . . 176
2.4. Cocos2D for iPhone . . . . . 176

**3. További nyílt forráskódú fejlesztőeszközök** **176**
3.1. SIO2 és a Blender . . . . . 176
3.2. OGRE3D . . . . . 177
3.3. haXe . . . . . 177

**4. Az OpenOffice.org szerepe a nyílt forráskódú játékfejlesztésben** **177**

**5. A Linux mint játékfejlesztői platform** **178**

**6. Ajánlott honlapok** **178**

# 1. Áttekintés

A független fejlesztők előtt az utóbbi időben számos lehetőség nyílt mind szoftvereik nyílt forráskódú fejlesztőeszközökkel történő kivitelezését, mind azok terjesztését tekintve. Korábban ezek a lehetőségek csak korlátozottan, vagy egyáltalán nem voltak elérhetőek.

A kereskedelmi forgalomba kerülő termékek terjesztési modelljében is jelentős változás állt be az elmúlt egy-két évben. Korábban a könyvkiadáshoz és lemeziparhoz hasonló fejlesztő-kiadó-fogyasztó lánc volt a jellemző, ami megnehezítette a kis, független fejlesztőstúdiók innovatív játékainak megjelenését. A digitális tartalomterjesztés előretörésével változás állt be. Kimaradnak a kiadók és a fejlesztők közvetlenül (fejlesztő-fogyasztó) vagy közvetve (pl. iPhone esetén fejlesztő-Appple-felhasználó) a végfelhasználóhoz juttatja el a termékét. Érdemes külön tárgyalni a casual és hardcore játékok terjesztését, mivel más a célközönségük.

A casual játékok általában olyan kisebb, szórakoztató szoftverek, amik az egész család számára kikapcsolódást nyújtanak, rövidebb, egyszerűbb játékmenettel. A hardcore játékok a kifejezetten sokat játszó embereket célozzák. Ezek nagyobb játékidejű, hosszabb tanulást igénylő, speciális területet lefedő termékek.

Linuxon, Windowson és MacOSX-en -a fejlesztő saját honlapján kívül- különböző portálokon érhetőek el ezek a termékek. Casual játékokat a következő honlapokon találunk:

- Casual Game Store: `http://www.CasualGameStore.com/`
- RealArcade: `http://www.realarcade.com/`
- Big Fish Games: `http://www.bigfishgames.com/`
- Pogo: `http://www.pogo.com/`

Hardcore játékok itt érhetők el:

- Steam: `http://store.steampowered.com/`
- Direct2Drive `http://www.direct2drive.co.uk/`

Játékkonzolokra és handheldekre a gyártók biztosítanak saját online boltokat. Ilyen pl. az iPhone AppStore, a Sony Playstation Network, WiiWare vagy az Android Market.

A fizetős játékok mellett elérhetőek – főleg Flash-es játékként – a reklámbevételekből finanszírozott ingyenes alternatívák:

- Kongregate: `http://www.kongregate.com/`

- Armor Games: `http://armorgames.com/`
- Newgrounds: `http://www.newgrounds.com/game`

Magához a fejlesztéshez számos olyan szabadon elérhető és módosítható editor, compiler, library áll rendelkezésre, ami megkönnyíti a munkát.

# 2. A Mahjong Zodiac fejlesztése és a felhasznált szoftverek

A Mahjong Zodiac egy 3-matching és mahjong keverékéből álló ügyességi játék, a demója a `http://www.CasualGameStore.com/mahjongzodiacinfo.html` oldalról érhető el. Fejlesztéséhez kizárólag nyílt forráskódú eszközöket használtunk.

## 2.1. Code::Blocks

A Code::Blocks egy szabadon elérhető, nyílt forráskódú, rendkívül jól bővíthető és konfigurálható C/C++ IDE (integrált fejlesztői környezet), mely elérhető Linux, MacOSX és Windows platformokon.

Valamennyi operációs rendszer alatt azonos kinézet és funkcionalitás segíti a gyors munkát, a hatékony fejlesztést. Felépítése moduláris, szabadon bővíthető. Valamennyi funkció, így a fordítás és a hibakeresés is pluginek segítségével kerültek megvalósításra.

A Code::Blocks import funkciót tartalmaz Microsoft Visual Studio és Dev-C++ projekt fájlok beemelésére. Az alábbi fordítókat támogatja: GCC, MSVC++, Digital Mars, Borland C++ 5.5, Open Watcom, stb. Az editor támogatja az automatikus formázását, a kód- kiemelést, kiegészítést és rejtést.

A C++ alapú forráskódja a `http://www.codeblocks.org/` címen, GNU GPLv3 licenc alatt érhető el.

## 2.2. GIMP

A GIMP egy nyílt forráskódú, bitmap grafikák készítésére és módosítására alkalmas szoftver, mellyel rendkívül rugalmasan hozhatjuk létre játékaink 2D elemeit, vagy módosíthatjuk fotókból vagy 3D renderelt forrásból érkező anyagainkat.

A GIMP moduláris felépítésű, felhasználói felülete egyedileg módosítható, saját ízlésünkre formálható. Az alkalmazás széles körű hardver támogatással bír. Scannerek és digitalizáló táblák kezelése teszi még könnyebbé a napi használatát. Ismeri az összes nagyobb fájltípust, importálja és exportálja a Photoshop psd formátumát.

Elérhető GNU GPLv2 licenc alatt Linux, Windows, MacOSX stb. platformokon az alábbi helyről: `http://www.gimp.org/`

## 2.3. SDL

Az SDL egy nyílt forráskódú grafikai és multimédia könyvtárgyűjtemény. Segítségével könnyedén érhető el az alacsony szintű hang, billentyűzet-, egér- és joystick kezelés, valamint OpenGL és frame buffer támogatás.

Támogatja a Linux, Winodws, MacOSX, xBSD, Solaris, Irix, QNX platformokat, és számos nem hivatalos portja is elérhető (pl. PSP, iPhone, NDS, AmigaOS stb.). Bár eredetileg C-ben íródott, de natív módon támogatja a C++ programnyelvet, és számos más nyelvhez is létezik interfész (pl. C#, Java, Lua, Objective-C, PHP, Python stb.).

A C alapú forrás GNU GPLv2 licenc alatt elérhető a `http://www.libsdl.org/` oldalról.

## 2.4. Cocos2D for iPhone

A Cocos2D egy eredetileg Python nyelven íródott 2D game engine, melynek natív, Objective-C-ben megírt iPhone verziója az egyik legnépszerűbb nyílt forráskódú fejlesztőeszköz jelenleg.

Teljes mértékben kihasználja az iPhone OpenGL ES alapú hardverét. A grafikai, beviteli és hang funkciókon kívül integrált fizikai engine-eket is tartalmaz (Box2D és Chipmunk). Az offline funkciók mellett egy beépített online high-score rendszer (CocosLive) segíti a fejlesztőket, hogy minél érdekesebb, a játékosok számára szórakoztató játékot készíthessenek.

A Cocos2D for iPhone forráskódja elérhető kiegészítéseket tartalmazó GNU LGPLv3 licenc alatt a `http://www.cocos2d-iphone.org/` címen.

# 3. További nyílt forráskódú fejlesztőeszközök

## 3.1. SIO2 és a Blender

A SIO2 a SIO2 Interactive által fejlesztett nyílt forráskódú 3D game engine. Különlegessége, hogy WYSIWYG editorként a szintén nyílt forráskódú Blender DCC alkalmazást használja. Integrált exporter és optimalizált adatstruktúra-szerkezetével az egyik legsokoldalúbb 3D game engine iPhone-on és iPod Touchon.

Maga az engine is nyílt komponenseken alapul. OpenGL ES 1.1 kompatibilis grafikai, Bullet fizikai, OpenAL hang- és Lua scripting motorra épül. Hátránya, hogy kizárólag iPhone-ra érhető el.

Letölthető a `http://www.sio2interactive.com/` címről. Az engine a saját, egyedi licenc szerződését tartalmazza.

## 3.2. OGRE3D

Az OGRE3D egy objektum-orientált, C++-ban készült 3D grafikai engine. Nem tartalmaz játéklogikával kapcsolatos funkciókat, kizárólag a grafikai elemek megjelenítésére korlátozódik a tudása. Exportert tartalmaz a Blenderhez, 3D Studio Max-hoz, a Mayahoz, XSI-hez, stb. Nagy előnye, hogy multiplatform. Jelenleg elérhető Linux, MacOSX, Windows és iPhone platformokra.

Desktop gépeken OpenGL 3.0 és Direct3D 9/10 renderer gondoskodik a grafika megjelenítéséről, míg iPhone-on az OpenGL ES 1.1 felel ugyanezért. A pixel és vertex shaderek kezelését Cg-vel, HLSL-lel és GLSL-lel is végezhetjük.

Forráskódja elérhető a `http://www.ogre3d.org/` címen. Az 1.6-os verzióig GNU LGPL licenc alatt került kiadásra. Az új, 1.7-es verzió már MIT licenccel.

## 3.3. haXe

A haXe egy multiplatform programozási nyelv, mely több platformra is képes fordítani. Jelenleg Flash, C++, Javascript, Neko és PHP kód generálására alkalmas a fordítója.

A haXe előnye, hogy egy forráskóddal több platform és szerver/kliens oldal is programozható, szabványos ECMA script nyelvet használ, mely erősen típusos. A fordítója sokkal gyorsabb, mint a Flex vagy Flash hasonló fordítói, valamint a generált kód is gyorsabb, mint mondjuk a Flash CS4-ben fordított. A haXe segítségével könnyedén fejleszthetünk 2D-s játékokat Flash (ActionScript), Linux, MacOSX, Windows, iPhone (C++) és Android platformokra.

# 4. Az OpenOffice.org szerepe a nyílt forráskódú játékfejlesztésben

Az OpenOffice.org egy jól használható alternatívája a drága, összetett, gyakran túlméretezett célszoftvereknek. A fejlesztés során megoldandó feladatok közt szerepel a dokumentációkészítés, a projektmenedzsment, statisztika készítése és adatkezelés.

Egy játék készítése során az első lépés mindig az ötlet kidolgozása, a design dokumentáció elkészítése. Ennek egyik eszköze lehet a Writer, amiben minden olyan feladat megoldható, ami egy megfelelő minőségű design dokumentációt eredményez. A Calc-ban készített ütemezési feladatok jól átláthatóvá és

tervezhetővé teszik még a nagyobb létszámú fejlesztéseket is, valamint a szoftver eladásának során jó szolgálatot tehet az eladási statisztikák nyomon követéséhez és a marketingstratégia kialakításához. Az Impressben a kiadók, befektetők számára látványos módon prezentálhatóak az elképzelések, korábbi munkák, jövőbeni tervek. A Base-ben adatbázisokat hozhatunk létre az elkészült assetek nyilvántartásáról, készültségi szintjéről. Elérhető GNU LGPLv2.1 licenc alatt a `http://www.openoffice.org/` címen Linux, MacOSX és Windows platformokra.

# 5. A Linux mint játékfejlesztői platform

Éveken keresztül a Linuxot nem tekintették a játékfejlesztésre alkalmas operációs rendszernek. Sem fejlesztői, sem célplatformként. A grafikus kártyák támogatása azonban már elérte azt a szintet, amikor használható alternatívát kínál a Winodwszal és MacOSX-szel szemben mint fejlesztői rendszer és mint célplatform is.

Nagyon sok esetben kényelmesebb, könnyebb Linux alatt fejleszteni, mint más platformokon. Tipikusan ilyen feladat lehet egy kliens-szerver oldali fejlesztést igénylő játék (pl. egy MMO vagy online Flash játék). A LAMP rendkívül megkönnyítheti a munkát.

Érdemes a munkához egy desktop környezetre kialakított disztribúciót választani, ami nagy felhasználói bázissal és megfelelő csomag ellátottsággal bír. Az Artex Studios 2005 óta használja ilyen célra az Ubuntu legújabb verzióit.

# 6. Ajánlott honlapok

`http://www.CasualGameStore.com/`
`http://www.gamasutra.com/`
`http://www.casualgameblogs.com/`
`http://www.casualgaming.biz/`
`http://www.indiegames.com/blog`
`http://www.mochiland.com/`
`http://www.pocketgamer.biz/`
`http://www.insidesocialgames.com/`
`http://www.gamasutra.com/`
`http://www.vgchartz.com/`

# Verziókezelés a Git használatával

Vajna Miklós
<vmiklos@frugalware.org>

## Tartalomjegyzék

**1. Miért jó a verziókezelés? 180**

**2. Az elosztott verziókezelők előnyei 181**

**3. A git előnyei 182**

**4. Felhasználóbarátság és pokol 183**

**5. Adatszerkezetek 183**

**6. Használat külsősként 185**

**7. Parancsok 186**

**8. Az index 186**

**9. Elérhetőségek 187**

# 1. Miért jó a verziókezelés?

Mielőtt megpróbálnánk megválaszolni ezt a meglehetősen összetett kérdést, próbáljuk meg részekre bontani. Bevezetésként legyen elég annyi, hogy jóval több ember használ verziókezelést, mint ahány tud erről. Gondoljunk csak arra, mikor kedvenc szövegszerkesztőnk *mentés másként* funkcióját használjuk. Meglehetősen kezdetleges módszer, de valójában máris verziókezelésről van szó: érintetlenül hagyjuk az eredeti példányt, és egy újat hozunk létre. Márpedig pont ez a verziókezelés lényege: egy projekt különböző állapotait nyilvántartani.

De hogy ne ragadjunk le a legtriviálisabb esetnél, a verziókezelés gyakorlatilag elengedhetetlen minden olyan munka esetén, ahol több ember dolgozik ugyanazon a projekten. Ha ugyanahhoz a fájlhoz ketten nyúlnak hozzá egyszerre, akkor a verziókezelő ezt vagy automatikusan kezeli, vagy segítséget nyújt az ilyen jellegű probléma feloldásához. Fontos megjegyezni, hogy a jó verziókezelő nem feltétlenül old meg mindent helyettünk, cserébe viszont saját maga kérdés nélkül nem hibázhat. Ha már hosszabb ideig használjuk a gitet, észrevehetjük, hogy vannak olyan esetek, amikor a git ütközést észlel, és a segítségünket kéri, míg ugyanazt a patch-et pl. a *patch(1)* egy figyelmeztetés mellett elfogadja. Ez pontosan azért van, mert jobb, ha nekünk kell feloldani egy ütközést, mint azt hinni, hogy minden problémamentesen lefutott, majd jóval később észrevenni (ha egyáltalán észrevesszük) az inkorrekt eredményt.

A verziókezelés ezen kívül segíti a hibakeresést. Ha van egy szkriptünk, ami reprodukálja a hibát, akkor a jó verziókezelő bináris kereséssel gyorsan megtalálja az első hibás commitot.

Végezetül a verziókezelő dokumentációs eszköz. Igényes commit üzenetek esetében egy-egy kiadás esetén a változások listáját már generálni lehet, illetve később a forráskód minden egyes sorához részletes többletinformációt találhatunk, ha a forráskódbeli megjegyzések nem lennének elegendőek.

Természetesen verziókezelő nélkül is lehet élni, legfeljebb nem érdemes. Ennek legkezdetlegesebb kiküszöbölése, mikor áttelefonálunk a kollégának, hogy *légyszi ne nyúljál most hozzá, mert dolgozom rajta*. Vagy ha az OpenOffice.org *változások követése* funkcióját használjuk, mely pár száz változtatás esetén már teljes bizonyossággal használhatatlan. Szövegfájloknál megoldható a kézi 3-utas (3-way) merge, de egy idő után ezt kézzel csinálni szintén unalmas játék. Itt jegyezném meg, hogy ha belegondolunk, alapvető igény, hogy más-más típusú fájlokat más algoritmussal merge-öljünk, mégis a legtöbb verziókezelőből kispórolták ezt a funkciót, és a merge algoritmus a verziókezelő egyik leginkább bedrótozott modulja. Természetesen a git szakít ezzel a hagyománnyal, és kedvenc programozási nyelvünkön írhatunk hozzá merge meghajtókat.

Hasonlóan érdekes, ámde értelmetlen próbálkozás a CVS névre hallgató, verziókezelőnek csúfolt program. A git szempontjából elemezve a legnagyobb prob-

léma vele, hogy nem állítható vissza maradéktalanul a projekt egy-egy időpillanatban fennálló állapota, abból következően, hogy nem a projekt teljes állapotát tárolja, hanem egy-egy fájl változásait. A *cvsps(1)* ezt próbálja meg korrigálni – több-kevesebb sikerrel.

Még egy általános problémát említenék, amiben a git szintén nem érintett. Elosztott környezetben verziószámokat használni egy-egy állapotra nem túl okos megfontolás. Ha több fejlesztő dolgozik egy projekten, akkor tipikusan egy ember feladata szokott lenni hivatalos kiadásokat készíteni, azoknak verziószámot adni már van értelme. De a köztes állapotokat valamilyen egyedi módon kell megcímezni, hiszen az 1.0 verzió után ha két fejlesztő is commitolt egy-egy változtatást, nem hívhatjuk mind a kettőt 1.0.1-nek, hiszen a kettő nem azonos.

# 2. Az elosztott verziókezelők előnyei

A git legnagyobb előnye, hogy elosztott. Sok más elosztott verziókezelő is létezik, és legtöbbjük sokkal több funkcionalitást nyújt, mint bármelyik központi verziókezelő. Számos olyan gitről szóló leírás látott napvilágot, amely nagyrészt az elosztott verziókezelők előnyeit hangsúlyozza, úgy feltüntetve, mintha ez a git kizárólagos előnye lenne. Ezzel két probléma van. Egyrészt ha erre a csúsztatásra rájön az olvasó, valószínűleg tovább sem olvassa a cikket, mert nem hiteles. A másik probléma, hogy a git még az elosztott verziókezelők mezőnyében is hihetetlen előnyökkel bír, és az ilyesfajta cikkek ezeket az előnyöket nem fejtik ki, pedig a git valójában ettől igazán innovatív, nem azért, mert sikerült a Linux kernel atyjának egy újabb elosztott verziókezelőt implementálnia.

Ennek ellenére nem feltételezhetjük, hogy mindenki tisztában van az elosztott verziókezelők előnyeivel, így röviden összefoglaljuk ezeket is. Talán a legfontosabb különbség, hogy egy repó letöltésekor nem csak az utolsó verzió kerül letöltésre, hanem a teljes repó, így minden repó egyenlő, ettől elosztott a rendszer, hogy nincs egy kijelölt központ. Ennek a következménye, hogy számos gyakran használt művelet (blame, log, diff, merge) nagyságrendekkel gyorsabb. Ez nagyon fontos, például ha a blame funkció 10 másodpercnél több időt vesz igénybe, akkor gyakorlatilag értelmetlen, a legtöbb eseten ennél több időt nem érdemes az egészre vesztegetni, akkor már inkább megnézzük az adott fájl történetét, abból is ki lehet bogarászni a számunkra érdekes információt. Tehát ha bizonyos funkciók lassúak, a felhasználók nem fogják használni, felesleges volt időt vesztegetni az implementálásukra.

Az előzőekből következik, hogy a legtöbb művelet így nem igényel hálózati kapcsolatot, eltűnik a központi repó, mint egyetlen hibalehetőség (single point of failure) problémája, megszűnik a commiter fogalma (hiszen mindenki commitolhat a saját repójába), minden egyes letöltés implicit módon egy backupot készít a

teljes repóról. Az előzőekből nem feltétlenül következik, de a git esetén a branch létrehozása és merge-ölése is nagyon egyszerűvé válik, egyrészt mert erre a kezdetektől odafigyeltek, másrészt a helyi commitok miatt.

# 3. A git előnyei

Csak a git előnyeiről külön könyvet lehet írni, így a teljesség igénye nélkül említünk párat, ami remélhetőleg arra már elég lesz, hogy az olvasó motivációt érezzen a git kipróbálásához.

Központi verziókezelők (például Subversion) esetén is létezik a branch és a merge fogalma, de meglehetősen korlátozottan. Készíthetünk egy branchet, abban dolgozhatunk, a trunkot merge-ölhetjük bele sokszor, majd ha készen vagyunk, akkor egy külön merge paranccsal merge-ölhetjük a branchünket a trunkba, és innentől hozzá ne nyúljunk a branchünkhöz, mert összedől a világ. A gitnél természetesen minden branch egyenlő, és bármelyik branchet bármelyik branch-be annyiszor merge-ölhetjük ahányszor jólesik. Összehasonlításképp például a darcs merge algoritmusa is enged elvileg ilyesmit, de nagyobb számú ütközés esetén általában végtelen ciklusba kerül. A bzr merge algoritmusa nem szenved ilyen problémával, de szinten be van drótozva a verziókezelőbe, az algoritmust nem cserélhetjük le a sajátunkra egykönnyen.

A rerere nevű szolgáltatás azt biztosítja, hogy ha egyszer feloldottunk egy ütközést, akkor ha legközelebb egy ugyanolyan ütközést kapunk, azt már automatikusan feloldja a rendszer. Ez rendkívül hasznos funkció patchsetek karbantartásakor.

A git implicit módon, futási időben és utólagosan ismeri fel a fájlok másolását és átnevezését. Mi több, ezt nem csak teljes fájlokkal, hanem nagyobb méretű kódblokkokkal is meg tudja tenni. Jelen pillanatban (2009. október) tudomásunk szerint nincs még egy verziókezelő, amely ilyen funkcionalitást nyújtana.

Példa, melyben az *a.c* fájlból átmásoltuk az *ip_get* függvényt a *b.c* fájlba, majd a git blame felismeri, hogy annak az utolsó tényeleges módosítása még akkor volt, mikor az régi fájlban volt:

```
$ git blame -C b.c
...
607001b8 b.c (Miklos 02:21:22 32)         g_free( t );
607001b8 b.c (Miklos 02:21:22 33) }
607001b8 b.c (Miklos 02:21:22 34)
^56bbfb0 a.c (Miklos 02:21:14 35) ipst_t *ip_get( char *ip_txt )
^56bbfb0 a.c (Miklos 02:21:14 36) {
^56bbfb0 a.c (Miklos 02:21:14 37)         unsigned int ip;
^56bbfb0 a.c (Miklos 02:21:14 38)         ipstats_t *l;
^56bbfb0 a.c (Miklos 02:21:14 39)         int p[4];
```

A combined diff egy olyan patch szintaxis, mely merge-ök eredményét tudja bemutatni, egyszerre összehasonlítva az eredeti saját, az eredeti másik és a végső verziót.

A git grep hasonlóan működik a *grep(1)* programhoz, viszont rekurzív grepelés esetén csak a követett fájlokat veszi figyelembe.

# 4. Felhasználóbarátság és pokol

A git tipikus UNIX eszköz, meredek tanulási görbével, azonban a szükséges tanulási szakasz után hihetetlenül hatékony eszközt kapunk. Ha az olvasó találkozott a *vim*, *emacs*, *mutt* vagy hasonló szoftverekkel, hogy csak néhány példát említsünk, akkor ismeri ezt a szituációt. Ha jobban megvizsgáljuk az okokat, az egyik leginkább szembetűnő a bőség zavara. A git 1.6.4-es verziójához összesen több, mint 600-an küldtek be módosításokat, így számos munkafolyamatot és speciális eseten támogat, melyek között elsőre nehéz lehet az eligazodás. A hivatalos dokumentáció elsősorban referencia jellegű, bár ez az utóbbi 2 évben jelentősen javult. Ezen kívül szintén az utóbbi egy-két évben több gittel foglalkozó könyv is megjelent, javítva az arányt.

# 5. Adatszerkezetek

A projekt történetét tároló objektumadatbázis adatszerkezetei meglepően egyszerűek. Négy féle objektumtípus van. A blob egy fájl egy adott változatát tárolja. A tree (fa) egy pozitív elemszámú listát tárol, melynek elemei treek vagy blobok lehetnek. Ezzel már el is tudjuk tárolni a projekt egy pillanatbeli állapotát. Mivel az állapotokat össze akarjuk kötni, bevezetésre került a commit, mely pontosan egy treere mutat, és nulla vagy több szülője lehet. Az utolsó típus a tag, ennek neve van, valamint egy objektumra mutat, ami tipikusan commit szokott lenni.

Látjuk tehát, hogy minden egyes commit tárolja a projekt teljes állapotát, valamint az egyes állapotok közötti változások (diff) mindig futási időben kerül kiszámításra. Ennek ellenére néhány esetben praktikus mégis úgy gondolni a commitra, mintha csak egy patch lenne az előző commithoz képest, a rebase kapcsán ez a szemléletmód még hasznos lesz.

A mutatók minden esetben a hivatkozott tartalom sha1 értékét tartalmazzák, aminek több előnye is van: ha két commit között csak egy fájl változott, akkor a legtöbb tree és egy kivételével az összes blob objektum újra felhasználható; a módosítás nélküli fájlmásolások és átnevezések detektálása triviális; valamint a legutolsó commit sha1-ét meghatározza a projekt korábbi összes állapota, így ha

digitálisan aláírunk egy kiadás alkalmával létrehozott taget, akkor az egyben hitelesíti a teljes korábbi történetet (kriptográfiai biztonságosság).

A git egyik legnagyobb előnye a fenti adatszerkezetek robusztussága. Ez olyannyira igaz, hogy ezeket az adatszerkezeteket kezelő könyvtárat (libgit), amely C nyelven sose íródott meg önálló formában, megírták már számos nyelven (Python, Ruby, Java, C#).

Az adatszerkezeteken kívül egy repóban még vannak referenciák, melyek a taghez hasonló módon egy-egy commitra mutatnak, viszont egyezményesen ha egy új commit születik, akkor automatikusan az új commitra illik állítani a referenciát; symrefek, amik olyan mutatók amik refekre mutatnak; hookok (hurkok) melyek bizonyos események bekövetkeztekor automatikusan végrehajtódnak; reflogok, melyek dokumentálják, hogy a referenciák milyen objektumokra mutattak korábban – ez kifejezetten hasznos patchsetek karbantartásakor; egy config (beállítási) fájl, valamint az index, melyről később még részletesen szó lesz.

Példa a reflog használatára:

```
$ git checkout "@{10 seconds ago}"
$ git checkout master
$ git log -g --pretty=oneline
402de8e HEAD@{0}: checkout: moving from 402de8e to master
402de8e HEAD@{1}: checkout: moving \
    from master to @{10 seconds ago}
402de8e HEAD@{2}: commit: B
c820060 HEAD@{3}: commit (initial): A
```

Gyakori kérdés, hogy mi a különbség a merge és a rebase között. Ha a git adatszerkezeteit nem ismerjük, akkor erre nehéz is válaszolni. A fenti bekezdések ismeretében viszont megérthetjük a különbséget az alábbi ábrák alapján. Vegyünk egy kiindulási állapotot:

$$\begin{array}{ccccccccc} & & & & A & \rightarrow & B & \rightarrow & C \quad topic \\ & & & & \uparrow & & & & \\ D & \rightarrow & E & \rightarrow & F & \rightarrow & G & & master \end{array}$$

Tehát a D…G commitok a projekt master branchét jelképezik, a fejlesztő pedig akkor, mikor az E commit volt a legutolsó a master branchben, nyitott egy új topic (téma) branchet, és ott létrehozott 3 commitot. Az ilyen topic branchek azért nagyon hasznosak, mert elindíthatjuk őket egy olyan állapotból amikor tudjuk, hogy a master branch biztosan stabil, dolgozhatunk nyugodtan, úgy, hogy mások nem zavarják a munkánkat, majd mikor az adott témát befejeztük, merge-ölhetjük a topic branchet a masterbe.

Nézzük, mi történik rebase esetén:

$$\begin{array}{cccccccccc} & & & & & & A' & \rightarrow & B' & \rightarrow & C' & topic \\ & & & & & & \uparrow & & & & & \\ D & \rightarrow & E & \rightarrow & F & \rightarrow & G & & master & & & \end{array}$$

Azt látjuk, hogy az A…C commitokat patchként tekintettük, elmentettük, az A…C commitokat eldobtuk, majd a G commitra raktuk rá a patcheket, ezzel új A…C commitokat létrehozva. Mivel a régi A szülője az E volt, az újé a G, emiatt az A commit sha1-e más lesz. Ez akkor hasznos, ha például az A…C commitok egy patchsetet képeznek, és a projekt 1.0 verziójáról a 2.0 verzióra akarjuk azt frissíteni. A rebase közben a git minden olyan patchnél, amit nem sikerült alkalmazni megáll, és lehetőséget biztosít, hogy feloldjuk az ütközéseket. Ez a megoldás szép historyt generál, hiszen úgy tűnik, mintha eredetileg is a 2.0-hoz készült volna a patchset, vagy például egy hibát is javíthattunk közben a B commitban, és azt hihetik a többiek, hogy eredetileg is úgy (tökéletesen) sikerült.

Felmerülhet a kérdés, hogy jó-e, ha az ilyen hibákat takargatják. Ha belegondolunk, hogy a későbbi hibakereséshez fontos, hogy minden *végső* (ami egy logikai fogalom, tehát például a master branchbe kerülő) commit olyan kell legyen, hogy lefordul a forráskód, akkor el kell ismernünk, hogy ez egy hasznos funkció. Ha a karbantartó kap egy jó patchsetet, amit be szeretne olvasztani, de van egy olyan patch, ami fordítási hibát okozna, akkor ezt így ki tudja javítani.

Nézzük, mi lesz merge esetén:

$$\begin{array}{cccccccccc} & & A & \rightarrow & B & \rightarrow & C & \rightarrow & H & topic \\ & & \uparrow & & & & & \nearrow & & \\ D & \rightarrow & E & \rightarrow & F & \rightarrow & G & & & master \end{array}$$

Azt látjuk, hogy egyetlen új commit született, aminek két szülője van. Így maradandó nyoma marad, hogy az A…C commitok egy külön branchben készültek. Ennek is megvan a maga előnye. Ha például valaki egy másik patchsetet készít a régi C commitra alapozva, akkor azt triviális rebase-elni a H commitra, míg ha a régi C-ről akar rebase-elni az új C-re, azt kézzel kell megadnia, hiszen a repónak nincs információja arról, hogy az új C commitnak volt régi változata is.

# 6. Használat külsősként

Ez az a szituáció, mikor találtunk egy projektet, tetszik, ahogy működik, de pár funkciót meg akarunk benne valósítani. Ilyenkor sorban megvalósítjuk a funkciókat, például minden egyes funkciót egy-egy commitban. Ahhoz viszont nem lesz

jogunk, hogy a saját repónkból ezeket a commitokat a projekt repójába írjuk. Tehát külsősök vagyunk. Ezt a módot is nagyon jól támogatja a git, a git format-patch paranccsal tudunk patchsetet generálni például egy branch csak helyben elérhető commitjaiból. Egy külsős sose merge-öl, hanem mindig rebase-el. Ez is jól támogatott, a helyi commitok sorrendjét át tudjuk rendezni, darabolni tudjuk őket, összeolvasztani. A karbantartó oldalán pedig az emailben vagy fájlként megérkezett patchsetet a git am parancs tudja újra commitokká alakítani.

# 7. Parancsok

A git 1.6.4 145 paranccsal érkezik, ezt elsőre nehéz áttekinteni. A legszűkebb részhalmaz az a néhány, amit a git help jelenít meg, először ezekkel érdemes megismerkedni. Egy tágabb halmaz a *porcelain* nevet viseli, mely a magas szintű parancsokat tartalmazza. Ezek azok, amiket egy tipikus verziókezelőben elvártnak gondolunk. Végül egy másik nagy halmaz a *plumbing* (csővezeték) nevet viseli, mely alacsony szintű parancsokat tartalmaz. Ezek paraméterezése, valamint a parancsok kimenete visszafele mindig kompatibilis, szkriptekből ezeket érdemes használni. Ezen parancsok létezésének eredménye az, hogy számos alternatív felhasználói felület is készült a githez.

Egy érdekes kezdeményezés támogatása a gitben a fast-import, illetve a fast-export parancs. Ezek a repó tartalmát egy verziókezelő-független folyammá alakítják, és ilyen importer és exporter létezik más rendszerekhez is, például a bzr-hez, darcs-hoz, mercurialhoz. Nyilvánvaló előnye, hogy így $N$ verziókezelő esetén csak $2 * N$ programot kell írni, és nem $N * N$-et.

# 8. Az index

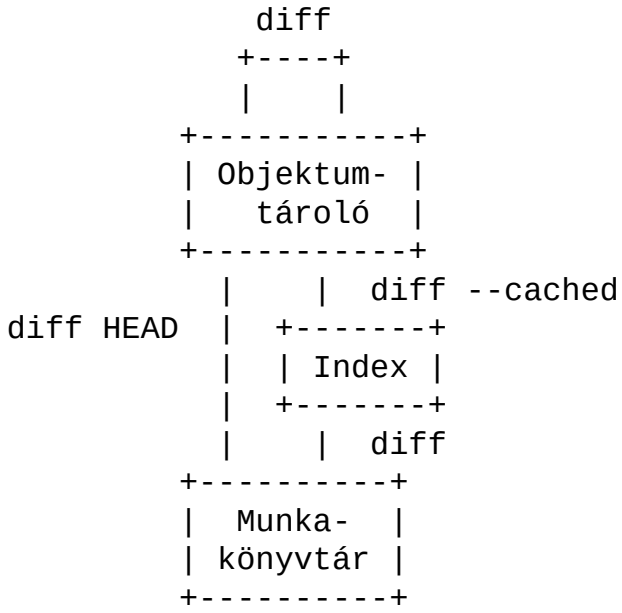
Az index egy köztes réteg a munkakönyvtár és az objektum-adatbázis között. A munkakönyvtárban változtatjuk meg a fájlokat, majd ezen változtatások egy részét az indexbe rakjuk (staging), végül a commit csupán csak az index tartalmát másolja az objektum-adatbázisba.

Új felhasználók gyakran elfelejtik ezt a fontos részletet. Ennek következménye például az, hogy ha a git add paranccsal egy létező, a munkakönyvtárban módosított fájlt az indexhez adunk, a fájlt újra módosítjuk, majd commitolunk, akkor a fájl indexbeli verziója lesz commitolva, nem a munkakönyvtárbeli!

Ez több szempontból is hasznos. Akkor például, ha egy fájlban két külön helyen két változtatást eszközöltünk, de a következő commitba csak az egyiket szeretnénk berakni. Vagy a karbantartónak is hasznos: ha merge-öl, és sok fájl változott, de csak egyben van ütközés, akkor a többi fájl bekerül az indexbe, és ha a

munkakönyvtárat összehasonlítjuk az indexszel, akkor csak a számunkra érdekes részt, az egyetlen ütköző fájl változásait fogjuk látni, a többi változást nem.

A három réteg (objektum-adatbázis, index, munkakönyvtár) közötti diffelés eszköze a git diff, git diff –cached és a git diff HEAD parancs:




# 9. Elérhetőségek

A szerző ezúton is elnézést kér, hogy sok – a cikkben szereplő – témát csak érintőlegesen említett, mint az korábban szóba került, a témáról vastag könyvet lehetne írni, a cél leginkább a figyelemfelkeltés volt. Az alábbi linkek további kérdések esetén remélhetőleg segítséget nyújtanak.

- GIT honlap: `http://git-scm.com/`
- Levelezési lista: `http://vger.kernel.org/vger-lists.html#git`
- IRC: #git @ irc.freenode.net
- A diák és ezen cikk elérhetősége: `http://vmiklos.hu/odp/`

ISBN 978-963-06-8276-3