Rajt! – UHU-Linux Office 1.2

2005 márciusában, az előző verziót követő egy éves fejlesztési időszak befejeztével az UHU-Linux Kft. kiadta az UHU-Linux Office 1.2-es változatát, mely a „Rajt!” kódnevet kapta. Ezt mutatja most be Koblinger Egmont, a cég egyik fejlesztője.

Célunk egy olyan disztribúció fejlesztése és tökéletesítése, amely a kezdő felhasználók számára is könnyedén és hatékonyan használható általános, mindennapos otthoni és irodai feladatokra, valamint kiemelkedően támogatja a magyar nyelvet és a magyar piac egyes speciális igényeit.
Ugyanakkor fontosnak tartjuk, hogy szakmai szemmel nézve is jó minőségű, „rendesen összerakott”, élvonalbeli megoldásokat tartalmazó rendszert nyújtsunk át felhasználóinknak, melyet a haladók könnyű szerrel használhatnak fejlesztésre, szerverüzemeltetésére vagy egyéb kevésbé szokványos feladatokra.
A gyakori félreértések miatt azonban leszögezném, hogy rendszerünket nem azoknak szánjuk, akik egy operációs rendszer összerakásának technikai részleteit szeretnék elsajátítani, avagy saját egyéni ízlésük és hitük (nem ritkán tévhitük) szerint testre szabni a rendszer egyes mélyebben fekvő részeit. Az ***UHU-Linux*** nem egy barkácsolási hajlamok kiélésére szánt „rakd össze magadnak” típusú rendszer, hanem mi igyekszünk a legjobb tudásunk szerint összerakni és így egy előre elkészített, tesztelt egészként nyújtani a felhasználók számára.
Az ***UHU-Linux 1.1*** és ***1.2*** között a fejlesztési erőforrásaink nagy részét a rendszer mélyebben fekvő komponenseinek átszervezésére fordítottuk, és a korábbi kiadásokhoz képest relatíve kevesebb (de még mindig nem kevés) újdonságot hoztak a kezdőbb felhasználó által is látott grafikus felületek és alkalmazások. Írásommal ezért nem a ***Linuxszal*** most ismerkedő, hanem inkább a haladóbb, technikai részletek iránt fogékonyabb olvasókat célzom meg, felvázolva az ***UHU-Linux 1.2*** által nyújtott technológiai újdonságokat.

Hardverigény, telepítés

Csomagjainkat az ***Intel Pentium*** (586-os) utasításkészletre fordítottuk, így a rendszer futtatásához ezeket ismerő processzorra van szükség. Természeten minél gyorsabb, annál jobb, a grafikus környezetekhez legalább 500 MHz ajánlott. Az ***UHU-Linux*** alapértelmezésben támogatja és kihasználja a többprocesszoros ***(SMP)*** rendszereket és a ***HyperThreading (HT)*** technológiát.
A rendszer telepítéséhez legalább 96 MB, futtatásához legalább 64 MB RAM szükséges, ugyanakkor a népszerűbb grafikus környezetekhez 256 MB igencsak ajánlott. Amennyiben csak 64 MB memória van a gépben, és a telepítést kölcsönkért memóriamodulokkal sem tudjuk megoldani, még mindig van egy kerülő lehetőség. Indítsuk a telepítőt hibakereső módban, a kapott parancssorban először particionáljuk a merevlemezt a `cfdisk` paranccsal, majd inicializáljuk és aktiváljuk a cserepartíciót az `mkswap` és `swapon` parancsok segítségével. Ezt követően már 64 MB memóriával is eldöcög a grafikus telepítő. (A telepítéskor a nagyobb igényt az indokolja, hogy a telepítő kódja teljes egészében a memóriában ül.)
Az első CD-ről történő teljes telepítéshez legalább 3 GB lemezterület javasolt.
A telepítés menete lényegében megegyezik az 1.1-es ***UHU***-éval. Egy nagyobb kaliberű változtatás történt, ez pedig a CD-n használt ***rendszerbetöltő (boot loader)*** cseréje. Az 1.1 idején még a ***syslinux CD***-re szánt változatát, az ***isolinux***-ot használtuk. Idő közben azonban a ***GRUB*** rendszerbetöltőben (melyet a telepített ***UHU-Linux*** indítására már korábban is használtunk) megjelent a CD-ről bootolás támogatása. Mivel a ***GRUB*** egy sokkal rugalmasabb rendszerbetöltő, a CD lemezen is átálltunk ennek a használatára. A ***GRUB*** egyik óriási előnye, hogy nemcsak az előre elkészített bejegyzések indíthatók, hanem (szöveges felületéből, melyre az ***Esc*** gombbal léphetünk ki) tetszőleges fájlt betölthetünk indítandó ***kernel*** és ***initrd*** gyanánt, illetve tetszőleges partíció boot szektorára is ugorhatunk. Ily módon egy kis gépeléssel akár a korábban telepített ***Linux*** rendszert is könnyedén el tudjuk indítani, ha a merevlemez elején lévő rendszerbetöltő megsérül. A ***GRUB*** részletes használatáról, beleértve az imént említett lehetőséget is, annak grafikus felületén a hypertext rendszerű súgóban olvashatunk.
A régi ***isolinux*** rendszerbetöltő a második CD elejére került, vésztartaléknak arra az esetre, ha valahol a ***GRUB*** indítása problémába ütközne. Ebben az esetben a második CD-ről indítsuk a rendszert, és amint bejelentkezik a bootképernyő, cseréljük ki a CD-t az elsőre, ezt követően válasszuk a telepítés opciót.

A kernel

Disztribúciónk előző verziója még a ***Linux*** kernel 2.4-es verzióján alapult, a mostani kiadás viszont már a 2.6-os sorozatra épült. A 2.6-os számtalan újítást hozott, egy teljes disztribúció felépítése tekintetében többet, mint a korábbi fő kernel verziók. Így ennek a lépésnek köszönhetően lehetővé vált több technikai váltás is, melyek a rendszert egyszerűbbé, áttekinthetőbbé és sokkal modernebbé tették.

Többszálú futtatás

A ***POSIX*** szabványok régóta rögzítik azt a programozói interfészt, mely segítségével több szálon futó alkalmazást lehet írni. Ugyanakkor a Linux kernel 2.4-es verziója még nem támogatta a több szálon futó folyamatokat. Éppen ezért a POSIX szálkezelést kivitelező függvénytár, az úgynevezett ***LinuxThreads*** minden egyes szálat külön kernelfolyamatra képzett le. Ennek eredményeképp a szálak közti váltás tulajdonképpen teljes folyamatváltást igényelt a kernel részéről. A 2.6-os kernelben jelent meg a több szálból álló folyamatok tisztességes támogatása, mely például a szálak közötti jóval kisebb erőforrás-igényű váltás miatt komoly teljesítménynövekedéshez vezethet többszálú programok esetén.
A kernelben lévő támogatással ugyanakkor semmit sem érnénk, ha azt a felhasználói programok nem használnák ki, vagyis ha a régi ***LinuxThreads*** függvénykönyvtárat meghagynánk. A kernelszintű szélkezelés kihasználása a függvénytár részéről olyan gyökeres átalakítást igényelt, hogy a fejlesztők a ***LinuxThreads*** jobbá tétele helyett újragondolt, újratervezett, elölről megírt alternatív szoftver mellett döntöttek, ez pedig az ***NPTL (Native POSIX Thread Library)*** nevet kapta.
A disztribúcióban tehát lecseréltük a régi ***LinuxThreads***-et az új ***NPTL***-re. A `ps ax` parancs kimenetében a többszálú programok (***Java***-alkalmazások (például ***jdictionary***), ***nscd***) ***LinuxThreads*** használatával több példányban jelentek meg, az NPTL rendszerrel azonban már egy folyamatként látszanak. A ***/proc*** fájlrendszerben a folyamat azonosítója alatt a ***task*** könyvtárban látszik, hogy az adott folyamat valójában több szálból áll.
Az új rendszer nemcsak jobb teljesítményt nyújt a többszálú programok futtatása előtt, hanem a szabványban leírtaknak (például szignál küldése) is pontosabban felel meg. Ugyanakkor, mint szinte minden ilyen átállás, apró hátulütői is vannak. Az új ***UHU*** binárisainak futtatásához 2.6-os kernel szükséges, 2.4-es kernel nem képes futtatni a programokat, így például nem lehetséges futás közben a régi ***UHU-Linux*** rendszert 1.2-re frissíteni (a frissítés CD-ről bootolva végezhető el), illetve futó ***UHU-Linux 1.1*** alatt nem lehetséges ***UHU-Linux 1.2***-höz csomagot gyártani sem. Továbbá az új rendszer a ***glibc 2.0***-s verziójával linkelt programokat már nem tudja futtatni, csak azokat, melyeket a ***glibc***-nek legalább az (egyébként immár 6 évvel ezelőtt megjelent) 2.1-es változatához fordítottak.

Sys fájlrendszer

Megjelent egy új virtuális fájlrendszer, a ***sysfs***, melynek tartalma a ***/sys*** csatolási pont alatt érhető el. Itt a kernel a rendszer hardver felépítésével kapcsolatos információkat teszi egységes formátumban elérhetővé a kíváncsi alkalmazások (leginkább rendszerprogramok) számára.

Udev eszközkezelés

A ***Unix*** rendszerek a legtöbb hardver eszközhöz és egy-két speciális szolgáltatáshoz a ***/dev*** könyvtár alatti virtuális fájlokon keresztül biztosítanak hozzáférést. A ***/dev*** könyvtár tartalmának összeállítása nem egyszerű feladat.
A legősibb megközelítés a statikus ***/dev*** könyvtár. Az összes lehetséges hardver eszközhöz tartozik egy bejegyzés, így nemritkán akár kétezer speciális fájl is lehet a ***/dev*** alatt.
A ***/dev*** könyvtár tartalma ezáltal nehezen menedzselhető, átláthatatlan, nehézkes a fájlok tulajdonosának, csoportjának és hozzáférési jogainak beállítása és karbantartása.
A bejegyzések nagy része pedig értelmetlen, fölösleges az adott gépen.
Amennyiben olyan eszközt próbálunk meg megnyitni, amelyhez nem tartozik meghajtó a kernelben, az esetben egy külső segédprogram segítségével a kernel megpróbálja betölteni a szükséges meghajtó modult, és ha sikerrel járt, akkor így az eszközfájl megnyitása is sikeres lesz. Ily módon megoldható az automatikus meghajtó-betöltés. Néhány évvel ezelőtt még tipikus gyakorlat volt, hogy ilyen módon például a hangkártya meghajtója akkor töltődött be a háttérben, amikor először megpróbáltunk zenét lejátszani az operációs rendszer indítása után.
A fenti rendszer nehézkes karbantarthatósága miatt pár éve a ***devfs*** volt a divat, ezt használták az ***UHU-Linux*** korábbi kiadásai is. Itt a kernel magára vállalta a ***/dev*** alatti eszközfájlok menedzselését. A ***/dev*** csatolási pont alá egy virtuális, ***devfs*** típusú fájlrendszert csatoltunk, mely alatt csak azok az eszközök voltak láthatók, melyek meghajtója be volt töltve. Ugyanakkor lehetőség volt nem létező fájl megnyitására is, a kernel ilyenkor (név alapján beazonosítva az eszközt) megpróbálta betölteni a modult. Utóbbi szolgáltatásra nem volt túl nagy igény, mivel szinte természetessé vált az elmúlt pár évben, hogy a hardverkezelő modulokat nem igény esetén, hanem már a rendszer indulásakor betöltik a disztribúciók.
A ***devfs*** megközelítésnek is voltak problémái, túl sok minden volt a kernelbe beégetve, olyan dolgok is, melyek felhasználói térben megoldhatók (és az ilyeneket nem szeretik a kernelben látni a fejlesztők), és ennek megfelelően a rendszer nem volt kellőképpen rugalmas. Végezetül, az ***udev*** alternatíva megjelenése kapcsán a fejlesztők a ***devfs*** rendszert elavulttá nyilvánították.
Az ***UHU-Linux 1.2***-ben mi is átálltunk a legújabb megközelítésre, az ***udev***-re. Itt a ***/dev*** könyvtár tartalmát egy felhasználói program (az ***udev démon***, ***udevd***) menedzseli, a kerneltől kapott úgynevezett ***hotplug üzenetek*** segítségével (a kernel, valahányszor hardver eszközzel kapcsolatos esemény történik, elindítja az ***/sbin/hotplug*** szkriptet, amely az ***udev*** rendszert is értesíti), valamint az újonnan megjelent ***/sys*** könyvtár tartalmára is erősen támaszkodik az ***udev***.
A ***/dev*** könyvtár lehetne egy közönséges könyvtár is (merevlemezen tárolt fájlrendszer részeként), de a javasolt megoldást követve ez egy virtuális, memóriában elhelyezkedő ***ramfs*** típusú fájlrendszer az ***UHU***-ban.
Az eredeti kernel maga tehát, azon túl, hogy a ***/sys*** fájlrendszer tartalmával és az ***/sbin/hotplug*** szkript indításával tájékoztatja az érdeklődő rendszerprogramokat a hardverek helyzetéről, semmit nem tesz a ***/dev*** alatti bejegyzések létrehozásával, jogosultságainak beállításával kapcsolatban, ezt

teljes egészében külső programra bízza. Az ***UHU*** kernelében ezen icipicit módosítottunk, a rendszer indítási folyamatát lényegesen leegyszerűsítendő a kernel végzi a */dev* könyvtár alá a ***ramfs*** típusú üres ***RAM***-ban található fájlrendszer csatolását, és létrehozza alatta a néhány legfontosabb bejegyzést, melyekre már az első folyamatként induló ***init*** program is számít.

A */dev* könyvtár alatt tehát memóriabeli fájlrendszer található, így ha itt bármit kézzel változtatunk, az a rendszer újraindítása során elvész. Viszont szükségünk lehet arra, hogy a saját ízlésünk szerint állítsuk be az egyes eszközfájlok tulajdonosát, csoportját, vagy hozzáférési jogosultságát. Erre az ***udev*** konfigurációs fájljait, elsősorban az */etc/udev/permissions.d* könyvtárat használhatjuk. Átírhatjuk az itt lévő ***50-udev.permissions*** fájlt is, de ajánlott ezt a fájlt inkább változatlanul hagyni, és más néven létrehozni egy új fájlt és abba írni saját igényeinket. Fontos a ***.permissions*** kiterjesztés megtartása, és értelemszerűen a fájlnév elején lévő sorszám szerinti sorrendben bírálják egymást felül a fájlok, így saját bejegyzéseink számára érdemes lehet például a ***90-sajatbeallitasaim.permissions*** fájlnevet választani.

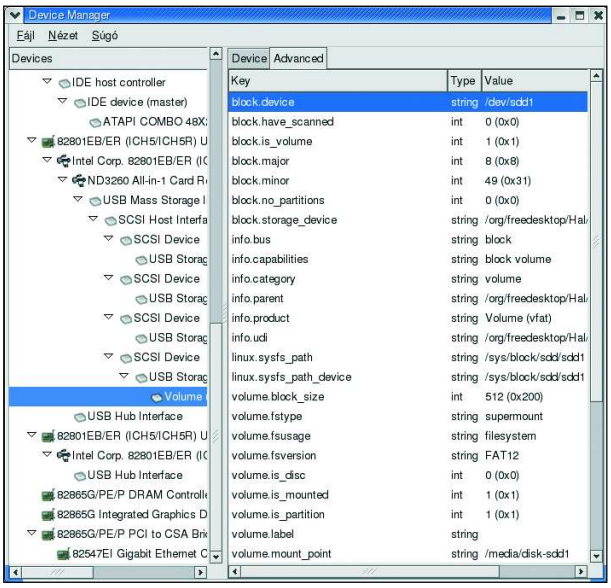
Az ***udev*** rendszernek előnye tehát, hogy a kernel részéről nem igényel támogatást, mindössze azt a hardverdetektálási infrastruktúrát, amely egyébként is rendelkezésre áll a kernelben, nemcsak az ***udev*** kedvéért. A többi felhasználói térben van kivitelezve, dinamikusabban fejleszthető, könnyebben és rugalmasabban testre szabható (akár az eszközfájlok neve is megváltoztatható).

Ugyanakkor van két apró hátránya is az ***udev*** rendszernek. Az egyik, hogy mivel nem speciális típusú a fájlrendszer, a ***devfs*** megoldással ellentétben itt nincs lehetőség a nem létező fájlra irányuló megnyitási kísérletet elkapni és gyorsan intézkedni a megfelelő modul betöltéséről. (Bár, mint láttuk, ezt már a ***devfs*** esetén sem igazán használtuk.) A másik apró probléma speciális alkalmazások készítőit bosszanthatja, amennyiben nem hardvert kezelő modulról van szó (hanem például hálózati protokoll támogatásáról). A két régi rendszerben elég volt megnyitni egy eszközfájlt: ha az azt kezelő modul még nem volt betöltve, akkor betöltődött a háttérben, majd ezt követően sikeresen folytatódott a program futása. Az új rendszerben a programnak explicit módon kérnie kell a modul betöltését, majd ezt követően bizonytalan ideig várnia, amíg a vele aszinkron módon futó ***udev*** rendszer létrehozza az igényelt fájlt. Ezekkel a problémákkal azonban az legtöbb felhasználó aligha fog találkozni.

A hagyományos statikus ***dev***-hez képest tehát az evolúció során két lépésben szinte teljesen „kifordult" a rendszer. Régen az eszközfájl megnyitása idézte elő a modul betöltését. Most viszont az elérhető hardverekhez az azonosítójuk (többnyire ***PCI*** azonosító) alapján előre betöltjük a modulokat, és a modul betöltése hozza létre (az ***udev*** rendszeren keresztül) a */dev* alatt a megfelelő eszközfájlt.

Hal

Az új */sys* fájlrendszert használja ki a ***HAL (Hardware Abstraction Layer, hardver absztrakciós réteg)*** is, amely szintén az */sbin/hotplug* szkript segítségével értesül az eseményekről, melyeket a központi ***hald*** nevű démon folyamat



dolgoz fel és továbbít (a ***D-BUS*** üzenetküldő rendszer segítségével) az arra kíváncsi felhasználói alkalmazások felé.

A ***HAL*** célja a rendszerben használt hardver elemekről, azok javasolt és tényleges felhasználási módjáról egy minél részletesebb valósidejű adatbázist karbantartani. Tehát éppúgy tárolja a hardverek fizikai jellemzőit, mint például olyan információkat, hogy adott partíción milyen fájlrendszer található, milyen módon ajánlott azt csatolni (az ajánlásokat további szkriptek dolgozhatják fel), hova csatoltuk valójában a fájlrendszert, és így tovább.

Egy lehetséges ilyen alkalmazás a ***HAL eszközkezelő (hal-device-manager)*** program, amelynek kifejezetten az a célja, hogy a ***HAL*** által nyilvántartott adatokat grafikusan megtekinthessük. Elsősorban tehát hibakereséshez vagy fejlesztéshez használható a program, de mindenképp érdemes vetni rá egy pillantást felhasználóként is. Az igazán izgalmas adatok a jobb oldali ***Advanced*** fülre kattintva jelennek meg. A háttérben munkálkodó üzenetküldő rendszer létezését mi sem demonstrálja jobban, mint hogy változás (például ***USB*** tároló csatlakoztatása) alkalmával a ***hal-device-manager*** által mutatott lista is azonnal megváltozik.

A ***GNOME*** grafikus környezet is támaszkodik a ***HAL*** démontól érkező eseményekre. Ilyen módon válik lehetővé, hogy új eszköz csatlakoztatásakor automatikusan megjelenjen számára egy ikon az asztalon, megnyíljon egy ***Nautilus*** ablak a fájlrendszer tartalmával, vagy éppenséggel automatikusan elinduljon a film lejátszása. Természetesen az automatikus programindítás beállítható, akár le is tiltható a ***Gnome Vezérlőpultban***, a ***Cserélhető adathordozók*** pont alatt.

Az ***UHU-Linux*** következő verziójában nemcsak a ***GNOME***, hanem a ***KDE*** grafikus felület kedvelői is élvezhetik majd a ***HAL*** által nyújtott lehetőségeket.

Végül, de nem utolsó sorban az ***uhu-automount*** rendszert is, mely az eszközök tartalmát automatikusan csatolja a */media* (korábban */mnt*) könyvtár alá, átültettük a ***HAL*** használatára, így az automatikus csatolást végző rendszer az alatta lévő infrastruktúrának köszönhetően egyetlen rövidke héjprogrammá (*/etc/hal/device.d/automount.hal*) redukálódott.

/media

Szóba került már, hogy az automatikus eszközöket a korábbi */mnt* helyett immár a */media* könyvtár alá csatoljuk. Ezt a váltást a ***Filesystem Hierarchy Standard*** ajánlását és több vezető disztribúciót követve léptük meg. Az */mnt* könyvtárat az ***UHU-Linux 1.2*** semmire nem használja, ideiglenes csatolások számára ideális csatolási pont.
A */media* könyvtár tartalma szintén egy memóriában létező (*ramfs* típusú) fájlrendszer. Ez azt jelenti, hogy ha itt bármit módosítunk kézzel (például új könyvtárat hozunk létre csatolási pont gyanánt), az a rendszer újraindítása után már nem lesz meg. Éppen ezért célszerű a */media* könyvtárat meghagyni teljes egészében az ***UHU automount*** rendszere számára, és ha másvalamit is szeretnénk csatolni (például hálózati kötetet), akkor számára például az */mnt* könyvtárat, vagy annak egy általunk létrehozott alkönyvtárát választani.
A */media* könyvtárral kapcsolatban megjegyzendő még, hogy tartalmához csak a *media* csoport tagjai férhetnek hozzá. A *media* csoport a korábbi *cdrom*, *cdwriter*, *floppy* csoportokat váltotta fel, mivel manapság annyiféle csatlakoztatható eszköz létezik, hogy értelmetlen és lehetetlen volna mindegyik számára külön csoportot létrehozni, így inkább összevontuk mindet eggyé.
A */etc/group* fájlban, vagy az ***UHU Vezérlőpultban*** hiába nézzük, azt fogjuk látni, hogy senki sem tagja a csoportnak. Ez azért van így, mert ebbe a csoportba nem fix felhasználók tartoznak bele, hanem dinamikusan (a *PAM* rendszer segítségével) azok kerülnek bele, akik helyileg jelentkeznek be a rendszerbe (használjuk az `id` parancsot a tagság ellenőrzésére). Így még ha hozzáférési jogot is adunk valakinek távoli belépésre a gépünkre, biztosak lehetünk benne, hogy a floppylemezen, CD-n, pendrive-on tárolt adatainkhoz nem fér hozzá, hacsaknem azt külön elérhetővé tettük számára. Ha ezt szeretnénk, betehetjük állandóra a *media* csoportba az illetőt, vagy a */media* könyvtár hozzáférési módját is átállíthatjuk megfelelőre (ám ez utóbbi változtatás a rendszer újraindítása során elvész, tehát gondoskodnunk kell róla, hogy bootolás során is végrehajtódjon a megfelelő `chmod` parancs).

CD-írás

A CD-írás tekintetében is hozott újdonságokat a 2.6-os kernel, és a disztribúció egyéb változtatásai. Ezek az újdonságok grafikus CD-író programok (például ***K3b***) használóinak aligha tűnnek fel, viszont a parancssor kedvelőinek annál inkább.
Lehetőség vált az *ATA* eszközökre történő közvetlen CD-írásra, így a korábbi *SCSI* emulációra immár nincsen szükség. A `cdrecord` parancsnak eddig szüksége volt egy olyasmi paraméterre, mint például `dev=1,0,0`. A lehetséges értékeket (SCSI eszközazonosítót) a root-ként futtatott `cdrecord -scanbus` parancs írja ki.
Az új *UHU*-ban *ATA* CD-író esetén a `dev=` opciónak értékül a `cdrecord -scanbus` kimenetéből kilesett értéket egy `ATA:` előtaggal (prefix) kell ellátni, például `dev=ATA:1,0,0`. Szerencsére van egyszerűbb lehetőség is, adhatjuk közvetlenül a */dev* fájlrendszer alatti eszköznevet (például `dev=/dev/hdc`), és mivel a */dev/cdwriter* szimbolikus linket beállítjuk a CD-íróra (több író esetén az egyikre) és a `cdrecord`-nak is megtanítottuk, hogy ez az eszköz az alapértelmezett, így az esetek túlnyomó részében egyáltalán nincsen szükség a `dev=` opció megadására.

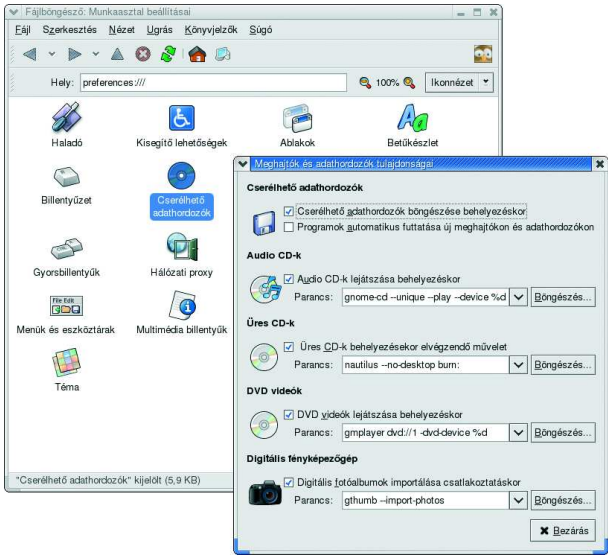
Kernelmodulok

Még egy rövidke fejezet erejéig maradjunk a 2.6-os kernelnél. A modulok betöltése terén is sok technikai részletet átszerveztek a kernelfejlesztők. A modulok betöltését a korábbi *modutils* helyett az újabb, *module-init-tools* nevű csomag programjai (`modprobe`, `insmod` stb.) végzik, melyek működése lényegében megegyezik elődjeik működésével. Különbség, hogy a `modprobe` konfigurációs fájlja, melyben a modulok alapértelmezett paramétereit adhatjuk meg, a korábbi */etc/modules.conf* helyett immár */etc/modprobe.conf* névre hallgat.
A modulok automatikus betöltését is kissé átdolgoztuk. A korábbi */etc/modules/AUTOLOAD* fájlt átkereszteltük */etc/modules.load*-ra, az ebben felsorolt modulokat a rendszer mindenképp betölti az indulás folyamán. Megjelent továbbá az */etc/modules.skip* fájl is, melyben soronként egy modul nevét adhatjuk meg, ezeket a modulokat semmiképp nem fogja betölteni a rendszer, akkor sem, ha a hardverdetektálás alapján szükségét érezné.
A 2.6-os kernel moduljainak nevében az aláhúzás és a kötőjel azonos szerepű karakterek, így nem szabad meglepődnünk, ha például az *snd-pcm* modul betöltése után *snd_pcm* jelenik meg a betöltött modulok listájában.

PATH környezeti változó

A *Linux* disztribúciók egyik kritikus gyenge pontja a környezeti változók beállítása a felhasználó számára. Különféle programok helyes működése számára fontos lehet egy-egy környezeti változó helyes beállítása, nem ritka, hogy egy alkalmazás több tucat ilyen változót is felismerjen és azok alapján másképp viselkedjen. Az egyik legfontosabb mind közül a *PATH* nevű, mely a programok keresési útvonalát tartalmazza, ennek segítségével válik lehetővé, hogy egy alkalmazás a teljes útvonal ismerete nélkül is indítható legyen, például elegendő legyen a `mozilla` programot indítanunk a `/usr/bin/mozilla` helyett, a rendszer megkeresse és megtalálja azt.
A környezeti változók folyamatonként külön léteznek, alap értelmezésben öröklődnek (a gyermekfolyamat megkapja az őt indító szülő környezeti változóit), de a szülő másmilyen változókkal is indíthatja a gyermek folyamatot.
A disztribúciók nagy része a környezeti változók beállítását a bejelentkező felhasználó nevében elsőként futó programra bízza (a `PATH` beállítása az ***UHU-Linux 1.1***-ben is még így történt). Ez a program szöveges bejelentkezés esetén az úgynevezett parancsértelmező (shell, héj), grafikus belépés esetén általában egy héjprogram (a parancsértelmező nyelvén megírt utasítássorozat), de lehet közvetlenül az ablakkezelő is. Idáig még nem reménytelen megoldani a problémát, viszont nehezít, hogy a felhasználó többféle parancsértelmező közül is választhat, mindegyiknek másképpen kell megadni a beállítandó környezeti változókat, így mindenképpen

a rendszerben több helyen is be kell állítani megfelelően ezeket a változókat, ami semmiképp sem szerencsés.
Az igazi probléma azonban ott kezdődik, hogy be lehet lépni úgy is a rendszerre, hogy ezen programok egyike se hajtódjon végre, vagy ha maga a héj végre is hajtódik, ne olvasson ki semmiféle inicializáló fájlt. Erre a legtipikusabb példa az, amikor `ssh`-val távolról úgy lépünk be, hogy rögtön egy végrehajtandó parancsot is megadunk, tehát nem parancsértelmezőt kérünk. Ilyenkor az általunk megvizsgált disztribúciókban mind-mind hibás volt a PATH értéke.
A PATH-t azért hangsúlyozom, mivel azt általában nehezebb a többi változónál beállítani. Míg a legtöbb környezeti változó számára a rendszer összeállítója (disztribútor, rendszergazda) fix értéket óhajt beállítani (vagyis az érték független a rendszer tulajdonságaitól, a felhasználótól), addig a PATH esetén ez nem így van, egy jó PATH elemei függenek a feltelepített szoftverektől (újabb csomag behozhat újabb keresési könyvtárat), valamint függenek a felhasználótól is, hiszen igencsak illik a felhasználó egyik könyvtárát (általában a ***~/bin*** könyvtárat) is felvenni, sőt, a root-nak az adminisztrátori teendők ellátására szolgáló parancsok könyvtárait (***/sbin***, ***/usr/sbin***, ***/usr/local/sbin***) is meg kell kapnia.
Ahhoz tehát, hogy a PATH-t beállítsuk, le kell futtatni valami e célra elkészített kódot, amelyik összerakja, hogy minek is kell az értéknek lennie.
A konstans értékű környezeti változók esetén több disztribúció is választotta már azt a megoldást, hogy a parancsértelmezők inicializáló fájljai és a grafikus bejelentkezéskor lefutó szkript helyett egy lépéssel korábbra, a ***PAM*** nevezetű hitelesítési rendszerbe helyezi azok beállítását. Erre a hivatalos ***PAM*** szoftvercsomagban is található modul, mely ***pam_env.so*** névre hallgat. Ekkor tehát maga a belépést engedélyező program állítja be a környezeti változókat (amennyiben a felhasználó azonosítása sikeres volt), és a felhasználó legelső folyamatát, vagyis a parancsértelmezőt vagy a grafikus környezetet már eleve a kívánt változókkal indítja. Ebben a megközelítésben megszűnik tehát a kódtöbbszörözés, és megszűnik a nem interaktív távolról parancsfuttatás problematikája is, hiszen ilyenkor is helyesen lesznek a környezeti változók beállítva. A PATH kivételével már az ***UHU-Linux 1.1*** is ezt a megoldást követte.
Több probléma adódott azonban az ***UHU***-ban abból, hogy a PATH változónk nem volt mindig helyesen beállítva.
A beállítást a parancsértelmező végezte, így grafikus környezetbe belépve a terminálokban indított parancsértelmezőket leszámítva a programok PATH értéke hibás volt (ezt még csak-csak meg lehetett volna oldani), illetve nem interaktív parancsfuttatás esetén sem a végleges PATH volt beállítva. Utóbbi problémát már csak a PATH beállításának koncepcionális átdolgozásával volt esély orvosolni, nyilvánvaló volt, hogy a ezt is a ***PAM*** hitelesítési rendszerbe kell átemelnünk, a többi változó mellé, és megoldani a PATH értékének kitalálását, lehetőség szerint plusz csomagok által könnyen bővíthető módon.
A megoldás megtervezése során végül is két részre szedtük a feladatot. Egyik komponensként elkészült a ***pam_envfeed.so*** nevezetű modul. Ez a modul annyit



csinál, hogy elindít egy külső programot (alap értelmezésben az ***/sbin/pam_envfeed***-et), figyeli ennek a kimenetét, és a kimenetben talált minden változó=érték párt beállít környezeti változóként. Ez tehát egy teljesen általános, bármely környezeti változó beállítására roppant dinamikusan használható modul.
Másik lépésként pedig megírtuk azt a ***pam_envfeed*** szkriptet, mely az ***/etc/envfeed.d*** könyvtár alatti, ***.sh*** kiterjesztésű szkripteket hajtja végig sorra, és minden ezek által ENVFEED_ kezdetű névvel beállított környezeti változóra kiírja azok nevét az ENVFEED_ előtag nélkül, valamint az értékét. Ily módon több szkript összjátékának eredményeképpen kitaláljuk a leendő PATH értékét az ENVFEED_PATH változóban, miközben nyugodtan használhatunk a szkriptekben sok egyéb változót (köztük a PATH-t is teljesen másmilyen értékkel), és végső soron ezektől függetlenül mondjuk meg, hogy mi legyen a leendő PATH.
Az átállásnak van egy érdekes mellékhatása is. Amennyiben a `su` paranccsal váltunk át rendszergazdává, az esetben a ***pam_env.so*** és ***pam_envfeed.so*** modulok nem hajtódnak végre (lásd az ***/etc/pam.d/su*** fájlt). A régebbi ***UHU***-kban a környezeti változók ilyenkor egy kivétellel változatlanok maradtak, az egy kivétel természetesen a PATH volt, melyet a rendszergazdaként indított héj beállított magának. Persze itt sem volt tökéletes a helyzet: ha a su-nak megadtunk indítandó parancsot, akkor itt sem állítódott be a PATH. Az új ***UHU***-ban azonban már egyáltalán nem állítódik be a PATH (hiszen a shell inicializáló fájljából kikerült az a kód, amit a ***PAM*** rendszerbe ültettünk át), vagyis olyan parancsértelmezőt kapunk, amely ténylegesen szigorúan véve kizárólag rendszergazdai jogosultságot nyújt nekünk, de nem rendszergazdai környezetet. Hiába gépeljük be az `ifconfig`, a `chroot` vagy bármely hasonló parancsot, azokat nem találja meg a parancsértelmezőnk, mivel nincsen benne a PATH-ban.
A fenti probléma kiküszöbölésére használható a `su -` parancs, mely rendeltetésének megfelelően már nemcsak rendszergazdai jogosultságot, hanem rendszergazdai környezetet is biztosít, beleértve többek között a PATH ennek szellemében történő beállítását is, hiszen végrehajtja

a *pam_env.so* és *pam_envfeed.so* modulokat is (lásd az */etc/pam.d/su-* és az általa hivatkozott *system-auth-rootok* és *system-auth* fájlokat).

Inotify, Gamin

A hagyományos *Unix* rendszerhívások között nem szerepel olyan, amellyel egy fájl megváltozását figyelhetnénk. A régi nagy *Unix* rendszerekben erre a szolgáltatásra úgy látszik hogy nem igazán volt szükség. A *Linux* asztali rendszereken történő elterjedése során jelent meg az igény erre elsősorban a grafikus fájlkezelők részéről, hiszen ezek szeretnék az ablakuk tartalmát megfelelően módosítani, mihelyst valamely általuk megjelenített fájl vagy könyvtár bármilyen tulajdonsága megváltozik (például új fájlt hozunk létre). Nyilvánvaló, hogy a *fájlstruktúra folyamatos lekérdezése (polling)* nem igazán működőképes megoldás, hiszen hatalmas az erőforrásigénye. Egy olyan rendszerre van szükség, amelyben a kerneltől kapunk értesítést, amikor minket érintő változás történik. Jelenleg mind a *Gnome*, mind a *KDE* grafikus felület támaszkodik erre a szolgáltatásra.
Az *UHU-Linux 1.2* mind a kernelben található támogatás, mind az erre épülő függvénytár terén újítással szolgál.
A kernelben a korábbi *dnotify* rendszer mellett megjelent az *inotify*. (Ez egyelőre nem része a hivatalos kernelnek, az *UHU* kernelbe külön foltként került bele.)
A *dnotify* rendszerben (a „d" betű a directory-ra, vagyis könyvtárra utal) könyvtárat van lehetőségünk figyelni, fájlt ezáltal csak közvetve (az őt tartalmazó könyvtárra értesülünk eseményről, majd innen kezdve az összes fájlt végig kell nézni annak eldöntéséhez, hogy melyik változott). Minden egyes figyelt könyvtár egy újabb nyitva tartott fájlleírót igényel, így nem kizárt, hogy egy folyamat beleütközzön az egyszerre megnyitható fájlok korlátjába. A megnyitás megfogja az adott könyvtárat, így azt például, ha az egy csatolási pont, nem lehetséges lecsatolni. Ezen felül a *dnotify* a felhasználói folyamattal szignálokon keresztül kommunikál, ami erőforrásigényes és nehézkesen kezelhető.
Az új, *inotify* nevű rendszer mindezeket a hibákat hivatott kiküszöbölni. Az „i" betű az i-node-ra utal, hiszen tetszőleges *i-node (fájlrendszerbejegyzés)* figyelhető, fájl és könyvtár egyaránt. A kommunikáció egy speciális eszközön (*/dev/inotify*) keresztül történik, a processz itt közli a megfigyelendő objektumok listáját, és itt értesül a változásokról is. Ezáltal az eseményre várakozás lényegesen könnyebben és letisztultabb módon programozható le, valamint lehetőség nyílik például arra is, hogy értesüljünk, ha a megfigyelt objektumot tartalmazó fájlrendszer lecsatolódik.
Az alkalmazások általában nem közvetlenül a *dnotify* vagy *inotify* interfészt használják, hanem egy köztes réteget, egy függvénytárat. Korábban a *FAM* nevű csomag szolgált erre, mely a háttérben a *dnotify*-ra támaszkodott. A *FAM*-mal is több probléma volt, többek között nem képes kihasználni az *inotify* nyújtotta előnyöket, valamint globális démonfolyamat futását igényli, ami megközelítés mind stabilitás, mind pedig biztonság terén igencsak megkérdőjelezhető döntés.

A *FAM* rendszert szintén lecseréltük modernebb utódjára, a *Gamin*-ra, amely az alkalmazás felől nézve visszafelé kompatibilis a *FAM*-mal, így a csere nem igényelt változtatást a *FAM*-ot használó alkalmazások (*Gnome*, *KDE* stb.) részéről. A *Gamin* alapértelmezésben (amennyiben rendelkezésre áll) az *inotify* rendszert használja, de futási időben képes visszaállni *dnotify*-ra, ha az *inotify* valami miatt nem elérhető. Továbbá ha egy megfigyelt fájl túl gyakran változik, akkor arra a fájlra automatikusan átáll lekérdezéses módba. A *Gamin* teljes egészében a felhasználó jogosultságával fut, nem tartalmaz központi, rendszergazdaként futó démont. Ugyanakkor technikai okok miatt indít egy külön folyamatot *gam_server* néven, amely az adott felhasználó összes folyamata számára szolgáltatja a megfelelő információkat. Így a rendszerben mindig felhasználónként mindössze egy *gam_server* folyamatot látunk, amelyet a legelső, ilyen szolgáltatást igénybe vevő program a háttérben elindít, a többi program már ehhez csatlakozik, és a *gam_server* akkor lép ki, amikor a felhasználónak immár egyetlen folyamata sem figyel meg egy fájlt sem. A külön folyamatok ily módon történő összefogása szintén erőforrás megtakarításához vezet.

A jövő

Nehéz kérdés most előre megjósolni, hogy milyen lesz az *UHU-Linux* következő verziója. Természetesen bőven vannak terveink, több kiadásra is elegendően.
Szinte biztos, hogy a következő kiadás 2.0 sorszámot fogja viselni, és főként inkább felhasználói szemmel nézve, a grafikus felületen fog újdonságokat hozni. Könnyen elképzelhető egy vadonatúj grafikai stílus megjelenése is.
Ami már egészen biztos, az az, hogy át fogunk állni *UTF-8* karakterkódolásra (ez a Unicode egyik ábrázolási formája), lényegében teljesen magunk mögött hagyva az idejétmúlt *Latin-2 (ISO-8859-2)* kódolást. Az átállást számtalan olyan ékezetkezelési probléma indokolja, melyek a régi rendszerben nem oldhatók meg. A mostani *UHU* a legtöbb helyen még a régi *Latin-2* karakterkészletet használja, de pár helyen már a modernebb *UTF-8* kódolást. Különösen a fájlnevek terén nagy a kavarodás, mivel bizonyos alkalmazások az egyik, míg mások a másik ábrázolást részesítik előnyben, így sok esetben az egyik programmal létrehozott ékezetes fájlneveket a másik program nem látja helyesen, és viszont. Az ilyen és ehhez hasonló problémákra fog végleges és megnyugtató megoldással szolgálni az átállás, onnan kezdve biztonsággal lehet majd gyakorlatilag bárhol használni az ékezetes betűket.
Egyéb elképzeléseinkről egyelőre nem írnék, ez maradjon meglepetés. A fejlesztés menete iránt érdeklődőket szívesen látjuk a *dev@uhulinux.hu* levelezési listán (feliratkozni a *lists.uhulinux.hu* címen lehet), ahol további fejlesztési kérdések megvitatásával is találkozhatnak, valamint szívesen fogadjuk az ötleteket. Ugyanakkor kérjük, hogy ezt a listát ne használjátok olyan felhasználói kérdések feltevésére, melyek nem az *UHU-Linux* fejlesztésével kapcsolatosak, az ilyen kérdések megvitatására üzemeltetjük a *kezdo@uhulinux.hu* és *halado@uhulinux.hu* listát, ahol szintén mindenkit szívesen látunk.

Koblinger Egmont
Informatikus, az UHU-Linux fejlesztője