Az Udev / D-Bus rendszer - a modern asztali Linuxok alapja

A D-Bus rendszer minden modern Linux disztribúcióban jelen van, sőt mára már a Linux, és más UNIX jellegű, sőt nem UNIX rendszerek (különösen a desktopon futó változatok) egyik legalapvetőbb technológiája, és az ismerete a rendszergazdák számára lehetővé tesz néhány rendkívül hasznos trükköt, az alkalmazásfejlesztőknek pedig egyszerűen KÖTELEZŐ ismerniük. Miért ilyen fontos a D-Bus? Mit csinál? D-Bus alapú technológiát teszik lehetővé többek között azt, hogy közönséges felhasználóként a kedvenc asztali környezetünkbe bejelentkezve olyan feladatokat hajtsunk végre, amiket a kernel csak a root felasználónak engedne meg. Felmountolunk egy USB meghajtót? NetworkManagerrel konfiguráljuk a WiFi-t, a 3G internetet vagy bármilyen más hálózati csatolót, és kapcsolódunk egy hálózathoz? Figyelmeztetést kapunk a rendszertől, hogy új szoftverfrissítések érkeztek, majd telepítjük ezeket? Hibernáljuk, felfüggesztjük a gépet? A legtöbb esetben ma már D-Bus alapú technológiát használunk ilyen esetben. A D-Bus lehetővé teszi, hogy egymástól függetlenül, jellemzően más UID alatt indított szoftverösszetevők szabványos és biztonságos módon igénybe vegyék egymás szolgáltatásait. Ha valaha lesz a Linuxhoz professzionális desktop tűzfal vagy vírusirtó megoldás, a dolgok jelenlegi állasa szerint annak is D-Bus technológiát kell használnia. A D-Bus technológia legfontosabb ihletője a KDE DCOP rendszere volt, és mára a D-Bus leváltotta a DCOP-ot, csakúgy, mint a Gnome Bonobo technológiáját.

1. D-Bus

A D-Bus FAQ szerint D-Bus egy interprocessz-kommunikációs protokoll, és annak referenciamegvalósítása. Ezen referenciamegvalósítás egyik összetevője, a libdbus könyvtár a D-Bus szabványnak megfelelő kommunikáció megvalósítását segíti. Egy másik összetevő, a dbus-daemon a D-Bus üzenetek routolásáért, szórásáért felelős. A dbus-daemon nélkül is folytathat egymással két partner D-Bus kommunikációt, de nem jellemző ez az elrendezés. A D-Bus protokoll gyakorlatilag komponens alapú szolgáltatások megvalósítására és azok felhasználására szolgál. A D-Bus kommunikáció bármely résztvevője lehet egyszerre "szerver" (aki a szoftverkomponenseket/objektumokat rendelkezésre bocsátja) és "kliens" (aki a rendelkezésre bocsátott szolgáltatásokat igénybe veszi) is, de nem igazán jellemző mindkét szerep ugyanazon ügyfél általi egyidejű betöltése.

A D-Bus rendszer szerverként funkcionáló ügyfelei ún. objektumokat bocsátanak a kliens szerepet betöltő partnerek részére. Ezen objektumok viselkedése nagyban hasonlít az objektumorientált programozás tárgyköréből megismert objektumokéra. Bár az objektumok működése C++ fogalmakkal is jól leírható, de rendelkeznek néhány olyan tulajdonsággal, ami inkább a modernebb objektumorientált nyelvekhez (pl. C#) teszi őket hasonlatossá: Az objektumok futásidőben felkutathatóak, szerkezetük lekérdezhető (ez az ún. introspection), stb. Az a tény azonban, hogy az objektumokat nem az őket felhasználó kliens programok hozzák létre, illetve törlik (ld. constructor/destructor), hanem kizárólag a szerver típusú ügyfelek, sőt, nem is a kliensek állapotterében (memóriájában) léteznek, hanem a szerverekében, a már említett komponens alapú rendszerekkel rokonítja őket, nem pedig valamely konkrét programozási nyelvvel. Az D-Bus rendszer által biztosított – az objektumok elérésére szolgáló – stabil ABI és a többféle programnyelvhez (sőt, parancssori eszközökön keresztül még shell szkriptekhez is) rendelkezésre álló API tovább növeli ezt a hasonlóságot.

A D-Bus szerver jellegű ügyfelei fájlrendszer szerű hierarchiában ajánlják ki az objektumaikat, amit az objektumok neve (a D-Bus nevezéktanában path vagy object path) is tükröz: Az objektumnevek az abszolút UNIX elérési útvonalakhoz hasonlóan a "/" karakterrel kezdődnek, és az objektumhierarchia szintjeit szintén a "/" karakter határolja. Példaként említeném meg, hogy a D-Bus démonhoz csatlakozó programok láthatják, hogy maga a dbus-daemon is megjelenik objektumszolgáltatóként, és a D-Bus kommunikáció többi részvevője számára rendelkezésre bocsát többek között egy "`/org/freedesktop/DBus`" nevű objektumot.

Az alacsony szinten működő libdbus könyvtár csak a D-Bus kommunikációs protokolljának megvalósítására szolgál, az objektumok magasabb szintű kezelésével nem foglalkozik, ezért a D-Bus rendszer szolgáltatásait igénybe vevő alkalmazások más, magasabb szintű könyvtárakat, illetve objektumokat használnak, mint pl. a GObject vagy a QObject. Ezek a magas szintű natív nyelvi objektumok tipikus esetben proxiként viselkedve teszik elérhetővé a D-Bus objektumokat, teljesen elrejtve a kommunikáció részleteit. A D-Bus tehát kétarcú rendszer: a saját fejlesztői, a disztibútorok és rendszergazdák irányában leginkább interprocessz-kommunkációs eszközként mutatja magát, a rá épülő szoftverek fejlesztő viszont komponens alapú szoftverfejlesztői eszközként gondolhatnak rá.

Minden D-Bus objektum tetszőleges számú osztályba tartozhat. A D-Bus nevezéktanában az osztályok neve interface. Azt, hogy egy objektum egy adott osztály tagja, D-Bus nyelven úgy mondjuk, hogy az objektum szolgáltatást biztosít az adott interfészen keresztül. Az interfészek révén az objektumok tagfüggvényekkel (a D-Bus nevezéktanában method, azaz metódus) és adattagokkal (property vagy attribute) is rendelkeznek. Az adattagok elérésére a D-Bus ABI nem biztosít közvetlen hozzáférést, de létezik egy szabványos interfész, amelynek tagfüggvényei lehetővé teszik, hogy lekérdezzük, illetve módosítsuk az adott objektum összes többi interfészének adattagjait.

A D-Bus interfésznevek fordított DNS domainnevekhez hasonló szerkezetűek. Nem kezdődhetnek "." karakterrel, de legalább egy "." karakternek lennie kell bennük. A fenti példában említett "`/org/freedesktop/DBus`" objektum pl. a következő interfészeket valósítja meg (C++ elnevezéssel élve a következő osztályokba tartozik): "`org.freedesktop.DBus`" és "`org.freedesktop.DBus.Introspectable`". Az két interfésznév hierarchikus kapcsolat vagy öröklés benyomását kelti, holott technikailag egymástól teljesen függetlenek. A D-Bus szabványban semmilyen kapcsolat nincs az interfészek (osztályok) között, legfeljebb a magasabb szintű függvénykönyvtárakban jelenik meg ilyesmi.

A D-Bus objektumok interfészei tagokból (member) állnak. A tagok lehetnek metódusok, szignálok vagy adattagok (property). A metódusok és a szignálok argumentumokkal is rendelkeznek (ezek a metódusok esetében a függyvényparaméterek, illetve visszatérési értékek szerepét töltik be). A property-k, amint korábben már említettem nem kezelhetőek közvetlenül D-Bus üzeneteken keresztül, csak közvetetten, egy kifejezetten ezt a célt szolgáló interfész (`org.freedesktop.DBus.Properties`) metódusainak segítségével. Amennyiben egy objektum valamely osztályának property típusú tagja (azaz adattagja) is van, akkor ennek az objektumnak az "`org.freedesktop.DBus.Properties`" interfészt is biztosítania kell, ellenkező esetben hozzáférhetetlenek maradnak az adattagok. Az interfészek tagjainak nevei kizárólag a "[A-Z][a-z][0-9]_" karakterekből állhatnak, vagyis az objektumnevektől és az interfésznevektől eltérően nem tartalmazhatnak "/" vagy "." karaktereket, amelyek hierarchikus elnevezéseket tennének lehetővé. A member-nevek tehát egytagúak.

A többtagú interfész- és path-nevekkel ellentétben az interfészek tagjainak nevei egytagúak. Az interfész és member-nevek a D-Bus üzenetekben ugyan külön adatmezőben utaznak, de számos D-Bus eszköz (pl. később bemutatásra kerülő dbus-send parancssori eszköz) lehetővé teszi az összevonásukat. Ez azt jelenti, hogy pl. az "`org.freedesktop.DBus`" interfész "`ListNames`" metódusára az egyszerűség (és az objektumorientált programozásból eredő megszokás miatt)

"`org.freedesktop.DBus.ListNames`" néven hivatkozhatunk. Ez a `ListNames` metódus egyébként arra szolgál, hogy az adott dbus-daemon példány minden aktív ügyfelét kilistázza.

A D-Bus szabvány definiál néhány szabványos interfészt, amelyeket a legtöbb D-Bus kompatibilis alkalmazás meg is valósít, ezzel megkönnyítve a D-Buson keresztül elérhető objektumok és szolgáltatásaik feltérképezését, kezelését. A szabványos interfészek közé tartozik a feljebb már említett "`org.freedesktop.DBus.Properties`", amely az objektumok adattagjainak kezelésére szolgál. Nagy jelentősége van az "`org.freedesktop.DBus.Introspectable`" interfésznek, amely az "Introspect" metódusa string típusú visszatérési értékében XML formátumú leírást ad az adott objektum szerkezetéről, beleértve az objektum interfészeit, az interfészek metódusait, adattagjait (properties) és szignáljait, valamint az objektum gyermekeinek nevét, amelyek az objektumhierarchiában közvetlenül alatta állnak. Ezeknek (és a többi itt nem említett) szabványos interfésznek a részletes leírása megtalálható a D-Bus specifikációjában.

A dbus-daemon saját objektumai és azoknak interfészei (elsősorban a már említett "`org.freedesktop.DBus.ListNames`" metódusra gondolok itt), valamint a D-Bus ügyfelek saját objektumainak szabványos interfészei (feltéve, hogy tényleg szabványosan vannak megvalósítva!) lehetővé teszik többek között azt, hogy grafikus D-Bus-szolgáltatás feltérképező- és kezelő alkalmazások készüljenek, mint pl. a kdbus vagy a DBusExplorer.

Egy rendszeren egyidőben tetszőleges számú dbus-daemon példány futhat párhuzamosan. Bármilyen szoftvercsomagnak lehetősége van rá, hogy saját D-Bus példányt indítson testreszabott konfigurációval, hogy ezzel segítse a saját összetevői közötti kommunikációt, kizárva belőle az illetékteleneket. Általában két jól ismert D-Bus példány van jelen egy tipikus Linux rendszerben. A legfontosabb a rendszer "system" dbus-daemon. Ez teszi lehetővé a bevezetőben említett modern desktop technológiák működését, mint pl. a NetworkManager, vagyis ezt közvetít az asztali környezet és a rendszerszolgáltatások között. Események érkeznek rajta keresztül az asztali környezet számára a rendszer– és hardverváltozásokról, mint pl. az akkumulátor töltöttségének változása, a tápkábel kihúzása és bedugása, új WiFi SSID-k megjelenése, térerőváltozások, új USB meghajtó csatlakozása, stb., majd – ha a házirendek (ld. PolicyKit és ConsoleKit/systemd-logind a későbbiekben) lehetővé teszik - a rendszerszolgáltatások végrehajtják a felhasználótól a D-Bus-on keresztül érkező utasításokat. A rendszer D-Bus mellett a nehézsúlyú asztali környezetek (pl. Gnome, KDE) is indítanak egy-egy dbus-daemon példányt minden munkamenethez (ez az ún. "session" bus), amelyen keresztül az asztali környezet jól integrált összetevői kommunikálnak egymással.

2. A D-Buson keresztül elérhető fontosabb szolgáltatások

A rendszeren futó különböző D-Bus példányok (pl. session D-Bus) elsődleges feladata, hogy keretet biztosítson különböző szoftverkomponensek együttműködéséhez. A rendszer D-Busról emellett azonban elmondható, hogy – mivel privilegizált és felhasználói szoftverkomponensek között közvetít, ezáltal bizonyos SETUID programfájlok kiváltására és a legkevesebb privilégium elvének megközelítésére alkalmas – a modern UNIX rendszerek biztonsági modelljének fontos építőkövévé kezd válni. A rendszer D-Bushoz csatlakozó, és azon jól ismert (well known) nevek alatt szolgáltatásokat biztosító démonokat ebben a blogbejegyzésben szolgáltató ágensnek nevezem. A szolgáltató ágensek a felhasználó – általában rendszergazdai jog híján közvetlenül el nem végezhető – utasításait hajtják végre, valamint multicast szórással az általuk felügyelt alrendszerek – többnyire végső soron a kernelből érkező – eseményeit is továbbítják a felhasználói felület vagy más érdeklődő szoftverkomponens felé. Az alábbiakban vázlatpontszerűen bemutatok néhány fontosabb szolgáltató ágenst:

- NetworkManager: Az `org.freedesktop.NetworkManager` nevű jól ismert D-Bus címet birtokolja. Lehetővé teszi, hogy a közönséges felhasználók is igényeiknek megfelelően

konfigurálják a számítógép hálózati beállításait, és az így keletkezett konfigurációt el is tárolja. Lehetővé teszi továbbá a hálózat menedzselését is, ami rendszergazdai jogok nélkül egyébként nem lenne lehetséges. Az ilyen hálózati menedzselési feladatok közé tartozik többek között a vezeték nélküli (vagy vezetékes) hálózat ki- és bekapcsolása, kapcsolódás az elérhető közelségben levő WiFi SSID-khoz, VPN szolgáltatókhoz, stb. A NetworkManager felhasználói felületét a Gnome nm-applet programja, a KDE plasmoid NetworkManagementje, és hasonló szoftverek biztosítják, amelyek természetesen a kizárólag a rendszer D-Buson keresztül érintkeznek a NetworkManager démonnal.

- UPower: Az `org.freedesktop.UPower` nevű jól ismert D-Bus címet birtokolja. A számítógép áramellátást felügyeli. A tápegységekkel és egyéb energiagazdálkodással kapcsolatos eszközök (AC tápegység, akkumulátor, laptopfedél, stb.) eseményeit (pl. csatlakoztatás, leválasztás, töltöttségi szint változása, nyitás, csukás, stb.) továbbítja a kerneltől a felhasználói felületek felé, lehetővé teszi ezen eszközök adatainak (gyártó, típus, kapacitás, várható élettartam, elhasználtság, stb.) lekérdezését, valamint különböző energiagazdálkodási műveletek (alvó állapotba lépés, hibernálás, stb.) végrehajtását. Az UPower rendszer funkcióit korábban az `org.freedesktop.Hal` címen elérhető HAL démon látta el, ennek fejlesztése azonban abbamaradt, és átvette a helyét a DeviceKit alrendszer. A DeviceKit projekt azonban rövid életűnek bizonyult, és a helyét több – az Udev projekt égisze alatt futó – kisebb alrendszer vette át. Ezek egyike az UPower.
- udisks: Az UPowerhez hasonlóan a HAL projekt leszármazottja, és az Udev szárnyai alatt kötött ki. Az `org.freedesktop.UDisks` címen kínálja szolgáltatásait. Fő feladta a blokkeszközök kezelése, és azok eseményeinek közvetítése a felhasználó felé. Lehetővé teszi, hogy a felhasználói felületek érzékeljék új blokkeszközök csatlakoztatását, és különféle műveleteket végezzenek rajtuk, mint pl. mountolás (olyan módon, hogy az adott felhasználó megfelelő jogokkal rendelkezzen a felmountolt fájlrendszeren), formázás, partícionálás, titkosítás, LVM kezelés, swap-eszközök kezelése, sőt a közeljövőben loop-eszközök (pl. CD vagy DVD képmásfájlok) felmountolására is képes lesz.
- PackageKit: Az `org.freedesktop.PackageKit` nevű jól ismert D-Bus címet birtokolja. Privilegizált démonja a packagekitd különböző backendek (pl. apt, yum, zypp) segítségével a rendszer szoftvercsomagjainak karbantartását (pl. telepítés, eltávolítás, frissítések keresése, frissítés) végzi. A rendszer D-Buson keresztül hozzá csatlakozó frontendekkel (mint pl. a KDE-féle Apper) kiegészülve lehetővé teszi, hogy rendszergazdai jog nélkül kezeljük a szoftvercsomagokat és repository-kat.

2.1. A D-Bus biztonsági modelljének központi eleme: PolicyKit (és ConsoleKit)

Szó volt már róla, hogy a rendszer D-Bus és szolgáltatásai a UNIX-ok biztonsági modelljének részévé kezdenek válni. Ha azonban a szolgáltató ágensek a felhasználóktól érkező minden kérést feltétel nélkül végrehajtanának, azzal nem igazán mozdítanák előre a biztonság ügyét. Az előző részben már ismertetett dbus-daemon man oldal bemutatja a dbus-daemon XML alapú konfigurációs fájljait. A `<policy>` elem megvalósít egyfajta biztonsági modellt, ez azonban csak a D-Bus kommunikációra vonatkozik, azaz alacsony szinten dolgozik, a D-Bus üzeneteket szűri tűzfalszerűen. Akár UNIX felhasználó és csoport szinten adhatjuk meg benne, hogy ki milyen D-Bus üzeneteket küldhet és fogadhat, milyen D-Bus neveket/címeket birtokolhat, stb.

A D-Bus rendszer biztonsági modelljének egy magasabb szintje a PolicyKit démonra épül. A polkitd a rendszer D-Bus példányhoz kapcsolódva az `org.freedesktop.PolicyKit1` néven keresztül nyújt szolgáltatásokat D-Bus többi ügyfelének. A PolicyKit alapvetően a következőképpen működik:

Rendszerindításkor a polkitd nevű démon csatlakozik a rendszer D-Bushoz, és az `org.freedesktop.PolicyKit1` D-Bus néven, a `/org/freedesktop/PolicyKit1/Authority` objektum `org.freedesktop.PolicyKit1.Authority` interfészén keresztül válik elérhetővé. Ezen felül a felhasználói munkamenetekben (pl. Gnome, KDE session-ök) is van egy-egy processz, amely a polkitd-től eltérően természetesen nem root-ként, hanem az adott felhasználó nevében fut. Ezt az utóbbi processztípust autentikációs ágensnek hívják, nem igényel magának jól ismert nevet a D-Bus-on, és a `org.freedesktop.PolicyKit1.AuthenticationAgent` interfészen keresztül biztosít szolgáltatásokat, amelyeket a polkitd vesz igénybe. Annak érdekében, hogy a polkitd megtalálja a jól ismert névvel nem rendelkező autentikációs ágenseket, induláskor minden ágens beregisztrálja magát a polkitd-nél az `org.freedesktop.PolicyKit1.Authority.RegisterAuthenticationAgent` metódus segítségével.

Amint erről már szó volt, a rendszer D-Buson keresztül elérhető szolgáltató ágensek (pl. NetworkManager, UPower, udisks, stb.) többnyire root joggal futnak, és a felhasználó asztali környezetéből D-Buson keresztül érkező utasításokat hajtják végre. Az utasításokat hordozó üzeneteket maga a D-Bus szűri, a PolicyKit célja pedig az, hogy segítsen a szolgáltató ágenseknek eldönteni egy adott felhasználói kérésről, hogy jogos vagy sem. Ennek érdekében a szolgáltató ágenseknek a felhasználók számára felkínált szolgáltatásai (pl. bizonyos metódusok meghívása, változók értékének megváltoztatása, stb.) ún. akciókká képződnek le a PolicyKit rendszerben. A legtöbb szolgáltató ágenst tartalmazó programcsomag a PolicyKit számára értelmezhető konfigurációs fájlokat is tartalmaz, amelyben definiálja az adott ágenshez tartalmazó akciókat, az egyes akciókhoz tartozó felugró üzenetek szövegezését különféle nyelveken, a felugró üzenettel együtt megjelenítendő ikont, és az akció alapértelmezett elbírálására vonatkozó utasításokat (pl. aktív munkamenetek számára engedélyezett, inaktív munkamenetek számára tiltott, stb.). Az akciókat leíró fájlok a `/usr/share/polkit-1/actions` könyvtárban vannak, formátumukat a PolicyKit man oldala írja le.

Mind a szolgáltató ágensnek, mind a polkitd-nek mind magának a dbus-daemonnak szüksége van olyan információkra, amelyek alapján megállapíthatják, hogy egy befutó D-Bus kérés mögött melyik felhasználó melyik munkamenete (esetleg annak pontosan melyik processze áll), illetve hogy az adott munkamenet éppen aktív vagy inaktív. esetleg zárolt-e? A munkamenetekre, felhasználókra, konzolokra vonatkozó információ legnagyobb részét egy külön erre a célra létrehozott D-Bus szolgáltató ágens, a ConsoleKit (vagy újabban a systemd démon) biztosítja.

Mivel a ConsoleKit aktív fejlesztése befejeződött, lassan átveszi a helyét a systemd systemd-loginctl szolgáltatása. Ez a ConsoleKit-éhez nagyban hasonló szolgáltatásokat biztosít. A két rendszer között a legalapvetőbb eltérés az adott PID-hez tartozó session azonosításának módja. A ConsoleKit (a `pam_ck_connector.so` PAM modulon keresztül) egy `XDG_SESSION_COOKIE` nevű változót állít be minden általa követett munkamenet első processzének rendszerváltozói között, ami aztán továbböröklődik a munkamenet összes processzére, ezáltal a PID ismeretében a `/proc/PID/environ` fájlokból kiolvashatja, hogy mely processzhez mely session cookie tartozik. A módszer előnye, hogy a különböző UNIX változatok között jól hordozható, de hátránya, hogy bármely processz törölheti ezt a változót a saját rendszerváltozói közül (vagy átírhatja az értékét), ezáltal kivonva magát a ConsoleKit ellenőrzése alól. A systemd nem rendszerváltozókkal dolgozik, hanem egyszerűen a Linux-specifikus control group-okat felhasználva csoportosítja a processzeket, ezáltal sokkal gyorsabb és hatékonyabb követést megvalósítva. A módszer előnye többek között, hogy különleges privilégiumok nélkül a processzek nem tudják kivonni magukat a systemd ellenőrzése alól. A ConsoleKit és a systemd-loginctl minden további nélkül működhet egymás mellett párhuzamosan.

3. Az Udev rendszer

Az udev rendszer alapvető feladata a UNIX eszközfájlok kezelése. A korai Linux-disztribúcióknál az eszközfájlok létrehozása nem automatizált mechanizmussal történt, hanem általában legkésőbb a telepítés során, egyrészt a telepítőkészletről történő átmásolással, részben hardverdetektálás eredményeképpen. Ezeken a rendszereken az eszközfájlok a mai, memóriabeli fájlrendszeres megoldásokkal ellentétben még közönséges merevlemezen levő fájlrendszeren tárolódtak, így a rendszergazdák által kézileg létrehozott eszközfájlokat is beleértve mind túlélték a rendszer újraindítását.

A statikus rendszerfájlok helyett a 2.4-es kerneltől fogva lehetőség volt a devfs fájlrendszer használatára. A devfs egy a proc és sys fájlrendszerekhez hasonló virtuális fájlrendszer volt. Amennyiben a devfs-t felmountoltuk a /dev könyvtár alá, akkor a rendszergazda, sőt bármilyen rendszerszoftver közreműködése nélkül automatikusan biztosította, hogy a kernelben jelenlevő (vagy frissen csatlakoztatott) eszközök /dev alatti eszközfájljai megjelenjenek, a leválasztott eszközök eszközfájljai pedig eltűnjenek. A devfs rendszer programkódja teljes egészében a kernelben helyzkedett el. A 2.4-es Linuxok korában az eszközfájlok valós időben történő, dinamikus létrehozásán és törlésén kívül egyéb feladatot nem végző devfs-t kiegészítette a hotplug alrendszer, amelynek fő feladata a menet közben csatlakoztatott hardvereszközök meghajtóprogramjainak (vagyis kernel-moduljainak) betöltése, és az eszközök esetleges bekonfigurálása volt. A hotplug rendszer működése azon alapult, hogy amikor valamilyen hotplug esemény (pl. hardvereszköz csatlakoztatása, eltávolítása) történt, akkor a kernel meghívta a /sbin/hotplug parancsot (pontosabban azt a binárist, amelynek a neve meg volt adva a /proc/sys/kernel/hotplug fájlban), és rendszerváltozókba kódolva átadta neki az esemény részleteit. A kernel által meghívott hotplug parancs aztán lefuttatta a megfelelő hotplug szkriptet (az ún. ügynököt), ami elvégezte a szükséges driverbetöltést, eszközkonfigurálást, stb.

A 2.6-os kerneltől kezdve a kizárólag a kernelben helyet foglaló, ezért rugalmatlan, és a kernel kódját feleslegesen bonyolító devfs-t kiváltotta az Udev rendszer. Az Udev legfontosabb összetevője a háttérben folyamatosan futó udev démon, amely a hotplug alrendszerhez hasonló módon a kerneltől érkező eseményeket dolgoz fel, ezek alapján dolgozik. Az udev rendszer feladata kezdetben nem igazán mutatott túl a devfs-én, de sokkal rugalmasabban dolgozik annál: A /dev könyvtárban (amely alá a korábbi speciális devfs helyére többnyire egy egyszerű, általános célú tmpfs fájlrendszer került) létrehozta, megfelelő fájljogosultsággal ellátta, vagy éppen törölte az eszközfájlokat. Amint az udev man oldalából kitűnik, az udevd összetett szabályrendszerek szerint dolgozza fel a kerneltől jövő eseményeket. A szabályoknak a udev fejlesztők által írt szabványos, "kőbe vésett" része a /lib/udev/rules.d könyvtárban található, míg a disztribútorok és rendszergazdák által írt (sőt, néha az udevd által futásidőben generált) szabályok a /etc/udev/rules.d könyvtárban kapnak helyet. Az udev szabályok szerencsére nem annyira bonyolultak, mint amilyenek elsőre tűnnek, és az eszközfájlok rendkívül rugalmas kezelését teszik lehetővé. Amikor Greg Kroah-Hartman a devfs Udevvel történő kiváltása mellett kardoskodott, az egyik legfontosabb érve az volt, hogy az udev kiterjedt konfigurációja lehetővé teszi, hogy ugyanaz a hardvereszköz – függetlenül attól, hogy a vele azonos típusú eszközök közül hányadikként lett csatlakoztatva a rendszerhez – az udev konfigurációjában eltárolható azonosítói (pl. kötetazonosító, MAC-cím, WWN, gyári szám, stb.) alapján mindig ugyanolyan néven szerepeljen a /dev könyvtárban, vagy ha ez a név esetleg pl. túl hosszú lenne, mindig legyen egy (vagy több) symlink a /dev könyvtárfában, aminek a segítségével egyértelműen megtalálhatjuk a keresett eszközt. Az udev tehát megoldotta a devfs által el nem ért célt, az állandó eszköznevek biztosítását, ráadásul úgy, hogy közben egyszerűsödött a kernel programkódja.

Manapság az udev rendszer számos rendszerközeli feladatot lát el, többek elvégzi a futásidőben szükségessé váló kernelmodulok betöltését, a /lib/firmware könyvtárból kiolvassa majd átadja a kernelnek a driverek által bekért firmware-eket, a sorozat második részében bemutatott ConsoleKit rendszer adatbázisát felhasználva beállítja az adott eszközfájlok jogosultságait, hogy csak az éppen

aktív konzol előtt ülő felhasználók érhessék el azokat, stb. A legérdekesebb, és egyben legironikusabb azonban az, hogy a legutóbbi időkben az udev elvesztette az eredeti funkcióját, azaz nem hoz létre eszközfájlokat a /dev könyvtárfában.

A legkorszerűbb disztribúciókban a /dev könyvtár alá sima tmpfs helyett devtmpfs típusú fájlrendszer mountolódik, amelynek a létrehozásában ugyanaz a Greg Kroah-Hartman működött nagy mértékben közre, aki annak idején "kifúrta" a devfs fájlrendszert. A devtmpfs védelmében elmondható, hogy a kb. 3600 soros devfs kóddal szemben mindössze 300 sorból áll, mert egyrészt teljes mértékben a meglevő tmpfs kódra épül, másrészt rendkívül egyszerű a működése: a létrejövő eszközfájlok UNIX tulajdonosát, csoportját és jogosultságait egyáltalán nem állítja be, mivel ez továbbra is az Udev feladata, csakúgy mint az eszközfájlokra mutató beszédes nevű symlinkek létrehozása, és a néhai hotplug rendszer funkcióinak (pl.kernelmodul- és firmware betöltés) kiváltása. A devtmpfs alapvetően közönséges tmpfs-ként működik, azaz bármilyen típusú fájlt létrehozhatunk benne a szokásos eszközökkel, az egyedüli különlegessége mindössze annyiban áll, hogy a kernel automatikusan létrehozza és törli benne a csatlakoztatott és leválasztott eszközök UNIX eszközfájljait.

Amint már szó volt róla, az uevent eseményeket a kernel küldi az udevd démonnak, amely az Udev szabályok alapján feldolgozza azokat. Az udevd a feldolgozott (és esetleg módosított) uevent eseményeket továbbküldi, és az udevd által kiküldött udev eseményekből sok esetben D-Bus szignál lesz a rendszer D-Buson. Feltételezhetnénk, hogy az udevd maga is csatlakozik a rendszer D-Busra, és saját maga küldi a D-Bus szignálokat a rendszer D-Buson hallgatózó D-Bus klienseknek. Mivel azonban az udevd nem várhatja meg a D-Bus indulását (és ha esetleg megvárhatná, sem függhetne tőle, mert rendkívül megbízhatónak kell lennie), nyilvánvaló, hogy az udevd nem tart fenn semmilyen közvetlen kapcsolatot a D-Bus-szal. Ehelyett van egy közvetítő réteg néhány olyan démon formájában, amelyek az udevd-vel és a D-Bus-szal egyaránt kapcsolatban állnak: fogadják az udevd-től érkező feldolgozott ueventeket, ismét feldolgozzák, és a feldolgozás eredményeképpen D-Bus szignálokat küldenek ki a rendszer D-Buson, ezen kívül pedig különböző D-Bus interfészek és azok metódusainak formájában olyan szolgáltatásokat nyújtanak a D-Bushoz kapcsolódó kliensek (többnyire grafikus felhasználói felületek, pl. KDE, Gnome) számára, amelyek lehetővé teszik a hatáskörükbe tartozó eszközök aszinkron lekérdezését vagy akár menedzselését. Ilyen köztes démonok pl. a már többször említett UPower, udisk és NetworkManager alrendszerek.

4. A systemd démon

A systemd feladatköre hagyományos UNIX ismeretek birtokában sokkal inkább megfoghatónak és jobban körülhatároltnak tűnik, mint a régi UNIX-okon ismeretlen - és rendkívül szerteágazó - funkciót betöltő Udev/D-Bus rendszeré, és mivel a fejlesztése viszonylag korai fázisban van és még eléggé centralizált, létezik hozzá néhány olyan minden vonatkozását lefedő dokumentáció, amelyeket némi UNIX, Linux-kernel és Udev/D-Bus ismeret birtokában átolvasva eléggé pontos képet kapunk arról, hogytulajdonképpen mi a systemd: A systemd az egész 1-es PID, azaz a /sbin/init teljes funkcionalitását átveszi, sőt a fejlesztői a jövőben nem csak a rendszer munkamenetét, hanem felhasználói session-öket is kívánnak vezérelni vele, azaz a gnome-session, kdeinit és hasonló démon processzek kiváltását is tervbe vették. Ez utóbbi lépés eredményeképpen a systemd-ből a dbus-daemonhoz hasonlóan több példány futna: egy rendszerpéldány, és felhasználói munkamenetenként egy-egy további session-példány.

A systemd célja nem csak /sbin/init, és a System V init rendszer központi vezérlőszkriptjének leváltása, hanem magukat az init szkripteket is ki akarják váltani. Ennek több oka van:

- Az init szkriptek futtatásához interpreter (többnyire UNIX shell) szükséges, ami a boot-folyamatban szerephez jutó többszáz szkript esetében hatalmas felesleges CPU, I/O és memóriaterhelést okoz.
- Az init szkriptek felépítése nehezen szabványosítható, még a különböző Linux-disztribúciók között is nagyok lehetnek a különbségek ilyen szempontból. A szkriptek funkcióját a systemd viszonylag egyszerű konfigurációs fájlok által vezérelve tölti be.

Az 1-es PID-del futó processz legfőbb különlegessége a UNIX rendszereknél egyébként nem is az, hogy az operációs rendszer bootolását és levezénylik, hanem az, hogy azokat a futó processzeket, amelyeknek a szülő processze "meghal", a kernel az öröklési fában áthelyezi alá, mintha az 1-es processz lenne a szülőjük. Ezzel a kernel azt a feladatot rója az init processzre, hogy a hozzárendelt gyermekprocesszek állapotát olvassa ki a wait rendszerhívással miután meghaltak, mert amíg ezt nem teszi meg, a kernel kénytelen fenntartani egy adatstruktúrát a már halott processz számára. Az ilyen halott, de wait rendszerhívással még le nem "aratott" processzeket zombi processznek hívjuk. A systemd természetesen betölti a korábban az init által végzett aratási feladatot, mert ha nem tenné, az elszaporodó zombi processzek kernelbeli adatstruktúrái szélsőséges esetben a memória elfogyásához is vezethetnének.

A systemd a teljes rendszerkonfiguráció szabványosításának ígéretével és az agresszív, eseményvezérelt párhuzamosításokkal rendkívül ambiciózus projektnek számít. Hasonló irányba indult el, mint az Ubuntu upstart rendszere, de még annál is radikálisabb, viszont annak ellenére, hogy messzebb jutott a kiindulóponttól, az upstarttal ellentétben mégis visszamenőleges kompatibilitást biztosít a System V init rendszerrel, tehát egyértelműen jobb választásnak tűnik. Valószínű, hogy idővel az Ubuntu is beépíti a saját operációs rendszerébe. Az egyszerű UNIX /sbin/init kiváltására egyébként már több próbálkozás is született az idők folyamán, mint pl. az Apple launchd rendszere, amelyhez a systemd fejlesztői legszívesebben hasonlítják a saját megoldásukat.

A systemd projektet a freedesktop.org gondozza, és amint láttuk, egyre jobban integrálódik az Udev/D-Bus rendszerekkel, sőt előfordulhat, hogy azok egy idő után működésképtelenek lesznek a systemd nélkül. A systemd erőteljesen támaszkodik több Linux-specifikus szolgáltatásra (pl. control group-ok, autofs/automount), és a fejlesztők nem is törekednek a systemd hordozhatóvá tételére, mert gyakorlatilag a Linuxot tartják az egyetlen életképes szabad operációs rendszernek. Ha ez az integráció tovább mélyül, elképzelhető, hogy a D-Bus-ra nagy mértékben alapozó asztali környezetek (pl. KDE, Gnome) nem, vagy csak csökkent funkcionalitással futnak majd a nem Linux kernelre épülő operációs rendszereken.

Jelen előadás anyaga kibővített, hivatkozásokat is tartalmazó, közösségileg megvitatott formában megtalálható az alábbi címeken:

```
http://hup.hu/node/112319
http://hup.hu/node/113796
http://hup.hu/node/114595
http://hup.hu/node/114629
```