1.344 Ft

II. Évfolyam 10. szám

ISSN 15865185

tech.net

A Microsoft Magyarország szakmai magazinja

Hatékony megoldás a jogsértő tartalommal szemben
23. oldal

Megértő gépek
23. oldal

HASH!
13. oldal

A „**tech.net magazin Brainstorm**" a Dupla KV rovathoz hasonló, ám a személyes kérdésfelvetést és vitát is lehetővé tevő rendezvény, melynek célja:

- **az elsőre talán ismeretlen technológiák élő bemutatása**
- **a cikkekhez kapcsolódó kódok megírása/kipróbálása**
- **a terjedelmi okokból kimaradt információk átadása**

E magazinnal együtt a rendezvényre érvényes belépőjegyet minden előfizetőnkhöz eljuttattuk.
További információk a belépőjegyen olvashatók.

Várjuk Önöket a NetAcademia Mesterkurzusokon

NETACADEMIA

A legjobbakat tanítjuk.

(Bár belépőjegyet adtunk, ez a tény önmagában nem biztosítja helyét a rendezvényen. A jegy célja előfizetőink elsődlegességének biztosítása, de a korlátozott résztvevői létszám (100 fő) miatt a regisztráció kötelező. Jelentkezzen, amíg nem késő!)

http://technet.netacademia.net/brainstorm

HunNet RL512W

512 kbit/sec-os szimmetrikus forgalomfüggetlen csatlakozás az internetre havi 35.000 Ft-ért

.hu Hungarian Networks

Tel: 06-40-hunnet (486-638)

2001 Október

tech.net

A Microsoft Magyarország Szakmai Magazinja

Szerkesztőség
Főszerkesztő: **Fóti Marcell**
marcellf@netacademia.net
Főszerkesztő-helyettes: **Fülöp Miklós**
mick@netacademia.net
Szerkesztőség címe:
1105 Budapest, Ihász utca 13.
Tel.: 263-2732
technet@netacademia.net
Nyilvános levelezési lista:
tech.net@lyris.netacademia.net

Kiadja és terjeszti
a **NetAcademia Kft.**
Terjesztési, előfizetési információ:
Tel.: 263-2732
terjesztes@netacademia.net
Megjelenik havonta, ára 1.344 Ft
Példányszám: 3.000

Előfizethető megrendelőlevélben a szerkesztőségnél:
1105 Budapest, Ihász utca 13.
Fax: 261-7145
http://technet.netacademia.net/subs

Hirdetésfelvétel:
Bársonykalapács Marketing
Felelős: **Balogh Zoltán**
Tel./Fax: 214-0923
info@velvethammer.hu
1027 Budapest, Fő utca 67. V. 1.
Grafikai tervezés, kivitelezés,
nyomdai előkészítés:
Bársonykalapács Marketing
Művészeti vezető: **Balogh Zoltán**

Nyomda:
Cerberus Kft.
1066 Budapest, Lovag u. 14.
Felelős vezető: **Schmidt Gábor**

ISSN 1586-5185

Szülinap

Egyévesek lettünk, sőt, már el is múltunk. Pont egy évvel ezelőtt, 2000 szeptemberében jelent meg lapunk első száma. Az elmúlt egy évben igen sok mindenről szó esett lapunk hasábjain, s mindig megpróbáltuk tartani magunkat az eredeti koncepcióhoz: valami olyasmit adjunk, ami máshol nem érhető el. Olyan olvasmányokat adjunk közre, melyek ha néha a kissé szabadabb stílus, szóhasználat miatt könnyednek tűnnek is *(hisz magunk között vagyunk, megtehetjük :-)* tartalmukban sűrűek. Sűrűek, és – amennyire csak ez lehetséges – hibátlanok, mi több, objektívek! Az objektivitás megőrzése kívülről talán nem tűnik egyszerű feladatnak, ha az ember a Microsoft Magyarország Szakmai Magazinjába ír, de szerencsénkre egy olyan Microsoft Magyarország áll a hátunk mögött, mely felismerte, hogy sokkal nagyobb, hosszú távú haszon rejlik egy-egy kényelmetlen, de őszinte kijelentésben, mint amennyit a tények csűrése-csavarása hoz. Ha egyáltalán hoz, és nem visz. Ez a stratégia állított olvasóink táborába nem egy hithű junikszost is – akik így legalább belülről ismerik meg az ellenséget.

A folyamatos, lelkes munkát egész évben végigkísérte a határidők folyamatos lecsúszása. Küzdünk ellene teljes akaratunkból, de sajnos mindig kiderül, hogy egy informatikai lap készítése is informatikai munka, azaz érvényes rá a négyes szorzó paradoxona. Az vesse ránk az első követ, aki rendszergazdaként, vagy méginkább programozóként valaha az életben is sikeresen megbecsülte volna egy projekt hosszát. A szorzóparadoxon miatt ugyanis ha helyesen becsülte fel, az értéket meg kell szoroznia néggyel, tehát nem helyesen becsülte fel. *(Agytorna: a sevillai borbély azokat borotválja, akik nem maguk borotválkoznak. Övele mi lesz?)*

Mit kapunk szülinapunkra?

Mint minden ünnepelt, mi is kaptunk meglepetéseket. Itt van mindjárt a címlapon lévő torta. Más ajándékra is fájt a fogunk, s meg is kaptuk olvasói visszajelzések formájában. A szeptemberi, kanálhajlítós szám valami elképesztő sikert aratott mind a külcsínt, mind a belbecset illetően. Köszönjük! Hálánk jeléül egyes előfizetőinknek hibásan írtuk fel a címlapra a Memóriakzelés szót, hogy példányuk még értékesebbé váljon. Mint egy hibásan fogazott bélyeg: aranyat ér!

Mit adunk?

Az európai kultúrkörben nem szokás ugyan, hogy az ünnepelt viszonozza az ajándékokat, mi mégis így teszünk. Azt szeretnénk, hogy minden kedves előfizetőnk érezze: mi egy csapat vagyunk, a közös cél a magyar informatikusok élvonalban tartása.

MesterQrzus

Az év folyamán született meg a MesterQrzus ötlete, mely egyfajta író-olvasó találkozó, és agytorna. A MesterQrzust azért vezettük be, hogy az esetleg nehezebben átadható-leírható mondanivalónak is helyet biztosítsunk. Én magam is érzékeltem, hogy a szó, mely ugyan elszáll, mennyit segít bizonyos témakörök feldolgozásában! A minden hónap utolsó péntekén tartott MesterQrzusok igen látogatott eseményekké nőtték ki magukat!

Mikulás

Tavaly decemberben jött el hozzánk először a Mikulás, akit ablakba rakott, kisuvickolt NetAcademia bögrével várt az a százhúsz olvasónk, aki eljött az akkori rendezvényre. Mivel nem vártak hiába, az a gyanúnk, hogy idén is eljön! Tegyék szabaddá magukat december ötödikén, hátha a Mikulás értékeli szorgalmukat, hogy még azon a napon is Kerberost hallgatnak, VBScriptet írnak, LDAP-pal bütykölnek stb. Mikulásrendezvényünk lesz az első alkalom, hogy kipróbáljuk az interaktív konferenciázást. Érezd a Kerberost!

NATE

Ez a vonat már elment, de jövő februárban megint indul a NetAcademia Továbbképzési Estek előadássorozat, ahol egy adott témakört több héten át boncolgatva alapos hozzáértést lehet szerezni. A csütörtök délutánonkénti bitfúrás-bitfaragás témaköreit kibővítjük. Marad a Win2k, de bejön a képbe még egy-két olyan téma, ahol a közös reszelés mindenkinek (írónak, olvasónak) előnyére válik. Ilyen lesz *(bízvást)* az SQL alkalmazásfejlesztés, *(talán)* a security *(Hacking is easy!)* és egyebek.

Könyvkukac

Az MSDEV listán merült fel először az igény, hogy olyanok is hozzájuthassanak méregdrága *(tizenötezer forintos, sőt drágább!)* könyvekhez, akik ezek megvásárlására nem rendelkeznek megfelelő anyagi háttérrel/értelmes főnökkel. Az ötletre azonnal lecsaptam, és már készül is a Napster csereberéléséhez hasonló elven működő könyvkukac web, ahol nemcsak kölcsönözni, hanem felkínálni is lehet majd köteteket. Csak és kizárólag előfizetőknek!

Msdownload.netacademia.net

Az msdownload szintén az Önöké, de nem mi adjuk, hanem a Microsoft Magyarország. Mi csak a kivitelezők és fenntartók vagyunk. Mostantól nem kell külföldről letöltögetni a javítócsomagokat, ingyenes programokat és hasonló jószágokat. Kereshető, magyarnyelvű leírásokat tartalmazó adatbázis, tíz gigabájtnyi adat, száz megabites hálózat – ez a http://msdownload.netacademia.net!

Jövőkép

S hogy mit hoz a jövő? A táguló világegyetem magával húzza, egyre tágítja az informatikát is. Lassan rájövünk, mi mindenre jó az Internet, elszaporodnak a .NET alkalmazások, s mi megint kezdhetjük elölről a tanulást. Stílszerűen kifejezve: SOAP-ás. De nem tartom fel Önöket tovább: olvassanak!

Fóti Marcell
marcellf@netacademia.net

Ntbackup.exe Windows 2000-rel

A Windows NT és a Windows 2000 beépített mentőszoftvere az NTBackup, minden feltelepített verzióban elérhető, s az ntbackup.exe-vel indítható. A Windows NT 4.0 óta sokat változott, és a Windows 2000-ben a már majdnem teljesen tökéletes ntbackup.exe-t találhatjuk. Sajnos azonban korántsem tökéletes, néhány gyermekbetegség még mindig felfedezhető rajta, és sok esetben az is észrevehető, hogy ami a Windows NT 4.0 alatt futó Backup programban egyértelmű volt, az hiányzik a Windows 2000 Backup-ból. Ennek oka nyilvánvaló: az egész programot átírták, nem maradt a régiből egy bit sem.

Az új NTBackup nem is Microsoft fejlesztés!

Mi inspirált a cikk megírására? Elsősorban az, hogy az ntbackup.exe és a Removable Media Storage *(későbbiekben RMS)* párossal igencsak meggyűlt a bajom, mire működésre bírtam, illetve a környezetemben, és a Windows 2000 levelezési listán is azt tapasztaltam, hogy mások számára sem volt egyértelmű ez a technológia. Másodsorban pedig az hajtott, hogy nem akartam elhinni, hogy a Microsoft *(majdnem)* tökéletes operációs rendszerében található beépített eszköz nem működőképes.
Cikkem során szeretnék túllépni a termékpáros *(ntbackup-RMS)* egyszerű bemutatásán. Megpróbálok mentési szisztémát ajánlani, illetve néhány elvárást támasztani a következő verziójú ntbackup.exe-vel szemben – hátha megfogadják Redmondban :).
A program neve Windows NT, és Windows 2000 alatt is ntbackup.exe *(hiszen a Windows 2000 is NT)*, pedig, mint említettem, a két változat korántsem egyforma. Előfordul olyan is, amikor sűrűn váltogatva ntbackup.exe-ként hivatkozom a Windows NT 4.0 és a Windows 2000 változatára. Ilyenkor mindig kiírom előtte, hogy melyikről írok.

Az ntbackup.exe régen és ma

A Windows NT 4.0-ban levő ntbackup.exe a maga nemében tökéletes volt, hisz tökéletesen működött, azonban a korral haladva fejlesztése elkerülhetetlenné vált. Vállalati környezetben nem elegendő az, hogy manuálisan tökéletesen tudunk menteni. Szükségünk van arra is, hogy off-line időpontokban, felügyelet nélkül tudjunk teljes mentést végezni. A Windows NT 4.0-ban található ntbackup.exe erre úgy adott lehetőséget, hogy saját készítésű mentési scriptet/scripteket kellett készíteni; ebben a verzióban beépített időzítési lehetőség még nincs, az új verzióban már ez is megtalálható. Nagyvállalati környezetben elengedhetetlen, hogy könnyedén tudjunk távoli erőforrásokat menteni. A régi verzióban erre szintén nincs közvetlen támogatás, azonban az új már ezt is magában hordozza. A harmadik nagy újdonság pedig az, ami az egész programot megváltoztatta és talán a legtöbb problémát okozza a rendszergazdák számára: a kazetta formátuma. A Windows NT 4.0 ntbackup.exe-je minden kazettára hajlandó volt menteni, amit csak a szalagos egységbe betettünk, a Windows 2000 esetében ez már egy kicsit másképp működik. A kazettákért *(media)* az RMS a felelős és megmondhatjuk, hogy mikor melyik kazettára menthetünk. Ezzel elkerülhető, hogy egy fontos mentést felülírjunk egy új mentési feladattal *(bővebben később)*.
Még egy újdonságot meg kell, hogy említsek. Az új ntbackup.exe-vel már fájlba is készíthető mentés, és arról bármikor elvégezhető visszatöltés. Így akár kisebb rendszerekről is készíthető mentés anélkül, hogy szükséges lenne drága mentési egységek beszerzése. Természetesen továbbra is ajánlott megfelelő mentési egység beszerzése.
Fontos újdonság, hogy míg a Windows NT 4.0 esetén a mentési egységen *(szalagon)* katalógusinformáció is helyet kapott, addig erre már nincs szükség az új ntbackup.exe esetén. Miért jó ez? Aki már állított vissza NT 4.0 mentésből az tudja, hogy a Catalog Status az egyik legfontosabb és a legkritikusabb része a visszaállításnak. Sajnos, ha a Catalog megsérül akkor nem lehet a mentést visszaállítani. Windows 2000 esetén nincs szükség a Catalog-ra így az elveszett mentések száma is csökken.

Távoli erőforrások mentése

Beszéltünk már arról, hogy a Windows 2000 ntbakup.exe-vel könnyedén lementhető bármely távoli erőforrás. Egyszerűen csak az ntbackup.exe-ben tallózva kiválasztjuk a mentendő erőforrást. NT 4.0 esetében is megoldható volt a távoli mentés: fel kellett csatlakoztatni *(MAP)* a kívánt erőforrást és utána már le lehetett menteni. Ennek oka az volt, hogy - mint az intézőben tallózva - csak a betűjellel ellátott meghajtókat tudtuk lementeni, a régi NTBackup nem ismerte az UNC útvonalakat *(\\kiszolg\megoszt)*. Ennek a módszernek óriási hátránya, hogy mivel nem UNC path-tal lettek lementve az adatok, mind a visszaállítás, mind a dokumentálás nehézkesebb. Egy tavaly júliusban lementett I: meghajtó kevésbé „beszédes", mint pl. a \\srv01\d$\exchsrvr\logs útvonal.
A Windows 2000 Backup-ban végre UNC Path segítségével is könnyedén menthetünk.

Beépített időzítő segíti a munkánkat...

...használatával a mentéseket jobokként kezelhetjük és minden jobnak megmondhatjuk, hogy mikor futhat, mennyi a maximális futási idő, kinek a nevében stb. Ezen újdonságok csak egy része újdonság valójában. A Windows NT 4.0-t telepítve is volt lehetőségünk programok, saját készítésű parancsfájlok, így mentések időzített indítására. Az at parancsot használva beidőzíthettünk parancsokat, a parancsfuttató felhasználót is beállíthattuk - ugyan nem parancson-

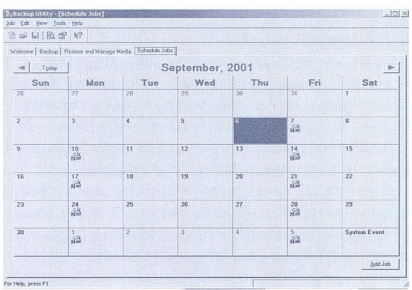
ként, de amennyiben a Scheduler szolgáltatás tulajdonságainál a Service Account nevét megváltoztattuk, az at-vel időzített parancs is a megadott felhasználó nevében, annak jogosultságaival futott le. Internet Explorer 5.0 telepítése esetén új szolgáltatás települ a Windows NT 4.0-ra is: a Task Scheduler *(természetesen, akár ki is hagyható a telepítése, de az alapértelmezett telepítés frissíti a rendszert)*. A Windows 2000 alapértelmezésben tartalmazza ezt az új szolgáltatást. Alapvetően nem változott sokat az at-hez képest, viszont sok új többletszolgáltatást hordoz magában:

- jobonkénti service account megadásának lehetőségét
- maximális futási időt
- kényelmes grafikus kezelhetőséget

Ezek nagyon fontos tulajdonságok, s a Windows 2000 Backup az új időzítő *(Task Scheduler)* minden plusz szolgáltatását tökéletesen kihasználja.

Egy mentési job elkészítésekor *(vagy utólag is)* megadhatjuk a futási környezetet. Beállíthatjuk, hogy mi legyen futáskor használt kezdőkönyvtár, mikor fusson, milyen rekurziót használjon, mekkora legyen a futás maximális időtartama, mi legyen a mentéskor használt felhasználó neve és jelszava. Az ntbackup.exe-ben egy naptárban ábrázolva láthatjuk, hogy mikor milyen mentési job fog futni.

Sajnos ha egy jobot adminisztrálunk, bármit módosítunk azon, jóváhagyáskor – amennyiben előzőleg megadtuk – a rendszer kéri tőlünk a felhasználónév/jelszó párost. Ez minden módosításra érvényes. Elég bosszantó funkció, de meg lehet kerülni: amennyiben a felugró ablakban a Cancel gombot választjuk, a job emlékezni fog az előzőleg megadott információra és megőrzi működőképességét. *(Jalos: köszönöm)*



Naptárnézet az ntbackup.exe-ben

Nem kell a katalógusinformáció...

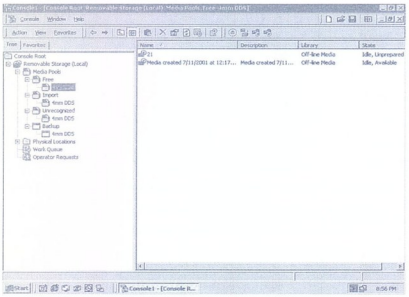
...mert a Windows 2000 a Windows NT 4.0-hoz képest teljesen másképp kezeli a kazettákat, médiumokat. A Windows NT NTBackup.exe minden használatkor a kazettáról olvasta le a katalógusinformációt, amit a használat után *(az ntbackup.exe bezárása után)* törölt, majd a kazetta ismételt használatakor ismételten leolvasta, ha tudta. *(A Temp könyvtárban hozta létre u01 végződéssel ezeket a fájlokat.)* A ravasz rendszergazda még az ntbackup.exe bezárása előtt lemásolta ezeket az eredeti helyükről egy másik könyvtárba. Ekkor megmaradt a katalógusinformációnk teljes értékű mentése.

Windows 2000-ben a kazettákért és minden más médiáért a Removable Media Storage *(RMS)* felelős. Az RMS elsődleges feladata, hogy a médiák *(Médiumok? Ehh! Ki tudja ezt ragozni? – a szerk.)* és az alkalmazások közötti „fordító" szerepet betöltse. Az RMS a médiákat úgy nevezett Media Pool-okba szervezi. Ezekben elvileg minden media megtalálható, amit a gépünkön használtunk; a médiákat GUID alapján különíti el egymástól. A media pooloknak két fő verziója létezik:

- System media pool: ezt csak az RMS használhatja, más alkalmazás nem módosíthatja. Az alkalmazások hozzáférhetnek a Free System Media Pool-hoz, az abban levő médiákat a saját Application media pool-jukba mozgathatják.
- Application media pool: az alkalmazások kezelik a poolokat, az alkalmazások az általuk használni kívánt mediákat a Free System media pool-ból veszik át. Egy gépen akár több Application media pool is lehet, amit akár az alkalmazás is létrehozhat, de természetesen kézzel is elkészíthetők a megfelelő pool-ok a Microsoft Management Console Removable Media Storage konzolból.

Az ntbackup.exe automatikusan, első indításkor létrehozza a Backup nevű Application Pool-t. Az alkalmazás *(ntbackup.exe)* bezárását követően az RMS megtartja, lementi a kazettához vagy más médiához tartozó katalógusállapotot. A média ismételt használatakor, már a berakást követően az RMS a GUID alapján a médiát felismeri és tudja, hogy mi található az egységen. Ez ugye egyben azt is jelenti, hogy ha egy média off-line állapotban van, az RMS akkor is tud a médiáról mindent. Tudja azt is, hogy a média épp off-line állapotban van. Miért is jó ez? Annak, hogy az RMS beépült a média és az alkalmazás közé, az a gyakorlati haszna, hogy míg eddig az ntbackup.exe *(Windows NT 4.0 alatt)* az egységben levő bármely médiához korlátlanul hozzáfért, addig Windows 2000 alatt az ntbackup.exe csak és kizárólag az Application Pool alatt található médiához fér hozzá: oda importálhat médiát. Ennek következménye az is, hogy egy mentési job elkészítésekor megmondhatom azt, hogy melyik médiára menthet és ezzel kizárom annak lehetőségét, hogy egy korábbi mentésemet felülírjam a jelenlegivel. Az RMS a médiák intelligens kezelését biztosítja.

Az RMS elérhető a Computer Managenet Snap-in – ből vagy a Microsoft Management Console Removable Media Storage Snap-in hozzáadásával. Megnyitás után láthatjuk a System Media Pool-okat *(Free, Import és Unrecognizable)* és az Application Media Pool-okat *(köztük a Backup Pool)*. Minden System Media Pool alatt megtalálható a mentési egység. Az én gépemben egy 4mm DAT egység található, de ha több eszközöm lenne, mindegyik megtalálható lenne a Poolok alatt. Az RMS segítségével a Poolok között a médiákat mozgathatjuk. Az alkalmazás csak az alkalmazás Pool-ban levő médiát használhatja *(az NTBackup csak a Backup Poolban lévőt)*, vagy a Free System Poolból a saját Pooljába mozgathat egy üreset.



Removable Media Storage (RMS)

CD író és a Windows 2000 RMS

A CD írók használatával kapcsolatban meg kell jegyeznem egy hasznos, fontos információt, mielőtt még többen nekiesnének az írónak, a TechNet-nek és az MS Supportnak. A Windows 2000 ntbackup.exe nem képes közvetlenül mentést készíteni a CD-R, CD-RW és DVD-R típusú lemezekre, eszközökre. Ennek gyakorlatilag elég egyszerű oka van: a Free System Media Pool alá nem tudjuk mozgatni a nyers, üres lemezeinket. Ennek hátterében az áll, hogy az RMS a Free Media Pool-ba csak a felismerhető fájlrendszert tartalmazó adathordozót képes mozgatni. Ez sajnálatos módon a by design kategóriába tartozik, vagyis a Microsoft ezt így tervezte, ez nem hiba. A CD író programok használata elkerülhetetlen, legalább a lemez formázására.

Azonban jó hír, hogy az október 25-én debütáló Windows XP már képes lesz a CD-R, CD-RW, DVD-R lemezek natív kezelésére. Sajnos a Windows XP-t futtató gépemben *(még)* nincs CD író eszköz, de hamarosan tesztelni fogom ezt a funkciót is, az eredményt az újságban közzétesszük.

Ntbackup.exe és a parancssor

A Windows 2000 ntbackup.exe-nek 18 parancssori kapcsolója létezik. A parancssori kapcsolók teljes leírása, ismertetése megtalálható a Help-ben *(az ntbackup /? Parancs beírásával megjelenik)*.

A Windows 2000 Backup bks kiterjesztésű fájlt használ a mentendő útvonalak eltárolására. A bks fájl text formátumban külön sorokba felsorolva tartalmazza a mentendő adatokat. A szintaktikája annyira egyszerű, hogy nincs is igazi szintaktikája. Egyszerűen fel kell csak sorolni külön sorokba a mentendő adatok elérési útját. Amennyiben valamit nem szeretnénk menteni, akkor a sor végére a /Exclude szöveget kell megadnunk. A következő pár sor egy bks fájl részlete, ami meghatározza, hogy a C:\CmdCons, a C:\Docs könyvtárakat lementjük, de a C:\Docs\Temp könyvtárat nem:

```
C:\CmdCons
C:\Docs
C:\Docs\Temp /Exclude
```

A parancssori kapcsolók közül kiemelném a Windows NT 4.0-hoz képesti újdonságokat: /T a kazetta neve; /N név az új kazettának; /D címke a mentésnek; /UM az első elérhető szabad kazettára ment; /P a media pool nevét határozza meg.

A tökéletes és automatikus mentés beállításához a következő lépéseket kell elvégezni:

- Be kell helyezni a mentésre kijelölt szalagot a meghajtóba, és egy kisebb mennyiségű adatot az ntbackup.exe-vel menteni kell. Erre azért van szükség, hogy az MTF *(Microsoft Tape Format)* a kazettára megfelelően felkerüljön. Ezzel egyidőben az Application Media Pool-ba vándorol a kazetta, így azt az NTBackup a későbbiekben is használni tudja majd.
- Az előző lépést minden használni kívánt kazettával el kell végeznünk. Érdemes mentés közben beszédes címkéket és leírásokat használni a kazettákhoz. Erre jó példa a mentési napok és a mentési típusok keveréke pl.: Monday Full Backup.
- Létre kell hozni az NTBackup.exe segítségével a mentési jobokat úgy, hogy miután megadtuk a mentendő adatokat, ki kell választanunk, hogy hová szeretnénk menteni. Ekkor a legördülő mezőben láthatjuk az előzőleg elkészített kazettákat és azok közül valamelyiket válasszuk ki. Vigyázat: ezzel azt érjük el, hogy arra és csak arra a kazettára, fog menteni az NTBackup.exe!
- Amennyiben mentési ciklusunkba új kazettát szeretnénk utólagosan felvenni, ismét helyezzük be a kazettát, mentsünk rá valami minimális adatot, majd a szükséges jobokat módosítsuk.

Tippek:

Az RMS a kazettákat GUID, és nem a nevük alapján különbözteti meg egymástól. A GUID-ok megtekintésére kiváló eszköz a Windows 2000 Resource Kitben található rsm_dbutil.exe.

Sajnos alapértelmezésben a bks fájlok a User Profil\User Name\Local Settings\Application Data\Windows NT\NtBackup elérési út alatt jönnek létre. Természetesen elmásolható erről a helyről, de utána kézzel ki kell javítani a job*(ok)*-ban a hivatkozási útvonalat.

A Backup log a User Profil\User Name\Local Settings\Application Data\Windows NT\NtBackup elérési út alatt jön létre. Ezt nem tudjuk módosítani. Ez bizony nagy-nagy hiba!

Az NTBackup.exe nem futtatható Interactive módban, ami azt jelenti, hogy nem tudja bármely bejelentkező felhasználó felügyelni a futást. Az ntbackup.exe programot csak az látja a képernyőm, akinek a nevében fut. *(A Windows NT 4.0-ban megszokott Interactive kapcsoló egyszerűen hiányzik. Sajnos a Windows XP-ben a hiba szintén megtalálható.)*

Az ntbackup.exe az /um kapcsolóval *(UnManagement)* futtatva az első szabad médiára ment.

Mentési stratégiák

Mikor beszélhetünk jó mentésről? Jó és sikeres mentés az, amikor a rendszer helyreállíthatóságához, és a felhasználók szempontjából minden hasznos és fontos adatot sikerül lementenünk. De elegendő-e csupán az, ha az adatokat le tudtuk menteni? Nyilván nem elegendő, hiszen ezeket helyre is kell tudnunk állítani. A jó és sikeres mentés kivitelezése elég összetett feladat. Rendszertervezésekor mentési stratégiát is kell készítenünk. A stratégiánknak tartalmaznia kell a következőket:

- Mit kell lementenünk, azoknak mekkora mérete?
- Milyen gyakorisággal kell mentenünk az adatokat?
- Ki felügyeli a mentések helyességét?
- Hány kazetta, hány egység áll rendelkezésünkre?

Mentési stratégiánkat úgy kell elkészíteni, hogy a rendelkezésünkre álló eszközöket és a mentési igényeket össze kell hangolnunk. Szerencsésebb eset, amikor csak az igények vannak meg, és a szakember a mentési stratégiát kidolgozva az ahhoz szükséges összes eszközt megkapja. A fenti adatok elengedhetetlenül fontosak a jó mentési stratégia meghatározásához és elkészítéséhez.

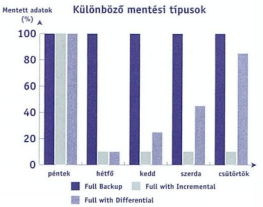
Meg kell határoznunk, hogy mik azok az adatok amiket mentenünk kell. A rendszer helyreállíthatóságával kapcsolatban is nehéz megmondani, hogy mit kell mentenünk, de általánosságban elmondható, hogy a SystemState mentése mindig ajánlott. A felhasználói adatok mentését mindig a rendszer sajátosságai határozzák meg számunkra. Meg kell keresni azokat a személyeket, akik ezeket a kérdéseket eldönteni hivatottak. Ezek után meg kell nézni a mentendő adatok tényleges méretét, azokat egy próbamentéssel ellenőrizni. Minden esetben ajánlott 10%-kal feljebb kalkulálni a mentést, mert amennyiben HOME könyvtárakat mentünk, előfordulhat, hogy a megnövekedett adatmennyiség már nem fér rá az egységre. A mentés során a kazetták archiválására és azok újrahasznosítására is gondolnunk kell. Ez szintén függ az adott rendszer felépítésétől. Gondolni kell arra is, hogy hány napra visszamenőleg szeretnénk megőrizni a mentéseket. A tökéletes mentési stratégia kidolgozásához elengedhetetlenül fontos ismernünk a mentési típusokat, azok előnyeit, hátrányait; néhány példát is megnézünk ezekkel kapcsolatban.

Eltérő mentési típusok

A Windows 2000-ben ötféle mentési típus közül választhatunk:

- Normal *(Teljes mentés; FULL)*: a kijelölt fájlok és könyvtárak mindig mentendők. Minden újabb mentéskor újra mentődnek. Visszaállítása egyszerű és gyors, mert elegendő a legutolsó helyes mentést előkeresni és azt visszaállítani. Használata során azonban sok kazettára van szükség, ha több napra visszamenőleg is szeretnénk a mentéseket megőrizni. Viszont egy-egy kazetta felülírása nem befolyásolja a visszaállíthatóságot. Az utolsó helyes kazettáról az adatok visszaállíthatóak.
- Incremental: csak az utolsó normal vagy Incremental mentés óta módosult fájlok jelölődnek meg mentendő adatokként. Használata során az utolsó sikeres normal mentés kazettájára és az összes inkrementális szalagra is szükségünk van.
- Differential: csak az utolsó normal mentés óta módosult adatok mentődnek le.
- Copy: minden mentéskor minden adat lemásolódik. A fájlok archive attribútuma nem változik.
- Daily Copy: az adott napon módosult adatokat menti le. A fájlok archive attribútuma nem változik.

Az öt különböző mentési típust a megfelelő mentési stratégia eléréséhez kombinálva vagy akár önmagában is felhasználhatjuk. A leggyakoribb mentési stratégiákat bemutatom. Elkészítettem egy szemléltető ábrát is, ami jelöli a mentendő adatokat a különböző mentési típusoknál.



Különböző mentési típusok

Normal Backup *(Teljes mentés)*:

Ezt a stratégiát használva minden nap minden adatot lementünk. Ennek a mentési stratégiának a hátrányai a következők: a mentési idő viszonylag hosszú *(mert minden nap mindent le kell menteni)*; minden napra egy ugyanakkora kazettára van szükségünk, mert a mentendő adat mindig azonos *(természetesen ha nem változik; %-ban értendő)* Előnye: amennyiben vissza kell állítanunk az adatokat, elegendő az utolsó sikeres mentéshez használt kazetta adatát visszaállítani. Archiválás esetén elegendő egy kazetta archiválása.

Normal with Incremental *(Teljes növekményes mentés)*:

Ezt a stratégiát használva teljes mentés készül pénteken ahogy az az előző ábrán is látható. Ezt követően hétfőn lementődik minden ami módosult péntek óta, kedden lementődik minden ami módosult hétfő óta, és így tovább, pénteken pedig ismét egy teljes mentés készül. Ha mondjuk a keddi állapotot kell visszaállítanunk, ahhoz szükségünk van az utolsó teljes mentés kazettájára, és az azóta készült összes inkrementális kazettára is. Ha bármely kazetta sérül, nem lehet visszaállítani a mentést. Előnye: a teljes mentés után az inkrementális mentések ideje lényegesen kisebb mint a teljes mentésé, s kisebb helyet is foglal. Az archiváláshoz el kell rakni egy egész ciklust, így lényegesen több kazettára van szükségünk.

Normal with Differential *(Teljes különbségi mentés)*

Ezt a stratégiát használva teljes mentés készül pénteken, majd hétfőn a péntek óda módosult adatok mentődnek, kedden szintén a péntek óta módosult adatok, stb. Ennek köszönhetően a mentendő adatok mennyisége – ahogy haladunk előre az időben – növekszik. *(lsd: ábra)* Amennyiben a szerdai állapotot kell visszaállítani, elegendő az utolsó teljes mentés *(péntek)* és a szerdai kazetta. Hátránya: minden nap tovább tart a mentés és egyre több adatot kell lementeni. Amennyiben archiválni szeretnénk bizonyos állapotot, legalább két kazettát el kell raknunk.

Nehéz feladat...

...a Windows 2000 és az RMS használata, valamint a megfelelő mentési stratégia megállapítása, de nem lehetetlen. Megpróbáltam a lehető legtöbb információt besűríteni a cikkembe, de természetesen van néhány információ ami kimaradt. Ha maradtak megválaszolatlan kérdések a kedves olvasóban, a Tech.Net levelezési listára, vagy közvetlenül nekem küldje el.

Harmath Zoltán
zoli@geniusgroup.hu
MCSE, MCP+i

Recovery Console

A Windows 2000 csökkentett módú *(safe-mode)* és Recovery Console rendszerindítási változatai több lehetőséget biztosítanak az el nem induló számítógép helyreállításához. A csökkentett mód lehetővé teszi, hogy a számítógép a működéshez minimálisan szükséges eszközmeghajtók és szolgáltatások betöltésével induljon el, a Recovery Console pedig a számítógép állapotától függetlenül mindig használható. Ebben az írásban ezeket ismertetjük, és néhány esettanulmány segítségével bemutatjuk használatukat.

Bevezető

Külső gyártó által készített szoftverek, vagy eszközmeghajtók telepítése elindíthatatlanná teheti a számítógépet. Erre példa lehet az, amikor a számítógép lefagy, egy „stop" hibaüzenetet kapunk, vagy nem vagyunk képesek bejelentkezni. Sok esetben a probléma megoldható egy szolgáltatás engedélyezésével vagy letiltásával, vagy az eszközmeghajtó lecserélésével.

A Microsoft Windows NT korábbi verzióinál nem lehetett egyszerűen, *(helyi vagy távoli)* bejelentkezés nélkül letiltani a szolgáltatásokat. A legtöbb esetben egy másik NT-t kellett az eredeti mellé telepíteni, hogy elérjük a meghibásodott rendszer regisztrációs adatbázisát vagy fájlrendszerét. Ezért telepítette sok rendszergazda az operációs rendszert FAT partícióra.

A Windows 2000-ben két olyan eszköz van, melynek segítségével hozzáférhetünk az el nem induló rendszerhez. Ezek a csökkentett mód, és a Recovery Console. A csökkentett mód az indítómenüből érhető el, és használatakor a Windows 2000 indításakor csak a minimálisan szükséges szolgáltatások indulnak el, így lehetővé téve például egy hibás eszközmeghajtó, vagy egy szolgáltatás eltávolítását.

A Recovery Console *(amit „Command Console"-nak vagy "Repair Console"-nak is szoktak hívni)* egy másik eszköz, ami az elindulni képtelen számítógép javítására, és a hibák felderítésére használható. A csökkentett móddal ellentétben a Recovery Console-nak nincs grafikus felülete, hanem parancssorra alapuló eszköz. A Recovery Console többek között használható a Chkdsk futtatására, egy hibás fájl lecserélésére, vagy akár egy szolgáltatás, vagy eszközmeghajtó letiltására is.

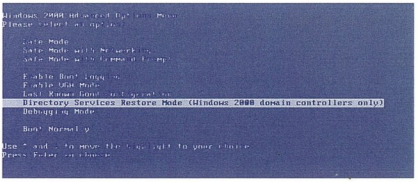
Csökkentett módú rendszerindítás

A csökkentett mód közvetlenül a karakteres üzemmódú bootoláskor, még a fekete képernyőn, az F8 megnyomása után érhető el: az Advanced Options menübe jutunk, ahol a következő lehetőségek közül választhatunk:

- Safe Mode *(csökkentett mód)*. Ennek választásakor az operációs rendszer csak a számítógép indításához elengedhetetlen eszközmeghajtókat *(például egér, billentyűzet)* tölti be.
- Safe Mode with Networking *(csökkentett mód hálózattal)*. Az előzőhöz hasonló, de az alapvető hálózati szolgáltatásokat is elindítja.
- Safe Mode with Command Prompt *(csökkentett mód parancssorral)*. Hasonló a Safe Mode-hoz, de a Windows Explorer nem indul el *(a Cmd.exe indul kezelőfelületként)*.
- Enable Boot Logging *(rendszerindítás közbeni naplózás engedélyezése)*. Ha ezt az opciót választjuk, létrejön a WindowsKönyvtár\Ntbtlog.txt naplófájl, melyből látható, hogy mely szolgáltatások és eszközmeghajtók indítása volt sikeres vagy sikertelen.
- Enable VGA Mode *(VGA mód engedélyezése)*. VGA módban indítja a számítógépet, ezzel lehetővé teszi az alapértelmezett megjelenítési beállítások megváltoztatását. Főleg akkor hasznos, ha a képernyő felbontását túl magasra állítottuk.
- Last Known Good Configuration *(rendszerindítás az utolsó ismert jól működő rendszerbeállítások alapján)*.
- Directory Services Restore Mode *(címtárszolgáltatások visszaállítása)*. Ez a tartományvezérlőkön használható indítási mód, ami lehetővé teszi az Active Directory adatbázisának javítását, vagy visszaállítását szalagos mentésről.
- Debugging Mode *(hibakeresési mód)*. Hibakeresési *(debug)* módban indítja a számítógépet. A hibakeresési információkat a COM2-re *(ez az alapértelmezett)*, vagy a COM1-re küldi *(ha nincs COM2 a számítógépen)*.
- Boot Normally *(normál rendszerindítás)*.

A Windows 2000 operációs rendszer indítója *(loader)* szabályozza az ehhez a menühöz való hozzáférést. Ha az indítómenü már eltűnt, még mindig van lehetőség a menü elérésére az F8 megnyomásával, de csak a színes Windows 2000 indítóképernyő megjelenése előtt. Egy menüpont kiválasztásakor a kért opció megjelenik a képernyő alján, de csak akkor, ha az indítómenün még nem jutottunk túl, különben a rendszer indítása folytatódik *(természetesen a kért mód használatával)*.

...most nyomjunk F8-at...



...az Advanced Options menü eléréséhez!

Ha ugyanazon a számítógépen használunk Windows 2000-et és Windows NT 4.0-t, az Advanced Options menü esetleg nem elérhető. Ennek az lehet az oka, hogy az NT 4.0-t a Windows 2000 telepítése után telepítettük. Ez esetben cse-

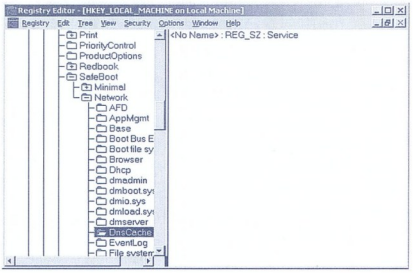
réljük le a Windows NT 4.0 indítófájljait *(az Ntldr-t és az Ntdetect.com-ot)* a Windows 2000-es verziókra.

Az elindított szolgáltatások listája

Csökkentett mód választása esetén csak a lehető legkevesebb szolgáltatás indul el. A választott opció alapján a következő regisztrációs adatbázis bejegyzésben szereplő szolgáltatások és eszközmeghajtók indulnak el:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
↳ Control\Safeboot
```

Az e bejegyzés alatt található Minimal és Network kulcsokban találhatók meg a Safe Mode és a Safe Mode with Networking opciókhoz tartozó szolgáltatások listái.



Ki gondolta volna, hogy elvileg átállíthatjuk, mi induljon az egyes Safe módokban? De azért ne babráljuk...

A Minimal kulcs alatt található azoknak a szolgáltatásoknak és eszközmeghajtóknak a listája, melyek a Safe Mode és a Safe Mode with Command Prompt opció választása esetén elindulnak. Ezek a következők:

Eszközmeghajtók

- Az adott géphez csatlakoztatott tárolóeszközök meghajtói *(CD-ROM és Jaz meghajtók, merevlemezek)*
- Beviteli eszközök *(billentyűzet, egér)*
- Alap videokártya meghajtó *(VGA)*
- Tárolóeszköz-vezérlők meghajtói *(IDE/SCSI vezérlők)*

Szolgáltatások

- Event log
- Logical Disk Manager
- Plug and Play
- Remote procedure call *(RPC)*

A Network kulcs alatt található azoknak a szolgáltatásoknak a listája, melyek a Safe Mode with Networking opció választása esetén indulnak el. Ha ezt az opciót választjuk, a következő eszközmeghajtók és szolgáltatások indulnak el:

Eszközmeghajtók

- A Minimalnál felsorolt eszközökön kívül a hálózati kártyák meghajtói *(ide értendők a például a PCMCIA eszközök is)*.

Szolgáltatások

- A Minimalnál felsorolt szolgáltatások
- Computer browser
- Dynamic Host Configuration Protocol *(DHCP)* client service
- Domain Name System *(DNS)* resolver cache
- Messenger
- Netlogon
- Server
- TCP/IP NetBIOS helper
- Workstation

Azoknak a szolgáltatásoknak a listáját, melyek mindegyik csökkentett módú indításkor elindulnak, nem szabad megváltoztatni. Lehet, hogy szeretnénk, ha egy adott szolgáltatás mindentől függetlenül elinduljon, és elérhető legyen, de lehet, hogy éppen ez a szolgáltatás okozza a problémát.

Környezeti változó

Ha bármely csökkentett módú indítást választjuk, a SAFEBOOT_OPTION környezeti változó jelzi azt a módot, amellyel a számítógép elindult. Ez a változó olyan programok futtatásakor hasznos, melyek a számítógép állapotától függő műveleteket hajtanak végre. Ez lehet például olyan jelentéskészítő eszköz, ami a számítógépen fut, és a változó értékétől függően a diagnosztikai információkat a számítógép merevlemezére, vagy a hálózaton előre meghatározott helyre menti. Az eszköz induláskor meghatározza a számítógép állapotát, és az információk mentéséhez szükséges megfelelő műveletet.

A SAFEBOOT_OPTION környezeti változó a következő értékeket veheti fel: Minimal, Network, vagy DsRepair. Ha a számítógépet a Safe Mode vagy Safe Mode with Command Prompt opcióval indítjuk, a SAFEBOOT_OPTION változó értéke a Minimal lesz. A SAFEBOOT_OPTION változó értéke csak a Windows 2000 Server vagy Windows 2000 Advanced Server tartományvezérlőkön lehet DsRepair.

Példák a használatra

1. példa

Tegyük fel, hogy veszünk egy új programot, vagy eszközt, amely szolgáltatásként telepítődik *(nevezzük AzÉnSzolgáltatásom-nak)*. Miután befejeztük a telepítést, és újraindítottuk a számítógépet, a "Stop 0x0000021a" hibaüzenetet látjuk a kék képernyőn amelynek leggyakoribb okozója egy usermódban futó folyamat, de az alábbit például szántszándékkal okoztam, így:

```
kill winlogon –f
```

Winlogon hiányában...

Mivel az utolsó változtatás az AzÉnSzolgáltatásom telepítése volt, megpróbáljuk csökkentett módban indítani a számítógépet. Nem meglepő módon elindul és be tudunk jelentkezni. Gyanús, hogy az AzÉnSzolgáltatásom okozta a hibát, ezért tiltsuk le az indítását a Computer Management használatával, és indítsuk újra a gépet *(most ne válasszuk a Safe Mode opciót)*. A számítógép elindul, és be tudunk jelentkezni. Távolítsuk el a programot az Add/Remove Programs eszköz segítségével.

2. példa

A csökkentett mód akkor is használható, ha bizonyos összetevők károsodtak, vagy véletlenül le lettek törölve. Például ha az Explorer.exe fájl károsodott, az asztal *(desktop)* nem jelenik

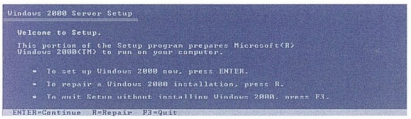
meg. Ez esetben indítsuk el a számítógépet, és válasszuk a Safe Mode with Command Prompt opciót az Advanced Options menüből. Ekkor a Cmd.exe-t használjuk kezelőfelületként az Explorer.exe helyett, és lecserélhetjük a hibás Explorer.exe-t egy új, hibátlan példányra.
Minden csökkentett mód opció kiválasztása esetén *(a Safe Mode with Command Prompt-nál is)* be kell jelentkezni, hogy hozzáférhessünk a számítógéphez. A Safe Mode with Networking kivételével a bejelentkezés az adott gépen levő SAM *(Security Accounts Manager)* adatbázis alapján történik.

3. példa

Van egy számítógépünk *(laptop)*, amelyet több irodában is használunk. Minden munkahelyen külön monitor van. A jelenlegi munkahelyünkön levő monitor működik magas képfrissítéssel és felbontással, de amikor átmegyünk egy másik irodába, az ottani monitor nem működik az előzőnél alkalmazott beállításokkal. Mivel nem látunk semmit a monitoron, nem tudunk bejelentkezni, és megváltoztatni a beállításokat. Ebben az esetben újra kell indítani a számítógépet, és az Enable VGA Mode-ot választani az Advanced Options menüből. Így a számítógép normálisan indul, de VGA felbontást használ, így meg tudjuk változtatni a beállításokat.

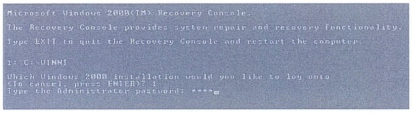
A Recovery Console

A Recovery Console arra használható, hogy hozzáférjünk a számítógéphez, amikor az egyáltalán nem hajlandó elindulni *(még a csökkentett módú opciók használatával sem)*. Az Administrator felhasználói fiók és jelszó használatával hozzáférhetünk a Windows 2000 rendszerfájlokhoz, futtathatunk néhány beépített diagnosztikai eszközt *(például Chkdsk)*, és korlátozottan bár, de hozzáférhetünk a regisztrációs adatbázishoz, hogy letilthassunk, vagy engedélyezhessünk szolgáltatásokat. Ezt FAT16, FAT32 vagy NTFS partíciókon tehetjük meg.
A Recovery Console futtatásához szükség van a Windows 2000 Professional, a Windows 2000 Server vagy a Windows 2000 Advanced Server CD-ROM-ra, és egy rendszerindításra képes CD-ROM meghajtóra. Azokon a számítógépeken, melyekben nincs az El Torito specifikációnak megfelelő CD-ROM meghajtó, a négy indítólemezt kell használni, vagy létre kell hozni azokat a CD-ROM-ról.
Indítólemezek létrehozásához készítsünk elő négy üres floppy lemezt, és futtassuk a Windows 2000 CD-ROM Bootdisk mappájában található Makeboot.bat fájlt.
Most indítsuk el a számítógépet. A CD-ROM, vagy az 1. indítólemez legyen a meghajtóban, és kövessük a képernyőn megjelenő utasításokat. Amikor a Welcome to Setup üzenet megjelenik...



...nyomjuk meg az R-t, hogy kiválasszuk a Repair a Windows 2000 Installation *(telepített Windows 2000 javítása)* opciót. Amikor a Windows 2000 Repair Options *(javítási opciók)* párbeszédablak megjelenik...

...nyomjuk meg a C-t. Ezzel elindítottuk a Recovery Console-le-t. Ha R-t ütöttünk volna, az Emergency Repair folyamatba kerültünk volna *(lásd később)*. A következő képernyőn a telepített operációs rendszerek listája látható. A szám leütésével kiválasztható a kívánt Windows 2000 rendszer.



Ezután gépeljük be a helyi Administrator jelszavát *(a nevét nem kell, a well-known-SID alapján választódik ki!)*, és nyomjuk meg az Enter-t. A Recovery Console-nál a helyi Administrator jelszó megadása szükséges a rendszert tartalmazó kötet eléréséhez. Három lehetőség van a helyes jelszó megadására, ha nem sikerül, a számítógép újraindul. Ha a gépen nem találhatók Windows 2000 rendszerfájlok, a parancssor automatikusan megjelenik.
A jelszó elfogadása esetén megjelenik a parancssor, és a választott rendszer SystemRoot mappájában kezdhetjük ténykedésünket *(például C:\Winnt)*.
A Recovery Console-lal általános hibaelhárítási tevékenységeket végezhetünk *(például a Chkdsk eszköz futtatása egy köteten, fájlok másolása CD-ROM-ról a merevlemezre)*, melyek segítségével a számítógép ismét indítható állapotba kerülhet. Csak befelé másolhatunk, kifelé *(a számítógépről lemezre vagy bármi másra)*, csak akkor, ha a biztonsági házirend ezt megengedi *(lásd később)*.
Nem kell mindig a CD-ROM-ot, vagy az indítólemezeket használni a Recovery Console indításához, ugyanis a Windows 2000 CD-ről telepíthető is. Gépeljük be az alábbi parancsot.

```
x:\i386\winnt32 /cmdcons
```

(ahol x a CD-ROM meghajtó, vagy a Windows 2000 telepítőkészletet tartalmazó megosztás betűjele).
Ez a parancs a Recovery Console futtatásához szükséges fájlokat a rendszert tartalmazó kötet gyökerében létrejövő Cmdcons rejtett mappába telepíti, és hozzáad egy bejegyzést a Boot.ini fájlhoz. Ezután az indítómenüben meg fog jelenni a Windows 2000 Recovery Console opció is az operációs rendszerek között. Kb. 7 MB helyet igényel.

A parancsok listája

Az alábbi parancsok használhatók a Recovery Console-ban:

Attrib	Delete	fixmbr	more
Batch	Dir	format	rd
Cd	Disable	Help	ren
Chdir	diskpart	listsvc	rename
Chkdsk	Enable	Logon	rmdir
Cls	Exit	Map	systemroot
Copy	Expand	Md	type
Del	Fixboot	Mkdir	set

A Recovery Console automatizált telepítése

Ahogy az előzőekben leírtuk, a Recovery Console futtatható a számítógép merevlemezéről is. Ha Windows 2000-et futtató gépeket helyezünk üzembe, és automatizáltan akarjuk telepíteni a Recovery Console-t, használjuk a következő parancsot:

```
winnt32.exe /cmdcons /unattend
```

Futtatható a Windows 2000 automatizált telepítése során is, *(a válaszfájl [RunOnce] részébe kell beírni)*, és használható a GUI módú telepítés során a Cmdlines.txt fájl használatával is. A válaszfájlról és a Cmdlines.txt fájlról további információ a Microsoft Windows 2000 Resource Kit-ben található.
Ha a Sysprep eszközt használjuk, hogy a számítógépeket lemezduplikálást használva helyezzük üzembe, csak akkor javasolt a Recovery Console telepítése a helyi merevlemezre, ha az összes telepítendő számítógépben ugyanolyan típusú és méretű merevlemez van, mint a forrásgépben, és ugyanazt a fájlrendszert is használják.

Házirend használata a Recovery Console biztonságosságának szabályozásához

A Recovery Console-ban sok olyan opció van, ami segíthet az el nem induló számítógép helyreállításában. Ezek közül van néhány olyan, melyek engedélyezhetők, vagy letilthatók, hogy a számítógép biztonságosabb, vagy könnyebben kezelhető legyen, amíg ebben a módban fut. Például a set parancs a következő paraméterek beállítására használható:

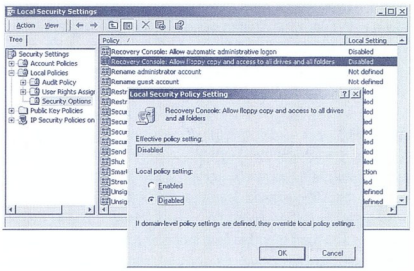
- AllowWildCards. bizonyos parancsoknál engedélyezi a helyettesítő karakterek használatát *(például del *.* parancs)*.
- AllowAllPaths. lehetővé teszi az összes útvonal elérését az összes köteten.
- AllowRemovableMedia. Lehetővé teszi, hogy a számítógépről floppy, vagy Jaz, vagy Zip lemezekre másoljunk fájlokat.
- NoCopyPrompt. Nem jelenik meg figyelmeztetés, ha egy fájlt felül akarunk írni.

Ezek a paraméterek alapértelmezésben „False"-ra *(nem engedélyezett)* vannak állítva és változtathatóságukat házirend szabályozza. Az alapértelmezett viselkedés az, hogy a rendszergazda automatikus bejelentkezése és a set parancs használata nem engedélyezett. Az alapértelmezett házirend megváltoztatásához szükséges lépéseket alább részletezzük.

Tartományhoz nem tartozó számítógépek

A tartományhoz nem tartozó gépeken a helyi biztonsági házirendet kell módosítani. A házirendet minden egyes gépen módosítani kell, ahol el akarunk térni az alapértelmezett beállításoktól. A helyi biztonsági házirend módosításához a következőket kell tennünk:

1. Start menü -> Settings -> Control Panel -> Administrative Tools -> Local Security Policy
2. Amikor a Local Security Settings elindul, válasszuk az alábbi mappákat:
   - Local Policies
   - Security Options
3. A beépülő modul jobb ablakában keressük meg a következő két házirendet:
   - Recovery Console: Allow automatic administrative logon
   - Recovery Console: Allow floppy copy and access to all drives and all folders
4. Alapértelmezésben ezek a házirendek le vannak tiltva. Engedélyezésükhöz kattintsunk a jobb gombbal a házirend leírására, majd kattintsunk a Securityra.
5. Amikor a Local Security Policy Setting párbeszédablak megjelenik, kattintsunk az Enabled-re, majd az OK-ra.



A Recovery Console biztonsági beállításait még jóval a katasztrófa bekövetkezte előtt gondoljuk át!

Tartományhoz tartozó számítógépek

A fent bemutatott lépések ugyanúgy alkalmazhatók a tartományi számítógépeken is, de ha a tartomány Windows 2000 alapú, a tartományi házirend felülbírálja az összes helyi házirendet. Ez esetben javasolt tartomány-szintű házirend használata *(ennél is alkalmazhatók a fent bemutatott lépések)*.

A Recovery Console használata tartományvezérlőkön

A Recovery Console Windows 2000-et futtató tartományvezérlőkön is használható. Minden, amit ebben a részben leírtunk, érvényes a tartományvezérlőkre is, egy apróság kivételével. Az Administrator jelszó, ami a Recovery Console elindításához használható, az lesz, ami a számítógép tartományvezérlővé válásakor volt *(SAM jelszó)*. Ezután ez a jelszó már nem fog megváltozni még akkor sem, ha közben megváltoztatjuk az Administrator jelszót.

Példák a használatra

1. példa

Egy frissített eszközmeghajtót telepítünk *(Driver1.sys)* a tárolóeszköz-vezérlőhöz, amit egy neves gyártótól *(OEM)* vettünk. Ez a frissítés lecseréli a Microsoft által szállított eszközmeghajtót, ami a számítógép telepítésével egyidejűleg települt. Amikor újraindítjuk a számítógépet, egy „Stop 0x7b" hibaüzenetet kapunk *(inaccessible boot device – a rendszer indításához szükséges eszköz nem elérhető)*. Amikor megpróbáljuk a rendszert csökkentett módban indítani, ugyanezt a hibaüzenetet kapjuk.
A hiba elemzése során látjuk, hogy az OEM által készített eszközmeghajtó nem működik jól az adott vezérlőkártyával. Használjuk a Windows 2000 CD-ROM-ot a Recovery Console elindításához. Miután bejelentkeztünk, keressük meg az OEM által készített eszközmeghajtót, és cseréljük le az eredeti Microsoftos verzióra. Ezután már el tudjuk indítani a számítógépet.

A Recovery Console használatakor a Windows 2000 CD-ROM-ról másolt fájlokat nem kell „kicsomagolni" *(expand)*, mert ez automatikusan megtörténik.

2. példa

Egy Windows 2000 Serveren szoftveres RAID 1 *(tükrözés)* megoldást alkalmazunk a rendszerpartíció tükrözésére. A Disk 0 merevlemez meghibásodik, a rendszer pedig tovább működik a Disk 1-ről. A rendszergazda kicseréli a meghibásodott merevlemezt egy újra. A számítógép az újraindításkor nem tud Windows 2000-ret indítani, mert az operációs rendszernek a tükrözött lemezről *(disk 1)* kellene elindulnia, de a boot.ini-ben az ARC *(Advanced RISC Computing)* elérési út az új merevlemezre *(disk 0)* mutat.

Mivel nem tudjuk, hogy milyen ARC elérési út szükséges, indítsuk el a Windows 2000 CD-ROM-ról a Recovery Console-t. Ezután a következő parancs segítségével megtudhatjuk a helyes ARC elérési utat:

```
map arc
```

Ez a parancs megmutatja az összes merevlemez és összes partíció elérési útját.

3. példa

A számítógépünk nem indul, ezért hívunk egy nálunk okosabb embert. A szakember magával hoz egy Windows 2000 CD-ROM-ot, melynek segítségével elindítja a Recovery Console-t, és egy lemezt, amelyen egy Recover.txt nevű fájl található. Miután elindítja a Recovery Console-t, a hibaelhárítás lépései automatizálhatók kötegelt *(batch)* parancsok segítségével. Emberünk ismeri a batch parancsot, mely így működik:

```
batch a:\recover.txt
```

Tulajdonképpen tetszőleges nevű és kiterjesztésű TXT fájlban összegyűjthetjük a végrehajtandó parancsokat, a batch parancs nem kényes a nevekre. A mi emberünk textfájlja az alábbi sorokat tartalmazza:

```
Set allowRemovableMedia = True
Set NocopyPrompt = True
Fixboot c:
FixMbr
Chkdsk c:
Attrib –r c:\ntldr
Attrib –r c:\ntdetect.com
Copy d:\i386\ntldr c:\ntldr
Copy d:\i386\ntdetect.com c:\ntdetect.com
Attrib +r c:\ntldr
Attrib +r c:\ntdetect.com
```

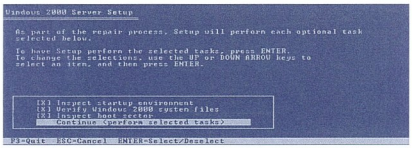
Kijavítja a boot sectort és az MBR-t *(Master Boot Record)*, lefuttatja a chkdsk parancsot a rendszerpartíción, új rendszerindító fájlokat *(Ntldr és Ntdetect.com)* másol a számítógépre, és a megfelelő attribútumokat is beállítja. Igazán leleményes fickó!

4. példa

A gép nem indul, mert egy utólagos NT 3.51 telepítés *(sic!)* felülírta NTDETECT.COM-ot és az NTLDR-t. Be is másolhatnánk a helyes verziókat, de mi a Recovery Console beépített parancsaira bízzuk magunkat.

```
C:\fixboot
```

Ez remekül újraírja a bootszektort, de – mint újraindításkor kiderül – nem cseréli vissza a fenti két fájlt. Ekkor az Emergency Repair megoldás mellett döntünk. Rendszertöltés CD-ről, press R, press R again, majd press M *(manual repair)*, s a következő lehetőségek tárulnak elénk:



Totózzunk! Vajon melyik javítja ki a fenti két fájlt?

- Az Inspect Startup Environment nevével ellentétben nem foglalkozik az indításhoz nélkülözhetetlen fájlokkal *(akkor mit csinál??)*
- Az Inspect Boot Sector azonos hatású a Recovery Console fixboot parancsával – azaz szintén nem törődik a fájlokkal
- A Verify System Files pedig ugyebár a WINNT könyvtárra vonatkozik

Tehát egyik sem? Nos, ha hiszünk a feliratoknak, és egyesével, logikus sorrendben kipróbáljuk a fenti opciókat, kiderül, hogy mégiscsak a Verify System Files segít rajtunk. Végül press L, mivel nincs a birtokunkban ERD *(Emergency Repair Disk flopilemez)*, majd press Enter, és beindul a reszelés. Amikor látjuk, hogy a Verify System Files nekiáll a WINNT könyvtár reparálásának, press F3 *(=Quit)*.

Fóti Marcell
marcellf@netacademia.net

Hash!

Most egy igazán haszontalan algoritmusokról írok önöknek: olyasmiről, ami kizárólag az adatok összekavarására és széttrancsírozására alkalmas: a hash, vagy magyarul tördelőalgoritmusokról. Egy biztonsággal foglalkozó könyvben olvastam azt a hasonlatot, hogy a hash algoritmus olyan, mint a húsdaráló, a beléhelyezett adat széttrancsírozódik. A folyamat egyirányú: nyúlból könnyedén készíthető fasírt, de fasírtból nyúl...? *(A hash szó jelentése szó szerinti fordításban: darálék, daralthús.)*

Egy hash algoritmus blokkdiagramja :-)

Mindennapjaink adatainak hasznosságához nem fér kétség; dokumentumaink, adatbázisaink, táblázataink, jelszavaink nélkül lehúzhatnánk a rolót. De vajon mi a csudára jó egy word dokumentum, vagy egy jelszó fasírtja?

Ha ez a fasírt közönséges, felismerhetetlen hústrutymó lenne, bizony nem lehetne értelmesen felhasználni, azonban a hashalgoritmusok kimenete olyan darálékot ad, mely

- *(jobbára)* egyértelműen utal az eredeti adatokra, de legalább az a feltétel teljesül, hogy egy adott dokumentumból mindig ugyanaz a hash állítható elő.
- minden megismételt futásra azonos. Az algoritmus viselkedése determinisztikus. A megismételt futásnak különös jelentősége van elosztott környezetekben, ahol ugyanaz a hash algoritmus a hálózat különböző gépein, egymástól függetlenül futkos.
- nem teszi lehetővé az eredeti adatok előállítását. Fasírtból ne lehessen nyulat alkotni!

Időről időre felröppennek hírek bizonyos algoritmusok feltöréséről, s ilyenkor mindig titkosítások megtörésére gondolunk, pedig egy hash algoritmus feltörése legalább olyan izgalmas feladat. Ha a fasírtból visszaállítható a nyúl, a hash feltörtnek tekinthető.

Mire jó?

Ahogy a fasírt univerzális étel, úgy a hash eredménye is igen sokoldalúan hasznosítható. Az alábbi néhány példa rávilágít a méltatlanul lenézett darált hús informatikai felhasználásának sokoldalúságára:

1. Jelentős szerepet kap hálózati bejelentkezéseinknél, hogy jelszavainkat ne kelljen olvasható formában átvinni a hálózaton. Ilyenkor titkosítás helyett használjuk, mert megfelelően ravasz módon felhasználva ugyanolyan biztonságos jelszóátvitelt tesz lehetővé, mint egy titkosítóalgoritmus, és még csak kulcscsereberére, vagy biztonsági tanúsítványra *(Certificate)* sincs szükség.
2. Napjaink oly divatos technológiája, a digitális aláírás sem létezne megfelelő hashalgoritmus nélkül. Itt az algoritmus visszacsinálhatatlanságát aknázzuk ki.
3. Minél olcsóbb a RAM, annál több van belőle, s minél többet használunk, annál több adatunk kerül gyorstárolókba *(cache)*. A hashalgoritmusok döbbenetes módon szerepet kapnak a cachememóriák kezelésében – nélkülük huszadakkora teljesítményre lennénk képesek.
4. Még az adatbáziskezelők is profitálnak a fasírtból. Az SQL Server hash-joint használ hatalmas és rendezetlen táblák közötti kapcsolatok *(join)* megvalósításánál.

Remélem sikerült ha érdeklődést nem is, de legalább gyanakvást kelteni a hashalgoritmusok iránt, így áttérhetünk a konkrétumokra.

1. felhasználás: autentikáció

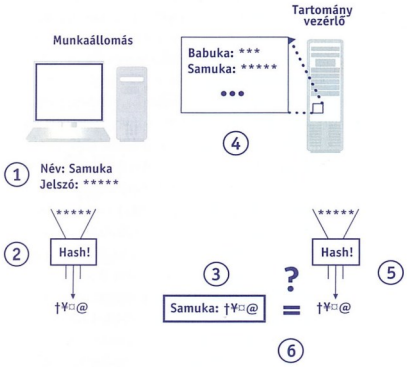
Napjaink ezergépes vállalati hálózatainál nélkülözhetetlen, hogy minden erőforrás felhasználását személyre szabottan tudjuk beállítani, engedélyezni és tiltani. Ehhez nincs is másra szükség, mint a hálózat számítógépei előtt üldögélő felhasználók egyszerű és egyértelmű azonosítására. Sokféle módszer szóbajöhet, de manapság egyértelműen a név+jelszó típusú bejelentkezési forma a legelterjedtebb, mivel némi kényelmetlenség árán megfelelően stabil felismerési lehetőséget biztosít – amíg a jelszavak nem kerülnek közkézre. Ez ellen nyilvánvalóan úgy védekezhetünk, hogy nagy ívben elkerüljük a titkosítatlan jelszóátvitel összes formáját, s lemondunk többek között az FTP-ről...

Titkosítsunk!

Mivel? Hát pl. 3DES-sel, mert az jó erős! Ha a titkosított átvitel mellett döntünk, felmerül egy valóban bosszantó probléma: mivel napjaink összes populáris titkosítóalgoritmusának *(DES, 3DES)* minden részlete, sőt forráskódja is ismert, kénytelenek volnánk a titkosítókulcsok rejtegetésével biztosítani az illetéktelenek távoltartását, hisz ha nem így tennénk, saját 3DES-ükkel olyan szépen visszafejtenék a jelszót, hogy ihaj!

A bejelentkezés megkezdése előtt tehát „titokban" el kellene juttatni a munkaállomásokra a megfelelő *(random)*, egyszerhasználatos 3DES titkosítási kulcsot. Igen ám, de hogyan? Hálózaton? Ahhoz előbb el kellene juttatni egy másik titkos kulcsot, aminek titkos eljuttatásához egy harmadik kulcs kellene, amit titkosítva viszont csak egy negyedik kulccsal juttathatnánk át, amihez kellene egy ötödik... Milyen jó is lenne, ha a kezdő 3DES kulcsot valahogy úgy oda lehetne varázsolni a munkaállomásokra, hogy Hacker Henry és Claudia Sniffer ne kaphassa el! Mondjuk postagalambbal? Ejnye, a szimmetrikus titkosítóalgoritmusok úgy tűnik felsültek!

És ekkor egy hirtelen ötlettel eszünkbe jutnak a szintén publikált, közismert *(MD4, MD5, SHA)* hashalgoritmusok. Hogyan lehetne ezekkel dolgozni? Az ötlet alapja az a tény, hogy a hashalgoritmusok egyirányúak. Egy jelszóhash-t *(fasírt)* minden további nélkül kirakhatunk a hálókábelre, mert ha ezt valaki elkapja, maximum mégegyszer ledarálhatja, hogy finomabb legyen – de nyuszimuszi az életben nem lesz belőle.



Jelszóellenőrzés bejelentkezéskor

A jelszókiértékelés menete a következő: ① a munkaállomás bekéri a felhasználó jelszavát, és ② ráereszti a maga hashalgoritmusát. A végeredményt a felhasználónévvel együtt elküldi a tartományvezérlőnek ③, aki szintén nem tud fasírtból nyulat eszkábálni, de neki is van egy nyula! Megvan nála ugyanis a jüzer éppen érvényes jelszava ④. Ledarálja ⑤, s a nyúl borzalmas földi maradványait összehasonlítja ⑥ a hálózaton át megkapott fasírttal. Ha a kettő egyezik, a két nyúl is egyforma volt. Fantasztikusan egyszerű ugye? És ráadásul feltörhetetlen...?!

És a LOpthCrack?

Hát igen. Az **[1]** címről letölthető LOpthCrack *(ejtsd: luftkrekk)* már évek óta tudja mindazt, ami lehetetlen: hashalgoritmussal titkosított jelszavakat „visszafejteni". Valóban megmondja a hálózaton látott hessek alapján az eredeti jelszót, de NEM visszafejtéssel, hanem sok-sok nyúl egymás utáni ledarálásával, másképpen próbálkozással. *(Mégpedig kétféle módon: szótárral, majd ha az kimerült, szisztematikusan végigpróbálgatva a lehetséges kombinációkat. Ez utóbbit brute force módszernek hívják. A* **[2]** *címről letölthető törőszótár a magyar nyelvhez készült, a weben leggyakrabban előforduló szavak szerepelnek benne.)* S ami tíz évvel ezelőtt merő fikció volt, az öt éve már bizonyítottan – bár lassan – működött, ma pedig száguld. Az egyre nagyobb teljesítményű gépek nyilvánvalóan egyre több nyulat darálnak le egységnyi idő alatt, így a jelszófeltörés is egyre könnyebb feladat. A régi típusú OS2/Windowsos *(LanMan)* jelszavak könnyű feltörhetőségéhez azonban még egy fontos dolog hozzájárult.

Security by obscurity

Avagy a ködösítéses titkosítás.

Ha valaki titkosítási algoritmus bevezetésén gondolkodik, ízzó vassal kergesse el azokat a sarlatánokat, akik saját fejlesztésű, hipererős megoldásokkal jelentkeznek, de nem árulják el az erősség mibenlétét. Ezek az alakok az esetek 108%-ában egyszerűen túl ostobák ahhoz, hogy megértsék a meglévő algoritmusokat, ezért hekkelnek valami ócskaságot, s azt hiszik, hogy mivel senki nem ismeri az algoritmust *(XOR, eltolva az ábécében, sőt, szorozva a gondolt szám háromszorosával és hasonló gyikságok)*, tákolmányuk mindjárt megfejthetetlen is. Sokszor meglévő algoritmusokon „tikosítanak" még egyet, ami a butaság legmagasabb foka. Ezek azok az algoritmusok, melyek pofonegyszerű kriptoanalízissel *(gyakorisságfigyelés stb.)* úgy nyílnak, mint a sufniajtó a szélben. Azt hihetnénk, hogy ebbe a csapdába nagy-nagy cégek nem sétálnak bele. Nos, ez nem így van. Az IBM-Microsoft duó által kifejlesztett LanMan hashalgoritmus egyik erősségét az adta, hogy nem publikálták az algoritmust. Ez az erő hónapokig tartott, utána egyszerűen visszafejtette a kódot egy lelkes egyetemista, s kitette az Internetre. Ennek nyolc éve. Ma már a Windows a szabványos Kerberos autentikációs protokollt használja, de Lan Managerek, OS/2-k és Windowsok generációinak kipusztulását kell kívárni, hogy mindenünnen eltűnjön a LanMan *(és az NTLM)*.

Kitérő: a jelszóalapú bejelentkezés halála?

A fenti gondolatmenetből következően *(egyre gyorsabb gépek, egyre rövidebb brute force)* egyre közelebb érünk ahhoz a jövőbeni pillanathoz, amikor a jelszóalapú hitelesítésnek végleg befellegzik. Küzdhetünk a gépek ellen, de sajnos az egyetlen megoldás, ha a jelszavak egyre bonyolultabbá válnak. Egy jelszó legyen

- hosszú
- bonyolult és
- könnyen megjegyezhető

Ez a három szempont körülbelül annyira elégíthető ki egyszerre, mint amit egy kőművessel szemben támasztunk *(munkája legyen gyors, olcsó és jó minőségű)*.

Hiszen ha egy jelszó hosszú és bonyolult, akkor nem könnyű megjegyezni: kiírjuk a monitorra. Ha hosszú és megjegyezhető, akkor az emberi gyarlóság miatt nem bonyolult stb. Kiutat a SmartCard logon jelent, ahol a „jelszó" a nyilvános kulcsunk, ami minimum 512 bit. Az emberi jelszavakhoz képest végtelenül hosszú, határtalanul bonyolult, és meg sem kell jegyezni!

2. felhasználás: digitális aláírás

Amit tudni akartál a digitális aláírásról, de nem merted megkérdezni: mi teszi lehetővé, hogy egy dokumentumról teljes bizonyossággal kijelenthetjük, hogy X.Y. írta, és azt senki meg nem módosította? Hát bizony a hash. Egészen pontosan az aláíró privát kulcsával titkosított hash. *(Lám, hamarosan meg kell írnom az RSA algoritmust is.)*

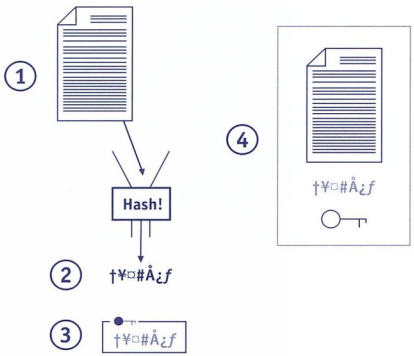
Mit tud biztosítani a digitális aláírás? Pontosan azt, amit a hagyományos szignó: a dokumentumot bárki elolvashatja, de a rendelkezésére álló erőforrásokkal és időkerettel „gazdálkodva" képtelen lesz meghamisítani. Ennek működése nagy vonalakban a következő *(deszkamodell)*:

1. végy egy elektronikus dokumentumot

2. titkosítsd saját privát kulcsoddal
3. küldd el mindenkinek, de tedd mellé a publikus kulcsodat

Minden címzett képes lesz ezek után a mellékelt publikus kulcs segítségével dekódolni és elolvasni az üzenetet, de egyikük sem lesz képes elolvasás után módosítani ÉS újra visszakódolni, mivel a privát kulcs mindvégig nálad van, volt és lesz.

Deszkamodellünk óriási hátránya, hogy nem hűen tükrözi a valóságot. Ennek oka a nyílt kulcsú titkosításban keresendő: végtelenül lassan lenne képes akárcsak egy néhány tíz kilobájtos dokumentumot titkosítani *(mert a titkosítás egyik lépése a dokumentum, mint irdatlan hosszú bináris szám felemelése a „kulcsadik" hatványra)*. Kellene ide egy olyan adatka, mely kicsi és aranyos *(villámgyorsan végez vele az RSA)*, ám egyértelműen összefüggésbe hozható az eredeti dokumentummal. Na mi ez a számocska? A dokumentumból képezett hash! Így a digitális aláírásképpzés valósághű modellje a következő:



Digitális aláírás képzése

1. végy egy elektronikus dokumentumot
2. képezz belőle egy csinos kis hash-t
3. a hash-t titkosítsd saját privát kulcsoddal
4. a következőket csomagold egybe, és küldd el mindenkinek
   a. a dokumentumot
   b. a hash-t
   c. a publikus kulcsodat

Az aláírás valódiságának ellenőrzése pedig így történik:

1. vedd ki a csomagból a dokumentumot
2. képezz belőle hash-t
3. vedd ki a csomagból a publikus kulcsot és a titokhash-t
4. nyisd ki a titokhash-t a publikus kulccsal
5. hasonlítsd össze az általad a második lépésben készített hashértéket azzal, amit épp az imént dekódoltál
6. ha a két érték egyezik, a dokumentum érintetlen

3. felhasználás: cache

A cache magyarul gyorstítótár. Vagyis célja valamilyen művelet gyorsítása. Triviálisan hangzik, de a cél eléréséhez sokat kell gondolkodni! Minél nagyobb ugyanis a gyorsítótár, annál lehetetlenebb benne villámgyorsan megtalálni, hogy egy igényelt objektum benne van-e, vagy mégiscsak utána kell szaladni az eredeti tárolóhelyére. Szándékosan nem nevezem nevén az eredeti tárolóhelyet, mert a RAM, mint lassú tároló gyorstárazása is bőven találunk példát a processzorok ilyen-olyan gyorstárai esetében.

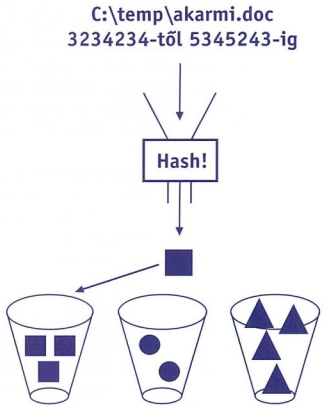
Az lenne az ideális, ha nulla órjelciklus alatt megállapítható lenne, hogy amit keresünk, az „kesselve" van-e már, ehhez viszont arra lenne szükségünk, hogy a cache minél intelligensebben meg tudja mondani, mi van benne. Evilági példával élve: keressük, hogy melyik nebuló rakott rágógumit a tanítónéni székére. Néhány csibész esetén majdnem mindegy, hogy önként jelentkezik-e a bűnös, vagy egyesével ki kell faggatnunk őket, de ha történetesen negyvenezer rosszcsontunk van, én az önkéntes megoldásra szavaznék: Pistike igenis álljon fel!

Az önkéntes megoldást asszociatív memóriával érhetjük el, melynek működése a becsületes csibészek gyülekezetéhez hasonló: egyszer kell feltenni a kérdést, s a következő pillanatban Pistike már fel is áll. Egyszerűen nem kell végigkeresni a halmazt! Micsoda fantasztikus memória! Vajon miért nem ilyet használunk mindenhol? Mit vacakolunk mindenféle keresési algoritmusokkal? Mert - finoman szólva - nem olcsó mulatság. Ahol azonban a keresgélés rombadöntené a gyorsítótár teljesítménynövelő hatását, ott igenis találkozunk vele, a Pentium processzorok Translation Look-aside Buffer-e például asszociatív memóriát használ. Erről a múltkori mesterkurzuson a memóriakezeléssel kapcsolatban tettem említést.

Amikor azonban egyszerű merevlemezadatokat „kesselünk" , akkor olcsó, ezerforintos RAM-ban gondolkodunk, amely sajnos egyet jelent azzal, hogy Pistikének esze ágában sincs bevallani tettét, és önként jelentkezni. Keresgélnünk kell. Egyszerűsítené a sok kis taknyos kivallatását, ha le tudnánk szűkíteni valahogy a kört azokra az ördögfiókákra, akik biztosan képesek lennének elkövetni a rágógumis csínyt.

(Ettől a ponttól kezdve a hasonlatom kissé kirekesztővé válik, bármely vélt hasonlóság ismert személyekkel kizárólag a véletlen műve! Ne feledjük: nem is rakott senki rágót a tanítónéni székére!) Alkossunk halmazokat: a világért sem tennének ilyet a szemüvegesek, és a lányok, azonban a kócos hajú és a szeplős fiúk gyanúsak nekem! Vallasuk ki először a szeplősöket, s ha egyikük sem vállalja, akkor a kócosokat. Én úgy gondolom, hogy már a szeplősök megríkatása is eredménnyel járna, a többiek akár haza is mehetnek.

IT terminológiával a történet így szól: van egy objektumigény *(blokk olvasása egy fájlból)*, rendelkezésre áll az objektumkérést egyedivé tevő adathalmaz *(fájlnév, beolvasási pozíció)*, s ezek alapján meg kell állapítanunk, hogy a gyorsítótárban ugyanebből a fájból ugyanez a darabka megtalálható-e. A gyorsítótár minden eleme természetes módon magán viseli saját egyedi azonosítóját *(fájlnév+pozíció)*, és ez alapján gyorsan meg is lehetne találni – ha nem lenne negyvenezer ilyen elem a tárban. Képzzünk belőlük csoportokat! Itt egy olyan hash algoritmus segít, amelyik nem egyedi értékeket ad, hanem a bemenő adatok egy bizonyos csoportjára mindig ugyanazt *(szemüvegesek)*, más csoportokra mást *(lányok)*. *(Egy ilyen algoritmus például a modulo 5, mely minden öttel osztható számot a nullás vödörbe rak, az egy maradék árán oszthatóakat az egyes vödörbe stb.)* Valójában ez a fajta csoportosítás nem a keresés pillanatában történik meg, hanem az elem már eleve a maga csoportjába kerül, mikor bekerül a gyorsítótárba. A csoportok neve: hash bucket, azaz fasírtosvödör.



Hash buckets

Amikor egy új kérés kiszolgálása előtt el kell dönteni, hogy egy kért elem hol van, rá kell ereszteni a kért adat azonosítójára a hash algoritmust, hogy kiderüljön a hovatartozása. Ha ez megvan, igen rövid idő alatt eldönthető, hogy bent van-e a kiválasztott vödörben, vagy nincs.

4. felhasználás: hash join

Ez a felhasználási mód igen hasonlatos a harmadikhoz, vagyis gigantikus, ámde rendezetlen halmazokban gyorsítja a keresést. Ám látom kollégáimat integetni itt a stúdióban az üvegfalon túlról, hogy túlléptem az időkeretet, így most áttérünk az ismertebb hashalgoritmusok elemzésére. Ígérem, visszatérek az SQL Server hash joinjaira, de ezt egy külön cikkben, a join stratégiák elemzésében teszem meg – valamikor még ebben az évben.

Elterjedt hashalgoritmusok

Az alábbiakban néhány különösen elterjedt hashalgoritmus működéséről lebbentem le a fátylat. Hihetetlen, de igaz: egy jól megtervezett hashalgoritmus egyszerre primitív *(gyors)* és hatékony. Úgy darál, hogy azt senki utána nem csinálja! Ez alól talán csak a LanMan kivétel, mely kizárólag primitív, de egyáltalán nem hatékony.

LanMan és NTLM

Kezdjük a sort a sokat szenvedett LanMan, NTLM párossal. Párban, kézenfogva járnak a hálózaton *(amíg le nem tiltjuk őket)*, elvileg azért, hogy a hitelesítő kiszolgáló kiválaszthassa az általa ismert hash-t a bejelentkezésből. Ennek az a gyakorlati hackerhaszna, hogy a két, egyenként is gyenge hash egymást árulja el. Ami nem deríthető ki az egyikből, megmondja a másik és vica versa. A security by obscurity tökéletes mintapéldányai ők, akik az algoritmus napvilágra kerülésével pillanatok alatt nevetségessé váltak. Hogy miért? Mert például a nagy hekkelésben belőjük fejlesztődött egy olyan megoldás, ami ügyesen meglékeli a hosszú jelszavak által elérhető biztonságnövekedést azáltal, hogy azokat hétkarakteres csoportokra bontja, s különkülön végzi el a „titkosítást" *(a jelszavak maximális hossza 14 karakter)*. Így nem meglepő, hogy egy nyolckarakteres jelszó semmivel sem biztonságosabb egy hétkarakteresnél, hisz a nyolcadik karakter önálló csoportot alkot, külön „titkosul". Ez meg is magyarázza, hogy a LOphtCrack miért olyan fura módon fejti meg a jelszavakat, hogy a vége *(a hetediken túli rész)* már rég megvan, az elejével meg még mindig 4 óra munkája van hátra. Szintén nem válik dicsőségére e párosnak, hogy az üres jelszavak semmilyen ellenállást sem tanúsítanak, azonnal „visszafejthetők". Ez pedig annak a következménye, hogy az öregebbik, a LanMan a hash lefuttatása előtt minden jelszót 14 karakteresre kerekít, legyen az bármilyen hosszú. A LanMan lépsei:

- a jelszó nagybetűssé konvertálása *(sic!)*
- kiegészítése szóközökkel, hogy 14 bájtos legyen
- a 14 bájtból 2 db DES kulcs készül *(14 bájt = 112 bit = 2*56 bit)*
- a két 56 bites DES kulccsal titkosítjuk a 0x4B47532140232425 „mágikus" számot *(sic!)*
- az eredmény 2*8 bájtnyi, összesen 16 bájtos „hash"

Hogy ez mitől hash, amikor DES? Ne kérdezzék. Security by obscurity és pont.

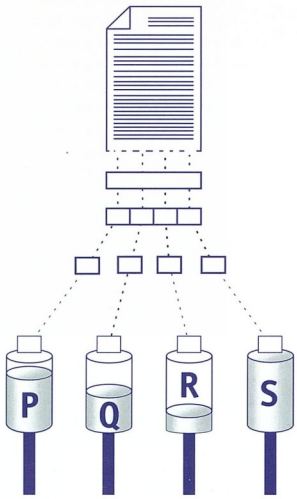
A fenti lépéssorozat folyománya, hogy ha a jelszó rövidebb, mint 8 karakter, a 7-14 bájtok mindig 0xAAD3B435B51404EE-re „hesselődnek". Az üres jelszó „titkosított" változata megegyezik hét darab szóköz titkosított kimenetével.

A LanMan hash tehát „megadja" Claudia Sniffernek a jelszavak nagybetűs változatát, s ezeket rápróbálva az NTLM-re *(MD4 alapú)*, pillanatok alatt megvan az eredeti jelszó. Mindenki sürgősen térjen át Windows 2000-re, vagy legalábbis tiltsa le a LanMant, s az NTLM-ből is az új változatot *(NTLMv2)* használja!

(Az NTLMv2 egyelőre állja az ostromot (MD4 helyett HMAC-MD5-öt használ), beállításáról terjedelmes cikkünk jelent meg tavaly novemberben.)

MD5 *(Message Digest 5)*

Az MD5 algoritmus társai közül gyorsaságával és hatékonyságával tűnik ki *(bár egy fokkal lassabb feltört elődjénél, az MD4-nél)*. Az RSA társaság készítette *(Rivest, Shamir, Adleman matematikusok)*. Ha elárulom a működését *(és miért ne tenném, RFC 1321, a* **[3]** *címen olvasható, s még C nyelvű forráskód is van hozzá)* nem fogjuk elhinni, hogy ez így bármire is jó, s valóban egyedi értékeket ont magából. A futási sebesség megköveteli, hogy ne legyen benne semmi rendkívüli. Van négy „dugattyúja", melyek 32 bites számokon végeznek primitív, fél órajeles műveleteket, s ezeken mintegy áttoljuk a trancsírozandó adatot. *(Persze a fél órajeles primitív függvények nagyon is bonyolult matematika eredményei.)*



Az MD5 algoritmus működés közben

A 32 bites egyszerű műveletek iszonyatos sebességet kölcsönöznek az algoritmusnak, a négy „munkahenger" pedig – több processzoros rendszerben – a valódi párhuzamos végrehajtás lehetőségét teremti meg. Anélkül, hogy részletezném a teljes működést, tekintsük meg a dugattyúk feladatát:

- P = F(X,Y,Z) = XY v not(X) Z
- Q = G(X,Y,Z) = XZ v Y not(Z)
- R = H(X,Y,Z) = X xor Y xor Z
- S = I(X,Y,Z) = Y xor (X v not(Z))

A dugattyúk összevárnak három 32 bites számot, és akkor robban a keverék. A figyelmes szemlélődő észreveheti, hogy F() nem más, mint bitenkénti if *(If X then Y else Z)*, míg H() egy paritásfüggvény a három bemenő adatra.

A feldolgozás végeredménye mindig egy 128 bites szám. Az MD5 *(egyelőre)* nincs feltörve, azaz még soha senkinek sem sikerült két olyan különböző doksit felmutatnia, amelyeknek az MD5 hash-e megegyezett volna. Ez nem is csoda, hisz ha a 128 bitet egyszerűen sorszámozásra használnánk, segítségével a föld minden négyzetcentiméterére tízezer egyedi dokumentum lenne elhelyezhető *(lásd még: GUID)*!

SHA-1 *(Secure Hash Algorithm)*

„Lapzárta után" (1997-ben :-) érkezett a hír: az MD5 mégiscsak rogyadozik: „MD5 has been recently shown to be vulnerable to collision search attacks [Dobb]." Kell hát valami új, valami megbízhatóbb. „SHA-1 appears to be a cryptographically stronger function." *(A két idézet a HMAC RFC-ből való (RFC 2104,* **[4]** *cím)*, az SHA RFC-je pedig a *3174-es, és az* **[5]** *címen olvasható)*.

Az SHA algoritmus is az MD4 utódjaként született, és bármilyen bemenő adatra 160 bites outputot szolgáltat. Szerzői közt szerepel a Cisco társaság. „Kéthengeres", de itt már nem négy, hanem nyolcvan *(80!)* függvény dolgozik úgy, hogy a hengerekben menet közben cserélünk dugattyút.

Salt

A hashalgoritmusok egyik erőssége, hogy ugyanazokon az adatokon végrehajtva ugyanazt a kimenetet produkálják. Ez néha - különösen titkosítási felhasználáskor - visszafelé sülhet el, hisz a gyakran ismételt minták *(például az üres jelszó hash-e)* könnyedén felismerhetővé válnak. Ezt elkerülendő egyes hash *(emlékezzünk: fasírt!)* algoritmusok lehetővé teszik, hogy paraméterezzük őket: ízlés szerit megfűszerezhetjük az outputot. A paraméterezés évszázados ötlet, az NTLM challenge/response azonosításnál előszeretettel használják. A 8 bájtos challenge valójában nem más, mint salt, melyet az NTLM algoritmus futása során a jelszóhoz keverünk. Az egyik legismertebb salt a HMAC algoritmus, melyet úgy terveztek, hogy a meglévő és bevált algoritmusokkal változtatás nélkül együttműködjön. Így jön létre a HMAC-MD5, HMAC-SHA-1, melyek az eredeti algoritmusok sózott változatai. A nevekből talán kikövetkeztethető, hogy a HMAC előfeldolgozást végez az adatokon, s annak kimenetét juttatja el az eredeti MD5, SHA-1 vagy tetszőleges algoritmushoz. Enkapszuláció.

A salt a hashalgoritmusok sava-borsa!

Fóti Marcell
marcellf@netacademia.net
MCSE, MCT, MCDBA, MZ/X

A cikkben szereplő URL-ek:

[1] LOPhtCrack
http://www.atstake.com/research/lc3/index.html
[2] Törőszótárak
http://wordlists.security-on.net/download.html
[3] Message Digest, MD5, RFC 1321
http://www.ietf.org/rfc/rfc1321.txt
[4] Keyed-Hashing for Message Authentication, HMAC, RFC 2104
http://www.ietf.org/rfc/rfc2104.txt
[5] Secure Hash Algorithm, SHA-1, RFC 3174
http://www.ietf.org/rfc/rfc3174.txt

Szálak *(II. rész)* Időzítés és prioritás

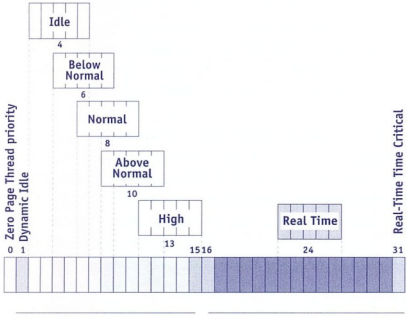
Időzítés és prioritás

Az előző számban megjelent cikk folytatásaképpen a következő oldalakon kicsit részletesebben bemutatjuk a Windows NT/2000 végrehajtási szálakat időzítő rendszerét, annak szabályait. Cikkünk – az előző részhez hasonlóan – David Solomon **[1]** Barcelonában, a Tech.Ed 2001-en elhangzott előadása, valamint ugyanő és Mark Russinovich Inside Microsoft Windows 2000 *(Third Edition)* **[2]** című könyvének aktuális fejezete alapján készült. Ez a könyv egyébként ajánlott, sőt, kötelező olvasmány mindenkinek, aki szeretne komolyabban foglalkozni a Windows lelkivilágával.

Egy kis ismétlés

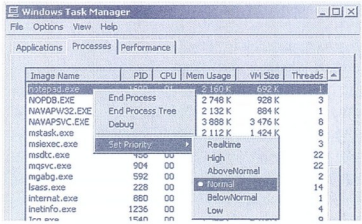
Foglaljuk össze az előző számban leírtakat: a Windows NT/2000 *(továbbiakban csak Windows, különös tekintettel arra, hogy nem a Windows 9x családról van szó)* operációs rendszer szinten végrehajtási szálakat futtat. Minden végrehajtási szál valamilyen processzhez tartozik *(egy processzhez akár több is)*; ezek a processzek képezik a tulajdonképpeni alkalmazásokat. A processz nem más, mint az általa birtokolt végrehajtási szálak részére fenntartott adminisztratív objektum és közös memóriaterület – a végrehajtási szálak időzítésében általában nem játszik szerepet. Kivételt képez ez alól a prioritás esete; az induló végrehajtási szálak kezdetben a szülő processz prioritásértékét öröklik, és a felhasználói felületen is csak a processz *(és általa az összes hozzá tartozó végrehajtási szál)* prioritása módosítható.

A Windows 32 prioritási szintet kezel; a magasabb érték magasabb prioritást jelent. A 32 prioritási szint két fő részle oszlik: az alsó félben *(0-15)* az úgynevezett dinamikus, a felső félben *(16-31)* pedig az úgynevezett valósidejű prioritásértékek találhatók. A 0 prioritásértéket kötelezően a Zero Page Thread viseli *(lásd a Windows memóriakezeléséről szóló cikket az előző számban)*.



A Windows 32 prioritási szintje és a prioritásosztályok elhelyezkedése

A fenti ábrán a Windows 32 prioritási szintje mellett a felhasználói felületen keresztül is elérhető, úgynevezett „prioritásosztályok" találhatók. A processzek *(és rajtuk keresztül a végrehajtási szálak)* prioritását ugyanis mi, a felhasználó is meghatározhatjuk és módosíthatjuk. Futás közben a Task Managerben jobb gombbal kattintva kiválaszthatjuk a kívánt prioritást.



Processz prioritásának módosítása futás közben a Task Manager segítségével

Az egyes prioritásosztályok beállítása során a processz az előző ábrán látható középértéket kapja *(a Normál prioritás középértéke például 8)*. A prioritás indítás előtti beállításának egyetlen módja a start parancs, amelynek paramétereként átadhatjuk a kívánt prioritásosztályt, valahogy így *(lásd még: start /?)*:

```
start /low notepad.exe
start /abovenormal notepad.exe
```

A processz *(figyelem! nem a végrehajtási szál!)* aktuális, alap *(bázis)* prioritásosztályát a Task Manager-ben, a Base Priority oszlopban láthatjuk, de jobban járunk ha *(például)* a Windows 2000 Support Tools-ból futtatjuk a Process Viewer-t *(pview.exe)*, mert akkor nemcsak a bázisprioritást, de a végrehajtási szálak aktuális *(current)* prioritásértékeit is megleshetjük. A végrehajtási szálak státuszának és prioritásának másik jó megfigyelőeszköze a Performance Monitor *(használatát az előző számban már bemutattuk)*.

Az operációs rendszer magja *(a kernel)* valósidejű prioritású végrehajtási szálakat futtat. Legyünk óvatosak a valósidejű prioritás használatával, mert előfordulhat, hogy létfontosságú I/O végrehajtási szálaktól vesszük el a futási időt. Az általa elérhető végrehajtási szálak prioritását bármelyik felhasználó módosíthatja, egy korlátozással: valósidejű prioritást csak az állíthat be, aki rendelkezik az „Increase scheduling priority" privilégiummal – ez alapértelmezésben csak az Administrators csoportra érvényes. Ha a prioritást módosítani próbáló felhasználó ezzel a privilégiummal nem rendelkezik, a processz valósidejű helyett „High"

***(dinamikus)* prioritásosztályba kerül. Végül: meg kell jegyeznünk, hogy miközben a valósidejű prioritásokról beszélünk, a Windows nem igazi valósidejű *(real-time)* operációs rendszer *(ld. még: [3])*.**

Alap és pillanatnyi prioritás

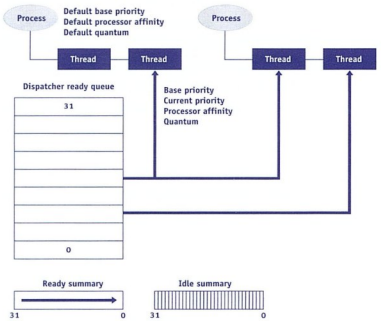
Egy végrehajtási szál pillanatnyi prioritásértéke két részből áll össze:

- A végrehajtási szál alapprioritása *(base priority)*.
- A relatív prioritás *(priority boost)*, ami az alapprioritáshoz hozzáadódik, és értéke idővel változhat.

A priority boost-ról később még lesz szó, de azt már most elmondhatjuk, hogy a valósidejű prioritásértékkel futó végrehajtási szálaknál a rendszer nem használja – ezek pillanatnyi prioritásértéke mindig megegyezik az alapprioritással *(ezért hívják a nem valósidejű prioritásértékeket dinamikus prioritásnak)*. A végrehajtási szálak futtatásának időzítéséhez az időzítő *(dispatcher)* mindig a pillanatnyi prioritást ellenőrzi.

A dispatcher várakozási sor-adatbázisa

A végrehajtási szálak időzítését végző komponens, a dispatcher *(ami a maga valójában nem is létezik, feladatát kernelszerte elszórtan található összetevők végzik)* minden prioritásértékhez egy-egy várakozási sort tart fenn. Ezekben a várakozási sorokban találhatók a futásra kész *(Ready állapotú)* végrehajtási szálak. A dispatcher adatbázisát a következőképpen ábrázolhatjuk:



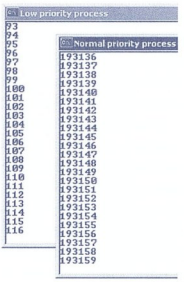
A Dispatcher várakozási sor-adatbázisa

Az ábra bal oldalán láthatók a különböző prioritásértékhez rendelt várakozási sorok *(Dispatcher ready queue)*. Az adatbázis a feldolgozás meggyorsítása érdekében két összegző mezőt is tartalmaz:

- A *Ready summary* bitmező valamely bitjének értéke azt jelzi, hogy az adott prioritású várakozási sorban található-e futásra kész végrehajtási szál *(azaz érdemes-e egyáltalán benézni a várakozási sorba)*.
- Az 3 bitmező pedig processzorokat tartalmaz: a bitek értéke az adott processzor szabad *(idle)* állapotát jelzi. Találós kérdés: legfeljebb hány processzort képes használni a jelenlegi legizmosabb Windows, a Datacenter Server?

A legfontosabbak mégiscsak a várakozási sorok: amikor a Windows a következő nyertes végrehajtási szálat keresi, ezeket a várakozási sorokat ellenőrzi, a legmagasabb prioritásútól a legalacsonyabbig. Ha egy várakozási sorban találunk futásra kész végrehajtási szálat, ő léphet futó státuszba. Ha egy a várakozási sorban egynél több futásra kész végrehajtási szál található, a dispatcher igazságos módon körbe-körbe adja majd a futási jogot *(round-robin)*. Ne feledjük, ez az igazságosság csakis azonos prioritású végrehajtási szálak között érvényes, az alacsonyabb prioritással rendelkezőkkel szemben sokkal keményebben viselkedik a Windows: amíg magasabb prioritású szál futásra kész, bizony az alsó házakban semmi sem történik.

Ennek demonstrálására készítettünk egy kis példaprogramot *(a* **[4]** *címről letölthető)*:



Harcban a processzorért: a két processz nagyjából egyszerre indult, különbség csak a prioritásukban van

Az ábrán látható tesztalkalmazás egyvalamire képes: számol. Teszi ezt addig, amíg a CTRL+C billentyűkombináció lenyomásával meg nem állítjuk. Az st.bat a start parancs segítségével gyors egymásutánban kétszer elindítja ugyanazt az alkalmazást, először egy alacsonyabb, majd egy magasabb prioritással. Az eredmény önmagáért beszél. Ha kipróbáljuk, érdemes megfigyelni, hogy a futásidő eloszlása korántsem sima: amikor néha végre az alacsonyabb prioritású végrehajtási szál is szóhoz jut, gyors egymásutánban 15-20 sort ír a képernyőre, majd újabb hosszú szünet következik.

Látható, hogy a magasabb prioritású processz *(és az ő egyetlen végrehajtási szála)* több nagyságrenddel többet futhat, mint az alacsonyabb prioritású változat. Az érdekes az, hogy ez utóbbi is szóhoz jut néha... nem mond ez ellent a pár sorral ezelőtt említetteknek? Nos, nem: az alacsonyabb prioritású processz nyilván akkor futhatott, amikor a magasabb prioritású éppen nem volt futásra kész állapotban *(például mert I/O műveletre várt)*, de az is lehet, hogy az operációs rendszer mentette meg az „éhhaláltól" – lásd később.

A működő rendszer egyik kulcsa a várakozás

A végrehajtási szálak leggyakoribb állapota a várakozás *(wait state)*. Egy szál sok mindenre várhat: I/O művelet befejezésére, másik végrehajtási szál lefutására, különféle rendszereseményekre, sült galambra *(ami tíz másodperc múlva érkezik)* – ezek mindegyike annyiban hasonló, hogy a várakozás során a végrehajtási szál várakozó státuszba lép *(onnan majd az adott esemény bekövetkeztekor maga a Windows lépteti vissza)*. A várakozó státusz pedig nem futásra kész státusz, ezért ilyenkor végre feljöhetnek a felszínre az addig fuldokló, alacsonyabb prioritású végrehajtási szálak is. Van még más eset is, amikor a Windows *(a Dispatcher)* új-

ra kiértékeli, ki lehet a következő futó szál – jusson eszünkbe a felsorolás az előző számból:

- Új végrehajtási szál futásra kész – mert új szál jött létre, vagy tért vissza várakozó státuszból
- Az éppen futó végrehajtási szál elhagyja a futó státuszt – mert lejárt a számára kijelölt időszelet; mert várakozó állapotba lépett, vagy mert végleg befejezte a működését
- Valamelyik végrehajtási szál prioritása megváltozik – ha önmaga, vagy akár az operációs rendszer módosítja a szál prioritását
- Ha a futó végrehajtási szál Processor Affinity értéke megváltozik

Priority Boost I/O műveletek után

A felhasználó felé nyújtott optimális reakcióidők kialakítása érdekében a Windows bizonyos esetekben átmenetileg megnövelheti egy-egy végrehajtási szál prioritását. Az I/O műveletet „végző" végrehajtási szál elküldi a kérést az eszközmeghajtónak, majd a művelet befejezéséig wait state-be kerül. Amikor a művelet befejeződik, a szál újra futásra kész státuszba kerül, ráadásul a Windows átmenetileg a prioritását is megnöveli. Hogy ez a növelés milyen mértékű, azt az adott I/O eszköz eszközmeghajtója határozza meg. Az ajánlott érték típusonként más és más *(videó, lemez, CD-ROM, párhuzamos port: 1, hálózat, soros port: 2, billentyűzet, egér: 6, hang: 8)*. Ez az érték mindig a szál alapprioritásához adódik hozzá és az összeg soha nem lépi át a 15-öt *(valósidejű prioritásnál pedig nincsen boost)*. A megnövelt prioritás csak rövid ideig érvényes, minden quantumegység lejártakor a prioritásérték is csökken eggyel, egészen addig, míg a végrehajtási szál újra el nem éri az alapprioritásértéket.

Priority Boost az előtérben futó végrehajtási szálnak

A Windows a wait state-ből visszatérő, előtérben futó végrehajtási szál prioritását is megnöveli, mégpedig az előző számban bemutatott Win32PrioritySeparation registry érték alsó két bitjén beállított értékkel *(a Quantum boostnál ez csak egy mutató volt („Boost értéke"), ezesetben viszont maga az érték számít)*. Ez az érték a végrehajtási szál pillanatnyi prioritásához adódik hozzá. Hasonlóképpen, további 2 boost-ot kap minden végrehajtási szál, ami ablakkezelő üzenetet kap *(ablak mozgatása, méretezése, stb)*.

Priority Boost az „éhhalál" *(starvation)* ellen

Minden operációs rendszer egyik fő problémája az alábbi: tegyük fel, hogy egy 7-es prioritású szál fut; egy 11-es prioritású szál egy olyan erőforrásra vár, amit éppen egy 4-es prioritású szál birtokol... A számlálós példában láthattuk, hogy a nagyétkű 7-es miatt a 4-es prioritású szál bizony nem lesz képes befejezni a munkáját, és elengedni az erőforrást – az eredmény az, hogy a 11-es prioritású szál is éhenhal. Az ilyen helyzetek elkerülése érdekében a Windows kernel „Balance Set Manager" nevű komponense folyamatosan ellenőrzi, hogy mely végrehajtási szálak nem jutottak szóhoz az elmúlt 300 órajel-intervallumban *(300x10ms = kb. 3 másodperc)*. Ha talál ilyet, a szegény elnyomott végrehajtási szál prioritását 15-re emeli, és dupla Quantumot ad neki *(!)*. Ezidő alatt a szál lélegzethez juthat. A dupla időszelet lejárta után azonnal minden *(prioritás)* visszatér a régi kerékvágásba. Indítsuk csak el újra a példaprogramocskát! Nem feltűnő, hogy az alacsonyabb prioritású végrehajtási szál kb. három másodpercenként jut szóhoz? :-) Természetesen ez a módszer sem tökéletes, ráadásul a Balance Set Manager *(teljesítményokokból)* egyszerre csak 16 elnyomott végrehajtási szálat ellenőriz – de az esetek többségében ez a kis segítség elég a patthelyzetek *(deadlock)* elkerülésére.

Többprocesszoros rendszerek

Többprocesszoros rendszerben a végrehajtási szálakhoz úgynevezett processzor-affinitást rendelhetünk. Ez tulajdonképpen nem más, mint egy maszk, ami meghatározza, hogy az adott végrehajtási szál melyik processzor*(ok)*on futhat, és melyeken nem. Még két másik fogalmat kell tisztázni:

- Ideal Processor: a végrehajtási szál létrehozásakor a Windows véletlenszerűen hozzárendel egy „kedvenc" processzort.
- Last Run Processor: az a processzor, amelyen a végrehajtási szál utoljára futott.

Amikor egy végrehajtási szál futásra kész állapotba lép, a Windows szabad *(idle)* processzort keres a számára. Ha megteheti, a végrehajtási szál ideális processzorát választja; ha az nem szabad, a last run processzort; ha pedig az sem szabad, akkor a Dispatcher adatbázisból keres egy ráérő példányt.

Ha nincs szabad processzor, akkor a Windows megkeresi az első, a végrehajtási szál által használható processzort, és ha azon a végrehajtási szál prioritásánál alacsonyabb prioritású szál fut *(running)*, vagy készül a futásra *(standby)*, akkor átveszi annak a helyét *(ez a preemption)*. Ha a prioritás nem elegendő, a Windows nem keres másik processzort a végrehajtási szálnak, hanem azt a várakozó sorba helyezi.

Amikor egy processzor felszabadul, az egyprocesszoros rendszerekkel ellentétben a Windows nem az első futásra kész végrehajtási szálat indítja, hanem előbb azt, amelyik megfelel az alábbiak valamelyikének:

- Utoljára az adott processzoron futott
- Az ideális processzora az adott processzor
- Legalább két Quantum óta futásra kész
- 24, vagy nagyobb a prioritása

Végül...

Szeretnék bemutatni egy, a weben talált tanulságos szimulációt. Kattintsunk a **[5]** címre, és elénk tárul egy kétprocesszoros rendszer *(Flash plug-in szükségeltetik)*, ahol követhetjük a három processz négy végrehajtási szálának futását. A végrehajtási szálak processzor-affinitást is tartalmaznak, a szimulációt a „Clock" feliratú kapcsoló nyomogatásával léptethetjük. Jó szórakozást!

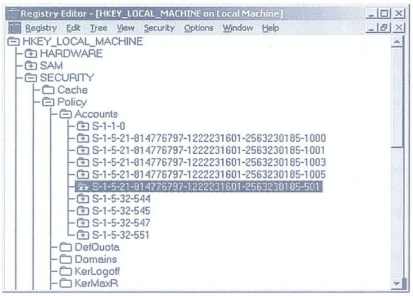
Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:

[1] http://www.solsem.com
[2] http://mspress.microsoft.com/prod/books/4354.htm
[3] http://support.microsoft.com/support/kb/articles/Q94/2/65.ASP
[4] http://technet.netacademia.net/download/threading
[5] http://www.808multimedia.com/winnt/simulator.htm

SID vita

Sok víz lefolyt a Dunán amióta a Microsoft megtiltotta népünknek a klónozószoftverek használatát, mondván, hogy a kopizott gépek összeakadnak a hálózaton, hisz azonos a SID-jük. A gép-SID *(Security ID)* az adott gép biztonsági adatbázisának egyedi azonosítója. Az ott később létrehozott biztonsági objektumok *(felhasználók, csoportok)* a gép-SID alapján, növekvő számsorrendben kapnak egyedi belső azonosítót. Erről tanúskodik az alábbi ábra is: világosan látszik az objektumok SID-jeinek közös gyökere. Ha a gép-SID-ek nem különböznek a hálózat összes gépén, akkor – minden emberi számítás szerint – lesznek azonos SID-del rendelkező felhasználók, és lesz nemulass!!



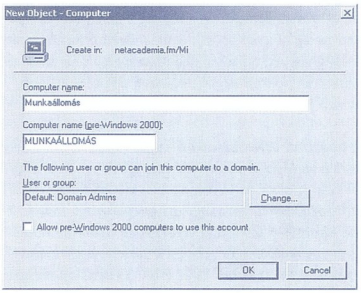
SID-ek a regisztrációs adatbázisban. A kijelölés a Guest SID-jén áll

A fennhangon hirdetett tiltás ellenére a MS maga is előszeretettel klónoz például konferenciáin, ahol egyszerre 500 gépre kell felhúzni az alkalmazásokkal zsúfolásig teletelepített operációs rendszert. Ilyenkor bizony nem mindegy, hogy klónozó *(Ghost, DriveImage stb.)* vagy automatikus telepítő *(unattend.txt vagy RIS)* használatát választjuk-e, az előbbiek ugyanis egy nagyon fontos dologban lekörözik az utóbbi megoldásokat: tudnak multicastolni. Az IP Multicast lehetővé teszi, hogy akár 500 gép telepítése esetén is csak egyetlenegyszer menjen át a WINNT könyvtár teljes tartalma a hálózaton – szemben az automatikus telepítésekkel, amikor minden géphez egyenként, külön-külön el kell juttatni ugyanazt az adathalmazt.

Most mondhatják erre Kedves Klónozó Olvasóim, hogy nincs itt ellentmondás, hisz a modern klónozószoftverek lehetővé teszik a gépek SID-jeinek utólagos átírkálását. Igen ám, de minek? Meggyőző, hibátlan példák igazolják, hogy klónozott, azonos SID-del még NT4 Backup Domain Controllerek is elketyegnek, nem beszélve a munkaállomásokról. Akkor hogy is van ez?

Elsőként vizsgáljuk meg, milyen körülmények között lenne „nemulass", okozna gondot a hálózaton felbukkanó két azonos gép-SID. Nyilván akkor, ha találkoznának. Ámde a Network Monitor tetszőlegesen szorgos használata sem elegendő ahhoz, hogy gép-SID-eket láthassunk a kábelen, azon egyszerű oknál fogva, hogy azok sohasem hagyják el a gazdagépet azt az esetet kivéve, amikor a merevlemezt átcipeljük egyik gépből a másikba. Ennélfogva össze sem ütközhetnek. Pont. Cikk vége...?

Ha Önök most azt mondják, hogy nincs igazam, mert láttak már Computer Accountot a tartományvezérlőkön, akkor erre azt mondom: az nem a gép eredeti, belső azonosítója, hanem egy, a tartományvezérlő által generált új felhasználó, új SID-del! Minden új felhasználó új SID-et kap. A Computer Account nem „átmegy", hanem ott születik a tartományvezérlőn a munkaállomás csatlakozásának pillanatában. Bizonyítékként szolgálhat, hogy Computer Accountot kézzel is létrehozhatunk *(akár egy olyan gép számára, mely még olyan messze áll a tartományi csatlakozástól, hogy hungarocel és hullámpapír fogságában sínylődik egy raktárban)*.



Computer Account létrehozása a címtárban

Az azonos SID-del rendelkezők egy személynek számítanak?

Egy picit most már elárultam az igazságot, így a babonások második érve előtt sem hullunk a porba: azt mondják, hogy ha két gépen azonos két felhasználó SID-je, akkor még abban az esetben is hozzáférnek a másik, sziámi iker adataihoz, ha annak teljesen más a bejelentkezési neve és jelszava. Hümm. Mondjak ellenpéldát, vagy nem is kell magyarázonom? Két ellenérvem is van:

1. A felhasználók azonosítása név+jelszó alapján történik. Az azonosítás pillanatában senkit sem érdekel a júzer SID-je. Azonosítás után meg úgyis annak a nevében dolgozhat, akinek tudta nevét és jelszavát, a kör bezárult.

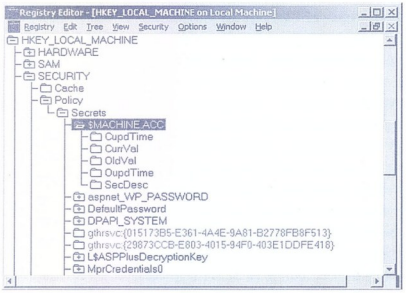
2. Vannak a hálózaton azonos SID-ek. Minden telepített Windowsban tejlesen azonos az Authenticated Users, az Everyone, az Administrators és még sok egyéb csoport SID-je. Ezeket Well Known *(közismert)* SID-eknek hívják, és az **[1]** címen olvashatunk róluk részletesebben. Néhány példa:

```
Administrator: S-1-5-domain/machine SID-500
Guest: S-1-5-domain/machine SID-501
Administrators csoport: S-1-5-32-544
```

De ugye nem találkoztunk még olyan esettel, hogy a Well-Known SID-ek miatt automatikusan be tudunk lépni rendszergazdaként a hálózat összes gépén? Pedig minden gépen tagja minden rendszergazda a mindig azonos SID-ű Administratos csoportnak. Hiába azonos a SID, mivel az a kutyát nem érdekli. Mindenki név+jelszó párossal fér hozzá a hálózati erőforrásokhoz – már ha tudja a jelszót. Ha nem tudja, megsütheti a Well Known és ráadásul azonos SID-jét! De térjünk vissza a Computer Accounthoz.

Mi is a Computer Account?

Egy név-jelszó páros, mellyel a bootolást végző munkaállomás hitelesíti önmagát a tartományvezérlők előtt. Ezt a két adatot *(név+jelszó)* mindegyik munkaállomás megjegyzi magának, hogy minden rendszerindításkor sikerrel vegye a bejelentkezési akadályt. Megjegyzi? Ugyan! Felírja magának, ide:



A munkaállomás ide „rejti" a saját tartományi nevét és jelszavát. A júzer meg kiragasztja a magáét a monitorra. Melyik a veszedelmesebb?

Sikeres név *(gépnév$)* és jelszó megadása után az adott masina belép a tartományba, és úgy is marad kikapcsolásig. Mindegy, hogy a júzerek be tudnak-e jelentkezni a tartományba – ő bent van és kész.

Megoldódik egy rejtély

Minden rendszergazda gépe egységesen zavaros. Van rajta vagy hat Windows, ebből négy munkaállomás, kettő kiszolgáló. Ha el akarjuk kerülni, hogy ide-oda bootolgatásunk miatt a Browse lista *(Hálózatok, Network Neigborhood ikon)* tele legyen a mi hat operációs rendszerünk kikapcsolt zombijaival, egyszerűen mind a hatnak ugyanazt a gépnevet adjuk. Úgysem futnak egyszerre!

Azonban amikor be akarjuk léptetni a tartományba mind a hatot, azt tapasztaljuk, hogy egyszerre csak egy tud bemenni. Miért? Mert a legelső, mely csatlakozik, *(és amely miatt létrejön a Computer Account)*, azonnal és ravasz módon meg is változtatja a hozzátartozó jelszót, hogy ne legyen más, aki be tudjon lépni ugyanezzel a gépnév$ login névvel. Hát nem is tud! A maradék öt Windowsunk hoppon marad: ugyanolyan névvel nem tudnak taggá válni ugyanabban a tartományban – sajnos.

Ha jobban szemügyre vesszük a munkaállomás és a tartomány kapcsolatát, észrevehetünk egy-két hasonlóságot a megbízotti *(trust)* kapcsolatokkal, hiszen:

- A tartományi felhasználók akár lokálisan, akár megosztásokon keresztül ténykedhetnek a munkaállomáson, mert az elfogadja a DC-k hitelesítését. Ha a DC azt mondja, hogy Bubuka az Bubuka, akkor ezt a munkaállomás nem kérdőjelezi meg, hisz neki *(we trust in dc)*.
- A munkaállomás lekérdezheti a DC-k használói adatbázisát *(bizalom, bizalom)*, hogy abból tetszésünk szerint válogathassunk pasasokat és csoportokat a jogosultságlistákba.
- A tartományi globális rendszergazdák a munkaállomáson is erős legények: belehullott a Domain Admins csoport a lokális Administrators csoportba.

Ez biza trust, még ha kicsit butított is. Miben különbözik az Active Directory tartományok közötti meghatalmazotti viszonytól? Hát szinte mindenben:

- Nem automatikusan jött létre *(kézzel be kellett terelnünk a gépet a tartományba)*.
- Nem kétirányú *(a munkaállomás lokális felhasználóiban és csoportjaiban nem bízik meg a tartomány)*.
- Nem tranzitív *(a munkaállomások nem fűzhetők fel minitrust-láncba)*.
- A munkaállomás nem léphet nászra másik munkaállomással a biztonsági adatbázis közös felnevelésére *(ahogy az igazi DC-k teszik)*.

De ezzel a különbségek végére is értünk. Ki kell mondanunk, hogy a Windows NT Workstation és a Windows 2000 Professional olyan TARTOMÁNYVEZÉRLŐ *(!)*, melynek kapcsolatfelvételi képességei drasztikusan le vannak korlátozva ugyan, de attól még a DC az DC! A hercegkisasszonyt is egyből felismerjük a dunnája alá rejtett borsószem révén, a munkaállomások poros fedőrétege alatt is megcsillan a sárarany.

Akkor talán értjük a trustokat is?

Két tartomány között a meghatalmazotti viszony nem más, mint bootoláskor egymáshoz történő kölcsönös bejelentkezés egy, a Computer Accounthoz kísértetiesen hasonló fiók, az Interdomain Trust Account segítségével. Ennek jelszavát éppolyan formában *(a kéménybe van felírva korommal)* rejtegetik a DC-k, mint ahogy kicsi munkaállomás feljegyzi a saját Computer Accountjához tartozót. Előfordulhat, hogy mégis gondot okoz a közös SID: ha van olyan alkalmazás, ami épít arra, hogy a gépek SID-je különböző – hiszen definíció szerint az, végülis megteheti – no akkor az az alkalmazás skizofrén állapotba kerül. De ha óvatosak vagyunk, mindez elkerülhető.

Fóti Marcell
marcellf@netacademia.net

A cikkben szereplő URL-ek:
[1] Well Known Security Identifiers
http://support.microsoft.com/support/kb/articles/Q243/3/30.ASP

Hatékony megoldás a jogsértő tartalommal szemben

avagy jó-e a notice and take down eljárás?

A notice and take down eljárás az Internetes önszabályozás keretében alakult ki, mára azonban ez a megoldás egyes jogszabályokban is helyet kapott. Az eljárás lényege az, hogy abban az esetben, ha egy adott oldalon jogsértő tartalom jelenik meg, a tartalom szolgáltatójának személyétől függetlenül a panaszt, és az eltávolítás iránti kérelmet az Internetszolgáltatóhoz *(ISP-hez)* kell eljuttatni, aki ideiglenesen megakadályozza a sérelmes információhoz való hozzáférést, vagy gondoskodik annak ideiglenes eltávolításáról. Az ISP egyidejűleg értesíti a tartalom szolgáltatóját a beérkezett kérelemről, aki rendelkezésre álló, általában néhány napos határidőn belül tiltakozhat a hozzáférés letiltása vagy az eltávolítás ellen. Ez esetben az ISP az információt visszahelyezi, illetve a hozzáférést helyreállítja. Amennyiben a tartalomszolgáltató ilyen tiltakozást nem jelent be, a sérelmes információt véglegesen eltávolítják. Ellenkező esetben annak, aki azt sérelmesnek találja, marad a hagyományos bírói fórum arra, hogy ott intézkedést kérjen a jogsértő állapot megszüntetésére, és kártérítésre perelje a tartalom szolgáltatóját.

Az eljárást kitalálók lényegében két legyet akartak egy csapásra ütni: gyors megoldást kínálni jogsértő tartalom esetén, és ennél sokkal lényegesebb: az Internetszolgáltatót mentesíteni mindenfajta felelősség alól.

Az önszabályozás keretében elterjedt megoldás azonban úgy működik, ha az alkalmazási terület nem korlátlan, arra nem kap bárki, bármilyen kérdésben felhatalmazást.

Jellemzően a szerzői jog területén vált be, és nyújt hatékony jogvédelmet azoknak, akiknek szerzői művét felhatalmazás nélkül helyezik el a hálón.

Az Internet önszabályozásának megfelelő példája a finn megoldás, ahol is a Finnországban működő szerzői jogvédő, az ARTISJUS ottani partnere, és a hangfelvételelőállítók finnországi közös jogkezelő szervezete állapodott meg az egyik legnagyobb Internetszolgáltatóval abban, hogy szerzői jogot sértő tartalomra vonatkozó értesítésük esetén a szolgáltató a tartalmat az oldalról eltávolítja. A szolgáltató további kötelezettséget is vállalt a tartalom szolgáltatójának azonosítására, ezzel elősegítve a későbbi felelősségrevonást. A rendszer nyilvánvalóan hatékonyan tud működni, hiszen leginkább a monopolhelyzetben lévő közös jogkezelő szervezet lehet alkalmas arra, hogy az adott mű vonatkozásában elbírálja, rendelkezik-e az azt a hálóra helyező valamilyen felhasználási joggal vagy sem. Így tehát az e szervezettől érkező jelzés minden bizonnyal megalapozott lehet, hiszen maga a szervezet is arra vállalt kötelezettséget, hogy tartalomeltávolítását csak alapos indokkal kérik. Az Internetszolgáltató önkéntes kötelezettségvállalása hátterében pedig az áll, hogy cserébe a közreműködéséért, a közös jogkezelő szervezetek garantálták, hogy nem kezdeményezik olyan jogszabály megalkotását, amely az Internetszolgáltató felelősségének megállapításáról szólna olyan tartalomért, amelyre nyilvánvalóan nem gyakorolt hatást.

A notice and take down eljárás törvényi szintre emelése az Amerikai Egyesült Államok jogalkotásában, a Digital Millenium Copyright Act, azaz a Digitális szerzői jogi törvény keretében történt meg. Amint azt a törvény címe is sugallja, itt is a szerzői jogi jogsértésekre redukálta a jogalkotó az eljárás alkalmazhatóságát, tehát panaszt csak akkor tehet bárki, ha szerzői mű jogosulatlan felhasználását tapasztalja a hálón. A kérelmezőnek az értesítésben azonosítania kell a tartalmat, annak megtalálási helyét, és olyan információkat kell szolgáltatnia, amelyek a jogsértés tényét valószínűsítik. A kérelmezőnek továbbá nyilatkoznia kell arról is, hogy jóhiszeműen cselekszik. A kérelemre az Internetszolgáltató köteles a megjelölt tartalmat tíz napra eltávolítani, és egyidejűleg erről a tartalomszolgáltatót értesíteni. A tartalomszolgáltató köteles haladéktalanul válaszolni az értesítésre. Amennyiben válaszában a jogsértést elismeri, vagy 10 nap alatt nem válaszol, úgy a tartalom eltávolítása végleges marad. Ellenkező esetben, vagyis ha a jogsértés tényét a tartalom szolgáltatója vitatja, a sérelmezett tartalom újra hozzáférhetővé válik, kivéve, ha a kérelmező igazolja, hogy bírósági eljárást kezdeményezett. Ez esetben az eljárás jogerős lezárásáig a kifogásolt tartalom továbbra sem válik hozzáférhetővé az adott tárhelyen.

A szolgáltató együttműködését e jogszabályban is a felelősség alóli mentesülés garantálja, hiszen ha az Internetszolgáltatónak nem volt tudomása a jogsértő tartalomról, illetve azt az értesítést követően onnan haladéktalanul eltávolítja, vele szemben kártérítési igénnyel nem lehet fellépni.

Mint látjuk, a notice and take down eljárás azért működik, mert az Internetszolgáltató számára megnyugtató megoldást ad, ha jól körülhatárolt körben, alaposan valószínűsíthető jogsértések esetén eleget téve a jelzésnek eltávolítja a tartalmat, és ezzel mentesül attól, hogy esetleg olyan cselekményért kelljen kártérítési vagy akár büntetőjogi felelősséget is vállalnia, amelyre tényleges befolyással nem rendelkezett. Abban az esetben azonban, ha az eljárást korlátlanul, bármely tartalom vonatkozásában és bárki számára alkalmazhatóvá tesszük, fennáll annak a veszélye, hogy a semmire sem vezető automatizmussal kiváló lehetőséget teremtünk arra, hogy egyesek jogaikat visszaélésszerűen gyakorolják.

Sajnos a magyar Országgyűlés előtt lévő, az elektronikus kereskedelemről és az információs társadalmi szolgáltatásokról hangzatos nevet viselő törvény tervezete pontosan ezt a megoldást tartalmazza!

Dr. Mayer Erika
Nádas & Mayer Ügyvédi Iroda

ISA Server *(IV. rész)*

Packet Filtering

A csomagszűrés érvényesíthető mind a befelé jövő, mind a kifelé irányuló kérésekre. A kifelé irányuló kéréseknél felfoghatjuk úgy is, hogy ez az utolsó védelmi vonal, vagyis ha a kérést átengedi egy protokollszabály *(protocol rule)* és egy webhelyre és –tartalomra vonatkozó szabály *(site and content rule)*, még mindig blokkolhatja egy tiltó csomagszűrő. Például:

- Protocol rule: a „web users" csoportnak engedélyezi a HTTP elérést
- Site and Content rule: minden webhelyhez engedélyezi a hozzáférést a „web users" csoportnak
- IP Packet Filter: blokkolja a hozzáférést a 110.110.110.100 IP címhez

Ez a szabályrendszer a „web users" csoportnak minden HTTP hozzáférést engedélyez, kivéve a 110.110.110.100 IP című webhelyhez – vagyis a tiltó csomagszűrő szabály elsőbbséget élvez az egyéb szabályokkal szemben. Ez különösen akkor lehet hasznos, amikor gyorsan kell megoldanunk egy biztonsági problémát. Egy adott hely elérésének letiltása könnyen megvalósítható a megfelelő csomagszűrő szabály létrehozásával.

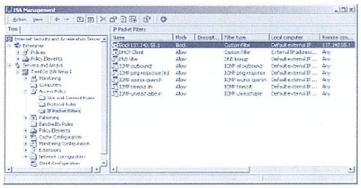
Emellett a csomagszűrési eljárást arra is használjuk, hogy az ISA Server felé tartó, illetve a tőle jövő kapcsolatokat szabályozzuk. Például ezzel biztosíthatjuk, hogy az ISA Server ne válaszoljon az ICMP echo *(ping)* kérésekre, amivel megakadályozzuk, hogy egy egyszerű pingeléssel bárki kideríthesse létezését.

A csomagszűrőket használjuk arra is, hogy a DMZ-hez való hozzáférést szabályozzuk, ha a DMZ-t egy háromlábú ISA felhasználásával alakítjuk ki. Bár sokak szerint a DMZ kialakítására nem ez a legjobb módszer, biztos vagyok benne, hogy – ha másért nem is, hát a költségek miatt – sokan ilyen rendszert építenek ki *(erről természetesen szó lesz a későbbiekben, de addig is érdemes lehet megnézni az ISA súgóját. Keressük a következőt: three-homed perimeter network configuration)*.

A csomagszűrési eljárás tehát minden esetben megvalósítja ezeket a szabályozásokat, és engedélyezi vagy tiltja a kapcsolatokat. Ne feledjük azonban, hogy mint minden csomagszűrés, ez is „szűklátókörű", vagyis viszonylag kevés adattal dolgozik. Háromféle csomagszűrést valósíthatunk meg. Az első a külső hálózat felől az ISA Server-re érkező kéréseket szabályozza. Hacsak nincs külön megengedve *(csomagszűrő vagy közzétételi szabály)*, ezek a kapcsolatok tiltottak. A második a külvilág számára biztosít hozzáférést az ISA Serveren futó szolgáltatásokhoz. Például, ha egy kívülről elérhető webkiszolgálót szeretnénk futtatni az ISA-n, meg kell nyitnunk a 80-as portot egy csomagszűrő szabállyal *(azért ha lehetséges, a webkiszolgálót helyezzük inkább a DMZ-be!!!)*. És végül létrehozunk egy olyan szabályt is, amely a fenti példának megfelelően a csomagszűrő a külső hálózat elérését fogja korlátozni, mondjuk egy olyan webhelyét, amelyről tudjuk, hogy káros kódot tartalmaz.

A csomagszűrést a protokolltípus *(például TCP, UDP)*, a portszámok *(például 25, vagyis az SMTP port)*, a helyi és a távoli számítógép IP címének megadásával definiáljuk. A távoli számítógép azt/azokat a külső hálózaton levő számítógépe*(ke)*t jelenti, amelyekre a szabály érvényesül. A helyi számítógép jelentheti magát az ISA Server-t, vagy (háromlábú ISA esetén) egy DMZ-ben levő gépet, amire a szabály érvényesül. Az IP csomagszűrés statikus, egy adott porton keresztül folytatott kommunikáció vagy mindig tiltva, vagy mindig engedélyezve van. A megengedő szűrők átengedik a bennük definiált *(adott portra irányuló)* forgalmat. A tiltó *(blokkoló)* szűrők pedig mindig megakadályozzák, hogy az adott csomagok áthaladjanak az ISA Server-en.

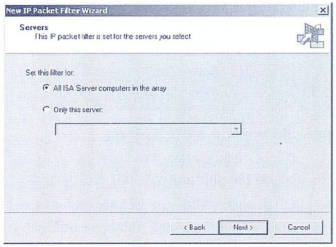
A következőkben bemutatjuk egy csomagszűrő szabály létrehozását. Ezeket a beállításokat az adott tömb neve alatt megtalálható access policy/IP packet filters alatt a „Create a new packet filter"-re kattintva *(1. ábra)* érhetjük el. Jelen példánkban a 110.110.110.100 IP cím felé irányuló összes kérés blokkolását mutatjuk be.



A csomagszűrők beállításainak helye

Az első párbeszédablakban névvel láthatjuk el szabályunkat. A áttekinthetőség fenntartása érdekében alakítsunk ki valamilyen „szabványos" elnevezési mintát, és a későbbiekben is ezt alkalmazzuk. Javaslat: a név tartalmazza a szűrő által végrehajtott cselekvést *(allow/block)*, és a főbb adatokat *(például IP cím)*.

A varázsló következő lépésénél megadhatjuk, hogy ez a szabály csak egy adott ISA Server-en, vagy az egész tömbön érvényesüljön. Erre nagyon figyeljünk! A későbbiek során jó tudni, hogy melyik ISA Serveren milyen szabályok is vannak! Válasszuk most az All ISA Server computers in the array opciót.

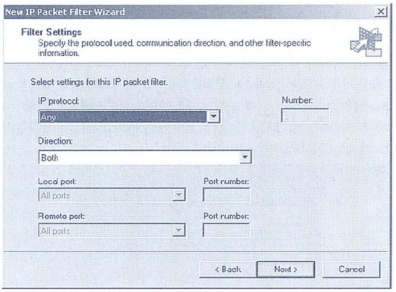


A szabályt érvényesítő tűzfalak kiválasztása

A következő lépésben kiválaszthatjuk a csomagszűrő típusát: allow vagy block. Válasszuk a block-ot.
Ezután választhatjuk ki a protokollt. Mivel jelen példánkban az adott IP címhez minden hozzáférést le akarunk tiltani, válasszuk az egyéni *(Custom)* opciót. *(Az előre definiált (predefined) opció választásával egy elég bőséges listából választhatnánk, ami csomagszűrőnk működését a kiválasztott protokollokra korlátozná.)*
Ezután kiválaszthatjuk az egyénileg megadott szűrőnk típusát. Lássuk miből választhatunk!

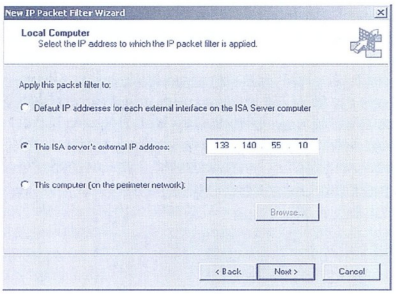
- Protokoll típusa *(IP Protocol)*: például TCP, UDP
- A kapcsolat iránya *(Direction)*: például befelé *(incoming)* vagy kifelé *(outgoing)* irányuló
- Portszámok

Példánk kedvéért válasszuk az Any-t az IP protocol-nál, és a Both-t *(mindkét irány)* a Direction-nál. Így kikényszerítjük, hogy az adott IP címhez minden hozzáférés tiltva legyen.



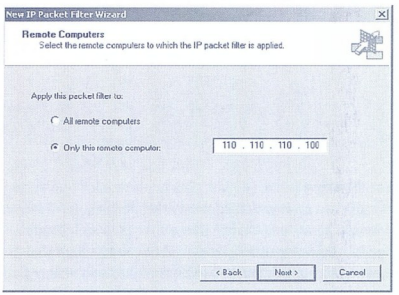
A protokoll megadása

A szűrést az ISA Server külső címén érvényesítjük. Vegyük észre azt is, hogy ezen a lapon a harmadik pontban adhatnánk meg egy a DMZ-ben levő számítógépre érvényesülő szűrést.



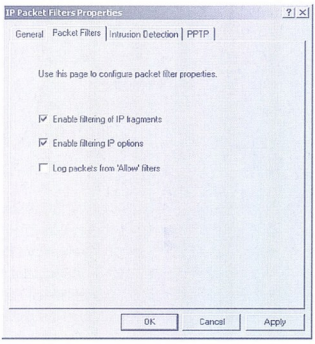
Csomagszűrés alkalmazása a külső lábon

Végül megadjuk, hogy szabályunk mely távoli számítógépre érvényesüljön.

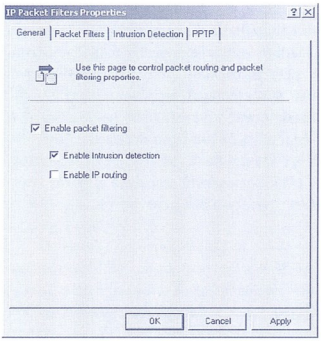


A távoli számítógép megadása

Van néhány dolog, amit érdemes tudni, ha alkalmazni akarjuk a csomagszűrést *(Hát már hogyne akarnánk!)*.
Először is az ISA Server bővítményei befolyásolhatják a csomagszűrés jellemzőit. Például ha az ISA Serverben megtalálható H.323 szűrő működik, az 1720-as számú port nyitva lesz, függetlenül a csomagszűrési beállításoktól. Vagyis ha nincs szükségünk a H.323 conferencing támogatásra, kapcsoljuk ki a H.323 szűrőt. A közzétételi szabályoknak *(publishing rules)* hasonló hatásai lehetnek. Ha elég elővigyázatosak vagyunk, és biztosítani akarjuk, hogy a csomagszűrő szabályaink a lehető legszigorúbbak legyenek, „támadjuk meg" az ISA Server-t a külső hálózat irányából egy port scannerrel, így láthatjuk, hogy mely portok vannak nyitva az ISA-n *(A port scanner nem okoz kárt, csak megnézi, melyek a nyitott portok. Használjuk bátran! Port scannerből több tucatot találhatunk az Interneten, íme egy példa: nmap.)*
Az ISA csomagszűrése képes az IP forgalom széttördelt részeinek *(fragment)* szűrésére. Ezek a töredékek nem károsak, hanem arra szolgálnak, hogy olyan adatok is átvihetők legyenek, melyek túl nagyok ahhoz, hogy egyetlen csomagba beleférjenek. A „hekkerek" oly módon használták/használják támadások kivitelezésekor ezt az eljárást, hogy olyan csomagokat hoznak létre, melyek első látásra ártalmatlanok, de összerakásuk után már kárt okozhatnak. Ennek a funkciónak a bekapcsolása erősen ajánlott.
Az utolsó fontos dolog, amit szeretnék megemlíteni, hogy az ISA képes az olyan csomagok szűrésére, aminél az IP options flag be van billentve. Az IP optionst is használják néha a „hekkerek". Így lehet például beállítani a source routing-ot, vagyis azt, hogy a forrásnál határozzák meg, hogy milyen útvonalon haladjon a csomag. Ennek a funkciónak a használata is javasolt.
A fenti beállítások elvégzéséhez navigáljunk az IP Packet Filters-hez, jobb klatty, majd válasszuk a Properties-t, ezután pedig a Packet Filters fület.



IP fragments és IP Options



A behatolásészlelés engedélyezése

Jónéhány megengedő csomagszűrés előre definiált az ISA Server-ben. Ezek a „konzerv" szűrők a telepítés végén engedélyeződnek. Lássuk mik is ezek:

- DNS Lookup. Ez teszi lehetővé, hogy az ISA Server elérje a DNS kiszolgálókat, a belső *(például webező)* ügyfelek kérései alapján. Ez a szabály mindenképpen szükséges, ha a névfeloldás ily módon történik. Természetesen a névfeloldást elvégezheti egy a belső hálózaton, vagy DMZ-ben található, megfelelően beállított DNS kiszolgáló is.
- ICMP szabályok: Ezek a szabályok engedélyezik az Internet Control Message Protocol *(ICMP)* csomagok ISA Server és külső hálózat közti küldését és fogadását.

E csomagok biztosítják például a flow controlt. *(ICMP Redirect, Source Quench, Time Exceeded stb. Lásd NetMon sorozat.)* Az alapértelmezett szabályok elég ártalmatlanok, például nem engedik meg, hogy az ISA Server válaszoljon a pingekre, vagyis akik így keresnek lehetséges célpontokat, pingeléssel nem fogják megtalálni ISA Serverünket. Mivel nem túl megengedőek, a megszokottól eltérően itt most meghagyhatjuk az alapértelmezett szabályokat.

Behatolásészlelés *(Intrusion Detection)*

Az ISA Server tartalmaz alapvető behatolás-észlelő funkciókat is. Az Intrusion Detection beállítások az MMC-ben két részre vannak osztva, a beállítások egyik fele az IP Packet Filters alatt érhető el, a másik fele pedig az Extensions alatt. Izgalmas nevük ellenére *(Land Attack stb.)* ezek sajnos mind jólismert, és ezer éve nem használt támadástípusok. Ennek ellenére érdemes megismerkednünk velük, sőt bekapcsolni őket, mert a legostobább script kiddyk hébe-hóba lefuttatnak egy-egy ősi támadást is. Nehogy ilyenbe haljunk bele!

A packet filters alatt beállítható behatolásészlelő funkciók alapértelmezésben nincsenek engedélyezve. Először navigáljunk a Packet Filters-hez, varázsoljuk elő a Properties lapot, és keressük meg a General fület. Ezután billentsük be az Enable Intrusion detection kapcsolót. Ennek előfeltétele, hogy a packet filtering is engedélyezve legyen.

A behatolásészlelés további ismertetése előtt megemlíteném, hogy ebben a párbeszédablakban található az „Enable IP routing" opció is. Erre most nincs szükségünk, tehát ne kapcsoljuk be. Kész, kivéseztük ezt a bonyolult ablakot, mehetünk tovább. Nem kell nagyon messzire mennünk, csak két fülnyit jobbra. Ez az Intrusion Detection fül, itt folytathatjuk a behatolásészlelés beállítását. A kis jelölőnégyzetek kipipálásával engedélyezzük a felsorolt támadástípusok észlelését:

Port scan attack

Ez a riasztás arról értesít, hogy kísérlet történt a számítógépen futó szolgáltatások feltérképezésére, mégpedig oly módon, hogy az összes portot végig próbálta az illető, hogy kiderítse, melyek vannak nyitva.

Ha ilyen riasztást kapunk, először is azonosítanunk kell a port scan forrását. Hasonlítsuk ezt össze a célszámítógépen futó szolgáltatásokkal. Ellenőrizzük az access logokat, találunk-e bennük jogosulatlan hozzáférésre utaló jeleket. Ha találunk ilyeneket, ellenőrizzük a kiszolgálót, hogy minden rendben van-e *(Ha nem találunk akkor is! Biztos, ami biztos.)* Ha a csomagszűrés be van kapcsolva, és az ajánlásoknak megfelelően van beállítva, csak a valamilyen célból megnyitott portok *(például közzétett kiszolgálók)* fognak válaszolni a port scan-re.

IP half scan attack

Ez a riasztás azt jelenti, hogy ismételt kísérletek történtek a TCP csatorna kiépítésére *(vagyis a TCP handshake, vagy ha úgy tetszik „chip-chip-choka" lebonyolítására)* egy távoli számítógéppel, de a távoli gép valamiért „elfelejtette" elküldeni a csatorna kiépítését befejező ACK csomagokat. Jó, jó. Biztosan feledékeny! De miért baj ez? Egy szabványos transmission control protocol *(TCP)* kapcsolat felépítése egy SYN csomag célszámítógép felé küldésével kezdődik. Ha a megcélzott vár kapcsolatot az adott porton, akkor egy SYN/ACK csomaggal válaszol. A kapcsolatot kezdeményező fél egy ACK csomaggal válaszol, és már fel is épült a kapcsolat. Ha a megcélzott gép nem vár kapcsolatot az adott porton, egy RST csomaggal fog válaszolni.

A legtöbb rendszer nem naplózza a lezárt kapcsolatokat, egészen addig, amíg az utolsó ACK csomag is meg nem érkezett a forrástól. Egy RST csomag utolsó ACK helyetti küldésével a kapcsolat sose épül fel, és éppen ezért a kapcso-

lat nem is lesz naplózva. Mivel a kapcsolatot kezdeményező könnyedén megállapíthatja, hogy a célgép SYN/ACK vagy RST csomagot küldött, azt is könnyedén megállapíthatja, hogy pontosan mely portok vannak nyitva, és mivel nem küldi el az utolsó ACK csomagot, az áldozat nem is fogja észrevenni a kísérletezést. Kivéve persze ha ISA Server-e van, és be van kapcsolva az Intrusion Detection!
Ha ilyen eseményt tapasztalunk, vessük jól az agyunkba, meg persze egy tiltó csomagszűrő szabályba a mókás kísérletező IP címét.

Land attack

Ez a riasztás arról tájékoztat, hogy olyan TCP SYN csomagot kaptunk, amelynek meghamisították a forrás IP címét és port számát, hogy megegyezzen a megcélzott IP címmel és port számmal. Ha a támadás sikerrel jár, egyes TCP megvalósításoknál hurkot eredményez, melynek eredményeként a számítógép bemutatja a kék halálnak megfelelő cselekvéssorozatot.

Ping of death attack

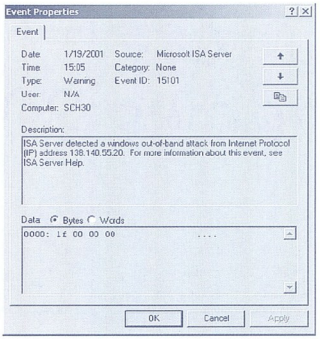
Valaki nagy mennyiségű adatot helyez el egy Internet Control Message Protocol *(ICMP)* echo request *(ping)* csomagban. Ha a támadás sikeres, egy kernel buffer túlcsordul, amikor a számítógép megpróbál válaszolni. Az eredmény ugye egyértelmű. Ha ilyet tapasztalunk, tiltsuk az Internet felől érkező ICMP echo request csomagokat. Alapértelmezésben ezek tiltva is vannak.

UDP bomb attack

Ez a riasztás azt jelenti, hogy valaki megkísérelt egy érvénytelen User Datagram Protocol *(UDP)* csomagot küldeni. Ez némely régebbi operációs rendszernél „kék halált" okoz, amikor megkapja a csomagot. Az okot igen nehéz megállapítani.
A további támadások a küldő IP cím blokkolásával megelőzhetők.

Windows out-of-band attack

A riasztás arról értesít, hogy egy out-of-band DoS támadást kíséreltek meg egy számítógép ellen, amit az ISA Server véd. A sikeres támadás eredménye: DoS.
A további támadások itt is a küldő IP címének blokkolásával védhetők ki.
Ha egy lehetséges behatolási kísérletet észlelt, az ISA riasztása elindíthat egy adott cselekvéssorozatot, amely az esemény naplózásától akár az ISA Server kikapcsolásáig terjedhet. A leginkább javasolt eljárás olyan riasztás küldése, ami a lehető leghamarabb magára vonja a rendszergazda figyelmét. A behatolásészlelés riasztása által az application logba írt események jól jönnek a behatolási kísérletek elemzésekor, és az azokra történő válaszadáskor. Az alábbi ábrán egy Windows out-of-band attack által generált eseménynapló-bejegyzés látható. Mint látható, a forrás IP cím naplózva van.



Behatolás nyoma az eseménynaplóban!

Fontos tudnunk, hogy a behatolásészlelő technológiák egyáltalán nem „bolondbiztosak", és ez alól az ISA sem kivétel. őszintén szólva az ISA Server elég szerény kísérletet tesz a behatolásészlelés megvalósítására, tehát ne ringassuk hamis biztonságérzetbe magunkat. A próbálkozás azért mindenesetre dicséretes.

Összegzés

Összegezve a fent leírtakat, a következő fontos ajánlásokat érdemes mindig betartani, a csomagszűrés, és a hozzá kapcsolódó behatolás-észlelés beállításakor:

- Kapcsoljuk be a csomagszűrést.
- Kapcsoljuk be az IP fragment-ek szűrését.
- Kapcsoljuk be az IP options szűrését.
- Kapcsoljuk ki a H.323 bővítést, ha nincs rá szükségünk.
- Kapcsoljuk be a behatolásészlelést, főleg akkor, ha nincs erre külön rendszerünk.
- Ha bekapcsoltuk a behatolásészlelést, gondosan állítsuk be az adott behatolási kísérletekhez tartozó riasztást *(és az általa végrehajtott cselekvéssorozatot)*.
- Minden lehetséges eszközzel teszteljük rendszerünk „ütésállóságát". Persze csak tesztrendszerben. Éles rendszerben csak olyannal kísérletezzünk, ami biztosan nem okoz kárt. Az eredményt használjuk fel a lehető legszigorúbb szabályrendszer kialakításához.

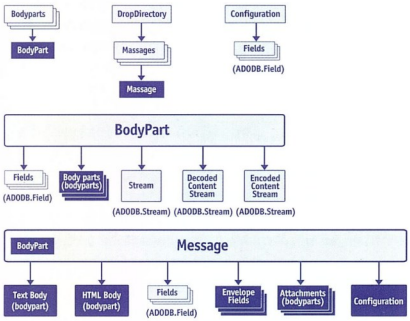
BP
MCSE

ASP Suli *(IX.rész)* CDO for Windows 2000

Az ASP Suli előző részében megismertük a kissé már poros CDONTS objektummodellt, most a friss, ropogós változat, a CDO for Windows 2000 következik. Az elmúlt havi cikkhez hasonlóan, a CDO for W2K sem szigorúan ASP objektummodell, bárki használhatja, aki a programozás során hozzáfér COM objektumokhoz *(scriptből, vagy akár programokból is)*. Mint mindig, az aktuális példakódok letölthetők az **[1]** címről.

A CDO for Windows 2000 objektummodellje

Az alábbi ábrán a CDO for W2K objektummodellje látható *(a teljes dokumentáció a weben a* **[2]** *címen található)*.



A CDO for Windows 2000 objektummodellje

Első pillantásra talán rémisztőnek látszik, de valójában nem az. Egy nagyon okosan megkomponált, könnyen érthető objektummodellel állunk szemben, aminek bizonyos elemei ráadásul nem is tartoznak a CDO for Windows 2000 részei közé: a levelek, levélrészek egyes részeit képező *(akár bináris)* adatfolyamok kezelését, azok betöltését-mentését az ADO *(Active Data Objects)* Stream objektum végzi. Az ADODB.Stream objektumot az ASP Suliban már többször használtuk, de akit komolyabban érdekel a téma, lapozza fel a weben az objektum referenciáját (**[3]**). Az objektum használata a példaprogramokban önmagáért beszél majd *(persze angolul :-))*. A másik, ADO-ból kölcsönvett objektum az ADODB.Field, amely tulajdonképpen nem más, mint egy névvel azonosítható tárolóobjektum. Ilyen Field objektumokat tartalmaznak a CDO for W2K objektummodelljének számos pontján megtalálható Fields kollekciók. A Stream-hez hasonlóan a Field objektum használata sem okozhat majd gondot, de a biztonság kedvéért bárki fellapozhatja a weben a hivatalos referenciát (**[4]**).

Levélküldés hirtelen felindulásból

Ha nem is egy sorból mint a CDONTS-ben, de gyakorlatilag egyetlen objektum létrehozásával is kitöltögetésével a CDO for W2K is képes levél küldésére *(cdo1.asp)*:

```
Set oMsg = Server.CreateObject("CDO.Message")
oMsg.From =
  """Fu Beno"" <beno@falatrax.hu>"
oMsg.To =
  """Gipsz Jakab"" <gipszj@falatrax.hu>"
oMsg.Subject = "CDO demo 1."
oMsg.TextBody = "Hello CDO for Windows 2000!"
oMsg.Send
```

Azaz: létrehozzuk az új üzenetet jelképező CDO.Message objektumot; kitöltjük a feladó és a címzett mezőket; megadjuk a levél témáját, a szöveges törzs tartalmát, végül meghívjuk a Send*()* metódust. Ami feltűnhet, az a címzés módja: szándékosan nem az egyszerű e-mail címeket adjuk meg, hanem az SMTP szabványnak megfelelően a delikvensek teljes nevét is *("név" <e-mail cím> formában)*. A másik dolog, ami feltűnhet, maga a cím: nem reklám, de utóbb kiderült, hogy az előző számban használt példaprogramok a Teszt Magazin nem létező felhasználóinak küldtek csini próbaleveleket *(@teszt.hu; az érintettektől ezúton kérünk elnézést :-)))*. A @falatrax.hu domain garantáltan nem létezik :-).

A Configuration objektum

A Configuration objektum a levelek elküldésének, kezelésének módját meghatározó beállításokat tartalmazza. Az általános alapértelmezés az alábbi:

- Ha van a gépen SMTP szolgáltatás, a CDO for W2K a kimenő leveleket az SMTP szolgáltatás PickUp könyvtárába másolja; a bejövő leveleket a szolgáltatás Drop könyvtárából veszi *(figyeljük meg, hogy ezesetben a CDO maga semmiféle hálózati kommunikációt nem végez)*.
- Ha nincs, megpróbálja a gépen található Outlook Express beállításait használni.

Minden létrehozott üzenetobjektum *(CDO.Message)* ezeket a beállításokat örökli, de ha akarjuk, a Configuration objektum segítségével ezt a viselkedést akár üzenetenként megváltoztathatjuk. A CDO.Configuration objektum kétféle képpen jöhet létre:

- Új, üres objektumként létrehozzuk, hogy majd később egy Message objektumhoz rendeljük:

```
Set oCnf =
  Server.CreateObject("CDO.Configuration")
```

- Egy létező Message objektum alapértelmezett Configuration mezőjét lekérdezve megkapjuk az üzenetre érvényes beállításokat tartalmazó példányt:

```
Set oMsg = Server.CreateObject("CDO.Message")
Set oCnf = oMsg.Configuration
```

A Configuration.Load() metódus segítségével az objektumba be *(vagy újra-)* tölthetjük az alapértelmezett beállításokat *(a metódus paramétere az, hogy honnan)*:

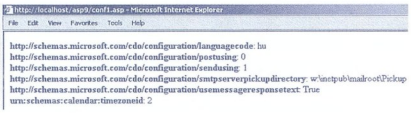
```
oCnf.Load(-1)  ' az alapértelmezések betöltése
oCnf.Load(1)   ' az SMTP Service beállításai
oCnf.Load(2)   ' az Outlook Express beállításai
```

A Configuration objektum ezen kívül egyetlen dolgot tartalmaz, ez pedig egy Fields kollekció, amely a beállításokat tartalmazza *(emlékszünk, sok-sok ADODB.Field objektumot)*. A conf1.asp kódja az alábbi:

```
Set oMsg = Server.CreateObject("CDO.Message")
Set oCnf = oMsg.Configuration

For Each oField In oCnf.Fields
  Response.Write "<b>" & oField.Name & "</b>: "
↳  & oField.Value & "<br>"
Next
```

A példakódból szépen látszik a Field objektumok használatának módja *(tulajdonképpen két fontos dolgot kell megjegyeznünk róluk: a mező nevét a .Name; értékét pedig a .Value jellemző tárolja)*. A For Each ciklus segítségével kilistázzuk a Configuration objektumban található összes mezőt és azok értékeit. Az eredmény:



A CDO alapbeállításai

Ne ijedjünk meg a mezők nevétől *(a sorok vastag betűs része)*, ez csak a manapság divatos névteresítés eredménye *(a CDO for W2K összes mezőjét ilyen nevekkel érhetjük el, hacsak nincsenek kivezetve, mint valamelyik objektum jellemzője)*. A Configuration mezők neve és azok jelentése egyébként a CDO referenciából kileshető (**[5]**). Ezek szerint például:

- A válaszüzenetek generálásakor a magyar nyelvet használjuk *(languagecode = hu)*.
- Az SMTP leveleket a helyi SMTP kiszolgáló Pickup könyvtárába másolva „küldjük el" *(sendusing = 1)*.
- Az SMTP kiszolgáló Pickup könyvtára *(smtpserverpickupdirectory)*.
- Válasz írásakor és levelek továbbításakor automatikusan generáljuk a válaszüzenet tartalmát *(usemessageresponsetext = True)*.

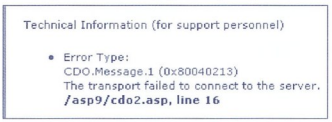
Állítsuk be, hogy a CDO ne *(fájlműveletekkel)* a helyi, hanem egy „távoli" SMTP kiszolgálót használjon *(cdo2.asp)*. *(Ez persze lehet a saját gépünk is, csak most hálózaton keresztül, „kívülről" érjük el az SMTP kiszolgálót.)* Ehhez az üzenetobjektum létrehozása után az alábbi néhány sor beszúrására van szükség:

```
Set oCnf = oMsg.Configuration
oCnf.Fields("http://schemas.microsoft.com/cdo/
↳  configuration/sendusing") = 2 'sendUsingPort
oCnf.Fields("http://schemas.microsoft.com/cdo/
↳  configuration/smtpserver") =
↳  "mail.netacademia.net"
oCnf.Fields.Update
```

A többszörös sortörésre a hosszú mezőnevek miatt volt szükség *(ahol a típuskönyvtár rendelkezésre áll, ott használhatjuk a mezők rövid neveit, lásd pl.:* **[6]**). De lássuk csak, mit tettünk:

- Az oCnf objektumba betöltöttük az üzenetobjektum konfigurációját.
- A „sendusing" mezőnek 2 értéket adtunk, ez azt jelenti, hogy hálózaton keresztül küldjük a levelet.
- Az „smtpserver" mezőben megadtuk a használni kívánt SMTP kiszolgáló címét *(ez lehetne egyébként NetBIOS név vagy IP cím is)*.
- Végül a legfontosabb: ha azt akarjuk, hogy a mezőkön végzett változtatások érvényre jussanak, a módosítások után meg kell hívnunk a Fields kollekció Update() metódusát! *(Ez a szabály a CDO összes mezőkollekciójára érvényes)*.

Ha megtehetjük, próbáljuk ki mi történik, ha a cdo1.asp-t és a cdo2.asp-t akkor futtatjuk, amikor a célbavett SMTP kiszolgáló éppen nem érhető el! A Pickup könyvtárat használó cdo1.asp simán lefut *(a levél bekerül a Pickup könyvtárba, és az SMTP Server újraindításakor el fog menni)*, míg a cdo2.asp hibát jelez:



Ha nem a helyi SMTP kiszolgáló Pickup könyvtárát használjuk, fel kell készülnünk a hálózati problémákra

Ezért írtam az előző számban, hogy a legtöbb esetben kénytelmesebb, ha a postázás kínját-baját az SMTP kiszolgálóra bízzuk. De van ennél cifrább is, például ha a levelező kiszolgáló tiltja a relay-zést *(rajta keresztül nem, vagy csak bejelentkezés után enged idegen tartományba levelet küldeni)*, a következő sokatmondó hibát kapjuk:

Ha a hálózati problémákat megoldottuk...

A CDO az ilyen problémák elkerülése érdekében képes bejelentkezni a távoli SMTP szolgáltatásba. Ezt megint kétféleképpen tehetjük meg:

- Ha a távoli SMTP kiszolgáló Windows, a CDO for W2K képes biztonságos NTLM authentikációval bejelentkezni. Ilyenkor csak egyetlen mezőt kell beállítani, a bejelentkezéshez az aktuális felhasználó adatait használja fel a rendszer.

```
oCnf.Fields("http://schemas.microsoft.com/cdo/
↳ configuration/smtpauthenticate") = 2 ' cdoNTLM
```

- Ha ez nem menne (vagy az SMTP kiszolgáló nem Windows), marad a nyíltjelszavas azonosítás. Ilyenkor meg kell adnunk a felhasználónevet és jelszót is *(ne felejtsük el az Update() meghívását!)*:

```
oCnf.Fields("http://schemas.microsoft.com/cdo/
configuration/smtpauthenticate") = 1 ' cdoBasic
oCnf.Fields("http://schemas.microsoft.com/cdo/
configuration/sendusername") = "usernév"
oCnf.Fields("http://schemas.microsoft.com/cdo/
configuration/sendpassword") = "jelszó"
```

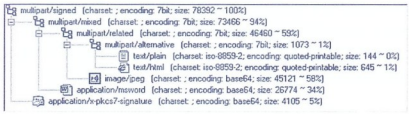
Ennek két buktatója van: egyrészt, ilyenkor a felhasználónév és jelszó a hálózaton titkosítatlanul utazik; másrészt az .asp kód önmaga tartalmazza a felhasználónevet és jelszót, ami – bár normális esetben a felhasználó az ASP oldal kódját nem láthatja – egyáltalán nem jó gyakorlat. Az első problémára megoldást jelenthet az, ha a levelező kiszolgáló támogatja az SSL csatornán keresztüli kommunikációt, bekapcsolhatjuk azt, és akkor a teljes levélküldés – az esetleges bejelentkezéssel együtt is – titkosított csatornán halad:

```
oCnf.Fields("http://schemas.microsoft.com/cdo/
configuration/smtpusessl") = True
```

Egy érdekes jellemzőre szeretném felhívni a figyelmet: előfordulhat, hogy hiába adjuk meg, hogy nyílt jelszavas felhasználóazonosítást szeretnénk; a CDO for W2K lekérdezi a kiszolgálót, és ha az támogatja a jóval biztonságosabb NTLM azonosítást, azt fogja használni!

Az üzenetobjektum *(CDO.Message)*

A konfigurációs lehetőségek után most lássuk a – korábban már egy példa erejéig használt – CDO.Message objektumot, ami egy üzenetet „testesít" meg. Mint tudjuk, az SMTP levél *(ha az MIME)*, belülről egy fastruktúra, ami a levél tartalmától függően lehet akár egyetlen *(fa)*levél *(tipikus SMTP levél, tiszta szöveges tartalommal)*, de lehet üzenetrészek bonyolult hierarchiája is. A már sokszor emlegetett MIME Browser képes nekünk grafikus formába önteni a MIME levelek tartalmát:



Egy MIME levél néha bonyolult stuktúrába szervezett üzenetrészekből áll

Azt már sejtheti az Olvasó, hogy az üzenet nem lesz más, mint üzenetrész *(BodyPart)* objektumok többszintű kollekciója *(lásd később)*, de mintha a pár oldallal ezelőtt a Message objektum gyors bemutatásakor nem lett volna szó üzenetrészekről. Címzés, téma, tartalom, és kész.

A magyarázat: a belső felépítéstől függetlenül minden levélnek valahol van szöveges „törzse" *(a fenti ábrán például a legalsó szinten látható text/plain üzenetrész tartalmazza)*. A levelekhez csatolhatunk fájlokat *(a fenti ábrán például az application/msword üzenetrész ilyen)* is. A Message objektum ezért – miközben részletekbe menően felboncolhatjuk – tartalmaz „kivezetett" részeket, csak hogy megkönnyítse a programozó dolgát. Az élveboncolás előtt lássuk tehát a fontosabb „shortcut" mezőket *(IMessage interfész, lásd még:* **[7]***)*:

- .From = a levél feladója.
- .To = a levél címzettje(i). Ha egynél több címet adunk meg, azokat vesszővel kell elválasztani.
- .Cc, .Bcc = másolat, vakmásolat. A címzettek megadását lásd fennt.
- .Subject = a levél témája.
- .TextBody = az üzenet szöveges törzse.
- .MIMEFormatted = True értéke jelzi, hogy az üzenet MIME formátumú.
- .HTMLBody = az üzenet HTML törzse *(szöveg)*.
- .AutoGenerateTextBody = True *(alapértelmezett)* értéke esetén, ha HTML levelet írunk *(.HTMLBody)*, a CDO automatikusan legenerálja hozzá a megfelelő tiszta szöveges változatot *(.TextBody)*.
- .MDNRequested = True értéke jelzi, hogy a levélről kérünk olvasási, kézbesítési visszajelzést, lásd még:
- .DSNOptions = értéke attól függ, hogy milyen típusú visszajelzést szeretnénk kapni. Az egyes számértékek összeadhatók, jelentésük pedig: 0,1 – nincs visszajelzés; 2 – visszajelzés kézbesíthetetlenség esetén; 4 – visszajelzés sikeres kézbesítés esetén; 8 – visszajelzés, ha a kézbesítés átmenetileg nem lehetséges *(késleltetett)*.
- .SentOn, .ReceivedTime = Ez a két dátummező csak akkor él, ha nem küldendő, hanem kapott leveleket dolgozunk fel. Értékük ilyenkor a levél küldésének illetve megérkezésének dátuma.

A Message objektum Fields kollekciója az üzenet fejlécében található mezők, beállítások tartalmát, valamint néhány röptében generált adatot tartalmaz. Az msgfields.asp betölti a html1.eml fájlba mentett levelet, és kilistázza a Fields kollekció tartalmát. Ha nézegetjük, észrevehetjük, hogy a mezők nagyjából leképezik az üzenet fejrészében található fejléceket. Ami feltűnhet:

- Az „urn:schemas:httpmail:htmldescription" mező a levél törzsének HTML; a „urn:schemas:httpmail:textdescription" pedig a tiszta szöveges változatát tartalmazza *(ez a Message objektum .HTMLBody illetve a .TextBody jellemzőinek értéke)*.
- A „urn:schemas:mailheader:x-sajatfejlec:" mező és annak értéke. Az X-SajatFejlec fejlécet mi csempésztük az elmentett .eml fájl fejrészébe, a CDO pedig szépen betölti azt. Ez visszafelé is igaz: ha egy üzenetobjektumban létrehozunk egy „urn:schemas:mailheader:x-akármi" nevű mezőt, az üzenet fejrészében megjelenik, mint fejléc.

A Message objektum .EnvelopeFields kollekciója csak az Event Sink-ek programozásánál használható, nekünk egyelőre nincs szükségünk rá. Ami még érdekes lehet, az az üzenetrész-objektumokat, illetve azok kollekcióit visszaadó jellemzők, úgyismint:

- .BodyPart = az üzenet fő üzenetrésze *(a MIME fa törzse)*; maga az üzenet
- .TextBodyPart = az üzenet szöveges törzsét tartalmazó üzenetrész-objektum
- .HTMLBodyPart = az üzenet HTML törzsét tartalmazó üzenetrész-objektum

Új üzenetrészeket a Message objektumon keresztül *(merthogy máshogy is lehet, lásd később)* a következő két metódus segítségével adhatunk az üzenethez:

- AddAttachment*()*: egy csatolt fájlt ad az üzenethez
- AddRelatedBodyPart*()*: egy üzenetkomponenst ad az

üzenethez *(amit az üzeneten belül felhasználhatunk, de mint csatolt fájl, nem látható)*.

Lássunk egy-egy példát mindkét esetre, először egy csatolt fájl képében *(attach.asp)*:

```
oMsg.HTMLBody = "<b>Girl: attached</b>"
oMsg.AddAttachment Server.MapPath("msgirl.jpg")
```

Az AddAttachment() metódusnak egyébként nem csak a csatolandó fájl nevét, hanem URL-jét is megadhatjuk *(ld. attach2.asp)*, ilyenkor a CDO letölti azt és csatolja a levélhez. *(Az esetleg szükséges felhasználónevet és jelszót a második és harmadik paraméterként adjuk át)*.

Lássuk most, hogy néz ki az, ha a képet nem a HTML levélhez csatolva, hanem abba beillesztve szeretnénk elküldeni *(related.asp)*:

```
oMsg.HTMLBody = "img src=""cid:pic.jpg""">"
oMsg.AddRelatedBodyPart
  Server.MapPath("msgirl.jpg"), "pic.jpg", 0
```

Kezdjük hátulról: az .AddRelatedBodyPart metódus három *(öt)* paramétert vár:

- A csatolandó fájl neve vagy URL-je
- Az a név *(ID)*, ahogy a csatolt elemre az üzenetben majd hivatkozunk
- A hivatkozás típusa: lehet ID alapján *(0)*, vagy relatív URL-lel *(1)*
- Az utolsó két paraméter ismét a fájl letöltéséhez esetleg szükséges felhasználónév és jelszó

Az üzenet kimentése fájlba / betöltése fájlból

A cikk elején szó esett már arról, hogy a CDO az adatmozgatáshoz az ADODB.Stream objektumát használja fel. Az üzenet kiírását és beolvasását azonban nem a Message, hanem a DataSource interfészen keresztül kérdezzük le *(egyébként ez az interfész is ugyanarra az objektumra mutat, mint a Message)*. A DataSource interfész tartalmazza a Stream-be *(és másba)* mentéshez, illetve onnan való betöltéshez szükséges metódusokat. Az üzenet elmentése például az alábbi sorokból áll:

```
Set oStream = Server.CreateObject("ADODB.Stream")
oStream.Open
oStream.Type = 1 ' adTypeText
oStream.Charset = "us-ascii"
Set oDS = oMsg.DataSource
oDS.SaveToObject oStream, "_Stream"
oStream.SaveToFile "file.eml", 2
'adSaveCreateOverwrite
```

Azaz, sorrendben: létrehozunk egy Stream objektumot; megnyitjuk azt, majd beállítjuk, hogy szöveges adatokat szeretnénk tárolni, azt is ASCII kódolásban *(hiszen az SMTP levél nem is tartalmazhat mást)*. Ezután a CDO Message objektum .DataSource jellemzőjének segítségével lekérdezzük az üzenet DataSource objektumát. A DataSource objektum .SaveToObject() metódusának első paramétere a célterület *(esetünkben a Stream objektum)*, második paramétere pedig a célobjektum típusazonosítója *(ez Stream esetén „_Stream")*. Ezután már csak a Stream objektum SaveToFile() metódusát kell meghívni, és máris kész az elmentett üzenet. Az üzenet betöltése ennek inverze, valahogy így *(az msgfields.asp példaprogramban megtalálható)*:

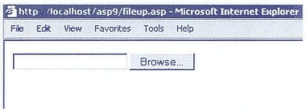
```
Set oStream = Server.CreateObject("ADODB.Stream")
oStream.Open
oStream.LoadFromFile Server.MapPath("html1.eml")
Set oMsg = Server.CreateObject("CDO.Message")
Set oDS = oMsg.DataSource
oDS.OpenObject oStream, "_Stream"
```

Fájlok feltöltése a webkiszolgálóra

Szép-szép, mondhatná a kedves olvasó, de hogyan kerülnek fájlok a kiszolgálóra? Ez központi kérdés a webprogramozás során, és nem csak a levelezés kapcsán: előbb-utóbb minden projektben feladat lesz a feltöltött fájlok fogadása. Ha megnézzük az évek óta érvényes HTML4 szabványt *(mellesleg ezt illik is nézegetni:* **[8]***)*, feltűnhet egy különleges HTML elem:

```
<INPUT type="file" name="filename">
```

„File" típusú <INPUT> mező! Ha pedig megjelenítjük, egészen érdekesen jelenik meg a böngészőben:



A „file" típusú <INPUT> mezővel már kereshetünk fájlokat a lemezünkön

Hurrá, hiszen ezzel lehet böngészni! Nosza, próbáljuk ki *(fileup1.asp)*! Válasszunk ki egy fájlt, majd küldjük el és nézzük, mi érkezik a kiszolgálóhoz... Ha jó fájlt néztünk, akkor bizony csak a fájl neve és elérési útja *(a kliensen!)*. Hát, ez még nem tökéletes. Azt is olvashatjuk, hogy ennek az elemnek a használatakor a <FORM> HTML elem enctype paraméterét át kell állítani, valahogy így:

```
<FORM method="post" action="fileup2.asp"
enctype="multipart/form-data">
```

Próbáljuk ki ezt *(fileup2.asp)*, hátha nagyobb szerencsével járunk! Jobb lett? Dehogyis... most még a fájl neve sem jött át... vagy mégis? Lessük meg valahogy a hálózati forgalmat, ami a böngésző és a kiszolgáló között zajlik, és valami ilyesmit látunk majd:

```
Content-Type: multipart/form-data;
  boundary=----------------------7d12443018907e2

----------------------7d12443018907e2
Content-Disposition: form-data; name="filename";
  filename="C:\teszt.txt"
Content-Type: text/plain

Ez egy teszt uzenet.
Ezt a fajlt probaljuk meg feltolteni a kiszolgalora.
Vajon sikerrel jarunk?
----------------------7d12443018907e2-
```

Ismerős? Ugye feltűnt, hogy a böngésző a kérdőív postázásakor egy korrekt MIME üzenetet hozott létre, Content-Type fejléccel, határolókarakterekkel, az egyes részek saját fejlécével *(Content-Disposition)*, adattípusával? És persze, megkaptuk a fájl teljes tartalmát – pontosabban nem mi, hanem a webkiszolgáló.

Most már csak keresni kellene valamit, ami képes ebből a MIME forgatagból intelligensen kikeresni a feltöltött adatokat *(plusz a kérdőív további mezőinek adatait, ha esetleg voltak, hiszen azok is ebbe az üzenetbe kerülnének)*.

A Microsoft ehhez kiadott egy ingyenes komponenst *(cpshost.dll)*, de azt körülményes használni. Ehelyett – és ez itt lehetne akár a reklám helye – bemutatunk egy ugyancsak ingyenes, viszont ügyes és könnyen használható komponenst:

Az aspSmartUpload komponens

A **[9]** címről ingyenesen letölthető komponens segítségével könnyen feldolgozhatjak a feltöltött adatokat. A komponens telepítése egyszerű: töltsük le a tömörített állományt, és a benne található két .dll-t másoljuk olyan helyre, ami benne van a path-ben! Ezután már csak egy dolgunk van, beregisztrálni a komponenst. Menjünk az aspsmartupload.dll-t tartalmazó könyvtárba és adjuk ki az alábbi parancsot:

```
regsvr32 aspsmartupload.dll
```

Az ASPSmartUpload használata egyszerű *(a weben elérhetjük az objektummodell teljes referenciáját és dokumentációját:* **[10]***)*. Először hozzuk létre az objektumot, majd töltsük bele a kérdőívből kapott adatokat *(form.asp és aspsmart.asp)*:

```
Set oAS = Server.CreateObject
  ("aspSmartUpload.SmartUpload")
oAS.UpLoad
```

Ezután már a kezünkben van az irányítás. Lássuk például, milyen fontosabb jellemzői vannak a SmartUpload objektumnak:

- .TotalBytes = az összes feltöltött adat mérete *(fájlok és a form többi eleme együttesen)*
- .TotalMaxFileSize = a feltölthető fájlok maximális mérete összesen
- .MaxFileSize = a feltölthető fájlok maximális mérete egyenként
- .AllowedFilesList = az elfogadott fájlok kiterjesztéseinek listája *(pl. „txt,doc,xls")*
- .DeniedFilesList = a fel nem dolgozandó fájlok kiterjesztéseinek

Az utolsó négy korlátozó jellegű beállítás, még az .UpLoad() metódus meghívása előtt állítsuk be. A fontosabb metódusok a következők:

- .UpLoad(): betölti a form adatait az objektumba
- .Save(): elmenti az összes feltöltött fájlt a megadott könyvtárba

Az objektum kollekciói pedig:

- .Files = a feltöltött fájlok listája *(File objektumok)*
- .Form = a kérdőív mezőinek listája *(Item objektumok)*

A File és az Item objektumok

Az aspsmart.asp példaprogramban bemutatjuk az objektumok használatának módját, most csak néhány fontos jellemzőjükre térünk ki. Először lássuk a File-t:

- .Name = az <INPUT> elem neve
- .FileName = a fájl megadott neve *(pl. attach.txt)*
- .ContentType = a fájl MIME tartalomtípusa
- .IsMissing = értéke True, ha az <INPUT> objektum nem tartalmaz feltöltött fájlt *(mindig ellenőrizzük!)*

Az Item objektumról csak annyit érdemes megjegyezni, hogy a .Values jellemzője egy vagy több értéket tartalmazhat. Feldolgozás előtt kérdezzük le a számukat *(erre is láthatunk példát az aspsmart.asp fájlban)*.

A fájlokkal viszont dolgunk van: ha már feltöltötték, fel kellene őket dolgozni. Például elmenthetjük fájlként *(a File objektum .SaveAs(„fájlnév") metódusával)* egy átmeneti könyvtárba, és ez nekünk pontosan kapóra is jön! Készítsük hát el és próbáljuk ki a csatolt fájlt is kezelni képes levélküldő komponensünket *(newmail.asp)*!

A *(jó-jó, kezdetleges)* példaprogrammal kapcsolatban két dolgot kell megjegyezni:

- Az aspSmartUpload objektum .UpLoad() metódusa hibát ad vissza, ha üres kérést *(nem POST-ot)* akarunk vele feldolgoztatni. Ezért mielőtt meghívnánk, ellenőrizzük a HTTP kérés típusát a Request.ServerVariables *(„REQUEST_METHOD")* segítségével
- A példában a Windows Temp könyvtárába mentjük átmenetileg a feltöltött fájlokat, majd megpróbáljuk azokat törölni *(általában az IUSR_ felhasználónak a törléshez nincs joga)*. Éles rendszerben alakítsunk ki egy megfelelő könyvtárat csak erre a célra, megfelelő jogosultságokkal, de a webről közvetlenül el nem érhető helyen.

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:

[1] http://technet.netacademia.net/download/asp/9
[2] http://msdn.microsoft.com/library/en-us/cdosys/html/_cdosys_cdo_for_windows_2000_object_model.asp
[3] http://msdn.microsoft.com/library/en-us/ado270/htm/mdobjstream.asp
[4] http://msdn.microsoft.com/library/en-us/ado270/htm/mdobjfield.asp
[5] http://msdn.microsoft.com/library/en-us/cdosys/html/_cdosys_schema_configuration.asp
[6] http://msdn.microsoft.com/library/en-us/cdosys/html/_cdosys_cdoconfiguration_module.asp
[7] http://msdn.microsoft.com/library/en-us/cdosys/html/_cdosys_interfaces.asp
[8] http://www.w3.org/TR/html4/
[9] http://www.aspsmart.com/
[10] http://www.aspsmart.com/liblocal/docs/en/aspsmartupload/help/Objects.htm

Még mindig fuss Forrest! – SQL optimalizálás

Bevezetés

Előző számunkban áttekintettünk néhány tervezési fondorlatot gyors SQL Server adatbázisok kialakításának reményében. Ott utalást tettem arra, hogy a fáradságos munkával, trigger segítségével kialakított denormalizálást az SQL Server is képes megvalósítani, számunkra minimális munkával. A titok nyitja az indexelhető nézetekben keresendő. Erről, valamint az indexek működéséről, belső felépítéséről és alkalmazásukról fogok ebben a részben értekezni.

Miért kell az SQL Servernek index?

Nem kell. Az SQL Server jól elvan indexek nélkül is, és végrehajt bármilyen parancsot anélkül, hogy egy árva indexet kapna. Indexek használatával viszont jelentősen felgyorsíthatók az adatbázisműveletek, és sokszor a forrásadatok mennyiségétől független sebességű adatbázisparancsokat hajthatunk végre. Hogy legyen összehasonlítási alapunk, először nézzük meg hogyan tárolja az SQL Server a táblák adatait indexek nélkül.

Adatbázisok - alulnézetben

Leegyszerűsítve nézzük meg az SQL Server adattárolási struktúráját. Az SQL Server a 7-es verzióval kezdve közönséges operációs rendszer fájlokban tárolja az adatbázisok adatait. A fájlokon belül az alapvető tárolási egység a lap *(page)*. Egy lap 8 kByte méretű, és ez a legkisebb mozgatható egység az adatbázisfájlok és a memória között. A táblák sorai lapokon tárolódnak, a sorok nem ívelhetnek át lapok között, azaz el kell nekik férniük egy lapon. Emiatt a maximális sorhossz 8196 Byte mínusz némi adminisztráció, a végeredmény az, hogy 8060 Byte marad a táblák sorainak tárolására.

A lapokat az SQL Server nyolcas csoportokba fogja, és ezt a formációt extent-nek hívja. Amikor a szerver lefoglal lapokat egy tábla vagy index részére, akkor ezt mindig extentenként teszi meg - az extent a minimális helyfoglalási egység. Azaz, ha létrehozunk egy táblát, az rögtön kap egy 64k-s extent-et. Ez elég furcsán nézne ki ha lenne 1000 táblánk, mindegyikben csak annyi sort tárolnánk, amennyi elférne egy lapon, ennek ellenére a szerver 1000x64k-t *(64MByte)* lefoglalna, mert minden táblát külön extentre pakolna. Hogy ez ne így legyen, egy extentnek több lakója is lehet, maximum nyolc objektum *(tábla vagy index)* élhet benne. Ők a mixed extent-lakók.

Amint egy tábla adatai már nem férnek bele egy extent-be, az SQL Server allokál egy újabb extentet neki, amit azután kizárólag ez a tábla birtokolhat. Ezt nevezik uniform extentnek.

Minden adatbázisfájl elején van egy bevezető lap, amit File Header-nek hívnak. Ez az adatbázisfájllal kapcsolatos általános adminisztrációs információkat tartalmaz.

A helyfoglalás nyilvántartásához az SQL Server saját adminisztrációs lapokat helyez el az általunk tárolni kívánt hasznos lapok előtt és között. Minden fájl elején *(a 2. és a 3. pozíción)* van két lap, egy Global Allocation Map *(GAM)* és egy Shared Global Allocation Map *(SGAM)*. Mindkettő egy bittérkép az extentek foglaltságáról *(egyetlen biten ábrázolja egy extent, azaz 64k foglaltságát!)*, és egyszerűen azt írják le, hogy egy extent szabad-e, részben foglalt *(mixed extent és van rajta szabad lap)*, vagy teljesen foglalt *(uniform extent vagy teli mixed extent)*. Mindkettő egy lapnyi, azaz 8192 Byte-nyi információt tud tárolni, azaz kb. 64000 extent-ről képesek nyilvántartást vezetni. Ez 64000 x 64k *(extenteket, nem lapokat tart nyilván)* = 4GByte. Azaz, ha egy adatbázisfájl mérete 4 GByte fölé megy *(nem ritka)*, akkor betelik az első GAM és SGAM.

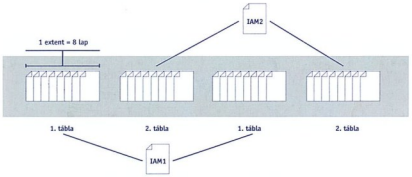
Mit tesz ilyenkor a szerver? Az adatbázisállomány végén lefoglal magának egy újabb adminisztrációs célokra fenntartott extentet, feltölt egy újabb GAM-ot és SGAM-ot, és ezen kezdi el jegyezni az újabb 4 Giga extentjeit. Így élnek, míg meg nem halnak...

A Page Free Space *(PFS)* nevű lap *(lásd előző ábra)*, azt tárolja, hogy az általa leírt lapok mennyire telítettek. Minden egyes byte egy lapot ír le, így egy PFS 8000 lapot képes nyilvántartani. Emiatt ő meg 8000 laponként ismétlődik a fájlokban.

Miért ez a sok redundáns információ az adatblokkok foglaltságáról? Hisz a PFS teljes egészében elég lenne, a GAM és SGAM nem mond többet nála. Azért, mert a GAM-SGAM páros pont a durva felbontása miatt rettenetesen nagy tartományokról ad képet az adatbáziskezelőnek, így két GAM-nyi, azaz 16 kByte-nyi információ felolvasásával azonnal meg lehet mondani 4 GByte-nyi részről, hogy abban hol vannak szabad helyek. Egy intenzíven dagadó adatbázisban gyakran kell allokálni újabb és újabb extenteket, és ilyenkor minimális órajelciklus alatt megtalálja a szabad helyeket az SQL Server. Azután jöhet a PFS azt eldöntendő, hogy a kikeresett extent melyik lapja szabad, vagy van rajta annyi hely, amennyi elég mondjuk a beszúrandó adatnak. Lehet azt mondani, hogy sokkal gyakoribb a módosítás egy adatbázisban, mint a beszúrások száma. Jó, de mi van akkor, amikor a módosítás után egy sor hosszabb lesz, mint előtte volt, és nem fér már el az eredeti lapján? Ilyenkor a szerver helyet keres *(GAM, SGAM, PFS)* és a módosított sort már ott helyezi el, persze az eredeti helyét felszabadítva.

Az extentek és a lapok katonás rendben vannak nyilvántartva. Honnan tudja a szerver, hogy mondjuk a 145345-edik lap melyik táblához vagy indexhez tartozó információt tárol? A GAM-ok és a PFS-ek erről egy szót sem szólnak. Ami-

kor létrehozunk egy táblát vagy egy indexet, akkor az SQL Server létrehoz egy uniform extentet, aminek első lapját lefoglalja a tábla vagy index által használt extentek nyilvántartására. Ezt a lapot hívjuk Index Allocation Map-nek *(IAM)*. A neve azért megtévesztő, mert nem csak az indexek, de a táblák lapjait is ezek tartják nyilván.



Mivel egy IAM csak korlátozott számú extentet tud leírni, az IAM-ok össze vannak láncolva. Minden index-hez vagy táblához létezik egy IAM lánc.

Ez itt egy nagy halom...

Heap. Ez a kulcsszó azokhoz a táblákhoz, amelyekhez nem hozunk létre indexet. Indexek nélkül az IAM az egyetlen logikai kapocs, amely egy tábla adatlapjait összeköti. Ez azt jelenti, hogy ha egy lekérdezést hajtunk végre egy index nélküli, azaz heap *(halom)* struktúrában tárolt táblán, akkor az SQL Servernek fel kell olvasni az IAM*(ok)* által leírt összes adatlapot, és azokban turkálva kell kikeresnie a kért információt. Ha például 10 millió sornyi információt tárolunk egy táblában, egy sor átlagos hossza 200 byte, akkor az SQL Server-nek 2 GByte-nyi adattömeget kell felolvasni a merevlemezről, és végigkeresgélni a memóriában. Tű a szénakazalban *(halomban)*. Ez még akkor is nagyon időigényes, ha az egész tábla a Buffer Cache-ben *(RAM-ban)* van. A keresési módszer neve table scan, és nagy táblák esetén üldözzük tűzzel-vassal. Meg indexszel.

Főzőcske indexekkel

Szeretnék enni egy kis Juhtúróval töltött gombakalapot. Mit tesz ilyenkor a jólnevelt informatikus? Nem, nem kezd el az interneten keresgélni. Az olyan mesterséges dolog. Előveszi Hargitai György Vegetáriánus ételek című könyvét. Felüti a könyvet a közepén. Mit lát ott? Mozarellás sült brokkoli. Ennél egy picit visszább lesz a Juhtúrós gomba. Megfelezi a maradék oldalakat, felüti a könyvet, és a szemébe ötlik a Francia sárgarépa. Töltötuk Béláim – mondaná Kállay Ferenc a Tanúban. Túlságosan előrementünk. Még néhány előrehátra lapozás, és megtalálja a hőn áhított ételt. Mit csinált az éhes informatikus? Clustered index seeket!

A clustered index seek azt jelenti, hogy az elérni kívánt információk már előre be vannak rendezve aszerint a feltétel szerint, ami szerint keresünk benne. Én az étel neve alapján kerestem, és a könyv lapjai pont így vannak berendezve, ezért nagyon gyorsan megtaláltam benne a keresett információt.

Megkívántam egy Pikáns csirkét. Az előző könyvben végrehajtott sikeres clustered index-es keresés után előveszem Péter Sándorné és Nagy Jánosné bestsellerét, az Esély szakácskönyvet. Némi lapozás után csalódottam állapítom meg, hogy nincs benne clustered index, nem az ételek nevei alapján van berendezve. Lehetőségeim: elkezdem lapozni a könyvet, és valahol majd csak kiszúrom a pipit, azaz a table *(könyv)* scant végzek. Nem biztos, hogy kivárnám ezt a csirkét.

A másik lehetőség, hogy fellapozom a könyv végén található tárgymutatót, ahol az ételek már betűrendben vannak. Az előbbi felezős módszeremmel kikeresem a csirkét. Ezt nagyon gyorsan meg tudom tenni, különösen, hogy a tárgymutató csak néhány lap hosszú. A Pikáns csirke a 260-adik oldalon lakik. Mit kezdek ezzel az információval? Megjegyzem ezt a kulcsot, és belekezdek egy újabb felezgetős keresésbe immáron a tényleges adatlapokon, kihasználva azt a tényt, hogy a könyv lapjai a lapok sorszámai alapján vannak bekötve. Viszonylag gyors kereséssel megvan a csirke.

Ez már nem volt olyan egyszerű fogás, mint az előbbi, de egy tétel esetén megéri vele vergődni. Ha viszont azzal bíznának meg, hogy gyűjtsem ki az összes olyan ételt, ami k betűvel kezdődik *(kb. 150 darab)*, akkor inkább azt választom, hogy végiglapozom a teljes ötszáz oldalas könyvet, semmint, hogy állandóan előre-hátra lapozzak a tárgymutató kénye kedve szerint.

Mit hajtottam végre a pipi keresése közben? Nonclustered index seeket! Az adatok nem a keresési feltétel szerint voltak fizikailag berendezve, de kaptunk egy struktúrát *(tárgymutató – index)*, amiben csak a keresendő adatok voltak felsorolva *(indexkulcs)*, és egy-egy a tényleges adatokra mutató pointer.

Egyedi, néhány sornyi adatok esetén nem nagy munka és idő ezen indirekció mentén felszedni a sorokat, de ha sok adatra vagyunk kíváncsiak *(a lekérdezés eredményhalmaza sok sort tartalmaz)*, akkor sokszor kisebb munka végigmenni a teljes táblán, és kikeresni a kért információkat az index használata helyett.

Itt a magyarázat arra, amire sokan panaszkodnak: „Raktam egy indexet a táblára *(nem tud róla, de nonclustered-et)*, ám ennek ellenére az SQL Server nem használja azt, table scan-t csinál!" Hogyne, mert okos, és tudja, hogy tovább tartana az index mentén felszedni a sorokat, mint direktben kikeresni őket. De honnan tudja, hogy egy lekérdezés eredménye hány sort fog eredményezni, hisz a szűrési feltételben lehet akár igen összetett feltétel is?

Nos, van neki egy tanácsadója, akit úgy hívnak: statisztika. Az SQL Server eloszlási információkat gyűjt a tábláinkban tárolt adatokról, és kis hisztogrammok formájában tárolja is őket. A lekérdezések végrehajtása előtt kielemzi a statisztikákat, és azok alapján meg tudja becsülni, hogy nagyjából a forrástáblák hány százalékát fogja érinteni a művelet. Csak nagyjából, mert ezek mintavételezett statisztikai információk, nem egzakt arányok. Miután tudja mekkora részét kell a táblának elérni, és tudja, hogy milyen indexek állnak ehhez a rendelkezésére, ki tudja választani, hogy melyik adatelérési stratégia lesz a legkisebb költségű, és azt fogja választani.

Ezek a döntések igen összetettek is tudnak lenni. Például több tábla JOIN-olása esetén egyáltalán nem mindegy, hogy milyen sorrendben hajtja végre az összekapcsolást. Lehet, hogy az egyik sorrend esetén csak pár száz sornyi információt kell átgyúrnia, míg más sorrendet választva több milliót. 4 tábla esetén 4! *(faktoriális)*, azaz 24 kombinációt kell kielemeznie. 10 tábla esetén 10! = 3628800-at. Mire ezt mind végigelemezné, lemenne a nap, és még nem csinált semmit, csak azon gondolkodott, hogy hogyan fogjon neki a munkának. Ezt a hozzáállást ismerjük embereknél, de az SQL Server nem ilyen. Nagyszámú lehetséges végrehajtási terv esetén heurisztikákat vet be, olyan dön-

téseket hoz, amelyek nem hoznak biztosan jó eredményt, de az esetek többségében igen. Így rengeteg végrehajtási tervet még az elején kidob a pályázók közül, így a végére már csak néhányat kell tényleg kielemeznie.

Mondja azt valaki, hogy az informatika egzakt tudomány! Nem egzakt, de nem is azt várjuk tőle. Azt várjuk tőle, hogy általában jól szolgáljon ki minket. Ha nem azt teszi, akkor még mindig jöhetünk a nagy eszünkkel, és ráerőltethetjük az általunk kieszelt végrehajtási tervet a szerverre – optimizer hintek használatával. Erről azonban nem írok bővebben, mert sok ember elbízza magát, és okosabb akar lenne az SQL Szervernél. Sok siralmasan teljesítő alkalmazás van emiatt a világban. Majd mi megmutatjuk a szervernek! Láttam már pár ilyet. Ilyenkor elhívnak szakérteni, ledobálom a hinteket, és csodák csodája, az alkalmazás gyorsabb, mint valaha.

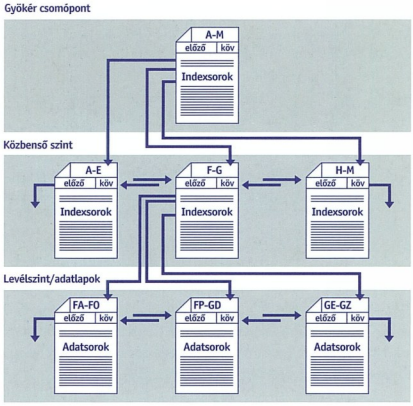
Az Indexek lelkivilága

Nagy vonalakban nézzük meg hogyan tárolja az SQL Server az indexeket. Ígérem nem lesz sokkal nehezebb a főzőcskénél. A szerver mindkét típusú indexet *(clustered és nonclustered)* B-fában tárolja. A B betű nem a binárisra, hanem a Balanced-ra *(kiegyenlített)* utal. A fa csomópontjaiból kiindulva több csomópontot is elérhetünk. Ha bináris lenne, akkor mindig csak kettéágazna. A fát úgy építi a szerver, hogy nagyjából mindig szimmetrikus legyen, azaz ne legyen felkopaszodás az egyik helyen, és lelógó ágak a másikon. Másképpen fogalmazva úgy, hogy az indexlapok nagyjából ugyanannyi sort tartalmazzanak. Erre utal a kiegyenlített jelző.

Minden egyes index egy-egy ilyen fát jelent. Az indexek ugyanolyan 8k-s lapokon élnek, mint az adatok, és minden indexnek van saját IAM-ja, ami az indexet alkotó extenteket fogja össze. Emellett az index lényegét az adja, hogy a fastruktúrában tárolt indexbejegyzések az indexelendő oszlop szerint vannak berendezve, így a fát természetes sorrendben bejárva az indexet alkotó kulcsok szerint rendezve kapjuk vissza a kulcsokat.

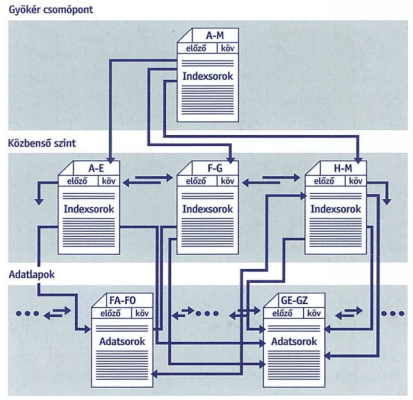
Legyen például egy emberek adatait tároló táblánk, amiben a családnévre hozunk létre egy indexet. Az SQL Server egy olyan fát épít, amelyet a gyökérszintről kiindulva a családnevek alapján rendezve járhatjuk be. Például keressük Fekete Lajost. A gyökérből elindulva a szerver látja, hogy az ábrán másodikként ábrázolt közbenső lapon találja meg az F és G betűvel kezdődő családnevű sorokat. Az abban található bejegyzésekből kiderül, hogy levélszinten a bal szélső lap tárolja az FA és a FO közötti kezdőbetűjű embereket, tehát ha van közöttük Fekete, akkor az csak azon a lapon lehet. Ezután a lapon található néhány sorból már könnyedén *(bináris kereséssel)* megtalálja a keresett sorokat, ha létezne. Látható, hogy egy sor megtalálásához nem kellett feltúrni az összes adatlapot, hanem néhány indexbejegyzés kiolvasása után elegánsan megvolt a keresett lap.

Ezt az indexstruktúrát, amikor a levélszinten egyből az adatokat találjuk meg clustered index-nek nevezzük. Miután tudjuk, hogyan működik, teljesen természetes, hogy egy táblán csak egy clustered index hozható létre, mert az index létrehozása egyben be is rendezi az adatlapokat az index által megkívánt sorrendbe. Márpedig fizikailag csak egyféleképpen lehet sorbarendezni az adatsorokat.



Clustered index

Mivel levélszinten rögtön ott vannak a keresett adatok, a clustered index akkor is nagyon hatékonyan tud működni, ha a lekérdezés a tábla jelentős részét érinti. Mint a fenti ábrán látható az adatlapok is és a közbenső szint*(ek)* lapjai is össze vannak fűzve egy kétirányú láncolt listában. Emiatt tartományok keresése anélkül is tud működni, hogy állandóan végig kellene menni a fa közbenső levelein minden egyes sor eléréséhez. Például Furfangos és Haragos közötti neveket lekérdezve az SQL Server meghatározza a legelső lapot, ahol a legkisebb kezdőbetűjű értékes információ nyugszik *(levélszint 2. lap)*, és a lista mentén addig halad előre az adatlapok olvasásában, amíg el nem éri a lekérdezésben definiált utolsó elemet.



Nonclustered index

A nonclustered index fája gyökér és közbenső szinten azonos a clustered indexével. A különbség levélszinten szembeötlő: míg clustered indexnél ott rögtön az adatokat találjuk meg, nonclustered index esetén ott csak egy mutató

van a tényleges adatsorra. Ez alapban egy adatbázisfájl azonosítóból, egy lapazonosítóból és egy sorazonosítóból áll. 100 sornyi információ felolvasásához 100-szor be kell járni a fát, és 100-szor kell hozzáférni az adatlapokhoz. Lehet, hogy fizikailag az összes sor egy adatlapon van, de akkor ahhoz az egy laphoz kell az SQL Server-nek 100-szor hozzáférni. Így érthető, hogy nagymennyiségű adatot *(tipikusan a tábla több mint 10%-át)* érintő lekérdezéseknél az SQL Server nem fogja használni a nonclustered indexet, inkább végigmegy közvetlenül az adatlapokon. A láncolt lista itt is fel van építve, így az elejét megfogva most is közvetlenül végig tud gyalogolni az adatokon.

Heap esetén, azaz, ha nincs semmilyen index egy táblán, a szerver az IAM lapokat elemezve tudja csak összevadászni az adatlapokat. Ebben az esetben egyedül az IAM teremt kapcsolatot a lapok között. Ez általában a legkevésbé hatékony módszer, ezért a táblákat mindig célszerű minimum egy clustered indexszel berendezni.

A clustered index nagy kincs, nem érdemes az elsődleges kulcsra elpazarolni. Elsődleges kulccsal úgyis pontszerű lekérdezéseket szoktunk végrehajtani, arra tökéletes egy nonclustered index is. Az viszont kell! Nehogy úgy tegye le valaki a cikket, hogy azt olvasta, hogy nem kell index az elsődleges kulcsra.!

Hol érdemes bevetni a clustered index-et? Olyan oszlopra, ami szerint gyakran végzünk szűrést, és a lekérdezés eredményes sok sort ad vissza. Olyan oszlopokra, amelyeknél előnyös lehet az adatok eleve rendezett tárolása. Például a GROUP BY és az ORDER BY szereti a clustered index-et. Emellett a gyakran keresett oszlopok, például az idegen kulcsok oszlopai is hálásak bármilyen indexért.

Indexelhető nézetek

Miután mindent tudunk az indexekről, itt az ideje, hogy rátérjünk az előző részben beharangozott csodára, az indexelhető nézetre. Ha visszaemlékszünk, ott egy egyszerű összegzést gyorsítottunk fel egy olyan segédtábla alkalmazásával, amiben mi tartottuk karban az előre összegzett eredményeket triggerek segítségével. Láttunk, hogy működött, de nem volt egyszerű. Van azonban egy nagy segítségünk, az indexelhető nézet.

A hagyományos nézetek *(View)* nem mások, mint megnevezett SQL lekérdezések. Azaz becsomagolunk egy SQL lekérdezést egy nézetbe, és a többi lekérdezésben úgy hivatkozunk a nézetre, mintha ő egy fizikai tábla lenne. Pedig a valóságban mindig lefut a törzsében kijelölt lekérdezés.

Az SQL 2000-ben nemcsak táblára, de nézetre is hozhatunk létre indexet. Ez azt eredményezi, hogy a nézetben kijelölt adatokat tényleg előre kiszámítja az SQL Server, és fizikailag egy táblában eltárolja. Ez számunkra láthatatlan, mi csak annyit észlelünk, hogy a felindexelt nézeten keresztül a lekérdezések nagyon gyorsak, és a szerver nem nyúl hozzá az alaptáblához. Az alaptábla *(táblák)* változásait át kell vezetni a nézet mögötti táblába és indexstruktúrába is. Ezt csináltuk triggerrel az előző részben. Örömteli, hogy az SQL Server ezt megoldja helyettünk teljesen automatikusan. Csak létrehozzuk az indexet a nézeten, és a szerver átvezet minden változást.

A felgyorsítandó lekérdezés így nézett ki:

```
SELECT
  OrderID,
  SUM(Quantity * UnitPrice * (1-Discount))
FROM [Order Details]
GROUP BY OrderID
```

A célunk a részösszegek előre kiszámítása. Hozzunk létre egy nézetet, ami a fenti lekérdezés kimenetét generálja, csak vegyük bele a sorok számát csoportonként. Ez szükséges feltétel az index létrehozásához a nézeten:

```
CREATE VIEW FastOrdersView
WITH SCHEMABINDING AS
SELECT
  OrderID,
  SUM(Quantity * UnitPrice * (1-Discount)) as
szumma,
  COUNT_BIG(*) AS sorokszama
FROM dbo.[Order Details]
GROUP BY OrderID
```

A nézeten csak akkor tudunk indexet létrehozni, ha több feltétel is teljesül rá. Ezeket az **[1]** címen található példakódban foglaltam össze, terjedelmi okokból nem részletezem őket. A fenti nézetben a WITH SCHEMABINDING is a feltételek része, ez megakadályozza, hogy a nézet mögötti táblák szerkezetét módosítsuk, így kirántsuk a talajt alóla. A nézet kész, már csak létre kell hozni rajta az indexet. Nézeten az első index kötelezően unique *(csak egyedi értékeket tartalmazó)* clustered index kell legyen. Ráadásul GROUP BY esetén a GROUP BY utáni oszlopra kell ráadni, úgyhogy sok választásunk nem maradt: OrderID oszlop.

```
CREATE UNIQUE CLUSTERED INDEX IDXCLU_OrderID
ON FastOrdersView (OrderID)
```

Az index létrehozásakor elkészül a nézet kimenetét tároló láthatatlan háttértábla és maga az indexfa, azaz a nézet materializálódik. A meglepetés akkor ér bennünket, amikor ismét végrehajtunk az eredeti, felgyorsítandó lekérdezést az alaptáblán, azaz az Order Details-en: az SQL Server nem fogja kiszámolni a szummát, hanem egyszerűen előveszi a már előre kiszámolt eredményeket a nézetből! Azaz létrehoztunk egy indexelt nézetet, és ettől felgyorsult egy olyan lekérdezés, amiben nem is hivatkoztunk a nézetre. Ez már tudomány!

Soczó Zsolt
Zsolt.Soczo@netacademia.net

A cikkben szereplő URL-ek:
[1]: A cikkben szereplő script-ek
http://technet.netacademia.net/download/sql

XMLgessünk A szappanopera folytatódik *(V. rész)*

Szappanoperánk előző részében koncepcionálisan áttekintettük a SOAP *(Simple Object Access Protocol)* alapjait. Akinek elég egy átfogó kép, az olvassa el még egyszer azt a részt, és ezt most hagyja ki. Akit érdekel az is, hogy a SOAP elmélet hogyan néz ki a gyakorlatban is, annak ajánlom ezt a fejezetet is.

Áttekintjük, hogy mely konkrét technológiák képesek testközelbe hozni az absztrakt SOAP szabványt, és hogyan lehet ténylegesen megírni egy SOAP alapú alkalmazás ügyfél- és kiszolgálóoldali komponenseit is.

Már megint a metaadatok!

Van egy felturbósított HTTP kiszolgálónk, amely boldogan várja, hogy SOAP kéréseket szolgáljon ki. A felturbósítás részleteivel, azaz, hogy egy szimpla webszerverből hogyan lesz SOAP szerver a cikk második felében foglalkozunk.

A SOAP segítségével nagyszerű üzeneteket csereberélő alkalmazásokat lehet írni, azonban a mai programozók objektumokban gondolkodnak, és nem akarnak egyedi SOAP borítékokat cipelni - arra való a postás. Nem akarnak foglalkozni a paraméterek és a visszatérési értékek bájtfolyammá kódolásával és dekódolásával. *(Nehéz a Serialization/Deserialization fogalmak magyarítása, de értelmében a bájtfolyammá kódolás írja körül leginkább a serialization fogalmát.)* Ki lesz az a postás?

Amikor egy COM komponens valamely metódusát meg akartuk hívni, akkor honnan tudtuk, hogy annak mely interfészei mely metódusokat támogattak? A Type Library volt az, ami specifikálta a COM komponensek interfészeit és az azokban megvalósított metódusok típusát, szignatúráját. A szignatúra valami olyasmi, mint amikor egy C nyelvű program header *(.h)* állományában leírjuk, hogy egy függvény milyen típusú paramétereket vár, azok közül melyik milyen irányú információt hordoz, mi a hívási konvenció és milyen a visszatérési érték típusa. Azaz egy metódus szignatúrája a metódus neve, a paraméterek száma és típusa, valamint a visszatérési érték típusa. Ahhoz, hogy egy műveletet vagy metódust használni tudjunk, szükségünk van annak szignatúrájára. Hasonlóan néz ez ki a SOAP világban is.

Egy SOAP kiszolgáló hogyan tudja publikálni a rajta meghívható metódusok szignatúráját? Erre találták ki a WSDL-t *(Web Services Description Language)*, magyarul Webszolgáltatások Leírónyelve. A WSDL egy XML alapú leírónyelv, amely - a COM Type Library-hez hasonlóan – a SOAP kiszolgáló által publikált metódusok szignatúráját dokumentálja.

Egy picit itt álljunk meg. Eddig SOAP szerverről beszéltem általában, most pedig előjöttem a WSDL fogalmával, aminek már a nevében sem a SOAP, hanem Web Service szó áll. Miért? A SOAP egy közös fejlesztés eredménye a Microsoft, IBM, Ariba, Developmentor *(Don Box cége, ismerős a neve?)* és még jópár cég részvételével. Az előző részben láthattuk, hogy a SOAP specifikáció nyitva hagyja az implementációs részleteket. Mivel mi most egy konkrét ügyfél-kiszolgáló alkalmazást írunk, mindenképpen ki kell kötnünk egy konkrét implementációnál. A világban sok SOAP megvalósítás létezik, ezek közül mi most a Microsoft Soap Toolkit lehetőségeivel fogunk foglalkozni. Habár a webszolgáltatások hátterében tetten érhetjük a Microsoft munkáját is, a technológia nem a Microsoft-é! A WSDL nyelv már a World Wide Web konzorciumnál jár szabványosításra, csak nyilván fiatal kora miatt még nem ajánlás státuszú. Egyébként is hamarosan világos lesz, hogy a hangzatos webszolgáltatás név mögött egy SOAP szerver lakik. Vissza a WSDL-re! Hogyan állítjuk elő egy konkrét SOAP szerver által publikált metódusok WSDL leírását? Például kézzel, notepad-ban. A WSDL az XML Schema szabványra épít, így aki tud sémát írni, az már majdnem tud WSDL-t is írni. Álljon itt egy WSDL részlet:

```
<definitions
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" >
...
<s:element name="Add">
 <s:complexType>
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1"
     name="A" type="s:float" />
    <s:element minOccurs="1" maxOccurs="1"
     name="B" type="s:float" />
  </s:sequence>
 </s:complexType>
</s:element>

<s:element name="AddResponse">
 <s:complexType>
  <s:sequence>
   <s:element minOccurs="1" maxOccurs="1"
    name="AddResult" type="s:float" />
  </s:sequence>
 </s:complexType>
</s:element>
...
</definitions>
```

Ez bizony egy XSD *(XML Schema)* darabka. Ez a részlet egy olyan metódust ír le, aminek C++ *(C#)* nyelvű szignatúrája így néz ki:

```
float Add(float A, float B)
```

Mivel a metódushívás SOAP-ra leképezve nem más, mint egy kérés és arra adott válasz, ezért az Add elem a kérést írja le, amiben a metódus paramétereit deklaráljuk *(float A, float B)*. A kérésben szereplő metódus nevét Result-tal kiegészítve kapjuk vissza válaszként, így a második elemdeklaráció a visszatérési értéket írja le.

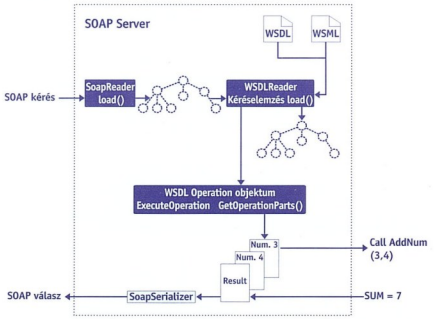
Egy igen fontos tényt kell megfigyelnünk a fenti leírásban: a paraméterek típusosak, például az A paraméter típusa float. Mindezt az XSD-nek köszönhetjük, amiben a legtöbb ismert és a szoftveriparban széleskörben használt adattípust definiáltak.

Egyszerű SOAP kiszolgáló

Első SOAP kiszolgálónkat a Microsoft SOAP Toolkit 2.0 SP2 segítségével írjuk meg. A SOAP Toolkit szabadon letölthető az **[1]** címről. A példákat külön kell letölteni, azok a **[2]** címen találhatók. Addig, amíg nincs végleges .NET-ünk, és élesben kell fejleszteni, addig a Soap Toolkit segítségével is teljes értékű SOAP szervert írhatunk.

A Toolkit segítségével SOAP üzeneteket egyszerű módon leképezhetünk COM objektumok hívására. Mivel COM objektumot VB6-ban könnyű írni, már csak kellene egy alkalmazás, ami elkapja a SOAP kéréseket, és azok hatására meghívja a COM komponensünk megfelelő metódusát. Honnan tudja egy közvetítő alkalmazás, hogy mely SOAP hívást melyik komponens melyik metódushívására fordítsa át? Erre szolgál a Web Services Meta Language *(WSML)* leíróállomány. Ez nem szabványos leírófájl, ne is keressék a World Wide Web konzorcium honlapján! Ez kifejezetten a SOAP Toolkit által nyújtott implementációhoz kell. Még visszatérünk rá.

A SOAP Toolkit kétféle kiszolgálóoldali ún. SOAP Listenert tartalmaz: Internet Server API *(ISAPI)* listener és Active Server Pages *(ASP)* listener. Mindkét típus feladata a webszerverhez érkező SOAP kérések fogadása, feldolgozása, a kért komponens metódusának meghívása és a visszatérési értékek SOAP csomagban történő visszaküldése. Mi magunk is írhatnánk ilyen programot, de minek bajlódjunk az összes SOAP fejléc és tartalom elemezgetésével, amikor ezt megteszik helyettünk? Lássuk, mi van ebben a közvetítő alkalmazásban!



A beérkező SOAP kérést a SoapReader komponens felolvassa és mint közönséges XML dokumentumot értelmezi. A végeredmény egy DomDocument objektum, ami a kérést reprezentáló bináris XML fát tartalmazza. Közben a WSDLReader sem tétlenkedik, az felolvassa és értelmezi a SOAP szerverünk szolgáltatásait leíró WSDL, és a meghívandó COM objektum felé a kapcsolatot definiáló WSML fájlokat. Mivel azok is XML formátumúak, szintén egy-egy XML DOM-ba töltődnek be.

A WSDLReader értelmezi a SOAP kérést, és készít egy WSDLOperation objektumot. Az meghívja a GetOperationParts() metódusát, ami a teljes WSDL/WSML fából kiszedi azokat a részeket, amelyek a konkrét kéréshez és válaszhoz tartoznak. Emlékeztetőül a WSDL állomány az adott SOAP szerver összes metódusát írja le, de nyilván a paraméterek ellenőrzéséhez és a végrehajtáshoz csak a meghívandó metódushoz tartozó részt kell figyelembevennie.

Ezek után a SoapServer minden egyes paraméterhez és a visszatérési értékhez készít egy SoapMapper objektumot, és feltölti őket a kérésben kapott paraméterekkel. A SoapServer meghívja a kért feladatot ellátó metódust, amit a WSML fájlban kijelölt COM objektumban keres. A COM metódus visszatérési értéke ismét egy SoapMapper objektumon keresztül jön vissza. *(Tudjuk, hogy egy metódushívás visszatérési értéke COM-ban mindig egy szám, HResult, de vannak kimeneti paraméterek, és például a VB is azzal szimulálja a visszatérési értéket).*

A megfelelő SoapMapper-ben rendelkezésre álló visszatérési értéket a SoapSerializer alakítja át a SOAP szabványnak megfelelő formátummá. A SoapServer-nek már csak fel kell kapni ezt, és visszaküldeni az ügyfélprogramnak egy HTTP Response-ban.

Amikor a SoapServer objektum vezényli le a teljes koncertet, akkor az ún. magasszintű API-t használjuk. Az esetek többségében ez megfelel nekünk, de akár alá is nyúlhatunk az imént vázolt folyamatnak, és a keretben látható objektumokkal mi is koordinálhatjuk a teljes folyamatot. Így például megoldhatjuk, hogy a háttérben nem COM objektumok ülnek, hanem Win32-es DLL-ek, vagy CORBA objektumok, vagy bármi, aminek a szolgáltatásait meg szeretnénk hívni. Térjünk vissza a listenerekre. Az ISAPI listenert akkor használjuk, ha ki akarjuk használni az ISAPI technológiában rejlő teljesítményt *(ISAPI alkalmazásként lehet a leggyorsabb webalkalmazást írni IIS – Internet Information Server alá)*, és ha közvetlenül nem akarjuk babrálni a SOAP üzeneteket. Ennek megfelelően az ASP megoldás lassabb lesz, de végrehajtás előtt kielemezhetjük a bejövő SOAP üzenetet, így például saját biztonsági réteget vonhatunk a hívó és a SOAP kiszolgáló közé. Erre szükségünk lesz, ha nem tudjuk kihasználni az IIS integrált autentikációját, azaz általában internetes környezetben.

A SOAP kéréseket fogadó ISAPI alkalmazás neve soapisap.dll, és alapértelmezett módon telepítve a

```
C:\Program Files\Common Files\MSSoap\
↳ Binaries\
```

mappában él. Alkalmazásához a wsdl állományokat tartalmazó webszerver könyvtárakban a .wsdl kiterjesztéshez hozzá kell rendelni ezt az ISAPI dll-t. A Toolkit telepítéskor ezt megteszi a gyökérwebre, de utólag a .NET Framework esetleg elhappolhatja a hozzárendelést. Ilyenkor jó tudni hol található a soapisap.dll.

Ha ezen a kész ISAPI alkalmazáson keresztül akarunk egy SOAP hívást végrehajtani, akkor a SOAP ügyfélprogramnak a megfelelő .wsdl állományra kell hivatkozni. A kérést elkapja az ISAPI DLL, és a wsdl/wsml leírások alapján végrehajtja a megfelelő COM objektum kért metódusának hívását.

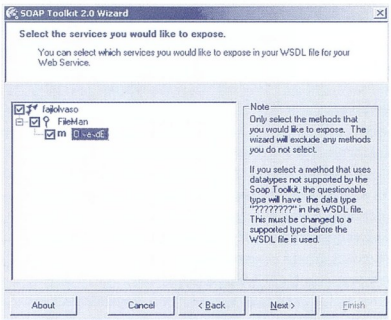
Írjunk metaadatot!

Létrehoztunk a IIS-en egy virtuális könyvtárat, ami mögé be van élesítve a soapisap.dll a .wsdl fájlokra irányuló kérések kezelésére. Már csak egy lépés van hátra a SOAP szerverünk működéséhez: meg kell írnunk a wsdl és a wsml állományokat.

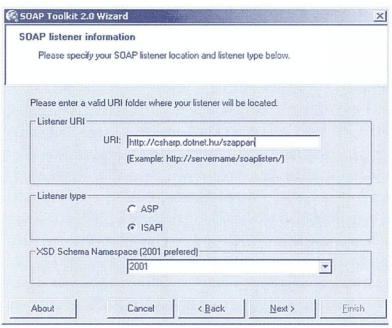
Ha itt tartana a SOAP Toolkit, akkor most nem írtam volna meg ezt a cikket, mert ki az, aki metaadatokat akar kézzel írni? Ha már ott van az összes szükséges metaadat a SOAP-on keresztül publikálandó COM objektumunk Type Library-jében, akkor miért ne lehetne automatikusan generálni a wsdl/wsml fájlokat?
Szerencsére kétféle wsdl/wsml generátort is kapunk, az egyik egy grafikus felületű manó *(varázsló)*, a másik pedig egy parancssorból hívható .exe. Mindkettő egy dologra való: COM Type Library alapján wsdl/wsml párost generál. A grafikus változat a Start Menü, Programs, Microsoft SOAP Toolkit, WSDL Generator menüpontból indítható, a parancssori változat, a wsdlstb.exe pedig a
C:\Program Files\MSSOAP\Binaries könyvtárban lakik.
A publikálandó COM komponensünk egy egyszerű VB komponens, amelynek PROGID-ja: SoapTestComp.FileMan, és az egyetlen metódusa:

```
Public Function OlvasdEl(ByVal strFilePath As
String) As String
```

Ez felolvassa a paraméterként megadott szövegfájlt, és azt visszaadja a hívónak. Egy sima webszerver is ezt teszi, de példának jó lesz. A komponens forráskódja letölthető a **[3]** címről, számunkra bőven elég a meghívandó metódus szignatúrája és a tőle elvárt működés.
Jöjjön a varázsló! Elindítjuk a Soap Toolkit 2.0 Wizard-ot, ami némi bemutatkozás után megkérdezi, hogy milyen néven generálja le a wsdl/wsml fájlokat, valamint, hogy melyik .dll-ben *(vagy másban)* van a publikálandó COM komponens Type Library-je. Példánkban a fájlolvasó wsdl/wsml nevet adtam, és a Type Library a VB által fordított DLL-ben van, úgyhogy azt kell kiválasztani a listából.
A következő lépésben a varázsló megmutatja a komponensünk által publikált metódusokat. Ha lenne több interfészünk, akkor mindegyiket kilistázná, és bármelyikből kiválogathatjuk a SOAP-ra ültetendő metódusokat *(én is megtettem, mind az egyet)*.

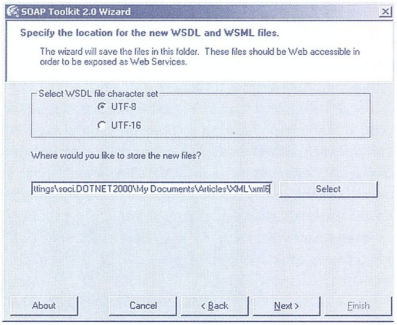


A varázsló figyelmeztet egy igen fontos tényre. Ha olyan adattípust használunk akár paraméterként, akár visszatérési értékként, amit az XML séma alapban nem ismer, akkor a generált wsdl/wsml dokumentumokban a kérdéses helyeken ????-ket fogunk látni, és nekünk kell kézzel megírni a típusdeklarációkat. Gondoljunk csak bele, milyen típusra képezné le a varázsló például valamely objektumunkat? Hisz semmi akadálya, hogy egy metódus egy objektumot adjon vissza, ugye?
Mit lehet ilyen helyzetekben tenni? Ilyenkor bizony át kell írni a komponenst, és nekünk kell olyan típusokat használni, amelyek kompatibilisek a XSD-ben lefektetettekkel. Ha nincs meg az eredeti forráskód, akkor ez akár azt is jelentheti, hogy írni egy csomagoló komponenst, ami aggregálja a kiinduló komponenst, és nekünk kell megírni az elvadult adattípusok *(például saját objektumok)* átalakítását XSD által ismert típusokká, például string-gé. Objektumok esetén az objektum teljes állapotát akarjuk serializálni string-gé, ezt átküldjük egy SOAP üzenetben a hívónak, majd az visszaállítja belőle az eredeti objektumot. Ez persze azt jelenti, hogy ügyféloldalra is kell írni egy olyan eljárást, ami képes a serializált objektumból újra valódi objektumot gyártani *(deserialize)*. Ember legyen a talpán, aki ezt mind végigcsinálja!
Én ismerek valakit/valamit, aki/ami ezt mind tudja, sőt képes arra, hogy transzparens módon, tetszőleges, általunk fabrikált objektumot serializál, átküld a dróton, majd visszaalakít, úgy, hogy ebből gyakorlatilag semmit sem lát a programozó. És nem kell hozzá alapjaiban megváltoztatni a publikálandó komponenst. Meglepő, hogy már megint a .NET-ről beszélek ?!
Akinek szüksége van saját típusok használatára még a .NET végleges megjelenése előtt, annak javaslom, hogy nézze meg a SOAP Toolkit-ben a Custom Type Mapper-ek kialakításának részleteit. Ez egy kicsit más megközelítésben *(XML DOM faragás)*, de ugyanerre a problémára ad megoldást.
De maradjunk a földön, a .NET-es megoldásról pedig majd a következő részben beszélünk. Tegyük fel, hogy jólfésült komponensünk nem használ semmilyen trükkös típust, amit nem ismer a wsdl generátor. A tömb szerencsére megy neki, mert bár az nincs benne az XSD-ben, a SOAP-ba belerakták. Igazából a SOAP-ban csak két új adattípust kellett felvenni a szerzőknek a XSD-ben találhatókon felül, a tömböt és a referenciát. Ezekkel majd későbbi számokban fogok foglalkozni, mert most nem maghasítót, csak egy egyszerű fájlolvasó szolgáltatást írunk.
Komoly szellemi munkával kiválasztottuk a publikálandó metódust, Click Next. Meg kell adnunk, hogy hol lehet majd elérni a webszerveren keresztül a SOAP végpontunkat. Itt olyan URL-t kell megadunk, ahol a generált a wsdl/wsml fájlokat lehet majd elérni HTTP-n keresztül. A varázsló nem hoz létre nekünk virtuális könyvtárat a webszerveren, a mi dolgunk, hogy az URL tényleg leíró fájlokra mutasson.



Középen megadjuk a listener típusát, mi az egyszerűség kedvéért most ISAPI listener-t választunk. A legalsó listában ki-

választhatjuk, hogy melyik séma dialektusban generálódjon a wsdl típusleíró szekciója. Persze, hogy a legfrissebben.
Az utolsó lépésben adjuk meg azt, hogy fizikailag hová kerüljenek a leíró fájlok. Az itt megadott mappát kell kipublikálni az előző lépésben megadott URL mögé. Esetemben ez azt jelentette, hogy létrehoztam egy szappan nevű virtuális könyvtárat erre az elérési útra.
Vigyázzunk, mert Windows 2000 alatt már legtöbbször nincs joga az Everyone-nak minden állományt olvasni, sőt, például a felhasználók „My Documents"-ében semmilyen joga sincs! Mint az alábbi képen is látható én a leírófájlokat a saját „My Documents"-emen belül hoztam létre, és innen a webszerver nem tud Anonymous kéréseket kiszolgálni, mert nincs joga a könyvárhoz. Megoldás: a .wsdl állományokra adni kell olvasási jogot az IUSR_Gépnév felhasználónak, mert ennek a nevében éri el az IIS az állományokat. Emellett nyilván a komponenst is el kell érnie, így arra is kell olvasási jog.



A kaméleon SoapClient, azaz ahol a való világ véget ér, ott kezdődik a varázslat

Láttuk, hogy kiszolgálóoldalon elvégzik helyettünk a piszkos munkát, és gyakorlatilag egy ötlépéses varázslóval elérhetővé tettük a COM objektumunkat a SOAP-pal támadó ügyfélprogramok felé. A másik oldal még egyszerűbb. Nézzük csak!

```
Dim sc As New MSSOAPLib.SoapClient
sc.mssoapinit
"http://localhost/szappan/fajlolvaso.wsdl"
```

Ahelyett, hogy mi magunk állítanánk össze a SOAP kérést, létrehozunk egy példányt a SoapClient objektumból, és megadjuk neki a SOAP szervert leíró wsdl fájl elérését. Ő letölti ezt, értelmezi, és felkészül a SOAP hívás indítására. És itt jön a füst és tűz. A sc objektumunk átalakult! Ő már többé nem *(csak)* SoapClient objektum, hanem SoapTestComp.FileMan objektum is egyben! Ez azt jelenti, hogy meghívhatjuk az OlvasdEl metódusát, mintha az mindig is az ő része lett volna:

```
Dim s As String
s = sc.OlvasdEl("c:\winnt\system32\eula.txt")
```

És a hívás hipp-hopp kiköt az előbb gondos munkával elkészített SOAP szerverünkön, ott meghívódik a VB komponensünk azonos nevű metódusa, az eredmények visszatérnek, és mi pontosan ugyanazt a viselkedést kapjuk vissza, mintha a komponens a mi *(ügyfél!)* gépen futna. Döbbenet.
A SoapClient ebben az esetben un. Stub-ként *(tuskó?)* viselkedik, azaz úgy csinál, mintha ő lenne a távoli komponens, de valójában csak úgy csinál, és a valódi munkához a távoli komponenst szólítja meg SOAP-on keresztül. Milyen jól elprogramozgatjuk a SOAP-ot, és még nem is láttunk egy darab SOAP borítékot sem. De ami késik, nem múlik!

SOAP Trace utility

Ő az utolsó lakója a SOAP Toolkit-nek. Segítségével beléphetünk a SOAP kommunikáció közepébe, mint egy átjátszóadó, és így ki tudjuk elemezni a SOAP forgalmat. A Network monitorhoz hasonlítható, csak ez nem passzívan hallgatózik, hanem aktívan figyel egy porton, elkapja a kapcsolatfelvételi kérelmeket, és átjátssza őket a megadott címre, majd onnan vissza a választ. Tulajdonképpen egy olyan SOAP proxy, amellyel még ki is tudjuk elemezni a forgalmat. A program használatához tudnunk kell, hogy a SoapClient miután letöltötte a wsdl állományt, akkor nem ugyanazon a címen kezdi el keresni a SOAP szolgáltatást, hanem a wsdl-ben kikeresi a service elemet, és azon belül a soap:address elem location attribútumában leírt címre küldi el a SOAP csomagot:

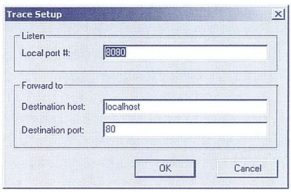
```
.Wsdl részlet

<service name="fajlolvaso>
 <port name="FileManSoapPort"
       binding="wsdlns:FileManSoapBinding">
   <soap:address  location="http://csharp.dotnet.-
hu/szappan/
↳ fajlolvaso.wsdl" />
</port>
</service>
```

Alapértelmezésben ez ugyanoda mutat, mint ahol a wsdl fájl lakik, de ettől nyugodtan eltérhetünk. A tracer használatához ezt át kell írnunk arra a címre, ahol a tracer várni fogja a SOAP csomagokat. Például ugyanazon a gépen a 8080-as portra:

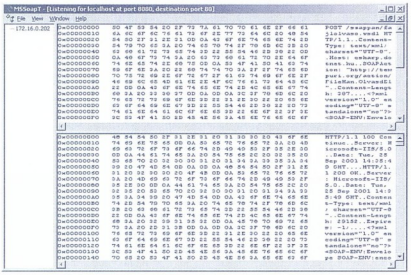
```
http://csharp.dotnet.hu:8080/szappan/
↳ fajlolvaso.wsdl
```

Ha így átírtuk a wsdl-t, akkor az eddigi ügyfélprogramok a fenti címre fognak postázni, azaz a nihilbe. Ez nem jó senkinek, jöjjön a tracer!
Amikor elindítunk egy trészelést, a következő dialógussal találjuk szembe magunkat:

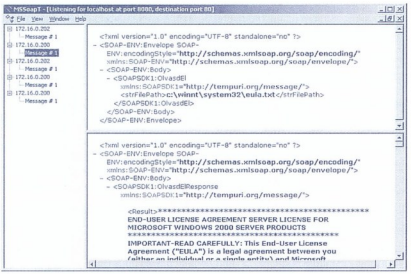


Megkérdezi, hogy melyik helyi TCP port-on várja a tracer a kéréseket. Ide nyilván azt írjuk be, amit a wsdl fájlba írtunk,

hisz itt fogják keresni az ügyfélprogramok a SOAP szervert. Az alsó részen állítjuk be a valódi SOAP szerver nevét és portját. Ide fogja átjátszani a fenti port-ra érkező kéréseket. A tracer-ben választhatunk a formázott és a nemformázott kimenetetek közül. A nemformázott olyan, mint amit a hexa editorokban láthatunk: a baloldalon hexában a bájtok, a jobboldalon a szöveges értelmezésük.



A baloldali részen láthatóak azok az ügyfélgépek, amelyek forgalmaztak a tracer-en keresztül. A jobb felső ablakban a SOAP kérés látható, az alsóban pedig a SOAP válasz.
Aki már elszokott a hexa editoroktól *(például én)*, annak javaslom a formázott trace-t. File, New, Formatted Trace.



Az elrendezés ugyanaz, csak ebben az esetben az Internet Explorer-től megszokott formázásban láthatjuk a SOAP borítékokat. Érdemes a képet kicsit tanulmányozni, akár úgy, hogy előveve a cikk előző részét azonosítani benne a különböző SOAP huncutságokat: a borítékot, ki milyen névtérben van, a válasz nevét, satöbbi.

Amiről nem beszéltünk

Mint említettem, az előző oldalakon felvázolt magasszintű SOAP API nem képes olyan típusok közvetítésére, amelyek nincsenek benne az XSD-ben. Azaz általunk írt osztályok, struktúrák esetén segítenünk kell a SOAP komponenseknek. Ehhez egyfelől át kell írnunk a wsdl típusleíró szekcióját, és az XSD szabványnak megfelelő formátumban leírni a komplex típusainkat. Másfelől írnunk kell egy Custom Type Mappert, ami a legelső ábrán látható architektúrában le tudja kezelni az általunk használt összetett típusokat. Valójában ez nem bonyolult, egyszerűen implementálni kell a ISoapTypeMapper interfészt *(VB Implements kulcsszó)*, azaz megírni néhány szükséges metódust. Emellett a wsml fájlban is hivatkozni kell a Mapper objektumra. Részletek a Toolkit helpjében.
Emellett levezényelhetjük a teljes SOAP kommunikációt mi magunk is, de erről majd a jövőben...

Zárszó

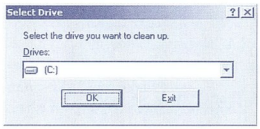
Ahogy a cikkben már többször beharangoztam, a következő részben a .NET Framework által szolgáltatott WebSzervizzel fogok foglalkozni, amely egy másféle SOAP implementáció. Addig is javaslom a példakódokat és a SOAP SDK-t letölteni, és kísérletezni vele. Nagyon szellemes példákat is adnak hozzá, érdemes azokat is kipróbálni.

Írta és rendezte: Soczó Zsolt
Zsolt.Soczo@netacademia.net

A cikkben szereplő URL-ek:
[1]: SOAP Toolkit 2.0 SP2
http://download.microsoft.com/download/xml/soap/2.0/W98NT42KMe/EN-US/SoapToolkit20.exe
[2]: SOAP Toolkit 2.0 SP2 Példák
http://download.microsoft.com/download/xml/soap/2.0/W98NT42KMe/EN-US/SoapToolkit20Samples.exe
[3]: Tesztalkalmazások
http://technet.netacademia.net/download/xml

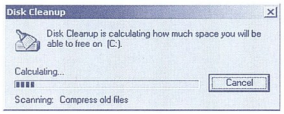
Hogyan tüntessük el nem használt fájljainkat?

Hogyan tüntessük el nem használt fájljainkat?
K: *Tud valaki egy módszert, amivel biztonságosan, de hatékonyan ki lehet pucolni a winnt-t, hogy a használaton kívüli fájlok eltűnjenek?*
V: A Windows 2000-ben a leghatékonyabb eszköz erre a feladatra a Windows-ba épített Lemez karbantartása *(Disk Cleanup)* eszköz.
Az eszközt a Programok *(Programs)*, Kellékek *(Accessories)*, Rendszereszközök *(System Tools)* menü alatt találjuk Lemez karbantartása vagy Disk Cleanup néven. A parancssorból cleanmgr néven indítható.
A program fő feladata a szükségtelen fájlok eltávolítása a merevlemezről, így a kisebb méretű lemezeken is értékes helyet szabadíthat fel a hasznos programok vagy adatok számára. Ehhez az egyébként – főleg a rendszerfájlok miatt – nem veszélytelen művelethez hatékony és biztonságos segítséget nyújt – a rendszerfájlok letörlésére lehetőséget sem ad. A program indítása után kiválaszthatjuk a takarításra kijelölt merevlemezt.



Lemezkiválasztás a Disk Cleanuppal

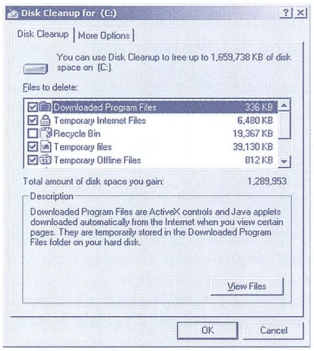
Majd őrült reszelés kezdődik...



A felirat tanúsága szerint nemcsak helykiszámítás, hanem némi fájltömörítés is zajlik a háttérben

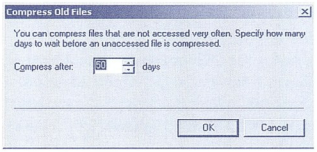
...aminek során az eszköz feltárja a felszabadítható területeket. Takarítás a következő fájlok között lehetséges:

- Temporary Setup fájlok (a telepítés után nincs szükség rájuk)
- Letöltött programfájlok
- Temporary Internet fájlok, az Internet Explorer gyorstárolója
- Régi chkdsk fájlok, a lemezellenőrzés során keletkezett fájltöredékek
- Lomtár *(Deleted Items)*
- Temporary *(ideiglenes)* fájlok, a Temp könyvtár tartalma
- Kapcsolat nélküli *(Offline)* fájlok, a hálózati kapcsolat nélkül is használható hálózati fájlok
- Tömörített régi fájlok
- A tartalomindexelő katalógusfájljai, amelyek a keresések felgyorsítására készülnek



Több mint egy giga felszabadítható ezen az alig használt gépen!

Ha nem a Windows 2000 telepített verzióját tartalmazó lemez kerül kiválasztásra, akkor csak a kuka és a tartalomindexelő katalógusfájljai kiválaszthatóak. A Compress Old Files igen figyelemreméltó tulajdonsága, hogy nem törléssel szabadít fel helyet, hanem a régi fájlok *(NTFS szintű)* tömörítésével. Ki tudta eddig, hogy ilyen okos a Windows? Itt például az ötven napnál régebbi fájlokra mondunk ki tömörítési „ítéletet".



Az ötven napnál régebbi fájlok tömörítése

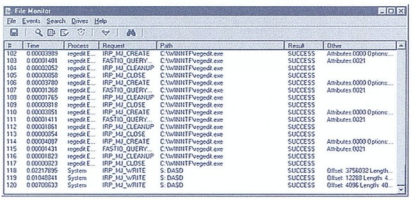
A More Options fülön választhatunk a Windows összetevők és a Telepített programok közötti takarításban is. Ha ezekre a pontokra kattintunk, a Windows elindítja a Windows összetevők varázslót, illetve a Programok hozzáadása és eltávolítása eszközöket.
Forrás : Netacademia Windows 2000 lista

Windows – Security Ki tekeri az üres floppymeghajtót?
K: *Mind a laptopomon, mind pedig a szerveren ugyanaz az elmebaj lett úrrá, tekerik az ÜRES flopimeghajtót. Ez mi? SP? Hotfix? Vírus? Tud erről valaki valamit? A dolog egyértelműen nemrég kezdődött, de nincs meghatározó kiindulási pont, hogy mitől.*

V: A tünetekből adódóan három alapvető okot lehet feltételezni a lemez keresésére. A lemezt
1. valamilyen program próbálja használni
2. a Windows próbálja használni
3. egy parancsikon *(shortcut)* miatt kell tekerni

Mivel egyszerű kereséssel az esetek egyike is csak a szerencsének köszönhetően oldható meg, célszerű valamilyen tudományos megoldás után nézni.

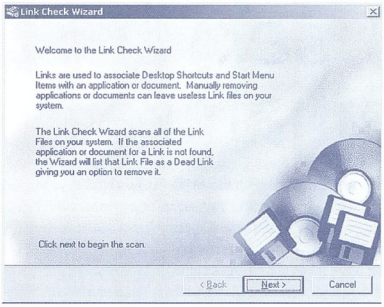
Az első két probléma megoldására jól használható a Sysinternals cég Filemon eszköze, mely valós időben vizsgálja a fájlrendszer aktivitását. Képes az aktivitás naplózására az olvasás, írás, törlés, terhelések időbeni rögzítésére.



A filemon működés közben

Jelen problémára is a Filemon segített megoldást találni. A fenti 2. kategóriába sorolható problémát az okozta, hogy a lemezre történő másolás után a Windowsból kiléptek anélkül, hogy a másoló ablakot bezárták volna, majd kivették a lemezt. A következő belépéskor a Windows azért kereste a lemezt, hogy megnyithassa az elmentett profilunk által „diktált" ablakot. Egy floppy behelyezése után felugrik az A: ablak, s annak bezárásával megoldható a „probléma".

A harmadik eset, mely nem használt – A: meghajtóra mutató – parancsikonokra vezethető vissza, a CHKLNKS.EXE eszközzel oldható meg. A programot a Windows Resource Kit tartalmazza.



A Link Check Wizard a levegőbe mutató parancsikonok gyilkosa

A chklnks megvizsgálja a rendszer használt és nem létező fájlokra mutató linkjeit. A programban a nem szükséges linkeket kijelölhetjük és eltávolíthatjuk. Az A: meghajtóra mutató linkeket is megtalálja és eltávolítja.

Forrás : Netacademia Security Lista

Windows 2000 - System 5 error occurred

K: *„System 5 error occurred" Honnan tudom kideríteni, hogy mit takar ez a hibaüzenet az Event View-erban?*

V: A Windows 2000 számmal jelzett hibaüzeneteinek jelentését a

```
NET HELPMSG #hibaszám
```

parancs segítségével kérdezhetjük le. A kérdésre a megoldás tehát:

```
NET HELPMSG 5
```

A hiba jelentése: Access Denied, azaz hozzáférés *(jogosultsághiány miatt)* megtagadva.

A NET parancs a Windwos NT-ben és a Windows 2000-ben már nem a DOS-ban megszokott egyszerű hálózatátirányítási funkcióval bír, használata mindig egy segédparaméterrel történik.

Forrás : Netacademia Windows 2000 Lista

Megértő gépek

A Csillagok háborújának 6 millió nyelvet beszélő protokolldroidja, az űrszekerek univerzális fordítógépe, az űrodüszszeia lágy hangú, de engedetlen központi számítógépe és Asimov emberszabású Daneelje, akit legfeljebb a szlengszavakkal lehetett zavarba hozni még mindig sci-fi. A természetes nyelvek feldolgozását fürkésző kutatók azonban rendületlenül dolgoznak a szinte végtelennek tűnő távolság csökkentésén. Feladatuk nem könnyű, sokak szerint pedig egyenesen lehetetlen. Igaz ugyan, hogy Bill Clinton két éve nyilvánosan ígéretet tett arra, hogy a közeljövőben valós idejű fordítógépekkel rukkolnak elő, abban azonban egy kicsit sem vagyok biztos, hogy ezek a szó nemes értelmében vett fordításokat készítenek majd.

De miben is áll a feladat nehézsége? Hiszen minden ember képes erre. Nem dramatizáljuk kicsit túl a feladat komplexitását? A természetes nyelvek elképesztően bonyolultak, ráadásul bizonyos értelemben önálló életet élnek. Számunkra ez kiváló kommunikációs eszköz, de a gépek *(és főként a megvalósításukon fáradozó kutatók, mérnökök)* számára maga a pokol. Ember legyen a talpán, aki képes megfigyelni, hogy mi zajlik le egy beszélgetés közben az elméjében, merthogy erre más módszer egyelőre nem létezik. Egy szöveg értelmezése nemcsak a mondatok szavainak egyszerű szótári elemzését jelenti. A többjelentésű szavak, a szövegkörnyezet, az átvitt értelmű megfogalmazások, a nyelvtani és szintaktika hibák felismerése stb. alaposan megnehezítik azt, ami amúgy is nehéz: a természetes nyelv formalizálását.

Szánt. Ez most ige vagy egy főnév tárgyraggal? És ha ige, akkor jelen idejű, vagy múlt idejű, merthogy a jelentése ráadásul még ettől is függ. Utóbbi esetben, ha például angolra fordítjuk, akkor a cselekvő nemét is jó lenne tudni. A kérdés eldöntése matematikai értelemben túl bonyolult ahhoz, hogy akár csak az érzékeltetésével is meg mernék itt próbálkozni.

A Microsoft viszont úgy döntött, hogy vállalkozik rá. 1991-ben megalapította első három kutatócsoportját, melyeknek egyike azóta is a természetes nyelvek feldolgozásával *(NLP – Natural Language Processing)* foglalkozik. Létszáma mára már 30 fő fölé emelkedett, és 1997-ben létrehozták azt a több mint 100 főt számláló csoportot, amelynek az a feladata, hogy a kutatási eredmények termékekben is megtestesüljenek. Az NLP csoport igazi különlegességnek számít, mert olyan általános megoldást próbál kifejleszteni, ami minden nyelvnél és alkalmazástípusnál felhasználható – olyanoknál is, amikről ma még nem is tudunk. A Microsoft nem először tűzi ki magának célul azt a jól bevált módszert, hogy mindent saját fejlesztéssel, az alapoktól kiindulva épít fel.

Az NLP rendszer alapját az elemzés *(parsing)* képezi, amely a gép számára értelmezhető formába önti a szöveg tartalmát. Ennek birtokában már viszonylag egyszerű információt keresni, fordítást készíteni vagy egy természetes nyelvi hatást keltő párbeszédet folytatni. Ezt segíti a MindNet névre keresztelt adatbázis, amelyet kifejezetten logikai képletek tárolására és keresésére fejlesztenek. A MindNet a precedenselvű feldolgozás területét képviseli, miszerint a számítógép a szöveges bemenetet korábbi információk figyelembevételével elemzi. Az adatbázist különféle dokumentumok előzetes vizsgálatának eredményeivel töltik fel. Ezek egyszerűsített gráfok, amelyek a mondatokon belüli szavak szemantikai kapcsolatrendszerét térképezik fel. A módszer általánossága többek között abban rejlik, hogy a gráfokat előállító kód minden nyelvnél ugyanaz. Az NLP rendszer tehát ezeket a logikai formulákat használja a szavak közötti kapcsolatok egy nyelven belüli vagy nyelvek közötti keresésére. A szavak közötti tárolt kapcsolatok adják az alapot ahhoz, hogy egy természetes nyelven feltett kérdésre válaszolni lehessen. A MindNetet több szótárral, és a Microsoft Encarta enciklopédiájával töltötték fel, hogy a megértés minél tökéletesebb legyen. A rendszer jelenleg a szavak közötti kapcsolatok 25 fajtájának felismerésére képes *(például tárgy, hely, idő)*.

A természetes nyelvek feldolgozásának képessége utat nyithat például a valósidejű fordítógépek és a felhasználói igényekhez alkalmazkodó, különféle információs forrásokon alapuló összefoglalók készítéséhez. Erre bizony nagy szükségem lenne akkor, ha például egy 100 oldalas japán dokumentumot kellene egyik napról a másikra elolvasnom. Az alkalmazások skálájának csak az emberi képzelőerő szabhat határokat.

És mikor lesz ebből valami? Bármilyen meglepő, a csoport eredményeit folyamatosan próbálják adaptálni a termékekhez. Egy profitból élő cégnél ez nem is működhet másként. A Microsoft az NLP esetében hosszú távra rendezkedett be: a kutatókat a nagy cél elérésére nagy szabadsággal ruházták fel, a „részeredményeket" pedig ügyesen beépítették a termékeikbe. A Microsoft Word-ben található helyesírásellenőrzőt 1997-től az NLP fejlesztések jóvoltából használhatjuk. Korábban ezt a feladatot egy másik cég terméke látta el.

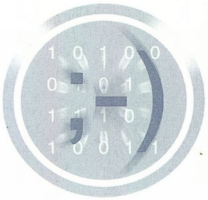
1999-ben az NLP csoport készítette el a Microsoft Encarta enciklopédia keresőmotorját. A felhasználók ebben már természetes nyelven is megfogalmazhatják a kérdéseiket, amire általában jobb választ kapnak, mintha a hagyományos módon, kulcsszavakból összeállított logikai lekérdezést használnának. Az NLP további fejlesztése magától értetődően ezeknek a programoknak a hatékonyságát is növeli majd. A közeli tervek között szerepel, hogy ezek az alkalmazások más nyelven is elkészüljenek. A Microsoft ugyanis, globális vállalat révén, nagy hangsúlyt fektet arra, hogy a nem angol anyanyelvű felhasználók is minél jobban ki tudják használni a számítógépek adta lehetőségeket. Az NLP csoport az angol mellett jelenleg hat másik nyelvvel is foglalkozik. A gépi fordítás mindemellett jól összefogja a különféle nyelvekkel kapcsolatban végzett fejlesztéseket.

Azt talán korai lenne állítani, hogy küszöbön áll egy újabb technológiai forradalom, de úgy tűnik, hogy a küszöb felé vezető út önmagában is nagyon gyümölcsöző.

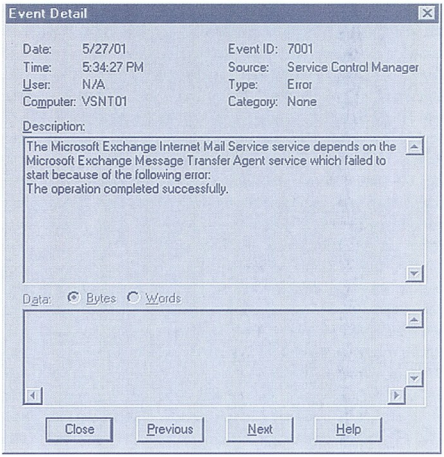
Aki pedig mindezzel elégedetlen lenne, az keresse fel az Interneten Alice-t *(http://www.alicebot.org)*, a 250 IQ-val felprogramozott csevegő robotot, és öntse bitekbe bánatát!

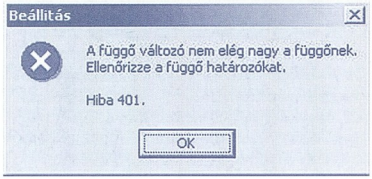
(Zacco)

Szoftverrazzia

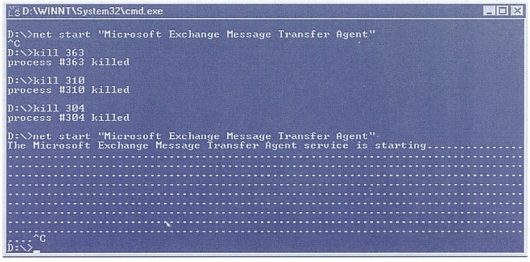


Nyomtatás HP JetAdmin „segítségével"
Beszta János harca a nyomtatásért nem várt kiütéses győzelmet hozott a HP-nak!





IMS konfúzió
Hívjon szakembert, aki még ezt sem tudja megoldani! A probléma egyébként az alábbi ábra tanúsága szerint nem is az IMS-ben, hanem a Message Transfer Agent táján keresendő. Vajon mire vár?



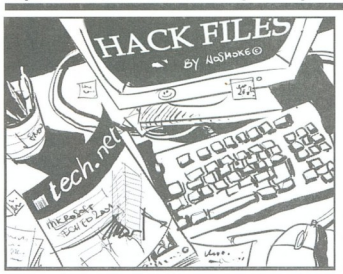
Szoftverrazzia
A következő városokba rendelünk szoftverrazziát novemberre:

Előfizetési szelvény

CÍMZETT: NETACADEMIA KFT.
FAXSZÁM: (1) 261-7145

FAX

Tisztelt Olvasónk!
Lapunk hírlapárusi forgalomba nem kerül, ezért ha kíváncsi megkezdett sorozataink folytatására, kérjük töltse ki és juttassa el hozzánk az alábbi megrendelőlapot. Előfizetőinket szeretettel várjuk Mesterkurzusainkon. Klubtagjaink kedvezményesen vehetnek részt konferenciáinkon, tanfolyamainkon stb.
Kérjük töltse ki ezt az előfizetési szelvényt és faxolja el az (1) 261-7145-ös faxszámra.



Csip úrnak most sikerült kivédenie a pótolhatatlan veszteséget. De mi lesz legközelebb? Hogy Ön, kedves olvasó elkerülje a bajt, egyszerűen csak adja át ezt az előfizetési szelvényt az Ön „Ince" barátjának!

Mottó: Van olyan dolog amit az informatikus nem oszt meg a legkedvesebb barátjával sem. *(tech.net)*

http://technet.netacademia.net/subs • e-mail: terjesztes@netacademia.net • fax: (1) 261-7145

Előfizetem a **tech.net** magazint: ...példányban egy évre (14.784 Ft)
...példányban fél évre (8.064 Ft)

Az előfizetés kezdete:...../...../.....

Előfizető neve:. . . . . .

Cég neve: . . . . . .

Cím: ☐☐☐☐ . . . . . .

E-mail cím: . . . . . .

Telefon: . . . . . .

Fax:. . . . . .

Fizetés módja: ☐ csekken *(postán küldjük)* ☐ átutalással

Kelt:...../...../..... Aláírás:.....

Amennyiben a számlázási cím nem egyezik meg a szállítási címmel, kérjük az alábbi részt is töltse ki!

Számlázási cím: **Szállítási cím:**

....

....

tech.net
A MICROSOFT MAGYARORSZÁG SZAKMAI MAGAZINJA

NetAcademia Kft.
H-1105 Budapest, Ihász u. 13.
Tel: +36 (1) 263-2732 Fax: +36 (1) 261-7145
E-mail: info@netacademia.net

A LEGJOBBAKAT TANÍTJUK!

Microsoft CERTIFIED Partner

tech.net

http://www.netacademia.net/mikulas **2001. december 5.**

Kérjük az oldal alján található **jelentkezési lapot** töltse ki és küldje el fax számunkra: 261-7145 vagy e-mail címünkre: info@netacademia.net!

A belépés díja: 20.000,-Ft*. Előfizetőink kedvezményesen **15. 000,-Ft*** –ért, előfizetők és listatagok **10.000,-Ft***-ért vehetnek részt a Mikulás rendezvényen.

Ebédet 1500,-Ft-ért biztosítunk.

Helyszín: **Central European University**
(1106 Budapest Kerepesi út 87.)

Bővebb információt honlapunkon talál, illetve a következő elérhetőségeinken kérhet:

Tel.: 263-2732
Fax: 261-7145

info@netacademia.net

JELENTKEZÉSI LAP (jelentkezési határidő: 2001. november 22.)

Megrendelő cég neve:	
Résztvevő(k) neve:	
Számlázási cím:	
Telefon / Fax szám:	
E-mail cím	

Kérjük jelölje meg, amennyiben a tech.net magazin előfizetője, vagy valamely levelezési lista tagja:

tech.net előfizető ☐ levelezési lista tag ☐

Ebédet kérek (+1500 Ft!) ☐

Dátum:

Pecsét Cégszerű aláírás

* Áraink az ÁFA-t nem tartalmazzák!

NetAcademia Kft.
H-1105 Budapest, Ihász u. 13.
Tel: +36 (1) 263-2732 Fax: +36 (1) 261-7145
E-mail: info@netacademia.net

A LEGJOBBAKAT TANÍTJUK!

Microsoft CERTIFIED Partner

tech.net

MEGHÍVÓ

A NETACADEMIA INTERAKTÍV* MIKULÁS KONFERENCIÁJÁRA

2001. december 5.

Regisztráció: fél 9-től. A laborok a szünetekben is nyitva lesznek. Megjelenésére feltétlenül számítunk!

1. téma (kezdés: 9:00) **LDAP, LDIF** Előadó: Fóti Marcell	Akkor tekintheti magát az Active Directory urának és parancsolójának, ha ugyanolyan bátran bele mer törölni a címtárba LDP.EXE-vel, mint amilyen bátran módosítgat a REGEDT32.EXE-vel a regisztrációs adatbázisban. Ehhez „csak" alaposan kell ismerni...
Interaktív ++ LDIF és ADSI szkriptek	Objektumok tömeges kezelése NOTEPAD „segítségével" • felhasználók létrehozása, mozgatása a hierarchiában • csoporttagság átállítása • OU készítése és mozgatása stb.

2. téma (kezdés: 11:00) **Golyóálló webkiszolgálók** Előadó: Fülöp Miklós	Alapvető elméleti tudnivalók, mindennapi biztonsági eszközök • az operációs rendszer biztosítása • a nemkívánatos szolgáltatások letiltása • a hozzáférés korlátozása • a kérések szűrése – a naplók elemzése
Interaktív ++ IIS kézimunka	Az IIS telepítése, konfigurálása – a biztonsági eszközök használata (hfnetchk, iislockd, urlscan, stb.) – a nem használt portok lezárása (IPSec policy)

3. téma (kezdés: 14:00) **RSA** Előadó: Fóti Marcell	Minden, amit tudni akarsz a nyílt kulcsú titkosításról, de nem mered megkérdezni • Hogy működik? Mi a matekja? Hogyan használjuk a gyakorlatban? • Biztos, hogy biztos? Mikorra várható az RSA összeomlása? • A Microsoft Certificate Server szerepe
Interaktív ++	Certificate Server telepítése, SSL megvalósítása IIS-en

4. téma (kezdés: 16:00) **Microsoft.NET emberközelben** Előadó: Soczó Zsolt	Mi újság a Matrixban? - CLR tanmese C# vagy JAVA--? - A MS-Sun boxmeccs Mi lesz veletek VB programozók? - A VB jelene és jövője Alkalmazás Puttony.NET – ASP+, az ígéret földje
Interaktív ++ .NET programozás	.NET programozási laborgyakorlatok C# és VB.NET nyelven. Kicsiknek és nagyoknak, kezdőknek és haladóknak különböző összetettségű feladatok (Hello Világ, Webszervizek stb.)

* **interactive** (*Computer Science)* Olyan programokkal kapcsolatos kifejezés, melyek válaszolnak a felhasználók igényeire. A ⇨NETACADEMIA MIKULÁS KONFERENCIA például lehetővé teszi, hogy a helyszínen telepített számítógépes laborban ***~ módon*** mindent kipróbáljunk!

Kihelyezve bejön!

SZÁMALK Hivatalos Microsoft tanfolyamok

Budapesten és kihelyezett formában már az ország több nagyvárosában is!

Kód	Megnezezés	Időt.	Ár	Kezdési időpontok						
1560	**Updating Support Skills to Windows 2000**	5 nap	139 000	okt. 15. nov. 19.	okt. 8. nov. 12.	okt. 1. nov. 5.	okt. 8. nov. 12.	okt. 1. okt. 15.	okt. 8. nov. 12.	okt. 1. nov. 5.
2152	**Implementing Windows 2000 Server and Professional**	5 nap	125 000	okt. 15. nov. 19.	okt. 8. nov. 12.	okt. 1. nov. 5.	okt. 8. nov. 12.	okt. 1. okt. 15.	okt. 8. nov. 12.	okt. 1. nov. 5.
2153	**Implementing Windows 2000 Network Infrastructure**	5 nap	125 000	nov. 19.	dec. 3.	nov. 26.	dec. 3.	nov. 19.	dec. 3.	nov. 26.
2154	**Implementing Windows 2000 Directory Services**	5 nap	125 000	dec. 10.	jan. 7.	dec. 17.	jan. 7.	dec. 10.	jan. 7.	dec. 17.
1572	**Implementing, Administering Exchange 2000**	5 nap	139 000	nov. 19.	dec. 3.	nov. 26.	dec. 3.	nov. 19.	dec. 3.	nov. 26.
1561	**Designing Windows 2000 Directory Services**	3 nap	95 000	dec. 10.	jan. 7.	dec. 12.	jan. 7.	dec. 10.	jan. 7.	dec. 17.
2159	**Deploying and Managing ISA Server 2000**	2 nap	79 000							

További tanfolyamokról és időpontokról érdeklődjön szervezőinknél! Kérésére részletes tájékoztatót küldünk.

Minőség, egyéneknek kedvező konstrukciók, cégeknek partneri kedvezmények!

Microsoft CERTIFIED Technical Education Center

SZÁMALK TOVÁBBKÉPZÉS

SZÁMALK

☎ 203-0304/3050 **www.szamalk.hu/tisza** 1115 Budapest, Etele út 68.