1.344 Ft

II. Évfolyam 12. szám

ISSN 1586-5185 12

tech.net

A Microsoft Magyarország szakmai magazinja

Kellemes Ünnepeket és
Sikerekben Gazdag Boldog Új Évet
Kívánunk minden kedves olvasónknak!

„Kijutni csak akkor
tudunk, ha a bálvány
talpa alól kihúzzuk
a térképet.
Na ez az RSA.
Aki nem tudja…"
(Indiana Jones)

19. oldal

RSA algoritmus

infoBYTE

INFOKOMMUNIKÁCIÓ
A távközlési piac liberalizálása és a mobiltelefóniában várható generációváltás érdekegyeztetési törekvéseiről tudósít ez a rovat.

CÍMLAPSZTORI
A hónap vezető cikke új technológiákról, megoldásokról.

NEMZETKÖZI SZEMLE
Külfödi hírek, kitekintés.

E-KORMÁNYZAT
Az Informatikai Kormánybiztosság 2001– 2002-ben összesen 36 különféle programot koordinál. Az információs társadalom kiépítésének lépcsőfokai ezek.

INFORUM
Az Inforum célja, hogy párbeszédet folytasson a szakma és a kormányzat között. Aktív szerepet vállalt a szerzői jogi, az egységes hírközlési és az elektronikus kereskedelmi törvény megalkotásában.

EU-INFORMATIKA
Ebben a rovatban nem elsősorban a technológiára, hanem a megvalósult projektekre, az EU-kompatibilitás kérdéskörére, pályázatokra koncentrálunk.

CIO.INFOBYTE.HU
Az infopen.hu webmagazin és az infoBYTE közös rovata, kifejezetten IT vezetők számára. A rovat egy aktív CIO-val készült átfogó interjúval indul, amelyet stratégiai jellegű technológiai áttekintések, szakcikkek követnek. A rovat hangsúlyos részei a vállalati IT megoldásokat bemutató esettanulmányok, de interjúk, konferenciatudósítások formájában a piac meghatározó megoldásszállító cégeinek üzleti és termékstratégiájának bemutatása is helyet kap.

KARRIER
Tapasztalatok szerint három-négy évente változtatunk mi, informatikusok állást, szakmát vagy szakirányt, és ez a szám a jövő évtizedekben valószínűleg csökkenni fog.

KONZOL ELŐTT
E rovatunkban olvasóink nevében és helyett Novell szakértőket kérdez a szerző.

DR. WATSON
A NetAcademia-féle mélyvíz-tanfolyamokra iratkozhatnak be azon olvasóink, akik Dr. Watson nyomában járnak.

MÉRLEG
Hardver- és szoftvertesztek röviden, velősen.

PROCESSZOR
Sokakat érdeklő CPU-újdonságok mélységei a fejlesztéshez közel álló szakértők tollából.

Kérjen mintapéldányt: minta@infobyte.hu

2001 December

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@technetklub.hu

Kiadja és terjeszti
a **NetAcademia Kft.**
Terjesztési, előfizetési információ:
Tel.: 263-2732
terjesztes@netacademia.net
Megjelenik havonta, ára 1.344 Ft
Példányszám: 4.000

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

Hirdetésfelvétel:
Bársonykalapács Marketing
Felelős: **Balogh Zoltán**
Tel.: 489-4665
Fax: 489-4660
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

Júzerstílusban

Júzerstílusban

Érdekes világ ez! Először kínainak tűnik az egész. Szakemberek beszélgetnek egymással, a júzer meg ül bambán és hevesen csattogtatja a klaviatúrát. Heves vitatkozás bontakozik ki a háttérben, a júzer pedig megpróbál kihámozni valamit a hallottakból - „ők tényleg más nyelvet beszélnek"! Júzerünk arra hivatkozik, hogy még csak most vágott bele a számítástechnika új nyelvének megtanulásába és a szükséges készségek elsajátításába, lesz még elég ideje, hogy megértse azt, felmentést ad tehát magának. Különben is, nem lehet mindent érteni a világ dolgaiból!

Felmentés megvan, már csak júzerként kell jól helytállnia főszereplőnknek. Megtanulja miért kell folyamatosan mentéseket végezni, miután elveszíti 10 oldalas gépelt dokumentumát sőt azt is, hogy az egeret valójában nem a képernyőn kell mozgatni, hogy be tudja célozni a megfelelő ikont :-) Idővel össze tud rakni egy gépet, már ami a perifériák csatlakoztatását jelenti; ebben a folyamatban - persze a gyengébbek kedvéért - a segítségére vannak a doboz hátulján lévő csatlakozók színkódjai is.

Számtalan dolgot meg tud már oldani egyedül. Már nem futkorászik és nyaggatja a rendszergazdákat, ha esetleg nem megy a nyomtatás - büszkén keresi meg Word dokumentumának nyomtatási beállításait vagy a nyomtató beállításait a start menüből. Költőien:

Change drum?
Ismerem!
Toner low?
Cserélem!

De akkor sem zavarja a rendszergazdát, ha esetleg semmi sem úgy működik a gépen, mint mondjuk tegnap este - tisztában van vele, hogy egy restart mindent megold! Vagy ha nem, hát a regedt32.exe. Ha gondja van, csak besétál a HKLM/Software/MSLicensing alá és

Törli, amit törölni kell,
Tudja, mi fán terem
a dúr akkord,
a ctrl+alt+del.

A „How are you?" - kezdetű levelet megtanulja csatolt dokumentumával együtt shift+del-lel megsemmisíteni.

Feliratkozik a NetAcademia levlistáira - mondjuk csak az OFF jöhet szóba - de a nyelvezettel még így is gondok adódhatnak *(de mi az a: rulez, warez, TOFL, szvsz meg a többiek? Hhűű!)* Néha megkapja a júzeres vicceket! Olykor érti is a poén tárgyát, amelyiket meg nem - hát a belső szobából kiáramló jókedv azért készteti mosolygásra, mert tudja, hogy ez bizony róla: a júzerről, és a tudatlanságáról szól. Hadd nevessenek! Én is röhögök a szőke nős vicceken ! :-)

A számítástechnika világában, júzerként inkább a „periférián" csücsülünk, de örömmel vesszük a kis magyarázatokat, amivel munkánk könnyebbé tehető, amivel valamivel többet megérthetünk a számítástechnikai csiki-csukikból. Persze ehhez júzerként is folyamatos tanulásra van szükség, illetve képzett szakemberekre, akik megoldják a „gondjainkat" helyettünk és értünk.

tech.net

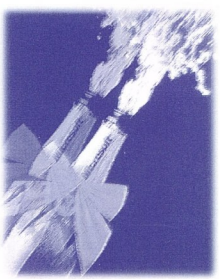
A tech.net idén ünnepelte 1 éves születésnapját, amit gyerekpezsgővel és tortával ünnepeltünk. Rengeteget köszönhetünk olvasóinknak, akik vették a fáradságot, hogy jelezzék nekünk pozitív és negatív észrevételeiket és buzdítsanak bennünket! Tartsák meg jó szokásukat, mert minden észrevétel számít.

E számunkban teret engedtünk fantáziánknak és jó hangulatunknak is: az ünnepi készülődés és a szilveszteri jókedv jegyében az utolsó négy oldalt komolytalanságoknak szenteljük.

Üzenetem végére érve, júzerként és egy emberként *(a júzer is ember!)* kívánok mindenkinek még sok türelmet hozzánk: júzerekhez, sok viccet rólunk, és kitartó munkát.

Szerkesztőségünk nevében kívánok kellemes karácsonyi ünnepeket, boldog új évet és zökkenőmentes áttérést a 2002. évre. Idén nincs bug? Tavaly mintha lett volna valami. Vagy tavalyelőtt...?

Kovács Barbara
barbi@netacademia.net

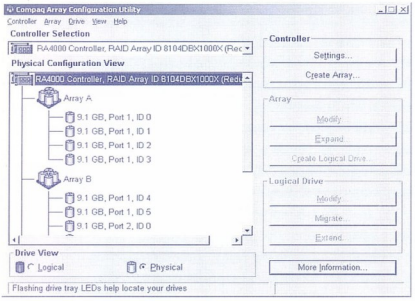


Farkasokkal táncoló *(II. rész)* Cluster a gyakorlatban

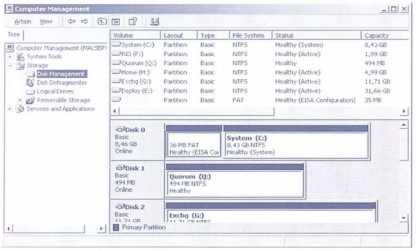
Miután elsajátítottuk az alapfogalmakat és összeállítottunk egy Achilles sarkoktól mentes konfigurációt, megkezdhetjük az előkészületeket a telepítésre, majd el is végezhetjük azt.

Előkészületek

El kell döntenünk, hogy hány erőforráscsoportot, illetve virtuális szervert szeretnénk létrehozni. A számuk idővel persze változhat, de most azért van erre mégis szükség, mert minden egyes csoporthoz külön lemezt kell rendelni. A lemez *(physical disk)* egy erőforrás, az pedig egyszerre csak egy csoporthoz tartozhat. A cluster bizonyára egy lemezalrendszert ér el, amely egy vagy több redundáns lemeztömböt *(disk array)* tartalmaz. Ezek a lemeztömbök valamilyen hardveres redundanciával kell, hogy rendelkezzenek attól függően, hogy milyen alkalmazás használja majd őket. A lemeztömbök fölött a diszkalrendszer számára logikai lemezeket lehet definiálni. Ezeket a logikai lemezeket az operációs rendszer fizikai lemezként fogja felismerni. Az alábbi ábra egy lehetséges lemeztömbfelosztást mutat be, a következő pedig azt, miként látja ugyanezt a Windows 2000.



A lemeztömbök a lemezvezérlő felől nézve...



... és a Windows 2000 szerint.

Valamennyi általam ismert lemezvezérlő képes arra, hogy működés közben új lemezeket fogadjon a többbe, s azokon újabb logikai *(a Windows számára fizikai)* lemezeket hozzon létre. Jóval kényesebb kérdés a már meglévő logikai lemezek megnövelése. Ez sem lehetetlen, de általában már csak borsos árú kiegészítőszoftverekkel lehet elvégezni. Ennek hiányában nem kis erőfeszítés lesz a megtelt lemezt nagyobbra cserélni.

Fontos kiegészítés, hogy a definiált diszkek éppúgy viselkednek, mintha tényleges lemezek volnának. Ez többek közt azt is jelenti, hogy rajtuk *(elsődleges)* partíciókat lehet létrehozni. Az erőforráscsoportok azonban nem ismernek „partíció" erőforrást, tehát ha egy lemezen több partíció van, akkor mindegyik azonos csoportba fog kerülni. A lemezeket NTFS formátumra kell formázni, és tilos átkonvertálni dinamikussá. Mindezen ismeretek alapján azt javaslom, hogy a cluster bőven tartalmazzon szabad merevlemezterületet. Egyrészt fölös terület szükséges a hardveres redundancia biztosítására. Másrészt minden egyes definiált lemez esetén lehetővé kell tenni az adatok számára a „szaporodást és sokasodást". Vagyis minél több lemezt definiálunk, annál több szabad hellyel kell számolni, hiszen a szabad hely felaprózódik közöttük. Például: ha a DHCP és a WINS szolgáltatást külön virtuális gépen definiáljuk, akkor külön diszket kell hozzárendelni az erőforráscsoportokhoz, tehát mindkét szolgáltatás jövőbeli tárkapacitásigényét külön kell megbecsülnünk. Ha egy csoportban helyezzük el őket, akkor lehetséges, hogy a számolt szabad kapacitás kisebb lesz, ezáltal a közös diszk sem lesz akkora, mint az első esetben.

Azt kell látni, hogy két, egymásnak ellentmondó igényt kell kielégíteni. Minél több virtuális csoportot kell létrehozni ahhoz, hogy a szolgáltatások egymástól függetlenek legyenek, ugyanakkor ez jóval több erőforrást igényel *(memóriát és főleg tárolóhelyet)*, ez viszont erőforrás-, végső soron pedig pénzpazarláshoz vezet. Ha a pénztárca és a konfiguráció igényeit össze akarjuk hangolni, érdemes bizonyos szolgáltatásokat összevonni, és csak néhány, „nagyobb" virtuális szervert létrehozni. Azt javaslom, hogy a virtuális szerverekhez az egyes szolgáltatásokat a karakterisztikájuk alapján soroljuk be. Nagyon leegyszerűsítve kétféle szerverszolgáltatás létezik: a fájl- és az alkalmazásszolgáltatás. Az előbbi intenzíven veszi igénybe a merevlemezeket, I/O csatornákat és hálózati kártyákat, míg az utóbbi rengeteg memóriát, és magas órajelű, gyorsabb processzorokat kíván. A Windows 2000-t – sőt már a korábbi NT verziókat is – egyszerűen lehet hangolni ezekre a feladatokra. Ha az állomást inkább fájlszolgáltatás jellegű feladatokra optimalizáltuk – ilyen a fájlmegosztás, a nyomtatás, az IIS – érdemes odaköltöztetni azokat a virtuális gépeket, amelyek erőforrásai hasonló feladatokat látnak el. Fordítva hasonlóképp: az alkalmazásjellegű erőforrásokat tartalmazó csoportokat az alkalmazáskiszolgálásra hangolt node fogadja be.

Ennyi tudás birtokában már papírra lehet vetni a virtuális szerverek tervét. Szükség van egy névre *(15 karakternél nem*

hosszabb, a NetBIOS nevekre vonatkozó korlátok figyelembe vételével kell megalkotni), IP címre *(kizárólag statikus cím lehet)*, valamint a szükséges szolgáltatásokra és merevlemez területekre. Az egyes merevlemezeket mindkét állomáson szigorúan azonos betűjellel kell ellátni. Egyes dokumentációk azt javasolják, hogy abc sorrendben visszafelé érdemes betűjeleket adni, de ez nem kötelező. Jómagam előnyben részesítem a funkció szerinti jeleket, pl. Q=quorum, E=Exchange stb.

Utolsó lépésként az előkészítés során egy megfelelő jogosultságokkal rendelkező fiókot kell létrehozni, ennek a kontextusa alatt működik majd mindkét állomáson a clusterszolgáltatás. Több forrás is egyértelműen állítja, hogy a fióknak adminisztrátori fióknak kell lennie, ez azonban nem teljesen pontos. Személy szerint nagyon nem szeretem, ha ritkán változó jelszóval rendelkező fiókok *(és a szervizek fiókjai általában ilyenek)* indokolatlanul nagy hatalomhoz jutnak. Ha valaki el szeretné kerülni, hogy tartományi rendszergazdává léptesse elő a clusterfiókot, akkor a következő jogok megadásával elkerülheti ezt:

- Lock pages in memory
- Log on as a service
- Act as part of the operating system
- Back up files and directories
- Increase quotas
- Increase scheduling priority
- Load and unload device drivers
- Restore files and directories

Az első három jogosultsággal alapértelmezetten a tartományi rendszergazdák sem rendelkeznek. A jogokat vagy tartományi vagy OU *(organization unit)* szinten, de akár az állomásokon is meg lehet adni. Ez utóbbi a legbiztonságosabb, habár sok munkával jár. Az OU szint egy elfogadható kompromisszum, feltéve, hogy az OU direkt a clusterek adminisztrálásának elkülönítésére jött létre.

A fiókot a Windows NT Account tartományban vagy az Active Directory-ban kell létre hozni, s ez logikus is, hiszen mindkét állomáson elérhetőnek kell lennie ennek a fióknak – a tartományi felhasználók ilyenek. A fiókot egyébiránt a szervizaccountokhoz hasonlóan érdemes kezelni *(a jelszó nem jár le; a felhasználó nem változtathatja meg a jelszót; hosszú, komplex és erős jelszót kell megadni stb.)*

Az állomásoknak kötelezően tartományi tagoknak kell lenniük, de ezen belül a szerepük nem korlátozott, lehetnek tagkiszolgálók és tartományvezérlők is. A szerepekről a későbbiekben még részletesen beszélünk. Most annyit érdemes tudni, hogy mindkét állomásnak azonos szerepet kell adni: vagy mind a kettő legyen tagkiszolgáló, vagy tartományvezérlő. A szerepeket a clusterszolgáltatás telepítése előtt kell beállítani.

A telepítés

Elég sokat tudunk már, fogjunk bele a telepítésbe. Feltételezzük, hogy mindkét állomáson működik a tartomány tagjaként a Windows 2000 Advanced Server. Vezérlőpult Programok hozzáadása Windows komponens telepítés Cluster service.

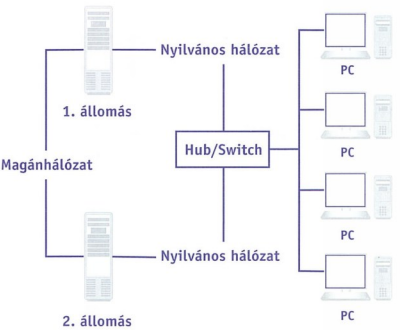
Az első párbeszédpanel arra hívja fel a figyelmünket, hogy a konfigurációnak, amelyre a fürtszolgáltatást telepítjük, szerepelnie kell a Microsoft által kiadott hardverkompatibilitási listának azon szűkített változatán, amely a kiszolgálófürtökre vonatkozik. Ez éles rendszer esetén evidens, a lehetséges szállítókat meg kell kérni, hogy igazolják eszközeik jelenlétét a listán, de ha kell, erre mi is képesek vagyunk. Ha olyan rendszerre telepítjük a clustert, amely nincs rajt a HCL-en, akkor a Microsoft részéről semmiféle segítséget nem kaphatunk. Tekintve a bekerülési költségeket és a felálló rendszer kritikusságát, ezt nem érdemes kockáztatni. A panelről az „I understand" *(Megértettem)* gomb megnyomása után léphetünk csak a következőre.

A fürttelepítő varázsló következő kérdése, hogy hányadik állomást telepítjük. Értelemszerűen kell válaszolni. Az első állomás esetén több teendője van a varázslónak, hiszen létre kell hozni a majdani „közös tudást". Ha az első node-ra telepítjük a fürtöt, akkor meg kell adnunk a cluster nevét. A fürtnév az első virtuális kiszolgáló neve. A művelet végén egy olyan erőforráscsoport jön létre, amelybe a cluster neve és IP címe, valamint a quorum disk kerül. Ezt a virtuális szervert adminisztrációs célokra kell használni, és bár lehet, én nem javaslom, hogy további erőforrásokat hozzunk létre benne. A fürt nevére a NetBIOS korlátok érvényesek.

A következő lépés a fürtszolgáltatás fiókjának és jelszavának megadása. Ha még nem adtuk meg a fiók jogait, akkor ezt a telepítő ellenőrzi. Figyelem! A varázsló a korábban felsorolt jogokból csak az első hármat adja meg automatikusan, a többit meglévőnek feltételezi *(vagyis, hogy Domain Admin a fiók)*. Ha nem akarjuk, hogy rendszergazda legyen, akkor a jogait előre be kell állítani.

A fiókinformációk után a varázsló a felügyelt lemezekről érdeklődik. A szakirodalom azt ajánlja, hogy minden lehetséges diszket azonnal rendeljünk a fürt felügyelete alá. Ez rendjén is lenne. A telepítés után viszont minden lemez külön csoportba kerül, ami nem biztos, hogy kívánatos. Nosza átdobáljuk majd őket a megfelelő helyre. Csakhogy a „physical disk" erőforrás nem dobálható át úgy, ahogy azt gondoljuk. Előbb offline állapotba kell tenni, majd átmozgatni, majd újra online állapotba kell helyezni. Nálam még így is előfordult, hogy rejtélyes hibaüzenetet kaptam. A tudásbázis áttanulmányozása után kiderült, hogy a lemezeket úgy kezeltem, mint bármely más erőforrást, és ez nem volt helyénvaló. Azt javaslom tehát, hogy csak a quorum disket hagyjuk a felügyelt lemezek között, és csak a telepítés után általunk létrehozott virtuális kiszolgálókhoz adjuk hozzá a többi lemezt.

Itt érdemes megemlíteni, hogy a quorum disk legyen feltétlenül dedikált, tehát ne használjuk semmi másra. A mérete legalább 50 MB kell, hogy legyen, a MS ajánlás 500 MB, amelyet érdemes betartani.



A varázsló ezután sorra veszi a lehetséges hálózati kapcsolatainkat, és arra kér minket, hogy határozzuk meg, milyen szerepet szánunk nekik. Háromféle szerep létezik: magánhálózat *(private network)*, nyilvános hálózat *(public network)* és vegyes hálózat *(mixed network)*. A fenti ábra segít eligazodni a fogalmak között.

Amint az látható, egy normál fürtkonfigurációban minden állomás legalább két hálózati csatolóval rendelkezik. Az első pár csatoló vagy egy hubon keresztül, vagy egy crosslink kábel segítségével közvetlenül van összekötve, míg a másik csatoló az ügyfelekkel tartja a kapcsolatot. A magánhálózaton olyan forgalom zajlik, amelyet a két állomás a folyamatos szolgáltatás érdekében folytat. Ezek: a szerver szívhang, az állapotinformációk replikációja, a clusterparancsok, valamint a cluster működésre felkészített alkalmazások állomások közötti kommunikációja.

Ha a nyilvános hálózat egyben a magánhálózat forgalmát is bonyolítja, akkor azt vegyes hálózatnak nevezzük. A fentiekből következik, hogy a két hálózati csatoló nem kötelező, csak nagyon ajánlott. Ajánlott továbbá, hogy ne nyilvános hálózatot definiáljunk a magán mellé, hanem vegyeset, mert így akkor is zavartalan maradhat a clusterkommunikáció, ha a magánhálózat valamilyen oknál fogva meghibásodik *(egy Achilles sarokkal kevesebb)*.

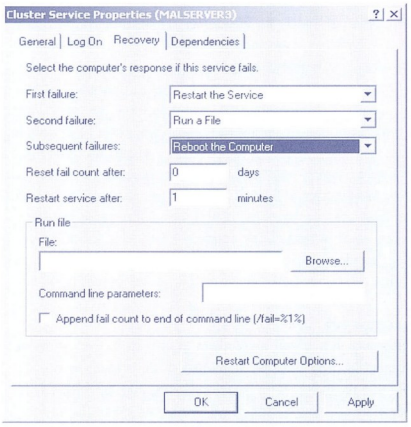
A magánhálózatok konfigurálásának további tudománya is van, amelyet a varázsló leállása után rögtön érdemes is alkalmazni. Javaslom, hogy nevezzük át a kapcsolatot megjegyezhető névre *(Pl. heartbeat)*. A hálózatok beállításánál az úgynevezett Binding Ordernél a magánhálózatokat a nyilvános mögé kell sorolni, másodikként. A csatolót olyan címmel kell ellátni, amely nem fordul elő a nyilvános hálózaton. *(Ha pl. 10.x.x.x a hálózati cím, akkor a private network esetén használjuk a 192.168.0.x címeket, mindegyik állomáson természetesen mást.)* Alapértelmezett átjáró, DNS és WINS bejegyzések ne legyenek a TCP/IP konfigurációban, továbbá ki kell kapcsolni a NetBIOS-t is erre az adapterre vonatkozóan. *(De csak erre!!)* Ha crosslink kábelt használunk a magánhálózatban, akkor állítsuk be a következő regisztrációs értéket:

```
HKLM\System\Currentcontrolset\Services\Tcpip\
Parameters\DisableDHCPMediaSense
REG_DWORD, értéke: 1
```

Ennek magyarázatát a Q254651 cikkben lehet megtalálni. Végezetül a TCP/IP beállító panel DNS fülén ki kell kapcsolni a „Register this connecton's addresses in DNS" kapcsolót. Ha ezt nem tesszük, akkor a cluster regisztrálja a magánhálózat hálózati kártyájának címét is, amelytől viszont sohasem fog válasz érkezni a külső kérésekre, és vehetjük elő a Network Monitort, hogy megállapítsuk, miért nincs néha a cluster a hálózaton, amikor ott van.

Térjünk vissza a varázslóhoz. Ha egy magán- és egy vegyes hálózatot adtunk meg, akkor még arra a kérdésre kell válaszolnunk, hogy milyen sorrendben használja a szerviz a hálózatokat a fürtforgalomra. Ezzel befejeződik a kérdezősködés, a varázsló telepíti a szolgáltatást, elvégzi a regisztrációs bejegyzéseket, és létrehozza a quorum disken a \MSCS könyvtárban a quorum adatbázist. Ha mindent elvégzett, megpróbálja elindítani a fürtszolgáltatást. Az előkészítés precizitásától függően ez általában sikerül is neki. Ez a néhány tíz másodperc elég arra, hogy észrevegyük, a cluster telepítésének inkább NT4-es formája és érzete van, mint Windows 2000-es. Ez a „gyanú" később igazolódni fog: a Windows 2000-ben nagyon sok mindent átírtak, és nagyon sokat fejlesztettek, beleértve a clusterszolgáltatást is, mégis maradt jócskán tennivaló a következő kiadásig.

A varázsló elvégezve feladatát leáll. Nekünk azonban akad további tennivalónk. Ellenőrizni kell, hogy a szolgáltatás valóban elindult-e, s ha igen, segíthetünk rajta, hogy ne nagyon akarjon többet állni. Vadonatúj Windows 2000-es szolgáltatás a szervizek működésének helyreállítása, és nemcsak a cluster szervizre.



Windows szolgáltatások védelme

Azt ajánlom, hogy még a virtuális szerverek üzembe állítása előtt teszteljük a lehetséges helyreállítási eseteket. A cluster szerviznél a helyreállítás akkor fontos, ha vannak olyan csoportok, amelyek kizárólag egy állomáson futnak *(például licencokok miatt)*, vagy maga a fürt egy állomásból áll *(mert a másik csak jövőre érkezik)*. Az újraindítás lehetőségét csak akkor szabad igénybe venni, ha nincsenek fürtön kívüli szolgáltatások az állomáson, vagy azok más módon redundánsak. *(pl. Active Directory)*

A teljességhez hozzátartozik, hogy két állomás esetén a másikon is le kell futtatnunk a varázslót. Ekkor már egy meglévő clusterhez kell csatlakoznunk, a szervizfióknak egyeznie kell az első állomásnál megadottal. A diszkekre vonatkozó kérés kimarad, mert azt a cluster adatbázis tartalmazza, válaszolni kell viszont a hálózati csatolók szerepét firtató kérdésekre. A sikeres csatlakozásról a varázsló jelentést tesz. Íme, megszületett a farkasfalka első tagja. A következő alkalommal megismerkedünk a Cluster Administratorral és létrehozzuk az első virtuális kiszolgálóinkat.

Lepenye Tamás (MCSE)
lepenyet@mal.hu

Idő-szinkronizáció

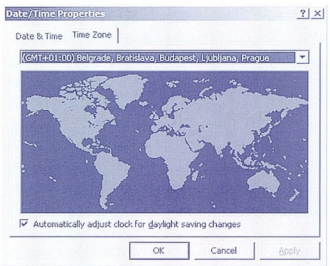
A Windows 2000 operációs rendszer alapértelmezett hálózati azonosítási protokollként a Kerberos V5-öt használja. Ennek tökéletes működéséhez az időszinkronizáció elengedhetetlenül fontos, mivel a rendszer időbélyegeket *(TimeStamp)* használ a kiosztott „jegyeknél" *(Ticket)*. Teszi mindezt azért, hogy ne lehessen a Kerberos Authentication Servert árverni hálózatból elkapott, és később visszajátszott csomagokkal. Ha az ügyfél és a kiszolgáló órája öt percnél jobban eltér egymástól, a bejelentkezés meghiúsul. Szerencsére a Windows 2000-ben az órák egyeztetéséhez szükséges szolgáltatás előtelepítve megtalálható, melynek neve: Windows Time Service vagy - kicsit hétköznapibban - w32time. Ez a szolgáltatás csendben teszi a dolgát, csak a kezdő rendszergazda őrül meg, hogy a hálózat egyik gépén képtelen beállítani az órát, mindig siet 9 órányit. Na az a gép Tijuana időzónájában tanyázik, s a központi géptől vett pontos időt rendre eltolja a saját időzónájának megfelelő értékkel. Az idő helyessége, az idő pontossága nem csak a Kerberos V5-nek fontos, hanem a csoportmunka egyik létfontosságú alapja is egyben.

A pontos idő

Előfordult már olyan a munkám során, hogy „A" városban kiosztottak egy Outlook feladatot a „B" városban dolgozó kollegának. A két város közti távolság 200 km, a találkozó helyszíne „A" város volt. A „B" városban dolgozó kolléga a KRESZ szabályait többször is áthágva taposta a gázpedált, hogy idejében „A" városba érjen, de hiába ért oda a megbeszéltnek hitt 8 órára; a találkozót ugyanis 10:00-ra tervezték. Miért történhetett ez meg? A hibát nem a felhasználóban kell keresni, és az idő pontatlansága sem okozhat ilyet. Azonban Einstein óta tudjuk: az idő relatív fogalom. Einstein ugyan még azt hitte, hogy a pontos idő a sebességtől függ *(minél jobban nyomja a kolléga a gázpedált, annál lassabban telik)*, de mi már tudjuk: bizony függ az az időzónától is. A pontos idővel kapcsolatban tisztázni fogjuk az időzóna szerepét, valamint megnézzük azt is, hogy a Microsoft Exchange kiszolgáló és az Exchange kliens pontos ideje között milyen összefüggés mutatható ki.
Hogy a fenti eset senkivel ne fordulhasson elő, illetve egyetlen rendszergazdának se okozzon kellemetlen perceket egy ilyen helyzet, ajánlom cikkemet mindenki figyelmébe.

Időzóna

Magyarország kis ország, s ebből következően azt hihetjük, hogy ha Záhonyban is 11 óra van, meg Hegyeshalomnál is, akkor a világ minden pontján ennyi az idő. Aki sokat utazik, észreveheti, hogy a nap más pillanatban kel és nyugszik itthon, Franciaországban és mondjuk Japánban. Bár ezek az apró jelek arra mutatnak, hogy a Föld gömbölyű, ezt mégsem kell tudnunk az időzóna fogalmának megértéséhez. Ha a Föld történetesen lapos, akkor is meg kell oldanunk a napfelkelték eltolódásának problémáját, amire az emberiség azt találta ki, hogy ahol később kel a nap, ott később van reggel nyolc. A modern kompjútereken nem kell áttekerni a mutatókat, hanem az operációs rendszernek meg lehet mondani, hogy vegye figyelembe helyváltoztatásunkat. Ezt az alábbi ábrán tehetjük meg. A kép alján bepipálható a téli-nyári időszámítás-váltás automatikus lekövetése, mely már 1996 óta nem okoz problémát a gépnek. *(1995-ben még gond volt vele, lásd később...)*



Az időzóna beállítása

Ide az időt!

Mindenféle furfangos szolgáltatás nélkül is lehetséges a Windowsos számítógépek óráinak egyeztetése, csak a megfelelő parancsot kell ismerni, és használni. A net parancsnak egy alparancsa a time, aminek segítségével Windows NT és Windows9x alatt is elérhető a manuális időszinkronizáció. A net time használatával minden előzetes telepítés nélkül akármelyik géptől elkérhetjük a „pontos" időt. Egyszerűsített használata a következő:

```
net time \\kiszolgáló /set /y
```

Sajnos a bejelentkezett felhasználónak rendelkeznie kell a helyi idő megváltoztatásának jogával *(Change the System Time)*. A net time a következő módokon használható még:

- **Net time /set /y:** a munkaállomás óráját a gép tartományában található valamely kiszolgálóval szinkronizálja
- **Net time /domain:domain_neve /set /y:** a munkaállomás óráját a megadott tartomány valamely kiszolgálójával szinkronizálja

Automatikus használata nehézkes, mert nem ritka, hogy a felhasználók nem rendelkeznek időállítási jogosultsággal, ámde mi mégis szeretnénk azt elérni, hogy a helyi időt rendszeresen szinkronizáljuk egy általunk pontosnak vélt időkiszolgálóval. Az előző parancsok akár egy LogonScript-ben is használhatók, de ha nincs jog, nincs jog *(a LogonScript a bejelentkezett felhasználó jogosultságával hajtódik végre)*.

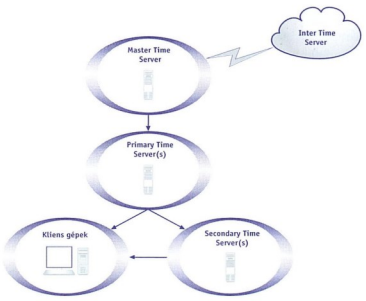
Lehet játszadozni a jogosultság megadásával, de előbb-utóbb az ember belefárad. Miért? Hát azért, mert a fent említett jogot minden egyes munkaállomáson helyileg kell megadni a felhasználóknak, azaz ha 500 gép van, akkor mind az 500-on! *(Lokális SAM jog.)*
Ennek a problémának az elkerülésére valók a – szolgáltatásként futtatott – komponensek, amit a gépekre telepítve a bejelentkezett felhasználó jogosultságától függetlenül ellátja az időszinkronizációs feladatokat. Így ha 500 gépünk van, akkor „csak" 500 telepítést kell elvégezni, és készen vagyunk. Sokkal egyszerűbb, mint 500 jogosultságállítás nemde?

RFC1305; RFC1769 *(NTP; SNTP)*

A legnépszerűbb időszinkronizáláshoz használt protokoll az RFC1305-ben leírt Network Time Protocol *(NTP)* ami relatíve elég összetett, komplex alkalmazás. Az NTP egy speciális eszköz segítségével *(például rádióadó, vagy műholdvevő)* egy másik eszköztől „elkéri" a pontos időt. Az NTP használata gyakorlatilag mikroszekundumos pontosság elérését teszi lehetővé, természetesen az eredmény függ a pontos időt szolgáltató eszköztől és az ahhoz vezető elérési úttól is. Az NTP használata azonban nem szükséges olyan esetekben, ahol nem követelmény a mikroszekundumos pontosság.
Az SNTP *(Simple Network Time Protocol – RFC1769)* egy olyan időszinkronizációs szolgáltatás, ami az NTP-vel szemben nem biztosít ekkora pontosságot. Azonban nem kell elkeseredünk: a pontosság itt is fontos és létezik, még ha nem is mikro, hanem milli az a szekundum. Az SNTP protkollt kimondottan ügyfél-kiszolgáló hálózatok számára készítették és tökéletesítették. *(Az SNTP és az NTP teljesen azonos adatcsomagokat használ, azonban az SNTP nem támogatja a hibaellenőrzést.)*

NT Time Service és W32Time Service

A Windows NT 4.0-hoz is be lehetett szerezni időszinkronizációs szoftvert, többek között a Resource Kiten találhattunk egy ilyet. A Windows 2000-be pedig, mint említettük, gyárilag „beszerelték" ezt az extrát. A két szolgáltatás természetesen ugyanazt, azaz időszinkronizációt tesz lehetővé Microsoft környezetben. Azonban a két eszköz néhány dologban gyökeresen eltér egymástól. A különbségeket meg kell ismernünk, továbbá ismernünk szükséges a korábbi időszinkronizációs eszközöket is, mert többnyire vegyes környezetekkel is találkozhatunk munkánk során.
A tökéletes pontosság eléréséhez, vagy legalább a rendszer konzisztenciájának megőrzése érdekében ki kell jelölni a szükséges feladatokat kiszolgáló gépeket. Nem lenne szerencsés, ha a hálózatunk minden számítógépe azonos tulajdonságot hordozva képes lenne időkiszolgálóként működni, és egy esetleges kérést kiszolgálna. Ennek elkerülése érdekében az időszinkronizációs folyamat csak egy előre, jól definiált szabályrendszerben történhet meg. A következő ábra szemlélteti a szinkronizációban részt vehető partnerek kapcsolati viszonyát.



NT TimeServ hierarchia

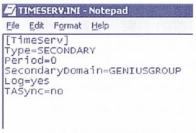
Elemezzük az ábrát! Minden NT TimeServ szolgáltatást futtató tartományban, vagy hálózatban szükséges legalább egy MASTER Time Server, egy PRIMARY Time Server és ezen kívül lehetnek még SECONDARY Time Serverek és egyszerű Time ügyfelek. A szabály szerint a MASTER Time Server egy, az Interneten keresztül elérhető Standard Time Provider-től veszi a pontos időt NTP protokollt használva. A PRIMARY Time Server a MASTER Time Serverrel áll kapcsolatban és tőle kéri el a pontos időt TCP kommunikáció során. A Secondary kiszolgáló egy olyan kiszolgáló ami a Primary Time Server ügyfele lehet *(szintén TCP kliens)*. Time ügyfeleknek hívom azokat a munkaállomásokat, amelyek a net time parancsot használják a pontos idő elkéréséhez. Egy Windows NT 4.0 Server is lehet Time ügyfél, ha nincs a gépre telepítve a TimeServ szolgáltatása, vagy az újabb W32Time szolgáltatás. A kliensekre *(WinNT Wks.;Win2000)* telepített TimeServ esetén Secondary-nek kell őket konfigurálni. Tehát mint láthatjuk, a szerepeknek igen fontos súlyuk van. De hol lehet beállítan, hogy melyik gép milyen szerepet tölt be? Az az időkezelő komponenstől függ. Sajnos ilyenből több is van:

1. A TimeServ szolgáltatás

A korábbi időszolgáltatóban nem túl barátságos módja van a szerepek konfigurálásának. A timeserv.ini fájlt kell kézzel módosítani. A TimeServ szolgáltatás három fájlból áll:

- TimeServ.exe: ez a futó alkalmazás, konfigurálást nem igényel. Helye a fájlrendszerben: %systemroot%\system32
- TimeServ.dll: az exe-hez tartozó dynamic link library. Konfigurációt nem igényel, azonban megléte fontos. Helye a fájlrendszerben: %systemroot%\system32
- TimeServ.ini: a konfigurációs fájl. A hierarchiában elfoglalt szerepet ebben a fájlban határozhatjuk meg. Helye a fájlrendszerben: %systemroot%

A TimeServ.ini fájl testreszabásával a hálózatunkban használt hierarchiát alakíthatjuk ki. A képen egy példát láthatunk egy TimeServ.ini-re:



TimeServ.ini

A Type paraméter megadásával beállíthatjuk, hogy milyen szerepet tölt be a gépünk. A lehetséges értékek: Master, Primary, Secondary. A Type=Primary esetében egy PrimarySource=\\kiszolgáló sort kell beírni a SecondaryDomain sor helyére. A Type=Master esetében pedig az NTP Server=ntp kiszolgáló sort kell az ini fájlhoz hozzáadni. Az ini testreszabása után telepíthetjük a TimeServ programot. Amennyiben a TimeServ.ini-t a szolgáltatás telepítése után módosítjuk, a szolgáltatást újra kell indítani ahhoz, hogy az érvényre jusson *(az újraindítás közben a szolgáltatás leállását követően a timeserv –update parancsot kell futtatni)*. Közvetlenül registry-ből is konfigurálhatjuk a TimeServ szolgáltatást. A

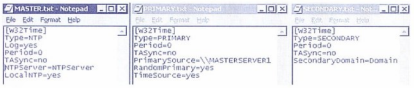
```
HKLM\System\CurrentControlSet\Services\
LanManServer\Parameters
```

kulcs alá létre kell hozni a TimeSource REG_DWORD típusú értéket. A Registry módosítása után a gépet sajnos újra kell indítani.

2. A W32Time szolgáltatás

A Microsoft az Y2K javítások keretében egy új, a Windows NT 4.0-ra is telepíthető időkiszolgálót is kifejlesztett, ez a W32Time Server. A program néhány újítást tartalmaz a TimeServ 1.55-höz képest. A „type" már három lehetséges értéket vehet fel: NTP, Primary és Secondary. Az NTP a MASTER TimeServer-t jelöli a hálózatban, amiből több is lehet. A hálózatunk legfeljebb egy Primary kiszolgálót tartalmazhat, ami az összes MASTER TimeServerrel szinkronizálhat. A hálózat többi számítógépe pedig Secondary lehet. További előnye az, hogy hálózatunkban saját NTP kiszolgálót definiálhatunk a LOCALNTP=yes sorral.

Nézzünk egy példát a funkciók definiálására. Az alábbi képen egy MASTER egy PRIMARY és egy SECONDARY w32time.ini fájlt láthatunk. Természetesen a MASTER esetén az NTPServer=NTPSERVER esetén az érték helyére egy NTPSERVER kiszolgáló címét kell megadni, ugyanez igaz a PrimarySource és a Secondarydomain-re is.



W32Time.ini

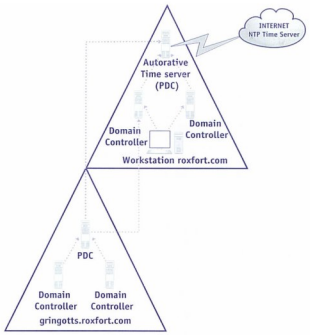
3. A Windows 2000 Time Server

A Windows 2000-hez egy új fejlesztés vette kezdetét de az új időmunkás neve megegyezik a Windows NT 4.0 alá készült Windows Time Serverrel: W32Time. A W32Time előtelepített szolgáltatásként az operációs rendszer részévé vált *(nem kell utólag 500 gépen telepíteni!)*. A szolgáltatás a gép indításakor elindul, a szinkronizálás automatikusan megtörténik. A W32Time kétféle módon képes szinkronizálni, ezek neve: NTP és NT5DS. Az NTP szinkronizáció során meg kell adni az NTP kiszolgáló nevét vagy IP címét. *(A Windows 2000 is képes NTP kiszolgálóként működni. LOCALNTP=1)*

A W32Time SNTP-t használ az időszinkronizációhoz.

NT5DS szinkronizáció során az Active Directory hierarchia szerint történik a szinkronizálás. A szinkronizálási hierarchia a következők szerint alakul:

- A gyökértartomány PDC Emulator FSMO tulajdonosa az AD erdő Authorative Time Servere. *(Windows 2000 alatt a MASTER Time Servert Authorative Time Servernek nevezzük.)* Az Authorative Time Server-t egy külső NTP kiszolgálóval szinkronizálni kell.
- A tartományban az összes tartományvezérlő a tartomány PDC Emulator gépével szinkronizálja az óráját.
- A gyermektartomány PDC Emulatora a gyökértartomány bármely tartományvezérlőjével szinkronizálhat.
- A gyermektartomány(ok) összes tartományvezérlője szinkronizálhat a szülőtartomány bármely tartományvezérlőjével vagy a gyermektartomány PDC Emulatorával.
- Bármely tartomány bármely ügyfele a saját tartomány összes tartományvezérlőjével szinkronizálhat.



AD hierarchia

Mint ahogy mondtuk, az Authorative Time Servert egy *(vagy több)* külső NTP kiszolgálóval szinkronizálni kell, mert ha nem tesszük, tele*(s)*írja az Event Logot. Ezt a Windows 2000-ben debütáló új net time parancs segítségével tehetjük meg a következő formában:

```
net time /setsntp:kiszolgáló címe(i)
```

A W32Time összes konfigurációs paramétere a Registry-ben tárolódik. A fenti parancs kiadása után az adatok a

```
HKLM\System\CurrentControlSet\Services\w32Time\
Parameters
```

kulcs alá kerülnek, így a kiszolgálók listáját nem kell újra és újra megadnunk: az már a registryben marad felülírásig. Meg kell említenem, hogy a TimeServ.ini és a W32Time konfigurációs lehetősége elég összetett. E cikk keretében nem az összes kapcsoló és funkció részletes bemutatását, hanem inkább a terminológia megértését tartottam szem előtt, ezáltal pedig a speciális funkciók használatához szükséges kapcsolók a HELP fájlok és termékdokumentációk közül könnyedén kikereshetők.

Ajánlatos legalább két különböző TimeServer-t megadni az Authorative Time Server számára, hibatűrő rendszer elérésének érdekében. Válogathatunk az Interneten található számtalan időkiszolgáló közül, vagy rácsatlakozhatunk a Microsoft által üzemeltetettre: time.windows.com.

A tűzfalakon természetesen a megfelelő portot ki kell nyitnunk ahhoz, hogy kiszolgálónk szinkronizálni tudjon. Az NTP és az SNTP az UDP 123-as portot használják. Ezt mindenképpen át kell engednünk, ellenkező esetben a szinkronizálás nem fog működni.

Mint láthatjuk, a tartományvezérlők *(DC)* szinkronizációs partnereként több kiszolgáló is szóba jöhet. Mi alapján dönti el, hogy kivel fog szinkronizálni? Elég komplikált módon választja ki. A következő algoritmus alapján dönti el, hogy kivel szinkronizál: amikor „ütött az óra", küld egy kérdést minden tartományvezérlőnek. Ezek válaszolnak neki a kérdésre, elküldik a kért infomrációt. A kérdés, és válaszok alapján eldönti, hogy a válaszoló géppel azonos telephelyen *(Active Directory Site)*-ban és tartományban van-e, vagy sem. Majd felállít egy sorrendet a tartományvezérlők között, ahol előnyt élvez a szülőtartomány tartományvezérlője a helyi tartomány tartományvezérlőjével szemben. Plusz pontot ér, ha azonos telephelyen vannak. E szerint a helyi tartományban, azonos Site-on levő tartományvezérlőt fogja választani a szülőtartományban, de más telephelyen lévő géppel szemben.

Munkaállomások és tagkiszolgálók

A munkaállomás a hitelesítést végző kiszolgálóval kicserél néhány *(száz)* adatcsomagot a bejelentkezés során, amiből megállapítják a kettejük közt zajló kommunikáció időkésleltetését. A munkaállomás ezután lekéri a vekker állását. Ha a kapott pontos időhöz képest a rendszer órája késik, a munkaállomás az órát azonnal beállítja, utánhúzza. Ha a kapott időhöz képest a munkaállomás órája maximum két percet siet, a munkaállomás a következő 20 percben a rendszer óráját lassítja. A lassítást úgy éri el, hogy az eltéréstől függően a következő értéket veszi az időmúlás alapjául: 1 min=61-66 mp. Amennyiben az eltérés több, mint 2 perc, a munkaállomás azonnal beállítja a rendszerórát. *(Ennek oka a következő: 2 min=120 mp, amennyiben 20 percen keresztül 66 mp=1 min akkor pontosan 20*6 mp=120 mp=2 min amit tudott a rendszerórán faragni. A Microsoft maximum 10%-kal engedi a rendszeróra lassítását, ami a 66 mp=1 min.)* A rendszeróra most már pontos, már csak a rendszeres egyeztetésről kell gondoskodni. Az első egyeztetés után a munkaállomás a rendszeróráját időközönként ellenőrzi. Alapértelmezésben ez 8 óránként történik meg. Amennyiben 8 óra után az eltérés mértéke nagyobb mint 2 másodperc, az ellenőrzések számát megkétszerezi, azaz a lekérési időt a felére csökkenti *(4 óra)*. Ha az eltérés 4 óra múlva is meghaladja a 2 másodpercet, az idő felezését addig folytatja amíg a két ellenőrzés közti időszelet el nem éri a 45 percet. Amennyiben kevesebb mint 2 másodperc az eltérés, az órát azonnal beállítja. A 45 perces küszöböt elérve pedig addig tartja ezt a gyakoriságot, amíg az eltérés nem lesz kevesebb, mint 2 másodperc a szinkronizálási idők között.

Down-Level ügyfelek időszinkronizációja

A Windows NT Workstation gépek mind a TimeServ, mind pedig a W32Time szolgáltatás futtatására képesek *(a type=Secondary értéket megadva)*. Vegyes környezetben is a type=Secondary értéket kell használni.

Meg kell jegyezni, hogy Windows9x-nél a net time parancs hibás: ha a kiszolgáló és a kliens gép különböző időzónában van, nem működik megfelelően. Például ha a kiszolgáló időzónája GMT, a munkaállomásé pedig GMT+1, és a szinkronizáláskor GMT idő szerint 10:00 a pontos idő, a kliens GMT+1 időzónában is 10:00-ra állítja be az időt – azaz nem veszi figyelembe az időzónát. A helyes idő GMT+1 szerint ekkor már 11:00. A Microsoft kijavította a nyilvánvaló hibát, és a Microsoft Windows 98 Resource Kit lemezén közzéadta. A nettime.exe és az rtzone.exe programot egyszerűen be kell másolni a %SystemRoot%\System32 könyvtárba *(plusz újraindítás)*. Az új nettime.exe program használata során a /set /y kapcsolókat nem szükséges megadni.

Ha már kliens és hiba, akkor meg kell jegyezni egy másikat is. A Windows 95 első, minden Service Pack nélküli változata – amiből reményeim szerint már nem fut olyan túlságosan sok - a téli időszámítás időpontját elvéti. Szeptember utolsó vasárnapját követően ő bizony átáll. Ezt pontosan egy hónappal korábban teszi meg mint kellene, s ekkor egy érdekes eset fordul elő. A hálózatban levő Windows NT és Windows 2000 gépek még nyári időszámításban számolnak, a Windows 95 pedig már téliben. Ilyenkor sajnos - függetlenül az időzónától - eltérés lesz a munkaállomások és a kiszolgálók között is. *(A téli GMT+1 10:00 bizony GMT+1 9:00-nak felel meg nyári időszámításban.)* Ez már elég kaotikusnak tűnik, s ha még hozzávesszük azt is, hogy a Windows9x-ek az időzónától függően rosszul szinkronizálják a helyi órájukat, nem biztos, hogy meg tudjuk válaszolni a klasszikus kérdést: mennyi az idő most.....? A hiba javítása egyszerű: az rtzone.exe programmal le kell gyártani az egyedi időzónát amiben a helyes dátumra javítjuk a téli időszámítás átállását, s az új időzónát terjesztjük a munkaállomásokon.

Ügyfél-kiszolgáló alkalmazások és a pontos idő

A csoportmunka talán a legkényesebb a pontos idő használatára. Csoportmunka megoldásokat legkönnyebben Exchange kiszolgálók segítségével vehetünk igénybe. Exchange kiszolgáló esetében a levelek küldési- és fogadási ideje abszolút időben mérődik. Ez azt jelenti, hogy ugyan ott van az időpont, de ez nem tartalmazza az időzónát. Ezért ha az időzónát átállítjuk, a levél küldési- és fogadási ideje is az időzónához igazodik, látszólag eltolódik. Ez érinti a feladatok kiosztását is, mert ha kiosztunk egy feladatot GMT+1 10:00-ra az GMT+2 időzónában megnézve 11:00 lesz, természetesen GMT+2 szerint. Egy kis matatás az időzóna beállítások között, s máris átkerül a megbeszélés tíz óráról nyolc órára... Ez az eltérés a téli/nyári átállás esetén is jelentkezik, tehát fontos, hogy a munkaállomások azonos időpontban váltsanak a téli és nyári időszámítás között.

Félreértés ne essék: az Exchange teljesen logikusan és egységesen kezeli az időket, a felhasználók trehánysága miatt előálló virtuális késések és sietések lekezelése nem az Exchange Server feladata. Ha butaságot kérnek tőle, butaságot mond.

Az Outlook lehetőséget biztosít, hogy két időzónát használjunk, de egyidőben akkor is legfeljebb egyet, ráadásul a módosítás nemcsak az Outlook által használt Időzónát állítja át, hanem a rendszerét is.

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

RRAS *(III. rész)* Útválasztási protokollok

Ez a cikk nem azonnal az útválasztási protokollokkal kezdődik, mert meg kell emlékeznünk a szeptemberi számból terjedelmi okokból kimaradt útválasztási hibákkal, melyek szinte minden hálózaton felbukkannak előbb-utóbb. Csak ezek után térünk rá a RIP útválasztási protokollra.

Útválasztási anomáliák

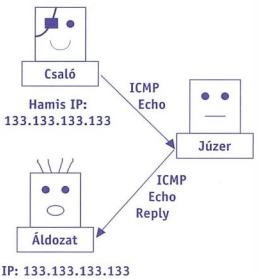
Három esetet veszünk szemügyre:
1. Forgalmazás belső IP címekről
2. Hamis IP címek használata
3. Körkörös routolás

1. Forgalmazás belső IP címekről

Ez az útválasztási „hiba" többnyire nem szokott galibát okozni, mert akik tudatosan használják a belső IP címeket hálózataikban, nem lepődnek meg azon, hogy ezeket a nagyvilágban, kint az Internet backbone-on nem fogadják el a routerek. Mely címekről volna szó? A 10.0.0.0/8, a 172.16.0.0/16, és a 192.168.0.0/16 tartományok címeiről, melyek a 1597-es RFC *(tech.net 2000 október)* szerint arra vannak fenntartva, hogy saját hálózatunkon szabadon használjuk őket. Ennek a szabadságnak egyenes következménye, hogy a világban számtalan számítógép fut például a 172.16.0.1 címmel. IP cím ütközés azért nem lép fel közöttük, mert sohasem kerülnek közvetlenül kapcsolatba egymással - kizárólag címfordítás útján érintkezhetnek. A címfordítást például az első részben elemzett NAT biztosíthatja. Tipikus hibakeresési zavar, amikor valaki szóvá teszi, hogy belső IP című gépéről nem tud kipingelni, bár minden más zavartalanul működik. Mi a hiba oka? Semmi. Nincs hiba, a NAT-ok, Proxyk a legritkább esetben emelik át az ICMP *(Ping)* csomagokat. Ne pingeljünk külső címekre.

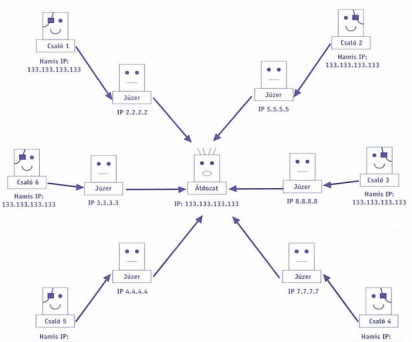
2. Hamis IP címek használata

Ritkán fordul elő véletlenül, de praxisomban jött már „szembe" velem olyan hálózati csomag, mely hamis IP címről érkezett. Az IP útválasztás meglepően hasonlít a hagyományos postai kézbesítésre: hamis feladóval éppúgy célbajuttathatunk IP csomagokat, mint postai levelezőlapokat. A postás bácsi csak a címzett címére összpontosít. S mint a való életben, a hamis címzés az IP csomag esetében is csak válaszoláskor okoz galibát: hova vigye a postás a személyesen a télapótól kapott levélre írt válaszlevelemet? Hamisított, de létező feladó *(President George W. Bush)* esetén pedig a válasz a gyanútlan harmadik félnél köt ki. Kéretlen *(válasz)* csomagok érkeznek egy gépre? Valószínűleg valahol valaki visszaél a mi IP címünkkel!



Hamis IP címről érkezett csomagra választ a cím gyanútlan tulajdonosa kap!

Volna az IP-ben lehetőség arra is, hogy a hamis IP címről érkező csomag mégis a hamisítóhoz jusson vissza. A megoldás *(source routing)* lehetővé teszi, hogy egy csomag az eredeti feladó által meghatározott úton jöjjön vissza, hasonlóan az elektronikus levelezés "reply-to" megoldásához. Ezzel a technológiával azonban gyakorlatilag sohasem találkozunk, mert minden jólnevelt útválasztó eldobálja a source routinggal „felszerelt" csomagokat. Hogy miért? Mert az ezzel kapcsolatos hekkelések már évek óta ismertek. A hamis IP cím azonban így is visszaélésekre adhat alkalmat, egyfajta DDoS támadás kivitelezhető vele: ha egyszerre ezer gép áll neki egyetlen másik nevében pingelni össze-vissza a neten, akkor az áldozat *(akinek az IP címét felhasználták)* a világ minden sarkából elkezd kéretlen ICMP Reply-ket kapni.



DDoS támadás az áldozat IP címének használatával. (Ez megmagyarázza, miért kap Bush elnök évente több tízezer levelet... :-)

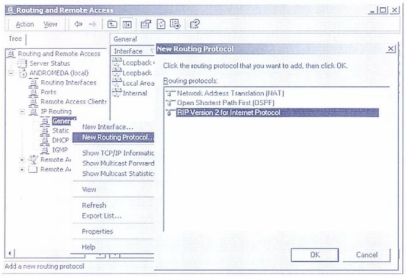
Így járt tavaly kora tavasszal a Yahoo!, az e-Bay és a többiek. A nyomozás rendkívül nehéz, ugyanis minden csomag úgy néz ki, mintha valóban az áldozat adta volna fel a kérést. Most nem tippeket óhajtottam adni, hanem rávilágítani, hogy mi mindenre „jó" egy kis szemfüles ravaszság, hogyan válik fegyverré minden. Régebben erre azt mondták volna, hogy elsült a kapanyél. Ma csak annyit mondunk: World Trade Center.

3. Körkörös routolás

A NetMon sorozat egyik részében már kitértem erre a jelenségre, amely akkor fordul elő, ha egy hálózatban az alapértelmezett útvonalak hurkot zárnak be, és egy hálózati csomag olyan címre irányul, amely ismeretlen mindegyik útválasztó előtt. Ebben az esetben ugyanis mindegyik a 0.0.0.0 útra küldi az ismeretlen csomagot, mely - digitális jel lévén - a végtelenségig keringhetne a hálózatban, ha valaki, vagy valami meg nem fogná. Emlékeznek még? A TTL *(Time to Live)* számláló csökkentése révén az IP csomagok csak egy maximális számú útválasztóátlépésre képesek. Mivel mindegyik router csökkenti a TTL-t, előbb-utóbb ennek értéke nullára fut, s a csomag elhal.

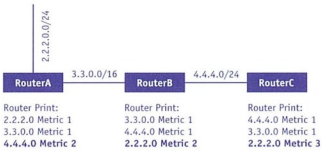
Útválasztási protokollok – a Routing Information Protocol

Az előző részben eljutottunk odáig, hogy kézzel képesek voltunk módosítani az útválasztók útvonaltábláit. Szép dolog, hibakereséshez kell is ez a tudás, de egy összetett vállalati hálózat útvonalterveinek kiszámításához és megvalósításához nem árt, ha számítógépeinket is segítségül hívjuk. A RIP protokoll telepítsd-és-felejtsd-el módon működik. Csak felhajigáljuk a megfelelő gépekre, és a tudás *(alhálózatok és átjárók ismerete)* szétterjed az egész hálózatban. Telepítés után még meg kell mondani neki, hogy mely hálókártyákon dolgozzon, és már megy is. A beállítások erdejét lásd később.



A RIP telepítése

Gyönyörűen működik, nem hiába a Xerox találta fel ezt is. Ugyanakkor routerguruk szájából mást hallani: a RIP így nem jó, a RIP úgy nem jó, és nagy hálózatokon teljesen használhatatlan. Most akkor jó, vagy nem jó? Vizsgáljuk meg a működését! Minden router ismeri a belőle kimenő vonalakat, valamint azt az IP címet, amely az adott alhálózaton a gateway *(ezt elemezgettük a route print paranccsal)*. A RIP az útvonalak hosszát az úgynevezett hop counttal méri, mely a route print kimenetében a Metric mező. Ha beteszünk egy hálózati kártyát egy gépbe, azon automatikusan 1-re állítódik a Metric, azaz minden olyan masina, ami azon a kártyán közvetlenül elérhető, 1 hopra van tőlünk. A Metric nem a bedugaszolt kártya tulajdonsága *(bármennyire is annak tűnik)*, hanem a rajta keresztül elérhető hálóé - így áttétlenesen az azon található közvetlen szomszédainké is. Most vegyünk fel három útválasztót az alábbi ábrán látható módon. Mi történne, ha ezek elmesélnék egymásnak, mit látnak a világból?

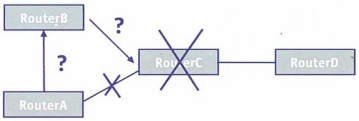


Három pletykás vénasszony. A vastaggal jelölt sorokat a szomszéd súgta.

RouterA elmondaná RouterB-nek, hogy lát egy 2.2.2.0/24 és egy 3.3.0.0/16 hálózatot. RouterB meghallgatja a történetet, a 3.3.0.0/16-ot nem veszi figyelembe, mert azt ő is tudja, de felveszi a 2.2.2.0/24 hálózatot az útvonaltáblájába, úgy, hogy a Metricet megnöveli eggyel. Nála a 2.2.2.0/24 útvonal 2-es Metriccel szerepel majd. Ezután pletykás vénasszony módjára továbbmondja RouterC-nek, hogy van ám egy 2.2.2.0/24 hálózat tőle 2 hopra, aki immár 3-as Metriccel jegyzi be, hisz a 2.2.2.0/24 hálózat tőle már 3 hopra van. A RIP-es gépek fix időközönként elküldik egymásnak teljes útvonaltáblájukat, benne saját, és „tanult" *(az ábrán vastagított)* bejegyzésekkel. Az eredeti RIP alapértelmezésben 30 másodpercenként jelentgeti társainak a saját világképét, mégpedig Broadcast címzéssel, hisz nem tudhatja, hogy egy adott subneten hány érdeklődő akad a bejelentésre. Ez az ő egyik legnagyobb baja. Hány géphez jut el a RIP üzenet? Mindhez. És switch esetén? Mindhez. A hálózati nyomtatóhoz és a kenyérpirítóhoz is? Igen. Ha egy útvonal megváltozik, akkor a 30 másodperces ütemezésen belül is indulnak úgynevezett triggerelt üzenetek a partnerek felé.

Konvergencia

A RIP maximum 15 hop átmérőjű hálózatokhoz való azon egyszerű oknál fogva, hogy a szabvány szerint ami 16 hopra, vagy annál messzebb van, az „unreachable". *(Egyes felmérések szerint a földgolyó 17 hoppal bármilyen irányban körbejárható. Ennek ellenére a RIP a maga 15 hopos korlátjával csak picike hálózatokra való.)* A 15-ös korlátozást azért kellett bevezetni, mert bizonyos hálózati hibákra a RIP igen furcsán reagál: ahelyett hogy egy router kiesését hipp-hopp észrevenné, a lenti ábrához hasonló hálózatban RouterC kiesésével RouterA és RouterB azt fogja hinni, hogy a közvetlen vonal ugyan megszakadt RouterD-hez, de egymás segítségével *(RouterA RouterB-n keresztül, RouterB pedig RouterA-n át)* mégis eljutnak oda.



A világ RouterA szemüvegén keresztül. Egy közbülső útválasztó halála esetén a RIP alternatív útvonalakat vél felfedezni ott is, ahol nincsenek!

Ennek a viselkedésnek az az oka, hogy a RIP mindig a legolcsóbb útvonalat veszi figyelembe, és amíg RouterC élt, addig RouterA a RouterB által ajánlott kerülőutat nem veszi figyelembe. Amint azonban RouterC meghal, mindjárt a 2 hopos, RouterB-n keresztülvezető kerülőút lesz a legolcsóbb, s RouterA figyelembe is veszi – sajnos nem a szakadás győz. És ami a legfurcsább, RouterA és RouterB innentől tanítgatják egymást a rég halott RouterC nemlétező útvonalával. S ahogy egymás fülébe súgják *(„te, én azt hallottam, hogy van út RouterD-hez")*, mindig hozzáadnak egyet-egyet a hopcounthoz, míg az el nem éri a 16-ot, és akkor VÉGRE kiesik a nemlétező út a pici agyukból. Ezt a jelenséget Counting to Infinitynek hívják, és igen bosszantó, mivel a hálózati hibák észlelése emiatt meglehetősen lassú. A konvergenciaproblémára van megoldás, lásd később. Mérgezett alma...

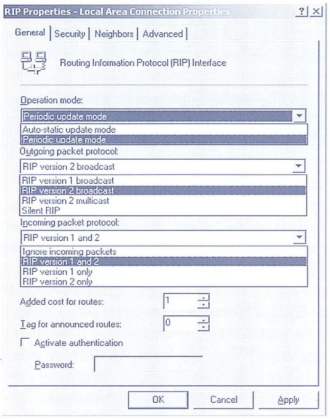
A RIP eredeti verzióját azonban nem a broadcastolás és a lassú konvergencia miatt felejthetjük el, hanem a boldog békeidők miatt. Amikor ez a szabvány készült *(***[1]***, 1988)*, még nem voltak fogyóban az IP címek, és senki sem gondolt arra, hogy egyszer majd az A, B és C osztályú IP címtartományokat a subnet mask bitjeinek beállítgatásával ripityára fogjuk szabdalni *(lásd CIDR, azaz Classless InterDomain Routing,* **[3]***)*. Íme a RIP csomag felépítése:

command	version	must be zero
Address family identifier		must be zero
IP address		
must be zero		
must be zero		
metric		
(az utóbbi négy sor ismétlődik max. 25-ször)		

A RIP eredeti verziójában kérem szépen nem szerepel a routerek közötti forgalomban az egyes hálózatok subnet maskja! De hisz ez így használhatatlan!

RIP v2

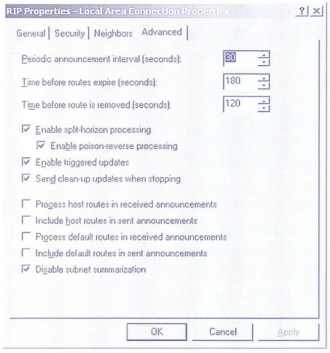
1994-ben látott napvilágot a RIP második verziója **[2]**, mely már figyelembe veszi a CIDR-helyzetet, nemcsak Broadcastolni tud, hanem Multicasttal is képes értesíteni a többi routert, továbbá némi szekuriti is van benne - no nem túl sok, egy kevés klírtext jelszó„védelem". Az a döbbenetes a RIPv2 csomagformátumában, hogy annak dacára, hogy csomó mást is tartalmaz, mint amiről a RIPv1 valaha is álmodott, ***_kompatibilis_*** az előddel, azaz RIPv2 csomagokból RIPv1 útválasztók tanulhatnak! Az IP cím utáni „must be zero" mező viszi a subnet maskot. Ez a gyönyörű az Internetben: a zökkenőmentes haladás, és az elődök szellemének tiszteletben tartása. Az új RIP, mint említettük, Multicastolni is tud, mégpedig a 224.0.0.9 címen. Az alábbi ábrán – egy kis csalással – megjelenítettem az RRAS-féle RIPv2 Interface tulajdonságlapjának első oldalán az összes beállítást. A csalás lényege, hogy egyszerre sikerült legördítenem az összes kombót. Ezt csinálja utánam valaki!



A RIPv2 beállításai

- **Operation mode:** ez a görgettyű állítja be, hogy RIP routerünk részt vegyen-e a 30 másodpercenkénti periodikus üzenetküldésben. Auto-static update módban csak és kizárólag RIP Requestekre válaszol, Periodic Update módban önként elárulja minden titkát
- **Outgoing packet control:** itt lehet beállítani, hogy - ha egyáltalán megszólal - hogyan tegye. RIPv1 nyelven csak Broadcastolni tud, és ne feledjük, búcsút mondhatunk a subnet mask átvitelnek. RIPv2-ül már két módon *(Broad és Multicast)* beszélhet, míg a Silent RIP azt jelenti, hogy csak hallgatózik és tanul, de senkinek nem mond semmit.
- **Incoming packet control:** mely bejövő csomagtípusokra reagáljon.
- **Added cost for routes:** ennyit ad hozzá a beérkező útvonalak hop countjához *(Metric)* mielőtt beveszi az útvonaltáblába. Emiatt ez az implementáció minimum 3-as Metriccel veszi fel a tanult sorokat.
- **Tag for announced routes:** itt két bájtunk van, ahova bármit beírhatunk. Eredetileg a RIP üzenet forrásának tipizálására való *(külső-belső)* de jobbára semmire sem használjuk.
- **Activate authentication:** egyfajta hamis biztonságérzetet adhat ez a jelszómező. Minden RIPv2 útválasztón egyformán kell kitölteni – ha kitöltjük egyáltalán. Arra való, hogy csak a jelszó birtokosaival replikáljon routerünk. Csakhogy a jelszó klírtext, vagy más néven ASCII titkosítással *(értsd: olvashatóan)* kerül ki a hálóra. Nem sok értelme van. Ennél sokkal többet ér a Security és a Neighbors fül, ahol explicite be lehet állítani, hogy kiknek küld, kiktől fogad, mely útvonalakat tesszük esetleg tiltólistára, hogy még véletlenül se íródjanak felül.

És végül az Advanced fül:



A RIP matematikája

- **Periodic announcement interval**: itt módosíthatjuk esetleg az alapértelmezett 30 másodpercenkénti értesítéseket *(nem ajánlott)*.
- **Time before routes expire**: ez a szám az útvonalbejegyzések TTL-je. Ha ennyi másodpercig nem frissíti a partner, érvénytelennek kell tekinteni. Azonban ekkor még nem törlődik, hanem mint „érvénytelen" kerül továbbadásra, hogy a többi partner is érvényteleníthesse
- **Time before route is removed**: a végleges törlés késleltetése

A most következő két pipa külön alfejezetet érdemel. Emlékszünk még az egymást badarsággal traktáló RouterA és RouterB esetére? Az útvonaltáblák trükkös továbbításával elkerülhető lenne ez a helyzet. Erről szól a horizontvonal, és a mérgezett alma.

Split horizon és poisoned update

Nagyon egyszerű elmagyarázni a horizontvonal-trükköt: az ezzel a beállítással futó RIP routerek nyilvántartják, hogy melyik útvonalat kitől tanulták. Ha csak ezt pipáljuk be, úgy kürtölik szét tudásukat, hogy a tanítónak sohasem feleselnek vissza. Vagyis ha RouterA éppen RouterB-től tanult valamit, akkor azt az útvonalat oda nem mondja vissza. Ezzel elkerülhető egymás fokozatos elléptetése 16 hopig, amikor is végre megáll az őrültség *(elérjük a végtelent. ∞ = 16)*. Ez a trükk kiegészíthető a mérgezett almával *(poisoned update)*. Ekkora tanuló visszamondja a leckét a tanítónak, de úgy, hogy „amit tőled tanultam az nem létezik", vagyis minden megtanult útvonalat 16-os Metriccel, azaz elérhetetlenként mond vissza *(a végtelen ugye elérhetetlen?)*

Mindaddig, amíg az eredeti út létezik, ez a mérgezett visszafeleselés nem változtat semmit a működésen, mert a tanár az eredeti útvonalat tekinti érvényesnek, hisz az nyilván közelebb van, mint amit a nebuló állít. Amint azonban RouterC meghal, és az eredeti útvonal megszakad, a nebuló által közölt végtelen lesz a legközelebbi út, s nem egyenként lépkedve érik el a 16-ot, hanem „azonnal".

Összegzés

Minden problémája ellenére használjuk bátran a RIP protokollt kis és közepes hálózatokon, mert a telepítsd-és-felejtsd-el működés iszonyatos mértékben csökkenti a TCO-t. RIP: amivel nem kell foglalkozni! Csak működik és kész. Menthetetlen hátrányai:

- Nagyobb hálózatokban *(10-12 hop)* az azonnali konvergencia percekig tart
- Vannak olyan – ismert és elkerülendő - hálózati topológiák, ahol nem használható, mert hurkot csinál
- Nagy hálózatok esetén *(alhálók száma > 25)* elég jelentősen nő az egyébként minimális sávszélességigénye, mert komplett útvonaltáblákkal dobálódzik

Miért? Van más?

Van bizony! Vannak olyan útválasztó protokollok, melyek tetszőleges nagyságú hálózatra alkalmazhatók, nem a komplett útvonaltáblát küldik szét, csak a vonalak állapotinformációit *(link state)*, amelyek mindig pillanatok alatt konvergálnak, amelyek sohasem hoznak létre hurkot, amelyek esetén kolbászból van a kerítés. Az egyik ilyen ütőképes protokoll az OSPF *(Open Shortest Path First)*, mellyel a következő RRAS cikkben foglalkozom részletesebben. Addig is egy kis fejtörő: ha az OSPF annyival jobb, mint a RIP, miért nem mindig azt használjuk?

Megfejtés. Mert az OSPF:

- bevezetése tervezést igényel *(a hálózatot mérettől függően Area-kra kell bontani)*
- bonyolult matematikán alapul, amit ha érteni nem is kell, de a létezésének felismeréséig el kell jutni
- nem csökkenti a TCO-t, azaz telepítés után is pátyolgatást igényel*(het)*
- a bonyolult matekozás miatt sok memóriát és processzoridőt igényel*(het)*

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

A cikkben szereplő URL-ek:
[1] A RIP prorokoll. RFC 1058, http://www.ietf.org/rfc/rfc1058txt
[2] A RIP version 2. RFC 1723, http://www.ietf.org/rfc/rfc1723txt
[3] CIDR (Classless InterDomain Routing). RFC 1467, http://www.ietf.org/rfc/rfc1467txt

Remote Installation Services *(II. rész)*

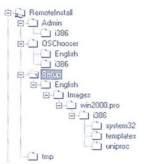
Aki olvasta az előző RIS cikket a novemberi számban, az már tudja mi is a RIS, milyen ügyfél- és kiszolgálóoldali összetevői vannak. Nem fog bambán nézni ha ezeket hallja PXE, rbfg, risetup. A telepítés és alapvető beállítás talán nem jelent problémát számára. Igen sok nyitott kérdés maradt azonban, amiről mindenképp szót kell ejteni. Ebben a részben szó lesz arról, mit tett a telepítés a kiszolgálóval, hogyan tudunk több telepítőkészletet használni, valamint a RIS kiszolgáló karbantartásáról. Ott tartottunk hogy feltelepítettük a RIS szolgáltatást, beállítgattuk a legalapvetőbb dolgokat a risetup varázsló segítségével, majd hitelesítettük a RIS kiszolgálót a tartományban.
Nézzük meg először mit művelt a telepítés a RIS gépen!

RIS kiszolgáló a beállítások után

Risetup futtatása után három új szolgáltatást találunk a Services MMC-ben nézelődve:

- **Boot Information Negotiation Layer** szolgáltatás – röviden BINL. Ez adja meg az ügyfeleknek a telepítéshez szükséges állományok nevét, ő kezeli az ügyféltelepítő varázsló *(Client Installation Wizard - CIW)* képernyőit; tehát vele beszélgetünk, mikor a varázsló képernyőinek válaszolunk a telepítés elindítása előtt. BINL szolgáltatás beszélget a tartományvezérlővel arról, van-e egyáltalán jogunk gépet beléptetni a tartományba, mi a helyzet a csoportos házirendekkel, ő közvetíti a munkaállomásoknak, hogy milyen telepítőkészletek elérhetők a kiszolgálón, és létrehoz egy SIF kiterjesztésű állományt a RemoteInstall\Temp könyvtárban az adott ügyfél telepítéséhez.
- **Trivial FTP Daemon** szolgáltatás - Az ügyfelek ennek segítségével töltik le a szükséges állományokat, a CIW képernyőket és a Windows 2000 Telepítő indulásához szükséges állományokat. Azért használható jól erre, mert neki nem kell megadni felhasználói nevet és jelszót, amikor le akar tölteni egy adott helyről, mint a hagyományos FTP esetén, és rendkívül gyors, mert IP protokoll fölött UDP-t használ *(míg a hagyományos FTP TCP protkollt)* az adatátvitelre *(mint ismeretes, az UDP nem érdeklődik a csomagok után hogy az megérkezett-e a címzetthez, hanem mindent ellenőrzés nélkül villámgyorsan küld át a hálózaton)*.
- **Single Instance Storage Groveler** szolgáltatás – röviden SIS amely uralma alá vonja az egész partíciót amelyre a RemoteInstall könyvtárat létrehozta. Az ő feladata biztosítani, hogy minden egyforma fájl csak egyszer forduljon elő a partíción *(ha minden image tartalmazza ugyanazt a fájlt, akkor is csak egy példányban tárolódjon)*. Igazi csemege ez a szolgáltatás, kár hogy RIS-től külön nem telepíthető.

A Remoteinstall könyvtárat tartalmazó partíción létrejön a következő könyvtárstruktúra:



A RemoteInstall könyvtár szerkezete telepítés után

RemoteInstall – A távoli telepítésekhez tartozó összes minden ez alatt a könyvtár alatt található. Ez a könyvtár Reminst néven meg is van osztva. Ez a hely a TFTP gyökérkönyvtára is, azaz amikor a PXE ügyfél a rendszerindítási folyamat során kapcsolódik rendszerállományának letöltéséhez, ez lesz a főkönyvtára.
Admin – Ez a könyvtár tárolja az összes segédprogramot, ami a RIS kiszolgálóval kapcsolatos. Az Admin/i386 könyvtárban a következő programokat találjuk:

- Riprep.exe – az ő segítségével kreálhatunk egy előkészített gépről pontos másolatot a RIS kiszolgálóra. Az ő segédei az ugyancsak itt található Imirror.dll és a Setupcl.exe
- Rbfg.exe – Távoli rendszerindító floppylemez-készítő *(Remote Boot Floppy Generator)*. Azokhoz a hálózati kártyákhoz készíthetünk vele indítólemezt, amelyeken nincs PXE boot ROM. Erről az előző részben esett szó.

Oschooser – Ez a könyvtár tartalmazza azokat a fájlokat, amelyekre a PXE ügyfélnek szüksége van a rendszer és az Ügyféltelepítő varázsló *(Client Installation Wizard, CIW)* elindításához és futtatásához. Ennek a könyvtárnak legalább két alkönyvtára van: I386 és %language%, ami megegyezik a kiszolgáló nyelvével, például English. Lássuk először az i386 könyvtárat:
I386 – Ez a könyvtár a következő fájlokat tartalmazza:

- Startrom.com – az első fájl, amit az ügyfél letölt, hogy elkezdhesse a rendszerindítást
- Ntldr – Végrehajtható állomány, amely a CIW indítását végzi *(a %windir%\system32\reminst\oschoice.exe kerül ide ntldr néven)*.
- Startrom.F12 – Ugyanazt a funkciót tölti be, mint a Startrom.com, csak ez automatikusan „lenyomja" az F12 funkciógombot is, így a PXE ügyfél mindig betölti a CIW-t.

%language% - Ez a könyvtár tartalmazza azokat a képernyőket, amelyek a CIW futtatása során megjelennek a felhasználónak. Ha testre akarjuk szabni a CIW képernyőit vállalatunk számára, az itt található fájlokat kell módosítani.
Setup – Ez a könyvtár tartalmazza a RIS kiszolgálón tárolt telepítőkészleteket. Amikor az ügyfél végez a varázslóval és elkezdi telepíteni az operációs rendszert, ebből a könyvtárból másolja a fájlokat. A teljes elérési útvonal a

```
\setup\%language%\Images\%directory_name%
```

ahol a %directory_name% annak a könyvtárnak a neve, ahol a telepítőkészlet található, ez esetben a win2000.pro. Itt gyakorlatilag megtalálható a teljes i386 könyvtár a telepítő CD-ről, valamint a templates alkönyvtárban van a SIF kiterjesztésű válaszfájl a telepítéshez. Ebből és a varázslóban megadott adatok alapján fogja a BINL előállítani az adott ügyfél telepítéséhez szükséges egyedi válaszfájlt.

Tmp – Ide kerül a BINL által előbb említett módon összerakott egyedi válaszfájl. Az állomány neve megegyezik az ügyfél GUID-jével SIF kiterjesztéssel. Ez a fájl elvileg addig marad itt, amíg a telepítés szöveges módú része véget nem ér, de láttam én már karón varjút..... Ennek az a célja, hogy ha az ügyfél valamiért újrakezdi a telepítést, akkor az előző telepítési kísérlet során felhasznált információk még rendelkezésre álljanak. Ha nincs folyamatban egyetlen RIS-es telepítés sem, és tudjuk hogy mindegyik sikeresen befejeződött, és itt mégis találunk fájlokat, azokat nyugodtan letörölhetjük, nincs rá szükségünk.

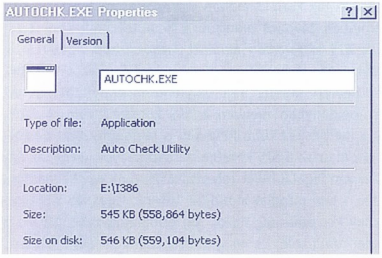
A RemoteInstall könyvtárat tartalmazó partíció gyökerén létrejön egy SIS Common Store nevű rejtett könyvtár ami azokat a fájlokat tartalmazza, amelyek szükségesek a SIS működéséhez ezen a köteten. Nézzük mi is ez a SIS!

Single Instance Storage Groveler

A RIS által lefoglalt partíción a SIS gondoskodik arról hogy minden egyforma állomány ténylegesen csak egyszer foglalja a helyet a partíción. Ez jól hangzik, milyen jó lenne ez egy fájlkiszolgálón ...hhhmm. A SIS használata azonban nem támogatott RIS nélkül :-(, pedig a RIS-nek nem kell a SIS, és a SIS is működik RIS nélkül. A SIS akkor hasznos igazán ha egy RIS kiszolgálón több telepítőkészletet is tárolunk, hiszen ezek nagy része azonos fájlokból áll, gondoljunk csak az i386 könyvtár tartalmára!

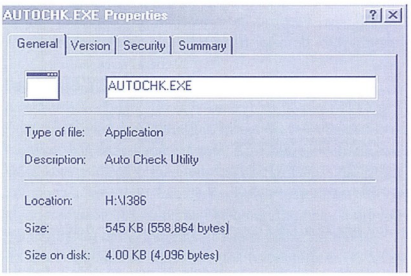
A SIS tulajdonképp két szolgáltatásból áll: a SIS szűrőből és a SIS Grovelerből vagy gyűjtögetőből.

A SIS Groveler szolgáltatás feladata a partíció átvizsgálása, valamint a duplikált állományok kezelése. A Groveler szolgáltatás megjelöli azokat a fájlokat, amelyek azonosnak tűnnek a kötet egy vagy több másik fájljával. Ezután alaposabb vizsgálat következik annak megállapítására, hogy a fájlok tartalma megegyezik-e. Az ellenőrzés után a SIS szűrő átmásolja a fájlt a SIS Common Store könyvtárba, átnevezi egy egyedi GUID-re és ellátja a .sis kiterjesztéssel. Megjegyzem, hogy csak 32KB-nál nagyobb fájlok esetén működik a dolog. Összehasonlításként álljon itt a telepítő CD-ről az autochk.exe:



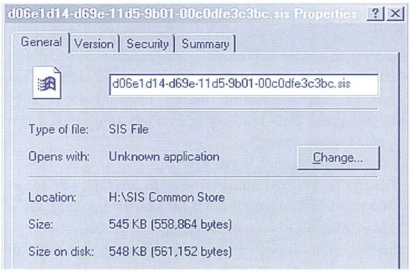
Autochk.exe a telepítő CD-n

Majd amikor a Groveler elvégzi a dolgát ez lesz belőle:



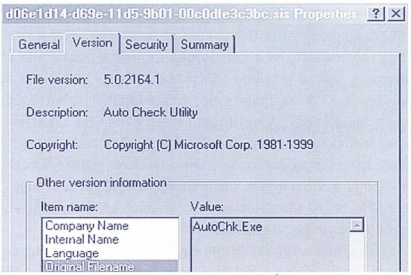
Egy SIS linké alakult autochk.exe

Jól látszik hogy maga a link csupán 4.00 KB helyet foglal el, valójában a fájl több mint 500 KB. Az eredeti példányt a SIS Common Store könyvtár alatt találhatjuk meg, és amit a fenti képen látunk az csupán egy link.



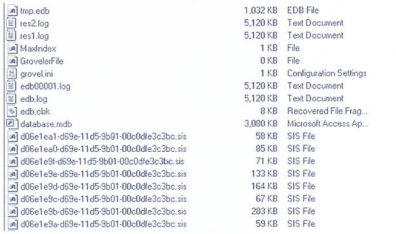
A valódi autochk.exe

A version fül „tutira" megmondja hogy melyik állomány is ez tulajdonképp:



a SIS fájl verziója?

A kötet megegyező fájljai helyén egy hivatkozási pont lesz erre a fájlra. Amikor egy program megpróbálja megnyitni valamelyik fájlt, a fájlrendszer minden fájl I/O műveletet a SIS Common Store könyvtárban levő GUID fájlhoz irányít át.



tmp.edb	1,032 KB	EDB File
res2.log	5,120 KB	Text Document
res1.log	5,120 KB	Text Document
MadIndex	1 KB	File
GroveleFile	0 KB	File
grovel.ini	1 KB	Configuration Settings
edb00001.log	5,120 KB	Text Document
edb.log	5,120 KB	Text Document
edb.chk	8 KB	Recovered File Frag...
database.mdb	3,080 KB	Microsoft Access Ap...
d06e1ea1-d69e-11d5-8b01-00c0dfe3c3bc.sis	58 KB	SIS File
d06e1ea0-d69e-11d5-8b01-00c0dfe3c3bc.sis	85 KB	SIS File
d06e1e9f-d69e-11d5-8b01-00c0dfe3c3bc.sis	71 KB	SIS File
d06e1e9e-d69e-11d5-8b01-00c0dfe3c3bc.sis	133 KB	SIS File
d06e1e9d-d69e-11d5-8b01-00c0dfe3c3bc.sis	164 KB	SIS File
d06e1e9c-d69e-11d5-8b01-00c0dfe3c3bc.sis	67 KB	SIS File
d06e1e9b-d69e-11d5-8b01-00c0dfe3c3bc.sis	283 KB	SIS File
d06e1e9a-d69e-11d5-8b01-00c0dfe3c3bc.sis	59 KB	SIS File

Így néz ki a SIS Common Store könyvtár belülről

A linkek és a fájlok összerendelését egy adatbázis tartalmazza. Ismerős a formátum *(database.mdb)*?
Érdekes, és valószínűleg soha meg nem válaszolódó kérdés, hogy a SIS miért nem az NTFS-ben örök időktől fogva meglévő Hard Linkekere épít. Mint ismeretes, a Hard Linkek lehetővé tennék, hogy egy fájltartalom több könyvtárbejegyzéssel tartsa a kapcsolatot, így egy fájl látszólag több könyvtárban is ott van. A Resource Kit tartalmaz is egy POSIX eszközt – ln.exe – mellyel képesek vagyunk Hard Linkeket faragni. Ebből következően Redmond is képes erre. Valószínűleg a Hard Linkek viszonylagos követhetetlensége miatt van ez így. Hamarosan szót ejtünk a SIS kötet mentéséről és helyreállításáról: bizony nem mindegy, hogy hány példányba kerül szalagra az az adat, amit egyszer már „egypéldányosítottunk".
Aki szeret kísérletezni, tegye ezt:

1. Másolja be a WINNT könyvtárba az ln.exe fájlt a Windows 2000 Server Resource Kit CD-ről, annak is \apps\posix könyvtárából
2. Szemeljen ki, vagy készítsen egy fájlt, melyen a műtétet megejti *(mondjuk WINNT\lanmannt.bmp :-)*
3. Adja ki az **ln lanmannt.bmp linkecske.bmp** parancsot
4. Nyissa meg a linkecske.bmp-t, firkáljon bele, Save
5. Nyissa meg az eredeti lanmannt.bmp-t. Ott a firka!

Gyűjtögető életmód

Amikor a gyűjtögető szolgáltatás hozzákapcsolódik egy kötethez, alacsony prioritással kezd el futni, így szabályozni tudja folyamatait, ha a számítógép terhelése megnő. Ha a köteten a szabad hely egy adott érték alá csökken, akkor a gyűjtögető CPU használata nő, függetlenül a számítógép egyéb terhelésétől. A gyűjtögető intelligens CPU használatának mellékhatása, hogy indulásakor néhány óráig nem fut teljes sebességgel *(akkor sem, ha a rendszer tétlen)*. Ennek oka, hogy a szolgáltatás megpróbálja megállapítani, mekkora CPU és I/O sávszélességet használhat anélkül, hogy az egyéb rendszerösszetevők működését akadályozná.
Ha módosítunk vagy felülírunk egy fájlt, ami így már egyedi lesz, az új adatok a fájl tényleges helyére, és nem a SIS Common Store könyvtárba kerülnek, s a hivatkozási pont is megszűnik. A többi fájl hivatkozási pontja a Common Store könyvtárra megmarad, akkor is, ha csak egy fájl maradt.
Vajon milyen alapon feltételezi a SIS Groveler, hogy egy fájl megegyezik egy másikkal? Hát úgy, hogy összehasonlítja a fájlok aláírásait *(signatures)*. Ezeket az aláírásokat a SIS Groveler készíti el - egy előre megadott függvénnyel - minden egyes fájlhoz, függetlenül attól, hány példányban szerepel a fájl a köteten. Az újonnan hozzáadott fájlok aláírásának elkészítése után megnézi, hogy azok már szerepelnek-e a SIS Common Store könyvtárban lévő adatbázisában. Ha talál olyan fájlokat, melyek aláírása megegyezik, akkor azokat bitről bitre összehasonlítja. Ha így is egyeznek, akkor a fájlokat hivatkozási ponttá alakítja át.
Ha a Groveler szolgáltatást leállítjuk, vagy a RIS-t leszedjük az Add/Removes Progams-on keresztül, akkor a linkelt állományok továbbra is elérhetőek maradnak, új linkek azonban nem jönnek létre, ugyanis a SIS szűrő megmarad és dolgozik a köteten - az első fomázásig.
Volt már arról szó, hogy a SIS Groveler a RIS beállítása során a kiválasztott partíció egészét az uralma alá vonja, nem csupán a RemoteInstall könyvtárat. Az ilyen partíciók mentéséhez és visszaállításához használt programoknak ugyan a linkeket nem feltétlenül kell ismerniük, de a reparse címkéket igen, enélkül fabatkát sem ér a mentés. Az Ntbackup.exe *(amely része a Windows 2000-nek)* jól kezeli ezeket a fájlokat.

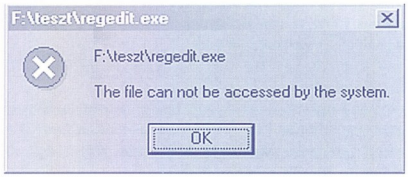
SIS kötet mentés/helyreállítás

Amikor egy SIS hivatkozásról biztonsági másolat készül, a backup progi felismeri az állomány ún. reparse címkéjéből, hogy ez csak egy hivatkozási pont és a SISbkup.dll segítségével meghatározza, hogy melyik fájlokat kell a SIS Common Store könyvtárból mentenie. A SIS hivatkozást úgy menti, ahogy a lemezen szerepel, azaz hivatkozásként és a közös könyvtárban levő állományként. Helyreállítás esetén az NTBackup felismeri a hivatkozási pontot, majd helyreállítja a fájlt, ahogy mentette *(szintén a sysbkup.dll segítségével)*. Ha a mentéshez használt program nem használja a sisbkup.dll-t de ismeri a reparse címkéket, akkor is lehet használni a mentéshez és a visszaállításhoz, csak akkor a médián az adatok teljes méretükkel fognak szerepelni: ha többször megvan ugyanaz a fájl, akkor mindegyik teljes terjedelmében ott lesz a szalagon. Ebben az esetben visszaállításnál nem a linkeket, hanem a teljes fájlokat fogja visszaállítani az adott backup program.
Ha a SIS által birtokba vett kötetet helyben mentjük, akkor a linkek *(úgy, ahogy kell)* szintén a szalagra kerülnek. Ha mi csak mondjuk a RemoteInstall könytárat és annak teljes tartalmát választottuk ki, akkor is ott lesz a szalagon a SIS Common Store könyvtár ha azt ki sem választottunk explicite, és azokat a fájlokat fogja tartalmazni, amelyek a mentésben szereplő SIS linkekhez tartoznak. Az ilyen fájlok visszaállításánál a SIS linkek és a hozzá tartozó sis kiterjesztésű fájlok korrekt módon visszakerülnek a helyükre.
Amennyiben távolról mentjük a RemoteInstall könyvtárat *(a \\< RIS kiszolgáló>\Reminst megosztáson keresztül)*, a médián a fájlok teljes terjedelmükben fognak szerepelni, linkek helyett minden fájl ténylegesen annyiszor lesz ott ahány példányban megvan a SIS köteten. Vegyük észre, hogy a mentéshez több hely kell a szalagon, mint amennyit elfoglalnak a SIS köteten az állományok. Vegyünk egy példát: van legalább két telepítőkészletünk egy közönséges köteten - ami elfoglal mondjuk 1Gb-ot. Ugyanez SIS köteten csak kb. 500-600 Mb-ot fog elfoglalni, mert ott a SIS Groveler dolgozik, és megszüntet minden duplikátumot. Amikor mondjuk távolról mentjük a SIS kötetet, akkor nem a linkek lesznek a médián, hanem a tényleges fájlok. Így ne

csodálkozzunk, hogy ami a vinyón elfoglalt 500Mb-ot, az a szalagon 1Gb-ot követel magának. Az Reminst megosztáson keresztül mentett állományok visszaállításánál nem a linkek, hanem a tényleges fájlok kerülnek vissza a kötetre.

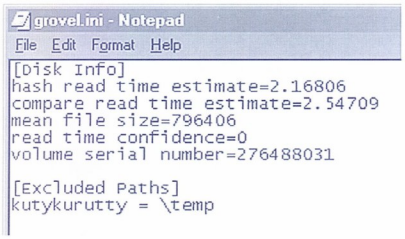
Amikor egy SIS által birtokolt kötetről helyben készített mentést vissza akarjuk állítani, fontos hogy a SIS szolgáltatásnak mindenképpen telepítve kell lennie, és át kell vizsgálnia a kötetet, mielőtt elkezdődik a helyreállítás.

Amennyiben ez nem így történik, a SIS linkek által hivatkozott adatokat nem lehet majd elérni, helyette ezt kapjuk, mikor kettőt kattintunk egy ilyen fájlon:



Egy rosszul visszaállított SIS link

Ha a kötetet ugyanazon a létező RIS kiszolgálón állítjuk helyre, ahol a biztonsági másolatot készítettük *(mert mondjuk a merevlemezt ki kellett cserélni)*, akkor az új merevlemezen létre kell hozni egy NTFS formátumú kötetet *(plusz valamilyen meghajtóbetűjellel lássuk el)*. Ezek után futassuk a risetup –check parancsot, ez majd létrehozza a RIS könyvtárszerkezetet, meg minden egyebet ami a SIS működéséhez kell. Csak mindezek után érdemes indítani a SIS kötet visszaállítását! Alapjáratban a SIS az egész kötetet felügyeli, azonban egyszerű Notepad segítségével ki lehet vonni könyvtárakat a SIS felügyelete alól úgy, hogy a SIS Common Store könyvtárban levő groveler.ini-be beleírjuk, mit is szeretnénk kihagyni. Valahogy így:

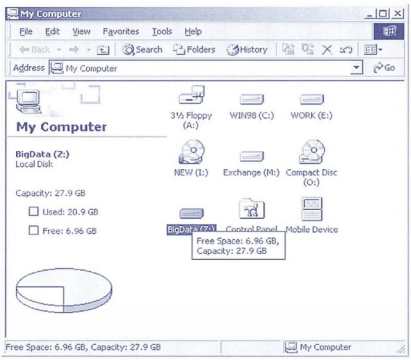


Megtiltjuk a SIS-nek a \temp könyvtár kezelését

Mindegy, hogy az egyenlőségjel bal oldalára mit írunk, a lényeg, hogy minden egyes könyvtárnak külön bejegyzés kell. Ezután csak újra kell indítani a SIS Groveler szolgáltatást, és többet nem fogja piszkálni az itt felsorolt könyvtárakat.

Együttélés a SIS-sel

A fentiekből – talán – az is következhet, hogy a SIS köteten saját adatainkat is tárolhatjuk, hisz a Groveler lebeszélhető bizonyos könyvtárak matatásáról. Azonban gondoljunk bele: a SIS köteten lévő szabad hely csak látszólagos! Ha azt látjuk, hogy a lemezen – mondjuk – 7 giga hely van, akkor egy olyan irányszámot látunk, ami megmutatja, hogy a Groveler pillanatnyilag ennyi helyet volt képes szabaddá tenni.
Ha most nekiállnánk újabb imagek feltelepítésének, a szabad hely mennyisége átmenetileg leesik, majd a SIS szorgoskodása nyomán ismét visszakúszik. Image töltögetés után érdemes – akár Performance Monitorral - megfigyelni! *(Ez utóbbi kissé macerás: először ugyanis parancssorból engedélyezni kell a Logical Disk objektumot a Diskperf paranccsal. Az alábbi ábra egyszerűbb módszert mutat, bár itt meg sűrűn kell ütni az F5-öt, hogy lássunk valamit...)*



Mennyi üres helyem van?

Folyt köv.

Dorner Csilla
MCSE
dorner.csilla@inteq.hu

Az RSA algoritmus

Felsőfokú tanulmányaim során a matek volt az egyik legérthetetlenebb, legunalmasabb tantárgyam. Vizsgára felkészüléskor *(amikor egyszerre félévnyi bizonyítást kellett volna elsajátítani)* igencsak nehezemre esett elolvasnom a matematikai regényeket, mert sajnos a beleékelődő képletek zavarták a folyamatos olvasást.

Erre tíz évvel később hasonló levezetésekkel fárasztom a tisztelt nagyérdeműt. Mivel tudom, hogy viszonylag kevesen matekoznak munkahelyükön, egyszerű képlettel kezdjük, hogy a berozsdásodott fogaskerekek fokozatosan indulhassanak be. De kérlek, ne add fel a harmadik oldalon! Ha kell, lapozz vissza, olvasd újra! Ha megemészted az RSA-t, soha többé nem fogod ugyanúgy látni a világot. Kitartás!

Gyermekkoromban minden nyári szünetben több hetet vidéken, nagyapáméknál töltöttem. Volt ott minden, mit városi gyerek csak rajzfilmen vagy kifestőkönyvben lát: háziállatok, kukoricagóré, lovas kocsi stb. Nagyapám minden évben elkápráztatott az alábbi „matematikai" bravúrral, s én minden évben újra és újra rácsodálkoztam, hogy mik vannak:

1. számmisztika

-Gondolj egy számot. *(Gondolj egy számot, Kedves Olvasóm!)*
-Adj hozzá hatot. *(No mi lesz?)*
-Szorozd meg hárommal. *(Megy ez!)*
-Vond ki belőle a gondolt szám háromszorosát. *(Ugye még ez is megy fejben?)*
-Az eredmény: 18

Ha kicsit jobban belegondolunk, hamar rájövünk a „trükk" nyitjára.

```
(X+6)* 3 - 3X = 3X + 6 * 3 – 3X
azaz 3X kiesik, marad 3 * 6 = 18
```

Engem ezzel a feladvánnyal már húsz éve nem lehet megetetni.

Rivest, Shamir, Adleman

Ez a három úr egyike *(hármika)* annak a néhány kiválasztott elméleti matematikusnak, akik eredményeiket a gyakorlatban is hasznosítani tudták. A többi matematikus „csak" bebizonyítja, hogy végtelen számú prímszám van, vagy kiszámolja π *(pí)* értékét négymillió jegyig - egyszóval semmi aprópénzre váltható nem alkot.

A mi embereink azonban 1977-ben megalkották a nyílt kulcsú titkosítás máig el nem avult, fel nem tört algoritmusát, amellyel komplett iparágakat teremtettek. Sőt! Magyarországon épp most készül a digitális aláírásról szóló törvény – a hármak bandája tehát országokat is mozgat!

Annyira egyszerű az RSA algoritmus képlete, hogy valószínűleg a dolog nem is működik. Fogod a titkosítanó dokumentumot, felemeled egy gigantikus hatványra, majd az eredmény modulóját veszed... és ezzel látszólag elveszíted az eredeti dokumentumot. De mégsem, mert ha egy másik hatalmas számmal az eredményt ismét meghatványozod, majd megint modulóját veszed, visszakapod az eredeti doksit. Hmmm. De miért? És meddig? Mi az esélye és veszélye annak, hogy az RSA-t megtörik?

E sok kérdésre egyelőre nem adom meg a választ, először ugyanis ismét számmisztikázunk. Hatványozni fogunk, így számológép használata javasolt!

2. számmisztika

– Gondolj egy számot *(T)*. *(2 és 10 között, különben kiakad a számológéped.)*
– Most keress egy tetszőleges prímszámot *(N)*, mely NAGYOBB, mint az előző számod.
– Emeld a gondolt számot a *(N-1)*. hatványra.
– Vedd az eredmény modulusát N-nel *(modulus = maradékos osztás)*
– A maradék: 1

Matekul ez így fest:

$$T^{(N-1)} \bmod N = 1$$ **(ha N nagyobb mint T, és N prímszám)**

A fenti „trükk" megfejtéséhez már némileg több beleérzőképesség kell. De nem túl sok. Ezt a játékot már az ókorban is ismerték, Euler pedig bebizonyította, hogy a képlet minden prímszámmal működik, HA a prímszám nagyobb, mint a gondolt szám. Ezek a prímszámok tudnak valamit, amire a többi szám nem képes. Vegyük például ezt a másik, hasonló kiszámolót:

3. számmisztika

– Gondolj egy számot *(T)*. *(2 és 10 között, különben kiakad a számológéped. Ne ugyanazt, mint az előbb!)*
– Most keress egy tetszőleges prímszámot *(N)*, mely NAGYOBB, mint az előző számod.
– Emeld a gondolt számot N. hatványra.
– Vedd az eredmény modulusát N-nel – A maradék: **a gondolt szám**.

Ez már igen érdekes! Matekul kifejezve:

$$T^{N} \bmod N = T$$ **(ha N nagyobb mint T, és N prímszám)**

Ez valójában nem más, mint a fentebbi, második számmisztikai egyenlet, melynek mindkét oldalát megszoroztuk T-vel. Íme előttünk az RSA algoritmus alapja, egy hatványozós, modulós képlet, mely visszaadja az eredeti számot! Ez a játék azonban egyfelhasználós, az RSA-t viszont párban kell játszani *(Titkosító, Megfejtő)*, másrészt ki engedte meg, hogy egy modulós képlet mindkét oldalát büntetlenül megszorozzuk T-vel? Ki mondta, hogy igaz marad az egyenlet? Egyelőre zsákutcába jutottunk.

Tegyük félre egy pillanatra a játszadozást. Mi is a cél?

Ariadné, Indiana Jones és a függvények

Célunk egy olyan függvénypár találása, melyek egymást kiegészítve visszavisznek a kiindulóponthoz. Ha a két függvény egyikét a Titkosító, másikát a Megfejtő alkalmazza, ugyanazt az adatot kapjuk vissza. Ilyen függvénypárokat igen egyszerű találni, mivel gyakorlatilag mindegyik ilyen. Legyen a titkosítás például az, hogy a titkosítandó számhoz hozzáadunk egyet *(hű, de bonyolult!)*:

```
Adat + 1 = Titok
```

Ebben az esetben a kiegészítő függvény, mely az eredeti adatot visszaadja:

```
Titok – 1 = Adat
```

Ugyanilyen formán a világ számtalan függvénye használható gagyi nyílt kulcsú titkosításra a SIN()-tól a négyzetre emelésig, de egytől egyig mind gyatra, mert tulajdonképpen nincs is szükségem a fordított függvényre – bőven elég, ha az eredeti műveletsorozatot lépésről lépésre visszafelé hajtjuk végre. Ariadné függvények, mert húzzák maguk után a fonalat, amelynek segítségével bármilyen bonyolult labirintusból kitalálunk. De vegyük csak Indiana Jones példáját. Bemegy egy barlangba, ahol a háta mögött leomlik a fal, árvíz mossa el a hidakat, karók jönnek ki a földből, és ha még ez sem elég, hát meggyullad valami. Ott áll hősünk a barlang legmélyebb pontján bezárva, látszólag reménytelenül *(orra előtt a dinkák ősi bálványszobra)*, de ebben a siralmas helyzetben hirtelen megtalálja a kivezető utat ábrázoló térképet. Ha nem Indiana Jones kavarodott volna be a barlangba, tutibiztos nem talált volna ki, erről tanúskodnak a falak mentén némán üldögélő csontvázak. *(Hollywood nem spórol a dramaturgiai kaptafákkal.)*
Ez kell nekünk! Olyan függvény, melybe ha besétáltunk, visszaút nincs többé. Hiába próbálkozunk a lépések fordítottjával, a mennyezetről lezuhant szikla utunkat állja. Kijutni csak akkor tudunk, ha a bálvány talpa alól kihúzzuk a térképet. Na ez az RSA. Aki nem tudja, hogy a bálvány lába alatt van a térkép, az meghal.
Az ilyen függvényeket csapda *(trapdoor)* függvényeknek nevezzük. Az első működő példával Diffe és Hellmann rukkolt elő 1976-ban, pontosan a második számmisztikára alapozva. Most pedig lássuk, mi kell még:

1. Moduloaritmetika és kongruencia
2. Relatív prímek
3. Euler fíje
4. A szorzás, mint a helyzet megmentője

1. Moduloaritmetika, kongruencia

Ha az a cél, hogy beomoljon az alagút, keresve sem találhatnánk jobb matematikai műveletet a modulonál. Oly kiválóan fejbecsapja az áldozat számot, hogy az eredményből soha nem leszünk képesek visszaállítani az eredetit. A modulo, vagy maradékos osztás életünk szerves részét képezi. Amikor azt mondjuk, kettőkor kezdődik a NATE, akkor valójában 14 óráról beszélünk, modulo 12. Modulo 12-vel fejbevágva a 14 tulajdonképpen egyenlő 2-vel. Ezt a furcsa egyenlőséget a matematikában kongruenciának hívják, és hármas egyenlőségjellel jelölik *(≡)*. Bármely szám bármely másikkal képzett modulusát igen könnyű vizualizálni. A

```
127 mod 21 = 1
```

nem más, mint egy 21 „órából" álló számkör, melyre a 127-et „felcsavarjuk". Többször körbefut, majd a vége valameddig elér a 21-es számkörön. Ez a maradék a moduló végeredménye. A fenti írásmód a „hétköznapi". Ugyanez matekul:

```
127 ≡ 1 (mod 21)
```

Ennek olvasata: a 127 valójában ugyanaz, mint az egy, ha modulo 21-ben gondolkozunk. A NATE 2-kor kezdődik matekul így fest:

```
14 óra ≡ 2 óra (mod 12-ben gondolkodva)
```

Hol találunk még modulót mindennapjainkban? Hát a számítógépeinkben. A mai elterjedt processzorok 32 bitesek, ami tulajdonképpen annyit jelent, hogy 2^32-nel modulálnak minden egész számon végzett matematikai műveletet. Ha túl nagy fába vágjuk a fejszénket, a túlcsordulás miatt könnyen azon kaphatjuk magunkat, hogy

```
1 + 1 ≡ 2^32 + 1 + 1
```

Hármas egyenlőségjellel persze. De akkor már bánhatjuk. Modulo a hét napjai *(mod 7)*, a beszélt nyelv *(220V= kettő-húsz, azaz mod 100)* stb. Modulo mindenütt.
Érdekes, és később hasznunkra válik majd, hogy a moduloaritmetikában *(szinte)* bármikor fejbecsaphatjuk a művelet tagjait a modulussal, a számítás előtt, vagy a végeredményen - egyre megy. Példa:
Most 11 óra van. Mennyit mutat majd a vekker 27 óra múlva?
1. számítás: a modulo a végeredményre sújt le:

```
11+27 = 38 ≡ 2 (mod 12 esetén)
```

2. számítás: a moduloval már a kezdőértékeket fejbecsapjuk

```
27 ≡ 3 (mod 12-vel lecsapva)
11 ≡ 11 (mod 12-vel lecsapva. Nem változik)
11 + 3 = 14 ≡ 2 (mod 12)
```

Ez a trükk ráadásul nemcsak összeadásra, hanem könnyen belátható módon gyorsított összeadásra, vagyis szorzásra is működik! Egy tyúk levágásának és feldolgozásának ideje 3 óra *(forralás, belezés, belsőségek megtisztítása, darabolás, csomagolás, fagyasztás)*. Sajnos 50 élő tyúkot kaptunk vidékről bele a panellakás kellős közepébe. Ha délelőtt 11-kor kezdem, vajon fent lesz-e a napocska, amikor befejezem? Egyedül csinálom, a feleségem nem ért hozzá *(nem sokat nyaralt vidéken gyermekkorában)*. Mivel a napocska hollétére vagyok kíváncsi, mod 24-gyel dolgozom:

1. számítás: mod 24 a végeredményre

```
3 * 50 = 150 ≡ 6 (mod 24) A nap épp
kelőben lesz.
```

2. számítás: mod 24 a kiindulási adatokra

```
50 ≡ 2 (mod 24)
2 * 3 = 6. A nap épp kelőben lesz.
```

Hogy én beledöglök-e a több mint hatnapi megszakítás nélküli tyúkpucolásba? Nem érdekes... Ennél sokkal fontosabb, hogy a modulo aritmetika bármikor bevethető. Például, ha egy modulós számítás során kezdene elszaladni az eredmény a végtelen felé. Csak lesújtunk rá a modulóval, és rögtön nyugton marad, mi pedig számolhatunk tovább.

2. Relatív prímek

Megy még a prímtényezőkre bontás? Hatodik osztályos anyag. És hogy kapom meg két szám legnagyobb közös osztóját? Segítek.

Prímtényezőkre bontás:

```
18 = 2 * 3 * 3
30 = 2 * 3 * 5
```

A legnagyobb közös osztó kiszámítása

Két szám közös prímtényezőinek szorzata. A fenti esetben a 2, és a 3 közös, így 18 és 30 legnagyobb közös osztója 6. Ha két szám prímtényezői egyáltalán nem egyeznek, akkor a legnagyobb közös osztójuk 1, s a két szám relatív prím egymáshoz képest. Példa:

```
28 = 2 * 2 * 7
45 = 3 * 3 * 5
```

Prímtényezőikben nincs közös, legnagyobb közös osztójuk 1. A 28 és a 45 egymáshoz képest relatív prímek. Hogy mi a csudára jó ez nekünk? Hát kipróbálhatjuk, hogy a hatványozós-modulós játék *(a 3. számmisztika)* egészen biztosan csak akkor működik-e, ha N prím, vagy netalán a relatív prímek is eleget „tudnak", és elég, ha N relatív prím T-hez képest?
HA a hatványkitevőt fel tudnánk bontani két részre, valahogy így: N=P*Q *(ami ugyebár addig nem megy, amíg ebben a számjátékban N kötelezően prím)* akkor

$$T^{(P*Q)} \equiv T \pmod N$$

ugyanazt adná, a hatványozást pedig két különböző emberre bízhatnánk

$$\text{Titkosító: } T^P \equiv R \pmod N \text{ nyomása alatt)}$$
$$\text{Megfejtő: } R^Q \equiv T \pmod N$$

miénk is lenne az RSA.

Első próba:

T=5 *(a titkosítandó szám)*
N=6 *(relatív prím T-hez, ha nem hiszed, számold ki)*
$5^6 \bmod 6 = \ldots 1$

Jaj. Ez nem az eredeti szám, ez biza nem 5. Ez nem jött be. Euler! Segíts!

3. Euler fíje – Mitől működik...

... a 3. számmisztika? Jobban utánaolvasva kiderül, hogy a hatványkitevő csak prímszámok esetén ugyanaz, mint a modulus *(N)*, és ott is csak azért, mert „véletlenül" így jön ki minden prímszámnál. A hatványkitevő ugyanis nem más, mint a modulus relatív prímjeinek száma + 1. Ez jó bonyolultan hangzik, úgyhogy vegyünk példákat:

- 9-hez képest relatív prímek *(azaz a legnagyobb közös osztójuk 1)*: 1, 2, 4, 5, 7 és 8. Összesen 6 darab.
- 7-hez képest relatív prímek: 1,2,3,4,5 és 6. *(Azaz minden nála kisebb szám, merthogy a 7 maga prím.)* Összesen megint 6 darab.

Euler a görög φ *(fi)* betűvel jelölte azt, hogy egy számnak hány relatív prímje van. Így φ(9) = 6, és φ(7) szintén 6 (bár merő véletlenségből annyi, nincs semmi köze a 7-nek a 9-hez). Prímszámoknál borzasztó egyszerű kiszámítani:

$$\varphi(N) = N-1$$

minthogy mindegyik nála kisebb szám relatív prímje egy prímnek. A 3. számmisztika kijavítva:

$$T^{\varphi(N)+1} \equiv T \pmod N$$

Lássuk az előző példát, hátha így összejön!

Második próba

T=5 *(a titkosítandó szám)*
N=6 *(relatív prím T-hez, 2*3 legnagyobb közös osztója 5-tel: 1)*
φ(6)=1 és 5, azaz összesen 2 darab
$5^{(2+1)} \bmod 6 = \ldots$jaj de izgulok ... = 5!

Hurrá! visszakaptuk az eredetit! Pedig a matekozáshoz használt N nem is prím! Ki meri azt állítani, hogy a 6-os prímszám? Elégtelen!
Ez már majdnem RSA, csak egyfelhasználós.

RSA!

Itt buktunk el az előbb: ha a hatványkitevőt fel tudnánk bontani két részre, valahogy így:

$$\varphi(N)+1=P*Q$$

ami most már megy, mert a 6 messze nem prím, akkor

$$T^{(P*Q)} \equiv T \pmod{N \text{ után}}$$

tök jól működne, a hatványozást pedig két különböző emberre bízhatnánk

$$\text{Titkosító: } T^P \bmod N \equiv R$$
$$\text{Megfejtő: } R^Q \bmod N \equiv T$$

és miénk lenne az RSA.

Harmadik próba:

A fenti példában a hatványkitevő 3. Ezt sajna csak így lehet szorzótényezőkre bontani: 1*3. Az 1 nem valami „jó" hatványkitevő, ugyanis nem csinál semmit a hatványozás során. Már megint kicsúszott a markunk közül ez a nyomorult RSA, de nem adjuk fel!

4. A szorzás, mint a helyzet megmentője
A második számmisztika gyúrása

Emlékeztetőül, ez volt az:

$$T^{(N-1)} \equiv 1 \pmod N$$

Most már tudjuk *(Euler segített)*, ez valójában

$$T^{\varphi(N)} \equiv 1 \pmod{N\text{-nel kupánvágva}}$$

Mindkét oldalt emeljük négyzetre:

$$T^{\varphi(N)} * T^{\varphi(N)} \equiv 1 * 1 \pmod N$$

Most köbre:

$$T^{\varphi(N)} * T^{\varphi(N)} * T^{\varphi(N)} \equiv 1 * 1 * 1 \pmod N$$

Bizvást menne negyedik hatványra is... Írjuk fel a köböst a hatványkitevők összevonásával:

$$T^{3*\varphi(N)} \equiv 1 \text{ (ne feledd: mod N!)}$$

Ebben az a röhej, hogy a moduloaritmetika miatt a hármas helyén akármilyen szám állhat, és nem zavarja köreinket! Hoppá! Így ha a φ(N)+1 felbontása olyan béna lenne, mint az előző bekezdésben *(1 és 3 jött ki)*, semmi gond, mert K*φ(N)+1 éppoly jó felbontási alany, így:
2+1 helyett bátran megpróbálkozhatunk akár a
7*2+1 *(=15)* felbontásával: 3 és 5.
Sokadik nekifutás.

Negyedik próba

T=5 *(a titkosítandó szám)*
N=6 *(relatív prím T-hez, 2*3 legnagyobb közös osztója 5-tel: 1)*
φ(6)=1 és 5, azaz összesen 2
K*φ(N)+1=7*2+1=15=P*Q. P=3, Q=5
Titkosító: T^e mod N=R, azaz 5^3 mod 6=5
Megfejtő: R^0 mod N=T, azaz 5^5 mod 6=5
Done!!!!!!!!! RSA-zunk!!!!!
A publikus kulcs P és N
A privát kulcs Q és N

Egy hétköznapi példa

A hét napjaival fogunk titkosítani. Az egyszerűség miatt az év negyedik napja lesz a titkosítottan átküldött infó, a modulus pedig 7, azaz vasárnap. Azért választottam ezt a példát, mert ebben oly természetes a modulus!
T=4, csütörtök *(a titkosítandó szám)*
N=7, azaz egy naptárlap napjainak száma (relatív prím T-hez, legnagyobb közös osztója 4-gyel: 1)
φ(7)=1, 2, 3, 4, 5 és 6, azaz összesen 6
K*φ(N)+1=4*6+1=25=P*Q. P=5, Q=5 *(ez bénaság, a két hatványkitevő sose legyen egyforma, de mi csak gyakorlunk)*
A titkosítás valójában a naptár előrepörgetése rengeteg nappal, így rengeteg lappal. Ahol az előrepörgetés megáll, megnézzük milyen napot írunk *(ez a Mod 7)*.

- Titkosítás: T^e mod N=R, azaz 4^5 mod 7=2, azaz kedd. Ezt küldöm el a partneremnek.
- Megfejtés: R^0 mod N=T, azaz 2^5 mod 7=4, azaz csütörtök.

A publikus kulcs P és N, azaz 5 és 7. Lássuk, ennek birtokában vissza tudjuk-e fejteni az eredeti napot. A modulo miatt sok „vért" veszítettünk, gyakorlatilag fogalmunk sincs, hogy hány napot rohantunk előre, kizárólag a maradék maradt. Ennélfogva T akármi lehet, mert végtelen sok olyan szám van, amit ha P-edik hatványra emelünk, majd mod 7-tel lefejezünk, keddet ad eredményül. Visszaút nincs. Előre pedig csak azok hatolhatnak, akik ismerik Q-t, a privát kulcsot.

Kulcsgenerálás

Már tudjuk, hogy a modulusnak nem kell prímszámnak lennie, bőven eléq, ha relatív prím a titkosítandó adathoz. Vajon hogy a csudába biztosítható, hogy a modulus:

1. Oltári nagy szám legyen
2. Relatív prím gyakorlatilag bármihez *(fontos.doc például)*
3. Meg tudjuk állapítani a φ-jét *(relatív prímjeinek számát)*

Hmm. Ez igazán nehéz feladatnak tűnne, ha vakon bökdösnénk a számok között. E helyett okosan a következőt tesszük:

1. Veszünk két bazinagy prímszámot, X és Y *(www.mersenne.org és egyéb módszerek, lásd BYTE cikkem)*
2. Ezek szorzata lesz N, a modulus, ami ugyan nem prím, de mivel két elvetemülten nagy prímszám szorzata, gyakorlatilag bármilyen nagy számhoz relatív prím lesz
3. Mindkét prímszámnak tudjuk a φ-jét *(prímszámokról lévén szó X-1 és Y-1)*, így N φje is adott: φ=(X-1)*(Y-1)
4. Ezután felbontjuk φ+1-et két szám szorzatára (P és Q). Nem nehéz, hisz „K*φ(N)+1 éppoly jó felbontási alany", lásd fenn. P és Q ideális esetben egymáshoz képest relatív prím, de ez nem szükséges feltétel
5. A bazinagy prímszámokat elhajítjuk. Többé nem kellenek.

A kulcspárok pedig (P, N) és (Q, N)

Gyenge pontok

Az RSA gyenge pontja nem az algoritmus, hanem a kulcsgenerálás. Mivel N része a publikus kulcsnak, az RSA addig „él", amíg nincs jobb módszer N prímtényezőkre bontásához, mint a próbálgatás. Az RSA Labs kihívja maga ellen a sorsot, és pályázatot hirdet prímtényezőkre bontásra tízezertől *(576 bites)* kétszázezer dolláros *(2048 bites)* díjért. Az **[1]** címen lehet nevezni!

Ellenőrzés

Ne mondd, hogy elsőre feldolgoztad. Nem igaz. Most légyszíves olvasd el elölről. Ami nem megy elsőre, majd megy másodikra. Ami nem megy másodikra, madj sikerül harmadikra. Ami nem megy harmadikra...

Fóti Marcell
MZ/X

A cikkben szereplő URL-ek:
[1] http://www.rsasecurity.com/rsalabs/challenges/factoring/index.html

Internet Security and Acceleration Server *(VI. rész)*

Biztonságos Windows 2000 DNS szolgáltatás kialakítása

Az e havi cikk némileg rendhagyó: nem lesz benne ugyanis szó az ISA Server-ről. Azért kívánkozik ide mégis ez a téma, mert a biztonságos DNS szolgáltatás éppúgy hozzátartozik egy jó tűzfalrendszerhez, mint maga a tűzfal. A Windows 2000 DNS szolgáltatása számos olyan funkciót tartalmaz, melyek kihasználásával biztonságosabb DNS környezetet építhetünk. Azt, hogy ezek közül melyeket használjuk ki, a DNS kiszolgáló rendszerben betöltött szerepe határozza meg.

A DNS kiszolgálók konfigurációja

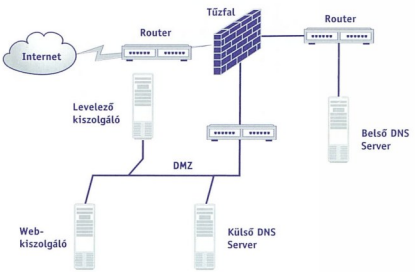
Számos módszert alkalmazhatunk a DNS környezet kialakításakor. Nyilvánvaló, hogy nincs egyetlen jól bevált recept, a rendszer felépítését mindig az adott cég igényei *(és persze pénztárcája határozza meg)*.

DNS zárt környezetben

Ilyenkor a DNS kiszolgálók védett környezetben vannak. Az, hogy egyáltalán akarjuk-e magukat a kiszolgálókat biztonságossá tenni, csak paranoiánk, és az erre fordítható időnk határozza meg. Zárt környezetről beszélünk, ami alatt azt kell érteni, hogy vagy egyáltalán nincs Internet-kapcsolatunk, vagy a külső router és/vagy tűzfal minden befelé tartó DNS forgalmat *(UDP és TCP port 53)* blokkol *(mármint azt a minimálisat, ami az egyáltalán nem publikált DNS kiszolgálónkra betévedne)*. Ha Active Directorynk is van, ajánlott az Active Directory Integrated DNS alkalmazása *(egyes ajánlások szerint... mások szerint viszont nagyon nem)*. Egyvalamit viszont mindenképpen célszerű korlátoznunk: a zónák letöltését *(zone transfer)* csak a Name Servers fül alatt felsorolt kiszolgálóknak engedélyezzük *(részletesebben lásd később)*. Megfontolandó az is, hogy az ügyfélgépek csak és kizárólag a belső DNS kiszolgálótól kérdezhessenek, az Internetről pedig csak a DNS kiszolgáló. Ha ilyen környezetet építünk, számoljunk azzal, hogy Internetes DNS szolgáltatásunk *(ezzel együtt gyakorlatilag az összes nyilvános szolgáltatásunk)* valamely Internetszolgáltatóra van bízva.

DNS Internetes jelenlét esetén

Ma már szinte nélkülözhetetlen az állandó Internet-kapcsolat. Ilyen kiépítésnél általában két megoldandó feladat vár ránk. Egyrészt DNS információkat kell biztosítanunk saját szolgáltatásaink eléréséhez *(az egész Internetnek)*, másrészt saját ügyfélgépeinknek az Internetről kell DNS információt biztosítanunk. Ilyenkor már célszerű lehet kettéválasztani a külső DNS szolgáltatást, és a Windows 2000 domain-hez kapcsolódó DNS-t.



DNS Internetes jelenlét esetén

Az ábrán látható eszközök funkciói:

Tűzfal

- Minden belső DNS felé irányuló kapcsolatot blokkol
- Csak a külső DNS 53-as portjáról és portjára menő forgalmat engedélyezi

Belső DNS kiszolgáló

- Windows 2000
- Megtalálható rajta a külső és belső kiszolgálók adatait tartalmazó Forward és Reverse Lookup zónák Primary vagy Secondary példánya
- A dinamikus frissítés engedélyezve van

Külső DNS Server:

- Bármilyen operációs rendszer
- A dinamikus frissítés tiltva van
- Csak külső kiszolgálókra vonatkozó bejegyzéseket tartalmazó Forward és Reverse Lookup zónák találhatók rajta

Ilyen rendszer esetén a következő biztonsági beállítások javasoltak:

- Az Active Directoryval integrált DNS kiszolgálókat csak belső kérések kiszolgálására használjuk.
- A zárt környezetben leírtakhoz hasonlóan *(hiszen végül is itt a belső kiszolgálóink zárt környezetben vannak)* csak a Name Servers fül alatt felsorolt kiszolgálóknak engedélyezzük a zónák letöltését.
- Tegyük meg ugyanezt a külső kiszolgálókon is. A zónák letöltését célszerű minél kevesebb külső kiszolgálónak *(vagy esetleg egyáltalán nem)* engedélyezni.
- Végezzük el a fájlrendszer és a registry biztonsági beállításait *(lásd később)*.
- Mint minden Internetre kihelyezett kiszolgálónál: csak a feltétlen szükséges szolgáltatások fussanak a külső DNS kiszolgálókon.
- A külső kiszolgálókon tiltsuk meg a zónák dinamikus frissítését *(Dynamic Updates)*.

A kliensek számára Internetes névfeloldást ilyenkor végezhet a belső kiszolgáló is, de még jobb, ha a belső kiszolgá-

ló*(ko)*n a külső kiszolgálókat használjuk forwarder-ként. Ha viszont a belső DNS kiszolgáló egy Active Directory fa/erdő egy részét szolgálja ki, az általa nem feloldható kéréseket a felette levő szint DNS kiszolgálójának továbbítja.

DNS Internetes jelenléttel és reverse lookuppal

Számos rendszerben szükség lehet arra, hogy ellenőrizzük, hogy valaki „honnan jött". Ezt az ellenőrzést gyakran a reverse lookup zónák segítségével végezzük el. E zónák segítségével nem a nevet oldjuk fel IP címre, hanem éppen fordítva *(innen származik a reverse lookup elnevezés is)*. Az IP címből feloldott nevet ezután használhatjuk annak ellenőrzésére, hogy valaki például „.hu" domain-ből nézelődik nálunk. Ezt a funkcionalitást két módon biztosíthatjuk:

1. Adjunk hozzá a külső DNS kiszolgálóhoz egy reverse lookup zónát, amely tartalmazza az összes Interneten megjelenő IP címünket. Az IP címeket társítsuk nevekkel. Ezek a nevek lehetnek akár fiktív nevek is, de célszerűbb inkább valósakat adni, mivel létezik olyan eset, amikor a teljes nevet ellenőrzi a fogadó fél. A DNS kiszolgálókat az előzőekben leírtak szerint állítsuk be. A hamis nevek használata akkor lehet célravezető, ha a belső kliensek legális IP címekkel rendelkeznek. Ha hamis neveket adunk meg, erősen korlátozzuk a potenciális támadókat a belső hálózat teljes és pontos feltérképezésében.

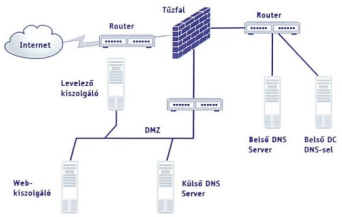
Az eszközök funkciói az alábbiak szerint alakulnak:

Tűzfal
- Minden belső DNS felé irányuló kapcsolatot blokkol
- Csak a külső DNS 53-as portjáról és portjára menő forgalmat engedélyezi

Belső DNS kiszolgáló
- Windows 2000
- Megtalálható rajta a külső és belső kiszolgálók adatait tartalmazó Forward és Reverse Lookup zónák Primary vagy Secondary példánya
- A dinamikus frissítés engedélyezve van

Külső DNS kiszolgáló:
- Bármilyen operációs rendszer
- A dinamikus frissítés tiltva van
- Csak külső kiszolgálókra vonatkozó bejegyzéseket tartalmazó Forward és Reverse Lookup zónák találhatók rajta. A Reverse Lookup zónában található nevek nem valódiak.

2. Ha a belső gépeink is legális címekkel rendelkeznek, megtehetjük azt is, hogy a külső DNS kiszolgálóhoz hozzáadunk egy reverse lookup zónát, amely a belső hálózat másodlagos zónája. A belső kiszolgálón adjuk hozzá a Name Servers fül alatt a külső DNS kiszolgálót is, így az képes lesz a zóna letöltésére. Állítsuk be úgy a tűzfalat és/vagy router-t, hogy a külső és egy belső DNS kiszolgáló között a zóna átvitele engedélyezve legyen *(ha hamis neveket szeretnénk megadni, állítsunk üzembe erre a célra egy belső DNS kiszolgálót, amely a hamisított reverse zónát tartalmazza)*. A DNS kiszolgálókat itt is az Internetes jelenlétnél leírtaknak megfelelően állítsuk be. Emellett, ha hamisított neveket akarunk használni, célszerű az e célt szolgáló kiszolgálót csak erre dedikálni, mivel a belső DNS kiszolgáló Start of Authority *(SOA)* bejegyzése megjelenik a reverse lookup zónában, vagyis az Interneten is.



DNS Internetes jelenléttel és másodlagos reverse lookup zónával

Az ábrán látható eszközök funkciói:

Tűzfal
- Minden belső DNS felé irányuló kapcsolatot blokkol
- Csak a külső DNS 53-as portjáról és portjára menő forgalmat engedélyezi
- Engedélyezi az 53-as porton menő forgalmat a külső és belső DNS kiszolgálók között

Belső DNS kiszolgáló
- Windows 2000
- Megtalálható rajta a Windows 2000 tartomány Secondary Reverse Lookup zónája
- Minden más szolgáltatás le van állítva rajta

Belső tartományvezérlő DNS-el
- Windows 2000
- Megtalálhatók rajta a Windows 2000 tartomány Primary Forward és Reverse lookup zónái
- A dinamikus frissítés engedélyezve van

Külső DNS kiszolgáló:
- Bármilyen operációs rendszer
- A dinamikus frissítés tiltva van
- Külső kiszolgálókra vonatkozó bejegyzéseket tartalmazó Forward Lookup zóna található rajta
- Van rajta egy Secondary Reverse Lookup zóna, melynek információit a belső kiszolgálóról tölti le

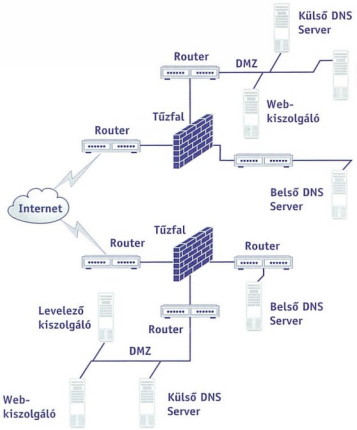
DNS Internetes jelenléttel és belső Forward és Reverse Lookup zónákkal

Ez ugyan nem ajánlott, de szükséges lehet *(például abban az esetben, ha egy Windows 2000 erdő vagy fa az Interneten keresztül van összekapcsolva)*. Ha a belső DNS bejegyzéseink kikerülnek az Internetre, a külső támadók egyszerű DNS lekérdezések segítségével képesek lesznek a belső hálózat teljes feltérképezésére.

Többféle megoldás létezhet, például:
- Használjunk biztonságos VPN csatornákat a domainek közti biztonságos zónaátvitelekhez. Ezzel védjük a belső DNS bejegyzéseinket az avatatlan külső szemlélődőktől. A külső kiszolgálókhoz használjuk a fent ismertetett megoldások valamelyikét. Egy ilyen rendszer látható következő ábránkon. Összegezve: ez a megoldás jó, mert az összes biztonsági igényt kielégíti.
- Csak azokat a bejegyzéseket adjuk hozzá a külső zónákhoz, melyek feltétlenül szükségesek a hálózat működéséhez. Ez már rosszabb megoldás. Bár a teljes belső hálózat valószínűleg nem térképezhető fel teljesen, az így közzétett bejegyzések bárki számára elérhetőek.

- Állítsuk be úgy a külső DNS kiszolgáló forward és reverse lookup zónáit, hogy azok egy belső DNS kiszolgáló másodlagos zónái legyenek. Ez az a megoldás, ami nem megoldás. Ilyen beállításokat még véletlenül se használjunk! *(Én is csak elrettentésként írtam ide)*

A fentí megoldások alkalmazásának mindegyike valamekkora kockázattal jár. Nyilvánvaló, hogy mindig van kockázat, ha a DNS zónáinkat az Internetet használva csereberéljük a tartományok között, de ha már így teszünk, törekedjünk a lehető legnagyobb biztonságra. Természetesen az elsőként említett módszer használata ajánlott. Ráadásul még pénzbe sem kerül.



3. ábra: DNS Internetes jelenléttel és belső Forward és Reverse Lookup zónákkal

Az ábrán látható eszközök funkciói:

Tűzfalak

- Minden belső DNS felé irányuló kapcsolatot blokkol
- Csak a külső DNS 53-as portjáról és portjára menő forgalmat engedélyezi
- Engedélyezi a routerek közti, titkosított forgalmat

Routerek

- A routerek építik fel a belső DNS kiszolgálók 53-as porton folytatott forgalmát védő VPN csatornát

Belső DNS kiszolgálók

- Windows 2000
- Megtalálható rajta a külső és belső kiszolgálók adatait tartalmazó Forward és Reverse Lookup zónák Primary vagy Secondary példánya
- A dinamikus frissítés engedélyezve van

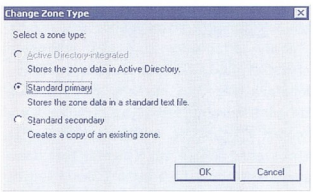
Külső DNS kiszolgálók:

- Bármilyen operációs rendszer
- A dinamikus frissítés tiltva van
- Csak külső kiszolgálókra vonatkozó bejegyzéseket tartalmazó Forward és Reverse Lookup zónák találhatók rajta

Ajánlott azoknak a helyeknek a biztonságossá tétele, melyeket a DNS kiszolgáló a zónainformációk tárolására használ. A zónainformációk vagy az Active Directory-ban, vagy a merevlemezen levő fájlokban tárolódnak.

Áttérés Active Directoryval integrált DNS kiszolgáló használatára

Lehetőségünk van a DNS zóna Active Directoryval integrált zónára való átalakítására. Ahhoz, hogy ezt megtehessük, a DNS kiszolgálónak Windows 2000 tartományvezérlőn kell futnia. Ha a DNS kiszolgáló nem tartományvezérlőn van, ez az opció szürke, tehát nem választható *(mint ahogy az a következő ábránkon látszik is)*.



Zónatípusok, ha a DNS kiszolgáló nem tartományvezérlőre van telepítve

Ha viszont a DNS kiszolgáló Windows 2000 tartományvezérlőn található, a zónatípusok között megjelenik az Active Directory-val integrált zóna. Ez a zónatípus számos előnyt nyújt, ezért használata *(feltételesen)* ajánlott. Ezek közé az előnyök közé tartozik például az, hogy a zónainformációk tárolásának, replikációjának és biztonságos helyen való tárolásának gondját az Active Directory leveszi a vállunkról *(ha még a sört is behozná...)*. Ha ilyen zónatípust használunk, bekapcsolt állapotba kerül az „Only secure updates" opció a dinamikus zónafrissítésekhez *(Dynamic Updates)*. Ennek a lehetőségnek a használata egyébként is javasolt, és DHCP használata esetén elég nehezen nélkülözhető a dinamikus frissítés használata.

A zónafájlok és a registry biztonsága

Ha a DNS kiszolgáló nem az Active Directory-ban tárolja a bejegyzéseket, ajánlott a fájlokat tartalmazó könyvtár biztonságossá tétele. A javasolt biztonsági beállítások:

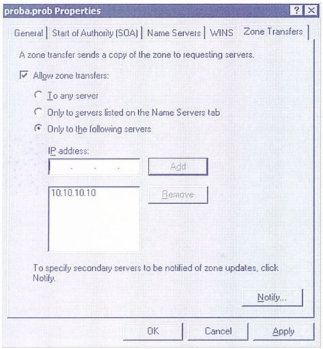
Könyvtár vagy fájl	Felhasználói csoportok	Ajánlott jogosultságok
A %SystemRoot%\DNS könyvtár, alkönyvtárai, és a benne található fájlok	System	Full Control

Ajánlott az összes DNS kiszolgálón a registry alábbi biztonsági beállításait elvégezni. Ezek a beállítások megakadályozzák, hogy a jogosulatlan felhasználók megtudják, vagy megváltoztassák a zónafájlok helyét.

Registery kulcs	Felhasználói csoportok	Ajánlott jogosultságok
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\DNS	Administrator	Full Control
	System	Full Control

A zónák letöltésének szabályozása

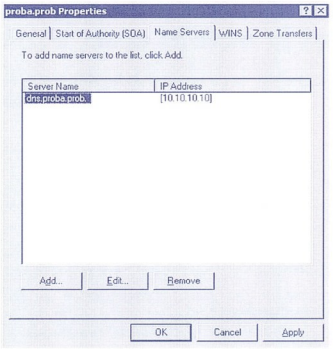
A DNS kiszolgálók biztonságossá tételének fontos lépése a zónafájlok letöltésének szabályozhatósága. A Windows 2000-ben a zónánként beállítható hozzáférési listák szabályozzák a zónák letöltését. Mivel a zónák átvitelekor az adott zóna összes bejegyzése átmásolódik egyik kiszolgálóról a másikra, nagyon fontos, hogy a Windows 2000 tartomány forward lookup zónáját ne vigyük át olyan kiszolgálóra, mely nem tagja a tartománynak. Az alábbi képen látható a zónaátvitelre vonatkozó négyféle beállítási lehetőség.



Zónák átvitelének szabályozása

A zónaletöltésre vonatkozó négy beállítási lehetőség:

1. Nem engedélyezzük a zónák letöltését:
   Ez a beállítás megakadályozza a kiszolgálóról a zóna letöltését. Természetesen a kiszolgáló ettől függetlenül tölthet le zónákat más DNS kiszolgálókról, és a DNS ügyfelek kéréseit is megválaszolja. Nevezhetnénk ezt akár alapbeállításnak is, melyet csak akkor változtassunk, ha mindenképpen szükség van rá.
2. A zóna letöltésének engedélyezése bármely kiszolgálónak:
   Ez a beállítás bármely szabályos zónaletöltési kérésre engedélyezi a DNS zóna letöltését. Bár ez az alapértelmezett beállítás, használatát egyik DNS kiszolgálón sem ajánlanám.
3. A zóna letöltésének megengedése azoknak a DNS kiszolgálóknak, melyek a Name Servers fül alatt fel vannak sorolva:
   Mivel ez a lista tartalmazza a tartományban található összes DNS kiszolgálót, ezt a beállítást arra az esetre javasoljuk, ha csak tartományon belüli zónaátvitel szükséges. Például ha a DNS kiszolgáló a Windows 2000 Domain zónaadatait tartalmazza, ez az opció lehetővé teszi a tartomány DNS kiszolgálóinak zónaadataik egymással való megosztását.



Ismert DNS kiszolgálók

4. A zóna letöltésének engedélyezése az IP cím listán szereplő IP címekre.Ez a beállítás a fent említett IP cím lista alapján engedélyezi, vagy utasítja vissza a zónák letöltését, vagyis adott, DNS domain-en kívüli kiszolgálóknak engedélyezhetjük a zónák letöltését. Ez a beállítás javasolt például abban az esetben, ha a kommunikáció védett DNS, és olyan kiszolgáló között zajlik, mely elérhető az Internetről. Az Active Directory DNS bejegyzéseit tartalmazó zónák Internetről elérhető kiszolgálókra való átvitele nem javasolt.

Router és tűzfal beállítások

A DNS forgalom az 53-as porton *(mind UDP, mind TCP)* zajlik. Éppen ezért a tűzfalon és a router-en át kell engedni az ezekre a portokra menő forgalmat, hogy a kliensek és más kiszolgálók is használhassák a DNS-t.
Az 53-as UDP portot a kliensek lekérdezései használják, a TCP portot pedig zónaátvitelhez. A legtöbb esetben a zónák kívülről történő letöltése nem megengedett, így ezt a portot a szűrést végző eszközökön blokkolni kell. Ha a DNS beállításai szerint megengedett a reverse lookup zónák átvitele a külső és a belső DNS kiszolgálók között, akkor a belső routert, a tűzfalat és a DMZ-ben található routert *(persze csak ha vannak ilyenek)* ennek megfelelően kell beállítani *(az 53-as TCP portot át kell engedni, de csak ezek között a kiszolgálók közt)*. Ha bármilyen problémát tapasztalnánk a Windows 2000-rel az 53-as UDP port DNS forgalomra való használatakor, keressük a Q260186 knowledge base cikket további információért.

Hát ennyi fért bele az e havi cikkbe. Minden Kedves Olvasónak Kellemes Karácsonyi Ünnepeket és Boldog Új Évet Kívánok!

BP

Biztonságos weboldalak: https://

Nem is hinnénk, hogy napi barangolásaink során mi mindent továbbítunk a hálózaton titkosítatlanul, prédaképp, csak egy gonosz hackerre várva. Cikkünkben bemutatjuk, hogyan készíthetünk titkosított SSL csatornán keresztül működő webhelyet a Windows 2000 IIS szolgáltatása segítségével.

Mi a baj a bejelentkezéssel?

Egy egyszerű példaalkalmazás segítségével bemutatjuk a böngészésben rejlő veszélyeket. Alkalmazásunk egyszerű, legyen például egy zártkörű webáruház egyik oldala: a felhasználó bejelentkeztetése után elkéri mondjuk a hitelkártyája számát és a hozzá tartozó PIN-kódot.

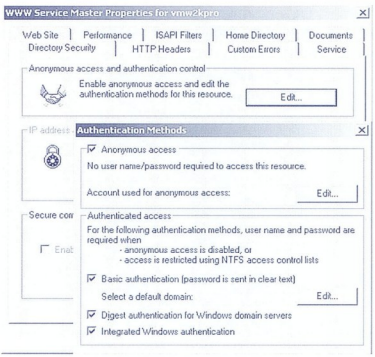
Webalkalmazásunkban a felhasználó bejelentkezés után megadhatja a személyes adatait

Kényes adatokat a kommunikáció során két ízben továbbítunk:

- A bejelentkezés során megadott felhasználónév és jelszó továbbításakor
- A bejelentkezés után a kérdőív elküldésekor

Az utóbbi – hacsak valami kliensoldali scriptvarázslattal nem titkosítjuk – mindenképpen titkosítatlanul halad a hálózaton, a bejelentkezéskor szerencsére más a helyzet, az ugyanis függ a kiszolgáló és a böngésző által kölcsönösen kiválasztott felhasználóazonosítási protokolltól. Az IIS5 által felajánlott protokollokat webkiszolgáló, webhely, könyvtár, sőt, akár fájlszinten is definiálhatjuk, ha az IIS kezelőfelületében megnyitható tulajdonságlapon megkeressük a „Directory Security" oldalt, abban pedig az „Anonymous access and authentication control" mezőben az Edit... gombra kattintunk. A lehetőségek a következők:

- Anonymous access – névtelen hozzáférés. Az erőforrás eléréséhez nincs szükség bejelentkezésre, ezért ilyenkor a jelszó nem is utazik a hálózaton. Az ügyfél egy beállítható felhasználó nevében tevékenykedik *(ez alapértelmezésben az IUSR_<számítógépnév>)*, de ez a párbeszédpanel felső részén található Edit... gomb segítségével módosítható. Ezzel az üzemmóddal minden böngésző kompatíbilis, hiszen használatához nem kell bejelentkezni.

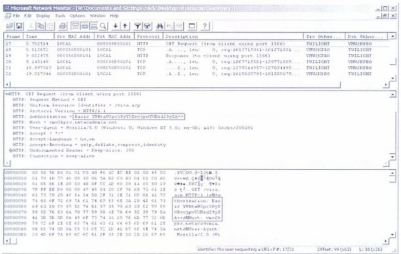


A felhasználóazonosítás módja az IIS-ben beállítható

A bejelentkezéssel járó protokollok pedig:

- Basic authentication – Nyíltjelszavas azonosítás. A felhasználónév kódolva, de nem titkosítva utazik a hálózaton *(erről később)*. Az üzemmódot a legrégebbi verzióktól számítva szinte minden böngésző támogatja. Az „Edit..." gomb hatására megjelenő dialógusablakban azt állíthatjuk be, hogy – ha nem ad meg tartománynevet, – a felhasználót melyik tartományban keressük.
- Digest authentication – az IIS5-ben megjelent azonosítási protokoll, előnye, hogy a hagyományos tűzfalakon keresztül is képes bejelentkezni. Bár voltak kezdeményezések, egyelőre csak Internet Explorer segítségével működik, ráadásul csak Windows 2000 tartományi kiszolgálók és tartománytag ügyfelek között.
- Integrated Windows authentication – Windows kiszolgálók és ügyfelek között *(nem feltétlenül csak tartományon belül)* használható azonosítási mód. Csak Internet Explorerrel működik *(akkor viszont akár automatikusan, a felhasználó megkérdezése nélkül, az aktuális felhasználónévvel/jelszóval is)*, viszont nem használható proxy tűzfalakon keresztül. *(Azaz: leginkább intranetes környezetben ajánlott)*.
- Tanúsítványalapú felhasználóazonosítás – ez nincs az ábrán, mert máshol kell keresnünk, csak a teljesség kedvéért soroltuk fel – egy későbbi cikkben ezt is bemutatjuk majd.

Látható, hogy széles körben – pláne, ha nem csak Internet Explorer böngészőket szeretnénk kiszolgálni, – egyedül a nyíltjelszavas felhasználóazonosítás tűnik járható útnak. A gond csak az, hogyha valaki a példaalkalmazásunk használata során generált hálózati forgalmat Network Monitorral, vagy bármilyen más hálózatfigyelő eszközzel meglesi, a következőt láthatja:

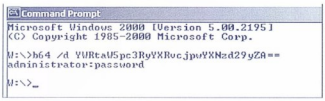


Nyílt jelszavas bejelentkezés a hálózaton

Talán olvasható az ábrán a kijelölt sor tartalma:

```
HTTP: Authorization = Basic YURtaW5pc3RyYXRvcjpwYXNzd29yZA==
```

A HTTP kérés Authorization fejlécében tehát egy kódolt sor utazik, a „Basic" sor a protokoll típusára utal. De mi történik, ha visszafejtjük a sor Base64 kódolását?



Basic authentication esetén a felhasználónév és jelszó nem marad titok

A hálózati forgalom természetesen ugyanígy titkosítatlan marad, olvashatóan *(még csak nem is kódolva)* utaznak a hálózaton a kérdőívbe bepötyögött értékek és persze a kiszolgálótól érkezett válasz is. Még ha a bejelentkezéshez erősebb protokollt választunk is *(ehhez persze le kell mondanunk az alternatív böngészőkről)*, a hálózati forgalom továbbra is szabad préda lesz. Marad a teljes hálózati forgalom titkosítása.

Webforgalom titkosított csatornán

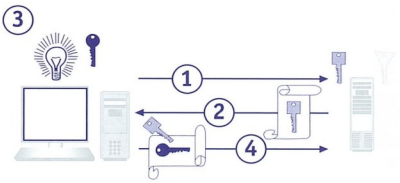
Ha a teljes kommunikációt *(beleértve már a bejelentkezést is)* titkosított csatornán bonyolítjuk, két *(vagy több)* legyet ütünk egy csapásra:

- Maradhat a nyílt jelszavas felhasználóazonosítás, és vele az alternatív böngészők is
- Titkosított lesz a teljes forgalom
- Adott esetben a proxykiszolgáló nem fog beleszólni a hálózati forgalomba *(ez pl. az Exchange 2000 Outlook Web Access esetén jöhet jól – ha az OWA nem megy proxy mögül http://-n, próbáljuk https://-en)*.

Az SSL csatornán folytatott webes kommunikációt *(nevezzük ezentúl HTTPS-nek)* gyakorlatilag az összes ma használatban lévő böngészőprogram támogatja *(ha emlékeim nem csalnak, talán a 2.0 verziótól felfelé)*, úgyhogy a teendőnk nem más, mint a webkiszolgáló felkészítése a nagy feladatra. Ehhez előbb azonban ismerjük meg a HTTPS kommunikáció alapjait! A HTTPS csatorna működése során a böngésző és a kiszolgáló egy közös kulcsú algoritmus segítségével titkosítja az adatokat. A kulcs minden alkalommal más és más *(a kapcsolatfelvétel során jön létre, és csak a csatorna bezárásáig „él")*. Ezt a kulcsot „session key"-nek, szakaszkulcsnak is nevezik, az ábrán kerek fejjel jelöltük.

Titkosítás közös kulccsal a már felépített HTTPS csatornán

Egy dolog azonban megoldatlan: hogyan jut el a közös kulcs mindkét félhez? Ehhez a nyílt kulcsú titkosítást vesszük alapul *(azaz olyan titkosítást, ami két, összetartozó kulcsot használ: ha az adatot az egyik kulccsal betitkosítjuk, azt csak és kizárólag a párjával lehet kinyitni)*. A webkiszolgáló rendelkezik egy ilyen kulcspárral, aminek egyik tagját közzéteszi *(ez a publikus kulcs)*, a másikat viszont megőrzi magának *(ez a privát kulcs)*. Lássuk, mi történik tehát, ha egy böngésző egy HTTPS kiszolgálót szeretne elérni:



A session key létrehozása nyílt kulcsú módszerrel

- ① A böngésző csatlakozik a kiszolgálóhoz
- ② A kiszolgáló elküldi a böngészőnek a saját publikus kulcsát tartalmazó tanúsítványát
- ③ A böngésző kitalál egy session key-t
- ④ A kitalált szakaszkulcsot a kapott publikus kulccsal betitkosítva visszaküldi a kiszolgálónak

A ④. lépésben utazó csomagot csak és kizárólag az tudja kinyitni, aki rendelkezik a titkosításhoz használt publikus kulcs privát párjával, az pedig egyedül a webkiszolgáló. Miután megkapta a csomagot, kiszedi belőle a szakaszkulcsot és máris kezdődhet a kommunikáció!

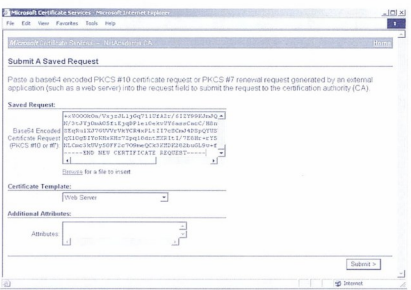
Kulcspár létrehozása és tanúsítvány kérése

A HTTPS használatához tehát az ügyfélnek nincs szüksége tanúsítványra, csak a webkiszolgálónak. A tanúsítvány valódisága azonban fontos dolog. Nyílt kulcsú tanúsítványokat külön erre szakosodott szervezetek *(Certification Authorities, CA-k)* adnak ki, de a Windows 2000 Server is rendelkezik ilyen szolgáltatással. Az ügyfelek számítógépén mindig található egy lista, amely azon CA-kat tartalmazza, amelyek által kiadott tanúsítványokban az ügyfél megbízhat. Ha egy olyan tanúsítvány érkezik, amelyet egy „ismeretlen" CA adott ki, a Windows erről figyelmezteti a felhasználót, aki eldöntheti, hogy mit szeretne tenni. A hivatalos tanúsítványkiadóktól származó tanúsítványok pénzbe kerülnek, a nemhivatalos *(pl. saját kiszolgálónkon futó)* tanúsítványkiadók által kiadottak esetén viszont – ha nem akarunk fennakadást –, frissítenünk kell majd a bizalmi CA-k listáját. Első lépésként generáljunk a webkiszolgálón egy kulcspárt, valamint készítsük el a CA-nak küldendő kérést! Ehhez kattintsunk az IIS felügyeleti eszközében a webhely tulajdonságlapjának „Directory Security" oldalán található „Server

Certificate" gombra! erre elindul a „Web Server Certificate Wizard". A varázslóban haladva válasszuk az új tanúsítvány létrehozását *(„Create New Certificate")*. Ha van a közelben elérhető CA szolgáltatás, a varázsló felismeri azt, és felajánlja a tanúsítvány azonnali, automatikus elkészí*(tte)*tését. Ha nincs, marad az offline mód: „Prepare the request now, but send it later". Ezután válasszunk egy nevet a tanúsítványnak *(célszerű a webhely URL-jét választani)*, majd döntsük el, milyen bithosszúságú titkosítást szeretnénk a tanúsítványon. Minél hosszabb kulcsot választunk, annál erősebb a titkosítás a tanúsítványon *(vigyázat! nem a későbbi HTTPS kommunikációban!)*. Írjuk be a szervezetünk, szervezeti egységünk nevét *(ezek informatív adatok)*, a Common Name oldalon pedig a webhely URL-jét. Az URL pontos megadása fontos, mert ha ez nem egyezik a böngészőbe írt címmel, a böngésző riasztani fogja a felhasználót. Ezután még meg kell adnunk az országot, megyét, várost *(ismét csak informatív adatok)*, végül pedig egy fájlnevet, ahova a varázsló a kérést elmenti.

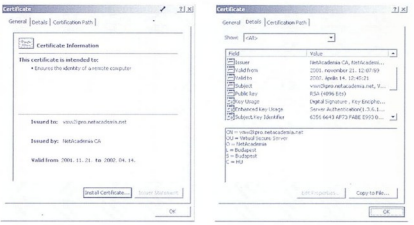
Így néz ki egy tanúsítványkérés belülről

Ha ezzel megvagyunk, keressünk egy tanúsítványkiadót, aki a kérésünket teljesíteni fogja. A Windows 2000 tanúsítványkiadó szolgáltatásának van webes felülete, cikkünkben azt fogjuk használni, de a lényeg tulajdonképpen mindig az, hogy a kérés tartalmát valamilyen módon eljuttassuk a tanúsítványkiadóhoz, ő pedig visszaküldhesse nekünk az elkészített tanúsítványt. Most egy Windows 2000 CA-tól fogunk tanúsítványt kérni. Kövessük a webes varázsló utasításait, egészen addig, míg nem kéri a kérés feltöltését.



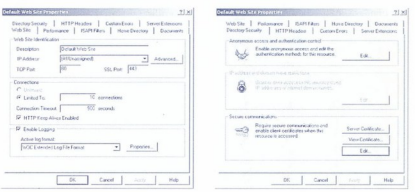
Útban a tanúsítvány felé

Ekkor illesszük be a kéréskor készült szövegfájlt tartalmát, válasszuk ki, hogy webkiszolgálóhoz szükséges tanúsítványt szeretnénk, majd haladjunk tovább. Ha minden sikerült, a következő oldalon már letöltheтjük a friss, meleg tanúsítványt.



A kész tanúsítvány

Ha elkészültünk, már csak telepítenünk kell a tanúsítványt. Ehhez indítsuk el újra a tanúsítványvarázslót az IIS felügyeleti eszközében. A varázsló indulásakor felismeri, hogy egy kérés már folyamatban van. Kéri, hogy adjuk meg a tanúsítványkiadótól kapott fájl elérési útját. A varázsló sikeres lefutása után nézzük meg a webhely tulajdonságlapjának két oldalát:



A tanúsítvány telepítése után már elérhetők a biztonsági beállítások

A kiszolgáló beállításai

Kezdjük a beállítások főoldalánál. Mint az ábrán is látható, a tulajdonságlap „Web Site" oldalán elérhető lett az SSL Port: mező. A varázsló általában ki is tölti ezt a mezőt *(az alapértelmezett portcím 443)*. Van azonban egy fontos dolog, amit nem szabad elfelejtenünk: az IIS lehetővé teszi, hogy egy IP címen több webkiszolgálót üzemeltessünk. A beérkező kéréseket ilyenkor az ún. Host Header mező értéke alapján továbbítja a megfelelő webhely felé. A HTTPS kommunikáció során azonban a Host Header fejléc is titkosítva van, és az IIS ezen a szinten még nem tudja azt visszafejteni. Emiatt a HTTPS kommunikáció mindig az adott IP címhez tartozó alapértelmezett webhelyre jut *(tehát ahhoz a webhelyhez, ahol nincs megadva Host Header a konfigurációban)*. Ha egy gépen több webhelyet is szeretnénk elérni HTTPS-en keresztül, akkor bizony több IP címre lesz szükség. A HTTPS beállítások másik ablaka a tulajdonságlap Directory Security oldalán, a Secure Communications mezőben található Edit... gombra kattintva jelenik meg.

A HTTPS csatorna bekapcsolása

Ha a Require secure channel *(SSL)* opciót bekapcsoljuk *(ezt megtehetjük nemcsak webhely, de könyvtár, sőt fájlszinten is)*, az adott erőforrás csak HTTPS-en keresztül lesz elérhe-

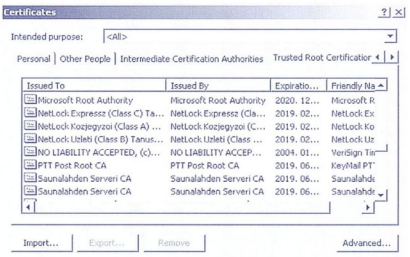
tő. Ha nem kapcsoljuk be, a HTTPS csatorna használata nem lesz kötelező *(de ettől függetlenül működni fog)*. A Require 128-bit encryption a forgalom erősebb titkosítását kapcsolja be, de ez csak akkor használható, ha mind a kiszolgálón, mind az ügyfélgépeken telepítve van a High Encryption Pack. A tulajdonságlap többi beállításáról következő számunkban lesz szó.

A puding próbája az evés

Első lépésben még ne kényszerítsük ki a HTTPS csatornát *(azaz a fenti ábrán látható ablakba ne tegyünk pipát)*, csak próbáljunk meg HTTPS-en keresztül csatlakozni. Ha a kiszolgáló tanúsítványa nem hivatalos CA-tól származik *(és ha a CA nem tagja a felhasználó tartományának, akkor ugyanis automatikusan megbízik benne)*, a következő ablakkal találkozhatunk:

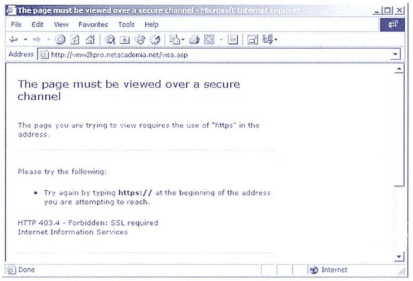
A tanúsítványt ismeretlen szervezet adta ki

A szolgáltatás ettől függetlenül használható, hiszen ha a Yes-re kattintunk, ettől függetlenül felépül a kapcsolat a kiszolgáló felé. Ha ezt a kellemetlenséget szeretnénk elkerülni, a tanúsítványkiadót *(vagy magát a tanúsítványt)* fel kell venni a megbízott tanúsítványkiadók listájába. Ehhez szükség van a tanúsítványkiadó saját tanúsítványára *(általában hozzáférhető a CA weboldalán, vagy más forrásból)*. Ha ezt sikerült megszereznünk, már csak telepítenünk kell. Ehhez kattintsunk az Internet Explorer Tools menüjének Internet Options parancsára. A megjelenő dialógusablak Content oldalán kattintsunk a Certificates... gombra, és megjelenik az aktuális felhasználó tanúsítványtára. Itt válasszuk a Trusted Root Certificates fület, majd az Import... gombra kattintva beimportálhatjuk a kapott tanúsítványt. *(Tartományi szinten mindezt természetesen Group Policy-vel is el lehet intézni)*.



A tanúsítványkiadók bizalmi listája

Ha a webkiszolgáló beállításainál kikényszerítjük a HTTPS csatorna használatát, a titkosítatlanul érkező ügyfél egy hibaüzenettel találja szembe magát:



Hasonló hibaüzenetet kap a felhasználó, ha a HTTPS csatorna használatára képes lenne ugyan, de a kiszolgáló által igényelt 128 bites titkosítást már nem tudja teljesíteni. A High Encryption Pack a Windows 2000 Service Pack 2-vel, illetve az Internet Explorer 5.01 SP1 verziójával kerül fel a számítógépekre.

Fülöp Miklós
mick@netacademia.net

ASP Suli (XI. rész)

Könyvjelzők és tárolt eljárások

E havi számunkban folytatjuk az előző hónapban megkezdett ADO objektummodell ismertetését. Eddig már eljutottunk az egyszerű lekérdezések eredményének lekérdezéséig, illetve az adatbázis tábláinak megnyitásáig, az adatok módosításáig – ebben a hónapban tovább részletezzük a Recordset objektum lehetőségeit, és bemutatjuk, hogyan érhetjük el az adatbázisban tárolt lekérdezéseket, tárolt eljárásokat. A példaprogramok az **[1]**, az ADO objektumok referenciája a **[2]** címen található.

Recordset megnyitása a Connection objektumon keresztül

Az előző részben egy mondatban utaltunk arra, hogy Recordsetet közvetlenül a Connection objektumból is meg lehet nyitni. Ehhez a Connection objektum .Execute*()* metódusát használhatjuk *(connex.asp)*:

```
Set oConn =
↳ Server.CreateObject("ADODB.Connection")
oConn.Open "File Name=C:\db\aspsuli.udl"

Set oRS = oConn.Execute("SELECT * FROM Products")
```

Az .Execute*()* metódus mindig csak olvasható, forward-only Recordset-et ad vissza, ezért ha ennél többet szeretnénk, a „hagyományos" módszerhez kell folyamodnunk. A metódus három paraméterrel rendelkezik, amelyből kettő elhagyható *(connex2.asp)*:

```
Set oRS = oConn.Execute("Products", , 2)
' 2 = adCmdTable
```

A fenti példa az adatbázisban található Products nevű táblát nyitja meg, a 2 értékkel jelezzük a providernek, hogy az első paraméter egy tábla neve lesz. A connex3.asp egy kicsit összetettebb és előremutató példa:

```
oConn.Execute "UpdatePrice1", nResults, 4 + 128
' 4 = adCmdStoredProc; 128 = adExecuteNoRecords
Response.Write nResults & " rekord módosult."
```

A példaprogram meghívja az adatbázisban található UpdatePrice1 tárolt eljárást *(a tárolt eljárások, lekérdezések működéséről részletesebben is lesz szó, most a hangsúly ezek meghívásán van)* – a harmadik paraméterként átadott 4-es érték ezt jelzi a providernek. A 128-as értéket *(adExecuteNoRecords)* akkor használjuk, ha a parancs futtatásakor nem várunk eredménytáblát *(például mint most: a tárolt eljárás tíz százalékkal megnöveli egyes termékek árait, de mi nem vagyunk kíváncsiak magára a listára)*. Ha nem kérünk eredménytáblát, akkor a feldolgozás is gyorsabb lesz *(természetesen ilyenkor elmarad a Set oRS = értékadás is)*.

A második paraméterről nem volt még szó: ide egy változót adhatunk meg, amibe a provider az eljárás futtatása után beírja, hogy az eljárás során hány rekord módosult. Ezt csak az úgynevezett „akció" eljárásoknál van értelme használni, amikor a végrehajtott parancs adatmódosulással jár.

Ne felejtsük el, hogy az adatmódosítással járó műveletekhez az aktuális felhasználónak (aki ASP kódok esetén az anonymous webfelhasználó) az adatbázisra, vagy annak érintett részeire írásjoggal kell rendelkeznie!

Könyvjelzők a Recordsetben

A könyvjelzők nagyon hasonlítanak a hagyományos könyvjelzőkhöz: egy-egy könyvjelző az aktuális Recordset valamelyik rekordjára mutat. Az aktuális rekordra mutató könyvjelzőt a Recordset .Bookmark jellemzőjéből olvashatjuk ki. Ennek a jellemzőnek értéket is adhatunk, ilyenkor a kurzor a könyvjelzővel jelölt rekordra lép *(ha ez lehetséges)*. A bookmarks.asp bemutatja a könyvjelzők használatát, néhány fontosabb sor a példából:

```
Set oRS = Server.CreateObject("ADODB.Recordset")
oRS.CursorLocation = 3 ' adUseClient
oRS.Open "Products", oConn, , , 2 'adCmdTable

vBookmark1 = oRS.Bookmark
oRS.MoveNext
oRS.MoveNext
vBookmark2 = oRS.Bookmark

oRS.Bookmark = vBookmark1
'vissza az első könyvjelzőhöz
```

Nem minden provider támogatja a könyvjelzőket, sőt, még a kurzor típusa és helye is számít *(az Access 2000 például csak kliensoldali kurzorral képes rá)*. Fontos, hogy minden könyvjelző csak abban a Recordsetben érvényes, ahonnan származik, az egyszerű könyvjelzők Recordsetek között nem vihetők át.

Szolgáltatások lekérdezése

Az, hogy egy adott provider, konkrétabban egy adott Recordset képes-e bizonyos dolgokra, egyszerűen lekérdezhetjük a Recordset .Support*()* metódusával. Az előbbi példából kiragadva *(bookmarks.asp)*:

```
If Not oRS.Supports( 8192 ) Then ' adBookmark
  Response.Write "Ez a Recordset nem támogatja a
↳ Bookmarkok használatát."
End If
```

A metódusnak paraméterként az egyes funkciók azonosítóit adjuk át *(ha több mindent szeretnénk egyszerre lekérdezni,*

az értékeket adjuk össze), amire a függvény igaz vagy hamis értékkel válaszol. Néhány funkció *(a teljes lista megtekinthető a* **[3]** *címen)*:

Konstans	Érték	Funkció
adAddNew	0x1000400 (16778240)	Új rekordok létrehozása (.AddNew())
adApproxPosition	0x4000 (16384)	Lapok használata (.AbsolutePosition és .AbsolutePage)
adBookmark	0x2000 (8192)	Könyvjelzők használata (.Bookmark)
adDelete	0x1000800 (16779264)	Rekordok törlése (.Delete())
adFind	0x80000 (524288)	Keresés (.Find())
adMovePrevious	0x200 (512)	A kurzor visszafelé is mozgatható (.MovePrevious())
adResync	0x20000 (131072)	Adatok szinkronizálása (.Resync())
adUpdate	0x1008000 (16809984)	A Recordset adatai frissíthetők (.Update())

A rekordok szűrése

Ha már van egy Recordsetünk, annak rekordjait tetszés szerint szűrhetjük is. Így egyetlen lekérdezés eredményét többféle nézetben is felhasználhatjuk, ha a kedvünk éppen úgy tartja. A szűréshez a Recordset objektum .Filter jellemzőjének kell egy szöveges feltételt megadni. A feltétel formátuma hasonlít a SELECT SQL utasítás WHERE után következő részéhez *(filter.asp)*:

```
oRS.Filter = "UnitPrice < 30 AND
ProductName LIKE '%ch%'"
```

A feltételben a mezőnevet egy műveleti jel, majd azt egy érték követi, az egyes feltételek között használhatók az AND *(és)* és az OR *(vagy)* operátorok és persze a zárójelek is. A műveleti jel lehet <, >, <=, >=, <>, =, vagy LIKE; ez utóbbinál egy szöveges értéket kell megadnunk, amiben használhatjuk a * vagy a % joker karaktereket. A dátumértéket # jelek közé írva adhatjuk meg. *(A példa a harminc-nál olcsóbb olyan termékekre vonatkozik, amelyek neve tartalmazza a „ch" karaktersorozatot.)* A szűrést kikapcsolhatjuk, ha a .Filter jellemző értékének üres stringet adunk meg, vagy ha az adFilterNone konstant *(értéke 0)* használjuk. Az alábbi sorok bármelyike kikapcsolja tehát a szűrést:

```
oRS.Filter = ""
oRS.Filter = 0 ' adFilterNone
```

A .Filter jellemző értékeként nemcsak szöveges értéket, hanem – mint láthattuk – konstanst is átadhatunk. A használható konstansokat és értéküket lásd a **[4]** címen. Végül: a jellemző értékének átadhatunk előzőleg feltöltött könyvjelzőkből álló füzért is:

```
oRS.Filter = Array( vBookmark1, vBookmark2,
vBookmark3) ' Előzőleg definiált könyvjelzők
```

Keresés

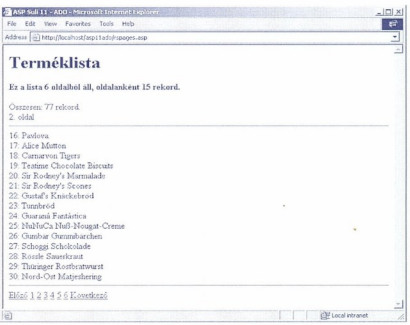
A szűrés mellett kereshetünk is a Recordset-ben, ehhez a .Find() metódust használhatjuk. A metódus négy paramétere a következő:

- Feltétel – a szűrésben használatoshoz hasonló feltétel, de itt csak egy mezőt ellenőrizhetünk
- Átugrandó sorok száma – számérték, amely azt jelzi, hogy a keresést az aktuális, vagy indulóként beállított pozíciótól számított hányadik sornál kell kezdeni, ha nem adjuk meg, értéke 0
- A keresés iránya – kereshetünk a Recordsetben előre *(adSearchForward (1), alapértelmezés), vagy vissza fele (AdSearchBackward (-1))*
- A keresés kezdőpontja – itt egy könyvjelzőt adhatunk meg; ekkor a keresés a könyvjelző által mutatott rekordtól kezdődik.

Ha a keresés sikeres volt, a kurzor a talált rekordra áll. Ha a keresés nem eredményezett találatot, akkor a keresés irányától függően az .EOF vagy .BOF jellemző értéke igaz lesz *(azaz a kurzor „lefut" a Recordset-ről)*.

Logikai lapok használata

A Recordseteket logikailag lapokra bonthatjuk *(persze csak azokat, amelyek ezt támogatják)*. A lapokra bontás tulajdonképpen nem jelent semmi lényeges változást, mindössze a Recordsetben található adatok feldolgozását könnyíti meg *(főleg webes környezetben, amikor nem listázhatjuk ki a felhasználónak a Recordset teljes tartalmát)*. A logikai lapok méretét a Recordset objektum .PageSize jellemzője adja meg, alapértelmezése 10. A kurzort továbbra is a megszokott módon mozgathatjuk, annak kezelésénél nem kell a lapokat figyelembe venni, azonban az .AbsolutePage jellemző értéke mindig azt adja vissza, hogy a kurzor által mutatott rekord hányadik lapon található *(megintcsak logikailag)*. Ha az .AbsolutePage értékét módosítjuk, a kurzor a beállított oldal első rekordjára áll. *(Az oldalak számozása 1-től kezdődik.)* A .PageCount jellemző értéke pedig azt jelzi, hogy összesen hány logikai lap található a Recordsetben. Végül: az .AbsolutePosition értéke mutatja, hogy az aktuális rekord hányadik helyen áll a Recordsetben *(függetlenül a lapok méretétől és számától)*. Kicsit sűrű volt? Próbáljuk ki *(rspages.asp)*!



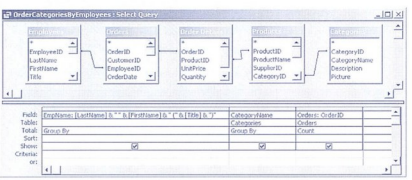
A terméklista, oldalakra bontva

A teljes kódot hosszúsága miatt itt nem mutatjuk meg, de a letölthető változatot gazdagon elláttuk megjegyzésekkel. A lényeg, hogy a Recordsetet csak a legelső látogatáskor hozzuk létre, és tároljuk azt a felhasználó Session objektumában. Ezután minden egyes látogatáskor a már meglévő Recordsetet vesszük elő; abból mindig csak .PageSize darab rekordot listázunk. Végül, az oldal alján található navigációs sor segítségével lehetővé tesszük a közvetlen mozgást az oldalak között *(ha a kód ilyen hivatkozást kap, automatikusan az adott oldalra lépteti a Recordsetet)*. A lapok használatához azonban erre nincs feltétlenül szükség, maga a kurzormozgatás is elég *(mi sem bizonyítja ezt jobban, minthogy az oldal egyszerű frissítgetésével végiglépkedhetünk az oldalakon – hiszen a kurzor a listázás során mindig előre lép egy-egy oldalnyit)*.

Tárolt eljárások

Az adatbáziskezelők *(még az Access is)* általában jóval többet tudnak, mint táblák megnyitását, módosítását, „ad hoc" parancsok, lekérdezések futtatását. A tárolt eljárások *(Access esetén tárolt lekérdezéseknek nevezik)* általában összetettebb, bonyolultabb feladatok végrehajtására is képesek, ráadásul lényegesen gyorsabbak, mintha ugyanaz*(oka)*t a művelet*(ek)*et egyenként, „kívülről" hajtanánk végre. A komolyabb adatbáziskezelők előre felkészülnek a létező tárolt eljárások használatára, futtatási terveket készítenek, esetleg gyorsítótárban megőrzik az eredményeket. Mindemellett a tárolt eljárások használata lényegesen kényelmesebb is, hiszen egy tetszőleges bonyolultságú folyamat elindítása részünkről egyetlen eljárás meghívásában merül ki.

A példák során az aspsuli.mdb tárolt lekérdezéseit fogjuk kezelni, de a módszer hasonló lenne akkor is, ha mondjuk SQL Servert használnák.



Egy összetettebb lekérdezés az Access-ben

A fenti ábrán látható lekérdezés *(OrderCategoriesByEmployees)* eredménye egy táblázat, amelyben az alkalmazottanként és árukategóriánként eladott termékek darabszáma szerepel. A lekérdezést meghívni azonban semmivel sem nehezebb, mint egy egyszerű táblát megnyitni *(query1.asp)*:

```
Set oRS = Server.CreateObject("ADODB.Recordset")
oRS.Open "OrderCategoriesByEmployees",
  oConn, , , 4 'adCmdStoredProc

While Not oRS.EOF
  Response.Write oRS("EmpName") & " / "
  Response.Write oRS("CategoryName") & " = "
  Response.Write oRS("Orders") & "<BR>"
  oRS.MoveNext
WEnd
```

Az egyetlen különbség, hogy a Recordset megnyitásakor az utolsó paraméterben jelezzük, hogy a célpont most egy tárolt eljárás *(adCmdStoredProc)*. A lekérdezés egy táblát ad vissza, a tábla mezőinek nevét pedig maga a lekérdezés definiálja *(innen az EmpName vagy az Orders mezőnév)*.

Az eljárást most a Recordseten keresztül nyitottuk meg, de van rá más, közvetlenebb és egyben rugalmasabb mód is, ez pedig a Command objektum használata. A Command objektum kifejezetten arra készült, hogy egy SQL parancsot, vagy tárolt eljárást jelképezzen és futtasson. Lássuk a fenti példát a Command objektummal megvalósítva *(query2.asp)*:

```
Set oCmd = Server.CreateObject("ADODB.Command")
oCmd.ActiveConnection = oConn
oCmd.CommandText = "OrderCategoriesByEmployees"
oCmd.CommandType = 4 ' adCmdStoredProc
Set oRS = oCmd.Execute
```

Első lépésként létrehozzuk az objektumot, majd az .ActiveConnection jellemzőnek átadjuk a korábban már megnyitott Connection objektumot. Ezután a .CommandText jellemző beállítása következik *(ez is lehet SQL parancs, tábla vagy eljárás neve)*, majd a .CommandType paraméter – szokás szerint – jelzi, hogy a parancs milyen típusú. Ha ezeket a paramétereket beállítottuk, nincs más hátra, mint meghívni az .Execute*()* metódust, ami egyébként pontosan ugyanazokat a paramétereket várja, amelyeket a Connection objektum .Execute() metódusánál átadhattunk. Ha megnézzük, hogy a query2.asp Command objektuma milyen Recordset-et adott vissza, láthatjuk, hogy kiszolgálóoldali, forward-only, csak olvasható kurzort kaptunk. Ez nem feltétlenül használható minden esetben – akkor viszont mire jó a Command objektum?

Command objektum, mint a Recordset forrása

Mit tehetünk tehát akkor, ha saját szánk íze szerint akarjuk formálni a Recordset típusát, de ugyanakkor nem akarunk lemondani a Command objektum – igazán csak később ismertetett – előnyeiről? Nos szerencsére a Recordset adatforrása, a .Source jellemző nemcsak szöveges értéket, hanem egy előkészített Command objektumra vonatkozó hivatkozást is elfogad. Ugyanez igaz a Recordset objektum .Open() metódusának első paraméterére is (query3.asp).

A példában először létrehozzuk a Connection, majd a Command objektumot. A Command objektumban beállítjuk a parancs adatait, valamint az aktív adatbáziskapcsolatot a .ActiveConnection jellemző segítségével *(ha a Recordset objektum forrásának Command objektumot adunk meg, az .ActiveConnection jellemzőt mindig csak a Command objektumnál kell kitölteni)*. Miután ez megvan, létrehozunk egy új Recordset objektumot, beállítjuk a kurzorjellemzőket és a zárolási stratégiát, majd beállítjuk a forrást *(ne felejtsük el a Set parancsot, hiszen objektumreferenciával dolgozunk!)*. Ezután már csak az .Open() metódus meghívása marad, és máris megy minden, mint a karikacsapás!

„Akció"-lekérdezés

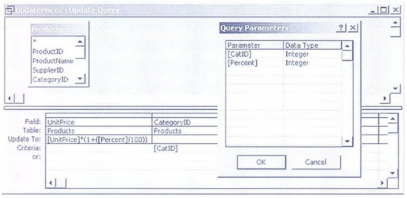
Korábban már említettük, hogy akció-lekérdezésnek nevezzük az olyan parancsokat, tárolt eljárásokat, amelyek nem adnak vissza eredménytáblát, csak csendben elvégeznek egy adott feladatot *(esetleg azt tudhatjuk, hogy hány rekord*

esett „áldozatul" a műveletnek, de semmi több). A példaadatbázisban található UpdatePrice1 lekérdezés az 1-es kategóriájú termékek árát növeli 10%-kal *(a cikk elején már bemutattuk).* Most lássuk, hogyan hívhatjuk meg ugyanezt a tárolt lekérdezést a Command objektum segítségével *(query4.asp)*:

```
Set oCmd = Server.CreateObject("ADODB.Command")
oCmd.ActiveConnection = oConn
oCmd.CommandText = "UpdatePrice1"
oCmd.Execute nResults, , 4 + 128
  'adCmdStoredProc + adExecuteNoRecords
Response.Write nResults & " rekord módosult."
```

Paraméterezett lekérdezések

A fenti tárolt lekérdezés mindig egy adott termékcsoport árait növeli, egy adott százalékkal. Az tárolt eljárásoknak, lekérdezéseknek azonban bemenő paramétereket is adhatunk, amelyeket az eljáráson belül felhasználhatunk. Így készült a példaadatbázis UpdatePrice2 lekérdezése:



Paraméterezett lekérdezés

A lekérdezés két paramétert vár, az egyik a kategória azonosítója, a másik pedig a százalékérték. A kérdés már csak az, hogy hogyan adhatjuk át a paramétereket a tárolt eljárásnak? A válasz az objektummodellben rejlik: a Command objektumhoz csatolt Parameter objektumokkal. Ezeket a Command objektum Parameters kollekciója tartalmazza, a kollekció a következő jellemzőket és metódusokat tartalmazza:

- .Count: a paraméterek száma
- .Item: egy adott paraméter
- .Append(): paraméter hozzáadása
- .Delete(): paraméter törlése
- .Refresh(): paraméterek frissítése

Az egyes paraméterek jellemzője a következő

- Név: a paraméter neve, ezzel hivatkozunk rá
- Típus: adattípus, az adattípus-konstansok értékét lásd az **[5]** címen
- Irány: egy paraméter lehet bemenő *(Input)*, kimenő *(Output)*, illetve visszatérési érték *(Return Value)*, ez általában magán a tárolt eljáráson múlik. A paraméter irányát az alábbi táblázatban szereplő konstansok segítségével határozhatjuk meg:

Konstans	Érték	Típus
adParamInput	1	Bemenő *(alapértelmezett)*
adParamInputOutput	3	Bemenő és kimenő
adParamOutput	2	Kimenő
adParamReturnValue	4	Visszatérési érték
adParamUnknown	0	Ismeretlen irány

A Command objektum Parameters kollekciójához az objektum .CreateParameter() metódusával előzőleg létrehozott Parameter objektumokat adjuk hozzá. A .CreateParameter() metódus paraméterei az alábbiak:

- Name: a paraméter neve
- Type: a paraméter típusa
- Direction: a paraméter iránya
- Size: a paraméterváltozó mérete *(a típusból adódik, nem kötelező megadni)*
- Value: a paraméter értéke *(később is megadható)*.

Az így létrehozott paraméterobjektumot átadjuk a Parameters kollekció .Append() metódusának. A paramétert ezután a Parameters kollekció .Item jellemzője segítségével érhetjük el *(név szerint)*.

A fenti példában szereplő lekérdezés két paraméterét tehát így címezhetjük meg *(paramq.asp)*:

```
Set oCmd = Server.CreateObject("ADODB.Command")
oCmd.ActiveConnection = oConn
oCmd.CommandText = "UpdatePrice2"
' Paraméterobjektumok létrehozása
Set oParam = oCmd.CreateParameter("CatID", 3, 1)
oCmd.Parameters.Append oParam
Set oParam = oCmd.CreateParameter("Percent", 3, 1)
oCmd.Parameters.Append oParam
' Értékadás
oCmd.Parameters.Item("CatID") =
  CInt(Request("category"))
oCmd.Parameters.Item("Percent") =
  CInt(Request("percent"))
```

A Parameters kollekció frissítése

Ha a provider azt támogatja, nem kell magunknak létrehozni a paraméterobjektumokat. Elég, ha a Command objektumnak beállítjuk, hogy mely tárolt eljárást szeretnénk meghívni, majd meghívjuk a Parameters objektum .Refresh() metódusát. Ez feltölti a Parameters objektumot a megfelelő paraméterekkel. Ezt a szolgáltatást az Access providere sajnos nem támogatja, de a listparams.asp példaprogramot kipróbálhatjuk, ha van egy SQL Server a közelben.

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:
[1] http://technet.netacademia.net/download/asp/11
[2] http://msdn.microsoft.com/library/en-us/ado270/htm/mdmscadoobjects.asp
[3] http://msdn.microsoft.com/library/en-us/ado270/htm/mdcstcursoroptionenum.asp
[4] http://msdn.microsoft.com/library/en-us/ado270/htm/mdcstfiltergroupenum.asp
[5] http://msdn.microsoft.com/library/en-us/ado270/htm/mdcstdatatypeenum.asp

SQL (XI. rész) A Query Optimizer

Talán már Magyarországon is lassan elérkezik a Clipper Summer '87 alapú alkalmazások hanyatlása, és előbb-utóbb minden adatbázis oda kerül, ahova való: valódi adatbázis-motoros, SQL alapú kiszolgálóra. Az átállás fájdalommal jár: egyszerre „vész el" a megszokott .DBF formátum, a szép DOS-os, karakteres alkalmazás és a szekvenciális fájlkezelés lehetősége. Ekkora traumát nehéz komoly lelki sérülés nélkül elviselni, s ennek tetejébe jön még a COM és az ADO (ActiveX Database Objects, lásd ASP Sulinkat). Nem csoda, ha sok programozó örül, ha az alap SQL utasítást összeeszkábálja...

```
SELECT * FROM VEVOK
```

...és felkínlódja egy VisBas formra. Semmi filter, egyszerűbb végigszaladni a rekordokon, és csak a kívánatos elemeket belepakolni abba az átkozott kombó boxba. Huh. Ezzel megvolnánk. Az alkalmazás elindul. Ám érthetetlen okokból fokozatosan és folyamatosan lassul. Fél évvel az átadás után már több, mint 11 másodpercbe telik az egykor oly fürge űrlap megjelenése. Pár hét múlva 15 másodperc. Itt tenni kell valamit! Vajon mi hiányzik? Nyilván egy index. Valahogy meg kellene adni a lekérdezésben, mint anno a Clipperben. Az hogy vágtatott ehhez a vacak SQL Serverhez képest! Egy hét alatt kiókumláljuk, hogy így kell odairni neki...

```
SELECT * FROM VEVOK (Index=VevoIdIndex)
```

...és ezzel elkövettünk mindent annak érdekében, hogy ez az alkalmazás a büdös életben ne futhasson optimális teljesítménnyel. Nem elég, hogy egy korlátozás nélküli *(unrestricted)* lekérdezést adtunk neki *(melynek elvégzéséhez egyszerűen nem kell index)*, hanem még ráerőszakoltunk egy alkalmatlan indeket is. Még jó, ha clustered, akkor legalább nem biztos, hogy lassít. De ha ne adj' Isten nonclustered, az eredeti 15 másodperc felfut 243-re. Hinnye. Mi van itt?? Vagy inkább ki?
<off>
Elnézést kérek a guruktól, akik szerint ilyen nincs. Hát én éppen két hónapja *(Krisztus után 2001, október havában)* menekültem el sikoltozva egy hasonló szörmedvény láttán. Szakérteni híttak, mert az a „tetves" SQL Server állandóan eldobálta a kapcsolatokat. Mondom nekik telefonba: deadlock. Nem értik. Na jó, odamegyek. Életem egyik legszörnyűbb optimizer hintekkel telehajigált alkalmazásához volt szerencsém. A programozó írt egy szekérderéknyi tárolt eljárást *(gondolom, hogy gyorsabb legyen az app)*, de egytől egyig mindet tönkretette bevarrott optimizer hintekkel. Akkor fogtam menekülőre a dolgot, amikor ezt megláttam:

```
SELECT * FROM VEVOK (Index=0)
    WHERE Name='Kovács'
```

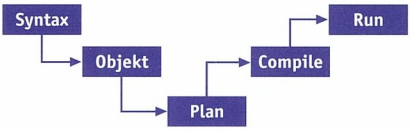
Itten paradigmaváltásra lenne szüség. Ebben a hazában. Jogosítvány nélkül nem vezethetsz, de elemi ismeretek nélkül, pusztán egy halom babonával felfegyverkezve jó pénzért programozhatsz. Piha!
</off>

Mér? Mi a baj?

Hát, eddig minden hibás, amit összegyűjtöttem:
1. **A SELECT * FROM VEVOK** lekérdezés visszaadja a tábla összes sorát és összes oszlopát, ergo indexet használni merőben felesleges. Akinek az egész tábla kell, annak az egész kell, punktum.
2. **A SELECT * FROM VEVOK *(Index=VevoIdIndex)*** egy indexszel jottányit sem gyorsítható lekérdezésre ráerőszakol egy indexet. Ha ez nonclustered, a futási idő több százszorosára nőhet! Mindezt úgy, hogy az SQL Server tudván tudja, hogy butaságot kérünk tőle.
3. **A SELECT * FROM VEVOK *(Index=0)* WHERE Name='Kovács'** sziporka pedig azt mondja az SQL Servernek, hogy még ha lenne is jó index, akkor se használja!
A legfőbb baj, minden teljesítményprobléma okozója azonban az a programozó, aki nem tudja, hogy egy nálánál intelligensebb teremtménnyel kommunikál, amikor SQL lekérdezéseket ír. Ő a Query Optimizer, akinek az a feladata, hogy a felhasználói lekérdezéseket a lehető legrövidebb idő alatt hajtsa végre. Kiváló munkát végez! Hagyjuk dolgozni!

Az optimalizálás menete

Valahányszor az SQL Server lekérdezésfuttatója új SQL parancsot kap - kevés kivételtől eltekintve - ugyanazon a lépéssorozaton megy végig, mire futtatásra kerülne a sor:



Az SQL lekérdezések feldolgozásának menete

1. Szintaktikai ellenőrzés. Ebben a lépésben az dől el, hogy a parancs megfelel-e a Transact SQL nyelv szintaktikájának. Hogy SELECT után egy értéket produkáló kifejezés van-e, EXEC után tárolt eljárás stb. Ha ezen a teszten átcsúszik a lekérdezés, jöhet a következő lépés:
2. Objektumhivatkozások feloldása. Vajon léteznek-e a hivatkozott táblák és mezők és függvények és változók és ...? Ha igen, ez a lépés összegyűjti az idevágó metaadatokat is *(mezők típusa stb.)* Amint ez megvolt, következhet a tuning:
3. Optimalizálás. Ebben a fázisban többek között a hivatkozott táblák összekapcsolása *(dzsoin)* és a megfelelő indexek kiválasztása történik meg. *(Hangsúlyozom: a megfelelő!!!)*

Ennek a lépésnek az eredménye az úgynevezett Query Plan, vagy végrehajtási terv, ami elmenthető a procedúra gyorsítótárba *(kess)*. Ezután jön a:

4. Fordítás. Pszeudokódra, mert az hordozható. Az SQL Servernek ugyanis nemcsak Inteles verziója van. Végül a pszeudót is fel kell dolgozni, ez lesz az utolsó lépés:

5. Futtatás. Fuss Forrest, Fuss!

Jelen dolgozat szempontjából a harmadik lépés a legfontosabb, feltételezzük, hogy létező objektumokra hivatkozó szintaxhelyes SQL utasításokat bárki képes készíteni. Először foglalkozzunk a „jó index" szindrómával. Ehhez ismernünk kell a Query Optimizer észjárását: vajon mi alapján dönti el, hogy hogyan érdemes lefuttatni egy lekérdezést? Költségalapon! Több végrehajtási módszert *(indexszel, anélkül stb.)* is mérlegel, majd ezek közül a legolcsóbbat választja. Már csak a „pénznemet" kell elárulnom: fizikai I/O. Amelyik lekérdezési stratégia a legkevesebb lemezrángatással jár, az a legolcsóbb, hisz az egész számítógép leglassabb eszköze mind a mai napig a mechanikus, forgótáras vincseszter. Ha egészen pontosak óhajtunk lenni: súlyos teljesítményköltséggel jár a 8 kilobájtos lapok *(a helyfoglalás alapegysége az SQL Serverben)* megmozdítása. Ergo a Query Optimizer arra törekszik, hogy a lehető legkevesebb 8k-s lapot kelljen megmozdítania. Vagyis lusta. De a lustaság, lám, fél egészség, hisz ez a stratégia adja egyben a leggyorsabb végrehajtást is!

Költséghatékonyság

Az Optimizer a következő módon választja ki a legolcsóbb végrehajtást: kiszámítja az összes lehetséges végrehajtási változat „árát", s a legolcsóbb mellett dönt. Konkrét példán könnyebb bemutatni a folyamatot; a VEVOK táblán dolgozzunk! *(Ez egy fiktív tábla. Képzeljük egyszerűen magunk elé.)* A VEVOK táblában mondjuk tízezer rekord van *(megy a biznisz)*, az átlagos rekordhossz 152 bájt. Így egy 8k-s lapra mintegy 52 rekord fér el, a tábla pedig 193 darab 8k-s lapon terül el *(52 * 193 = 10036)*. Vegyük ismét a bevezető-dühöngőben már említett lekérdezést, de ezúttal optimizer hint nélkül:

```
SELECT * FROM VEVOK
    WHERE Name='Kovács'
```

És most tételezzük fel, hogy a VEVOK táblán van egy clustered index a Name, és egy nonclustered index a Name, Street mezőpároson. Ha véletlenül mégsincs, gyorsan hozzuk létre *(fejben)*!

```
CREATE CLUSTERED INDEX NameIndex on VEVOK(Name)
```

És...

```
CREATE NONCLUSTERED INDEX NameStreetIndex
 on VEVOK(Name, Street)
```

Már ehhez az egyszerű lekérdezéshez is három különböző stratégia választható

- index nélküli futtatás
- a clustered index használata *(a rekordok fizikai sorrendje az indexkulcsnak megfelelő)*
- a nonclustered index használata *(hagyományos, pointeres index)*

Az **index nélküli** futtatás költsége számítható a legegyszerűbben: az összes 8k-s lapon végig kell szánkáznunk a Kovácsok megtalálásához *(e stratégia neve Table Scan)*. Ez annyit tesz, mint 193 peták *(=193 darab 8k-s lap)*.

A **clustered index** árának kiszámítása már bonyolultabb, de még mindig könnyen megérthető. Mivel a clustered index fizikailag lerendezi az adatokat az indexkulcs alapján, a VEVOK tábla rekordjai ideális fizikai sorrendben, Name szerint sorban terülnek el a lemezen. Így a Kovácsok megtalálásához egyszerűen rákeresünk az első Kovácsra *(ez eddig kb. 2-3 peták, az index szintjeinek bejárása)*, majd szép sorban addig olvassuk a - helyes sorrendben lévő - rekordokat, amíg túlfutunk az utolsó Kovácson. Ez még akkor sem több, mint 19 peták *(lap)*, ha a vevők tíz százaléka hallgat Kovács névre. Összesen tehát maximum 3+19, de ha szerencsénk van akkor inkább 2+5 vagy valami hasonló. A legnehezebb a helyzet a **nonclustered**, azaz hagyományos indexszel, lássuk miért! Ebben az esetben az index csak rávezet minket a megfelelő rekordokra, amelyek valamilyen - akár kaotikus - sorrendben helyezkednek el a táblában. Noha a mi gondolatkísérletünkben a fizikai sorrend véletlenül kedvező *(mert pont a Name mezőn van egy clustered index is)*, ezt az Optimizer nem veszi figyelembe, mivel normális esetben a helyzet nem ilyen rózsás. Elvégre csak egy őrült pakol ugyanarra a mezőre egyszerre egy clustered és egy nonclustered indexet is :) A kaotikus elrendezés megvilágítására álljon itt egy példa, a telefonkönyv. Azon a clustered index *(a fizikai sorrend)* vezetéknév+keresztnév szerinti. De milyen sorrendben vannak benne a keresztnevek? Hol vannak a telefonkönyvben a Dzsonik és Dzsokik? Összevissza ugye? No most gondoljuk végig, ha van egy indexünk a keresztnevek alapján, mely megmutatja melyik lapokon vannak Dzsokik, kábé hány oldalt kell megfognunk? Ez sokmindentől függ, és nem csak a Dzsokik számától, hanem azok eloszlásától is. Ha történetesen a káosz úgy hozta, hogy egy-két telefonkönyvoldalon nyüzsög a Dzsokik túlnyomó többsége, szerencsénk van. Ennél azonban valószínűbb a Dzsokik egyenletes eloszlása, amikor is annyi lapot kell érintenünk, ahány Dzsokink van. Az SQL Server is így kalkulál: a lehető legrosszabb esetre felkészülve annyi petákot számol fel a nonclustered index használatáért, ahány találatunk lesz. De hagyjuk a Dzsokikat, térjünk vissza a mérendő lekérdezésre. Ha az előző pont feltételezésével, azaz tízszázaléknyi Kováccsal számolunk *(ami igazán magas arány)*, akkor a nonclustered index bizony ezer *(!!!)* petákba kerül, ami nyolcszor annyi, mint a table scan!

Persze ha kevés Kovácsunk van, mondjuk egyetlen egy, akkor indexfa bejárása+1 peták az ár. A nonclustered index ára, hasznossága tehát alapvetően függ a találati aránytól.

Az index-ráerőszakoló optimizer hintek kizárólag tesztelési célból használandók, hisz lehet, hogy egy adott index ugyanazon a táblán gyorsítja a lekérdezést egy ritka mezőérték keresése esetében, de csúfosan lelassít(hat)ja ugyanazt, ha egy másik érték szerint keresünk. Ez itt kéremszépen a Clipper halála. Csak az intelligens adatmotorok képesek figyelembe venni az adatok eloszlását a lekérdezés futtatásakor!

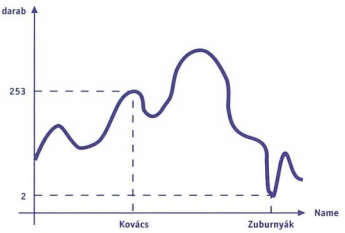
Így hát ezt előre kell tudnia a Query Optimizernek. No de honnan? Ha azt írom

```
WHERE A=75
```

akkor vajon hány rekordot kapunk vissza? Egyet? Meglehet. Nullát? Hát, az is lehet. Ezeregyet? Noné, az is lehet! Jó lenne ide valami tudományosabb dolog...

Indexstatisztika

A Query Optimizer nem hasraütéses módon próbálja előre megbecsülni, hogy egy adott lekérdezés hány rekordot fog érinteni, hisz mint láttuk, a jóslás pontossága döntő fontosságú az indexek hasznosságának eldöntésekor. Van neki ugyanis egy segédeszköze, az úgynevezett indexstatisztika. Ez megmondja, hogy ha Kovácsot keresünk, akkor nagy valószínűséggel mondjuk 253 találatunk lesz *(és a nonclustered index használata lassabb, mint a Table Scan, mert 253>192)*, ha pedig Zubornyákot, akkor 2 rekordot kapunk, így ugyanaz az index most sokkal-sokkal olcsóbb. Az alábbi ábra egy indexstatisztika idealizált képét mutatja, benne az előző két példával. Természetesen a statisztika nem grafikus formában tárolódik, megtekintése igazi kihívás *(DBCC SHOWSTATISTICS)*, amivel most nem fárasztanám a tisztelt olvasóközönséget. Majd MesterQ-n meghekkeljük.



Ilyen lenne az indexstatisztika diagramja, ha ábrázolnánk

A statisztika frissen tartása kulcsfontosságú. Ha ugyanis tartalma eltér a valóságtól, a Query Optimizer megtéved, és rosszkor mond ámént az indexekre. Ennek következménye katasztrofális teljesítményben mutatkozik meg. Az adatbázisokon gyárilag be van pipálva az „Auto Create Statistics" és az „Auto Update Statistics". Listázzuk ki, hogy az indexeken be van-e pipálva az autostat:

```
sp_autostats vevok
```

De azért csak hagyjuk úgy a pipákat, ha jót akarunk! Ha ennek ellenére gyanús tévedéseket vélünk felfedezni, használjuk az

```
UPDATE STATISTICS VEVOK
```

parancsot, mely végigszalad a táblán, és felfrissíti az indexstatisztikát *(jó sok opciója van ennek a parancsnak, érdemes végigolvasni az Online Bookban)*. Az sp_updatestats pedig az adatbázis összes táblájának összes indexén megcsinálja a statisztikafrissítést. És most térjünk a lényegre! Honnan a csudából tudható meg, hogy az SQL Server mit használ az egyes lekérdezések futtatásakor?

Query Plan

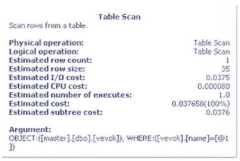
Az SQL parancsok fordításának harmadik lépése, a végrehajtási terv minden további nélkül megtekinthető a Query Analyzer eszközzel, ha a Query menüből a „Display Execution Plan" menüpontot választjuk - csak győzzük értelmezni! Márpedig értelmezni muszály, mert a „Designing..." vizsga tele van olyan kérdésekkel, hogy ha ezésez a Query Plan, akkor mit kellene tennünk, hogy a lekérdezés teljesítménye növekedjen. A problémát a planalkatrészek horribilis száma okozza. Üssük csak fel az Online Bookot a „Graphical Showplan" oldalon, és csodálkozzunk rá a rengeteg ikonkára. Hát nem bevehetetlen vár a mi SQL Serverünk? Ha egyszerre próbálnánk feldolgozni, megérteni valamennyit, nyilván belepusztulnánk, ezért a kis lépések stratégiáját javaslom. Kezdjük talán ezzel:

Table Scan

Mi sem egyszerűbb: a lekérdezés nem használt indexet. Hogy miért? Hát vagy azért, mert drága volt, vagy azért, mert egyáltalán nem volt. Mindenesetre ez az ikon irtandó, minél nagyobb a tábla, annál nagyobb baj, ha hemzseg tőlük az alkalmazás. Ellentipp: ha egy tábla oly kicsiny, hogy elfér 1-2 darab 8k-s lapon, akkor ne is erőlködjünk annak indexelésével, mert az 1 lapos Table Scan mindig olcsóbb lesz akármiféle indexhasználatnál! Nagy táblánál azonban illik indexelni a keresésben szereplő mezőket.

A Table Scan stratégia

Húzzuk csak a Table Scan ikonja fölé az egeret! Mit látunk? A Query Optimizer által kiszámított árakat!



A Query Optimizer számításai: sorok száma, költség, keresési feltétel stb.

A Table Scan ára 0.037658 peták. És most következzen egy ikerpár:

(Clustered vagy nonclustered) Index Scan és

(Clustered vagy nonclustered) Index Seek

Ez a két jómadár index használatára utal. A fenti, sokat emlegetett „kovácsos" lekérdezés így fut:

Az Index Seek alatt olvasható a használatba vett index neve, és „beszerzési" ára, százalékosan

A *(nonclusterd)* Index Seek ára: 0,006408 peták. Egy nagyságrenddel olcsóbb, mint a Table Scan. De ne örüljünk korán: az Index Scan *(amelyiknek dagi nyíl van az ikonjában)* becsapós. Ha ugyanis van egy táblán clustered index, többé sohasem látjuk az előző, Table Scan ikont, mert olyankor clustered index scan látszik *(mivel a fizikai sorrend megegyezik az indexsorrenddel)*, még akkor sem, ha valójában az történik. Kérdezzük ezt:

```
SELECT * FROM VEVOK
    WHERE street='sadsdf'
```

És tekintsük meg a Query Plant:

Az Index Scan néha Table Scan!

Itt bizony huncutság van, mert a street mezőn nincs is clusterd index, ez a Plan mégis Clustered Index Scanról beszél. Pedig ez biza Table Scan! Ára: 0.037658 peták, vagyis fillérre ugyanannyi, mint a Table Scan ára – és ez nem véletlen!

Lábas volt a fedőnevem

Az Index Scan nemcsak hazudós-becsapós tud lenni, hanem nagyon hasznos is. Index Scan esetén tulajdonképpen táblaként használjuk az indexet, vagyis holmi ódivatú, Clipperes fabejárás helyett nekiesünk a legalsó szintjének, mintha az tábla lenne *(az előbb láttuk: clustered esetén valóban a tábla van az index „alján")*, és szépen végigfutunk rajta. Hol vehetjük ennek hasznát? Netán tudatosan fel is használhatnánk? Emlékezzünk a nonclustered indexünkre, mely a *(Name, Street)* mezőkre készült. Most kérjük meg az SQL Servert, hogy listázza ki a vevőket utcanév, azon belül név szerint:

```
SELECT Street, Name FROM VEVOK
        ORDER BY Street, Name
```

Használni fogja vajon a pont fordított sorrendű indexünket? Gondoljunk ismét a telefonkönyvre *(Last, First clustered index)*. Ha keresztnév alapján kellene outputot produkálnunk, mire mennénk az indexszel? Semmire. Hacsak...

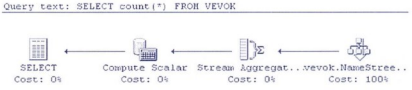
Vegyük észre, hogy az indexben benne van minden érték, ami alapján készült. Mint egy kicsi tábla. A mi nonclustered indexünkben is benne van az összes Name és Street, még ha nem is a kívánatos sorrendben. Ha nincs jó index, akkor ugyebár Table Scanra fanyalodunk. De ha a lekérdezés csak olyan mezőket kér, amelyek amúgy is megvannak az indexben? Sokkal gyorsabb az indexen Table Scanolni, mint a táblán! Sőt, ennek más előnye is van. Ha nem megyünk le a táblához az értékekért, akkor nem teszünk rá semmilyen zárat *(lockot)* sem! A fedő index *(mely egy lekérdezés összes mezőjét tartalmazza, mégha rossz sorrendben is)* egy kincs. Ügyes használatával sokszorannyi párhuzamos felhasználó dolgozhat adatbázisunkban!

Matek

Most pedig számoljuk meg, hány vevőnk van:

```
SELECT count(*) FROM VEVOK
```

Nem tudom, ki mire számított, én arra tippeltem volna, hogy emiatt nem megy végig sem táblán, sem indexen, hanem az egyik rendszertáblából megválaszolja. De nem. Parádés outputot produkál:



Hány vevő van?

Végigfut a táblán a clusterd index mentén, s közben csoportosítva számol *(GROUP BY: ez a Stream Aggregate)*, majd az egyetlen csoport eredményét aláveti egy „skalarizálásnak", amitől az egyébként is egy érték végre egy darab értékké válik, és kimehet az outputra. Úgy tűnik, egy kicsit elgaloppírozta magát. De ha megnézzük az ikonok alatt a költségeket, kiderül, hogy gyakorlatikag az index mentén történt végigfutás viszi el a költség egészét, így a fejben-feleslegesen-skalarizálás nem oszt, nem szoroz. Úgy deríthetjük ki pontosan, hogy mit bénázik egy ilyen egyszerű SELECT-tel, ha összehasonlítjuk ezzel a kovácsszámoló lekérdezéssel *(hány előfordulása van egy-egy névnek?)*.

```
SELECT Name, count(*) FROM VEVOK
        GROUP BY Name
```

A Query Plan ugyanis pontosan ugyanaz. Az Optimizer nem tesz különbséget a GROUP BY nélküli, és a GROUP BY-os skaláris összegzések között.

Hosszú történet a Query Optimizer, ígérem, visszatérünk rá. Legközelebb a zárolásokról mesélek, utána pedig – az októberi Hash! Cikkre építve – a kapcsolási *(join)* stratégiákat elemzem.

Fóti Marcell
MCDBA is.

XMLgessünk (VIII. rész) XML.NET I.

Bevezetés

Minden Microsoft fejlesztő számára egyértelmű: alapvetően meg fog változni az a háttér, amely támogatásával jelenleg Windows alkalmazásokat fejlesztünk. A radikális változás oka a .NET marketingnéven beharangozott technológiai csomag. A .NET egyik alapköve az XML. Keresztül-kasul XML-be botlunk a rendszerosztályok használata során is. Emiatt komoly támogatást várhatunk a szokásos XML-es feladatok elvégzéséhez is: értelmezés, XSLT transzformáció, érvényesítés *(validation)* séma alapján. Valójában ennél sokkalta többet kapunk. Lássuk!

Bemosakodás

Aki még nem találkozott a .NET-tel, annak talán szokatlan lesz a cikkben található programok nyelvezete. A kódokat C# *(ejtsd: szí sárp)* nyelven írtam, és a Beta2 állapotú .NET Framework-öt használtam fordításhoz és futtatáshoz. Mire ez a cikk az Ön kezébe ér már december lesz, amikor már lehet, hogy az Ön gépén a végleges verziójú .NET keretrendszer fog csücsülni. Ebben az esetben lehet, hogy a példa forráskódok nem fognak hibátlanul lefordulni, mert megváltozott valamelyik .NET osztály definíciója. Nagyon nagy változás nem várható, de probléma esetén *(és egyáltalán)* bátorítom a kedves olvasót az MSDN Library használatára. Ez elérhető a **[1]** címen is, de CD-n vagy DVD-n megrendelve sokkal kényelmesebben használható. Microsoftos fejlesztő meg sem tud mozdulni nélküle.

Attól, hogy valaki még nem ismeri a C# nyelvet nyugodtan nekiállhat a cikk olvasásának. A nyelv C++ *(JAVA)* hasonlósága miatt könnyen olvasható az előbbi nyelvek valamelyikének ismeretében. Kód írásakor már nem ilyen egyértelmű a dolog.

Aki pedig szeretné közelebbről is megismerni a .NET-et, annak ajánlom figyelmébe a jövő januártól induló .NET-ről szóló sorozatunkat, ahol a kezdetektől elindulva hónapról-hónapra fedezzük fel a .NET világát.

Az MSXML és a .NET

Cikksorozatunk eddigi részeiben az MSXML3, illetve néha az MSXML4 parser lehetőségeivel foglakoztunk. Ezek biztosítják számunkra az XML technológiák használatát COM alapon, többé-kevésbé szabványos API-kon keresztül. Az MSXML4 már a jelenlegi XML, XSD *(XML Schema Description)*, XSLT és XPath w3c ajánlásoknak megfelelően működik *(a 3.0 még nem, hisz nem volt még kész az XSD a programcsomag írásakor)*.

A .NET keretrendszer fejlesztése a legújabb XML technológiákkal párhuzamosan történik, így a termékben olyan új technológiák is benne vannak, mint például a SOAP. Emellett a .NET XML osztályok nemcsak egyszerűen az MSXML4 funkcionalitást tudják, hanem sok új eszközt is tartalmaznak, amelyek segítségével egyszerűbben vagy gyorsabban lehet a megszokott feladatokat elvégezni.

Ennek ellenére a .NET XML osztályok esetében szó sincs radikális váltásról az MSXML-hez képest, nem úgy mint az ADO –> <– ADO.NET esetén. Mivel a COM világ még sokáig el fog éldegélni, az MSXMLx és a .NET osztályok alapszolgáltatásai egymással többé-kevésbé párhuzamosan fognak fejlődni, egészen addig, míg egyszer csak a Microsoft úgy érzi, itt az ideje túllépni a COM-on. Ez akkor következik be, amikor a legtöbb XML-t alkalmazó Microsoft termék már .NET alapú lesz *(IE, Office, stb.)*. Mielőtt ez utolérne minket *(azért ez nem holnap lesz)*, érdemes megismerni a .NET XML osztályok szolgáltatásait. Nemcsak azért, hogy haladjunk a korral, hanem mert sokszor egyszerűbb lesz az életünk.

Az XML, mint a .NET alapköve

A COM világban az MSXML nem más, mint egy egyszerű COM DLL. Amelyik alkalmazás XML-lel akart foglakozni, az felhasználta a benne lakó objektumok szolgáltatásait. De ettől még a COM alapjainak semmi köze sem volt az XML-hez. Nem így a .NET.

A .NET központi technológiái, a webszolgáltatások, ASP.NET, Remoting *(~ távoli objektumok aktiválása és metódushívása)*, az alkalmazások konfigurációs állományai mind XML alapúak. De az igazán izgalmas lehetőség az adatelérést biztosító ADO.NET és az XML osztályok integrációja. Egy adatbázislekérdezés eredményét láthatom, mint memóriabeli objektumot, de ha kell, XML adatforrásként vagy adatfolyamként is kezelhetem. Az OLEDB által megvalósítani kívánt univerzális adatelérési stratégiát *(amely nem terjedt el széleskörűen, lévén, hogy COM alapú, így a Microsofthoz kötődik)* most az XML, mint mindenki által ismert adatreprezentáció révén az ADO.NET már sikerre viheti. A relációs és a hierarchikus világ közötti szakadékot valószínűleg sikerült áthidalni a szorosan összeintegrált adatelérési és XML kezelő osztályok segítségével. De egyelőre maradjunk az alapszolgáltatásoknál.

XML alaposztályok

Az alapszolgáltatásokat nyújtó osztályok a System.Xml névtérbe vannak csoportosítva. Fizikailag ők a System.Xml.dll-ben találhatók meg, mely a többi alapvető assembly-vel együtt a c:\winnt\microsoft.net\framework\v1.0.2914\ elérési úton található meg a Beta2 verzióban. Később is valószínűleg csak a verziószám változik meg.

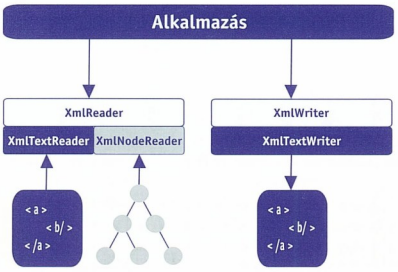
A legfontosabb alaptechnológiák külön névteret kaptak, így az XPath-szal kapcsolatos osztályok a System.Xml.XPath névtérben, az XSLT a System.Xml.Xsl-ben, a séma *(XSD)* osztályok a System.Xml.Schema, míg az objektumok állapotának XML formátumban történő tárolását és visszatöltését a System.Xml.Serialization névtér lakói látják el.

Az egyszerű használat érdekében érdemes beimportálni ezeket a névtereket a programunk elején a C# using kulcsszóval *(VB.NET: Imports)*. Ettől nem lesz nagyobb a kész programunk, maximum tovább tart a fordítás.

```
using System.Xml;
using System.Xml.Schema;
using System.Xml.XPath;
```

```
using System.Xml.Xsl;
using System.Xml.Serialization;
```

Az MSXML-től eltérően a .NET XML osztályok már eleve modularizáltan, a könnyű bővíthetőség jegyében vannak megtervezve, kihasználva a futtatókörnyezet, a Common Language Runtime *(CLR)* objektumorientált lehetőségeit.
Két absztrakt alaposztály alkotja az XML kezelés gerincét: XmlReader és XmlWriter. Az absztrakt szó azt jelenti, hogy belőlük nem lehet konkrét objektumpéldányokat létrehozni a new operátorral. A VB.NET-ben az absztrakt osztályokat a MustOverride *(kötelező felüldefiniálni)* kulcsszóval illetik, ami az abstract szónál kifejezőbb, habár az objektumorientált elméletben az absztrakt szó a megszokott. A .NET nyelvek és maga a háttér kódkönyvtár is nagyon támaszkodik az objektumorientált elmélet*re (öröklés, virtuális metódusok, interfészek, egységbezárás satöbbi)*, ezért objektumorientált technikák pontos ismerete nélkül a .NET gondolkodásmódja elég nehezen érthető. Fogjuk tudni használni *(kopi-paszta a példakódokból)*, de tudatosan csak az OOP ismeretek fényében érthető meg a „miért". Emiatt az OOP-vel egy külön cikk keretében fogunk még foglalkozni, illetve a **[2]** címen elérhető tanfolyamon meg lehet ismerni részleteiben a C#-pal együtt.
Miért nem lehet létrehozni példányt egy absztrakt osztályból? Azért, mert a legtöbb metódus nincs megvalósítva bennük, azt majd a belőlük örökléssel leszármaztatott gyermekosztályok implementálják. Azaz ki van jelölve, hogy van neki mondjuk egy ReadAttributeValue() metódusa, de nincs konkrét megvalósítása, azaz nincs programkód mögötte, mert a tényleges megvalósítást majd csak a leszármazott osztályok tudják megtenni, ők tudják mit kell csinálni egy adott metódusban.
Az XmlReader XML adatfolyamok egyirányú, nem módosítható *(read only, forward only)* feldolgozására szolgál. A párja, az XmlWriter az XML 1.0 és az XML Namespaces ajánlásoknak megfelelő XML adatfolyamot képes magából ontani. Az adatfolyam forrása illetve célja sokféle lehet: fájl, hálózati adatfolyam, memória, egy String, HTTP csatorna, ...), és ez is tetszés szerint bővíthető.
Azaz XML-t olvasni és feldolgozni kívánó alkalmazások az XmlReader-t, XML-t generáló programok pedig az XmlWriter-t fogják használni. Ezen absztrakt alaposztályok tényleges használatához a Microsoft többféle implementációt is írt. Az XmlReader-ből van származtatva az XmlTextReader, XmlValidatingReader és az XmlNodeReader. Az első általános feldolgozásra, a második sémával szemben ellenőrzött *(validation)* olvasásra, míg a harmadik XML DOM-ból képes adatokat olvasni.
Az XmlWriter-nek egy megvalósítása van, az XmlTextWriter.



Amennyiben olyan funkcionalitásra van szükségünk, amelyet nem fednek le ezek az osztályok, bármikor leszármaztathatunk saját osztályt a fenti alaposztályokból, és ezek után csak rajtunk áll, hogy honnan olvassuk vagy írjuk az információt. Pont ez egy hatalmas előnye az általános tervezésnek.

XMLReader

Tekintsük meg az XmlTextReader osztályt működés közben!

```
1 using System;
2 using System.Xml;
3
4 public class XMLTeszt
5 {
6   public static void Main()
7   {
8     XmlTextReader r;
9     r = new XmlTextReader("a.xml");
10     while (r.Read())
11     {
12       switch (r.NodeType)
13       {
14         case XmlNodeType.Element:
15         Console.WriteLine("Elem: "
16                           + r.Name);
17         while (r.MoveToNextAttribute())
18         {
19           Console.WriteLine(" " +
20           r.Name + "='" + r.Value + "'");
21         }
22         break;
23         case XmlNodeType.Text:
24          Console.WriteLine("Szöveg: "
25                            + r.Value);
26         break;
27       }
28     }
29   }
30 }
```

(A sorszámozás csak a könnyebb hivatkozás kedvéért van a példában. Ez nem Simon's basic. :)
Létrehozunk egy XmlTextReader példányt ⑨. A konstruktorban rögtön megadjuk az olvasandó XML fájl nevét, így rögtön készen állunk az XML adatfolyam olvasására. A Read() metódussal lépkedünk előre az adatfolyamban ⑩, amely mindaddig true-t ad vissza, míg el nem értük a forrás végét. Minden egyes olvasás után visszakapunk egy node-ot, aminek típusát a NodeType tulajdonság adja vissza. A fenti példában csak kettővel foglalkozunk, a többit egyszerűen átlépjük. A node fajták az XmlNodeType felsorolás *(enumeration)* típusban vannak deklarálva. Minden node-hoz tartozik rengeteg tulajdonság, amelyeket további jellemzőkön keresztül érhetünk el. A 15-16. sorban például kiíratjuk az élkapott elem nevét. A 24-25. sor egy elem szöveges tartalmát írja ki. Figyeljük meg, hogy az elemek attribútumai nem jelennek meg egyedi módon, hanem az őket tartalmazó elemnél lehet őket kiolvasni. Egy elemen állva a MoveToNextAttribute()-tal végiggyalogolhatunk az adott elem összes attribútumán. Ez összhangban van az XML specifikációval, miszerint az attri-

bútumok logikailag NEM gyermekei az őket tartalmazó elemnek, hanem csak annak kiegészítései, jellemzői.
Az egyes attribútumok értékét és nevét ugyanazon Value és Name jellemzőkkel érhetjük el, mint az elemekét. Ha belegondolunk, az adatbázisokhoz hasonlóan az XmlReader-ben is van egy kurzor, ami mindig az aktuális node-ra mutat. A 15. sorban egy elemre, a 20.-ban egy attribútumra, a 25.-ben egy elem szöveges tartalmára.
A Read() és a MoveToNextAttribute() is ezt a kurzort mozgatja, s közben kiolvashatjuk annak a node-nak a jellemzőit, amin pillanatnyilag áll.
A példát a

```
csc xmlreader.cs
```

paranccsal fordíthatjuk le.
Legyen a bemeneti fájl:

```
1. <?xml version="1.0" encoding="ISO-8859-2"?>
2. <NetAcademia>
3.   <soci labmeret="45">Soczó Zsolt</soci>
4.   <marci labmeret="49"></marci>
5. </NetAcademia>
```

Ekkor a futtatás eredménye:

```
Elem: NetAcademia
Elem: soci
 labmeret='45'
Szöveg: Soczó Zsolt
Elem: marci
 labmeret='49'
```

Látható, hogy sorban olvassa fel az XML fájlt, és ahogy elér egy-egy node-ot úgy visszatér a Read(), mi pedig kiíratjuk a kapott tartalmat.
Olvasás közben számtalan hiba felléphet, leggyakrabban formátumbeli hibákra számíthatunk, azaz az olvasandó XML tartalom nem jól formázott, vagy egyszerűen nem olvasható a forrásállomány. Ilyenkor az XmlReader dob egy kizárást *(exception)*, melynek konkrét típusa XmlException. Erre fel kell készülnünk, azaz el kell kapnunk, és le kell kezelnünk.
Az előbbi példára alkalmazva ezt a Main() metódus teljes tartalmát rakjuk bele egy try blokkba, és írjuk egy catch blokkot, ami lekezeli a kizárást:

```
try
{
  //Ami a Main()-ben volt.
}
catch(XmlException ex)
{
 Console.WriteLine("Gáz van a " + ex.LineNumber
       + ". sorban, a "
       + ex.LinePosition + " karakternél.");
 Console.WriteLine("A hiba oka: " + ex.Message);
}
```

Ha például nincs meg a megadott XML fájl:

```
Gáz van a 0. sorban, a 0 karakternél.
A hiba oka: There was an error accessing
file:///C:/Documents.../a.xml
```

Ha rosszul formázott a bemenet, például kitörlöm a soci elem záró tagját:

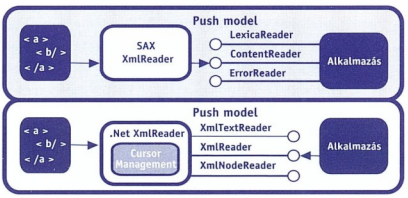
```
Elem: NetAcademia
Elem: soci
 labmeret='45'
Szöveg: Soczó Zsolt

Elem: marci
 labmeret='49'
Gáz van a 5. sorban, a 3 karakternél.
A hiba oka: The 'soci' start tag on line '3'
doesn't match the end tag of 'NetAc
ademia' in file 'file:///C:/Documents... /a.xml'.
Line 5, position 3.
```

Látszik, hogy elkezdett dolgozni, felolvasta a soci elemet, felolvasta a marci elemet, hisz azt hitte, hogy az a soci elem gyermekeleme, és akkor döbbent rá, hogy ez nem igaz, amikor az 5. sorban megtalálta a NetAcademia kezdőtagot.
A példa felhívja arra is a figyelmet, hogy a .NET-ben vége a HRESULT-oknak, a hibakódok figyelésének és egyéb bohóckodásnak. Strukturált hibakezelést lehet és kell alkalmaznunk, ha nem tetszik akkor is, mert az alaposztályok is ezt használják. És ez így van jól.

Az XmlReader és a SAX

Nem, nincs semmi zenei vagy pajzán felhangja a címnek. :) Aki ismeri az XML DOM-ot, az már látja, hogy az XmlReader teljesen más felfogásban olvassa fel az XML doksikat. A DOM felolvassa a teljes fájlt, és felépít belőle egy fát, amelyben minden levélobjektum a forrásdokumentum egy-egy node-jának felel meg. Ha százmegás a forrás, akkor felépít egy több száz megabájtnyi memóriát fogyasztó struktúrát. Emiatt a DOM nem alkalmas nagy adattömegek feldolgozására. Mielőtt azonban kifütyülnénk a DOM-ot érdemes átgondolni, hogy a már felépített fában könnyedén lehet mászkálni előre-hátra, míg az XmlReader-nél szó sincs erről: csak előre megy. Valamint a DOM-ba betöltött XML dokumentum bármely részét módosíthatjuk, és a módosítást a DOM vissza tudja írni.
1998-ban, az XML specifikáció és az XML DOM megszületése után hamar nyilvánvalóvá vált, hogy a DOM nem jó mindenre. A memóriaproblémáról már beszéltem. Sokszor nem akarunk oda-vissza mozogni, csak előre, amelyet nyilvánvalóan sokkal hatékonyabban is meg lehetne tenni, nem kell betölteni az egész doksit a memóriába. Emellett sokszor nem érdekel minket a teljes XML forrás, csak annak egy része. A számunkra érdekelten részeket egyszerűen át akarjuk ugrani. A problémák megoldására kifejlődött egy másik XML elérési módszer, melynek neve SAX *(Simple API for XML)*. Ez nem w3c ajánlás, ennek ellenére a legtöbb gyártó XML programcsomagjában megtalálható. Az MSXML is tartalmazza.
A SAX push modellt használ. Ez azt jelenti, hogy a SAX parser elkezdi felolvasni a doksit, és amikor megtalál egy node-ot, akkor visszahívja az alkalmazásunk megfelelő metódusát. Ehhez nekünk implementálni kell egy ContentHandler inter-

fészt, amit a SAX API definiál, és a saját megvalósításunkat kell átadni paraméterül a SAX parsernek.



A módszer működik, gyors, teljesíti azokat a felsorolt igényeket, amelyeket az előbb összeszedtünk, azonban ez a callback *(visszahívásos)* jellegű működés elég idegen a legtöbb fejlesztő számára. Általában azt szeretjük, ha a mi kezünkben van a karmesteri pálca, és mi mondjuk meg, mikor mi történik a program folyamában. Itt viszont elindítjuk az értelmezést, és zúdulnak be a visszahívások a SAX readertől: „nesze, itt egy elem, melynek jellemzői: ...", „tessék, itt egy szöveges tartalom, kezdjél vele valamit"... Ha megnézzük, ez szinte ugyanaz az elv, mint amit a .NET-es XmlReader használ, csak abban mi cibáltuk elő a következő node-ot a Read() hívásával. Ezért mondják, hogy a .NET-es megvalósítás pull jellegű. Megvan az összes jó tulajdonsága, mint a SAX-nak, csak könnyebb használni, mert mi jelzünk, ha kell a következő node.
Ha valaki mindenáron SAX-olni akar, akkor az MSXML-t kell használnia, mert a .NET-ben nincs SAX parser.

XmlWriter

Nézzük meg a visszafelé vezető utat! Hogyan lehet XML kimenetet generálni nem hagyományos eszközökkel? Például összeragasztgatjuk sztringműveletekkel a kívánt XML kimenetet. Nagyon flexibilis, de miért kell nekem pontosan ismernem az XML és a Namespace szabványt ahhoz, hogy XML tartalmat állítsak elő? Nem a kacsacsőrökről van szó, annál azért sokkal összetettebb dolgok is vannak a szabványban. Karakterentitások *(zűrös karakterek)* lekezelése, például az idézőjel (")helyett ". A bináris információk Base64 kódolása. Leellenőrizni, hogy jól formázott-e az összetákolt XML-nek látszó ASCII szöveg? Ezek nem rám tartoznak.
Engem nem érdekel a pontos XML formátum, csak az, hogy milyen struktúrát, milyen információtartalmat akarok XML formátumban látni. Én elemekben és attribútumokban gondolkodom, az XML szabvány meg kacsacsőrökben és idézőjelekben. Másképpen fogalmazva engem csak az XML dokumentum Infoset-je érdekel, azzal akarok dolgozni, nem ASCII szövegekkel. Emiatt az első megoldás kiesett.
Második ötletként létrehozok egy XML DOM objektumot, és azon keresztül már tényleg Infoset-ben gondolkodva tudom felépíteni a dokumentumot. Például azt mondom a DOM-nak, hogy adjunk hozzá egy soci nevű elemet, melynek szöveges tartalma kelbimbó. Hogy ez a végén egy <soci>kelbimbó</soci> karakterfolyam lesz? Hogyne, de ne kelljen nekem ezzel törődöm. Magyarul, az Infoset-tel dolgozva a DOM megfelelő metódusaival felépítem a kimeneti dokumentumot, és elmentem azt egy fájlba. A módszer működik, csak ugyanaz a probléma vele, mint ami olvasáskor is előjött: a memóriában rakjuk össze a kimenetet, és csak a legvégén írjuk ki fájlba. Ez kis doksiknál oké, nagyoknál gáz.
Talán a SAX? Igen, a SAX-nak a vissza irányra is van streaming modellje, azaz tudok úgy XML kimenetet generálni, hogy a memóriában mindig csak az aktuálisan generálandó node van, a már elkészült tartalom folyamatosan íródik ki a kimeneti állományba. Aki COM világban szándékozik nagy XML állományokat generálni, az mindenképpen vegye fontolóra a SAX API MXXMLWriter objektumát.
Nézzünk egy negyedik variációt, ami már .NET-re épít! Az XmlWriter nagyon hasonlóan gondolkodik, mint a SAX XMLWriter-e, csak egyszerűbb a használata. Lássunk egy egyszerű példát, ami a korábban látott XML kimenetet generálja:

```
1 using System;
2 using System.Xml;
3 using System.Text;
4
5 public class XMLTeszt
6 {
7   public static void Main()
8   {
9     XmlTextWriter w;
10     w = new XmlTextWriter("b.xml",
11         Encoding.GetEncoding("ISO-8859-2"));
12     w.Formatting = Formatting.Indented;
13     w.Indentation = 2;
14     w.QuoteChar = Char.Parse("'");
15
16     w.WriteStartDocument();
17     w.WriteStartElement("NetAcademia");
18     w.WriteStartElement("soci");
19     w.WriteAttributeString("labmeret",
20       XmlConvert.ToString(45));
21     w.WriteString("Soczó Zsolt");
22     w.WriteEndElement();
23     w.WriteStartElement("marci");
24     w.WriteAttributeString("labmeret",
25       "49");
26     w.WriteEndElement();
27
28     w.WriteEndElement();
29     w.WriteEndDocument();
30     w.Close();
31   }
32 }
```

A kimenet:

```
1 <?xml version="1.0" encoding="iso-8859-2"?>
2 <NetAcademia>
3   <soci labmeret='45'>Soczó Zsolt</soci>
4   <marci labmeret='49' />
5 </NetAcademia>
```

Majdnem azonos a korábbival, de mégsem. Először is, az encoding értékében a betűk kisbetűkkel vannak írva, míg én kézzel írt példában naggyal írtam. Van jelentősége? Nincs. Aztán, én a marci elemet lezártam egy záróelemmel, az XmlWriter viszont helyben lezárta az elemet. Van különb-

ség a két formátum között? Mint szövegeket nézve természetesen igen. Információtartalmában van különbség közöttük? Nulla, semmi! Ezért találták ki az Infoset **[3]** szabványt, amin keresztül nem ASCII szöveget, hanem elvont fogalmakat, elemeket, attribútumokat, megjegyzéseket, vezérlő utasításokat és egyebeket találunk. A DOM, a SAX, és az összes .NET-es XML osztály is az Infoset-tel foglalkozik, a konkrét formátum csak ezen eszközök íróit érdekli.

Mit látunk a példaprogramban? Létrehozunk egy XmlWriter példányt ⑩. Első paraméterként meg kell adnunk egy Stream-et, amibe fog csurogni a generált XML dokumentum, vagy egy fájlnevet és egy karakter kódolási osztályt. Mivel a nemzeti karakterek kezelése most nem a fő témánk *(de még kapni fog egy teljes cikket a jövőben)*, azért egyelőre legyen elég annyi, hogy a magyar karakterkészlet helyes kezeléséhez a legtöbb esetben szükségünk lesz egy, a Kelet Európai nyelvek karaktereit leíró Encoding osztályra, amit a Encoding.GetEncoding(...)-gal kaphatunk vissza. Paraméterként a kódlap nevét vagy számát kell megadni, magyarhoz ISO-8859-2-t, vagy 1250-et.

A továbbiakban megadjuk, hogy a kimenetben beljebb lesznek tolva a gyermekelemek ⑫, mégpedig 2 karakterrel ⑬. Az attribútumok szövegét határoló karaktert a QuoteChar jellemzőn keresztül aposztrófra állítjuk. Az alapértelmezett az idézőjel. Nem feltétlen állítgatjuk át sűrűn, de jó tudni róla, hogy van ilyen lehetőségünk.

A kimenetbe még nem generáltunk semmit, csak beállítottuk, hogyan viselkedjék az író. Jöhet a tartalom! Létrehozzuk az XML dokumentumot ⑯, ennek hatására elkészül az XML deklaráció, azaz a kimenet 1. sora. Az encoding attribútum értéke az induláskor beállított encoding-ból jön.

A további tartalmat a sokféle WriteXyz metódussal generáljuk. Logikus a felépítésük, ezért inkább a XmlConvert.ToString-et emelem ki ⑳. Ha van egy nem string típusú értékünk, akkor azt stringgé, szöveggé kell konvertálnunk, hisz a végeredmény egy szöveges dokumentum. Praktikusan olyan formátummá érdemes alakítani ezen típusokat, amelyekből könnyű őket visszaalakítani az eredeti jelentésükké – majd ha visszaolvassuk a doksit. Ebben segít az XmlConvert osztály, mely képes a legtöbb alaptípust stringgé ill. visszafelé, stringből az eredeti típusra konvertálni. Elemezgesse csak helyettünk a változatos lebegőpontos szám és egyéb típusok szöveges ábrázolását! Példánkban egy egész számot, a 45-öt konvertáljuk át egy „4" és „5" karakterekből álló szöveggé.

A WriteString() ügyel arra, hogy a kimenetben ne jelenhessenek meg foglalt karakterek *(idézőjel, & jel, stb.)*, azokat mind kódolja az XML szabványnak megfelelően.

Ha bináris adatokat szeretnénk írni az XML kimenetbe, akkor azt először át kell alakítani szöveggé, lehetőleg valami szabványos algoritmussal. A Base64 kódolás erre teljesen jó, és ha erre van szükségünk, akkor a WriteBase64 metódus siet a segítségünkre.

```
w.WriteBase64(new byte[]{54, 189, 26}, 0, 3);
```

Azaz egy hárombájtos tömböt íratunk ki Base64 kódolással, a 0. bájttól kezdődően, 3 byte hosszan. A kimenet:

```
Nr0a
```

A három bájtól négy karakter lett, hisz ez a kódolás 64 kimeneti értékre, karakterre kódolja le a 256 féle értéket kihasználó bemenetet *(bájt)*, így valahol el kell tárolni a kieső két bit tartalmát.

Az XmlReader ReadBase64 metódusa örömmel fogadja és visszaalakítja a lekódolt szöveget bájttömbbé. Ezzel a módszerrel könnyedén tudunk bináris adatokat is pakolni egy XML dokumentumba.

Zárszó

A .NET XML osztályok két oszlopos tagját immáron – részben - megismertük. Sok mindent viszont még nem tárgyaltunk. Az XML DOM rendelkezésünkre áll .NET-ben is. Nem néztük meg hogyan működnek az XSL transzformációk, illetve hogyan lehet navigálni egy dokumentumban XPath-al. Ezeket a témaköröket fogjuk áttekinteni a következő számban.

És ha már .NET, akkor se csüggedjen az Olvasó, ha az itt látott programok kínaiul hatottak. Januártól egy új rovatot indítok az újságban, amely a .NET alapjaitól indulva fogja hónapról-hónapra kalauzolni Önöket a .NET nem rögös, de meredek útján. Akinek mélyebb és gyorsabb tréningre van szüksége, annak ajánlom a **[2]** megtekintését.

Az XML rovat nem szűnik meg, de az valószínű, hogy egyes részeiben már .NET alapon fogunk gondolkodni. Annyi hasznos technológia lapul még az XML ládikájában, hogy évekig csak erről írhatnánk...

De addig is kellemes, békés Karácsonyt és boldog új évet kívánok minden Kedves Olvasómnak!

Soczó Zsolt
Zsolt.Soczo@netacademia.net

A cikkben szereplő URL-ek:

[1]: Microsoft Developer Network Library http://msdn.microsoft.com
[2]: Introduction to C# Programming for the Microsoft .NET Platform http://www.netacademia.net/courses/2124.asp
[3]: XML Infoset http://www.w3.org/TR/xml-infoset
[4]: Letölthető példák http://technet.netacademia.net/download/xml

Régi-új profilok

Windows 2000 - Active Directory régi-új profilok

K: *Tudna valaki segíteni, hogy hogyan kell egy Windows 2000-et visszahelyezni az AD-be úgy, hogy ne csináljon új profilt a felhasználónak? Újra kellet csinálnom a domaint hardverhiba miatt, és most az AD-ben (Active Directory) nincsenek meg a gépek. Ha újra beléptetem őket, akkor meg teljesen új profilt hoz létre a felhasználónak és lehet konvertálgatni a régi beállításait, leveleit, Office újraaktivál, stb... A régi profilra meg azt mondja: ismeretlen.*

V: A megoldást, mint sok más esetben a Windows registry megfelelő kulcsának szerkesztésével oldhatjuk meg.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\
CurrentVersion\ProfileList
```

Ezen registry kulcs alatt vannak a felhasználók Security ID-i, illetve a profilok elérési útja *(ProfileImagePath)*. A ProfileList alatt a user SID-jének megfelelő kulcs alatti ProfileImagePath érteket kell úgy megváltoztatni, hogy a régi profilkönyvtárra mutasson, így a felhasználók régi profiljukat „visszakapják". Ha a user SID-ek is újak, akkor a fentiek nem elegendőek. A jogosultságokat is meg kell változtatni a profilkönyvtáron ÉS a HKEY_CURRENT_USER kulcson is *(amit az ntuser.dat fájlból tölt be a rendszer)*!

1. Elsőként a régi profilkönyvtáron be kell állítani az ntfs jogosultságokat az új felhasználónak. *(Alapértelmezésben az Administrators csoportnak, a SYSTEM accountnak és magának a felhasználónak van full control jogosultsága az adott profilon)*
2. Másodiként a profilkönyvtár gyökerében lévő ntuser.dat registry hive-ban is be kell állítani az engedélyeket. Ez amiatt fontos, mert ha nem állítjuk be, akkor a felhasználónak nem lesz joga a saját HKEY_CURRENT_USER kulcsára, és a belépés meghiúsul. Ezt megtehetjük regedt32-vel *(be kell tölteni a struktúrát (hive) a HKEY_USERS kulcs alá, módosítjuk az engedélyeket (permission), majd elmentjük (struktúra eltávolítása (unload hive))*.

Meg lehet spórolni a registry szerkesztését a Vezérlőpult-Rendszer-Felhasználói Profilok *(Control Panel-System- User Profiles)* menüpont alatt. Itt az összes engedéllyel kapcsolatos műveletet automatikusan elvégzi a Windows a MÁSOLÁS*(Copy to)* funkcióval. Itt a HASZNÁLHATJA *(Permitted to use)* alatt kell megadni másolás előtt azt a felhasználót, akié a profil lesz.

Forrás : Netacademia Windows 2000 Lista

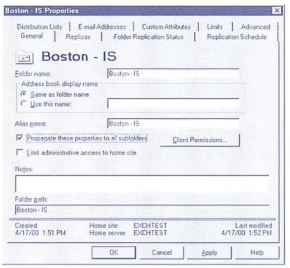
MS Exchange - elveszett jogok visszaállítása

K: *PST-ből (Personal Storage Folder) lett visszatöltve egy public folder, és a jogok elvesztek. Van arra valami scriptlehetőség, hogy az almappák jogait be tudjam állítani, vagy marad a manuális szögelés az Outlook vagy az ExchAdmin-PublicFolders alatt?*

V: A Microsoft Backoffice Resource Kit tartalmaz két hasznos szoftvert a Public Folderek jogosultságainak ellenőrzésére és beállítására.

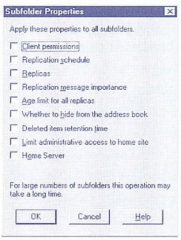
- A pfinfo.exe program parancssorból indítható és egy fájlban helyez el információkat a Public Folder jogosultságairól.
- A pfadmin.exe program parancssorból indítható és a Public Folder jogosultságainak beállítására, megváltoztatására szolgál.

A probléma megoldásához nem kell ezeket a kényelmetlenül használható eszközöket elővenni. Kiszolgálóoldalon a Public Folder tulajdonságainak beállításakor lehetőség van a beállított jogokat az alkönyvtárakra is átadni. Ezt a „Propagate these properties to all subfolders" kiválasztásával oldhatjuk meg. Ez a következő képen látható:



A public folderek beállításai lemásolhatók az alkönyvtárakra is

A beállítás a hierarchiában minden alsóbb könyvtár jogait megváltoztatja. Ha csak az adott könyvtár jogait kellene beállítani, akkor ez problémát okozhat! Van egy utolsó lehetőség, az „OK" vagy „Apply" gomb megnyomása után. Feljön egy ablak, melyben kiválasztható a továbbadni kívánt tulajdonságok listája.



Mit másoljunk le a mappákra?

Forrás : Netacademia Exchange Lista

Millenium avagy hálózati Gaia módra

A XXI. század kalmárja virtuális boltot nyit az információs sztráda egyik forgalmas csomópontjánál. Éppenséggel jól megy neki, de mehetne jobban is. Merthogy a Karácsony az még mindig Karácsony, és az Interneten is december végén van. Kénytelen hát jól felvértezni magát dögös kiszolgálókkal, hogy bírja az ünnepi rohamot. Bírja. Ez az év is elmúlt, és sajnos jövőre sincs kilátás arra, hogyan hasznosíthatná méregdrága, a Jézuska következő látogatásáig lustálkodó számítógépeit.

Ne írjon, ne telefonáljon! A Microsoft régóta tud a problémáról. Egy ideig türelmesen figyelte az informatikus társadalmat, amely Szent Grálként tekintett a láthatatlan hálózat problémájára, míg végül 1996-ban úgy döntött, hogy maga veszi kézbe ezt az ügyet *(is)*.

Millennium. Raktározzuk el memóriánkban ezt a nevet, mert néhány éven belül egészen biztosan gyakrabban találkoznunk majd vele. *(Nem a Windows 98 utódjáról beszélünk! Ez a Millennium nem az a Millennium!)* A képernyővédő-szindróma már másnak is szemet szúrt. Nemes dolog bekapcsolódni a SETI programba, a rákkutatásba meg főleg, de itt most még ennél is többről van szó.

A hangzatos fedőnév egy elosztott operációs rendszert takar. Kalmárunk ennek módfelett örül, mert rögtön tudja, hogy így nem kell annyi dögös gépet vásárolnia. A vevőözön idején a Millennium replikálja a túlterheltség jeleit mutató webkiszolgáló tartalmát olyan gépekre, amelyek kihasználtsága ekkor kisebb mértékű. Közös teherviselés van: a cél érdekében a munkaállomások éppúgy munkára foghatók, mint a kiszolgálók. Az áradat elvonultával az erőforrások újra felosztásra kerülnek az igények optimális kielégítéséhez.

A Millennium a COM+ technológiára épül, és ugyanúgy kezeli majd elosztott módon az alkalmazásokat, ahogy a mai operációs rendszerek teszik ezt egy központi számítógépen. A Microsoft, filozófiájának megfelelően, a bonyolult technológiát elrejti az avatatlan szemek elől. Kalmárunk az egész háttérfolyamatból mit sem lát, és saját bevallása szerint egy cseppet sem érdekli, amíg a vevők elégedettek a kiszolgálás sebességével.

Üzleti szempontból az előny jól látható: a feldolgozási teljesítmény, a memória és más eszközök optimális elosztásával egy adott feladatot a leghatékonyabban lehet végrehajtani, ami idő és pénz megtakarítását jelenti. Közelítsük most meg a lehetséges előnyöket felhasználói oldalról: kihasználhatjuk a hálózaton lévő eszközök funkcióit, felgyorsul az alkalmazások végrehajtása, nem kell többé foglalkoznunk a hely kérdésével, hiszen egy nagy számítógéppel nézünk szembe.

Tegyük fel például, hogy diktálni szeretnénk egy szöveget a kis palmtopunknak, amely majd a hangot átalakítja elektronikus dokumentummá. Erre bizony nem egy palmtop a legmegfelelőbb eszköz. Gond egy szál se, biztosan van a hálózaton nagyobb teljesítményű gép is, amelyre átterhelhető a feladat. Ezt persze legfeljebb sejthetjük, hiszen, az egészből semmit sem érzékelünk – hacsak nem azt, hogy a palmtop hihetetlenül gyors.

Nem tudom más hogy van vele, de én bizonyos munkáimból két példányt is kénytelen vagyok tartani: egyet a munkahelyi, egyet az otthoni gépemen. Ha az egyiken változtatok, indulhat a cserebere. Milyen szép is lesz majd, ha nem kell azon gondolkodnom, hogy hol van az x.txt, egyszerűen csak meg kell nyitnom. A Millennium majd szépen megkeresi helyettem.

A kutatás két fő területre összpontosul. Az egyik az absztrakció mértékének növelése, hogy a programozók számára az információ a mainál egyszerűbb formában jelenjen meg. A másik az önbeállítás és az önszabályozás kérdése. A tervek szerint a hálózatba kerülő új eszközöket automatikusan felismeri majd, miután a munkák elosztását az új erőforrásviszonyokhoz igazítja. Az elosztott rendszerek vizsgálatával ellentétben, ez a két terület eddig nagyon is kevés figyelmet kapott. Bár az automatikus eszközfelismerést a Microsoft Universal Plug and Play-e és a Sun Microsystems Jini-je is jól tudja, itt megintcsak többről van szó. Az egyik kutató példájával élve e két megoldás ahhoz hasonlítható, mikor a telefonokból telefonrendszert építünk. A Millennium ezzel szemben olyan, mintha úgy beszélhetnénk valakivel, hogy ő közben akár a világ túlsó felén is lehet, ráadásul még arról sincs tudomásunk, hogy a beszélgetéshez egyáltalán telefont használunk-e.

A Microsoft szándékai szerint a kutatási eredmények már az elkövetkezendő néhány évben is bekerülhetnek egyes termékekbe. A kutatócsoport már több prototípussal is előállt, amelyek jól megvilágítják egy Millennium típusú operációs rendszer rendkívüli lehetőségeit.

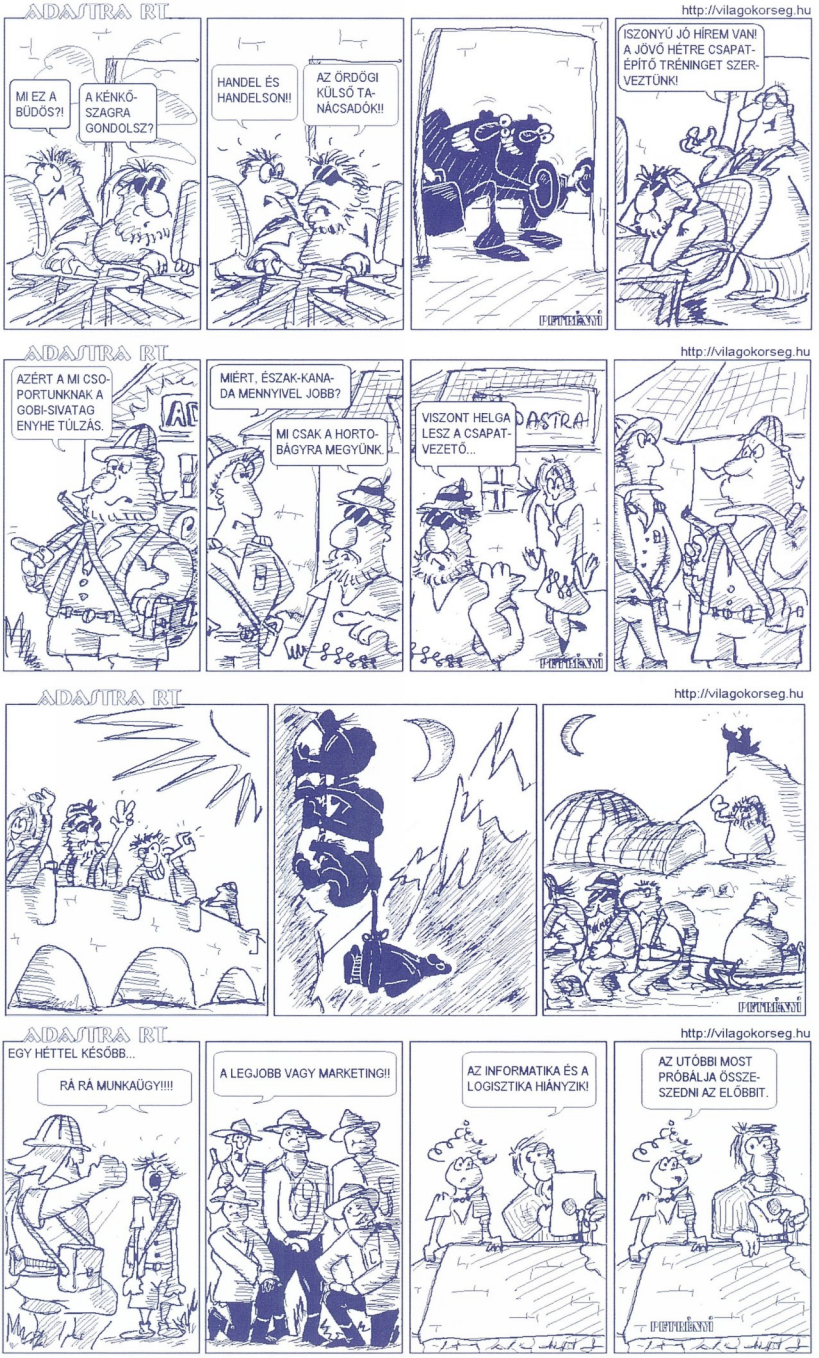
A Coign például nem elosztott alkalmazásból elosztottat készít anélkül, hogy ehhez az eredeti kódra is szüksége lenne. A mai alkalmazások így könnyen és gyorsan átalakíthatók a Millennium követelményrendszere szerint.

A COP a számítógéphez csatlakoztatott eszközök megosztását teszi lehetővé, legyen szó akár egérről, billentyűzetről, monitorról, merevlemezről vagy bármi másról. A felhasználók és az alkalmazások számára azonban a használatba vett eszközök egyetlen számítógépként jelennek meg, ami az absztrakció fokozódását és az információk érthetőbb megjelenítését mutatja be.

A Millennium Falcon *(amely, mint tudjuk, másfélszer gyorsabban repül a fénynél :-)* jelentősen csökkenti az alkalmazások különböző gépeken futó komponenseinek kommunikációjához szükséges időt. Ez a prototípus azt hivatott szemléltetni, hogy az információ feldolgozásának sebessége közel olyan nagy lehet akkor is, ha az adatoknak több számítógép között kell áramlani, mintha azokat csak helyben használnánk fel.

Tehát ismét egy forradalmi technológia *(vagy inkább technológiai forradalom?)* születésének lehetünk tanúi. A Millennium a számítógépesített eszközök hálózatát egyetlen gigantikus elektronikus aggyá szervezi. Erőforrás itt kihasználatlanul nem maradhat. A Millennium a feladatok végrehajtásának hatékonyságát tekintve már-már az emberi agy működésére, vagy legalább ennyire a Lovelock elméjében megfogant élő bolygóra, a Gaiára emlékeztethet bennünket.

(Zacco)


ADASTRA RT
http://vilagokorseg.hu
MI EZ A BÜDÖS?!
A KÉNKŐSZAGRA GONDOLSZ?
HANDEL ÉS HANDELSON!!
AZ ÖRDÖGI KÜLSŐ TANÁCSADÓK!!
ISZONYÚ JÓ HÍREM VAN! A JÖVŐ HÉTRE CSAPATÉPÍTŐ TRÉNINGET SZERVEZTÜNK!
PETIKÉNYI
AZÉRT A MI CSOPORTUNKNAK A GOBI-SIVATAG ENYHE TÚLZÁS.
MIÉRT, ÉSZAK-KANADA MENNYIVEL JOBB?
MI CSAK A HORTOBÁGYRA MEGYÜNK.
VISZONT HELGA LESZ A CSAPATVEZETŐ...
EGY HÉTTEL KÉSŐBB...
RÁ RÁ MUNKAÜGY!!!!
A LEGJOBB VAGY MARKETING!!
AZ INFORMATIKA ÉS A LOGISZTIKA HIÁNYZIK!
AZ UTÓBBI MOST PRÓBÁLJA ÖSSZESZEDNI AZ ELŐBBIT.

Adatátvitel galambpostán

Ehavi számunkban két kísérleti szabványt mutatunk be: a repülő adathordozókon alapuló IP datagramtovábbítás eredeti változatát *(RFC 1149,* **[1]***)*, valamint a szolgáltatásminőségi *(Quality of Service)* elemekkel kibővített, jóval kiforrottabb változatot *(RFC 2549,* **[2]***)*, amely az első után csaknem tíz évet váratott magára *(érdekes módon, mindkettő kiadási dátuma április 1)*.

RFC 1149: IP datagramok átvitele repülő adathordozók segítségével

Az RFC 1149 kísérleti szabvány jellemző felhasználási köre a Városi Hálózatok *(Metropolitan Area Network, MAN)* területe. A repülő adathordozókon *(avian carrier)* alapuló hálózat nagy késleltetési idejű, kis sávszélességű, kis magasságú szolgáltatás. A hálózat topológiája pont-pont, hordozónként; kora tavasztól több hordozó egymás zavarása nélkül képes a hálózati csomópontok *(hostok)* közötti egyidejű adatátvitelre. Mindez a 3D adatátviteli térnek köszönhető *(szemben az IEEE 802.3 (Ethernet) 1D adatátviteli terével)*. A rendszer beépített csomagütközés-elkerülési megoldással rendelkezik. Egyes más megoldásokkal *(pl. mikrohullámú rendszerekkel)* ellentétben nem okoz gondot, ha a hálózati csomópontok takarásban vannak. Nagyobb városokban *(pl. Velence)* kapcsolatorientált szolgáltatás is rendelkezésre áll, általában központosított csillagtopológiában *(lásd Szent Márk tér)*.

Az adatátviteli keretformátum

Az átvinni kívánt IP datagramokban található adatokat hexadecimális formában kis papírra nyomtatjuk, majd azokat feltekerjük. A kis tekercset ezután a hordozó lábára rögzítjük. A datagram széleit szigetelőszalaggal védjük. A sávszélességet a lábak hossza korlátozza; az egyidejűleg átvihető adatmennyiség *(Message Transfer Unit, MTU)*, paradox módon a hordozó korával egyenes arányban nő. Az MTU tipikus mérete 256 milligramm *(a datagramok méretét szükség esetén ki lehet tölteni)*. Az adat megérkeztekor a fogadó eltávolítja a szigetelőszalagot, majd a datagramot szkennerrel elektronikus formátumba alakítja vissza.

Egyéb jellemzők

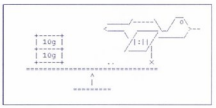
A rendszer további jellemzője a beépített féreg-felismerés és megsemmisítés. A sérült hordozók idővel regenerálják önmagukat. A rendszerben nem ismeretes a broadcasting fogalma, viharok adatvesztést okozhatnak. A hordozó az adatot többször megpróbálja kézbesíteni, mielőtt eldobná. A hordozók nyomkövetését segítik az automatikusan létrejövő „bejegyzések" *(utcákon, háztetőkön, köztéri szobrokon)*.

RFC 2549: Minőségi szolgáltatás – Quality of Service

Az RFC 2549 számos helyen javítja, illetve új elemekkel egészíti ki az eredeti szabványt *(bár továbbra is a „kísérleti" (Experimental) státuszt kapta)*. Az új változat minőségi szinteket vezet be: **Concorde**, **Business Class** és **turistaosztály**. A Concorde és a Business Class külön előnye, hogy semmilyen más adatátviteli metódus során nem jutunk törzsutaskedvezményekhez.

A szolgáltatás szintjét a hordozó szárnyára ragasztott vonalkóddal jelzik. Amikor a hordozó az útválasztóhoz ér, a vonalkódolvasó leolvassa az adatokat, majd a hordozót beállítja a várakozási sorba. A hordozó itt addig vár, míg el nem indulhat a célállomás felé *(a várakozás során a hordozó esetleg alhat egy kicsit)*. A hordozókat a várakozási sorban piros festékkel törlésre jelölhetik ki.

A súlyozott sorbaállítási algoritmus *(Weighted Fair Queuing, WFQ)* mérleg segítségével, az alábbi ábra alapján használható *(ábra az eredeti dokumentumból)*:



Weighted Fair Queuing

Amint az az ábrából is látható, a hordozók a hosszabb várakozás esetén nyomokat hagyhatnak. A repülő adathordozók általában nem használják a hidakat *(bridges)*, illetve az alagutakat *(tunnels)* a hálózati csomópontok között. A beágyazást *(encapsulation)* **folpackkal** oldhatjuk meg. Struccok is használhatók alternatív hordozóként: ezek nagyobb adatcsomagok egyidejű átvitelére alkalmasak, viszont lassabbak, valamint a tartományok között kénytelenek a hidakat *(bridges)* igénybe venni. Manapság divat a pingvinek használata is, azok ugyancsak nem tudnak repülni *(viszont ritkábban fagynak meg)*.

A multicasting csak klónozással oldható meg. A hordozó elvesztésével is számolnunk kell, ha éppen azt a fát vágják ki, amin a fészek található. A hordozók átlagos élettartama *(Time to Live, TTL)* 15 év. A hordozók nem állnak lexikai sorrendben, viszont gyakran szerveződnek V-alakba.

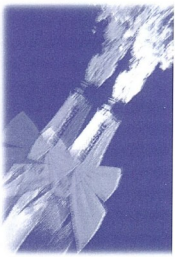
A szabványok gyakorlati alkalmazása

Nem megbízható hírforrások szerint két holland informatikus megpróbálkozott a gyakorlati alkalmazással: állítólag egy völgy két oldala között próbálkoztak „pingeléssel". A próba sikerrel járt, bár a késleltetés elég nagy volt, az egyik csomag ugyanis félúton leszállt egy háztetőre turbékolni...

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:
[1] http://www.ietf.org/rfc/rfc1149.txt
[2] http://www.ietf.org/rfc/rfc2549.txt

Hogyan írjunk karbantarthatatlan kódot?



Bevezető

Ahhoz, hogy meghiúsítsuk a kódunkat karbantartó programozó munkáját, meg kell értenünk, hogyan gondolkodik. Megkapja a monumentális programunkat. Nincs ideje, hogy elolvassa az egészet, és még kevesebb, hogy meg is értse azt. Minél gyorsabban meg akarja találni azokat a pontokat, ahol változtatnia kell a programunkon, lehetőleg úgy, hogy a változtatásnak nem lesznek káros mellékhatásai.

Ő egy WC papír csövén keresztül látja a kódunkat: egyszerre mindig csak egy pici részét látja a programunknak. Biztosak akarunk lenni afelől, hogy ebből soha nem áll neki össze a nagy kép a programunkról. Minél jobban meg akarjuk nehezíteni a dolgát, amikor egy számára fontos kódrészletet keres. De ami még fontosabb, hogy a lehető legkínosabbá kell neki tenni, hogy bármilyen apró részletet is biztonságosan átugorhasson.

A programozók a konvenciók miatt jól érzik magukat egy kódban. Azonban hébe-hóba ravaszul áthágva a konvenciókat, arra kényszeríthetjük a kódunkat karbantartó fejlesztőket, hogy nagyítóval olvassák és értelmezzék kódunk minden sorát.

Előbb-utóbb észrevesszük, hogy egy programnyelv összes funkciója magában rejti a karbantarthatatlan kód írásának lehetőségét, csak megfelelően vissza kell velük élni.

Elnevezések

A karbantarthatatlan kód írásának alapvető titka a változók és metódusok művészi elnevezése. Ez a fordítóprogramokat egyáltalán nem érdekli, viszont hatalmas mozgásteret hagy a karbantartó programozók megzavarására.

A Keresztnevek kézikönyvének új felhasználása

Ha megvesszük a fenti könyvet, akkor soha nem fogyunk ki jó változónevekből. A Mari remek név, és még könnyű is gépelni. Emellett, ha könnyen gépelhető nevekre vágyunk, használjuk ki billentyűzünk szomszédos betűit, így remek asdf vagy erty neveket kapunk.

Egybetűs változónevek

Ha a-nak, b-nek, c-nek hívjuk a változóinkat, akkor az egyszerűbb szövegszerkesztőkben nehéz lesz rájuk keresni, hisz nagyon sok szó tartalmazni fogja őket. Emellett senki nem fogja tudni melyiket mire használtuk fel. Ha valaki megkérne, hogy felejtsük el azt a Fortranban bevezetett szokást, miszerint a ciklusváltozókat i, j és k betűkkel jelölik, és arra kér, hogy cseréljük ki őket ii, jj, kk nevekre, emlékeztessük arra, mit tett a spanyol inkvizíció az eretnekekkel.

Kreatív elírások

Ha arra kényszerítenek minket, hogy beszédes neveket adjunk a változóknak, akkor írjuk el őket. Ha néha elírunk egy nevet, néha nem, akkor könnyen kilőhetjük a fejlesztőeszközök keresési funkcióit.

Legyünk elvontak

A változók és függvények elnevezéseiben használjunk minél elvontabb neveket: az, bármi, adat, kezelő, cucc, izé, eljárás, és használjunk számokat is bennük: EljárásX23, VégrehajtAdatFüggvény, Csináld, KezeldLeAzAdatokat3.

Rokonértelmű szavak használata

Unaloműzésként használjunk minél többféle rokonértelmű szót ugyanannak a fogalomnak a leírására. Pl. megjelenít, megmutat, kijelez. Homályosan utaljuk arra, hogy finom különbségek vannak közöttük, amikor valójában teljesen azonosak. Ezzel ellentétben, ha két függvény között jelentős különbség van, használjuk rájuk ugyanazt a szót. Például a nyomtat jelentse azt, hogy papírra nyomtatni, de azt is, hogy fájlba írni, vagy a képernyőn megjeleníteni. Semmilyen körülmények között ne engedjünk a követelésnek, hogy szójegyzéket írassanak velünk, ami egyértelműen lefektetné a projektben használt szavak jelentését. Ez egyértelmű megszegése lenne az információelrejtés tudományos elméletének.

Használjuk más nyelvek többesszámú szavait

Például egy VB program nyilvántartja a „statii"-kat, amelyeket különféle „Vaxen"-ekből kapunk vissza. Nagyszerűen használható nyelvek az Esperanto, a Klingon és a Hobbitese. Valamelyik szláv vagy északi nyelv is megfelelő lehet. Így programunk egyúttal a világbékét is szolgálja. Ha a kimeneti „kanal" nincs lezárva, akkor írjunk bele 3 „pripremu"-t. Vagy egy elágazásban, ha a valaki „uzeri" neve „Björk", akkor küldjünk neki „irgumburgum" üzenetet.

Használjuk újra fel a neveket

Ha az általunk használt nyelv megengedi, adjuk ugyanazt a nevet osztályoknak, konstruktoroknak, metódusoknak, tagváltozóknak, tulajdonságoknak, paramétereknek és lokális változóknak. Külön pont jár a {} blokkokban újradefiniált lokális változókért. A fő célunk, hogy a kódunkat olvasó programozó kénytelen legyen kielemezni minden példány érvényességi területét *(szkópját)*. Javaban például érdemes közönséges metódusokat konstruktoroknak álcázni.

Használjuk ki a fordítók maximális azonosítóhosszát

Ha például a fordító csak az első 8 karaktert veszi figyelembe egy azonosító feldolgozásakor, akkor változtassuk a nevek végét. Egyszer legyen karakter_beállít máskor karakter_kinyomtat. A fordító mindkettőt karakter-nek fogja értelmezni.

Az aláhúzás a mi jóbarátunk

Használjuk az _ és a __-t mint változóneveket.

Ékezetes betűk

Nekünk magyaroknak különösen kedves a módszer. Használjunk ékezetes betűket, ahol lehet:

```
typedef struct { int i; } int;
```

Szerencsére sokszor nehéz észrevenni, hogy van-e ékezet egy betűn. Emellett különös örömet okozunk egy olyan embernek, akinek a nyelvében nincsenek ékezetes betűk, így ha ki akarja bővíteni a programunkat meg sem találja a megfelelő betűt a billentyűzetén – mert nincs is rajta.

Keverjük a nyelveket
Véletlenszerűen keverjünk össze különböző nyelveket a változók elnevezésénél. Például PendingBorítéks *(c)*. Ha a főnökünk rákérdez miért nem az általa megadott nyelvet használjuk, akkor mondjuk el neki, hogy az általunk választott nyelven sokkal könnyebben tudjuk összeszedni a gondolatainkat. Ha ez nem elég neki, akkor emlegessük fel neki a nyelvi diszkriminációt, és ebből kifolyólag fenyegessük meg nagyösszegű perrel.

Nevek a matematikából
Válasszuk matematikai operátorok neveit változónévként:

```
NyitóZárójel = (osztás + szorzás) / egyenlő
```

A kis l nagyon hasonlít az 1-re
A long típusú konstansokban használjuk kis l-t nagy L helyett. Így például a 10l-et könnyen 101-nek lehet olvasni, míg a 10L-t senki nem tévesztené el.

Használjuk újra a globálisan definiált neveket mint private változókat
Deklaráljunk egy globális tömböt az A modulban, majd deklaráljuk egy private-ot a B modul fejlécállományában. Így a B modult programozva úgy tűnik, mintha az A-ban definiált globálisat használnánk, pedig nem. A megjegyzésekben természetesen ne tegyünk erről említést.

Használjuk újra a változóinkat
Ha a programozási nyelv megengedi, használjunk fel újra változóneveket teljesen más célokra. Hasonlóan, ugyanazt az átmeneti változót használjunk különböző célokra, veremhely spórolás címén. Az ördögi variant típust kihasználva *(VB :)* változtassuk meg a változó típusát vagy értelmezését. Például egy hosszú függvény elején rakjunk bele egy tömböt, majd a függvény közepe táján konvertáljuk át az 1-gyel kezdődő címzésű tömböt 0 kezdetűvé. Semmiképpen ne dokumentáljuk a változtatást.

Félrevezető nevek
A metódusaink mindig csináljanak egy kicsit többet, vagy kevesebbet, mint amire a nevük utal. Például egy ÉrvényesE*(x) (ebben a magyar miatt már több tanács is benne van)* mellékesen átkonvertálhatja x-et binárissá, és eltárolhatja adatbázisban.

objAlma, objConn, objRS
Minden osztálypéldány nevét kezdjük o vagy obj betűkkel, jelezve, hogy nagy, polimorfikus képben gondolkodunk.

Érthetetlen filmutalások
Használjunk olyan konstansneveket, mint LancelotKedvencSzine a kék helyett, és adjunk neki értékül valamilyen hexa színkódot: $0405FA. A képernyőn ez a szín egy közönséges kéket fog eredményezni, azonban a karbantartónak nem lesz könnyű dolga kitalálni azt. Csak a Monthy Python és a Szent Grál lelkes ismerői tudják, hogy Lancelot kedvenc színe a kék. Ha meg valaki nem tudja fejből idézni az összes Monty Python filmet, akkor hogy meri magát programozónak nevezni?

Álcázás

A karbantarthatatlan kódírás művészete az álcázás művészete. A dolgok elrejtéséé, vagy hogy másnak tűnjenek a dolgok, mint amik. Sok trükk azon alapul, hogy a fordítóprogramok képesek olyan finomságok észrevételére és kezelésére, amire az emberi szem és egy egyszerűbb szövegszerkesztő nem képes. Tanuljunk a profiktól!

Kód, ami megjegyzésnek tűnik, és viszont
Írjunk olyan kódrészeket, amelyek megjegyzésblokkban vannak, de ez első ránézésre nem tűnik fel.

```
for(j=0; j<array_len; j+ =8)
{
total += array[j+0 ];
total += array[j+1 ];
total += array[j+2 ]; /* A ciklus középső része
total += array[j+3]; * ki van bontva a
total += array[j+4]; * nagyobb sebesség kedvéért.
total += array[j+5]; */
total += array[j+6 ];
total += array[j+7 ];
}
```

Észrevették, hogy a középső három sor megjegyzésben van?

Rejtsük el a makrók definícióját
Rejtsük el a makrók definícióját ostoba, értelmetlen megjegyzések közé. A programozók elunják végigolvasni a hosszú megjegyzéseket, így nem veszik észre a köztük megbúvó makrókat. Feltétlen legyen olyan a makró, hogy például egy egyszerű értékadást cseréljen le valami bizarr műveletre:

```
#define a=b a=0-b
```

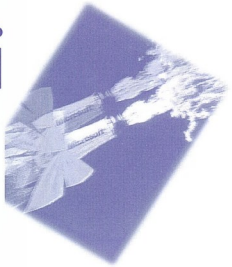
Tűnjünk elfoglaltnak
Írjunk függvényeket makrókként, amelyek egyszerűen megjegyzésekbe teszik a paramétereket:

```
#define fastcopy(x,y,z) /*xyz*/
...
fastcopy(array1, array2, size); /* nem csinál
semmit */
```

Szabadon fordította: Soczó Zsolt

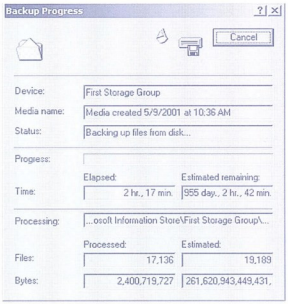
A cikkben szereplő URL-ek:
[1] Az eredeti, teljes szöveg (angolul)
http://mindprod.com/unmaint.html
[2] Letölthető példakódok:
http://technet.netacademia.net/download/szilveszter

Szilveszteri Dupla KV



K: *Gondoltam megosztom Veletek mai szívásom egy notebookkal: Ma délelőtt egy Mezei Notebook User (MNU) felhív, hogy furcsán fagyogat a gépe, méghozzá akkor, ha felkapcsolják a villanyt a szobában. Mondom neki „peeeersze... amikor a villanyt felkapcsolják... hehe... én meg a helyi káplár vagyok". De ő csak mondja, hogy így van, vazze, nézzem meg. Felbaktatok, és látom is, villany ég notebook szétszállt - csak a kikapcs segít. Láttam már szétfagyott gépet, úgyhogy restart, és várok, mikor, és mitől száll el megint. De hiába vártam semmi. MNU közli, hogy így nem fog elszállni, de felkapcsolja a lámpát, és majd akkor. Nem hittem a szememnek, a gép azonnal lefagyott. „Aaaaa" gondoltam magamban. Biztos véletlenül történt. Gyors restart, Task Manager be, és nézzük mi történik. Hát semmi. MNU ismét megjegyzi, hogy kapcsoljuk fel a lámpát és... agyam eldobom megint szétszállt. No ennek fele se tréfa, biztos szívatnak, kandi kamera van a szobában, de megnyugtatnak nincs szívatás. De akkor mi a frász van??? Ismét restart. Task Manager be, lámpa fel, gép elhal. Hmmm. No még1szer. Detto. Ekkor arra gondoltam, hogy tutira baj lehet az árammal meg a táppal, így nézzük mi van, ha akkuról megy a gép - de az eredmény ugyanaz. Kezdett a dolog egyre misztikusabbá válni, de nem adtam fel, kell lennie rendes magyarázatnak!! Ekkor jöttem rá, hogy ezekben a gépekben van infra is, ez lehet a bűnös. Gyorsan leragasztottam, gép be, lámpa fel - semmi. No még1szer... Semmi. Ok ragacs le - semmi, lámpa le - semmi, lámpa fel - halál. Ez az! Megvan!!! Most már csak az érdekelne, hogy a rákban döglik meg egy T... 4200 notebook infrája úgy, hogy - ha felkapcsoljuk a szobában lévő neonsort - szétfagy a gép?????? Hisz a neon kékes fényt ad! Így ha esetleg Ti is találkoztok MNU-val, aki közli, hogy ha felkapcsolja a lámpát, lefagy a gépe, hát... én már nem röhögném ki. Annyit még hozzá lehet tenni, hogy azóta felhívtam a T... szervizt, akik felvették a kapcsolatot a németekkel. Kiderült, hogy ez egy ismert probléma, és adtak új drivert, amivel mar nem fagy ki a cucc - annyira.*
V: nincs
Forrás: NetAcademia Offtopic lista

K: *Mit tegyek? A főnököm azt mondta, addig nem mehetek haza, amíg a mentés le nem fut. Egyelőre még csak két óra telt el, de valószínűnek tűnik, hogy további három évig nem mehetek haza. Ti mit tennétek az én helyemben? És ha valóban nem mehetek haza, mivel üssem el a hártalévő 955 napot?*

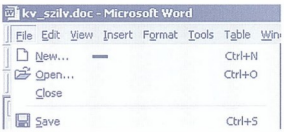


Egyes cégeknél Tera, sőt Peta, sőt Exabájtos tárolórendszerek vannak. Ezek mentése bizony soká tart!

V: Türelem!
Forrás: NetAcademia Offtopic lista

K: *A gépem egészen furcsán viselkedik. Amikor megbokrosodik, egyszerűen magától töröl mindent, amit ér. És ha már mindent kitörölt például egy wörd dokumentumból, akkor meg csönget. Mit tegyek?*
V: Vedd le azt a nehéz tárgyat a Backspace billentyűről, ami rajta van
Forrás: NetAcademia Offtopic lista

K: *Baj van a wördömmel. El akartam menteni a munkámat, és bementem a fájl menübe. Kattintottam a Save ponton, és erre eltűnt. No sebaj, gondoltam, majd a Save As elmenti nekem. Katt, és az is eltűnt. Most bármelyik menüpontra kattintok, az eltűnik. Gyors segítség kéne, mert már alig van menüpontom! Jaj, mit tegyek??*
V: Sikerült belemásznod a wörd testreszabási lehetőségeinek legveszélyesebbjébe: a menütörlésbe. Ide úgy szoktak emberek betévedni, hogy egy spirálfüzetet ráejtenek a billentyűzetre. Ekkor jó eséllyel egyszerre lenyomódik a CTRL, az ALT és a mínuszjel, s a kurzor is átalakul egy bazinagy mínusszá. Amire ezután kattintasz, az eltűnik. Lehet próbálkozni!



A File->New... menüpont percei meg vannak számlálva. Mindjárt lesújt az egér!

Forrás: NetAcademia Offtopic lista

Microsoft
CERTIFIED
Systems Administrator

kicsiknek és NAGYRA vágyóknak

MCP – túl kevés? MCSE – túl sok(k)? **Legyen MCSA!**

Az MCSA címre pályázóknak **három kötelező és egy szabadon választott vizsgán** kell megfelelniük. Bizonyítaniuk kell tudásukat a Windows 2000 Professional (070-210), a Windows 2000 Server (070-215) valamint a Windows 2000 környezet adminisztrálása területén (070-218) valamint egy Backoffice témakörben.

A SZÁMALK Továbbképzés 2002. év elejétől speciális összeállítású Microsoft rendszeradminisztrátori képzést indít, mely az alábbi tanfolyamokra épül:

A kötelező vizsgákhoz:

- 2151, 2152 – Implementing Microsoft Windows 2000 Professional and Server (üzemeltetési ismeretek, kiegészítve az alapvető hálózatos és Active Directory részekkel a 2151-es tanfolyam tematikájából)
- 2126 – Managing Microsoft Windows 2000 Network Environment (a Windows 2000 rendszer üzemeltetése)

A tanfolyamokat különböző csomagokban kínáljuk:

Standard csomag: 2151+2152, 2126 tanfolyam, ajándék táska: **220.000 Ft/fő**

Professional csomag: 2151+2152, 2126 tanfolyam, ajándék táska, plusz magyar nyelvű Kis Balázs könyvek (Windows 2000 Prof., Win. 2000 Server, magyar nyelvű 070-215 Readiness Review vizsgafelkészítő könyv, több mint 20.000 Ft értékben): **230.000 Ft/fő**

Premium csomag: 2151+2152, 2126 tanfolyam, ajándék táska, plusz a teljes Microsoft Press Windows 2000 Core Requirements tréning kit (Win. 2000 Prof., Server, Network, Active Directory, CD-melléklettel, melynek értéke több mint 60.000 Ft): **259.000 Ft/fő**

Meghirdetett időpontok tavaszra: 2002. január 28 – február 1., február 25 – március 1., március 4–8., március 25–29.
A tanfolyamok vidéki helyszíneinken is elérhetők lesznek. Kérjük tekintse meg weboldalunkat.

Az egy további szabadon választható vizsgához az alábbi tanfolyamainkat ajánljuk:

- 2153 – Implementing Microsoft Windows 2000 Network Infrastructure (hálózatüzemeltetés)
- 2159 – Implementing and Administering ISA Server 2000 (ISA Server 2000 üzemeltetés)
- 2072 – Administering Microsoft SQL Server 2000 Databases (SQL adatbázis adminisztráció)

Az MCSA képzésen résztvevő hallgatóink ezen tanfolyamok díjából további kedvezményt kapnak.

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.