1.344 Ft

II. Évfolyam 9. szám

ISSN 15865185 09

tech.net

A Microsoft Magyarország szakmai magazinja

Virtuális memóriakezelés

A Windows NT és a Windows 2000 memóriakezelése

Biztonság — Törölhetetlen mappák

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

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

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

Várjuk Önöket a NetAcademia Mesterkurzusokon

NETACADEMIA

A legjobbakat tanítjuk.

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

http://technet.netacademia.net/brainstorm

HunNet RL512W

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

.hu Hungarian Networks

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

2001 Szeptember

tech.net

A Microsoft Magyarország Szakmai Magazinja

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

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

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

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

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

ISSN 1586-5185

Köszöntő

BUG

Közismerten bugnak nevezik a szoftverekben található, jobbára átgondolatlanság miatt előforduló hibákat. Általánosan elterjedt legenda szerint a bug *(bogár)* kifejezés eredete abba a korba nyúlik vissza, amikor a számítógépek még sok ezer tűzforró elektroncsőből épültek fel. Ezek a „számítógépek" szinte percenként hibásodtak meg teljesen maguktól is, egész egyszerűen a csövek átlagos élettartamának hossza, és a magas felhasználási darabszám miatt. Az idő előrehaladtával és a csövek minőségének javulásával *(élettartamának növekedésével)* egyre inkább csökkent a statisztikai módszerekkel előre megjósolható leállások száma – maradtak viszont a látszólag értelmetlen, kiszámíthatatlan leállások, illetve számítási hibák. Ezek a gépek olyan méretűek voltak, hogy egy technikus simán besétálhatott a belsejébe, hogy forrasztópákájával rendet tegyen. Egy ilyen alkalommal talált egy bogarat, mely hősi halálával rövidre zárt egy áramkörészt, s ezzel okozott számítási hibákat.

Az eredendő bug tehát nem emberi mulasztásra vezethető vissza, manapság azonban szinte kizárólag ilyen értelemben használjuk: ha bugos a szoftver, akkor az emberi hiba következménye. A bugokra sok esetben „építeni" is lehet, azaz a hibát még nagyobb hiba kiváltására lehet felhasználni.

Itt van mindjárt a Code Red. Semmi különös sincs a viselkedésében, hisz egy közismert programozási hiányosságot, a veremre pakolt paraméterek ellenőrizetlen átvételét *(buffer overrun)* használta ki. Volt viszont ebben a kódban valami ördögi: közismert hibák és gyengeségek kiaknázásával eddig ismeretlen méretű fertőzési hullámot okozott. Ismét meg kellett tanulnunk, mit jelent, ha egy program memóriarezidens.

De nemcsak a szoftverek bugosak, hanem a társadalmak is. Nyár végén, augusztusban történt, hogy elvetődtünk Ópusztaszerre. Aki járt már ott, tudja: a bejárati pavilon előtti tűző napon, a sík betonon elviselhetetlen a forróság. Jelentős sor állt bebocsátásra várva, s én csakhamar az ájulás szélére kerültem. Félig ép elmével, elhomályosuló tekintettel észrevettem, hogy be lehet osonni a piramis alá, az árnyékba. Fogtam a két gyereket, és megmentettem őket is a biztos napszúrástól – csak a feleségem maradt a sorban, de ő néha önszántából is kimegy a napra, így nem kellett féltenem. Bent látom ám, hogy személyit, útlevelet kérnek a látogatóktól. Hinnye. Vajon miért? A torlódást tehát az igazoltatás okozta, ami olyan komoly volt, hogy-nekem gyerekestül be kellett vigyorognom az ablakon a nénihez, hogy mi bizony mi vagyunk, s a kilenc éves fiam rendelkezik diákigazolvánnyal! S mindez miért? Mert aznap minden magyarnak *(határon innen és túl)* ingyenes volt a belépés. Ottlétünk alatt a látogatók 100%-a bizonyult magyarnak. Kérdem én: nem lett volna egyszerűbb minden belépőnek feltenni egy-egy kérdést magyarul? Aki tud válaszolni, az magyar, és már mehet is be...

Ehhez hasonló bugok sokasága tartja béklyóban a magyar cselekvőképességét! Bezzeg Amerika! Abban az országban olyan szervezettséget tapasztalunk, hogy egyszerűen nincs helye a hatékonytalan működésnek.

- Az adószint olyan, hogy nem éri el a polgárok érzékenységi küszöbét – ebből következően fizetnek adót.
- Egyes fizető autópályákon a belépéskor kapott kártya alapján kilépéskor nemcsak a pályadíjat számítják ki, hanem a megtett távolság és az eltelt idő segítségével megállapítódik, hogy történt-e gyorshajtás, s annak díját is azonnal, automatikusan ráverik a polgárra. Nincs kecmec és csúszópénz: ha valaki X távolságot kevesebb, mint Y idő alatt tett meg, akkor túllépte a sebességhatárt, akármit handabandázik is. Mi meg most szereljük le az autópályakapukat.
- Közintézményekben az ajtók mindig a menekülés irányában, nyomásra nyílnak; kilincs sincs, nehogy akadálya legyen az épület kiürítésének. Ez a szabály a toalettajtókra is igaz: a rohanás irányába nyílnak – tehát befelé :)

A szervezettség kiterjed a turisztikai látványosságok körüli csődület kezelésére is: a repülőterekről ismert, terelőszalagokkal határolt labirintust olyan ügyesen szabják a tömeg méretére, hogy mindenki azt hiszi, halad a sor. A World Trade Center aljában a biztonsági őrök bekapcsoltatják a turistával a videokamerát, mert kíváncsiak, kamera-e egyáltalán.

És mégis, ennyi szervezettség ellenére kiderült, hogy minden ember alkotta rendszer bugos. A mások által akár fel sem fedezett bugokra bizony éppúgy lehet „építeni" a valóságban, mint a szoftverek esetében. Cracker és terrorista ellen nincs orvosság. Aki az életét teszi fel arra, hogy rendszereket döntsön romba, megkeresve a gyenge pontokat, az célt fog érni. Próbálná csak valaki „meghekkelni" mondjuk a természeti törvények valamelyikét. Ott nincs esély a gonoszkodásra.

Az ördögi támadás léket ütött eddigi világképünkön. Változások előtt állunk, melyek reményeim szerint az elmaradott népek felemelése felé fognak irányulni, mert különben a nyomor elkeseredettsége újra és újratermeli az öngyilkosjelöltek hadseregét. Ne sajnáltassuk magunkat: Magyarország népe messze a nyomorszint fölött él. Nálunk a legutolsó szakadt alkoholista koldusnak is naponta sokszor jut fehér kenyér, senkinek a gyermekét nem sorozzák be gerillahadseregbe, senkinek a családját, rokonságát nem fenyegeti sem éhezés, sem hajléktalanság, sem népirtás. Na jó, árvíz fenyegetheti. De nem bangladesi méretű!

Nekünk is adnunk kellene valahogyan javainkból a világ elmaradottjai számára, de sajnos a mögöttem karattyoló TV híradó megszólaló személyiségei ilyesmiről nem beszélnek. Most akkor én adjak fel minden nap egy kilo kenyeret légipostán? ...

Na ebből sem lesz már köszöntő, az látszik. Odalett az informatikai vonal. Ez is bug: De még a terroristák műve sem hibátlan: nem tervezték be WTC épületeinek összeomlását. Harun Al Rasid (vagy mi a fene a neve) megköszönte Allahnak, hogy feltette az i-re a pontot. Pedig nem Allah kéze van a dologban: bugos az egész világ!

FM

A Windows NT és a Windows 2000 memóriakezelése

avagy

What is the Matrix?

Sokszor elhangzó kérdés, hogy ha egy hálózatban X számú felhasználó van, Y darab megosztott könyvtárban dolgoznak Z darab fájllal, akkor mennyi RAM kell a Windows NT/SBS/2000 kiszolgálóba. Sajnos erre a kérdésre a Microsoft modern operációs rendszerei esetén nincs egyértelmű válasz, mert a virtuális memóriakezelés virtualizálja a fogalmakat is. Mit tekinthetünk foglalt memóriának? Mi a szabad? Mikor kell lapozni? Ezekre sincs egyértelmű válasz: tiszta Matrix az egész. Amikor Neo megkérdezi, hogy a valóságot látja-e, Morpheus így válaszol: „What is real?"

A valóság megismeréséhez kemény tanuláson kell átesni. Az alábbi cikk nem felületes olvasmány, akinek nem megy elsőre, olvassa át mégegyszer! Ha kérdései vannak akkor pedig írjon a Windows 2000 listára. Vitassuk meg!

Mit is jelent a virtuális memóriakezelés?

How do you define real?

Különösen fájó ez a gumivalóság, ha hardverbővítésnél kell okosat mondani, bár napjaink RAM árai mellett már könnyű benyögni a gigát - úgysem okoz gondot a kifizetése. Ennek ellenére úgy gondolom, érdemes megismerkedni a memóriafoglalás rejtelmeivel, mert teljesítménytuningoláskor nem árt a tisztánlátás!

Kezdjük a virtuális memóriakezeléssel *(VM)*: áldás, vagy átok? Sokak számára ez a fogalom az állandóan zörgő merevlemezzel és a lassan vánszorgó rendszerrel azonos, legszívesebben kikapcsolnák – ha lehetne. Mindazok, akik a VM kikapcsolásával próbálkoznak, vasvillával hányják a diót a padlásra. A Windows XP-ben meg lehet szabadulni a lapozófájltól – de ez nem egyenértékű a VM kikapcsolásával. Aki komolyan gondolja, hogy neki nem kell VM, az telepítsen egy Windows 3.1-et, és a WIN.COM-ot mindig /r *(real)* kapcsolóval indítsa!

Valójában a VM sokkal több előnnyel, mint nyűggel jár, nem csoda, hogy lassanként a konkurensek *(Novell Netware, Apple Macintosh)* is elérkeznek ide. Az sem igaz, hogy a VM egyet jelentene a lapozófájl zörgetésével, bár ha kevés a RAM, bizony előfordul ilyesmi. A VM alapvetően az Intel *(és más gyártók)* processzorainak hardverlehetősége arra, hogy az alkalmazások elől elrejthető legyen a párhuzamosan futó több száz végrehajtási szál egymástól elkülönített memóriaterületeinek borzalmasan bonyolult kezelése és védelme. Ha kikapcsolható lenne a Windowsban a VM, megszűnének a védett memóriaterületek, és a rendszer stabilitása is! *(Egyik Macintosh-használó ismerősöm csodálkozott rá a múltkor, hogy a Windows alatt van Task Manager, s hogy ha egy alkalmazás lefagy, attól még nem kell újraindítani a gépet. A Mac a mai napig nem tart itt!)*

A hardveres alaplehetőségre épít, és azt bonyolítja tovább a Windows, amikor kihasználja, hogy ha olyan memóriaterületre bökünk a címtérben, amely mögött nincs fizikai RAM, akkor a gép nem lefagy, hanem ún. Page Fault megszakítást kezdeményez. A megszakításkezelő rutin pedig bepótolhatja a hiányzó memóriát, és utasíthatja a processzort, hogy térjen vissza a félbehagyott műveletre. A Windows esetében a memórialapok mérete egységesen 4 kilobájt, ami némi lapozási pazarlással jár, ha csak 3 k-t kellene belapozni, de busásan megtérül abban, hogy a lapozófájl kezelése sokkal egyszerűbb: nem kell változó hosszúságú blokkoknak helyet keresni, nem kell töredék helyeket egybenöveszteni – egyszóval nincs szükség szemétgyűjtésre *(Garbage Collection)*.

Először 1997-ben dühödtem be a VM működésének érthetetlenségén. A rendszer megfigyelésével néhány olyan gondolat merült fel bennem, mely végül elvezetett a memóriakezelés megismeréséhez. Ezt követően került kezembe az Inside Windows NT egyik kiadása, melyben meglepve olvastam korábbi natív spekulációim igazolását. Az i-re a pontot pedig David Solomon mostani Tech.Ed előadása tette fel: tiszta a kép!

Az első apró jelek

Like a splinter in my mind that drives me mad

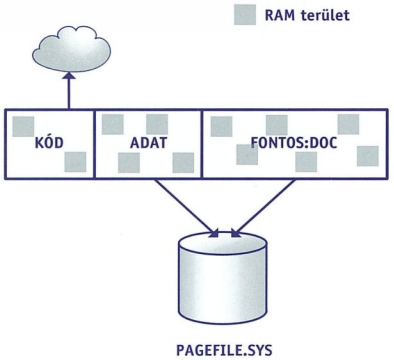
Először gyakorló rendszergazdaként vettem észre, hogy valami nem stimmel a világgal. Volt egy hálózatom több száz olyan munkaállomással, melyeknek merevlemezére az Office egyszerűen nem fért fel. Mit tesz ilyenkor a rendszergazda? Elolvassa a dokumentációt, és felfedezi, a SETUP.EXE /A módszert, mellyel az Office hálózatos telepítésére is lehetőség nyílik. S fut a WINWORD.EXE a hálózati megosztásról, mint a kisangyal - de meddig?

A kiszolgálók néha-néha lepihennek, ha nagyon fáradtak. *(Nem, a Linux az nem. Meg a NetWare sem. S ha már itt tartunk: a Windows sem. Akkor biztos ZX Spektrum kiszolgálóim voltak. Ki emlékszik erre ma már?)* Egy szó mint száz, az a VALAMILYEN kiszolgáló, amiről a Word futott, néha lepihent. Mit gondolnak, milyen közvetlen hatással volt ez a több száz felhasználóra? Döbbenetes módon

S E M M I L Y E N

hatással sem! Piri és Rozi zavartalanul csépelte tovább a Wordöt. Ez nem csoda - gondoltam - hisz a gépek leszedték a hálózati megosztásról az egész WINWORD.EXE-t a lapozófájlba *(pagefile.sys)*. Egy pár perc múlva azonban szóltam Pirinek, hogy jobb lesz, ha mégis elmenti a művét, mert a kiszolgáló öt perce leállt. És abban a pillanatban, hogy a mentésre kattintott, a Word úgy elszállt, hogy öröm volt nézni. Piri is örült, s ennek egy cifra káromkodással adott hangot. Bár nem értettem a jelenséget, a zavartalanul dolgozó Rozihoz új stratégiát agyaltam ki: neki azt mondtam, tegye vágólapra a művét. És csodák csodája, az akció sikerült, a Word még elbillegett egy darabig, majd ugyanúgy kinyiffant, mint Pirinél, de a doksi megvárt minket a vágólapon. Mi a két eset között a különbség? Egyáltalán miért esett el a Word, ha egyszer bekerült a helyi Windows lapozófájljába? Hát, mert nem került bele.

Sok-sok kísérletezés után rájöttem arra, amit kapásból tudnom kellett volna: nem az egész WINWORD.EXE lapozódik ki, hanem csak az adatszegmensei! A kódszegmenseket felesleges lapozófájlba tenni, hisz tökéletesen visszaállítható állapotban megtalálható valahol máshol: a megosztott könyvtárban a kiszolgálón! A Word addig képes futni, amíg képes pótolni a hiányzó EXE-falatkákat az eredeti fájlból.
Tehát a lapozás, és a lapozófájl csörgése nem sok összefüggést mutat a programok kódszegmenseinek használatával. Ha kevés a memória, a Windows igen hatékonyan „lapozza ki" a kódszegmensek tartalmát: egyszerűen eldobja! Az alábbi ábra ízelítőt ad a valóságról, s leolvasható róla a futó WINWORD.EXE memóriadarabkáinak kilapozási útvonala.

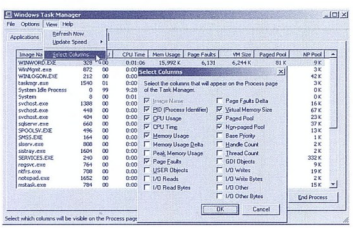


Egy alkalmazás kilapozása memóriahiány esetén. A szürke négyzetek a fizikai memóriában vannak – a többi terület nincs ott!

A kódszegmensek elpárolognak, míg az adatszegmensek és a heap memóriablokkok *(FONTOS.DOC)* valóban a lapozófájlba kerülnek.

Egy pár szó a Task Managerről

All I'm offering is the truth. Nothing more.
Próbáljuk megállapítani az előbb emlegetett alkalmazás, a WINWORD.EXE összesített memóriafoglalását! Elsődleges memóriamérőeszközünk a Task Manager, a szokásos, alapszámlálókon kívül további memóriamennyiségek mérésére is képes. Az alábbi ábra bemutatja, milyen sokféle mennyiséget jeleníthetünk meg akár ezzel az egyszerű eszközzel is.

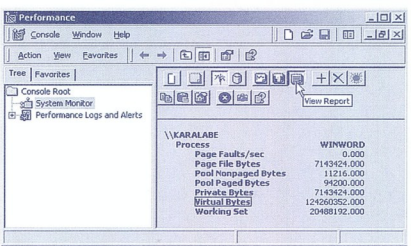


A Task Manager lehetőségei

Ha megjelenítjük mind a Memory Usage, mind pedig a Virtual Memory Size oszlopokat, érdekes felfedezést tehetünk: a két számoszlop között nem sok összefüggés mutatkozik, hol az egyik a nagyobb, hol a másik. Az alapnézetben is látható Mem Usage az alkalmazásnak juttatott fincsi fizikai memória, míg a VM Size az alkalmazás pagefile.sys-be kilapozott része. Ha pontosan tudni akarjuk az alkalmazás memóriaéhségét, fejben gyorsan összeadjuk a kettőt, s megkapjuk azt a számot - melynek semmi köze semmihez...
Ez azért van így, mert a korábbi, kilapozós ábra tanúsága szerint a végrehajtható kódrészek *(a kódszegmensek)* nem kerülnek ki a pagefile.sys-be, hanem „elpárolognak"!
Falba ütköztünk: nem vagyunk képesek megmondani egy alkalmazás összesített memóriaigényét – legalábbis a Task Manager nem segít ebben.
Térjünk át a sokkal kifinomultabb Performance Monitorra *(Windows 2000-ben System Monitor)*!

A Performance Monitor segítsége

You mean I can dodge bullets?
Első lépésként egyeztessük óráinkat, azaz állapítsuk meg, hogy az alábbi, Report nézetben használt PerfMon milyen számlálókat mutat a kiválasztott Process objektumon, s ezek vajon megegyeznek-e a Task Manager valamelyik számlálójával.



Mit mond a Performance Monitor a WINWORD.EXE-ről?

Rövid számológép-sanyargatás után rájövünk, hogy ami:

...a PerfMonnál..	a Task Managerben...
Page File Bytes	WM Size
Working Set	MEM Usage

Így csak öt eddig nem említett számláló maradt. Ezek közül három megjeleníthető a Task Managerben, de nem sok segítséget nyújtanak:

- A Pool Paged Bytes az a kilapozható memóriamennyiség, melyet az alkalmazás részére a kernelben tart fogva az operációs rendszer *(ablakkezelők (hWND) stb.)*
- A Pool Nonpaged Bytes az a NEM lapozható memóriamennyiség, melyet szintén az alkalmazás részére a kernel foglalt nekünk
- A Page Faults/sec a másodpercenkénti ki-be lapozások mérőszáma

Ismeretlenként tehát itt maradt a Private Bytes és a Virtual Bytes. Hátha ezek választ adnak a kérdésre: mennyit fogyaszt a WINWORD.EXE?

- A Private Bytes azon memóriamennyiség, mely az alkalmazás védett területén van, más processz nem férhet hozzá kívülről
- A Virtual Bytes pedig az alkalmazás által kért összes memória mennyisége

Hopp! Úgy tűnik, helyben vagyunk! A Virtual Bytes megválaszolja a kérdésünket! Lássuk csak: 140951552 bájt, ami annyi mint 134,42 megabájt. Uramisten! Csak nem kap ennyit egy nyavalyás word?

Konklúzió

Welcome to the real world!

Az élet szép, de a helyzet szomorú. A Virtual Bytes szépen megmutatja, hogy az adott alkalmazás mennyi memóriát KÉRT, de azt sajnos nem, hogy mennyit KAPOTT! *(Ez utóbbi lenne a Committed Bytes, lásd később.)* De legalább tetten érhető a virtuális memóriakezelés egyik igen jelentős előnye: az alkalmazások memóriaigénye akkor kerül kiszolgálásra, ha az általuk „lefoglalt" területre valóban szükségük lesz, oda írni, vagy onnan olvasni akarnak *(Demand Paged Virtual Memory)*. Ilyenkor Page Fault *(laphiány)* megszakítás keletkezik, s az operációs rendszer gyorsan odadob egy adag memóriát, hadd higgye azt a Word, hogy megkapott mindent, amit kért!

FONTOS!

Valójában minden processz 0 (nulla) méretű Working Settel indul. Az alkalmazás mintegy „befaultolja", „behibázza" magát a memóriába, mert kezdetben nem kap egy fikarcnyi RAM-ot sem (kivéve az EXE legelejét), és ahogy futni kezd, csapkod ide-oda, mindannyiszor üres lapot talál, amit a VM kezelő bepótol neki!

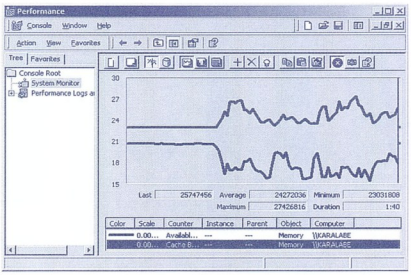
Ez a Windows 2000-nél még elég bután megy, mert - bár minden egyes alkalmazás mindig ugyanúgy indul - a 2000 nem adja neki oda az indulólapokat, az EXE lassan „befaultolódik". Ez magyarázza, hogy például egy vacak kis CALC.EXE közvetlenül indítás után miért mutat 334 Page Faultot, pedig senki sem bántotta!

A Windows XP-n gyorsabban fognak indulni az alkalmazások, mert meg fogja tanulni az induláskor szükséges lapokat, leteszi egy .PF *(PreFetch)* fájlba, és második indulásnál már egy használható készletet tesz RAM-ba; nem nulláról kell bemásznia szegény alkalmazásnak.

A lapozásról

How deep the rabbit hole is?

A Process objektum számlálóival tehát nem jutottunk a dolog végére. Nem tudjuk, hogy egy adott pillanatban mennyit fogyaszt egy alkalmazás, sőt, azt sem tudjuk pontosan, miért nem tudjuk, amit nem tudunk. Térjünk át most a Memory objektum számlálóira, hátha így közelebb jutunk a működés lényegéhez. Itt van mindjárt az Available Bytes és a Cache Bytes számláló. Ha egy gépet terhelésnek teszünk ki, ez a két számláló gyönyörű szimmetrikus ábrákat rajzol a képernyőre.



Available Bytes és Cache Bytes a PerfMonban

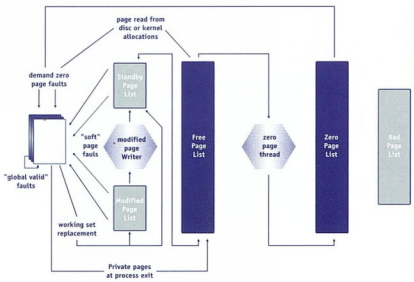
Mintha látszólag minden bájt, mely lefoglalódik, a cache területre kerülne, és fordítva. Mintha soha semmi nem kerülne át például a programok hatáskörébe. *(Persze átkerül, ha indítgatunk és leállítgatunk programocskákat, de ha csak a meglévőkkel manipulálunk, ez ritkán látszik.)* Mi valójában az „Available Bytes"? És mi a „Cache Bytes"?

Kövesd a fehér rabbit

Az operációs rendszer futása közben állandóan zajlik a lapozás. Ennek oka, hogy minden processznek korlátos a Working Set mérete, s még rengeteg fizikai memória esetén is előbb-utóbb utóléri a végzet: bizonyos lapokról le kell mondania. Azonban ettől még nem fog zörögni a merevlemez. Ugyanis a Working Setből *(LRU algoritmussal)* kilapozott blokkok általában nem a merevlemezre lapozódnak, hanem előbb az úgynevezett Standby *(rendelkezésre állási)* memórialistára kerülnek. A Standby területen a memóriablokkok változtatás nélkül tárolódnak, hátha az LRU tévedett, és hamarosan ismét szükség lesz rájuk, így innen a Working Setből kilapozott blokkok mindenféle merevlemez-tekerés nélkül visszalapozhatók. A memóriából memóriába történő lapozást Soft Page Faultnak hívjuk, ellentétben a valóban lapozófájlműveletel járó Hard Page Faulttal.

A Task Manager és a PerfMon->Process objektum nem képes különbséget tenni a Hard és a Soft Page Fault között, így azok a számlálók gyakorlatilag használhatatlanok a lapozófájlhasználat felbecsülésére! Egyedül a PerfMon->Memória objektum ad valós képet a Pages/sec számlálóval, mert az csak a Hard Page Faultot méri

A Standby listán kívül további memórialistái is vannak az operációs rendszernek, amint az az alábbi, David Solomontól lopott ábrán látható:



A Windows memórialistái

A Working Setből a 4 kilobájtos lapok annak megfelelően szorulnak ki vagy a Standby Listre, vagy a Modified Page Listre, hogy tartalmuk módosult-e – azaz kód, vagy adatszegmensről van-e szó. Ha egy alkalmazásból kilépünk, annak összes memórialapja a Free Page Listre kerül, felszabadul. Biztonsági okokból a Free listáról kizárólag olyan processz kaphat lapot, akinek amúgy joga lenne az adott lap olvasásához. Mivel azonban a kilapozott blokkban jelszavak, RSA kulcsok és egyéb érzékeny adatok lehetnek, a lapok törlésen esnek át, mielőtt akármelyik processz kaphatna belőlük. Így kerülnek át lenullázva a Zero Page Listre. Ha a szabad memória mennyiségére vagyunk kíváncsiak, nehéz helyzetben vagyunk, mert míg a Zero Page List nyilván szabad memória, addig a Free és a Standby így is, úgy is értelmezhető: ha visszalapozzuk eredeti helyére, akkor inkább „cache", ha viszont újra kiadjuk, akkor szabad...

A Task Manager és a PerfMon által mutatott Avaliable Bytes *(a fenti gyönyörű diagramon az alsó vonal)* valójában a Free, a Standby és a Zeroed listákon lévő memóriablokkok összes területe!

Már csak egyetlen dolgot kell megválaszolnunk: mi a Cache Bytes?

Cache Bytes

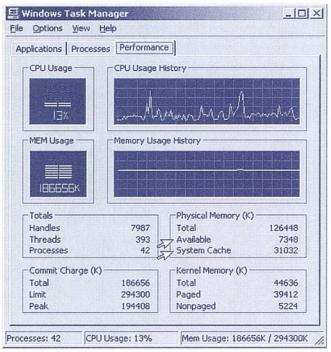
There is no spoon

A PerfMon szerint: „Cache Bytes is the sum of the System Cache Resident Bytes, System Driver Resident Bytes, System Code Resident Bytes, and Pool Paged Resident Bytes counters." Vagyis mindenféle System vicik-vacak által elfoglalt memóriaterületek összessége!

Ennek magyarázata a következő: nincs is olyan memóriatípus a Windowsban, hogy cache, mert a fenti listák gyakorlatilag elvégzik a gyorstárazást. Cache mechanizmus persze van: a Read Ahead, Lazy Write és a többi jól ismert algoritmus itt is megvan, de nem egy különálló cahce managerben, hanem a VM memóriakezelés részeként. A trükk a következő: ha egy alkalmazás beolvas egy nagy fájlt, akkor a Read Ahead nem a hívó processz memóriaterébe dobálja az előrefutó olvasás eredményét, hanem az operációs rendszer saját Working Setjébe, hogy ha esetleg rosszul „gondolkodott", és a Read Ahead eredménye mégsem kell az alkalmazásnak, ne kelljen külön eltávolítania az alkalmazás memóriateréből a kéretlen cuccot. E tény ismeretében érthető, hogy a Task Manager által mutatott Cache Bytes mást tartalmaz NT4 és Windows 2000 esetén – hisz egyiknek sincs semmi köze a valósághoz. A „File Cache":

- NT4 esetén valójában a system working set mérete *(paged pool + NtosKrnl.Exe, az eszközmeghajtók kód- és adatszegmensei stb.)* Semmi köze semmihez!
- Windows 2000 esetén az NT4-es zagyvaság, plusz a Standby List mérete.

A fura az, hogy a Standby List mérete beleszámítódik az Available Bytes-be is! Az Available és a Cache tehát sziámi ikrek, melyek a Standby Listnél fogva össze vannak nőve :)



A Task Manager buta arca. A dupla kurzor a sziámi számlálókat mutatja

A Commit Charge pedig a lapozófájl méretéről ad közelítőleges információt: A Memory->Committed Bytes számláló ugyanis azt a mennyiséget mutatja, amennyit a Windows a kért memóriából valóban kiosztott, s melynek számára a lapozófájlban fészket is rakott, hogy ha majd kilapozódik, ne kelljen helykeresgéléssel bajlódnia. Milyen kár, hogy a Committed Bytes csak a Memory objektumon mérhető, s a processzeken nem!

Fóti Marcell
marcellf@netacademia.net

Whistler *(III. rész)*

Idén már két számunkban is éltünk a kiszivárogtatás populáris eszközével, de harmadszorra sem tudtam ellenállni a csábításnak, ezért íme egy újabb bennfenntes beszámoló a Windows következő változatának, a Whistlernek újdonságairól.

Active Directory javítások

Az Active Directoryban összesen több, mint 400 módosítás történik. Sémabővítés nem sok lesz, ugyanis kompatibilis marad a jelenlegi Windows 2000 sémával. Ami viszont a sémamódosítások visszavonhatóságát illeti: nos, ilyet az eredeti Windows 2000 nem tud végrehajtani, így vegyes *(2000 és Whistler)* tartományokban nem élvezhetjük az utóbbi összes újdonságát. Tisztán Whistleres környezet kell mind a négyszázvalahány újdonság használatba vételéhez.

A szituáció ismerős, szinte déjá-vu: mint tudjuk, a Windows 2000 Active Directory összes képességét nem használhatjuk vegyes *(NT4 és 2000)* tartományban. Csak akkor hullanak le a rabláncok a W2K AD-ról, ha az összes tartományvezérlő kétezres verziójú ÉS átállítjuk a tartomány Mixed-ről Native üzemmódra. A Whistler esetében is hasonló módon zajlik majd a frissítés. Miután az összes tartományvezérlő átállt Whistlerre, engedélyezni lehet a turbofícsöröket, át lehet állítani a tartomány Functionality Leveljét 0-ról 1-re. A nullás szint a Windows 2000, az egyes a Whistler, a kettes pedig a még fejlesztési stádiumban sem lévő 2005 utód lehet. *(Látható, hogy most okosabb a Microsoft, mint két évvel ezelőtt, mert tetszőleges számú verzióváltást lehetővé tevő lehetőséget adtak maguknak a Functionality Level bevezetésével, szemben a Mixed->Native megkülönböztetéssel, melynek lehetőségei máris kimerültek.)* A Functionality Level átállítása a jelenlegi tervek szerint a sokak által utált, parancssori NTDSUTIL segítségével történik majd.

Szintén ide kapcsolódik az az új tartományvezérlők távoli telepítését megkönnyítő lehetőség, hogy a születő DC a kezdeti replikációs feltöltést nemcsak hálózaton át lesz hajlandó befogadni, hanem *(szalagos)* mentésről is. Ezáltal lassú vonalak túlvégén is bátran belevághatunk majd a DCPROMO-ba.

Active Directory Migration Tool

Ha a tervek szerint halad a munka, az ingyenesen letölthető tartományátszervező eszköz, az ADMT is okosodik egy nagyot. Eddig is lehetett vele objektumokat mozgatni tartományok között, de sajnos a jelszavak átvitelére nem volt képes. Az új verzióban állítólag megoldják ezt a nem kis problémát.

64 bites Windows

A Whistler lesz az első Microsoft operációs rendszer, mely átlép a 64 bites világba. Talán itt lenne az ideje a többi, szokásos korlát átlépésének is. A BIOS 7,8 gigabájtos álomhatárától éppúgy megszabadulhatunk végre, mint a 2 terabájtos maximális partícióméretről, mert magától a partíciós táblától válunk meg végre-valahára. MBR helyett jön a GPT, azaz GUID Partition Tables. Már megint dobhatjuk ki jól bevált lemezkezelő programjainkat...

Rollback Driver

Biztosan sokan belefutottak már hibás eszközmeghajtó okozta problémákba. Az új driver lecseréli a régit, és felülfirkantja a registryt. Már az NT korábbi verziói is lehetővé tették a regisztrációs adatbázis módosításainak visszavonását az úgynevezett Last Known Good indítás segítségével, ám ez a lépéssorozat nem állítja vissza magát az eredeti eszközmeghajtó fájlt *(*.SYS)*. Ha tökéletesen szerencsétlenek vagyunk, a frissítési szándékkal telepített változat által letörölt korábbi, működő SYS volt az utolsó fellelhető példány a földkerekségen. Már nem gyártják, s a weben sincs hozzá eszközmeghajtó.

A Whistler azonban nemcsak Update Driver, hanem Rollback Driver nyomógombbal is dicsekedhet, mert új meghajtó telepítésekor nem letörli, hanem elraktározza az előző verzió*(ka)*t.

DNS forwarding

A DNS nélkülözhetetlen szolgáltatás az Active Directory alatt, így a vállalat összes tartományi gépe azt a DNS kiszolgálót használja, amelyiken az AD regisztráció van, különben nem működne a tartomány. Ez azzal jár együtt, hogy minden DNS kérés - beleértve az Internetes névfeloldási műveleteket is - ugyanezen a DNS kiszolgálón köt ki, s az továbbítja a megfelelő helyre. Illetve nem a megfelelő, hanem *(és ez nagy különbség!)* a beállított helyre. Elágazást nem lehetett eddig megvalósítani, így néha igen trükkösen kellett felfűzni a vállalat DNS Servereit egy elágazásmentes Forwarder láncba. A Whistler lehetővé fogja tenni, hogy FQDN alapján szétválogatva kerüljenek továbbításra a kérések. A belső kérések például a junixra, az Internetesek pedig az Internetszolgáltatóhoz.

NTFS Permission Calculator

Megkaptuk amit kértünk: tíz év vajúdás után megszületett az effektív jogokat kiértékelő eszköz. Csak beírjuk neki, hogy melyik fájlon, és kinek a jogait szeretnénk megtudni, és már indul is a folyamat: allow + deny + örökölt jogok + ownership + csoporttagság + rendszerszintű jogok *(pl. back up files and directories)* stb. beszámításával megkapjuk a kiszemelt felhasználó jogait egy adott objektumon. *(A Novell NetWare 3.11 nyomán...)*

Fóti Marcell
(már) MCSE 2000

További információ:
www.microsoft.com/hwdev/storage

RRAS (II. rész) – útválasztási alapok

Emlékszem, pár évvel ezelőtt az útválasztás nemigen okozott fejtörést az IT szakemberek többségének: a hálózatok összekapcsolása *(ha voltak egyáltalán hálózatok!)* még nagyvállalatoknál is ritkaságszámba ment, s Internetkapcsolata sem volt senkinek. Nem túlzok: senkinek! De elmúltak már azok a boldog békeidők, amikor hálózat alatt egy másfél méteres koax kábelt, két T dugót és két lezáró ellenállást értettünk, s annak is vége, hogy „majd a Novell két kártyával elintézi", hisz az IPX protokoll is kihalóban van. Nézzünk szembe a tényekkel: a hálózatok rohamos burjánzása következtében a rendszergazdák folyamatosan foglalkoznak a problémával, s heti gyakorisággal kerülnek döntéshelyzetbe: kell-e új subnet? S ha igen, milyen IP tartományt kapjon? Mi fogja összekötni az új alhálózatot a régivel? Egy router? Milyen gyártmány? Milyen routing protokollt kell ismernie? Mennyibe kerül?

Minden döntési helyzetben célszerű legelőször megvizsgálnunk a szerszámosládát, hátha már birtokunkban van az a szerszám, amire éppen szükség van. Az RRAS szerszámosládája igen gazdag. Nem mindenki tudja talán, de egy középkategóriás hardver útválasztót meghazudtoló funkciógazdagságú és teljesítményű eszköz van a birtokunkban! Ismerkedjünk meg a használatával, hátha beválik. A ládafia átkutatása előtt azonban ismerkedjünk meg az IP útválasztás gyönyöreivel és gyötrelmeivel.

Az alapprobléma

Miért kell egyáltalán törődnünk az útválasztással? Miért nem oldják meg a gépek önállóan ezt a feladatot? Mert a két pont közötti csomagátvitelért felelős IP protokoll nincs felkészítve ilyesmire *(!)*, lehetőségei nagyon is korlátozottak: egyszerűen nem is teszi lehetővé egynél több hálózat használatát! Az eredeti IP specifikáció ugyanis két, egymással szorosan összefüggő korlátozást is tartalmaz egynél több hálózat esetére:

1. szabály: Minden gép csak a saját hálózatára küldhet csomagot.

2. szabály: Ha mégis távoli hálózatra kellene küldeni a csomagot, akkor az első szabály lép életbe.

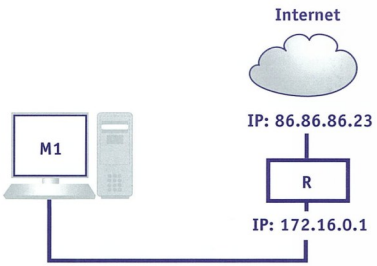
Tisztára, mint a „férjnek mindig igaza van" feliratok az ajándékbolt falán. Vagy mint egy börtön: minden alhálózat egy-egy külön börtön.

3. szabály: A rabok kintről csak az őröktől kaphatnak csomagot.

4. szabály: Ha mástól jönne a csomag, akkor az első szabálylépne életbe.

Egymással természetesen szabadon, a fegyőr közvetítése nélkül is kommunikálhatnak: arra való a radiátorcső :-)

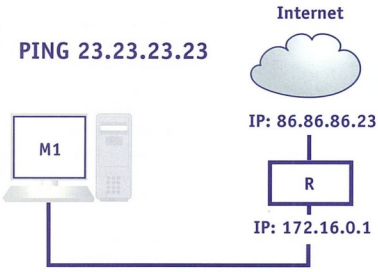
Az örök „segítsége", közreműködése nélkül nincs lehetőség külső kapcsolatra. Ugyanez a helyzet az IP-vel. Amint külső hálózatra forgalmazunk, rá kell vennünk a fegyőrt, hogy segítsen. Mindenképpen szükségünk van egy segédre, egy postásra, egy útválasztóra. Egyszerű esetben alhálózatonként egyetlenegy „fegyőr" van - ennek címét írjuk be a Default Gateway, vagy magyarul alapértelmezett átjáró helyére.



IP: 172.16.0.22
SM: 255.255.255.0
DG: 86.86.86.23

Mi a hiba a fenti ábrán? (Rossz a DG.)

Gyakori hiba, s MCP vizsgákon számtalan ravasz formában visszatérő feladat a Default Gateway helyes beállítása. A fentiek alapján egyértelmű, hogy ha egy számítógépen a beírt Default Gateway nem a saját routerünk innenső IP címe, hanem mondjuk például a túlsó lábát, vagy ad abszurdum az Elender egyik gépét adjuk meg *(mondván, hogy az közelebb van az Internethez, hű de jó gyors lesz a kapcsolat)*, akkor a gép nem lesz képes külső kapcsolatot építeni. Idegen nem állhat szóba velünk a saját fegyőrünk közvetítése nélkül! Nincs csalási lehetőség: ha nem él, vagy nem ismert a kapu őrzője, nem jöhet létre a kommunikáció a külső és belső felek között. A fegyőrös hasonlat egyetlen ponton sántít: ha hibás címet adunk meg alapértelmezett átjáróként, akkor nem a fegyőr állítja meg a forgalmat - odáig el sem jutunk. Az IP stack helyes átjáró hiányában helyben eldobja a kifelé irányuló csomagokat. A szétválasztás egyébként a saját és a címzett IP címéből képzett úgynevezett Network ID-k összehasonlításán alapul: ha a két NetID azonos, közös alhálón vagyunk. Ha nem - nem. A művelet a saját Subnet Mask segítségével történik olymódon, hogy a Maskkal kétfelé vágjuk a címeket, és ha a „bal" fele *(Network ID)* azonos a rab börtönazonosítójával, akkor irány a radiátor. A következő ábra az előző alapján készült, ahol immár helyes a DG címe, és megpróbáljuk megpingelni a 23.23.23.23 című gépet. Az ábra alsó felében látható, ahogyan a gép eldönti, vajon azonos hálón van-e a végállomással.



IP: 172.16.0.22
SM: 255.255.255.0
DG: 172.16.0.1 ← most jó!

nyissz!

Network ID	Host ID
172. 16. 0	22
23. 23. 23	23
255.255.255	0
Nem egyforma!	

Az útválasztás módszere: az IP címek szétvágása a Subnet Mask segítségével.

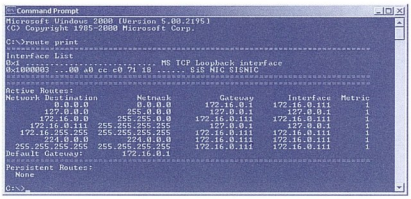
Itt jegyzem meg, hogy manapság minden Subnet Mask balról csupa egyes, jobbról csupa nulla, például:

```
255.255.192.0=11111111.11111111.11000000.00000000
```

Az eredeti szabvány megengedett „fésűs" maszkot is, melyben vegyesen szerepelhettek egyesek és nullák. Ezt egy későbbi RFC felülbírálta, valószínűleg azért, mert nem embernek való a vegyes maszkkal történő fejben számolás.

Minden Windows router?

A legelső szabálypár miatt minden fegyencnek el kell tudnia dönteni, hogy kommunikációs partnere belsős, vagy külsős. Ezen döntések meghozatalára minden IP alapú eszköznek szüksége van, azaz mindegyik IP eszköz egy-egy egyszerű útválasztó is egyben. Erről könnyen meg is győződhetünk, ha parancssorban kiadjuk a ROUTE PRINT parancsot. Az alábbi ábrán például egy Windows 2000 Professional *(!)* útvonaltábláját láthatjuk, ennek sorai egy-egy feldolgozandó irányt jelölnek. Későbbi példáim könnyebb értelmezhetősége miatt célszerűnek tartom a táblázat részletesebb elemzését - hamarosan route tábla műtétre is sor kerül!



A Windows 2000 Professional útvonaltáblája

A táblázat első két oszlopában a célhálózatokat olvashatjuk, a szokásos formában:

IP címtartomány eleje	Subnet Mask
199.188.177.0	255.255.255.0

Ez a két szám egyértelműen meghatároz egy IP tartományt. Könnyedén kiszámítható a vége, a Subnet Mask ugyanis megadja, hogy hány IP cím esik ebbe a tartományba: ahol nulla áll a maszkban, az mind ide tartozik *(az a szabadságfokunk)*, ergo az utolsó IP cím ebben a kupacban a 199.188.177.255. Újabb dokumentumokban egy másik jelölésmód is előfordul:

```
199.188.177.0/24
```

A törtjel utáni szám megadja, hogy a Subnet Maskban hány egyes *(1)* bit van, a 24 tehát megegyezik a 255.255.255.0-val. A fenti ábrán látható útvonaltáblában tehát a következő hálózatok szerepelnek:

IP	SM	Hálózat
0.0.0.0	0.0.0.0	Default Gateway
127.0.0.0	255.0.0.0	SELF
172.16.0.0	255.255.0.0	Saját alháló
172.16.0.111	255.255.255.255	Ez a gép
172.16.255.255	255.255.255.255	Subnet Broadcast
224.0.0.0	224.0.0.0	Multicast
255.255.255.255	255.255.255.255	Globális IP Broadcast

A következő két oszlop a küldési MÓDSZER meghatározására való. Alapvetően háromféle módszer létezik:

Ha a 3. oszlop...	...a továbbítás
egy „idegen" IP cím a saját hálónkról (Itt: 172.16.0.1)	Célzott. A csomag az „idegen" (router, fegyőr) címre kézbesítendő
saját címünk (Itt: 172.16.0.111)	Üzenetszórás. Az Ethernet majd lerendezi.
127.0.0.1	Lokális. A csomagot nem KI, hanem BE kell küldeni az oprendszerbe.

Az Interface oszlopban a fenti három továbbítási mód kijáratait, hálókártyáit láthatjuk, végül a Metric oszlop egyelőre nem fontos, majd ha jönnek a Routing protokollok, visszatérünk rá.

```
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    172.16.0.1    172.16.0.111       1
        127.0.0.0        255.0.0.0      127.0.0.1      127.0.0.1       1
       172.16.0.0      255.255.0.0    172.16.0.111   172.16.0.111       1
     172.16.0.111  255.255.255.255      127.0.0.1      127.0.0.1       1
  172.16.255.255  255.255.255.255    172.16.0.111   172.16.0.111       1
      224.0.0.0        224.0.0.0    172.16.0.111   172.16.0.111       1
  255.255.255.255  255.255.255.255    172.16.0.111   172.16.0.111       1
```

Mi hova?

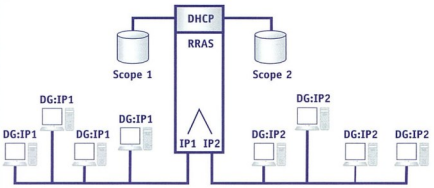
A kiemelt sor értelmezése ezek szerint: a 172.16.0.111 címre „induló" *(valójában érkező)* csomagokat a 127.0.0.1 *(SELF)* címre kell továbbítani, tehát be a gépbe. De az is kiolvasható, hogy ha a célgép IP címe mondjuk 193.193.193.193, akkor ... hogyis? Meg kell keresnünk a 193.193.193-as subnetet a legelső oszlopban, s ennek sorából kiderül, hogy hova kézbesítendő. Nos, 193-as sort nem találunk, de ott a 0.0.0.0, mely minden ismeretlen címre vonatkozik, „mindent visz": a 193-as címek, mint teljesen ismeretlenek, célzottan a 172.16.0.1-es címre továbbítódnak, ami nem más, mint a DG. Ezzel a trükkel megspórolhatóvá vált, hogy a világ összes IP tartományát fel kelljen sorolni a listában: ami nincs bent, azt lenyeli a 0.0.0.0 sor.
A saját hálózatra való csomagok célcíme egészen másképp fest: mintha önmagunknak küldenénk *(127.0.0.1 <> 172.16.0.111)*! A paraméter helyes értelmezése: ezen a címen/kártyán kell kieregetni, de nem kell vele „célozni", az Ethernet majd elintézi a célba juttatást *(üzenetszórás, ARP stb.)*.
Fontos még ismerni a 127.0.0.1 címet: ezek megint mi vagyunk, ez a speciális IP cím minden gépen önmagára mutat. A sor értelmezése: ami nekünk jött, az nekünk jött. Nem kell továbbküldeni. Ha megpingeljük ezt a címet

```
PING 127.0.0.1
```

Network Monitorral megfigyelhető, hogy a csomag ki sem jut a kábelre *(nem látszik belőle semmi)*. Most lássuk, mit kell tennünk az útvonaltáblával a következő esetekben!

Két hálózat összekapcsolása

Ha mindössze két hálózatot *(mondjuk két irodát)* szeretnénk összekapcsolni útválasztóval, nem kell törődnünk sem az RRAS útvonaltáblájával, sem a telepítővarázslóval. A lényeg, hogy az RRAS routing engedélyezve legyen. Ilyenkor elegendő, ha az RRAS gépbe beteszünk két kártyát, azokon beállítunk két helyes IP címet, a két oldal munkaállomásainak DG-jét pedig egyszerűen az RRAS-ra irányítjuk. Ez azért elegendő, mert bár a munkaállomások „ismeretlen" cím miatt, a 0.0.0.0 címre küldik a csomagokat, a középen álló RRAS mindkét hálózatról tud, így minden beérkező csomagot ügyesen átemel a másik lábára.



Két alhálózat összekapcsolása

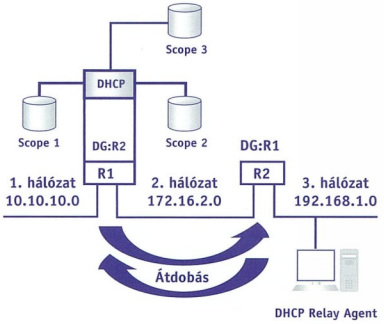
Még egyszerűbb a hálózat gépeinek felkészítése, ha az RRAS-ra felteszünk egy DHCP Servert, és mindkét kártya IP címtartományára felveszünk egy scope-t *(IP cím készletet)*, ahol a DG mindkét szkópban a megfelelő RRAS lábra mutat. Többkártyás gépen ennél többre nincs is szükség, mert a DHCP Server meg tudja különböztetni egymástól a bejövő kéréseket, és automágikusan a megfelelő tartományból ad címet.

Három alhálózat összekapcsolása

Három hálózat láncba fűzése esetén azzal kell számolnunk, hogy a lánc végén tanyázó routerek már nem fogják ismerni a kettővel odébb található alhálózatot, mert oda nincs közvetlen hálózati kapcsolatuk. Lássunk példát!

Három alhálózat esetén már lesznek a routerek számára ismeretlen, „távoli" hálózatok is

Itt R1 számára a 3. hálózat közvetlenül már nem érhető el, ismeretlen. Első ránézésre úgy tűnik, hogy elkerülhetetlen R1 és R2 felokosítása, útvonaltáblájuk felkészítése a távoli célok eléréséhez, hisz ha az 1. hálózat egyik munkaállomása R1-hez továbbít egy csomagot, melynek végcélja a 3. hálózat, akkor R1 csak néz bután, nem tudja merre van a tovább.
Egy kis furfanggal azonban elodázhatjuk az útvonaltábla hekkelését. Mit is kell csinálni minden ismeretlen csomaggal? Hozzávágni a DG-hez! Mi tiltja meg, hogy R1-nek is beállítsunk DG-t? A következő ábrán a görbe nyilak a csomagátdobás útvonalát mutatják, ha mindkét routeren felvesszük a MÁSIKAT DG-nek. Fűszerezzük meg az egészet egy kis DHCP-vel is, és igazán könnyen kezelhető hálózathoz jutunk!



Három hálózat,routolás és DHCP

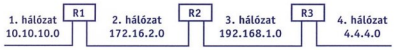
A fenti ábrán a DHCP Servernek már három szkópja *(IP címkészlete)* van, de csak kettő alhálózat gépei képesek közvetlenül címet kérni tőle. A harmadik hálózaton ilyen esetekben el lehet helyezni egy DHCP Relay Agentet, mely onnan a DHCP címkérési broadcastokat unicasttá alakítva átdobja a routeren túlra, majd a választ ismét broadcasttá alakítva visszajuttatja a kérelmezőhöz.

Alapprobléma II. - gép a köztes hálózaton

Három alhálózatból álló környezetben már előáll az a probléma, amit az augusztusi NetMon cikkben részletesen elemeztem: a középső, második hálózaton mi legyen a munkaállomások DG beállításával? Ha R1, az is rossz, ha R2, az is rossz, hisz mindkét esetben menthetetlenül duplázott hálózati forgalom kerekedik, amint az „ellentétes" irányba szeretnénk küldeni valamit. Egyszerre kettő DG nem lehet, mert - mint emlékszünk - hiába adunk meg egynél többet, hármat vagy négyet, sajnos már a másodikat sem használja a masina mindaddig, amíg az első elérhető. Emlékszik még valaki, mi erre a megoldás? A köztes gép útvonaltáblájának kiegészítése, amit okos routerek esetén maga az útválasztó elintéz, ICMP Redirect üzenetek kiküldésével. A II. alapprobléma sokszorosan visszaköszön, ha hálózatunk egyre több alhálózatból épül fel, és eljön az a bonyolultság, amikor az ICMP Redirect nem oldja meg a problémát. Nem marad más hátra, hozzányúlunk az útvonaltáblákhoz...

Négy hálózat összekapcsolása

Ahogy bonyolódik a hálózat, úgy válik egyre nehezebbé pusztán ravaszsággal megúszni az útválasztótábla matatását. Ha már négy hálózat felfűzésére van szükség...



Négy alhálózat összekapcsolása

...akkor már nem segít a DG-trükk, ugyanis a középen álló R2-nél a „gép a köztes hálózaton" esete forog fenn, és két DG-t kellene beállítani neki, ami nem megy. Nézzük meg, vajon R1 és R3 esetén beválik-e még a DG-csel:

- R1 ismeri az 1. és a 2. hálózatot.
- Ha minden további ismeretlen csomagot gondolkodás nélkül áthajít R2-nek, akkor jól továbbítja a 3. és 4. hálóra való csomagokat!
- Ennek tükörképe az R3, mely így szintén jól működik, ha a DG-je „középre", R2-re mutat.

Az R2 tanítása viszont elkerülhetetlen. Ismerkedjünk meg a Route parancs használatával!

ROUTE ADD	Új bejegyzés készítése az útvonaltáblába
ROUTE CHANGE	Meglévő bejegyzés módosítása
ROUTE DELETE	Sor törlése a táblából

Így taníthatjuk meg R2-nek, merre van az első hálózat:

```
route add 10.10.10.0 mask 255.255.255.0 172.16.2.33
```

Ahol a 172.16.2.33 cím R1 „innenső" lába. Hasonlóképpen tanítható meg neki a negyedik hálózat holléte:

```
route add 4.4.4.0 mask 255.255.255.0 192.168.1.111
```

ahol 192.168.1.111 az R3 „innenső" lába. Szabadon kísérletezhetnek Kedves Olvasóink a route tábla rongálásával, mert egyfelől mindig rendelkezésre áll a ROUTE DELETE, másfelől minden bajtól megszabadít egy jóízű reboot. Így nyugodtan kipróbálható, mit szól a masina, ha kiadjuk a következő parancsot:

```
ROUTE DELETE 0.0.0.0
```

Megszűnik a távoli hálózatok elérhetősége, de a dolog nem halálos, a

```
route add 0.0.0.0 mask 0.0.0.0 172.16.0.1
```

visszateszi a DG címet *(Ahol a 172.16.0.1 a Kedves Olvasó alhálózatán lévő router címe)*!

N darab hálózat összekapcsolása

A következő megoldandó probléma N darab alhálózat összekapcsolása. Az eddigiekből könnyen belátható, hogy a végeken található routerek esetén van némi remény DG-trükkre, de csak addig, amíg azok összesen két hálózat összekötését végzik. Ha elkezdünk pókhálót alkotni, azonnal elillan ennek esélye, s marad a tanítás. Sajnos a routerek és útvonalak számával exponenciálisan növekszik a szükséges ROUTE ADD parancsok száma, és akkor még nem beszéltünk a hálózat önkarbantartásáról. Körülbelül négy hálózatig érdemes kézzel bíbelődni az útvonaltáblákkal, ennél összetettebb hálózatok esetén kézimunkázni őrültség.

A megoldást valamilyen útválasztó protokoll használata jelentheti, melyek közös tulajdonsága, hogy az egyes routerek által természetesen ismert hálózatok adatait automatikusan átvezetik a többi routerre is. Természetesen ismert hálózatnak nevezem azokat, melyek közvetlenül a masinába csatlakoznak: nem kell megtanítani egy routert olyan hálózat elérésére, mely közvetlenül csatlakozik hozzá!

Két útválasztó protokollról szólunk a következő hónapban: a RIP és az OSPF-ről, sőt, megemlékezünk a hamis IP címek által okozott útválasztási agyrémekről is.

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

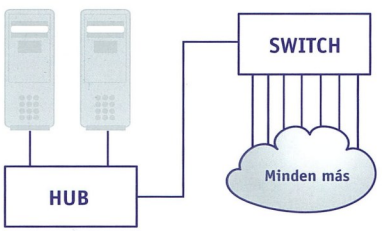
Netmon (IX. rész): Jegyzetek a tűzvonalból

Egy probléma megkeresése és megoldása nem túl hálás feladat. Arról szól, hogy valami nem úgy működik, mint a nagykönyvben, mi megkeressük a hibát, azután meg minden úgy pöfög, ahogy kell. Elpocsékolunk egy csomó időt arra, hogy úgy működjenek az eszközeink, ahogyan maguktól is kellene. Néha csak egy-két órát kell rááldozni a feladatra, néha egy hetet, de igazából sohasem tudjuk az elején, hogy mennyi lesz a feláldozandó, „kárba vesző" idő. Azért lássuk be: nagyon sok „misztikus" eseménnyel találkozunk „kiszámítható" számítógépes világunkban, és ez felettébb bosszantó. Főleg akkor lehet dühös az ember, ha úgy érzi, nincs a kezében eszköz, amellyel a probléma felgöngyölíthető lenne. Logikai problémakizárással, próbaszerencse elven, vagy „kezdjünk mindent tiszta lappal" módszerrel dolgozunk ilyenkor, és csak reméljük, hogy sikerül. Már azt sem bánjuk, ha nem tudjuk meg a hiba pontos okát, csak legalább működjön már az a fránya rendszer.

Az alábbi eset megtörtént, és - okulására mindenkinek - szeretném megosztani, hogy akik nap mint nap a „tűzvonalban" tevékenykednek, azokat bátorítsam a problémák újfajta megközelítésére.

A környezet és a probléma

Pár héttel ezelőtt üzembehelyeztünk egy Windows 2000 szervert, rajta egy terminál szerviz szolgáltatást. Az eszközt hasznossága és népszerűsége miatt hamar túlterhelték a felhasználók – nosza, itt az alkalom egy hálózati terhelésmegosztás *(Network Load Balancing Service)* életre keltésével



egy másik szervert is beállítani. Némi elméleti kérdés tisztázása *(unicast és multicast üzemmód, MaskSourceMAC és switch elárasztás probléma stb.)*, olvasgatás és próbálkozás után felállt az NLBS cluster a következő konfigurációban: két Windows 2000 Advanced Server NLBS-sel telepítve és egy hubra kötve. A szerverek multicast üzemmódban működtek. *(Csak dióhéjban: Az NLBS kétféle üzemmódban képes működni: unicast illetve multicast módban. Az unicast és multicast itt „level 2" szinten értendő, és nem IP szinten. Unicast esetén a szerverek elrejtik a saját MAC címüket, és egyetlen, közös MAC címmel jelennek meg. Ennek a megoldásnak előnye a kompatibilitás, hiszen egy unicast csomag a mindenki által ismert normál csomag. Hátrány viszont, hogy a szerverek az azonos MAC cím miatt nem tudnak egymással kommunikálni, ezért gyakran egy második hálózati csatolót is tesznek a szerverekbe, amellyel a node-ok közti forgalom lebonyolítható. További hátránya az unicast üzemmódnak, hogy a switcheket zavarja, ha két portján is ugyanaz a MAC cím jelenik meg. Ezt ugyan el lehet rejteni egy registry beállítással (MaskSourceMAC=1), ekkor viszont a switch egyáltalán nem tudja megtanulni a MAC címet és minden switchportot elároszt forgalommal. A multicast üzemmód esetén az eredeti MAC cím megmarad, és „megállapodás születik" egy közös MAC címről is. Így elkerülhető a két hálókártya telepítése és a switch elárasztása is, kommunikálhatnak egymással a szerverek, viszont a multicast forgalom kompatibilitási gondokhoz vezethet elsősorban a routereknél.)* Az 1. ábrán látható módon felállt a rendszer és ment is, ahogy az a nagykönyvben meg volt írva.

Aztán két hét múlva elromlott az NLBS. A jelenség nagyjából úgy festett, hogy a kliens megpróbált csatlakozni, már majdnem sikerült, feltűnt a halványkék bejelentkező képernyő, majd hirtelen megszakadt a kapcsolat, pontosabban a szerverek megszakították. A szolgáltatás pedig már épp eléggé fontossá vált ahhoz, hogy meg kelljen oldani a feladatot.

Problémamegoldás kizárásos módszerrel

Ahogy ez lenni szokott, először a problémát szerettük volna elhatárolni. Mindezt persze „takarékosan". Ugyan alapszabály, hogy egyszerre csak egy dolgot változtass, de ez annyira lelassíthatja a problémamegoldást, hogy a gyakorlatban ritkán van türelme a rendszergazdának alkalmazni. Mi is eltértünk néha a szabálytól. Biztosan az NLBS-sel van probléma - gondoltuk, mert a szerverek terminál szolgáltatása elérhető volt, ha a saját nevük, és nem a clusternév alapján akartuk elérni. A switch és a hub nem lehetett hibás, hiszen akkor forgalom sem lehetett volna. S valóban, unicast üzemmódra átkapcsolva, megjavulni látszott a helyzet. Csakhogy így elveszítettük a lehetőségét, hogy egyik node-ról a másikat el lehessen érni, ezt pedig nem akartuk. Mindenképp meg kellett oldani a multicast üzemmódú működést.

Kezdjünk mindent tiszta lappal!

A teljes újratelepítést a végső időkre tartogatva, először csak az NLBS-t vettük le, és telepítettük újra – az eredmény változatlan. Vagy nem tudja megjavítani a telepítő a jelenséget, vagy az NLBS-nek nincs köze az egészhez, ezért nincs is hatása az újratelepítésnek. De biztos az NLBS a hiba forrása. Nem volt más hátra, újra kellett telepíteni legalább az egyik szervert. Megtörtént, és az NLBS-t újra beizzítva lass csodát: az eredmény ugyanaz, a hiba továbbra is jelentkezett. Ebből az következik, hogy: 1. Felesleges volt az újratelepítés. 2. Nem a Load Balacingben van a hiba, mert kizárt dolog, hogy egy frissen és hibátlanul felhúzott rendszeren sem működik hibátlanul. 3. Korábban már megállapítottuk,

hogy az NLBS-sel van a hiba, hiszen az unicast/multicast átkapcsolás megszüntette a hibás működést. Láthatóan egymásnak ellentmondó következtetésekre jutottunk „holtbiztosan hibátlan" logikával. Valamit újítani kellett.

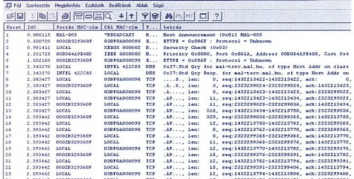
Próba - szerencse

Vajon tényleg hibátlan az új telepítés? S ha igen, akkor hogyan lehet erről meggyőződni? Hát például egy teljesen elszeparált mini hálózatban, ahol csak egy egytagú NLBS szerver üzemel, és egy ügyfél szeretne rákapcsolódni az eszközre. Ezt hamar össze lehetett hozni, hiszen csak egy hubra kellett kötni a klienst és a frissen telepített szervert, a hubot pedig le kellett választani a hálózatról. És a 2. számú csoda: az előbb még nem működő NLBS egyszerre megjavult. Vagyis: 1. Az újratelepített NLBS-nek kutya baja. 2. Ha egy zsinórátdugással megoldható a rejtély, akkor egyáltalán nem a Windows 2000 táján kell keresni a megoldást. Hanem akkor hol? Nos – akármennyire is fura – a hálózatban „valahol". Végül is van ebben logika: az untig használt unicast forgalommal működik minden, de az „egzotikus" multicast gondot okoz.

No ez meghaladta a tudásunkat – szakértőket hívtunk. Jöttek. Hoztak egy sniffer programot, amely remek statisztikákat adott a hálózat forgalmáról, például, hogy a switch minden portján rengeteg a broadcast és multicast üzenet. A forrás az NLBS cluster közös MAC címe. Hát ez meg mi? Egy switch ugye azért switch, hogy megtanulja a MAC címeket, hogy azután csak a megfelelő portok között jöjjön létre kapcsolat. Egyébként a program nem ismerte fel a csomagok pontos típusát, viszont azt látta, hogy minden másodpercben jön egy. Kellene már valami, ami látja, hogy mi van azon a fene hálózaton!!!

Network Monitor

A szakértők sajnos nem értettek a Network Monitorhoz, illetve az volt a véleményük, hogy nem kell még a csomagokat mikroszkóp alatt vizsgálni, a napnál is világosabb, hogy az NLBS-sel van a hiba és a Windows 2000-ben, tehát egy, a Windows 2000-hez és a Load Balancinghoz értő másik szakértőre van szükség. Én viszont a NetMonra szavaztam *(mégsem hiába jártatom a szám? – F.M.)*, és az alábbi eredményt kaptam:



A nyomtató közbeszól

A „beszélgetés" normálisan kezdődik. A kliens egy DNS névfeloldás után felveszi a kapcsolatot az NLBS adta szerverrel. Háromcsomagos „kézfogás" *(chip-chip-choka)*, aztán belemerülnek a terminálkapcsolat részleteibe. A harmincadik csomagot kiemeltem, mert az igen furcsa volt. A szekvenciaszám 0-0, az ack pedig a háromcsomagos „kézfogás" után az első csomagra vonatkozik. De ennél is érdekesebb, hogy a MAC címe nem azonos a korábbi szereplőkéivel. Egy idegen. Egy UFO. És mit csinál? „..A.R.." – vagyis: Reset Connection. Egészen pontosan végigböngészve a csomagot kiderült, hogy az állomás IP címe megegyezik a cluster IP címével: 10.0.0.152. És ettől kezdve az UFO folyamatosan válaszolgat a kliensnek: „Reset Connection", „Reset Connection" stb. stb.

Itt a bibi. – A MAC címet ellenőrizzük. Volt ilyen állomás: az egy hete üzembe helyezett K... típusú szép új hálózati nyomtatónk. Csakhogy annak van saját IP címe, 10.0.0.74. Hoppá! A történet most már kikerekedik: A switchünk képtelen megtanulni a multicast címeket, ezért kiküldi minden portra. *(Ezért tapasztalt a szakértőnk rengeteg multicast címet.)* Amikor az ügyfél és a cluster kommunikálni kezd, ezt is multicast csomagokkal végzik, amelyet szintén kiküld a switch minden portra. A K... nyomtató ezt a multicast folyamot a magáénak érzi *(sic!)*, majd hazudik egy jó nagyot *(sic!)*, és azt mondja, hogy az ő IP címe azonos a szerverével. Végül közli a klienssel, hogy neki nincs „terminal service" portja, ezért minden ilyen irányú kérésre „Kapcsolat megszakad" a válasza. És tényleg, ez megfelel a jelenségnek. Már majdnem felépül a kapcsolat, amikor megszakad. Ellenpróba: K... nyomtatót eltávolítjuk – és minden működik. A nyomtatót visszadugva a hiba újra előáll.

És a megoldás? Ideiglenesen kitöröltük a nyomtató alapértelmezett átjáróját, így a távoli telephelyek felhasználói újra birtokba vehették a clustert. Azután egy nyomtatószervert tettünk az – amúgy hálózati – nyomtatónk elé, és már helyben is használhattuk a clustert. Végül kaptunk egy ígéretet a gyártótól, hogy a nyomtató új firmware-t kap, amely majd nem tartalmazza a hibát. Egyelőre az örök rejtélyek birodalmában tanyázik, hogy vajon miért nem tudja az amúgy vadonatúj, C... típusú, nagyon korszerű switch megjegyezni, hogy honnan és hová kellene a multicast forgalmat irányítania.

Tanulságok

Nem mindig ott a hiba, ahol az felbukkan. Az egész hibakeresést hátráltatta, hogy végig a Windows 2000-re fogtuk a problémát, pedig az ártatlan volt. A „hagyományos" módszerek nem voltak célravezetők. Sok logikai hibát is elkövettünk – ezeket a történetben is benne hagytam, mert így őszinte. A szakértők néha nem tudnak segíteni. Végül: saját kútfőből és ingyenes eszközökkel megoldottuk a problémát, csak rá kellett szánnunk magunkat egy kis hálózati-forgalom elemzésre. – Szóval tényleg: Netmon, netmon, netmon...

Lepenye Tamás, MCSE
lepenyet@mal.hu

Ajánlott irodalom:

1. Network Load Balancing Technical Overview, (TechNet, 2001. május)
2. Windows 2000 Server Training, Network Load Balancing, (TechNet 2001. május)
3. Q238219; Q240997; Q243523;
4. Q247297; Q256124; Q264645;
5. Windows 2000 Resource Kit, Chapter 19: Network Load Balancing
6. Windows 2000 Network Load Balancing Help

Ki tartja a kezében a szálakat?

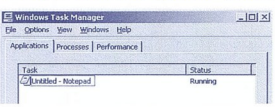
Remélhetőleg az Olvasó is rájött már, hogy nem az aktuális bel-, vagy akár világpolitikai színpad mögé fogunk bepillantani a következő néhány oldalon *(ha jól csinálják, oda úgysem lehet...)*. Természetesen az operációs rendszer által futtatott folyamatok *(processzek)* végrehajtási szálairól *(thread-ek)*, azok időzítéséről, futtatásáról lesz szó, egészen pontosan arról, hogy mikor ki „fut", ki birtokolja a processzor*(oka)*t – az is kiderül majd, hogy a címben feltett kérdésre a Windows-világban nincsen konkrét válasz. Cikkünk David Solomon **[1]** Barcelonában, a Tech.Ed 2001-en elhangzott előadása, valamint ugyanő és Mark Russinovich Inside Microsoft Windows 2000 *(Third Edition)* **[2]** című könyvének aktuális fejezete alapján készült. Ez a könyv egyébként ajánlott, sőt, kötelező olvasmány mindenkinek, aki szeretne komolyabban foglalkozni a Windows lelkivilágával.

Nagyvonalakban

A Windows NT/2000 *(továbbiakban csak Windows)* preemptív, multitaszk operációs rendszer, amelyben a végrehajtási szálak prioritása határozza meg azok végrehajtási sorrendjét, időzítését. A preemptív szó jelentése körülbelül „előjog", ami bizony azt jelenti, hogy – igencsak nem demokratikus, de mégis működő módon – ha jön valaki, akinek nagyobb prioritása van, mint az éppen futó végrehajtási szálnak, azt bizony előreengedi a sorban *(sőt, szerszámait eldobálva félreáll)*, bármit is csinált éppen. Hogy ne legyen teljes káosz és anarchia, az „elnyomottakat" az operációs rendszer néha különféle mannákkal segíti: hol a szál végrehajtási idejét *(kvantum)* növeli meg picit *(ha egyszer mégis rákerült a sor és futhat)*, hol a prioritásán emel valamennyit *(hogy mégis rákerüljön a sor és futhasson)*. Mielőtt azonban az időzítés rejtelmeibe belemennénk, tekintsük át a folyamatok és végrehajtási szálak kapcsolatait.

Folyamatok és végrehajtási szálak

A felhasználó számítógépén alkalmazásokat futtat: ilyen alkalmazás a szövegszerkesztő, a böngésző, vagy mondjuk egy könyvelőprogram. Az alkalmazás általában *(bár nem mindig)* egy folyamat, processz *(process)*. Amikor az alkalmazás elindításáról, leállításáról beszélünk, tulajdonképpen mindig az alkalmazást „képező" processzt értjük alatta.



Ha hinnénk a Task Manager-nek, azt hihetnénk, a gépen egyetlen alkalmazás fut...

Ha a Task Manager Applications oldalára kattintunk, láthatjuk az éppen futó alkalmazásokat – azazhogy a kép ezt sugallja, holott ez természetesen nem igaz. Egyrészt, itt nem láthatók a rendszert önmagát képező folyamatok, a rendszerszolgáltatások, de még azok az alkalmazások sem, amelyek a TaskBar-on búj nak meg az óra mellett:

... miközben a TaskBar-on sorakoznak az ikonok

Ezek szerint a személyes tűzfalam, a vírusirtóm, az SQL Server Service Manager, vagy az ICQ-m nem fut, esetleg nem is alkalmazás? Dehogynem! Csak a Task Manager nem teljesen mond igazat *(ez egyébként általában jellemző rá)*: az Application oldalon azok az alkalmazások láthatók, amelyek az aktuális felhasználó munkaasztalán nem rejtett ablakokat birtokolnak. Nyissuk ki az ICQ-t! Aktiváljuk a vírusirtót! Rögtön megjelennek a Task Manager-ben is.

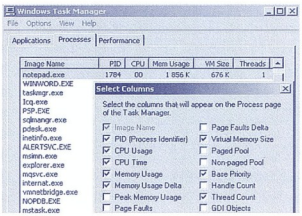
Van más apró hazugság is ezen az oldalon: az alkalmazás neve mellett látható állapotjelentés *(Status)*, ami az esetek többségében „Running" *(azaz: fut)*, illetve „Not Responding" *(azaz: nem válaszol)*. A feliratok azonban a háttérben teljesen mást jelentenek:

- Running: a folyamat válaszol a Windows üzenetekre *(felhasználói beavatkozásra vár)*; viszont nem fut sehova,
- Not Responding: a folyamat nem reagál a Windows üzenetekre, így a felhasználói beavatkozásra sem; de ettől még lehet, hogy teljesen jól érzi magát, sőt, fut, csak el van foglalva. Ilyenkor általában valamilyen I/O művelet *(lemezes vagy hálózati kommunikáció)* befejezésére vár.

Szerencsére a Task Manager részletesebb információkkal is szolgál a Processes oldalon. Itt már a számítógépen éppen futó összes processz *(folyamat)* látható, köztük megtalálhatók az előző oldalon felsorolt alkalmazások, de a többi alkalmazásunk éppúgy, mint a rendszerszolgáltatásokat, sőt az operációs rendszer alapszolgáltatásait képező folyamatok is.

Processzek és szülők

Vegyük kicsit közelebbről szemügyre, mit árul el a Task Manager egy folyamatról:



A Task Managerben kiválaszthatjuk, milyen információt szeretnénk látni a processzekről

Az alapértelmezésben kiválasztott információk mellett más adatokat is megjeleníthetünk, ha a Task Manager View menüjében a Select Coloumns... parancsot választjuk. Az ábrán is látható listából a számunkra most a PID *(Process Identifier)*, a Base Priority, valamint a Thread Count mező érdekes. Kapcsoljuk is be ezeket a mezőket, később még jól jöhetnek *(az oszlopok sorrendjét egyébként a Task Managerben húzd-és-ejtsd módszerrel át is rendezhetjük)*.

Minden processz rendelkezik egy saját azonosítóval *(ez a PID, Process Identifier)*. Bármilyen műveletet végzünk, a háttérben mindig ennek az azonosítónak a segítségével hivatkozunk az adott folyamatra. Ezen kívül minden folyamat tudja magáról azt, hogy ki a szülője *(ki hozta létre)*. Ez az információ a Task Manager-ben közvetlenül ugyan nem látszik, de a tudást használja: ha egy folyamat nevére jobb gombbal kattintunk, a megjelenő menüben választhatjuk az „End Process Tree" parancsot is, ilyenkor a Task Manager a szülőazonosítók segítségével felkutatja a rokoni szálakat, feltérképezi a családfát, és módszeresen végez a famíliával *(legalábbis a kiszemelt áldozattal és annak leszármazottjaival)*. Tiszta maffia.

Igen ám, csakhogy minden processz csak a saját szülőjére emlékszik, a távolabbi rokonokra már nem.

Hozzunk létre egy próbakörnyezetet:

- Nyissunk egy közönséges parancssori ablakot.
- A parancssorba gépeljük be: start cmd. A parancs hatására egy második parancssori ablak jelenik meg.
- A második parancssori ablakba pedig írjuk ezt: notepad.exe.

A létrehozott családfát jól jelképezi a Windows 2000 Support Tools-ban *(amely minden Windows 2000 CD /support/tools könyvtárából feltelepíthető)* található tlist /t parancs kimenete:

```
C:\>tlist /t
...
explorer.exe (976) Program Manager
  CMD.EXE (1768) W:\WIN2KPRO\System32\cmd.exe
    CMD.EXE (816) W:\WIN2KPRO\system32\CMD.EXE
      notepad.exe (812) Untitled - Notepad
...
```

A parancs eredményéből csak az aktuális részt vágtuk be: látható, hogy az „ősapa" az explorer.exe. A szülő-gyermek viszonyokat a behúzások jelzik; ha valakinek nincs őse *(ilyen az explorer.exe is)*, az a sor bal szélére kerül. Ha az 1768-as PID-jű cmd.exe-t és leszármazottjait leállítjuk *(Task Manager, Processes oldal, End Process Tree parancs)*, az magával rántja a másik parancssori ablakot és – természetesen – a NotePad-et is. Ha viszont a köztes parancssori ablakot bezárjuk *(a hagyományos módon)*, a tlist /t kimenete így módosul:

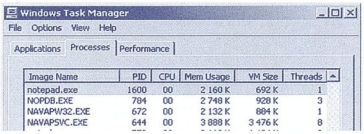
```
C:\>tlist /t
...
explorer.exe (976) Program Manager
  CMD.EXE (1768) W:\WIN2KPRO\System32\cmd.exe
...
notepad.exe (812) Untitled - Notepad
...
```

A 816-os PID-jű cmd.exe már nem fut; a notepad.exe pedig „elanyátlanodott" *(egészen a sor elejére csúszott)*, és pontosan ez menti meg az életét! Ha most végeznénk a cmd.exevel és gyermekeivel, a NotePad még vígan működne tovább.

Az ne zavarjon meg senkit, ha esetleg egy gyermek processz PID-je alacsonyabb, mint a szülőjéé: a PID-k újrafelhasználhatók, ha valaki elengedi, egy következő létrehozott folyamat felveheti.

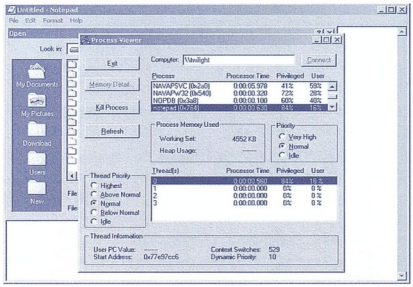
A végrehajtási szálak

Minden processz legalább egy végrehajtási szálat tartalmaz, ez a processz úgynevezett „Initial Thread" *(kezdeti végrehajtási szál)*. Az operációs rendszer szempontjából igazán a végrehajtási szálaknak van értelme, az időzítés is végrehajtási szálakat kezel – a processz nem más, mint egy vagy több végrehajtási szál közös adminisztratív környezete. A kezdő thread később újabb szálakat hozhat létre egy-egy részfeladat elvégzésére. Ha a Task Manager-ben megfigyeljük, a „Threads" oszlopban láthatjuk, hogy melyik processz hány végrehajtási szál képében „testesül meg".



A Task Manager „Threads" oszlopában a végrehajtási szálak száma látható

Észrevehető, hogy míg a notepad.exe megelégszik egyetlen végrehajtási szállal, a navapsvc.exe *(egy vírusírtó rendszerszolgáltatása)* már nyolc szálat futtat. De próbáljunk csak a jegyzettömbben megnyitni valamit! A végrehajtási szálak száma azonnal megnő *(a dialógusablak kezeléséhez a rendszer további szálakat hozott létre)*.



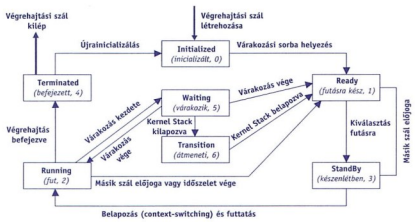
Process Viewer

A fenti ábrán egy másik nagyon hasznos eszköz látható. Ez a Process Viewer *(pviewer.exe)*, ami ugyancsak a Windows 2000 Support Tools része. A Process Viewer jóval részletesebb információkkal szolgál, mint a Task Manager, láthatjuk például, hogy az egyes folyamatok, végrehajtási szálak milyen prioritással rendelkeznek, illetve azt, hogy az egyes végrehajtási szálak mennyi processzoridőt használtak fel felhasználói és

privilegizált *(rendszer)* futási módban. Ha akarjuk, a „Kill Process" gomb segítségével leállíthatjuk a kiválasztott folyamatot, de vigyázzunk, ez az eszköz véres kezű, és véresen komolyan veszi a kérésünket még akkor is, ha a szóban forgó áldozat éppen akár egy rendszerszolgáltatás *(a Task Manager például az ilyen processzeket jogosultság hiányában nem hagyja leállítani)*.

A végrehajtási szálak futásának időzítése

Időzítés tekintetében a processznek, mint fogalomnak nincs jelentősége: a Windows a végrehajtási szálak futását kezeli. Bár a rendszerben látszólag egyszerre akár több száz végrehajtási szál is futhat, valójában egyidejűleg csak annyi fut, ahány processzor van a gépben *(ha több processzorunk van, alapértelmezésben bármelyik processzoron bármelyik szál futhat, de a végrehajtási szálak kiválaszthatják a nekik „szimpatikus" processzor(oka)t – ez a Processor Affinity –. Ilyenkor a végrehajtási szál csak a kiválasztott processzoron fog futni, még akkor is, ha a másik éppen üresjáratban van.)* Hogy mégis fenntartsuk az egyidejűség látszatát, minden végrehajtási szál csak egy bizonyos ideig futhat, azután át kell adnia a helyét a következőnek. Azt, hogy a sorban ki következhet, a végrehajtási szál pillanatnyi prioritása dönti el. Mindenekelőtt tekintsük át egy-egy végrehajtási szál életciklusát:

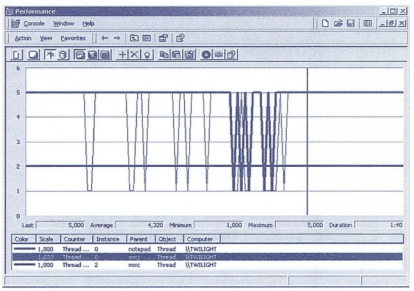


A végrehajtási szálak életciklusa

A végrehajtási szál létrehozásakor inicializált *(Initialized)* majd futásra kész *(Ready)* állapotba kerül. A legtöbb szál ebben az állapotban időzik a legtöbbet. Amikor a rendszer a végrehajtási szálat kiválasztja futásra, készenléti *(StandBy)* állapotba kerül; innen pedig – hacsak közben nem „érkezett" egy magasabb prioritású szál – futó *(Running)* állapotba jut. Ebben az állapotban csak egy ideig maradhat, ezt az időt a szál részére fenntartott időszelet érték, a Quantum határozza meg. Ha az időszelet lejárt, a végrehajtási szál újra a futásra kész állapotba lép. Ha a végrehajtási szál végleg befejezte a munkáját, leáll: befejezett *(Terminated)* állapotba kerül. Ezután a végrehajtási szálat vagy megszüntetjük, vagy később újrainicializáljuk *(reinkarnáció)*. A végrehajtási szálak azonban sokszor nem töltik ki a rendelkezésükre álló időszeletet: egyrészt, önszántukból várakozó *(Waiting)* státuszba léphetnek, amíg egy I/O művelet, vagy kernelfunkció visszatérésére várnak. Másrészt, a Windows saját hatáskörben felfüggesztheti egy végrehajtási szál futását, ha azt észleli, hogy egy fontosabb szál kész a futtatásra. A Windows a következő esetekben dönthet az éppen futó szál megszakításáról:

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

A fenti esetek mindegyikében a Windows megvizsgálja a futásra kész szálak prioritását, és ha azok közül van, amely magasabb, mint az éppen futó végrehajtási szálé, bizony helycsere következik. Az aktuális állapot elmentődik, és a kiválasztott szál állapota töltődik be a helyére – ezt a műveletet nevezzük kontextusváltásnak *(context switching)*. A végrehajtási szálak aktuális állapotát egyébként a Performance Monitorban is figyelemmel követhetjük:



A felső (vastag vonallal rajzolt) érték a NotePad, a másik kettő a Performance Monitor egy-egy végrehajtási szálának állapotértékei

Az egyes értékek jelentését az előző ábrából leolvashatjuk. Látható, hogy a NotePad *(felhasználói inputra)* várakozó és futásra kész állapot között mozog, de futó státuszba – látszólag – soha nem lép. Hogy miért? Azért, mert egyprocesszoros gépen az adatok begyűjtésének pillanatában bizony a Performance Monitor adatgyűjtő végrehajtási szála fut!

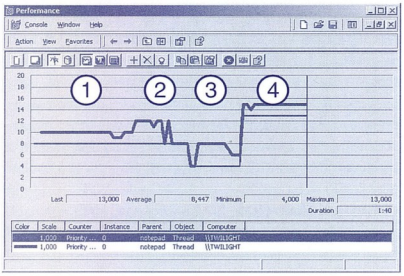
A prioritás

A processzorhoz jutás előjogát a végrehajtási szálak prioritása határozza meg. A rendszer „belül" 32 különböző prioritási szintet különböztet meg. A 32 szint két fő részre van osztva; az alsó értékek *(0..15)* az úgynevezett dinamikus, a felső rész *(16..31)* pedig az úgynevezett valósidejű prioritási értékek. A 0 prioritási érték nem osztható ki, az mindig a Zero Page Thread tulajdona. A prioritási értékek ilyen durva szétválasztására azért van szükség, nehogy magát az operációs rendszert képező végrehajtási szálak is áldozatul essenek a prioritásért vívott harcban. Az operációs rendszer *(pontosabban a kernel)* végrehajtási szálai tehát eleve sokkal magasabb prioritásértéken futnak, mint a felhasználói szálak.

A felhasználói felületre kivezetett prioritásértékek nem számszerűek, hanem szöveges megjelenésűek, úgynevezett prioritásosztályok. Minimális, közép és maximális értékeik a következők:

		min.	alap	max.
Idle, Low	üresjárati, alacsony	2	4	6
Below Normal	normális alatt	4	6	8
Normal	normális	6	8	10
Above Normal	normális felett	8	10	12
High	magas	11	13	15
Real-Time	valósidejű	22	24	26

Minden szálnak két prioritási jellemzője van: az úgynevezett alap *(base)* és a pillanatnyi *(current)*. Valósidejű prioritás esetén ez a két érték mindig megegyezik, dinamikus prioritás esetén viszont nem feltétlenül: a Windows a pillanatnyi prioritást az igényeknek megfelelően változtatja. A prioritások értékét többek között a Task Manager, a Process Viewer és a Performance Monitor segítségével tekinthetjük meg *(a feladatkezelővel az alap prioritást módosíthatjuk is)*.



A NotePad végrehajtási szála prioritásértékének változása: a vékonyabb vonal az alap (base) a vastagabb a pillanatnyi (current) prioritás

A fenti ábrán látható, hogyan változott a NotePad végrehajtási szálának prioritása, miközben várakozott ①, miközben dolgoztunk vele ②, illetve amikor alacsony *(Low, ③)* vagy amikor magas *(High, ④)* prioritást állítottunk be neki a Task Manager-ben.

Priority Boost

A dinamikus prioritás változásai az úgynevezett Priority Boost *(prioritás-növelés)* következménye. A végrehajtási szál dinamikus prioritásértékét a Windows a következő esetekben emelheti meg:

- I/O művelet befejezése után
- Kerneleseményekre való várakozás után
- Az előtérben futó alkalmazás várakozó állapotból való visszatérésekor
- A felhasználói felület végrehajtási szálainál felhasználói aktivitás esetén
- Amikor egy végrehajtási szál már régóta nem futott

A megemelt prioritású végrehajtási szálak előjogot kapnak a processzor használatára. A „boost" mindig a végrehajtási szál alapprioritásához adódik hozzá, és hatása idővel el is múlik (a prioritás visszaáll az eredeti értékre).

A Quantum

A dinamikus működés másik biztosítéka a végrehajtási szálak futási időszeletének változtatása *(ha egy szál végre hozzáfért a processzorhoz, legalább hadd használja kicsit tovább)*. Az időszelet hossza egyébként eleve más-más a Professional és a Server operációs rendszereken, de ez a beállítás módosítható. A Quantum alapértelmezett értéke Professional esetén 6, Server esetén 12; minden egyes óramegszakításkor *(x86 processzorokál általában 10; multiprocesszoros rendszereknél 15 ms-ként)* az időszelet értéke 3-mal csökken. Így könnyen kiszámolható, hogy egy-egy végrehajtási szál mennyi ideig használhatja a processzort. *(Akit érdekel, miért pont 3-mal, az olvassa el az említett könyv idevonatkozó fejezeteit.)*

Az optimális teljesítmény érdekében a Windows a végrehajtási szálak időszeletének értékét is megnövelheti *(quantum boost)*, mégpedig akkor, amikor az adott végrehajtási szál ablaka az előtérben van. Az alapértelmezett időszelet értéke; a boost engedélyezése, sőt még a boost értéke is beállítható, mégpedig a

```
HKLM\System\CurrentControlSet\Control\
↳ PriorityControl\Win32PrioritySeparation
```

registry érték segítségével. Az érték alsó hat bitje számít, az alábbi kiosztásban:

5	4	3	2	1	0
Short / Long		Variable / Fixed		Boost érték	

- Short / Long: A Quantum hossza. 1: hosszú; 2: rövid; a többi érték esetén az alapértelmezés érvényesül *(Pro: rövid, Server: hosszú)*
- Variable / Fixed: Legyen-e Quantum Boost ? 1: igen; 2: nem; a többi érték esetén az alapértelmezés érvényesül *(Pro: igen, Server: nem)*
- Boost értéke: ha a Quantum Boost engedélyezve van, itt adhatjuk meg a növelés szintjét *(értéke 0 .. 2 között mozoghat)*.

	Short			Long		
Boost	**0**	**1**	**2**	**0**	**1**	**2**
Variable	6	12	18	12	24	36
Fixed	18	18	18	36	36	36

A lehetséges Quantum értékek táblázata

A System tulajdonságlapról megnyitható Performance Options dialógus jól ismert kérdése *("Optimalize performance for: Applications / Background Services")* is pontosan ezt a registry értéket módosítja, mégpedig az alábbiakra *(a táblázatban szürkével jelöltük)*:

- Optimize for Applications: Short; Variable; Boost 2 *(18)*
- Optimize for Background Services: Long; Fixed *(36)*

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:
[1] http://www.solsem.com
[2] http://mspress.microsoft.com/prod/books/4354.htm

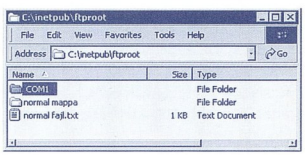
Törölhetetlen mappák

A lusta rendszergazda súlyos tévedése

Főszereplőnk egy lusta rendszergazda, akinek biztosítania kell, hogy egyetlen felhasználó a következő 2-3 napban valamikor Interneten keresztül fel tudjon tölteni egy nagyobb fájlt egy fájlszerverre. A rendszergazda tudja, hogy a kiszolgálón Windows 2000 van, amelyben található egy FTP szerver. Mi sem egyszerűbb, mint azt elindítani, engedélyezni rajta az Anonymous elérést és a Write műveleteket, valamint az IUSR_GepNev felhasználónak Full Control jogot adni a C:\INETPUB\FTPROOT mappára. Eközben a következő feltételezésekkel él:

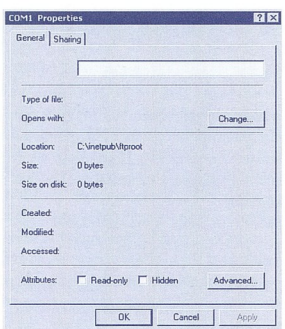
- A szerver létezéséről még nem tud senki, hiszen tegnap telepítettem.
- Csak egy felhasználónak árulom el, hogy elérheti FTP-n keresztül, akkor más miért is próbálná.
- Három nap múlva kikapcsolom az FTP servert, addig nem lehet gond, hisz csak három nap.

Tényekedései eredményeként már másnap a következő érdekes mappát találja a szerveren:



Gyanús mappa

Mikor a mappa tulajdonságait is megnézi, meglepőt tapasztal, amikor pedig törölni sem tudja, kezd pánikba esni. Egyáltalán milyen üzenet az, hogy Cannot read from source file or disk ?



Ez egy érdekes mappa, nem sokat tudunk róla

Az FTP szerver eseménynaplója pedig annyi bejegyzést tartalmaz, hogy képtelen végigrágni magát rajta. Mindössze egyetlen nap telt el!

Nagyon sok egyéb biztonsági ajánláson kívül azt sem vette figyelembe, hogy attól még, hogy ő nem árulta el senkinek a kiszolgáló létezését, az igenis látszik az Internetről. Nem kell hozzá más, mint egy egyszerű portscanner program, a jobbak még azt is kipróbálják, hogy be tudnak-e lépni Anonymousként az FTP szerverre **[3]**.

Adott tehát egy könyvtár, amit le kéne törölni, de nem sikerült sem Explorerből, sem pedig parancssorból, akárhogy próbálkozik az idézőjelekkel vagy a joker karakterekkel. Természetesen ír a NetAcademia Windows 2000 levelezési listájára, ahonnan nagyon sok jó ötletet kap, azonban mindegyik látványosan csődöt mond. *(Nem mindig vagyunk ilyen bénák – a szerk.)* Tartalékként maradnak még olyan javaslatok, hogy mountolja fel Linux alól a meghajtót vagy valami DiskEdit szerűséggel szerkesztgesse a lemezt közvetlenül, de ez már egy kicsit túlzás lenne. Nem marad más hátra, mint körülnézni az Interneten, hogyan is lehet azt a COM1 könyvtárat letörölni?

MS-DOS kompatibilitás

Egy ehhez hasonló probléma megtalálható az összes Microsoft operációs rendszer termékben – ráadásul szándékosan *(Q216654 [1])*. Ennek oka pedig a kompatibilitás, méghozzá nem is akármivel, hanem a történelemkönyvekből jól ismert MS-DOS-szal. Még a legújabb Windows verziók is megőrizték azt a tulajdonságukat, hogy a fájlműveletek parancssorból is elvégezhetők, például a COPY, DEL vagy REN utasítás éppúgy használható, mint tíz évvel ezelőtt. Azonban ezeknek az utasításoknak szükségszerűen voltak és vannak határai, mert az MS-DOS idején bizonyos nevek device-ként szolgáltak, ezért ezeket nem használhattuk kötetlenül: CON, PRN, AUX, CLOCK$, NUL, COM1-COM9, LPT1-LPT9. Windows 2000 alatt is remekül tudunk fájlt létrehozni az alábbi módon:

```
C:\>copy con ujfajl.txt
Ez lesz a fajl tartalma. Befejezes CTRL+Z-vel.
^Z
        1 file(s) copied.
C:\>
```

Ha azonban a kívánt fájl neve például COM1.TXT, próbálkozásunk szánalmasan csődöt mond, miközben bezsebelünk egy **"Access is denied"** üzenetet. Átnevezés esetén próbálkozásunk jutalma egy **"A duplicate file name exists, or the file cannot be found"** hibaüzenet. Vicces, hogy egyik üzenetből sem látszik, hogy a problémát valójában a foglalt név használata okozza.

Ha a foglalt nevekkel Windows 2000 alatt Windows Explorerből játszunk, a hibaüzenet már egy kicsit többet mond *(The filename you specified is invalid or too long)*. Ha azonban Windows 95 alatt próbálunk egy COM1 nevű mappát létrehozni, akár sikerrel is járhatunk. A probléma akkor jön elő, amikor ettől a jólfésült könyvtártól szeretnénk megválni, mert sem az egyszerű DEL vagy RD parancs, sem pedig a Windows

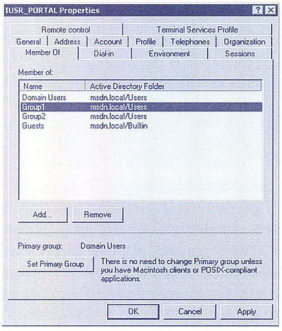
Explorer nem lesz hajlandó ezt letörölni, a Windows 95 ugyanis a soros portot próbálja szorgalmasan kiírtani, eredménytelenül. Ekkor az alábbi szintaxis vezethet célra:

```
DEL \\.\c:\mappanev\com1
```

A Windows 95-höz készült javítócsomag kiküszöböli a problémát. A Windows NT és 2000 annyira immunis ezzel a problémával szemben, hogy el sem tudják képzelni, hogy ilyen könyvtár létezhet. Ezért ha mégis lenne ilyen mappánk és azt törölni próbáljuk, a válasz: **"Cannot read from source file or disk"**. Emiatt a cikk elején említett mappától sem tudunk így megszabadulni, még a bonyolított szintaxissal sem.

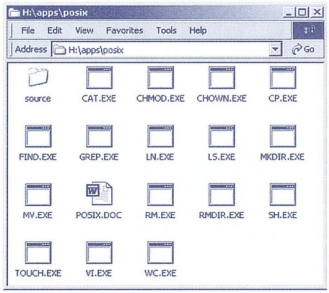
POSIX

Ha a mappatörléssel kapcsolatban tovább kutakodunk az Interneten, előbb-utóbb belefutunk a Q120716-os Tudásbázis cikkbe *([1])*, ami ugyanezt a problémát tárgyalja Windows NT és 2000 környezet esetére. A Windows NT-nek és a 2000-nek ugyanis van egy POSIX *(egyfajta juniksz – a szerk.)* alrendszere, ami igenis támogatja ezeket a foglalt neveket. Ennek az alrendszernek a segítségével lehet olyan mappát vagy fájlt létrehozni NTFS partíción, aminek a neve a korábban említett foglalt kulcsszavak közül származik. Ha viszont POSIX bigyó hozta létre, akkor POSIX bigyó törölje is le, azaz a sima Windows 2000 ezt nem támogatja! Nincs más hátra, szerezni kell POSIX-os törlő programot.
A Windows NT ősidők óta hordozza magában a POSIX kompatibilitást, az esetek többségében azonban nem találkozunk ezzel. Ha Windows 2000 Serveren egy Active Directory-ban tárolt felhasználó csoporttagságát szerkesztjük, találhatunk egy Set Primary Group opciót, aminek tipikus esetben az égvilágon semmi hatása nincs, egyedül a POSIX alrendszer és Macintosh kliensek számára hordoz információt.



Csak POSIX alkalmazásoknak fontos

A POSIX alrendszer akkor aktiválódik, ha olyan alkalmazást indítunk, ami azt kihasználja. Ekkor elindul a SYSTEM32 mappában található POSIX.EXE, ami ténylegesen el fogja indítani a PSXSS.EXE-t, a POSIX Subsystemet. Ez később tovább fut, a Task Managerben megtalálható, sőt még az End Processre is fittyet hány. Hol lehet ilyen alkalmazásokat találni? Legkönnyebben a Windows 2000 Professional és Server Resource Kit CD-n az APPS\POSIX mappában, itt ugyanis számos UNIX-szerű segédprogram figyel.



POSIX segédprogramok a Resource Kit CD-n

Az egyes eszközökről részletes leírást az ugyanebben a mappában leledző POSIX.DOC szolgáltat. Ebbe belekukkantva ráakadhatunk az RMDIR.EXE-re, ami egy parancssori eszköz és segítségével mappákat lehetne eltávolítani *(ReMove DIRectory)*. Ennek használata előtt érdemes tudni, hogy a POSIX parancsok érzékenyek a kis- és nagybetűk különbségére, valamint hogy a mappákra és fájlokra a már megszokottól eltérően kell hivatkozni. Például:

```
rm –d "//C/FoMappa/AlMappa/COM1"
```

Ez valószínűleg remekül működik, a bevezetőben említett mappa törlésére azonban teljesen alkalmatlannak bizonyult.

FTP !

Miközben sikerült megfejteni a Windows 9x-ek és a POSIX alrendszer hiányosságait és nehézségeit, háttérbe szorult az a tény, hogy a kérdéses mappát a rosszarcú júzer FTP-n keresztül hozta létre, márpedig ez nem közömbös!
Ha egy FTP sessionben az MKDIR parancs segítségével létrehozunk egy mappát az látszólag teljesen ugyanúgy viselkedik, mintha azt Explorerben vagy parancssorban az MD parancs segítségével tennénk. A kettő között igenis van különbség, még ha ez elég apró, akkor is. Példaként hozzunk létre mindkét módon egy új mappát a következő szintaxissal. FTP-n keresztül simán megy:

```
ftp> mkdir ./ujmappa/ /
257 "./ujmappa" directory created.
```

De parancssorból nem:

```
C:\>md" .\ujmappa\ "
The syntax of the command is incorrect.
```

Pontosan ez az apró szintaxis-különbség, és ennek tovább bővített változatai adnak lehetőséget rosszalkodásra **[2]**.

Törölhetetlen könyvtárak

Az alapötlet az, hogy létrehozunk egy „ ” (azaz szóköz) könyvárat, ami nem látszik. Ehhez jelentkezzünk be a saját gépünkön levő IIS FTP szerverre, ahol már korábban engedélyeztük a Write műveleteket is *(IIS MMC snap-in, Default FTP Site Properties, Home Directory fül, Write beklikk)* és NTFS szinten is van jogunk matatni a mappában. Ezután próbáljuk ki a következő parancsot:

```
ftp> mkdir "./ ujmappa / /"
257 "./ ujmappa / /" directory created.
```

Az eredmény ugyanaz, mint amit a bevezetőben láthatunk, csak a mappa neve más. Akár be is léphetünk a mappába, de csak az alábbi szintaxissal *(ez megy FTP nélkül, parancssori ablakból is)*:

```
ftp> cd "./ ujmappa / /"
250 CWD command successful.
```

Ha törölni próbáljuk, akkor **"Cannot read from source file or disk"** illetve **"The system cannot find the path specified"** üzenettel lehetünk gazdagabbak. Próbáljuk inkább átnevezni, az átnevezés után már tudjuk törölni:

```
ftp> ren "./ ujmappa / /" ujnev
350 File exists, ready for destination name
250 RNTO command successful.
ftp> rmdir ujnev
250 RMD command successful.
```

Ugyanezzel a módszerrel készítette a felhasználó a bevezetőben szereplő COM1 mappát is és ugyanígy lehet megválni tőle. Természetesen semmi akadálya, hogy a szóközök számát variálva még jobban kiszúrjon bárki is a rendszergazdával, ugyanis a mappa törléséhez feltétlenül szükséges a pontos szintaxis, amit a „hacker" használt. Ezt az FTP szerver naplófájljaiból tudhatjuk meg, melyek alapértelmezés szerint a SYSTEM32\LogFiles\MSFTPSVC1 mappában találhatóak. Hogy az FTP szerver naplójába milyen mezők értékei kerüljenek, azt az Internet Information Services snap-inben a Default FTP Site Properties ablakában az első fülön a Properties gombra kattintva tudjuk beállítani.

Törölhetetlen fájlok

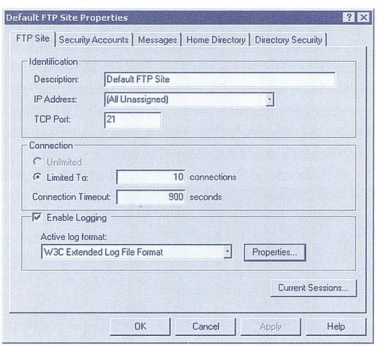
Hasonlóan van lehetőség törölhetetlen fájl készítésére is, mégpedig átnevezéssel:

```
ftp> rename matrix.mpg "ujnev ./ /"
350 File exists, ready for destination name
250 RNTO command successful.
```

Ezt a fájlt letörölni nem lehet, csak ugyanilyen átnevezéses módon felülírni egy üres fájllal. Ott lesz, de legalább nem foglal helyet. *(Megjegyzés: nekem még úgy sem sikerült letörölni az így „írásvédetté" tett fájlt!)*

Az így elérhetetlenné és törölhetetlenné tett fájlok és mappák elérésére nem minden FTP kliens alkalmas, a Windows 2000-ben található FTP klienssel nekem nem sikerült használnom a mappákat. Ez természetesen nem jelenti azt, hogy nem léteznek akár Windows, akár más operációs rendszer alá a célnak megfelelő alkalmazások. Mivel a tapasztalatok szerint az utólagos törlés nagyon nehézkes, célszerű megelőzni a bajt a megfelelő korlátozások beállításával. Csak néhány ötlet a sok közül:

- Szükségem van-e egyáltalán FTP elérésre?
- Ha igen, feltétlenül a 21-es porton kell-e lennie?
- Szükséges-e Anonymous elérés?
- Szükséges-e, hogy a felhasználó írni is tudjon az FTP szerver mappáiba?
- Melyik meghajtón található az FTP szerver gyökérkönyvtára, leformázhatom-e azt a meghajtót akármikor?
- Milyen NTFS jogokat adjak a mappára?
- Milyen quota limitet célszerű beállítanom a meghajtóra?
- Akarok-e egy héten belül jó néhány crackelt alkalmazást vagy mozifilmet?
- Feltelepítettem-e az összes javítócsomagot? *(!)*
- Elég részletes-e a naplózás? *(!)*



Az FTP szerver naplózási beállításai

Akit további részletek érdekelnek, nézzen körül a **[4]** címen és böngésszen végig a **[2]** címen található leírást, amely így kezdődik: „This documentation should not be used to exploit webservers." Én elolvastam ezt a mondatot is.

Balássy György
balassy@avalon.aut.bme.hu

A cikkben szereplő URL-ek:

[1] http://search.support.microsoft.com/kb/c.asp
[2] http://home.hetnet.nl/~paulus17/how2stop_deleters.txt
[3] http://www.eeye.com/html/Products/Retina/index.html
[4] http://www.netknowledgebase.com/forum

Elektronikus aláírás

Végre nem kell sorba állnunk a hivatalokban, avagy mégis?

Az elektronikus aláírás kezdetei.

Június közepén jelent meg a magyarországi informatikai jogalkotás első mérföldkövét jelentő törvény az elektronikus aláírásról. Ezzel mintegy két év elmaradással követtük az Európai Unió jogalkotását, ahol a szabályozás keretét adó irányelv már 1999-ben napvilágot látott. A Magyar Köztársaság és az Európai Unió között megkötött társulási szerződéssel ránk háruló jogharmonizációs kötelezettség a magyar jogalkotási szabadságot igencsak korlátozta ezen a területen is, így nem tehettünk mást, mint hogy átvettük az Európai Unióban már - és tegyük hozzá nem is rosszul - kitalált normákat, kicsit megspékelve a magyar piaci sajátosságokkal. Ebből készült el a sült, amit most feltálaltak nekünk, bár megízlelni csak a hatálybalépés szeptember 1-i dátumát követően tudjuk. Így most a gyakorlati próba előtt a törvényben leírtakkal tudunk megismerkedni.

E cikk olvasója nyilván nálam sokkal jobban ismeri az aláírás technikai hátterét, így ennek részleteit mellőzve most csak arra mutatok rá, hogy az aláírás során a kódoló algoritmusos megoldást alkalmazzák. A titkos kulcs *(a törvény szövege szerint: aláíráslétrehozó adat)* szolgál az aláírásra, míg a nyilvános kulcs *(aláírásellenőrző adat)* arra, hogy a címzett azt dekódolja, és így megállapítsa, hogy az üzenet szövegében történt-e bármilyen változtatás.

A törvény az elektronikus aláírás elfogadására határozott garanciát ír elő, amikor kimondja, hogy az elektronikus aláírással ellátott dokumentum elfogadását - ide értve a bizonyítási eljárás során bizonyítékként történő értékelést is - nem lehet megtagadni csupán azért, mert az irat kizárólag elektronikus formában létezik. Ez alól azonban rögtön kivételt is megállapít, a házassági és a családjog területén. További szűkítése a lehetséges felhasználási területnek, hogy bírósági és államigazgatási eljárásban csak akkor lehet kizárólag elektronikus dokumentumokat felhasználni, ha arra az adott eljárási jogszabály kifejezett felhatalmazást ad. Ez utóbbi rendelkezések gyakorlatilag olyannyira leszűkítik az elektronikus aláírás pillanatnyi jelentőségét, hogy azt csupán a magánjogi kapcsolatokra korlátozza. Egyedül csak a cégbírósági eljárásban válik lehetővé szeptember 1-től az iratok elektronikus úton történő benyújtása. Érthető egyrészt, hogy a bíróságok és államigazgatási szervek mai technikai fejlettsége nem teszi lehetővé, hogy elektronikus okiratokat tömegével fogadjanak, szomorú viszont ez a spanyol tangó szerű jogalkotás: két lépés előre, egy lépés hátra. Vagyis az a szép remény, hogy szeptembertől a gép előtt ülve intézhetjük ügyes-bajos dolgainkat a hivatalban történő hosszas sorbanállás nélkül, egyelőre csak utópia marad. Továbbra is papírokkal felszerelkezve zarándokolhatunk a polgármesteri hivatalok és bíróságok folyosóin, mindaddig, amíg azok a jogszabályok meg nem jelennek, amelyek az eljárásokban lehetővé teszik az elektronikus okiratok alkalmazását.

Az ügyfelek részére viszont fontos garanciát jelent az, hogy jogszabály nem teheti részükre kötelezővé az elektronikus forma használatát.

Megkülönböztet a törvény minősített elektronikus aláírást a „sima" elektronikus aláírással szemben. Az előbbi olyan fokozottan biztonságos elektronikus aláírás, amely biztonságos aláíráslétrehozó eszközzel készült, és amelynek hitelesítése céljából minősített tanúsítványt bocsátottak ki. Természetes követelmény, hogy erre csak minősített hitelesítésszolgáltató legyen jogosult, aki az erre vonatkozó minősítést a Felügyelettől megszerzi. A fokozott biztonságú szolgáltatás végzését továbbá annak megkezdését megelőzően 30 nappal korábban be kell jelenteni a Felügyeletnek. A minősített elektronikus aláírásnak a jogkövetkezmények szempontjából van jelentősége, mivel olyan esetben, ahol jogszabály valamely szerződés vagy nyilatkozat érvényességéhez az írásbeli formát kötelezővé teszi, az elektronikus okirat csak a minősített elektronikus aláírás alkalmazása esetén váltja ki a szükséges jogkövetkezményt. Így például adásvételi szerződést csak minősített aláírással köthetünk elektronikus úton. *(Más kérdés, hogy jelenleg viszont nem lesz arra alkalmas az így megkötött szerződésünk, hogy a földhivatal a tulajdonjogunkat bejegyezze.)*

Az elektronikus aláírás szolgáltatás alatt ténylegesen három különböző szolgáltatásról beszélünk:

a) elektronikus aláíráshitelesítés szolgáltatás
b) időbélyegzés
c) aláíráslétrehozó eszközön az aláíráslétrehozó adat elhelyezése

Az elektronikus aláíráshitelesítés szolgáltatás keretében a hitelesítésszolgáltató azonosítja az igénylő személyét, tanúsítványt bocsát ki, nyilvántartásokat vezet, fogadja a tanúsítványokkal kapcsolatos változások adatait, valamint nyilvánosságra hozza a tanúsítványhoz tartozó szabályzatokat, az aláírásellenőrző adatokat és a tanúsítvány aktuális állapotára *(különösen esetleges visszavonására)* vonatkozó információkat.

Időbélyegzés során a szolgáltató az elektronikus dokumentumhoz időbélyegzőt kapcsol. Az időbélyegző tulajdonképpen semmi másra nem való, mint hogy az okirat keletkezésének időpontját hitelt érdemlően bizonyítsa.

A szolgáltató választása szerint akár csak egy, akár több, vagy valamennyi szolgáltatás nyújtására is jogosult.

A magyar törvény az európai megoldást követve nem köti meg különösebben a szolgáltatóvá válás feltételeit. A fokozott biztonságú, illetve minősített elektronikus aláírás szolgáltatás esetén már említett kötelezettségek állnak fenn. A szolgáltatás végzésére minden elképzelhető vállalkozási forma feljogosítást kapott, mivel a belföldi lakóhelyű vagy belföldön tartózkodó természetes személy, jogi személy vagy jogi személyiség nélküli szervezet is jogosult. A törvény ilyen széles körű megfogalmazása azt sejteti, hogy nemcsak vállalkozási, hanem nonprofit tevékenység keretében is lehet elektronikus

aláírás hitelesítést végezni. A közeli csatlakozás reményének jegyében az Európai Unió tagországaiban kiadott hitelesítést a magyar törvény minden további nélkül elfogadja.

Az eddig ismertetettekből következik, hogy igazán értelme csak a minősített hitelesítésszolgáltatás nyújtásnak van. Az ezen a területen történő szolgáltatóvá válásnak azonban olyan technikai feltételei is vannak, amelynek teljesítéséhez nyilván komoly díjakat kell fizetni a szükséges igazolások és tanúsítványok kiállítása fejében. Ugyancsak jól járunk mi, jogászok is, hiszen a Felügyeletnek be kell nyújtani a szolgáltatásra vonatkozó általános szerződési feltételeket és a szolgáltatási szabályzatot. Ezek megfelelő elkészítése pedig a megfelelő gyakorlat kialakulásáig komoly úttörő jogi munkát jelent. Külön jogszabály írja elő azt a pénzügyi hátteret és felelősségbiztosítást, amivel a minősített szolgáltatónak ugyancsak rendelkeznie kell. A szolgáltató illetve alkalmazottja személyére vonatkozó előírások, mint a büntetlen előélet és a szakmai képzettség igazolása már csak kisebb nehézséggel jár. Látható hát, hogy nem kell attól félni, hogy a piacot ellepik a minősített hitelesítés szolgáltatók, hiszen a felsoroltaknak már igen kevesen tudnak megfelelni. A jogszabály ravasz megoldásával tehát nem a szolgáltatás végzésére feljogosítottak köre került korlátozásra, hanem a feltételeknek való megfeleléssel szűkítik le a potenciális piaci szereplők körét.

A szolgáltatót természetesen teljes körű kártérítési felelősség terheli mind a vele szerződéses kapcsolatban álló aláíróval, mind azon harmadik személlyel szemben, akinek a szolgáltatás szabályainak megszegésével kárt okoz. A szolgáltató azonban jogosult arra, hogy a minősített tanúsítványban, amelyet a minősített aláírás hitelesítéséről állít ki, meghatározza a tanúsítvány felhasználásának tárgyibeli, földrajzi vagy egyéb korlátait, illetve az egy alkalommal vállalható kötelezettség legmagasabb értékét. Ez ugyanakkor azt is jelenti, hogy nem felel azokért a károkért, amelyeket e korlátokat meghaladóan kibocsátott elektronikus okiattal okoznak.

A hitelesítésszolgáltató tevékenység befejezése esetére biztosítani kell az aláírók jogait. Így a törvény erre vonatkozóan szigorú szabályokat állapít meg: legalább 60 nappal a tevékenység befejezését megelőzően értesíteni kell a kibocsátott és még vissza nem vont tanúsítványokban aláíróként megjelölt személyeket, valamint a Felügyeletet. Köteles intézkedni az iránt, hogy legkésőbb a tevékenység befejezésekor más szolgáltató átvegye az aláírás nyilvántartásokat. A tevékenység befejezését megelőzően legalább 20 nappal köteles az általa kibocsátott és még vissza nem vont tanúsítványokat visszavonni.

A szolgáltatót különös kötelezettség terheli az aláírók adatainak kezelése, az adatvédelmi előírások biztosítása területén. Csak az aláírótól közvetlenül, vagy annak előzetes egyetértésével gyűjthetnek személyes adatokat és csak olyan mértékben, ami a tanúsítvány kiállításához szükséges. Az adatokat más célra gyűjteni vagy felhasználni tilos. Az elektronikus aláírás felhasználásával elkövetett bűncselekmény alapos gyanúja estén a felderítés vagy a hasonló bűncselekmények megelőzése céljából, vagy nemzetbiztonsági érdekből a nyomozó hatóság vagy a nemzetbiztonsági szolgálat megkeresésére az érintett személyi azonosságát igazoló adatok azonban kiadhatók. Így tehát, ha a rendőrség a szerzői és szomszédos jogok megsértése miatt nyomozást indít mondjuk a BSA feljelentése alapján, az elektronikus aláírás hitelesítők újabb adatbázist jelenthetnek a személyek azonosítására. A szolgáltató jogosult a szolgáltatási szerződés megkötésekor okmányaink *(személyi, útlevél, jogosítvány)* megtekintésére.

Ezek után kérdés, hogy mire jogosult az aláíró? Az aláíró jogosult az aláíráslétrehozó adatot birtokolni. Ezen túl az aláírónak csak kötelességei vannak: így köteles a szolgáltatót haladéktalanul tájékoztatni az azonosításához szükséges személyazonosító adatainak változásáról, vagy az általa képviselt szervezet adatainak változásáról, az aláíráslétrehozó adat illetéktelen személy birtokába jutásáról vagy elvesztéséről, az aláírással vagy az elektronikus okirattal kapcsolatban észlelt rendellenességről, a tanúsítvánnyal ellátott elektronikus dokumentummal kapcsolatos jogvita megindulásáról. Az aláírót az értesítési kötelezettsége elmulasztásából eredő károkért kártérítési kötelezettség terheli.

Az aláíró kérheti tanúsítványa felfüggesztését vagy visszavonását. A tanúsítvány érvényességét a szolgáltató felfüggeszti, vagy azt visszavonja akkor is, ha a szolgáltatással kapcsolatos rendellenességről szerez tudomást, megalapozottan feltételezhető, hogy a tanúsítványban foglalt adatok nem felelnek meg a valóságnak, vagy az aláíráslétrehozó adat nem az aláíró kizárólagos birtokában van, a Felügyelet jogerős és végrehajtható határozatában így rendelkezik, illetve ha a szolgáltató tevékenységét befejezi vagy a tanúsítvány érvényességi ideje lejár.

A szolgáltatók tevékenységét a Felügyelet ellenőrzi. E körben sok eszköz áll rendelkezésére, a helyszíni ellenőrzés során a szolgáltató figyelmének felhívásán túl a nyilvántartásból történő törlésig. Ez utóbbi, mint legsúlyosabb szankció mellett igen érzékenyen érintheti a szolgáltatókat a bírság is, amelynek összege ötvenezer forinttól tíz millió forintig terjedhet az elkövetett jogsértés jellegétől függően. Íme ez hát a szép új világ egyik jelentős vívmánya, az elektronikus aláírás. Bevallom, közelről számomra több akadály látszik, mint amit eredetileg elképzeltem, így félek tőle, hogy az alkalmazás kezdeti időszaka nem lesz zökkenőmentes. De hát a tálalást követően az étkezés ideje nem jött még el, így a kritikai hang is kicsit korai. Ígérem, a vacsora végén visszatérek a témára.

Dr. Mayer Erika
Nádas & Mayer Ügyvédi Iroda
mayer@nadas-mayer.hu

ASP suli *(VIII. rész)* a CDO programozása

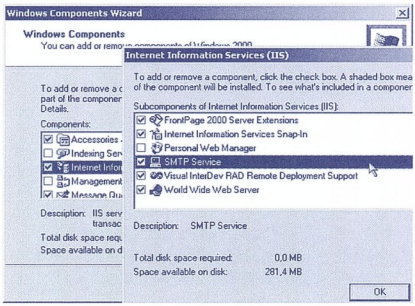
Ahogyan azt korábban megígértük, ma nyílt napot tartunk az ASP suliban. Olyan dologról lesz szó ugyanis, ami már nem kötődik szorosan az ASP programozáshoz *(és ezután már egyre több lesz az ilyen, „külső" objektum)*, ezért a következőkben leírtakat bátran ki lehet próbálni egyszerű parancssori scriptből, vagy akár „igazi" programozási nyelvből is. Az **[1]** címről letölthető példaprogramok a hagyomány kedvéért továbbra is ASP kódok, de megfelelő átalakítás *(az objektumok létrehozása, a ki- és bemenő adatok kezelésének testreszabása)* után más területen is használhatók.

A CDO család

A CDO *(Collaboration Data Objects)* általános feladata, hogy lehetővé teszi a levelezés, a hírcsoportok, és az egyes csoportmunka-szolgáltatások – például naptár – programból történő kezelését. Ezek közül cikkünkben egyedül az egyszerű levélküldésre térünk ki, akit a téma bővebben érdekel, az látogasson el az MSDN Library CDO mappájába **[2]**. Az objektumcsalád az évek során fejlődött, egyre több taggal bővült. A legelső széles körben elterjedt verzió a CDO 1.2.1 volt, ami az Exchange MAPI alapú levelezési és csoportmunka szolgáltatásainak kihasználását volt hivatott segíteni. Ezt a verziót az Exchange Server 5.5 és az Outlook 98 tartalmazta. A CDO 1.2.1 mellett megjelent egy „butított", egyszerűsített változat is, az CDO for NTS *(Collaboration Data Objects for Windows NT Server)*. Ez az objektummodell az Internet Information Server *(IIS4)* telepítésével már felkerül a rendszerre, így minden Windows NT 4.0-n használható, ahol az IIS is fut. Cikkünkben ez lesz az első objektummodell, amit kipróbálunk. Az Exchange és a CDO azóta jelentős fejlődésen ment keresztül. A CDO objektumkönyvtárak 2.0-s változata a Windows 2000-ben jelent meg, CDO for Windows 2000 néven *(ugyancsak minden Windows 2000-ben megtalálható)*. Az Exchange 2000 saját CDO-ja, a CDO for Exchange 2000 az előbbivel 100%-ban kompatibilis, annak csak funkciókészletét bővíti ki. A CDO könyvtárak új változatai már nem a MAPI-t, hanem az SMTP protokollt használják. A következő részben a CDO for Windows 2000 lesz terítéken.

CDO for Windows NT Server *(CDONTS)*

Ez a legegyszerűbb CDO objektummodell; segítségével akár két sorban is küldhetünk levelet. Mindehhez azonban a számítógépen futó SMTP szolgáltatás támogatása szükségeltetik. Ha belegondolunk, nem is baj, ha egy SMTP kiszolgáló átveszi a vállunkról a levelek továbbításának terhét: mi csak kiadjuk a parancsot, az pedig jó szolga módjára elküldi a levelet. A későbbi CDO verziók képesek „távoli" SMTP kiszolgálók használatára; a korábbi CDO verziók pedig helyi kiszolgáló híján hajlamosak voltak az érvényes MAPI kliens *(Outlook, Outlook Express)* segítségét kérni. Brrr. :-) Maradjunk annyiban: ha a CDONTS áldásos szolgáltatásait használni szeretnénk, telepítsük a gépünkre az IIS SMTP komponensét!



Ha telepítjük az IIS SMTP szolgáltatását, az leveszi a vállunkról a levélküldés terhét

A CDONTS már önmaga is egy butított változat, de még a CDONTS is tartalmaz olyan objektumot, amely egymaga mindent tud *(persze minimális szinten)*. Ezt az objektumot akár külön is szállíthatnák, hiszen nemhogy a CDO, de a CDONTS más objektumaihoz sincs semmi köze. Önálló lény: hirtelen levélküldésre való. Akár egy sorban. Ő a CDONTS.NewMail objektum. Lássuk *(nts1.asp)*:

```
Dim oNewMail

Set oNewMail =
  Server.CreateObject("CDONTS.NewMail")

oNewMail.Send "felado@ceg.hu", "cimzett@ceg.hu",
  "CDONTS NewMail demo", "Hello NewMail!", 0

Set oNewMail = Nothing
```

Ennyi. És működik. Négy sor, de csak mert jó gyerekek voltunk. Lássuk sorra: definiáljuk a változót, létrehozzuk a CDONTS.NewMail objektumot. Ezután meghívjuk az új objektum Send metódusát, végül megszüntetjük az objektumot. Az utolsó sor fontos; a NewMail objektum tiszavirág életű; mindegyik példánya egyetlen levél elküldésére képes, azután elhaláloz. Ha újra használni szeretnénk, új példányt kell létrehoznunk, különben hibaüzenetet kapunk. A Send() metódus paraméterei a következők lehetnek:

- Feladó – Az üzenet feladója. Írhatunk tisztán e-mail címet is, de ha akarjuk, az SMTP szabályok szerint megadhatjuk a feladó teljes nevét is, emígyen:

```
Teszt Elek <elek@ceg.hu>
```

- Címzett – Az üzenet címzettje(i). Ha több van, a címeket pontosvesszővel válasszuk el egymástól.
- Téma – A levél témája *(szöveges érték)*
- Törzs – A levél törzse *(szöveges érték, vagy IStream interfész, például ADODB.Stream objektum)*
- Fontosság – A levél fontossága. Ha normál, értéke 1, és ez az alapértelmezés is *(ha nem adjuk meg)*. Ha alacsony, értéke 0, ha pedig magas, 2.

A fenti paraméterek közül egyik megadása sem kötelező, ugyanis ha nem akarjuk egy sorban elintézni, a NewMail objektum jellemzőit a Send() meghívása előtt szépen fel lehet tölteni ugyanezekkel az adatokkal *(nts2.asp)*:

```
Set oNewMail =
  Server.CreateObject("CDONTS.NewMail")

oNewMail.From     = "Teszt Elek <elek@teszt.hu>"
oNewMail.To       = "Fu Beno <beno@teszt.hu>"
oNewMail.CC       = "Tar Janos <tjani@teszt.hu>"
oNewMail.Bcc      = "Gipsz Jakab <jgipsz@teszt.hu>"
oNewMail.Subject  = "CDONTS.NewMail Demo 2."
oNewMail.Body     = "Hello NewMail !"

oNewMail.Send
```

Küldjünk csatolt fájlt a levéllel! Ehhez nincs más dolgunk, mint a Send() metódus előtt meghívni az AttachFile() metódust, például így *(nts3.asp)*:

```
oNewMail.AttachFile Server.MapPath("attach.txt"),
  "ujnev.txt", 1
```

Az AttachFile() metódus három paramétere a következő:

- A csatolandó fájl neve. Ha a kódot ASP-ben futtatjuk, a Server.MapPath() segítségével keressük meg az igazi fájlnevet. Más esetben erre természetesen nincs szükség. Ezen kívül itt is megadhatunk IStream interfészt *(pl. egy ADODB.Stream objektumot)*, ami már közvetlenül a csatolandó adatokat tartalmazza.
- Fájlnév a levélben – a levélben megjelenő fájlnév nem feltétlenül egyezik meg az eredetivel. Nem kötelező megadni.
- Kódolási mód – 0 értéke esetén UUEncode, 1 esetén Base64 kódolással kerül az üzenetbe a csatolt állomány. Ha nem adjuk meg, az alapértelmezés érvényesül, ami pedig tisztán szöveges levelek esetén 0, MIME levelek esetén 1.

Ha további fejlécinformációkat szeretnénk csatolni a levélhez *(például, hogy hova kell küldeni a választ)*, használjuk a Value jellemzőt *(nts4.asp)*:

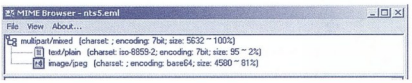
```
oNewMail.Value("Reply-To") = "valasz@teszt.hu"
```

Hódít a MIME

A levelek tartalmának további bonyolításához a MIME nyújt segítséget. Aki nem olvasta a magazin előző néhány számában bemutatott MIME cikksorozatot, annak ajánlom, hogy vesse bele magát, mielőtt vakon tapogatózva MIME leveleket kezdene készíteni. Aki olvasta, annak az elmélet után most talán jól jön egy kis gyakorlat :-). Egy NewMail objektum egyetlen sorral MIME levéllé alakítható *(nts5.asp)*:

```
oNewMail.MailFormat = 0 ' MIME
```

A MailFormat jellemző értéke 0, ha MIME, illetve 1, ha tisztán szöveges levelet szeretnénk küldeni. Ha a [3] címről letölthető MIME Browser *(vigyázat! csak Windows 2000-ren működik!)* segítségével megnyitjuk az nts5.asp által küldött levelet, láthatjuk a MIME formátumot és a – jelenleg még egyszerű – felépítését is *(az nts5.asp a levélhez csatol egy képfájlt is)*.



Egyszerű MIME levél, egyszerűen

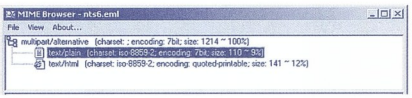
Ha már itt tartunk: küldjünk HTML levelet! Ehhez nincs más dolgunk, mint a levél törzsét eleve HTML-ben írni, valamint az eddigieket megspékelni plusz egy sorral *(nts6.asp)*:

```
oNewMail.BodyFormat = 0 ' HTML
oNewMail.MailFormat = 0 ' MIME
```

Nem véletlenül két sor az a plusz egy: hiszen aki járatos a MIME-ban, tudja, hogy minden HTML levél egyben MIME levél is. Hiába állítjuk a BodyFormat jellemzőt 0-ra *(ez jelzi a HTML tartalmat, ezek után az 1 nyilván a szöveges törzs kódja)*, ha a MailFormat nem MIME. Az nts6.asp a levél törzsébe ezt írja:

```
oNewMail.Body =
  "<h1><center>Egyszeru HTML level</center></h1>"
```

Ha az eredményként létrejött levelet megnyitjuk a MIME Browserrel, érdekes dologra lehetünk figyelmesek:



A NewMail automatikusan elkészíti a HTML törzs tiszta szöveges verzióját

Az üzenet – jó HTML levél módjára – multipart/alternative típusú, és a HTML formátum mellett *(egészen pontosan előtte)* a törzs tiszta szöveges változatát is tartalmazza. A NewMail objektum tehát automatikusan elkészíti nekünk a HTML levelek szöveges komponensét, és továbbítja azt az üzenettel. Ezután már csak egy lényeges szolgáltatása maradt a NewMail objektumnak, ez pedig a kapcsolódó komponensek kezelése. Magyarul: hogyan küldhetünk olyan HTML levelet, amiben a HTML oldal közepén, és nem csatolt fájlként szerepel egy kép? A megoldást az nts7.asp kódban láthatjuk; a kulcs az AttachURL() metódus:

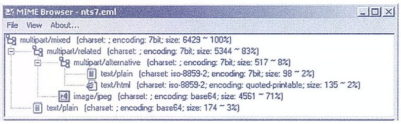
```
oNewMail.AttachURL Server.MapPath("msgirl.jpg"),
  "kep.jpg"
```

A metódus első paramétere a csatolandó fájl *(vagy Stream, szokás szerint)*, a második pedig egy „fantázianév". Ezt a nevet használhatjuk a levél HTML törzsében, ha fel szeret-

nénk használni a csatolt objektumot. A fenti képet tehát így illeszthetjük a HTML levelünk kellős közepébe:

```
oNewMail.Body =
↳ "<b>Itt a kep: <img src='kep.jpg'></b>"
```

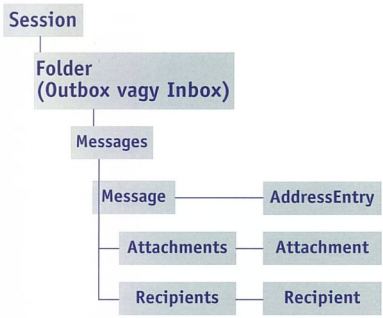
Mindezek mellett persze semmi sem akadályoz meg minket abban, hogy egyszerű csatolt fájlokat is hozzácsapjunk a levélhez, a már ismert módon. És az eredmény? A MIME cikkben bemutatotthoz hasonló, komoly *(de szabályos!)* MIME szörnyszülött, ami az ottanitól csak annyiban különbözik, hogy a levélben nincs digitális aláírás. Mindenesetre a struktúra meggyőző!



NewMail in action: hát kell ennél több?

A NewMail objektum tehát képes viszonylag összetett levelek létrehozására és elküldésére is. Sok mindent azonban nem tud: mindenekelőtt, szinte minden jellemzője csak írható *(és nem olvasható)*. Nem kezeli túl intelligensen a címlistákat *(feladó, címzettek)*, a csatolt fájlokat *(ha egyszer csatoltuk, azt már nem vonhatjuk vissza)* és nem utolsó sorban minden levélhez újra létre kell hoznunk az egész objektumot. Az objektum képességei – a korlátai figyelembe vételével – azonban sok esetben bőven kielégítik a felmerülő igényeket.

A CDONTS objektummodellje



A CDONTS objektummodellje már komoly hierarchiára épül

A CDONTS használatához meg kell ismernünk az objektummodellt *(aminek, mint látható, a NewMail objektum egyáltalán nem is része)*. Az objektummodell részletes dokumentációja megtalálható a **[4]** címen. A CDONTS kommunikáció egy Session *(munkamenet)* objektum létrehozásával kezdődik. Ilyenkor „bejelentkezünk" a rendszerbe, megadjuk a felhasználó nevét és e-mail címét. Az itt megadott adatok lesznek feltüntetve a kimenő levelek feladó mezőjében. A bejelentkezésnek a levél küldéséhez – úgy tűnhet –, nincs értelme, de a valóságban szükség lehet rá, ha Exchange környezetben dolgozunk, vagy ha esetleg nem levél küldésére, hanem fogadására készülünk. *(A CDONTS segítségével belenézhetünk az IIS SMTP szolgáltatására érkezett levelekbe is!)* Ilyenkor már nyilvánvaló, miért van szükség a bejelentkezésre. A *cdonts1.asp* bemutatja a levélküldés legegyszerűbb módját, e kód segítségével fogjuk bemutatni az objektummodell egyes elemeit.
Első dolgunk tehát a bejelentkezés:

```
Set oSession =
↳ Server.CreateObject("CDONTS.Session")
oSession.LogonSMTP "Teszt Elek", "elek@teszt.hu"
```

Létrehozunk egy Session objektumot *(ez a hierarchia csúcsa)*, és meghívjuk az objektum LogonSMTP() metódusát. A metódus két, kötelező paramétere a felhasználó neve és címe *(de az adatok érvényességét nem ellenőrzi a rendszer)*. Bejelentkezés nélkül a CDONTS objektummodell csaknem 100%-ban működésképtelen.
A következő lépés a mappa *(Folder objektum)* megnyitása. A CDONTS összesen kétféle mappát kezelhet: a bejövő *(Inbox)* és a kimenő *(Outbox)* levelek mappáját. Ezekhez a Session objektum jellemzőin, illetve metódusain keresztül férhetünk hozzá. A kimenő postaládához például így:

```
Set oFolder = oSession.Outbox
```

Ez egyébként egyenrangú a következővel:

```
Set oFolder = oSession.GetDefaultFolder( 2 )
```

A Session objektum GetDefaultFolder() metódusa visszaadja a paraméterben meghatározott alapértelmezett mappát. A paraméter lehet 1 *(Inbox)* és 2 *(Outbox)*.
Miután hozzájutottunk a mappához, az üzenetek kollekciójának *(Messages)* megszerzése a feladat. A Folder objektum tulajdonképpen mást nem is tud, mint ezt:

```
Set oMessages = oFolder.Messages
```

A Messages objektum egy kollekció, amely Message objektumokat *(azaz üzeneteket)* tartalmaz. Ennek megfelelően is viselkedik: van Add(),GetFirst(), GetLast(), GetPrevious(), GetNext() metódusa, amelyek mind-mind egy Message objektummal térnek vissza. A Delete() töröl egy adott üzenetet, a Count jellemző például visszaadja a kollekcióban található üzenetek számát.
Ha CDONTS-sel új levelet szeretnénk küldeni, akkor a kimenő postaládában kell új üzenetet létrehoznunk:

```
Set oNewMsg = oMessages.Add
```

És ebben a pillanatban a kezünkben van az üzenetobjektum, amit nincs más dolgunk, mint rendesen feltölteni, majd elküldeni. Az egyszerű téma és a tisztán szöveges törzs még csak-csak megy:

```
oNewMsg.Subject = "CDONTS Demo"
oNewMsg.Text = "CDONTS uzenet torzse."
```

... de a címzettekkel már gondunk lehet. A címzés ugyanis kicsit bonyolultabb, mint eddig volt. Minden üzenet tartalmaz egy Recipients *(„címzettek")* kollekciót, ami Recipient *(„címzett")* objektumokat tárol. Nincs külön To, Cc, Bcc kollekció; azt, hogy egy-egy címzett éppen milyen típusú, az adott Recipient objektum határozza meg. Egy hagyományos címzett felvételéhez tehát fogjuk az üzenet Recipients kollekcióját:

```
Set oRecipients = oNewMsg.Recipients
```

... majd a szokásos módon hozzáadunk egy új elemet:

```
Set oNewRecip = oRecipients.Add
```

A visszakapott objektum egy frissen létrehozott, üres címzett objektum. Ha megnézzük a Recipient objektumot, három lényeges jellemzőt találunk:

- Type – a címzett típusa. Értéke 1, ha To; 2, ha Cc; és 3 ha az adott címzett Bcc.
- Address – a címzett e-mail címe
- Name – a címzett neve

Ezeket a paramétereket kell kitöltenünk az újonnan létrehozott Recipient objektumon, és már kész is a címzés:

```
oNewRecip.Name = "Teszt Elek"
oNewRecip.Address = "elek@teszt.hu"
oNewRecip.Type = 1
```

Ezután nincs más dolgunk, mint elküldeni az üzenetet és szépen kijelentkezni a munkamenetből:

```
oNewMsg.Send()
oSession.Logoff()
```

Csatolt fájlok kezelése

A csatolt fájlok kezelése hasonló a címzettek felvételéhez: egy-egy csatolt fájlt *(legyen az beillesztett vagy ténylegesen csatolt adat)* egy-egy Attachment objektum tartalmaz. A csatolt fájlokat az üzenet Attachments kollekciója tárolja. A kollekcióhoz az Add*()* metódus segítségével adhatunk újabb tagot. Eldönthetjük, hogy egy üres Add*()* hívással üres Attachment objektumot hozunk létre, és aztán beállítgatjuk annak jellemzőit, vagy pedig egy megfelelően felparaméterezett Add*()* hívással „készen szállítjuk" a csatolt fájlt. *A cdonts2.asp* példában az érthetőség kedvéért előbbi megoldást választottuk, de akit érdekel, böngészse át az Add*()* metódus paraméterezését. Mindenekelőtt: ha már fájlokat csatolunk, állítsuk az üzenetet MIME formátumúra, majd következhet az Attachment létrehozása:

```
oNewMsg.MessageFormat = 0 ' MIME

Set oAttachments = oNewMsg.Attachments

Set oAttachment = oAttachments.Add()
oAttachment.Type = 1 ' CDOFileData
oAttachment.ReadFromFile
  Server.MapPath("attach.txt")
oAttachment.Name = "ujnev.txt"
```

A csatolmány típusa *(Type)* fájl *(1)*; az üzenetben „ujnev.txt" néven jelenik majd meg. Az fájl valódi tartalmát a ReadFromFile() metódus tölti be az Attachment objektumba.

HTML levelek

HTML levél küldése az előbbiek ismeretében már nemördöngösség. Az első és legfontosabb különbség, hogy a HTML levél törzsét nem a Text, hanem a HTMLText jellemzőn keresztül állítjuk be, valamint nem felejtjük el az üzenetet MIME típusúra állítani *(cdonts3.asp)*:

```
oNewMsg.MessageFormat = 0 ' MIME
oNewMsg.HTMLText = "<i>CDONTS HTML uzenet.</i>"
```

Ha pedig üzenetbe illesztett objektumot *(mondjuk képet)* szeretnénk küldeni, akkor – a NewMail objektumnál bemutatott példához hasonlóan – két dolgot kell tennünk. Először is, létre kell hozni egy speciális csatolt fájlt az üzenetben, amit ellátunk egy egyedi azonosítóval. Másodszor pedig, a HTML üzenet törzsében ezt az egyedi azonosítót kell használnunk *(cdonts4.asp)*:

```
oNewMsg.HTMLText =
  "<i>Kepet ide: <img src='kep.jpg'></i>"

Set oAttachment = oAttachments.Add("kep.jpg",
  1, Server.MapPath("msgirl.jpg"), "kep.jpg")
```

Az Add metódus általunk használt paraméterei sorrendben a következők:

- Név – a csatolt fájl neve, ami a levélben megjelenik
- Típus – csatolt fájl *(1)*, vagy csatolt üzenet *(4)*
- Forrás – a Típustól függően vagy a csatolni kívánt fájl neve, vagy pedig a csatolni kívánt üzenet
- ContentLocation – a ContentLocation MIME fejléc; a levelezőprogram az itt megadott azonosító *(„kep.jpg")* alapján ismeri fel a HTML kódba beillesztendő csatolt üzenetrészt

Ennyit a CDONTS-ről. A fenti példa elkészíti és el is küldi ugyan a HTML levelet, de a megvalósítás kicsit sántít. Aki kitalálja, hogy a *cdonts4.asp* és az *nts7.asp* miért nem azonos tartalmú levelet küld, és elküldi nekünk a megfelelő CDONTS kódot, nyereményre számíthat! :-)

Legközelebb a CDO for Windows 2000 objektummodell rejtelmeibe ássunk bele magunkat, addig is jó levelezést!

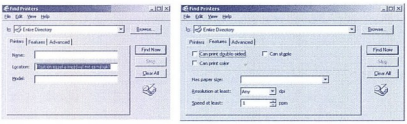
Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:

[1] http://technet.netacademia.net/feladatok/asp/8
[2] http://msdn.microsoft.com/library/en-us/cdo/html/_olemsg_overview_of_cdo.asp
[3] http://technet.netacademia.net/download/MIME/
[4] http://msdn.microsoft.com/library/en-us/cdo/html/_denali_cdo_for_nts_object_model.asp

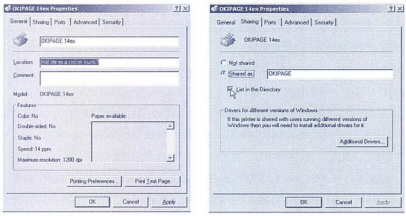
AD trükkök: Nyomtatók publikálása

Az Active Directory számtalan objektumtípus tárolására használható: felhasználók, csoportok, számítógépek, nyomtatók, megosztott könyvtárak stb. helyezhetők el logikus, hierarchiába szervezett módon. Ezen objektumtípusok közül szinte mindegyik „adja magát", könnyű őket létrehozni, megtalálni, módosítani, átszervezni. A nyomtatók kezelése azonban valamilyen rejtélyes okból fogva nehézkesre sikeredett: a publikált nyomtató ugyan kereshető, de sejtelmünk sincs, hol a csudában helyezkedik el a hierarchiában. Elvileg telephelyek szerint is könnyedén rábukkanhatunk a hozzánk legközelebb eső printerre, de az AD nem kínálja fel a helyszíneket. Lássuk csak a nyomtatókeresési ablakot *(Start menü->Search...->For Printers)*:



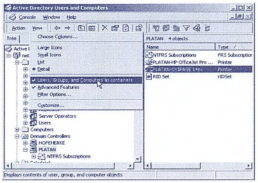
A nyomtatókereső párbeszédpanel két arca.

A baloldali ábrán jól megfigyelhető, hogy a helyszín kiválasztásában nem kapunk segítséget, míg a jobboldali arról tanúskodik, hogy azért a nyomtatómeghajtók alapos munkát végeznek: publikáláskor az összes tulajdonságuk *(színes? kétoldalas? tűzőgépes?)* automatikusan bekerül a címtárba. De pontosan hova? Ez a kérdés azért lényeges, hogy az egyes nyomtatókat szét tudjuk teríteni a megfelelő szervezeti egységekbe, de amíg nem látszódnak az Active Directory Users and Computersben *(ADUC)*, addig az egérrel is viszonylag nehéz eltalálni őket :)
Első lépésként azt kell biztosítanunk, hogy a nyomtató valóban bekerüljön a címtárba. Ezt a megosztási ablakban pipálhatjuk be *(mellette balról az egyelőre használhatatlan Location információ)*:



Bal oldal: a helyszín beállítása (lásd később) jobb oldal: nyomtató publikálása a címtárba

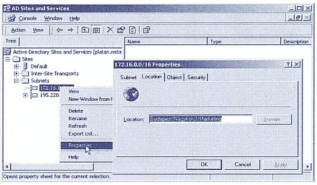
Ezután az ADUC megfelelő nézetének beállításával meg is tekinthetjük a nyomtatót a címtárban! Titka: View menü -> Users, Groups and Computers as containers! Magyarul: mutasd a *(felsorolt)* objektumokat úgy, mintha mappák lennének, lenne bennük ez + az!



És lőn. A számítógépünk alatt – sok egyéb mellett, melyekkel most nem foglalkozunk - megjelenik az imént publikált nyomtató:

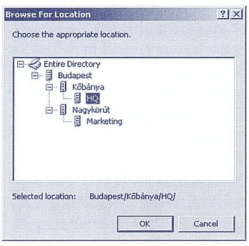
A számítógép - mint mappa - elég sok dolgot tartalmaz

Kezünkben az egér, és a kulcs ahhoz, hogy jobbra-balra hurcibáljuk a nyomtatót a címtárban. Már csak egyetlen dolog hiányzik a boldogságunkhoz, ez pedig a nyomtatók keresése földrajzi elhelyezkedésük szerint. Első lépésként az Active Directory Sites and Services *(ADSS)* segítségével fel kell vennünk a telephelyek *(egészen pontosan a telephelyeket alkotó IP Subnetek)* evilági elnevezését.



IP subnetek elhelyezése a földtekén ország/város/stb

Jól megfigyelhető, hogy a Browse gomb szürke, azaz egyelőre a rendszer nem vesz tudomást a bütykölésünkről. Hátra van még ugyanis a legtitkosabb lépés: a megfelelő házirend *(Prepopulate printer search location text)* engedélyezése! Javaslatom szerint a legalkalmasabb hely e policy bebillentésére maga a teljes tartomány, így biztosítható, hogy a nyomtatólokátor nem fog *(látszólag)* következetlenül működni. Ezután mars vissza a nyomtató tulajdonságai közé, ahol immár szépen világít a Browse gomb, hogy beállíthassuk a helyszínt. A „Mi a szösz" felirat helyett immár válogathatunk az ADSS-sel imént felvett helyszínek között. Amit kiválasztunk, az válik a nyomtató helyszínévé, s erre akár kereshetünk is a cikk elején mutatott módszerrel, hisz ott is megjelenik a Browse gomb.



F.M.

WebDAV

WebDAV – az IIS, mint webalapú fájlkiszolgáló

Cikkünkben az Internet Information Services 5.0 egyik új szolgáltatását mutatjuk be. A WebDAV *(Distributed Authoring and Versioning for the World Wide Web)* protokoll segítségével a webkiszolgálónk komplett fájlszerverré alakulhat át, megfelelő alternatívát nyújtva a biztonsági szempontból egyaránt kifogásolható FTP és Windows hálózati szolgáltatásokkal szemben. A példák, NetMon kommunikációs naplók, demonstrációs anyagok *(a cikkben dőlt betűvel szedett fájlok)* az **[1]** címről tölthetők le *(WebDAV mappa, olvasásra bárki megnyithatja – lehet gyakorolni :-))*.

Mire képes a WebDAV?

Mindenekelőtt: akit a WebDAV protokoll teljes leírása érdekel, olvassa el az eredeti RFC 2518 **[2]** specifikációt. Cikkünkben röviden összefoglaljuk a WebDAV IIS5-ben megvalósított funkcióit, és bemutatjuk használatát. A WebDAV lehetővé teszi számunkra, hogy:

- erőforrásokat, dokumentumokat kezeljünk *(hozzunk létre, olvassunk, módosítsunk, töröljünk, másoljunk, stb.)* a *(web)*kiszolgálón megosztott WebDAV-könyvtárakban
- lekérdezzük, módosítsuk a dokumentumok szokásos jellemzőit *(dátum, méret, stb.)*
- a dokumentumok egyidejű módosításának elkerülése érdekében a dokumentumot a kiszolgálón zároljuk – míg egy dokumentum zárolt, csak olvasásra érhető el
- keressünk a dokumentumokban

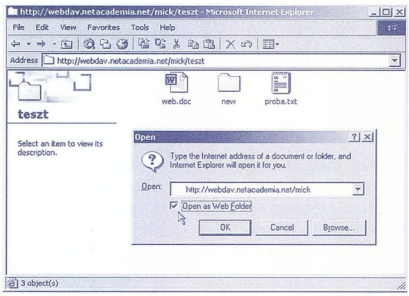
Teszi mindezt úgy, hogy a munka során

- a HTTP protokollon keresztül működik *(a szabványos porton, így a tűzfalakon általában „átmegy")*
- a szabványos parancsokat hét új paranccsal egészíti ki *(ezekről később)*
- az adatokat korszerű XML formátumban kezeli

Mire van szükség a WebDAV használatához?

Természetesen szükséges egy WebDAV kiszolgáló *(esetünkben az IIS5)*; a felhasználói oldalon pedig keresnünk kell a WebDAV-ot támogató alkalmazásokat. Szerencsére ilyenek bőven akadnak, most csak felsorolásképpen:

- Maga a Windows képes WebDAV könyvtárak megnyitására. Windows 2000-ben a My Network Places-ben hozhatunk létre új helyet, ami WebDAV könyvtárra mutat *(csak írjuk a cím elé a http:// előtagot)*
- Régebbi Windows verziók *(W9x, NT 4)* az Internet Explorer 4 telepítése után a My Computer mappában található Web Folders ikon segítségével tehetik meg ugyanezt
- A hamarosan megjelenő Windows XP *(és egyelőre sajnos csak az)* lehetővé teszi, hogy WebDAV mappát hálózati meghajtóhoz rendeljünk hozzá. A dolog egyetlen hátulütője, hogy ez a megoldás SSL titkosított *(https://)* mappákhoz nem használható
- Az Office 2000 család és utódainak alkalmazásai közvetlenül képesek WebDAV mappákkal dolgozni
- Az Internet Explorer 5.0 verziótól kezdve segíti a WebDAV szolgáltatások használatát. Ha egy könyvtárat nem böngészőben, hanem mappaként szeretnénk megnyitni, az IE File / Open ... parancsára megjelenő dialógusablakba gépeljük be az elérési utat, majd kattintsunk az „Open as Web Folder" jelölőnégyzetre. Az IE ezután a címet webmappaként nyitja meg *(lásd a következő ábrát)*.
- Külső gyártók WebDAV kliensei, amelyek vagy közvetlen elérést, vagy pedig a WebDAV mappák hálózati meghajtóhoz való hozzárendelését segítik – több-kevesebb sikerrel



Háttérben WebDAV mappa az Internet Explorerben megnyitva; az előtérben a kurzor a webmappa megnyitásához szükséges jelölőnégyzetre mutat

A WebDAV parancsok

Mint azt már említettük, a WebDAV a szabványos HTTP csatornán kommunikál, a HTTP 1.1 **[3]** bővítményeként viselkedik. Bár a WebDAV használatához e parancsok ismerete nem feltétlenül szükséges *(a felhasználói felület elrejti előlünk a teljes kommunikációt)*, úgy gondolom, érdemes ismerni őket – hiszen például a webkiszolgáló naplófájljaiban biztosan összetalálkozunk velük. A WebDAV-os kommunikáció során szerepet kapnak az eredeti HTTP parancsok:

- GET, HEAD: erőforrás, dokumentum letöltése. HEAD kérés esetén a dokumentum törzse nem, csak annak fejlécinformációi jutnak el az ügyfélhez *(ez például akkor hasznos, amikor egy fájl már a helyi gyorsítótárban van, csak ellenőrizni szeretnénk, hogy nem változott-e meg a kiszolgálón – ilyenkor felesleges a teljes fájlt letölteni, elég az utolsó módosítás dátuma, vagy egy ellenőrzőösszeg)*. A HEAD parancsot a WebDAV például egy-egy fájl meglétének ellenőrzésére használja előszeretettel.
- PUT: erőforrás, dokumentum feltöltése a kiszolgálóra
- DELETE: erőforrás törlése a kiszolgálóról. Mappák törlése természetesen a tartalmuk elvesztésével jár.

- OPTIONS: a használható parancsok lekérdezése.

A WebDAV ezeken kívül további parancsokat is definiál:

- COPY: erőforrás másolása. A másolás forrása a COPY parancs paramétere, a cél pedig a HTTP kérésben található Destination fejléc értéke. Fájl és mappa egyaránt másolható. Példaképpen lássunk egy másolási kérést *(copyhttp.txt)*:

```
COPY /mick/teszt/proba.txt HTTP/1.1
Content-Language: en-us
Accept-Language: hu, en-us;q=0.2
Overwrite: F
Destination: http://webdav.netacademia.net/
↳ mick/teszt/proba%20(2).txt
Translate: f
User-Agent: Microsoft Data Access Internet
↳ Publishing Provider DAV
Host: webdav.netacademia.net
Content-Length: 0
Connection: Keep-Alive
```

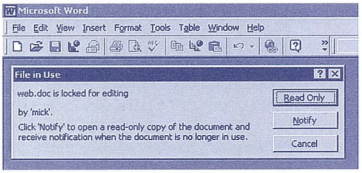
Az első sorban látható a COPY parancs, valamint annak paramétere – a fájl, amit másolni szeretnénk. A Destination fejléc értéke az új fájl neve *(...proba (2).txt; a szóközt URL-kódolásban (%20) tartalmazza)*. Az Overwrite: fejléc hamis értéke *(F)* azt jelzi a kiszolgálónak, hogy ha a célfájl már létezik, ne írja felül *(ilyenkor 412 Precondition Failed hiba érkezik)*. Ha ez a fejléc nincs a kérésben, a cél mindig felülíródik. Vessünk egy pillantást a User-Agent: fejléc értékére. Itt általában a kliensalkalmazás *(böngésző)* azonosítója szerepel; most a WebDAV kliens mutatkozik be szépen. Ne csodálkozzunk tehát, ha a kiszolgáló naplójában ilyesmit találunk.

- MOVE: erőforrás mozgatása, fájl és mappa egyaránt mozgatható. Az átnevezés is MOVE műveletet generál. A MOVE parancs a COPY-hoz hasonlóan működik, azzal a különbséggel, hogy a MOVE esetén a forrásállomány törlődik.
- MKCOL: új kollekció *(mappa, könyvtár)* létrehozása.
- PROPFIND: ez a parancs a WebDAV lelke. Erőforrások tulajdonságainak lekérdezésére való, de a PROPFIND parancs segítségével történik például a mappák tartalmának lekérdezése is. A kiszolgáló a választ XML formában küldi vissza. A *propfind0.xml* egy WebDAV mappa 0 mélységű, a *propfind1.xml* pedig egy 1 mélységű lekérdezésére adott válasz; utóbbiban az is látható, hogy a mappa mit tartalmaz.
- PROPPATCH: erőforrás tulajdonságainak *(szerző, cím, stb.)* módosítása.
- LOCK és UNLOCK: erőforrások zárolása és feloldása, részletesebben lásd a következő bekezdést.

Konfliktusok nélkül

Minden közös dokumentumtárban megoldandó probléma a konfliktusok *(egyazon dokumentum többszörös, egyidejű módosítása)* kezelése. Mi történjen, ha egyszerre ketten nyitnak meg egy dokumentumot, mindketten – egymástól függetlenül – módosítják azt, majd elmentik? Biztos, hogy jó megoldás az, hogy aki később mentett, az győz? Egyáltalán nem.

A WebDAV ezt a problémát az erőforrások zárolásával oldja meg. Amikor valaki szerkesztésre megnyit egy dokumentumot, zárolhatja azt (a LOCK) parancs segítségével. Ha valaki egy zárolt dokumentumhoz próbál hozzáférni, figyelmeztetést kap, és legfeljebb olvasási joggal nyithatja meg azt.



Zárolt dokumentumot csak olvasásra nyithatunk meg – mondja a Word

Amikor a felhasználó befejezi a munkát, *(az UNLOCK parancs segítségével)* feloldja a dokumentum zárolását, és attól a pillanattól kezdve mások is módosíthatják azt – persze egyidejűleg mindig csak egy valaki. A felhasználónak egyébként ezzel sem kell közvetlenül foglalkoznia, mert az alkalmazás elvégzi a szükséges zárolási feladatokat.

A dokumentumok zárolásának valódi folyamata persze ennél kicsit összetettebb. Először is, minden zárolásnak van egy lejárati ideje, hogy ha az ügyféllel bármi történik, a zárolt dokumentum ne maradjon „örökre" kalodában. Ezt az értéket maga az ügyfél állítja be a zárolás kérésekor, a LOCK parancs paramétereként átadott XML fájlban, valahogy így *(lockreq.xml)*:

```
<?xml version="1.0" encoding="UTF-8" ?>
<lockinfo xmlns="DAV:">
    <locktype>
        <write />
    </locktype>
    <lockscope>
        <exclusive />
    </lockscope>
    <owner>mick</owner>
</lockinfo>
```

Azaz, írásra (write) és exkluzív zárolási jogot kérünk; ezen kívül elküldjük még a zárolás tulajdonosának nevét is *(nézzük csak meg a fenti ábrát!)*. A kiszolgáló a sikeres zárolás után természetesen XML-ben felel *(lockresp.xml)*.

Ha megnyitjuk a fenti fájlt, a válaszból látható, hogy a zárolás sikeres volt. Visszakaptunk egy egyedi azonosítót, később ezzel hivatkozhatunk a zárolásra; a timeout érték pedig jelzi, hogy a zárolás 180 másodpercig érvényes. Ha a zárolást ezután is fenn szeretnénk tartani, a zárolást még az idő lejárta előtt *(egy újabb LOCK paranccsal)* meg kell újítani. A munka befejeztével *(a dokumentum bezárásakor)* a Word automatikusan UNLOCK parancsot küld a kiszolgálónak, ami felszabadítja az UNLOCK parancs fejlécében átadott azonosítójú zárolást.

Néhány WebDAV által használt HTTP fejléc

- DAV: a használt DAV séma verzióazonosítója
- Depth: a legtöbb parancsnál megjelenő fejléc; értékétől függ, hogy a parancs csak az adott objektumra, vagy annak gyermekeire is érvényes. Ha értéke 0, csak az adott erőforrásról beszélünk; ha értéke 1, akkor az erőforrásról és közvetlen gyermekeiről *(a PROPFIND pa-*

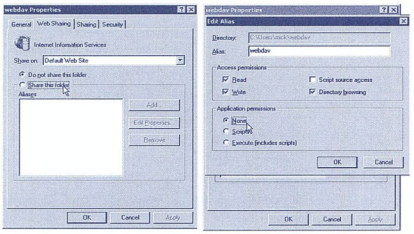
rancs például így listázza ki a mappa tartalmát), ha pedig „infinity", akkor a parancs az objektumra, valamint összes leszármazottjára érvényes. Néhány parancs alapértelmezett Depth értéket használ *(a DELETE például értelemszerűen az infinity-t)*.

- Destination: a COPY és MOVE parancs használata esetén a másolandó erőforrás célcímét tartalmazza
- Lock-Token: a zárolási azonosítót tartalmazza *(UNLOCK parancs)*
- Overwrite: a COPY ill. MOVE művelet esetén használt; „F" esetén a cél – ha létezik – nem írható felül, míg „T" esetén a felülírás megengedett.
- Timeout: Zárolás *(LOCK)* esetén itt határozhatjuk meg a zárolás élettartamát.

A WebDAV teljes parancskészletének, az általa használt HTTP fejlécek, státuszkódok, a parancsok és válaszok törzsében található XML elemek leírása a RFC 2518 WebDAV **[2]** szabványban található meg.

Mappák megosztása

Ennyi alapozás után végre elkezdhetjük használni a webmappákat: először azt nézzük meg, hogyan oszthatunk meg egy közönséges könyvtárat. Ha megnézzük a könyvtár tulajdonságlapját, észrevehetjük, hogy a fülek között a „Sharing" mellett szerepel egy "Web Sharing" oldal is.

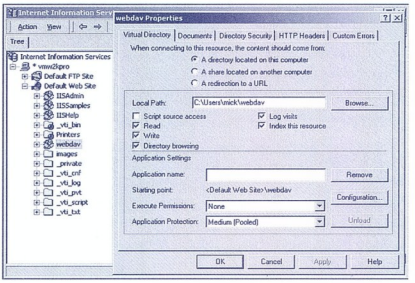


Mappa megosztása

Egy mappát több néven *(sőt, ha a kiszolgálón több web létezik, több weben is)* megoszthatunk. A jobboldali ábrán látható dialógusban meghatározhatjuk, hogy a mappa milyen néven, milyen jogosultságokkal tűnjön fel a webkiszolgálón. Az első és legfontosabb a mappa saját NTFS biztonsági beállítása – a mappa tulajdonságlapjának a Security oldalán állíthatjuk be. Ellenőrizzük az itt megadott jogokat, mert az Everyone – Full Control alapértelmezés nem tartozik a biztonságos beállítások közé. Ha az anonymous hozzáférést engedélyezni szeretnénk, adjunk megfelelő jogokat az IUSR_<számítógépnév> felhasználónak. Ezután következhet a globális, egész mappára érvényes WebDAV hozzáférési jogosultságok beállítása, amelyek a következők lehetnek:

- Read – a mappa WebDAV felületen keresztül olvasható
- Write – a mappa tartalma WebDAV felületen keresztül írható
- Directory Browsing – a mappa tartalma WebDAV felületen keresztül böngészhető
- Script Source Access – hozzáférés a futtatható fájlok tartalmához. Ha ezt nem engedélyezzük, az Execute Permissions alatt beállított állományok nem letöltődnek, hanem a kiszolgálón végrehajtódnak. Az Execute Permissions None értéke esetén ennek beállítására nincs szükség, a mappa teljes fájlkiszolgálóként üzemel. Ha a WebDAV könyvtárunk egyben „hagyományos" web része is, szükség lehet a scriptek futtatására, ilyenkor az Execute Permissions értékét állítsuk Script-re, és a Script Source Access-t pedig engedélyezzük. Ha az Execute Permissions értéke nem None, és a globális jogosultságok között az írás *(Write)* is engedélyezve van, legyünk nagyon óvatosak a mappa NTFS jogainak beállításaival. Hibás beállítás esetén ugyanis előfordulhat, hogy a felhasználó képes bármilyen fájlt feltölteni, majd futtatni a kiszolgálón!

Akinek ezek a beállítások valahonnan már ismerősek, nem téved: az előbbiekben virtuális alkönyvtárt hoztunk létre a webkiszolgálón.

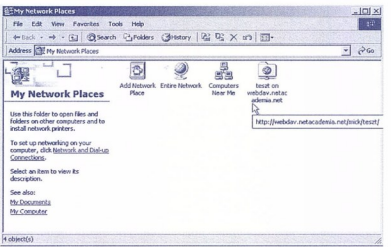


A webmappa megosztása nem más, mint egy virtuális alkönyvtár létrehozása

Az IIS konzolban magunk is létrehozhatunk új alkönyvtárakat, vagy ellenőrizhetjük, módosíthatjuk a meglévők beállításait. Miután pedig a WebDAV mappák tulajdonképpen az IIS web alkönyvtárai, a velük végzett műveletek bekerülnek az IIS naplójába.

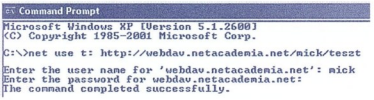
Hozzáférés a webmappákhoz – az ügyféloldal

Az ügyfél feladata ezután már csak az, hogy csatlakozzon ezekhez a megosztott mappákhoz. A webmappák megnyitását Internet Explorer-en keresztül a cikk elején már egy kép erejéig bemutattuk *(File / Open / Open as Web Folder)*. Ha meguntuk mindig kézzel begépelni a megnyitni kívánt címet, a My Network Places alatt létrehozhatunk állandó hivatkozást is. Az „Add Network Place" ikonra kattintva elindul a varázsló, amely néhány lényegre törő kérdés után létrehozza nekünk a kívánt parancsikont:



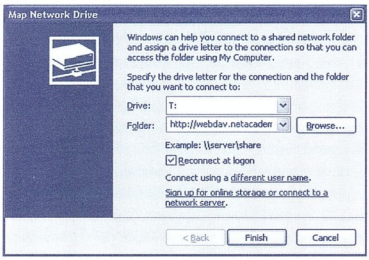
Webmappa-hivatkozás a My Network Places alatt

Miután ezt a hivatkozást létrehoztuk, a mappa a fájlrendszer részévé válik. Windows 9x és Windows NT 4.0 esetén az eljárás hasonló, ott – legalább Internet Explorer 4.0 telepítése után – a My Computer alatt keressük meg a Web Folders ikont. Sajnos a webmappákhoz egyelőre nem rendelhetünk hálózati meghajtót, így pl. a DOS-os alkalmazások nem részesülhetnek az előnyökből, de ez már nem tart sokáig: jön a Windows XP, ahol redmondi barátaink már megengedik, hogy hálózati meghajtóhoz WebDAV mappát rendeljünk. Tessék csak nézni:



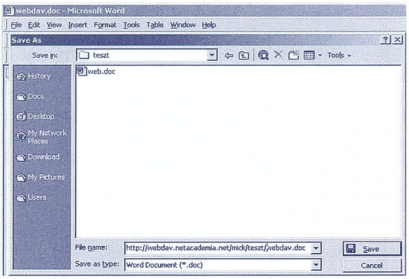
Szentségtörés Windows XP-ben: net use és http ?!

Természetesen nem kötelező mindezt parancssorból művelni, használhatjuk a Map Network Drive varázslót is:



Map WebDAV Drive :-)

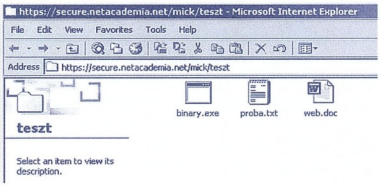
Említettük azt is, hogy az Office alkalmazások teljeskörű WebDAV-támogatást tartalmaznak *(a FrontPage például a kiszolgálóhoz való csatlakozáskor ellenőrzi, hogy a WebDAV rendelkezésre áll-e, és ha igen, akkor a saját fájlátviteli protokollja helyett ezt használja)*. Ha például egy Word dokumentumot webmappába szeretnénk menteni, ne habozzunk, adjuk meg az elérési utat és hajrá!



Az Office alkalmazások 2000 óta WebDAV-kompatibilisek

A WebDAV kommunikáció biztonsága

Két fő területet kell megvizsgálni: a felhasználóazonosítást, valamint az átvitt adatok biztonságát. A felhasználóazonosítás megegyezik az IIS beépített felhasználóazonosítási módszereivel; ha Internet Explorer-t használunk, a beépített Windows felhasználóazonosítás valamelyest biztonságosnak mondható *(mindenesetre biztonságosabbnak, mint például az FTP protokoll nyílt jelszavas azonosítása)*. Az átvitt forgalom alapértelmezésben nem titkosított. Azonban mind az IIS, mind a böngésző lehetővé teszi az SSL kapcsolatok használatát *(ezek kiszolgálóoldali telepítéséről következő számunkban lesz szó)*. Ha tehát egy https:// webmappa a rendelkezésünkre áll, azt minden további nélkül megnyithatjuk, használhatjuk, miközben mind a felhasználói, mind az átvitt adatok biztonságban vannak. Ehhez mindössze annyit kell tennünk, hogy a megnyitandó WebDAV mappa elérési útjába http:// helyett https://-t írunk *(Windows XP-n https:// webmappához sajnos nem rendelhetünk hálózati meghajtót, de minden más módszer működik ott is)*.



Titkosított WebDAV kapcsolat egy kattintással

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:
[1] http://technet.netacademia.net/download/WebDAV
[2] http://www.ietf.org/rfc/rfc2518.txt
[3] http://www.ietf.org/rfc/rfc2616.txt

Fuss Forrest, fuss! – avagy SQL lekérdezések optimalizálása

Bevezetés

Többen kérték, hogy írjak az SQL Server lekérdezések teljesítményoptimalizálásáról. A téma az egyik személyes kedvencem, pont emiatt vissza is kell fognom magam. Rengeteg Query Tuning-ról találhat cikket a kedves olvasó a Microsoft website-ján, amelyeket nem akarom megismételni, a cikk végén található URL-ekre elevickélve bárki elolvashatja őket. Inkább ad-hoc jelleggel olyan optimalizáláshoz kapcsolódó ismereteket *(is)* szeretnék megosztani a Kedves Olvasóval, amivel nem nagyon fog találkozni a megadott linkeken.

A mindent eldöntő indulás: a tervezési fázis

A tervezési szakasz az, ami a legnagyobb befolyást gyakorolja az elkészült adatbázis használhatóságára és sebességére. Általában nagyon nehéz egy, a tervezés során elrontott adatbázisból építkezve teljesítményt *(és adatokat)* kihozni, ezért különösen hangsúlyos a teljesítményhangolás szempontjából a megfelelő kiindulási alap.

Minden adatbázis élete úgy indul, hogy megszületik az igény: bizonyos üzleti adatokat adatbáziskezelő program segítségével, struktúrált módon szeretnének tárolni. A kezelendő adatoknak általában van egy belső logikája, ami magából a kiinduló üzleti folyamatból következik. Az adatbázis tervezése során összegyűjtjük a modellezendő valódi vagy fogalmi szinten létező objektumokat *(entitiásokat)*, és ezeket relációs adatbázisokban táblákkal modellezzük. Az entitások tulajdonságait a táblák oszlopaival írjuk le. A való életről adatbázisra történő leképezés eléggé intuitív folyamat, és ha ugyanazt a problémát több ember, egymástól függetlenül elkezdi modellezni, valószínű, hogy hasonló, de nem teljesen azonos adatbázisokat kapunk eredményül. Ez általában nem probléma, mert legtöbbször első körben még nem a végleges struktúrát alkotjuk meg.

Az entitások azonosítása után látunk neki az adatbázis normalizálásának, melynek célja a redundancia eltávolítása az adatok funkcionális függőségének elemzésével, a táblák módosításával és új táblák bevezetésével. A normalizálási folyamatnak több lépcsője van, melyeket normálformáknak hívunk, és 1-től 5-ig számozzuk őket. Általában a 3-as vagy afeletti állapotban levő adatbázisokat normalizáltaknak hívjuk.

A normalizálási folyamat vége általában még nem végállomás. Egy normalizált adatbázisban kevés redundáns információ van, így optimális a tárolási igénye, viszont az adatok nagyon sok táblára vannak szétosztva. Ez előnyös a módosítások szempontjából, mert a táblák sorai rövidek, így kis adatmennyiségekkel kell az SQL Servernek foglalkoznia, és egy adatlap *(8k)* felolvasásával sok információt el tud érni a kiszolgáló. A módosításokra optimalizált adatbázisokat Online Transaction Processing *(OLTP)* rendszereknek hívjuk. Egy normalizált alkalmazás majdnem kész erre a működésre. A gyakorlatban természetesen nincsnek tisztán OLTP adatbázisok, hisz mit érünk egy olyan adatbázissal, amibe csak lapátoljuk befelé az információkat, de sose olvassuk ki őket onnan? A másik adatbázistípus, amiben ritkán történnek módosítások, viszont az adatokat sokszor és sokféle szempont szerint kérdezik le: Decision Support System *(DSS)* a nevük. Az SQL Server OLAP szolgáltatása ezt a fajta működést támogatja.

DSS elemzés céljára általában nem ideális egy normalizált adatbázis, mert a lekérdezések csak sok JOIN árán tudják összeszedni a szükséges információkat, azaz általában lassúak lennének. Ilyenkor nyúlunk a denormalizáláshoz, amely során mesterségesen viszünk be némi rendundanciát az előzetesen már normalizált adatbásunkba. Így ugyan nehézkesebbé és lassúbbá válik az adatok módosítása, de a lekérdezések sokkal gyorsabbakká válhatnak, köszönhetően az előre kiszámolt *(redundáns)* adatoknak.

A hétköznapi adatbázisok általában félig OLTP, félig DSS működésűek, azaz vannak benne olyan információk, amelyeket sűrűn módosítanak, és vannak, amelyeket sűrűn kérdeznek le. Ha a kétféle igény által érintett táblák különbözőek, akkor nincs nagy gond, a DSS rész által igényelt lekérdezésekhez hangolva denormalizáljuk az adatbázis ezen részét, míg a gyakran módosított részeket meghagyjuk normalizált állapotban.

A gyakorlatban sajnos a kétféle funkciót sokszor ugyanazon táblákon kell megvalósítani. Ilyenkor jönnek a kompromisszumos megoldások, és megpróbáljuk az időket úgy szétosztani, hogy a lekérdezések eléggé gyorsak legyenek, de a módosítások is lefussanak elfogadható idő alatt.

Az eddigiekből következő nagyon fontos tervezési szempontok:

Egy jól teljesítő adatbázis tervezéséhez nem elég az entitások pontos azonosítása és modellezése, hanem figyelembe kell venni a várható terhelés jellegét is. Mely entitások mely tulajdonságát fogják sűrűn módosítani? Milyen adatokat kérdeznek le sűrűn? Milyen jellegű adatokat kell sűrűn aggregált eredményként visszaadni?

Ezen információk kinyerhetők az üzleti követelményekből. Meg kell tudjuk előre mondani, mely táblákat fogják naponta többezerszer módosítani. Át kell gondolni, melyek lesznek azok a lekérdezések, amelyeknek a végrehajtása időkritikus lesz, mert például emberek fogják felhasználni a kimenetét, és nekik fontos a gyors válaszidő.

Remélem az eddigiekből már látszik, hogy a jó adatbázistervezés legfontosabb feltétele, hogy jól ismerjük a modellezendő folyamatot, és így azokat a részeket optimalizáljuk már a tervezési fázisban, amelyeket sűrűn használnak, vagy nagyon szem előtt vannak. Ha egy éjszakánként generálódó riport generálását sikerül 4 óráról 3 órára felgyorsítani lehet, hogy megdicsérnek, de igazából az senkit nem érdekel. De ha az ügyfélszolgálatnál egy keresés idejét lehúzzuk mondjuk 3 másodpercről 0.5 mp-re, akkor nagyon sok hálás hangot fogunk hallani. Ez nem tisztán mérnöki munka. Nem lehet azt

mondani, hogy egy adatbázis úgy van tervezve, hogy a lehető leggyorsabb legyen *(sokszor ez a kérés)*. Azt lehet kijelenteni, hogy ebben az adatbázisban az ilyen és ilyen jellegű terhelés esetén optimális válaszidőket fogunk kapni. Ebből következik, hogy ha menet közben akár mi, akár más továbbfejleszti a rendszerünket a kibővült vagy megváltozott üzleti igényeknek megfelelően, akkor az optimális tervünk lehet, hogy elkezd köhögni és lassulni. Ilyenkor indexekkel és néhány egyéb trükkel még lehet, hogy utána tudjuk húzni az adatbázist, de előbb-utóbb eljön az a pont, amikor nincs mit tenni, az új igényeknek megfelelően az alapoktól át kell tervezni a rendszert. Ez általában fájdalmas döntés, de az elhalogatott döntés miatt bekövetkező üzemzavar általában nagyobb visszhangot kelt.

Az adatbáziskezelő termék kiválasztása

A tervezési fázisban még nem nagyon beszéltem konkrét termékről, mert a tervezés első *(logikai)* részében még nem érdekes, hogy milyen relációs adatbáziskezelőn fog futni az adatbázisunk. Azonban a második, fizikai tervezési szakaszban meg kell határoznunk egy konkrét adatbáziskezelő terméket, és annak lehetőségeire építve megvalósítani az adatbázist. Ha azt a feladatot kapjuk, hogy írjuk minnél jobban hordozhatóra az adatbázist, akkor kevesebb lehetőségünk marad a teljesítményoptimalizálás figyelembe vételére, míg egy konkrét termékre specializálva általában sok eszköz a rendelkezésünkre áll. A következő fejezetekben vizsgáljuk meg, hogy egy jó, a megfelelő helyeken denormalizált adatbázis esetén milyen háttér áll a szolgálatunkra, ha SQL Server 2000-en akarjuk az adatbázist megvalósítani.

Általános irányelvek SQL Serverhez

Rövid sorok

Az SQL Server 2000 8 kByte-os lapokon tárolja a táblák sorait. Ha egy tábla sorai rövidek, akkor egy lapra sok fér, így az SQL Server sok sort ki tud nyerni egy lapból, azaz kevesebb IO művelettel jár az adatok kiolvasása. A hosszú sorok másik hátránya, hogy mivel nem nyúlhatnak át a lapokon, sok lap nem lesz teljesen kihasználva. Például két, egyenként 3 kByte-ot tartalmazó sor mellett kimarad 8-6=2 kByte-nyi hely, amelyet a sorok eléréséhez teljesen feleslegesen fel kell olvasni a lemezről. Egy normalizált adatbázis általában sok, keskeny, azaz kevés oszlopból álló táblából áll, így azokat az SQL Server hatékonyan tudja tárolni és kezelni.

Rövid kulcsok

Bár az SQL Server megengedi akár 900 byte-os kulcsok létrehozását is, ne éljünk a lehetőséggel. Az elsődleges kulcsok legyenek minél rövidebbek, a legjobb, ha számok, vagy néhány karakteres szövegek. JOIN-ok során a hosszú kulcsok összehasonlítása jelentős időbe kerülhet, ami tovább lassítja a lekérdezéseket.

Mivel az elsődleges kulcs létrehozása során mindig létrejön egy index is a kulcsként definiált oszlop*(ok)*on, és ez alapértelmezésben egy CLUSTERED index, azért különösen fontos, hogy rövid legyen a kulcsunk. A Clustered index kulcsa megjelenik ugyanazon tábla ÖSSZES NonClustered indexében is, ezért a nagyobb méret az összes index hatékonyságát lerombolhatja. *(Érdemes elgondolkodni azon is, hogy egyetlen Clustered indexünket mire „pazaroljuk", mert ez az indextípus tartománykeresésekre a legalkalmasabb. Mit szoktunk ilyen formán lekérdezni? JOIN-ban a gyermekrekordokat!)*

Válaszidő vagy végrehajtási idő?

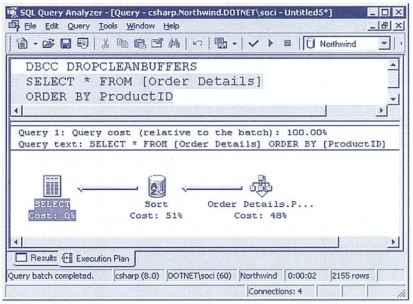
Ha ki kell listáznunk egy felhasználó részére tízezer sornyi információt *(amiből jó, ha egyszerre ötvenet lát)*, akkor mi a lényeg: az, hogy az összes sor minél gyorsabban lejöjjön, vagy az, hogy az első lapnyi *(50)* sort minél gyorsabban megkapja? Mivel ember ül a számítógép előtt, neki bőven elég, ha az Enter leütése után kap valami infót, amit nézegethet. Csakhogy az SQL Server pont ellentétesen gondolkodik! A kiszolgáló olyan stratégiával hajtja végre a lekérdezéseket, hogy azok minél gyorsabban kész legyenek!

Tegyük fel, hogy egy rendezett listát generáltatunk, és a rendezendő oszlopra pont van egy nonclustered indexünk. Sok sor esetén az SQL Server nem fogja az indexet használni, hanem inkább végigolvassa a teljes táblát, mert az gyorsabb. Így bár mondjuk 5 másodperc múlva ránkzúdítja a 10000 sort, de 5 másodpercig nem történik semmi, mert addig nem ad vissza semmit, amíg le nem rendezte a teljes eredményhalmazt. Ezzel szemben, ha például indexet használna, akkor lehet, hogy akár 20 másodpercig is eltartana a lekérdezés, de az index által megvezetett olvasás következtében már az első sor visszaolvasása után el tudja kezdeni visszaadni az eredményhalmazt, mert „magától" rendezve jönnek az adatok, nem kell utólag rendezni.

SQL Server esetén az OPTION *(FAST n)* opcióval - ahol az n a gyorsan elérni kívánt sorok száma – lehet elérni, hogy a kiszolgáló olyan hozzáférési stratégiát használjon, ami lehet, hogy nem a leggyorsabb végrehajtást eredményezi, de az első n sort gyorsan vissza fogja adni.

```
DBCC DROPCLEANBUFFERS
SELECT * FROM [Order Details]
ORDER BY ProductID
```

A lekérdezés végrehajtási terve:

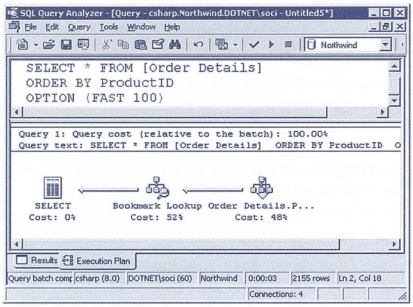


Azaz látható, hogy egyszerűbbnek találta a kiszolgáló végigolvasni a táblát, és azután lerendezni a sorokat, sémmint hogy használja a ProductID-n található nonclustered indexet. Ha megkérjük, hogy az első száz sort nagyon gyorsan adja:

```
DBCC DROPCLEANBUFFERS
SELECT * FROM [Order Details]
```

```
ORDER BY ProductID
OPTION (FAST 100)
```

akkor más végrehajtási tervet kapunk, amely már használja a nonclusted indexet:



A DBCC DROPCLEANBUFFERS célja, hogy a mérések előtt kidobálja a felolvasott lapokat a buffer cache-ből, így a táblát illetve az indexet mindig lemezről kelljen felolvasni. Így reálisabb az összehasonlítás, nem befolyásolja a cache pillanatnyi állapota.
Kedvenc eszközünk, a Query Analyzer ennél több információt is ad nekünk, ha bekapcsoljuk a Show Server Trace és a Show Client Statistics eszközöket, amelyek a Query menüpont mögött laknak.
A Server Trace lekérdezésünkre vonatkozó információit összefoglaltam a következő táblázatban:

Text	Duration	CPU	Reads	Writes
SELECT ...	209	60	28	0
SELECT ... OPTION (FAST 100)	231	60	224	0

Látszik, hogy a FAST 100 hatására a lekérdezés 209 msec helyett 231-re nőtt, ami az indexen keresztüli adatbejárás 224 logikai lapolvasása miatt van. Jelentősebb számú *(>10000)* sor esetén nagyobb lenne a különbség.
Sajnos pont a lényeget nem tudom demonstrálni, ami azt mutatná meg, hogy a második esetben az első néhány sort tartalmazó hálózati csomag sokkal hamarabb érkezik meg, mint az első esetben. Egy egyszerű ADO alkalmazással ezt is ki lehetne mérni, ez házi feladat a Kedves Olvasóknak. A jó ötleteket várom a fejlesztői levelezési listánkon, bögrés feladat :)

Tárolt eljárások

Az egyik „legolcsóbb" teljesítménynövelő eszköz, mégis sokan nem élnek vele. Miről is van szó? Egyszerűen arról, hogy az alkalmazásunk működéséhez szükséges lekérdezéseket csomagoljuk be egy tárolt eljárásba, és a működtetéshez szükséges paramétereket emeljük ki a tárolt eljárás paraméterlistájába.
Miért gyorsabb egy lekérdezés végrehajtása, ha az tárolt eljárásban van elhelyezve? A titok nyitja akkor tárul fel, ha megnézzük, hogyan hajt végre az SQL Server egy SQL parancsot.
A végrehajtás első lépésében a kiszolgáló nyelvi elemzést végez a kapott parancson, azaz megnézi, hogy nyelvtanilag megfelelő Transact SQL kódot adtunk-e neki. Sikeres elemzés után a parancsokat átalakítja egy belső bináris reprezentációvá, amit sequence tree-nek hívnak. Ezt követi egy normalizációs folyamat, amely kifejti a hivatkozott view-kat, belerakja a fába az implicit konverziók parancsaival együtt. Ha a kapott SQL parancs egy adatmódosító parancs *(UPDATE, DELETE, INSERT)*, akkor a sequence tree-ből egy query graph-ot képez a kiszolgáló, amelyet az optimizer, az SQL Server esze fel tud dolgozni. Az optimizer az eloszlási statisztikák, maga a lekérdezés és az indexek alapján legenerál egy általa leghatékonyabbnak ítélt végrehajtási tervet. Ez a hosszú folyamat előzi meg a parancs valódi végrehajtását. Mielőtt azonban a végrehajtás megkezdődne, egy igen fontos lépés zajlik le. A kiszolgáló eltárolja végrehajtási tervet a procedure cache-ben *(ami egyébként a buffer cache része)*, hiszen lehet, hogy legközelebb mégegyszer futtatni akarják ugyanezt a lekérdezést, és akkor már nem kell legenerálni a végrehajtási tervet. A tárolt eljárásokba be nem csomagolt lekérdezéseket általában ad-hoc *(magyarul talán csipcsup)* lekérdezéseknek hívjuk. Ezeket az SQL Server nem szívesen rakja be a cache-be, hisz viszonylag kicsi a valószínűsége, hogy az utolsó bitig ugyanazt a lekérdezést valaki újra végrehajtsa. Emiatt az ad-hoc lekérdezéseket a legtöbb végrehajtás során újra kell fordítani, azaz lassúak lesznek.
Teljesen más a helyet a tárolt eljárásokkal. Ők kifejezetten arra vannak kihegyezve, hogy a végrehajtási tervük a procedure cache-ben lakjon, így általában a tárolt eljárások végrehajtási tervét ritkán fordítja újra a kiszolgáló. Azaz kimarad sok időigényes lépés, egyszerűen csak végre kell hajtania a kiszolgálónak a korábban elkészített tervet. Ez lekérdezésektől függően több tíz vagy több száz százaléknyi teljesítménynövekedést jelenthet, és szinte nem kell érte tenni semmit! Érhetnek azonban bennünket meglepetések. Berakunk több SQL parancsot egy tárolt eljárásba, hogy majd jól felgyorsulnak, a fordítási lépés kihagyása miatt. Végrehajtjuk az eljárást, és kiderül, hogy az még akár lassabb is lett, mintha közvetlenül lefuttattuk volna egy batch-ben a részparancsokat. Mi történt?
Az SQL Server a tárolt eljárás végrehajtásának kezdetén készít egy tervet, amelyben benne foglaltatik az eljárás összes parancsa. Elkezdi a végrehajtást, és egyszer csak találkozik egy olyan paranccsal, ami alaposan belenyúl az adatbázisba, például létrehoz egy indexet valamelyik táblán. Ekkor az SQL Server azt mondja, hogy volt jó terv, nincs jó terv, hiszen az új index miatt lehet, hogy sokkal hatékonyabban is le lehetne futtatni a maradék parancsokat. Kidobja a meglevő tervét, és újrafordítja a tárolt eljárást, a-tól z-ig, majd folytatja a végrehajtást ott, ahol abbahagyta, csak már az új tervnek megfelelően.
Egy eljárás futtatása során akár többször is újrafordulhat az egész eljárás, amiből mi nem látunk semmit, csak azt, hogy lassú a végrehajtás. Az SQL Server Profiler segítségével megjeleníthetjük az újrafordítási eseményeket is, így egy gyanús rendszert érdemes a profilerrel működés közben megnézni. Ha azt látjuk, hogy egyes tárolt eljárások futtatásuk során többször is újrafordulnak, akkor meg kell néznünk mi az oka, és azt meg kell szüntetni. A pontos okokat és azok feloldását az **[1]** címen található tudásbázis cikk részletezi.
Mikor érdemes gyanakodnunk az újrafordításokra? A legalapvetőbb okok az átmeneti táblák használata, és a DDL és DML parancsok keverése, azaz például létrehozunk egy táblát *(DDL)*, majd módosítunk egyet *(DML)*. A legtöbb újrafordítási problémát a sorok átrendezésével könnyedén meg lehet oldani, de a hivatkozott cikkben a bonyolultabb eseteket is ismertetik.

Denormalizálás

A tervezésnél már említettem, hogy a DSS rendszerekben mesés eredményeket érhetünk el a denormalizálás alkamazásával, azon az áron, hogy az adatmódosítások lassabbak lesznek. Nézzük meg, hogy SQL Server 2000 esetén milyen eszközök könnyítik meg a denormalizált adatok kezelését.

Tegyük fel, hogy kereskedelmi rendszerünkben *(Northwind)* nagyon gyakori az típusú a lekérdezés, amelyben a vásárlások által megmozgatott pénzmennyiségekre kíváncsiak. A normalizált adatbázison így nézne ki a feladatot végrehajtó parancs:

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

Természetesen az aggregálás nagy adattömegek esetén nagyon időigényes tud lenni, valahogyon előre ki kellene számítani és eltárolni az aggregált eredményeket *(ezzel viszünk be redundanciát)*, és utána a lekérdezésben felhasználni a kész eredményeket.

Mit tehetünk a gyorsabb lekérdezés érdekében? Például írhatunk triggert, ami egy másik táblában nyilvántartja a részösszegeket az OrderID alapján. A trigger az alaptáblán történt minden egyes adatmódosítás esetén kiszámolja a módosított összeget, és azt módosítja a gyorsítótáblában. Egy ilyen trigger írása nem is olyan egyszerű!

Tegyük fel, hogy a tábla szerkezete a következő:

```
CREATE TABLE GyorsSzumma (
  OrderID INT NOT NULL PRIMARY KEY CLUSTERED,
  Szumma money NULL )
```

A módosításokat átvivő trigger működése szavakban:

- INSERT és UPDATE esetén: nézzük végig az összes módosított vagy beszúrt sort. Minden egyes érintett OrderID alapján számítsuk ki az összegeket, és azt tároljuk el a segédtáblánkban. Az egész eljárás problémás pontja az előbbi mondat „tárolni" igéje. Ugyanis ez attól függően, hogy az adott OrderID-jú szumma létezik-e a segédtáblában vagy UPDATE-et, vagy INSERT-et jelent.
- DELETE esetén: ha még létezik ugyanezzel az OrderID-val más tétel is az alaptáblában, akkor nincs más dolgunk, számoljuk ki az összeget az alaptáblában a törölt sorokban található OrderID alapján, és módosítsuk a segédtáblát. Ha pont az utolsót töröltük, akkor a segédtáblánkból törölni kell a megfelelő sort.

A trigger első részét kétféleképpen írhatjuk meg. Az első verzióban írunk egy kurzort a beszúrt sorokra, azaz az inserted virtuális táblára, és végigmenvén az összes módosított tételen mindegyiknél IF EXISTS(...)-tel lekérdezzük, hogy létezik-e már az összeg a másik táblában, és ha igen, akkor UPDATE-eljük azt az új értékre, ellenkező esetben INSERT-et hajtunk végre.

Ez kezdő gondolat volt. Miért kell egyidőben elintézni a kétféle kezelést igénylő összegeket? Összuk ketté a feladatot! Először írjuk meg azt az UPDATE-et, amely a már létező összegeket módosítja, majd egy külön utasításban szúrjuk be azokat az összegeket, amelyek még nem szerepeltek a segédtáblában. Mondhatná valaki, hogy miért nem írok külön triggert az UPDATE-re és az INSERT-re, és akkor nem kellene kínlódni a szétválogatással? Azért, mert nem az a lényeg, hogy beszúrás vagy módosítás történt az alaptáblán, hanem az, hogy az behozott-e új OrderID-t a képbe? Mert mind az INSERT, mind az UPDATE képes erre.

Nézzük hát a trigger első részét, ami azokat a részösszegeket számítja ki és módosítja a segédtáblában, amelyek már léteznek:

```
CREATE TRIGGER updinsSzummaSzamolo
ON [Order Details]
FOR INSERT, UPDATE
AS

UPDATE GyorsSzumma
SET
  Szumma =
  (
   SELECT
     SUM(Quantity * UnitPrice * (1-Discount))
   FROM
     [Order Details]
   WHERE
     OrderID = gy.OrderID
   GROUP BY OrderID
  )
FROM
  GyorsSzumma gy
INNER JOIN
  inserted
ON  --csak a módosított sorokra fusson
  gy.OrderID = inserted.OrderID
```

A zárójelben található lekérdezés számítja ki a megfelelő összegeket. Típusát illetően ő egy ún. Correlated Subquery, mert mint látható nem csak a levegőben számol, hanem a WHERE feltételében össze van kötve az UPDATE OrderID-jával, így mindig az éppen UPDATE-elendő sorhoz számolja ki a részösszeget. Az inserted táblával – amiben a frissen beszúrt vagy módosított sorokat adja át nekünk az SQL Server – azért kell összeJOINolnunk az UPDATE-et, hogy tényleg csak a módosított sorokra számoljuk ki az összegeket. Látható, hogy nem szűrjük ki azokat a sorokat, amelyekhez még nincs is bejegyzés a segédtáblában. Ezt megteszi helyettünk az UPDATE maga, mert ha olyan OrderID-ú sort akarunk módosítani, ami nincs is a céltáblában, egyszerűen nem csinál semmit.

Folytassuk a triggert, és nézzünk most azokat a sorokat, amelyekhez nincs még szumma a segédtáblában!

```
INSERT GyorsSzumma
  (OrderID, Szumma)
SELECT
  od.OrderID,
  SUM(od.Quantity * od.UnitPrice *
     (1-od.Discount))
FROM
  [Order Details] od
INNER JOIN
  inserted
ON
```

```
  od.OrderID = inserted.OrderID
LEFT OUTER JOIN
  GyorsSzumma gy
ON
  gy.OrderID = inserted.OrderID
WHERE
  gy.OrderId IS NULL
GROUP BY
  od.OrderID
```

Az INSERT bemeneti adatait egy SELECT-tel generáljuk. A SELECT-ben a LEFT JOIN-nal és a gy.OrderId IS NULL feltétellel szűrjük ki azokat a sorokat, amelyek még nem léteznek a céltáblában.
Nézzük a törlés kérdését! Ezt egy külön triggerbe helyezzük, ami csak a DELETE-re lesz kiélezve. Azokat az összegeket kell törölni a segédtáblából, amelyeknek az ID-je nem található meg az eredeti táblában, de benne van a deleted táblában, azaz most törölték az utolsó tételt egy bizonyos OrderID-vel. Ne feledjük, hogy a DELETE trigger futásakor a forrástáblában már nincsenek benne a törölt sorok, mert a normál triggerek az őket kiváltó műveletek után futnak le *(ezért is hívják AFTER triggernek)*.

```
CREATE TRIGGER delSzummaSzamolo
ON [Order Details]
FOR DELETE
AS

DELETE
  GyorsSzumma
FROM
  GyorsSzumma gy
INNER JOIN
  deleted
ON  --csak a törölt sorokra fusson
  gy.OrderID = deleted.OrderID
LEFT OUTER JOIN
  [Order Details] od
ON
  deleted.OrderID = od.OrderID
WHERE
  --nem maradt ilyen tétel az
  --alaptáblában
  od.OrderId IS NULL
```

Ezzel még csak a segédtáblából törlendő tételeket határoztuk meg és töröltük ki. Most még azokkal az OrderID-jű összegekkel kell foglalkoznunk, amelyekhez még maradt a forrástáblában azonos OrderID-jú tétel. Ezekhez újra ki kell számítani az összeget, és azt módosítani a segédtáblában. A script teljesen ugyanaz, mint amit az előző UPDATE-INSERT triggerben láttunk, csak most az inserted helyett a deleted táblában találjuk meg a számunkra érdekes sorokat:

```
UPDATE GyorsSzumma
SET
  Szumma =
  (
... mint a korábbi UPDATE-INSERT triggerben
  )
FROM
  GyorsSzumma gy
INNER JOIN
  deleted
ON  --csak a törölt sorokra fusson
  gy.OrderID = deleted.OrderID
```

És kész. A triggerünk csak a létrehozás után képes a segédtábla adatokat karbantartani, ezért a trigger létrehozása előtt vagy legyen üres a forrástábla, vagy kézzel töltsük fel a segédtáblát. Az utóbbira egy megoldás:

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

Lássuk munkánk gyümölcsét! Hogyan lehet nagyon gyorsan lekérdezni az összegeket?

```
SELECT OrderID, Szumma FROM GyorsSzumma
```

Ha az alaptáblából más adatokra is szükség van *(ez a gyakori)*, akkor egyszerűen össze kell JOIN-olnunk az eredeti és a segédtáblánkat az OrderID mentén.
Sajnos a szokásos „amit nyerünk a réven, elvesztjük a vámon" tétel most is működik, hisz az alaptábla módosításait jelentősen lelassíthatjuk az aggregálással. El kell döntenünk mikor vállaljuk fel ezt az időt. Egyszer úgyis rá kell szánnunk, a kérdés csak az, hogy mikor nem *(annyira)* zavaró.

Zárszó

Az előző két oldalon egy igen primitív problémára, redundáns szummák karbantartására írtunk elég bonyolult triggert. Tényleg nekünk kell kielemezni a lehetséges végrehajtási ágakat, és tucatnyi hasonló kaptafájú triggert írni? Nos, SQL 2000-től már nem. Az előbbihez hasonló problémák megoldása eleve bele van építve a termékbe. A neve: meglepi. A jövő hónapban kiderül :) Visszavárom Önöket!

Soczó Zsolt
Zsolt.Soczo@netacademia.net

A cikkben szereplő URL-ek:

[1]: INF: Troubleshooting Stored Procedure Recompilation Q243586
[2]: A cikkben szereplő script-ek http://technet.netacademia.net/download/sql
[3]: MS SQL Server 7.0 Performance Tuning Guide Chapter 20 - Transact-SQL Optimization and Tuning Inside Microsoft SQL Server 7.0 14. fejezet: Optimizing Query Performance

XMLgessünk *(V. rész)*, – szappanopera

Bevezetés

Sok időbe telik, mire a fejlesztők megegyeznek valamiben. Minden csoport hű az általa használt technológiákhoz, és nagyon nehezen vehető rá, hogy áttekintse a konkurens gyártók, technológia-diktátorok módszereit. S mi issza meg ennek a levét: a rendszerek együttműködési képessége. Létrejönnek alkalmazásszigetek a nagyvilágban, amelyek „házon belül" nagyon nyitottak, de a más típusú, önmagukban szintén nyitott rendszerekkel képtelenek kommunikálni. Lesz itt valaha megegyezés? Talán, igen. A kulcsszó: SOAP, Simple Object Access Protokol.

Az elosztott alkalmazások kora

Néhány éve még az volt a menő programozó, aki MFC-ben vagy Delphi-ben díszes, dokkolható és lebegtethető toolbar-okat és egyéb csicsás, lyukas és matyómintás ablakokat tudott létrehozni. Azonban a világ változik. Amióta a számítógépeket összekötötték hálózati kábelekkel, azóta megvan a fejlesztőkben és a megrendelőkben az igény, hogy olyan alkalmazásokat írjanak, amelyek több löerővel működnek, azaz több számítógép együttműködésével végzik el a feladatukat. Így a rendszerek nemcsak teljesítmény, de sokszor megbízhatósági előnyökhöz is jutnak, és mindezt sokszor olcsóbban, mint egy nagy böhöm vason megvalósított monolitikus társaik.

Az elosztott alkalmazások igénye olyan technológiák kifejlesztését követelte meg a nagy gyártóktól, amelyek elfedik a hálózati réteg bitfolyam jellegét, és a programozó számára teljesen átlátszó módon képesek más gépen elhelyezkedő programrészek, függvények meghívását biztosítani. Sőt, mivel a 90-es évek elején a hagyományos, moduláris programozási paradigmát elhomályosította az Objektum Orientált szoftverfejlesztés tana, a feljesztők olyan háttérinfratruktúrára vágytak, amely objektumok vagy komponensek távoli létrehozását és kezelését biztosítja, számukra a lehető legegyszerűbb és legtranszparensebb módon.

Az igény megvolt, és a világ elindult a cél felé. Ahogy azt már megszokhattuk, a Microsoft elindult az egyik csapáson, és a UNIX-os világ is vágta a maga ösvényét, látszólag más irányba, a valóságban egymással párhuzamosan. A 80-as években két jelentősen elterjedt módszer volt a távoli eljáráshívásokra. Az egyik a Sun RPC *(Remote Procedure Call, Távoli Eljáráshívás)*, amelynek egyik legismertebb alkalmazása a UNIX rendszerekben elterjedt Network File System vagy ismertebb nevén NFS. A másik oldalon a Microsoft a DCE RPC-t használja még ma is a Windows-os, elsősorban NT alapú rendszerekben.

Mindkét távoli eljáráshívási technológia működő, életképes, az évek során bizonyított, azonban nem objektumorientált. Mivel az OOP elvek keresztülverekedték magukat az egyetemek vastag falain kívülre és a programozók fején belülre is, jogos volt az igény, hogy kellene valamilyen felsőbb réteg a RPC protokollok tetejére, amelyeken keresztül már objektumokat is rángathatunk a drót végén, nem csak globális függvényeket hívogathatunk. Megszülettek az ORPC protokollok, amelyek megpróbálták összebékíteni az objektumorientáltságot a hálózati protokollokkal. Ezt egyfajta cookie alkalmazásával oldották meg, azaz minden egyes kérésben elküldtek egy objektumazonosítót is a kiszolgáló folyamatnak, amely a saját adminisztrációs információiból tudta, hogy az ügyfélprogram melyik objektumot akarja megszólítani.

Egy tipikus ORPC kérés és válasz így néz ki:

Kérés

Objektum Végpont Azonosító (Melyik objektumot szólítjuk meg?)
Interfészazonosító (Melyik interfészen van a keresett objektum?)
Metódusazonosító (Melyik metódust kell meghívni?)
Kiegészítő fejlécek (Mit felejtettünk ki a protokollból?)
Paraméterblokk (Bemeneti és bemeneti-kimeneti paraméterek)

Válasz

Státuszkód (Sikeres volt a hívás?)
Kiegészítő fejlécek (Mit felejtettünk ki a protokollból?)
Paraméterblokk (Bemeneti és bemeneti-kimeneti paraméterek)

Az Objektum Végpont Azonosító az előbb emlegetett cookie, amin keresztül a kiszolgáló tudja, melyik objektummal szeretnénk foglalkozni. Az Interfészazonosító és a Metódusazonosító írja le a meghívni kívánt függvényt. A Kiegészítő fejlécek szolgálnak olyan bővítések későbbi becsempészésére, amelyek még nem ismertek a protokoll kifejlesztésekor. A Paraméterblokkok célja nyilvánvalóan a metódusok paramétereinek szállítása.

Manapság két jelentős ORPC implementáció virágzik: a Microsoft részéről a DCOM *(Distributed Component Object Model)*, a UNIX-os világban pedig az Internet Inter-ORB Protokol, röviden IIOP, mely a CORBA protokollja. Mindkettő nagyjából a fenti struktúrákat használja a végpont azonosítására, de természetesen a konkrét mezők neve és száma különbözik.

Mindkét protokollban ki kellett dolgozni a paraméterek átalakítását byte-folyammá, majd vissza az eredeti jelentésükké *(Stringgé, Integerré, satöbbi)*. Ezt az oda-vissza alakítást hívják Serialization-nek illetve Deserialization-nek. Érdemes a szavakat megjegyezni, mert a .NET kapcsán sokszor fogunk még róluk hallani. A DCOM az ún. Network Data Representation *(NDR)* formátumot használja, a IIOP Common Data Representation-t. A két formátum hasonló, de azért annyira különböznek, hogy a kétféle rendszer nem tud szót érteni egymással.

Miért nincs egységes protokoll?

Miért van az, hogy a világ egyik pólusa DCOM-on, a másik CORBA-n keresztül kommunikál? Mindkét protokoll hozzá van láncolva az őt kidolgozó céghez. Létezik ugyan DCOM implementáció UNIX-okra, de alig -ha egyáltalán- használják őket. Sajnos az a helyzet, hogy a DCOM annyira hozzá van gyógyítva a Windows NT háttérinfrastruktúrájához, hogy nehéz bármilyen más platformon teljes hatékonyságában kihasználni. A CORBA már sokkal több rendszeren implementált, ám a különböző megvalósítások között is vannak olyan apró részletkülönbségek, amely miatt csak alapszinten működik jól az együttműködés, és - hasonlóan a DCOM-hoz - pont a finomságoknál *(tranzakciók, biztonság)* lesznek gondok a heterogén rendszerek együttműködésében.
Mindkét technológia jól működik zárt, jól adminisztrálható belső rendszerben, leginkább cégen belüli kiszolgálók közötti kommunikációra. Azonban mindketten megbuknak, ha bejönnek a képbe a tűzfalak és az Internet. Márpedig ha tetszik, ha nem, bejöttek. A mai rendszereknél egyre fokozódó igény van az összetevők Interneten keresztüli elérhetőségére. Annak az esélye, hogy a DCOM-hoz szükséges számtalan RPC portot kinyissa egy „tűzfalgazda", közelít a nullához.
Melyik az a port, amely még a legtöbb cégnél nyitva van? Igen, a 80-as, amely a webböngészéshez szükséges. Akkor miért nem lehet egy olyan réteget fejleszteni a DCOM illetve a CORBA fölé, amely a VPN-hez hasonló módon egy csatornára tereli a teljes protokoll adatforgalmát? Például a HTTP protokollra, amit olyan szívesen átengednek a tűzfalak.
Ezek a technológiák léteznek, például Windows 2000-ben a DCOM-hoz COM Internet Services Proxy néven létezik egy kiegészítés, ami rá tudja ültetni az DCOM-ot HTTP protokollra. Azonban ettől még nem fog kommunikálni a világ két ORPC rendszere, nem beszélve a konfigurációs bonyodalmakról...

A HTTP, mint egy jobb RPC

Miért pont HTTP? A HTTP-t hasonlóan az RPC-hez nagyon sokan használják, egyszerű, és ellentétben az RPC-vel a legtöbb tűzfal szívesen átengedi magán. A HTTP kéréseket általában webkiszolgálók fogadják, és legtöbbször rajtuk keresztül induló kiegészítő alkalmazások dolgozzák fel *(pl. CGI alkalmazások)*.
A DCOM-hoz és az IIOP-hez hasonlóan a HTTP is kérés-válasz típusú protokoll. Az ügyfélprogram egy megadott TCP porton hozzákapcsolódik a HTTP szerverhez, amely általában a 80-as porton figyel. Miután a TCP csatorna felépült, az ügyfélprogram elküldi a kérését, azt a kiszolgáló értelmezi, feldolgozza, és visszaküld egy választ. A teljes kommunikáció nyelvezetét a HTTP protokoll írja le.
A következő kódrészlet egy egyszerű HTTP kérés:

```
POST /szappan HTTP/1.1
Host: 172.16.0.200
Content-Type: text/plain
Content-Length: 11

NetAcademia
```

Látható, hogy a kérés szöveges formátumú, ami hibakeresésnél igen nagy segítség, hisz például Network Monitorral könnyedén visszafejthető a HTTP kommunikáció *(ugye, ugye? – a szerk)*.
Az első sor három elemet is tartalmaz: a HTTP metódust *(POST)*, a kért URI-t *(/szappan)*, és a protokoll verzióját *(HTTP/1.1)*. A metódus a HTTP *(és WebDAV)* szabványban rögzített értékeket vehet fel, a példában szereplő POST-ot általában akkor használjuk, ha a webkiszolgáló felé valamilyen információt szeretnénk eljuttatni. Ez a hagyományos webfejlesztésben például egy kitöltött form adatait jelentheti.
Az URI azonosítja a célobjektumot. Közönséges webszervereknél ez a letölteni kívánt állomány elérési útja, DCOM vagy IIOP hasonlattal élve az elérni kívánt objektum azonosítója.
A harmadik és a negyedik sor a kiszolgálónak küldött tartalom típusát és hosszát azonosítja. A típus segítségével tudja az ügyfélalkalmazás a kiszolgáló tudtára adni az általa használt tartalomleíró nyelvezetet. DCOM-nál ez az NDR. HTTP-nél általában text/plain, ami közönséges ASCII szöveget jelent, vagy text/html, ami a HTML kódolt tartalmat jelenti.
Mivel a HTTP fejléc hossza kérésenként más és más lehet, az utolsó fejléc után két kocsivissza-soremelés karakter is van, innen tudja a feldolgozó alkalmazás, hogy hol végződik a fejléc-blokk, és hol kezdődnek a valódi adatok, melynek hosszát és kódolását a Content-Length és Content-Type fejlécek azonosítják. Példánkban a tartalom *(payload)* hossza 11 byte, és tartalma NetAcademia.
Miután a webkiszolgáló feldolgozta a kérést, visszaküld egy választ az ügyfélalkalmazásnak. A válasznak kötelezően tartalmazni kell egy sikerességet jelző kódot, és ha akar, akkor a kéréshez hasonlóan további tartalmat is szállíthat.

```
200 OK
Content-Type: text/plain
Content-Length: 12

aimedacAteN
```

A 200-as státuszkód a szabványos sikerességet jelző kód a HTTP protokollban. Az érdeklődők a teljes HTTP 1.1 szabványt a 2616-os RFC-ben találhatják meg a **[1]** címen.

Az XML, mint egy jobb NDR

Láttuk, hogy a HTTP protokoll nagyban kiválthatja az RPC-t, azonban van egy funkció, ami hiányzik belőle: a távoli metódushívások paramétereinek szabványos leírása. És itt jön a képbe az XML.
Hasonlóan az NDR-hez és a CDR-hez, az XML is egy adatleíró protokoll, amely ráadásul szabványos és platformfüggetlen is. Segítségével könnyedén átalakíthatjuk a paramétereket ASCII adatfolyammá *(Serialization)*, amelyet könnyedén lehet hálózaton átvinni, és a céloldalon dekódolni *(Deserialization)*. Igen előnyös tulajdonsága, hogy szinte az összes platformon bőséges a támogatottsága, szöveges

alapú, így egyszerű eszközökkel is könnyű feldolgozni, és nagyon könnyű egy már meglévő XML alapú formátumot egyértelmű módon kibővíteni. *(Remélem cikksorozatunk rendszeres olvasóinak már kigyúlt az agyában a NameSpace fogalom lámpása, mint a bővíthetőség zálogа.)*

A paraméterek struktúrájának leírására is már van kész eszközünk *(2001. április óta)*, amely nem más, mint az XML Schema. Az alábbi részlet egy olyan XML dokumentumot ír le, amelyben az elemek az urn:schemas-netacademia-net:SztringKolbaszolas névtérben laknak, a gyökérelem neve HatraArc, aminek kötelezően van egy gyermekeleme mit néven, annak tartalma string típusú, és mögötte lehet még akárhány darab és típusú további elem.

```
<schema
  xmlns='http://www.w3.org/2001/XMLSchema'
  targetNamespace='urn:schemas-netacademia-
↳ net:SztringKolbaszolas'>
  <complexType name='HatraArc'>
    <sequence>
      <element name='mit' type='string' />
      <any minOccurs='0'
           maxOccurs='unbounded'
           processContents='skip' />
    </sequence >
  </complexType>
</schema>
```

Azaz látható, hogy a mai állás szerint van egy adatreprezentációs szabványunk *(XML* **[2]***)*, és típus *(struktúra)* leíró szabványunk *(XSD* **[3]***)*. Mi akadálya ezekkel kiváltani az NDR-t és társait? Semmi! Helló SOAP!

HTTP + XML = SOAP

A SOAP szabvány nem tesz mást, mint összefogja az XML-t, mint a paraméterek kódolási szabványát és a HTTP-t, mint adatátviteli mechanizmust. Egy SOAP metódushívás nem más, mint egy egyszerű HTTP kérés és arra adott válasz, amelyben a kérés és válasz fejlécei, és azok tartalma megfelel a SOAP szabványban előírtaknak. Ennyi az egész. A SOAP szabvány nem szól semmit arról, hogy a webszerveren mit történjék egy SOAP formátumú HTTP kérés hatására. Nem írja elő, hogy ettől egy COM objektumra képződjön le a kérés, egy sima DLL-nek hívjanak meg egy függvényét, egy PERL modul álljon a kiszolgálás legvégén, vagy éppen egy WebSzerviz legyen a kiszolgáló. Bizony-bizony, a WebSzervizek hátterét kőkeményen a SOAP adja. Mi más? A SOAP kérés tulajdonképpen egy HTTP POST kérés. A Content-type kötelezően text/xml. A Request-URI természetesen kötelező, hisz ez azonosítja, hogy milyen objektumot vagy szolgáltatást akarunk meghívni a kiszolgálón. A HTTP kérésben kötelezően benne kell lenni egy fejlécnek, ami a meghívandó metódus nevét tartalmazza. E fejléc neve SOAPMethodName, és a tartalma a meghívni kívánt metódus neve egy URI-val megelőlegezve:

```
SOAPMethodName: urn:schemas-netacademia-
↳ net:SztringKolbaszolas#HatraArc
```

Azaz szeretnénk meghívni a HatraArc metódust a urn:schemas-netacademia-net:SztringKolbaszolas névtérben. A névtér szerepe pont ugyanaz, mint DCOM-ban az interfészazonosító. Felfoghatjuk úgy is, hogy a kettő együtt azonosít egyértelműen egy metódust.

A SOAPMethodName fejléccel tudatjuk a kiszolgálóval, hogy melyik metódust kívánjuk meghívni. A paraméterek a HTTP kérés törzsében utaznak, és a kódolásukra speciális SOAP szabályok vonatkoznak. Mik ezek? A paramétereket egy Envelope elembe, és azon belül egy Body elembe kell foglalni. A Body-n belül kell lenni a metódus nevét tartalmazó gyermekelemnek, melynek a SOAPMethodName fejlécben a metódus nevét megelőző névtérben kell lenni.

```
POST /szappan/Amo HTTP/1.1
Host: 172.16.0.200
Content-Type: text/xml
Content-Length: 163
SOAPMethodName: urn:schemas-netacademia-net:
↳ SztringKolbaszolas#HatraArc

<Envelope>
  <Body>
    <na:HatraArc
     xmlns:na='urn:schemas-netacademia-net:
↳ SztringKolbaszolas'>
      <mit>NetAcademia</mit>
    </na:HatraArc>
  </Body>
</Envelope>
```

A SOAPMethodName-ben található metódusnévnek egzaktul egyezni kell a Body gyermekelemével *(a névteret is beleértve)*, ellenkező esetben a fogadóalkalmazásnak meg kell tagadni a kérés kiszolgálását. Ez egy szeniális húzás, melynek segítségével a tűzfaladminisztrátorok anélkül tudják szabályozni, hogy mely metódusokat hívhatnak meg a tűzfalon keresztül, hogy a tűzfalnak bele kellene nézni a kérés tartalmába! Így nem kell kiegészíteni a tűzfalak alkalmazásszintű protokollszűrőit, mert a HTTP fejlécek szűrése minden komolyabb tűzfalba be van építve, és paraméterezhető. Emellett egy fejléc sokkal rövidebb lehet, mint a tényleges tartalom *(gondoljuk például egy többezer elemű tömbparaméterre)*, így a szűrés során nem kell nagy XML adattömegeket elemezgetni, ami igencsak lefoglalná a tűzfalak processzorát.

A SOAP válasz nagyon hasonló szerkezetű a kéréshez. A kimeneti vagy kétirányú paraméterek most is az Envelope/Body elemek vendégei, és az őket befogladó elem neve a hívott metódus nevéből képződik egy 'Response' utótaggal kiegészítve.

```
200 OK
Content-Type: text/xml
Content-Length: 181

<Envelope>
  <Body>
    <na:HatraArcResponse xmlns:na='urn:
↳ schemas-netacademia-net:SztringKolbaszolas'>
      <result>aimedacAteN</result>
    </na:HatraArcResponse>
```

```
  </Body>
</Envelope>
```

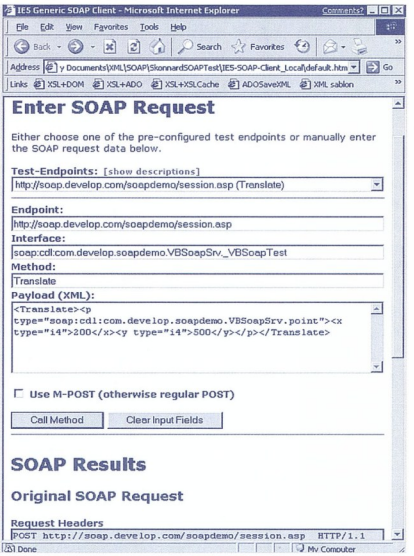
Láthatjuk a szabványos HTTP státuszkódot *(sikeres kiszolgálás - 200)*, és a borítékba zárt választ, amely ugyanabban a névtérben van, mint a kérés. A SOAPMethodName fejlécre most nincs szükség, ezért nincs is jelen.

Ami elsőre furcsa, hogy a szabvány egy szót sem szól arról, hogy mi történjen a szerveren a kérés hatására. Az IIS például képes közvetlenül SOAP kérésekre válaszolni? Természetesen nem, mint ahogy más webszerverek sem. De mindegyiket könnyű kiegészíteni olyan módon, hogy alkalmas legyen erre. Windows 200x platformon a hamarosan végleges állapotúvá érő .NET Framework az, ami Microsoftos környezetében a legkényelmesebb hátteret biztosítja SOAP kiszolgálók írására. Az, hogy történetesen a Microsoft fölé húzott egy absztrakciós réteget, amit WebSzerviznek hívnak, senkit ne zavarjon meg. Amikor WebSzervizeket írunk .asmx kiterjesztésű fájlokban, akkor tulajdonképpen egy SOAP kiszolgálót implementálunk. A .asmx lapnak SOAP formátumú kéréseket küldve ő nagyszerűen fog válaszolni, és a lapban implementált osztályok metódusait játszi könnyedséggel elérhetjük. Szerencsére a Microsoft egy lépéssel továbbment, és nekünk még csak nem is kell törődni a „csúnya" SOAP fejlécek és tartalom írásával, illetve a válasz elemzésével. Egy wsdl.exe nevű kis eszköz legyárt nekünk egy *(C#, VB, JScript)* forrásnyelvű .NET assembly-t, ami egy olyan osztály leírását tartalmazza, amely pontosan azon metódusokat tartalmazza, amiket a SOAP szerverünk *(WebSzerviz)* számunkra felkínál. A generált osztály mögött egy SoapHttpClientProtocol nevű osztály áll, ami magábazárja a SOAP protokoll által előírt összes részletet. Ami külön jó, hogy mi még ezzel sem kell, hogy érintkezzünk, mi csak kapunk egy osztályt, amelynek metódusait meghívva egy távoli osztály aktiválódik, annak meghívódik az azonos nevű metódusa, a paraméterek jönnek-mennek *(és nem csak egyszerű integerek és társaik, hanem akár komplett osztályok is!)*, és az egészből semmit nem látunk ügyféloldalon. Sőt, még kiszolgálóoldalon sem! De ez már egy másik történet, amelyről még sokat fognak hallani Kedves Olvasóink. Kösz SOAP, kösz .NET framework!

Hogyan lássunk neki?

Remélem sikerült felcsigáznom a SOAP iránti érdeklődésüket, és már alig várják, hogy kipróbálhassák valami kis példaprogramon a technológiát. Milyen lehetőségeink vannak a kezdésre ügyfél és kiszolgálóoldalon?

Kezdjük az ügyféloldallal. A Microsoft Interactive Developer 2000. januári számában Aaron Skonnard publikált egy cikket SOAP: The Simple Object Access Protocol címmel. A cikk megtalálható az MSDN library-ban is. Áron írt egy nagyszerű SOAP teszt ügyfélalkalmazást Internet Explorer 5-re. A cikkhez mellékelt önkicsomagoló állományban a default.htm-et megnyitva indul el a SOAP teszter alkalmazás. A megadott SOAP szerver-végpont és metódusnév megadása után a Call Method gomb megnyomására induló függvény összeállítja a SOAP kérést, és az XMLHTTP objektum segítségével elküldi a kérést a SOAP kiszolgálónak. A kérést is és a kapott választ is megtekinthetjük a lap ablakaiban. Az alkamazás letölthető a **[4]** címről is.



Zárszó

Nem beszéltem a másik oldalról, a SOAP kiszolgálókról. A témakör megér még *(minimum)* egy misét, így azzal a következő részben fogok foglalkozni. Megnézzük a SOAP Toolkit szolgáltatásait is, és írunk egy VB alapú és egy PERL alapú SOAP kiszolgálót IIS5 alá. Akinek van kedve, az akár egy Linux-os gépen is megírhatja a kiszolgálókomponenseket Apache, vagy egyéb http daemon alá is, köszönhetően a PERL hordozhatóságának. Hol van már a „a Windowsos programok csak Windwosos progamokkal működnek együtt" világ! A múltban, igen nagy örömünkre.

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

A cikkben szereplő URL-ek:

[1]: RFC2616
http://www.w3.org/Protocols/rfc2616/rfc2616.html
[2]: XML szabvány
http://www.w3.org/TR/2000/REC-xml-20001006
[3]: XML Schema
http://www.w3.org/TR/xmlschema-0
[4]: Letölthető kódok
http://technet.netacademia.net/download/xml
[5]: SOAP szabvány
http://www.w3.org/TR/SOAP

MS Exchange – Miért indul lassan az Outlook?

MS Exchange - Miért indul lassan az Outlook?

K: Az egyik ügyfelünknél újratelepítettük a szerverüket *(Windows 2000-ről SBS 2000-re)*. A munkaállomásokat átállítottuk az új kiszolgálóra, telepítettük az Outlook 2000-et a megosztott clientapps könyvtárból. A kliensek némelyikén iszonyatosan lassan indul el az Outlook, némelyik azt is mondja, hogy a kiszolgáló nem elérhető, de "újra próba" után megtalálja. Emlékeim szerint ennek megoldására jó a NetMon: valami névfeloldással kapcsolatos fejtegetés rémlik, de több nem jutott eszembe...

V: Az Outlook indulásakor *(és minden további kommunikáció során RPC (remote procedure call))* kapcsolatot kezdeményez, és használ az Exchange kiszolgálóval, mely a Registry RPC kötési sorrendjén alapul *(RPC binding order)*. Az Outlook úgy keresi meg a kiszolgálót, hogy ellenőrzi a rendelkezésre álló hálózati útvonalakat és névfeloldást végez. A bejegyzés meghatározza, hogy az Outlook milyen hálózati protokollokat, és milyen sorrendben használ a kereséshez.
Az alapértelmezett beállítás:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Exchange]
Az Exchange Provider értéke:
"ncalrpc,ncacn_ip_tcp,ncacn_spx,ncacn_np,netbios,
↳ ncacn_vns_spp"
```

Azaz elsőként helyi feloldást végez *(lrpc=local rpc)*, amely ha nem sikerül *(és miért is sikerülne? Az Outlookok elenyésző hányada fut ugyanazon a gépen, mint az Exchange kiszolgáló!)*, próbálja a következőt, s ez egészen addig megy, amíg meg nem találja a megfelelő módszert - vagy hibaüzenetet ad vissza, ha nem találta meg. Ha kellően hátul van a sorban a megfelelő protokoll, ez a folyamat igen sokáig, akár fél percig is eltarthat.
A megoldás így – mivel az Outlook NetBIOS néven keresi az Exchange kiszolgálót - a protokollsorrend megváltoztatása. A named pipes minden protokoll esetén sikeres gyors csatlakozást biztosít, így ezt célszerű első helyre tenni:

```
"ncacn_np,ncalrpc,ncacn_ip_tcp,ncacn_spx,netbios,
↳ ncacn_vns_spp"
```

A hosts fájl módosítása javíthat a helyzeten, hiszen a névfeloldást gyorsítja, de a korrekt megoldás a fenti Registry bejegyzés megváltoztatása.
Forrás : NetAcademia Exchange 2000 lista

Windows 2000 – az automatikus programindítás letiltása

K: Hol lehet a Windows 2000 Server-ben letiltani a bekapcsoláskor elinduló programokat? Feltettem a Real Player 8.0-t és le szeretném tiltani, hogy minden bejelentkezéskor elinduljon, és az óra mellé rakja ikonját.

V: A Windowsban *(sajnos, vagy hála Istennek)* több automatikus programindítási lehetőség található. Ha meg szeretnénk szüntetni az automatikus indítást, akkor az összes helyet, a rendszer indítópultját *(Startup folder(ek))*, és a Registry meghatározott bejegyzéseit végig kell néznünk a törléshez.
A registrybejegyzések a következők:

```
RUN és RUNONCE kulcsok
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
↳ CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\
↳ CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
↳ CurrentVersion\RunOnce
HKEY_CURRENT_USER\Software\Microsoft\Windows\
↳ CurrentVersion\RunOnce
```

A RUN és a RUNONCE kulcsok a HKLM és a HKCU listában is szerepelnek, azaz mind globális, mind felhasználónkénti bejegyzések rendelkezésre állnak. A HKLM-ben lévő bejegyzések a felhasználó desktopjának létrehozása előtt, a HKCU-ban lévők a desktop létrehozása után indulnak. A RUN bejegyzés alatt a programok minden felhasználói belépéskor elindulnak. Az adat értéke a futtatandó parancs. *(Csökkentett üzemmódban ezek nem indulnak el.)* A RUNONCE bejegyzés egyetlenegyszer végrehajtandó parancsokat tartalmaz, ezek a parancs végrehajtása után törlődnek. Vannak olyan bejegyzések is, melyek bejelentkezés előtt is lehetővé teszik alkalmazások futtatását, ezek a:
RUNSERVICES és RUNSERVICESONCE kulcsok.

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
↳ CurrentVersion\RunServices
HKEY_CURRENT_USER\Software\Microsoft\Windows\
↳ CurrentVersion\RunServicesOnce
```

A RUNSERVICES és a RUNSERVICEONCE kulcsok alatti programok minden rendszerindításkor elindulnak. Az adat értéke egy parancssor. A registrybejegyzések és az indítópultban szereplő programok végrehajtási sorrendje a következő:

```
HKLM\Software\Microsoft\Windows\CurrentVersion\
↳ RunServicesOnce
HKLM\Software\Microsoft\Windows\CurrentVersion\
↳ RunServices
[Bejelentkezés]
HKLM\Software\Microsoft\Windows\CurrentVersion\
↳ RunOnce
HKLM\Software\Microsoft\Windows\CurrentVersion\
↳ Run
HKCU\Software\Microsoft\Windows\CurrentVersion\
↳ Run
Indítópult
```

```
HKCU\Software\Microsoft\Windows\CurrentVersion\
↳ RunOnce
```

Forrás : NetAcademia Windows 2000 Lista

Windows 2000 - IP cím gyors megváltoztatása

K: Hogyan lehet maximum két kattintással IP címet cserélni Windows 2000-ben?

V: A feladat megoldható, de jelen esetben nem a Windows mélyén rejtőző titkos menüpontot kell keresni: a megoldást a Windows beállítási ablakain kívül találjuk.

Kevesek által ismert és használt eszköz a parancssorból indítható netsh.exe. Ez a program nélkülöz minden dekorációt *(ablakot, checkbox-ot stb.)*, egyszerű és hatékony, a karakteres terminálablakokhoz szokott szíveket melengeti. Emellett igen nagy előnye a parancssorból való indítás lehetősége és a parancssori paraméterezés. *(a jelen feladat megoldásához ezt a tulajdonságát lehet kihasználni)*.

Néhány szó a netsh használatáról:

A netsh a következő beállítások elvégzésében segít:

- interfészek
- routolási prorokollok
- szűrők
- routolás
- RRAS
- Beállítások kíiratása tetszőleges routeren
- Parancsvégrehajtás adott routeren

Az egyes célfeladatokat a feladathoz csoportosított környezetben tudjuk elvégezni. *(pl. interface, routing ... környezet)*. Ha interaktív módban használjuk, parancssorból a netsh parancs kiadásával indítható egyszerű promptot kapunk: *(netsh>)*,a parancsokat ez után kell kiadni és ENTER-rel lezárni.

A legfontosabb parancsok:

- ? - segítség kérése adott környezetben
- exit - kilépés a programból
- interface - belépés interface környezetbe
- .. - feljebb lépés alkörnyezetből
- dump - környezet beállításainak listázása
- set - beállítás

A fenti lista korántsem teljes, de elegendő a feladat megoldásához. A közelebbi ismerkedéshez a ? egyértelmű segítséget ad. Az előző parancsok a netsh paramétereiként megadva egy sorban lefuttathatók. Így az eredeti feladathoz a megoldás első lépése a végrehajtó parancssor elkészítése, mely a következőképpen néz ki:

```
netsh interface ip set address 10.1.1.111
```

Az IP cím egy előre kiválasztott cím, de kis további bűvészkedéssel ezt megadhatóvá tehetjük a parancsunk számára. Ezt a sort kell már csak kattintással elindíthatóvá tenni, amit legegyszerűbben egy parancsikonnal tehetünk meg.

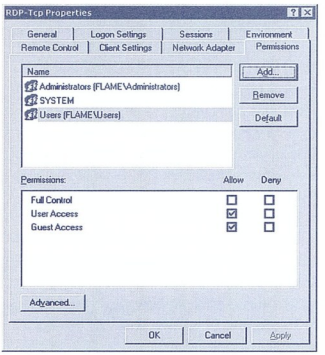
Forrás : Netacademia Windows 2000 lista

Windows 2000 - TerminalServer jogosultságok

K: Adott egy NT4-es domain és egy Win2k-s TerminalServer. Hogyan tudom megtenni azt, hogy csak bizonyos domain usereknek engedélyezem a TS használatát?

V: A kliensek RDP-TCP kapcsolaton keresztül jelentkeznek be a Terminal Serverbe. Egy szerveren egy hálózati kártyára egy RDP *(Remote Desktop Protokol)* kapcsolat konfigurálható. A Terminal Services Configuration menüben be lehet állítani az RDP kapcsolat következő paramétereit:

- a kliens maximális kapcsolódási ideje
- titkosítási szint
- felhasználói jogok
- csoportos jogok



A Terminal Services jogai az RDP protokollon állíthatók be

Ha permissions pont alatt csak egy bizonyos csoportnak *(pl.:TERMSERVER-USERS)* van bármilyen jogosultság beállítva, akkor csak a csoport tagjai tudnak belépni, a többiek nem. *(Ezzel a beállítással egyébként felülbírálható, hogy Administration üzemmódban kik férhetnek hozzá a Terminal Serverhez. Gyárilag csak a rendszergazdák, de ha ügyesek vagyunk...)*

Forrás : Netacademia Windows 2000 lista

Windows 2000 - DNS áttelepítés

K: Hogyan lehet egy Windows NT 4.0 DNS kiszolgáló bejegyzéseit *(esetleg beállításait)* egy másik NT 4.0 DNS-re egyszerűen átrakni *(esetleg Windows 2000-re is)*?

V: A Windows NT 4.0 DNS szervere kétféleképpen működhet.

1. BIND *(Berkeley Internet Name Domain)* kompatibilis módban, ami rendkívül sok névszerverhez hasonló paraméterfájlok használatát jelenti.
2. Registry alapú módban a registryben tárolt bejegyzések képezik a DNS paramétereket a

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
↳ Services\DNS\Parameters
```

kulcs alatt.

A BIND kompatibilis mód két típusú szövegfájlt használ, mindkettő a %system%\system32\dns könyvtárban található.

- A BOOT fájl *(BIND elnevezése általában named.boot)* a szerver alapbeállításait tartalmazza.
- A zónafájlok, melyek az egyes név-cím hozzárendeléseket tartalmazzák a különböző domainekre; elnevezésük a rendszergazda feladata.

Registrymódban a fenti fájlok tartalmának megfelelő kulcsokban állíthatók be a DNS paraméterek. A legtöbb beállítás elvégezhető a DNSADMIN eszközzel, de néhány adat csak a Registryben szerkeszthető.

A két mód közötti átjárhatóságot a

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
↳ Services\DNS\Parameters\EnableRegistryBoot
```

kulcs jelenti, mely hamis értéke a BIND kompatibilis módot, igaz értéke a Registrymódot állítja be. Figyelem! BIND alapú módba kapcsoláskor a Registryben beállított értékek nem kerülnek át a szövegfájlokba!
Az adatok átvitele két DNS kiszolgáló között a két üzemmódban különböző. Elsőként állítsuk le a DNS kiszolgálót *(pl.:parancsmódban: net stop dns)*, majd:
BIND kompatibilis módban:

1. a %systemroot%\system32\dns könyvtár teljes tartalmát másoljuk át az új gép ugyanezen könyvtárába.
2. állítsuk be a Registryben a fenti kulcsot a használatnak megfelelően.
3. indítsuk el *(újra)* a DNS szolgáltatást

Registrymódban:

1. a %systemroot%\system32\dns könyvtár teljes tartalmát másoljuk át az új gép ugyanezen könyvtárába.
2. másoljuk át a forrás gép Registryjében lévő HKLM\SYSTEM\...\DNS *(lásd fentebb)* kulcs tartalmát az új gépre
3. indítsuk el *(újra)* a DNS szolgáltatást

Ja, és a Registry szerkesztése előtt végezzünk biztonsági mentést!
Forrás : Netacademia Windows 2000 Lista

Windows 2000 - Service Pack a telepítőben, slipstreaming
K: A slipstreaming használatáról keresek valami leírást *(how-to-t)* A technet elég szűkszavú, csak az update kapcsolók között említi.
V: A slipstreaming a Windows 2000 egyik legígéretesebb újdonsága, mely a Service Pack 1-ben volt először jelen. Lehetővé teszi olyan *(hálózaton megosztott)* telepítőkészletek létrehozását, melyekben a Service Pack javításai jelen vannak. Ezáltal a rendszertelepítést követően nincs szükség a SP külön telepítésére, s a későbbi komponenstelepítéseknél sem kell „újrahúzni" a javítócsomagot. Így készül:

1. Az eredeti Win2k telepítőkészletet *(a könyvtárstruktúra megtartásával (bootdisk, clients, i386...))* másoljuk fel egy könyvtárba *(pl. c:\w2kdistr)*
2. Csomagoljuk ki a Service Pack-et egy másik könyvtárba, *(pl. c:\win2kspack, de ha kicsomagolva megvan pl. CD-n, az is jó)*
3. Indítsuk el az update.exe-t a következő formában: update.exe /s:c:\w2kdistr
4. Ez végrehajtja a telepítő frissítését, amiről innentől kezdve utólagos Service Pack telepítés nélkül lehet Win2k-t feltenni.

Tippek és tapasztalatok:

1. Az új telepítőkészlet nemcsak merevlemezről és hálózaton tud működni. A bootolható CD sok helyen jobb megoldás lehet. Bootolható telepítő CD készítéséhez információt a következő weboldalakon találhatunk: **[1]**
2. Ha az új *(SP-kel kijavított)* telepítőkészletről készítünk boot floppyt, akkor az i386 könyvtárból másoljuk rá az első telepítő floppyra az új *(SP-vel kijavított)* txtsetup.sif fájlt
3. A recovery console-t nem frissíti inplace upgrade esetén, csak ha újrafuttatjuk a winnt32.exe-t /cmdcons kapcsolóval *(a slipstream-mel javított telepítőről)*.
4. RIS szerver esetén szükséges egy kis plusz munka, mivel próbálkozáskor a következő hibaüzenetet kapjuk: „The server to which you chose to replicate does not contain a cd-based image. The version of the cd-based image on the server must match the version of the system you are attempting to copy. Select a different server or add a cd-based image to this server."

Ez utóbbi problémát a következőképpen oldhatjuk meg:

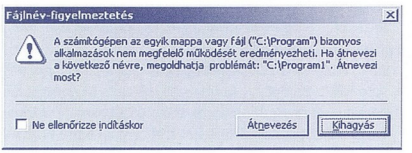
1. Másoljuk a teljes Win2000 telepítőkészletet egy megosztott könyvtárba
2. A fent leírt lépéseknek megfelelően végezzük el a slipstream frissítést *(update.exe /s:c:\w2kdistr)*
3. Mihelyst elkészült a javítás, a RIS szerveren el kell indítani a RISETUP.EXE programot, hogy új image-et adjunk a szerverhez. A javított könyvtárat kell megadni forrásként

Forrás : Netacademia Windows 2000 Lista

A cikkben szereplő URL-ek:
[1] http://www.windows2000faq.com/Articles/Index.cfm?ArticleID=13914

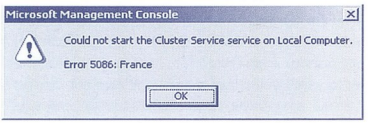
Problémamegoldás felsőfokon!

Problémamegoldás felsőfokon!



Petrény Zsolt a kihagyás mellett döntött: rosszul tette!

Az a sok-sok nemzet!



Úgy tűnik, a Francia jogi szabályozás nemcsak bizonyos titkosítóalgoritmusok, hanem a Cluster Service használatát is korlátozza!

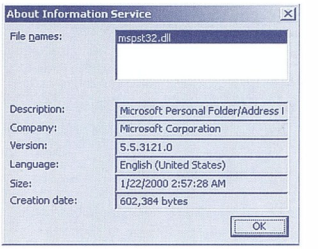
Multiuser environment

Gyúri Attila ugyanazon a bonyolult fájlon próbált dolgozni, mint Unknown. De ki az az Unknown?

Az univerzum mégsem tágul?

Nagypál Gábor az emberi tudás határait feszegetve bukkant a következő hibaüzenetre: a Windows súlyosan korlátozza szárnyaló fantáziáját!

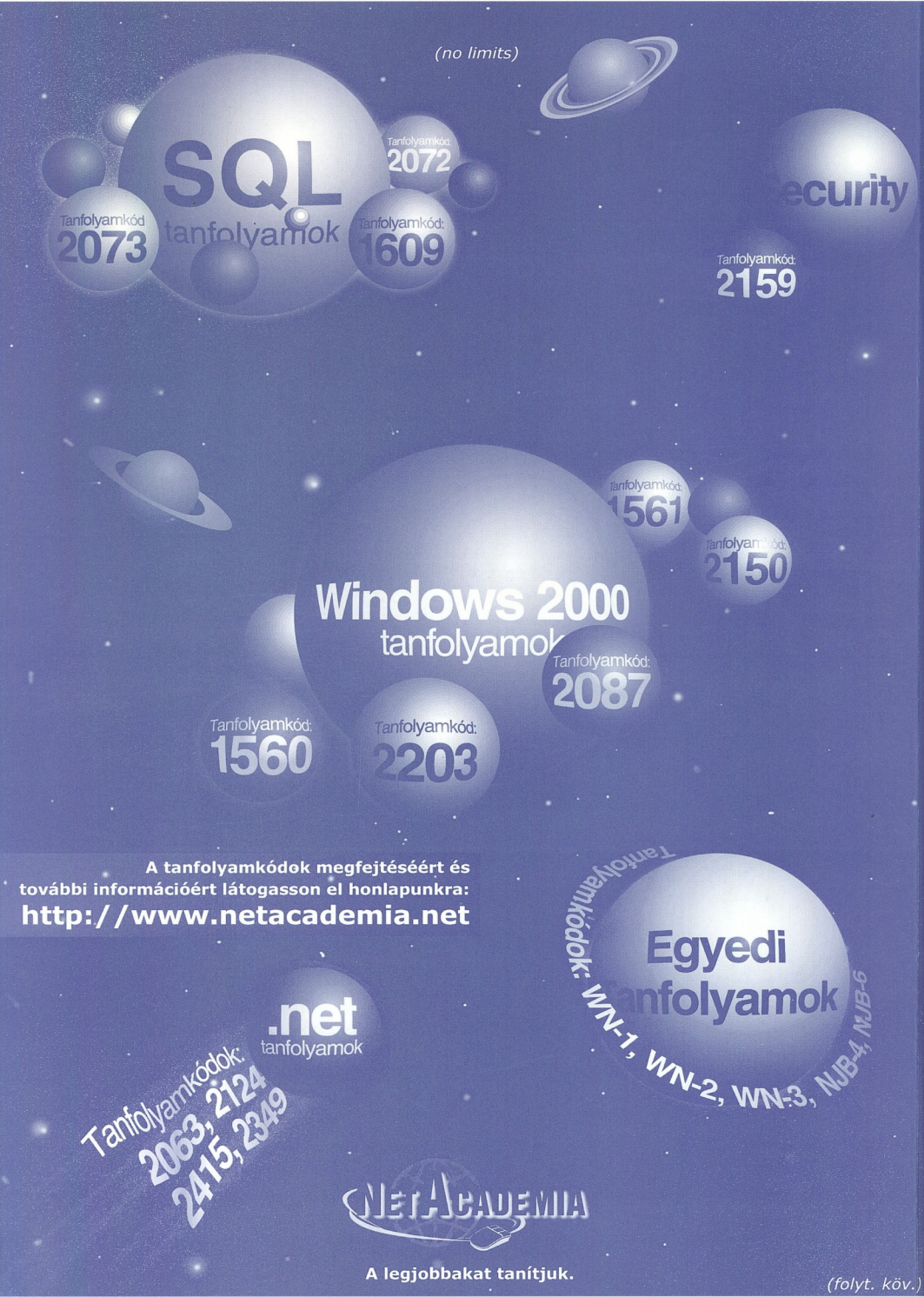
Tömény információ



Zalavári Balázs bukott el ezen a párbeszédpanelen, ugyanis képtelen volt felismerni a dátum formátumát: ritkán, de előfordul, hogy bájtban adják meg! 1 bájt = 0,2542e2 másodperc. A méret kifejezésére pedig komolyabb helyeken a DD/MM/YYY HH:MM:SS formátumot használják, mert az pontosabb mérést tesz lehetővé!

Helyreigazítás és bocsánatkérés

Sajnálattal értesültünk a napilapokból, hogy augusztusi szoftverrazzia tréfánk valóra vált, s az ott feltüntetett négy helyiségből háromban valóban volt rendőrségi akció. Akkor még senkit sem kívántunk kellemetlen helyzetbe hozni ártatlan tréfánkkal, de most, teljhatalmunknál fogva a következő városokba rendelünk szoftverrazziát októberre :-)


(no limits)
SQL
tanfolyamok
Tanfolyamkód:
2072
Tanfolyamkód:
2073
Tanfolyamkód:
1609
ecurity
Tanfolyamkód:
2159
Tanfolyamkód:
1561
Tanfolyamkód:
2150
Windows 2000
tanfolyamok
Tanfolyamkód:
2087
Tanfolyamkód:
1560
Tanfolyamkód:
2203
A tanfolyamkódok megfejtéséért és
további információért látogasson el honlapunkra:
http://www.netacademia.net
Egyedi
tanfolyamok
Tanfolyamkódok: WN-1, WN-2, WN-3, NJB-4, NJB-6
.net
tanfolyamok
Tanfolyamkódok:
2063, 2124
2415, 2349
NETACADEMIA
A legjobbakat tanítjuk.

(folyt. köv.)

Előfizetési szelvény

FAX

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

tech.net
A MICROSOFT MAGYARORSZÁG SZAKMAI MAGAZINJA

Tisztelt Olvasónk!

Lapunk hírlapárusi forgalomba nem kerül, ezért ha kíváncsi megkezdett sorozataink folytatására, kérjük töltse ki és juttassa el hozzánk az alábbi megrendelőlapot. Előfizetőinket szeretettel várjuk Mesterkurzusainkon ahol – rövid regisztráció után – tech.net klubtaggá is válhatnak. Klubtagjaink kedvezményesen vehetnek részt konferenciáinkon, tanfolyamainkon stb.

Kérjük töltse ki ezt az előfizetési szelvényt és faxolja el az (1) 261-7145-ös faxszámra.

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

Előfizetem a **tech.net** magazint: ...példányban 12+3 akcióval* (16.128 Ft)
...példányban egy évre (14.784 Ft)
...példányban fél évre (8.064 Ft)

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

Előfizető neve:. . . . . .

Cég neve: . . . . . .

Cím: ☐☐☐☐ . . . . . .

E-mail cím: . . . . . .

Telefon: . . . . . .

Fax:. . . . . .

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

Kelt:...../...../..... Aláírás:.....

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

Számlázási cím: **Szállítási cím:**

....

....

* A 12+3 akció lényege, hogy az éves előfizetés mellé még 3 visszamenőleges számot is rendelhet.

tech.net
A MICROSOFT MAGYARORSZÁG SZAKMAI MAGAZINJA