1.344 Ft

II. Évfolyam különszám

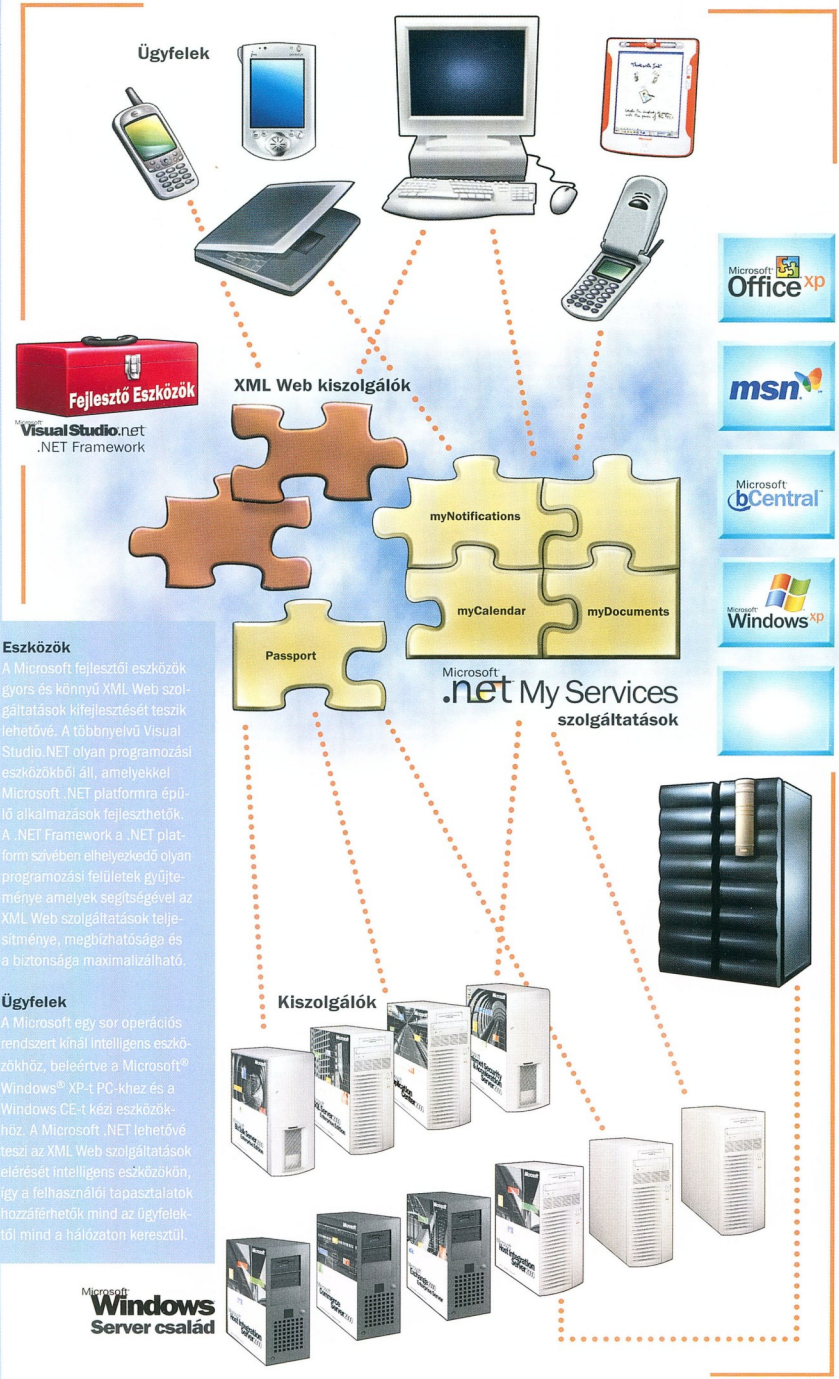
ISSN 15865185 13

tech.net

A Microsoft Magyarország szakmai magazinja

Microsoft Windows

A Microsoft .NET platform



Eszközök

A Microsoft fejlesztői eszközök gyors és könnyű XML Web szolgáltatások kifejlesztését teszik lehetővé. A többnyelvű Visual Studio.NET olyan programozási eszközökből áll, amelyekkel Microsoft .NET platformra épülő alkalmazások fejleszthetők. A .NET Framework a .NET platform szívében elhelyezkedő olyan programozási felületek gyűjteménye amelyek segítségével az XML Web szolgáltatások teljesítménye, megbízhatósága és a biztonsága maximalizálható.

Ügyfelek

A Microsoft egy sor operációs rendszert kínál intelligens eszközökhöz, beleértve a Microsoft® Windows® XP-t PC-khez és a Windows CE-t kézi eszközökhöz. A Microsoft .NET lehetővé teszi az XML Web szolgáltatások elérését intelligens eszközökön, így a felhasználói tapasztalatok hozzáférhetők mind az ügyfelektől mind a hálózaton keresztül.

Tapasztalatok

A Microsoft elérhetővé teszi felhasználói tapasztalatokat hozzáértő dolgozók, a vásárlói nagyvállalatok, kisvállalkozások, és a fejlesztők számára. Microsoft Office-szal, az MSN Internet eléréssel, a bCentral kisvállalkozói portállal és a Visual Studio.NET-tel kapcsolatban már eddig is széles kör felhasználói tapasztalatokat szereztünk. Ezen .NET tapasztalatok kötik össze az XML W szolgáltatásokat és a végfelhasználói szoftvereket, amelyek révén az egyedi felhasználói igényeket integrált módon le kielégíteni.

XML Web szolgáltatáso

Azon túlmenően, hogy XML Web szolgáltatásokat hoz lé a Microsoft a szolgáltatások olyan igényi építőelemeit alk ja meg, amelyek rutin feladato és tevékenységeket végezne el, és amelyek a fejlesztőnek biztos alapot jelentenek. Az létrehozott XML Web szolgált tások első csoportját .NET M Services-nek nevezték el. Ez a szolgáltatások sokkal inká felhasználó-központú és emb orientáltak, mint csupán spec lis eszközök, hálózatok vagy kalmazások. A .NET My Services alapja a Microsoft Passport felhasználói jogosult gi rendszer. A .NET My Servic a felhasználók számára a sz ség szerinti aktuális informác kat biztosítja az általuk éppe használt eszközön az általuk választott preferenciák szeri

Szolgáltatások

A Microsoft Windows kiszolg család és a Microsoft .NET vá lati kiszolgálók jelentik a legi infrastruktúrát XML Web szolg tatások kiépítésére, használa ra és működtetésére azáltal hogy támogatják az XML-t, a skálázható megoldásokat és üzleti folyamatok teljes skál az alkalmazásokon és a szolg tatásokon keresztül. Ezeket kiszolgálókat a működés sz pontjából kritikus teljesítmé szem előtt tartásával tervezt így biztosított a vállalati rends rek, alkalmazások és partne XML Web szolgáltatásokon resztüli integrálása és a vált üzleti körülményekhez való al mazkodás – a kellő gyorsaság

Microso

2001 Linux különszám

tech.net

A Microsoft Magyarország Szakmai Magazinja

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

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

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

Hirdetésfelvétel:
Bársonykalapács Marketing
Felelős: **Balogh Zoltán**
Tel./Fax: 214-0923
info@velvethammer.hu
1027 Budapest, Fő utca 67. V. 1.
Grafikai tervezés, kivitelezés,
nyomdai előkészítés:
Bársonykalapács Marketing
Művészeti vezető: **Balogh Zoltán**
Bársonykalapács © Copyright 2001

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

ISSN 1586-5185

Pingvinek és ablakok

Izgalmas vállalkozás eredményét tartja kezében Kedves Olvasóm. Amint a címlap sugallja, megpróbáltunk egy olyan különszámmal kirukkolni, mely lehetővé teszi a pingvineseknek, hogy benézzenek az ablakon.

Tudom, hogy sok Linuxos eleve előítélettel tekint mindenre, ami ablakos, és nem hiszem, hogy ez az egyetlen különszám meg tudná téríteni az ily vakhitű egyedeket. A nyitott elméjű, kíváncsi pingvinek kukucskálására azonban feltétlenül számítunk!

Mit lát az, aki benéz?

Nem egy monolitikus, önmagának való operációs rendszert és alkalmazás-családfát fog látni, hanem szabványokon alapuló, nyitott eszközöket. Régen elmúlt már az az idő, amikor a Microsoft helyes stratégiának tartotta a szabványok semmibe vételét. Most olyan időket élünk, amikor Billy *(Gates)* tucatjával hajítja ki termékeiből az oly sokat kritizált zárt, önmagának való megoldásokat. Szemétre kerül lassan a NetBIOS, a MAPI, az SMB, s helyüket fokozatosan átveszi a natív TCP/IP, az SMTP és a HTTP. A folyamat valamikor 1995-ben kezdődött, amikor Billy először szembesült azzal, hogy ha a Microsoft nincs tekintettel a világra, akkor a világ sincs tekintettel őrá. Mindannyian tudjuk, hogy az Internet vonatára az utolsó pillanatban ugrottak fel, amikor a szerelvény már elhagyta a peront. A Windows 95 kibocsátásakor egy-két hónap leforgása alatt vett 180 fokos fordulatot a cég, és állt a szabványok támogatása mellé. Ennek már hat éve. Ezért is megdöbbentő számunkra, hogy a régi, berögzült mondókákat egyesek akár fél évtizeddel azután is lelkesen, nyíltan hajtogatják, hogy azok létalapja megszűnt.

Szabványok mindenütt

A tech.net magazin egyik legnépszerűbb rovata a „szabványok". Ebben hónapról hónapra egy-egy RFC-nek, Internetes szabványnak eredünk nyomába, s tárjuk olvasóink elé ezek lényegét. Így került sor a Microsoft Magyarország Szakmai Magazinjában a MIME *(levelezés)*, az MD5 *(titkosítás, hash)*, a HTTP kiegészítése *(a WebDAV)*, az XML *(adatleírónyelv)*, a SOAP *(eljáráshívás)* és sok más technológia ismertetésére. Még ott is szabványokkal találkozunk, ahol nem is gondolnánk. Nem lennénk képesek az Internet adta lehetőségek kihasználására, ha operációs rendszereink hálózati felépítése nem lenne alapjaitól a tetejéig szabványos.

És mi a helyzet a kernellel?

Be kell vallanunk, hogy a kernel nem szabványos. De nem ám. Viszont tegye a szívére szárnyacskáját minden pingvin: a Linux kernele szabványos? Ugyan! Éppúgy az adott operációs rendszer lelkivilágához faragott egyedi kód, mint a Windows esetében. Sőt, továbbmegyek: egy tőről fakadnak! Honnan is?

WNT

Ez a Windows NT rövidítése. Játszunk el e három betűvel, matekozzunk rajtuk, hátha bűvös számmisztikánkkal eljutunk valahova, ahol még senki sem járt. Csúsztassuk el mindhármat egyel visszafelé az ASCII kódtáblában. Mit kapunk? Igen, az az! VMS!

VMS

A Windows NT operációs rendszer kernele szoros rokonságot mutat a Digital ősi „nagygépes" operációs rendszerével, a VMS-sel. A hasonlóság annyira szembetűnő, hogy egyszerűen nem lehet a véletlen műve. A virtuális memóriakezelés, a végrehajtási szálak prioritása, a preemptív multitaszk belső működése egyszerűen Digitalos és kész. *(A kernel működéséről 2001 augusztusi és szeptemberi számunkban jelent meg részletes elemzés.)* A jelenség magyarázata egyszerű: miután a Digital „nagygépes" üzletága végleg a PC háború áldozatává vált, Bill Gates, a hős, megmentette az ottani kiváló szakembereket a biztos éhenhalástól, és átvette a VMS fejlesztőcsapatát a Microsofthoz. A VMS atyja, David Cutler lett 1992 táján a születő új operációs rendszer, a Windows NT atyja!

HAL

A Windows platformfüggetlen, azaz nem csak Intel processzoron fut. Nem igazán tudtak elterjedni az Intel ellenlábasai, a PowerPC konzorcium, az Alpha és a MIPS, de volt Windows változat ezekre a masinákra, mivel az operációs rendszer jelentős részét C nyelven írták, így könnyedén lefordítható volt más gépi kódra is. Majdnem az egész oprendszer C-ben készült, annak a roppant hardverközeli kódnak a kivételével, mely például a processzorok védelmi szintjeit, a hardver virtuális memóriakezelésének különbözőségeit kezeli le. A HAL nem más, mint a Hardware Abstraction Layer, azaz a „hardvereldugási réteg". Ennek a résznek bizony gépi kódban kell készülnie, hogy gyorsabb legyen, mint a villám. Matekozzunk ezzel is! Csúsztassuk el ezt a három betűt is az ASCII kódtáblában, ezúttal felfelé. Na mi jön ki? Az bizony!

IBM

International Business Machines. A Microsoft a kilencvenes évek elején az IBM-mel karöltve reszelgette az OS/2-t. Akkoriban Billy azt hangoztatta, hogy a jövő operációs rendszere egyértelműen az OS/2. Majd összerúgták a port, és Billy különvált. A kész kódot azonban mindkét fél magáénak tekintette, és magával is vitte. Így került a Windows NT-be jó csomó IBM megoldás...

Fóti Marcell

Hálózati terheléselosztás

A Network Load Balancing *(továbbiakban NLBS)* a Windows 2000 Advanced Server és a Windows 2000 Datacenter Server operációs rendszerek beépített szolgáltatása. Az NLBS nagyobb rendelkezésreállást és stabilitást biztosít az Internet-kiszolgáló alkalmazások *(például web- vagy FTP, és más, nagy fontosságú kiszolgálók)* számára. A Windows 2000-t futtató számítógép önállóan a nagy rendelkezésreállási elvárásoknak csak korlátozottan felel meg, de az NLBS segítségével fürtbe rendezett Windows 2000 Advanced Server-ek kielégítik a nagy fontosságú és –rendelkezésreállású rendszerekkel szemben támasztott biztonsági és teljesítmény-elvárásokat.

A szükséges kiszolgálóalkalmazások *(web-, FTP-, telnet-, vagy levelezőkiszolgálók)* a fürt minden tagkiszolgálóján futhatnak. A szolgáltatások egy része *(például a webkiszolgáló)* az összes tagkiszolgálón egyidejűleg működik, az NLBS pedig a beérkező hálózati terhelést elosztja a tagok között, míg más szolgáltatásokat *(például a levelezést)* egyidejűleg csak a fürt egy tagkiszolgálója látja el. Minden esetben az NLBS feladata, hogy hiba esetén a kiesett kiszolgáló forgalmát egy másik, működő taghoz irányítsa át.

Az NLBS működéséhez szükséges konfiguráció

A Windows 2000 NLBS szolgáltatása hálózati meghajtóként működik, méghozzá olyan alacsony szinten, hogy a TCP/IP hálózati összetevők az NLBS jelenlétét nem is észlelik.

A legjobb teljesítmény elérése érdekében az NLBS általában számítógépenként két hálózati csatolót használ: az egyik csatoló feladata az ügyfelek kiszolgálása *(mint rendesen)*, a másik pedig az NLBS fürt „adminisztratív" hálózati forgalmát kezeli. Természetesen nincs akadálya annak sem, hogy minden forgalom gépenként egyetlen hálózati csatolón keresztül bonyolódjon.

Az NLBS fürtalkalmazások adatbáziselérése

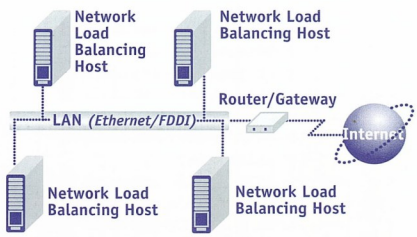
Sok kiszolgálóalkalmazás működése közben adatbázisban dolgozik. Ha egy fürtben több ilyen alkalmazás működik egyidejűleg, akkor különös gondot kell fordítani az adatbázisműveletek szinkronizálására. Egyszerűbb esetben minden tagkiszolgáló dolgozhat a saját, offline adatbázismásolatán, majd szükség esetén a változásokat frissítik *(migrálják)* egymás között, vagy a központi, „valódi" adatbázisban. Máskor egyidejűleg, hálózaton keresztül érik el az adatbáziskiszolgálót, de előfordulhat a fenti két eset kombinációja is. Például: a weblapok forrásállományait nyugodtan felmásolhatjuk az összes tagkiszolgálóra, így rövidebb válaszidőt és nagyobb hibatűrést kapunk. A weblapokon keresztül érkezett adatbáziselérést igénylő műveleteket ugyanakkor kezelheti különálló, megbízható adatbáziskiszolgáló, amely egyszerre az összes tagkiszolgálót ellátja.

A fontos vállalati alkalmazások a hibatűrő szolgáltatás biztosítása érdekében nagy rendelkezésreállású adatbáziskonfigurációt igényelnek. Ilyenkor maga az adatbáziskiszolgáló is fürtbe rendezett számítógépeken fut, így biztosítja a nagy rendelkezésreállást és a méretezhetőséget. Ilyen adatbáziskiszolgáló lehet például a Cluster szolgáltatás segítségével megvalósított kétpontos fürtre telepített Microsoft SQL Server. A Cluster szolgáltatás biztosítja, hogy ha a két csomópont közül az egyik kiesik *(a csomópontot megvalósító számítógép meghibásodik)*, a másik csomópont átveszi a hibás egység feladatait. Teheti mindezt azért, mert a fürt két csomópontja közös merevlemez-alrendszert használ. Az SQL Server ügyfelei pedig az átállásból csaknem semmit sem vesznek észre.

Fontos, hogy a két fürtözési módszert megkülönböztessük egymástól. Az első, az NLBS fürt feladata elsősorban a beérkező TCP/IP hálózati forgalom elosztása a rendelkezésre álló tagkiszolgálók között. Ezek a független tagkiszolgálók egy NLBS fürtöt képeznek. A második módszer a Cluster szolgáltatás segítségével megvalósított fürt, amelyet két vagy több, fizikailag szorosan összekapcsolt, közös lemez-alrendszert használó számítógép valósít meg. A Cluster szolgáltatás segítségével leggyakrabban nagy rendelkezésreállású adatbáziskiszolgálókat építenek *(amit azután természetesen akár NLBS fürtök háttérkiszolgálójaként is lehet használni)*. A két fürttípus együttes használatával teljeskörű, hibatűrő és nagy rendelkezésreállású fürtkörnyezetet teremthetünk.

A hálózati terheléselosztás működése

A hálózati terheléselosztás segítségével megbízható és méretezhető hálózati kiszolgálót hozhatunk létre két vagy több tagkiszolgáló számítógép fürtbe rendezésével. Az internetes ügyfelek a fürtöt a fürt saját IP címén *(címein)* keresztül érik el. Az ügyfelek számára a fürthöz való csatlakozás ugyanolyan, mintha közvetlenül a fürt egy tagkiszolgálójához kapcsolódtak volna; a kiszolgálón futó alkalmazások sem észlelik, hogy egy fürtbe rendezett számítógépen futnak. *(A Cluster fürt előnyeit csak speciális, erre tervezett alkalmazások képesek kihasználni.)* Az NLBS fürt azonban mégsem ugyanolyan, mintha több különálló számítógépen futtatnánk az alkalmazásokat, hiszen a tagkiszolgáló hibája esetén a szolgáltatás nem szakad meg, csak a bejövő hálózati kapcsolatok átkerülnek a fürt még működő tagkiszolgálóihoz. A fürt ezenkívül – a terheléselosztással működő portokon – nagyobb hálózati terhelést képes elviselni, miközben gyorsabb válaszidőt biztosít az ügyfelek számára.



Ha egy NLBS fürt tagkiszolgálója meghibásodik, a Network Load Balancing a beérkező hálózati forgalmat automatikusan a még működő tagkiszolgálók között osztja el. A leálló tagkiszolgálóval létesített hálózati kapcsolatok elvesznek, de a szolgáltatás a fürtben továbbra is elérhető marad. A legtöbb esetben *(például webkiszolgálók esetén)* az ügyfélszoftver automatikusan újra megpróbálkozik a megszakadt kapcsolatok felépítésével *(tudtán kívül, de ezúttal már egy másik, még működő tagkiszolgálóval)*, az ügyfél pedig ebből mindössze csak a néhány másodperces várakozást veszi észre.

Az NLBS a fürt egy vagy több saját IP címére érkező hálózati forgalmat elosztja a rendelkezésre álló tagkiszolgálók között. A tagkiszolgálók egyidejűleg válaszolnak a különböző ügyfelek kéréseire, akár egy ügyfél több kérésére is. Például: az ügyfél által megtekintett weboldal forráskódja a fürt egyik, míg az oldalon található képek a fürt egy másik tagkiszolgálójáról érkezik. Ez meggyorsítja a kiszolgálóműveleteket és csökkenti a válaszidőt.

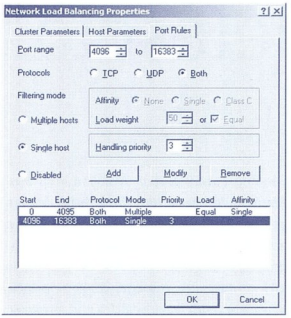
Az NLBS fürt minden – közös alhálózaton található – tagkiszolgálója észleli a teljes bejövő hálózati forgalmat. A tagkiszolgálókon található NLBS eszközmeghajtó szűrőként működik a fürtre csatlakozó hálózati kártya és a számítógép TCP/IP komponensei között: a bejövő forgalom egy részét továbbengedi, más részét elnyeli *(a TCP/IP nem is tud róla, hogy fürtben működik – hiszen hozzá már csak az a hálózati forgalom jut el, amit az NLBS eszközmeghajtó jónak lát)*.

A Network Load Balancing teljesen elosztott algoritmus segítségével az ügyfél IP címe, a port és egyéb információk alapján rendeli a beérkező kéréseket a tagkiszolgálókhoz. A beérkező csomagokat minden tagkiszolgáló megkapja és kielemzi, majd az algoritmus segítségével eldönti, melyik tagkiszolgáló dolgozza fel a kérést. Ez a hozzárendelés egészen addig érvényben marad, míg a fürtben található tagkiszolgálók száma meg nem változik. Az NLBS szűrőalgoritmus sokkal hatékonyabb és nagyobb sávszélesség kiszolgálására képes, mint a központosított terheléselosztó megoldások, amelyekben a központi egység a hálózati csomagokat módosítja, majd továbbküldi a megfelelő tagkiszolgálóhoz. Az NLBS a tagkiszolgálókon fut, így működése nem korlátozódik adott processzortípusokra vagy hálózati megoldásokra.

A hálózati forgalom elosztása

Az NLBS a következő módon osztja el a beérkező Transmission Control Protocol *(TCP)* és User Datagram Protocol *(UDB)* csomagokat a tagkiszolgálók között: a fürt saját IP címére érkező csomagokat a fürt minden tagkiszolgálója megkapja, majd az NLBS szűrők a csomagokban használt TCP és UDP portok alapján megszűrik ezt a forgalmat, mielőtt azok még a TCP/IP meghajtót elérnék.

Az NLBS csak a TCP és UDP csomagokat dolgozza fel, nem szűri a beérkező Internet Control Message Protocol *(ICMP)*, Internet Group Membership Protocol *(IGMP)*, az IP- és hálózati címek *(MAC address)* összerendelésére használt Address Resolution Protocol *(ARP)* és más IP protokollokat sem. Minden ilyen forgalom akadály nélkül továbbítódik a tagkiszolgálók TCP/IP meghajtói felé. A TCP/IP felépítéséből következik, hogy nem okoz gondot a fürt IP címéről, több tagkiszolgálótól egyidejűleg érkező többszörös válasz, bár ez a pont-pont alapú TCP/IP alkalmazások *(például a ping)* esetén probléma lehet. Ilyenkor a tagkiszolgálók saját, közvetlen IP-címét kell használni.



A hálózati terheléselosztás szabályainak beállítása

Konvergencia

Az NLBS fürt tagkiszolgálói szabályos időközönként broadcast vagy multicast üzenetekkel hangolják össze a működésüket, és felügyelik a fürt működését. Amikor a fürt állapota megváltozik *(ez történhet egy tagkiszolgáló leállásával, kilépésével vagy éppen megjelenésével)*, a fürtben elindul egy folyamat, melyben a tagkiszolgálók feltérképezik a fürt új állapotát, és alap értelmezett tagot választanak *(ez a legkisebb prioritású tagkiszolgáló lesz)*. Ezt a folyamatot nevezzük konvergenciának. Miután ez megtörtént, a Windows 2000 eseménynaplójában megjelennek a konvergencia befejezésére utaló bejegyzések.

A konvergencia alatt a kiesett tag kivételével minden kiszolgáló folytatja a működést, a változás a még működő tagokkal létesített hálózati kapcsolatokat nem érinti. A konvergencia befejeztével a kiesett tagkiszolgáló felé létesített hálózati kapcsolatokat a még működő tagkiszolgálók átveszik. A hálózati forgalom úgy oszlik el a tagkiszolgálók között, hogy a hálózati terhelés egyenletes maradjon. Ha a fürtben új tagkiszolgáló jön létre, a konvergencia során belép a fürtben működő tagkiszolgálók sorába. A fürt bővítése nem zavarja a meglévő hálózati kapcsolatokat, sem az ügyfél-, sem a kiszolgálóalkalmazásokat. A több TCP kapcsolatot egyidejűleg használó alkalmazásoknál azonban előfordulhat, hogy a konvergencia során – az affinitás ellenére – a két TCP kapcsolat különálló tagkiszolgálóra kerül. *(Hiszen az affinitás csak a fürt felépítésének megváltozásáig érvényes.)*

A Network Load Balancing fürt tagkiszolgálója addig számít „élőnek", míg az folyamatosan részt vesz a fürt tagkiszolgálói között folyó kommunikációban. Ha a tagkiszolgálók adott ideig nem kapnak üzenetet egy másik tagkiszolgálótól, azt hibásnak veszik, és megkezdődik a konvergencia a kieső tagkiszolgáló terhelésének elosztására. Az üzenetek közötti időtartam és a konvergencia indításáig kimaradó üzenetek száma beállítható, az alapértelmezés 1000 ms *(egy másodperc)*, és 5 kihagyott üzenet. Mivel ezek a paraméterek ritkán változnak, a Network Load Balancing dialógusablakából nem módosíthatók: értékük szükség esetén a regisztrációs adatbázisban változtatható meg.

Kerberos

A Kerberos Version 5 a Windows 2000 új hitelesítési protokollja. A protokoll önmaga egyáltalán nem új, immár több mint egy évtizednyi fejlesztés eredménye. Ebben a cikkben áttekintjük a Kerberos alapjait, használatának előnyeit, valamint azt, hogy miképp használja a Windows 2000 a Kerberost mint hitelesítő protokollt.

Hitelesítés a Windows 2000-ben

A Windows 2000 rendszer erőforrásainak eléréséhez a felhasználónak hitelt érdemlően azonosítania kell önmagát. Ez az azonosítás általában a rendszerbe való belépéskor történik meg, ezután a felhasználó egy testre szabott „access token"-t kap, ami tartalmazza jogosultságait, csoporttagságát, stb. A szolgáltatást nyújtó eszközök ezt az access tokent ellenőrzik, majd ez alapján döntik el, hogy a felhasználó hozzáférhet-e egy erőforráshoz *(megosztott fájlhoz, számítógéphez, stb.)*, vagy sem. A Windows 2000 hálózati *(tehát nem telefonos)* kapcsolaton keresztül felhasználóit két protokoll segítségével hitelesítheti:

- Kerberos Version 5 – A Windows 2000-t használó számítógépek között Windows 2000 tartományban alapértelmezett hitelesítési protokoll.
- Windows NT LAN Manager *(NTLM)* – Az NTLM a Windows NT 4.0 alapértelmezett hitelesítési protokollja volt, a Windows 2000 elsősorban kompatibilitási okokból tartalmazza. A különálló, nem tartományi tag *(stand alone)* Windows 2000 operációs rendszer – később látni fogjuk, hogy nyilvánvaló okokból – is NTLM hitelesítést használ.

A Windows régebbi verziói *(Windows 3.11, Windows 9x, Windows NT 4.0)* az NTLM *(illetve LanMan)* hitelesítést fogják használni a Windows 2000-hez való csatlakozáskor. Ugyancsak NTLM hitelesítést használ a Windows 2000 is, amikor Windows NT 4.0 számítógép vagy tartomány erőforrásaihoz szeretne hozzáférni. Ha azonban a Windows 2000-nek lehetősége van választani, választása minden lehetséges esetben a Kerberos Version 5 protokoll lesz.

A Kerberos hitelesítés előnyei

Vajon miért ragaszkodik a Windows 2000 ennyire a Kerberos protokollhoz? Az NTLM hitelesítésnek köztudottan több hátránya van: egyrészt, a kapcsolatfelvétel során a felhasználó jelszavából létrehozott „kivonat" *(hash)* többször megjelenik a hálózaton – ez bizonyítottan nem igazán biztonságos –, másrészt pedig a tartományok közötti hitelesítés folyamata kettőnél több tartomány esetén olyannyira elbonyolódik, hogy gyakorlatilag nem is lehetséges. Érthető tehát a szakemberek és a Windows 2000 fejlesztőinek törekvése: nem elég, hogy egy új, biztonságosabb, hatékonyabb hitelesítési protokollra van szükség, de lehetővé kell tenni, hogy ez a protokoll teljes Windows 2000 környezetben 100%-ig kiválthassa az NTLM hitelesítést.

A Kerberos használata ezenkívül számos további előnnyel jár: Gyorsabb kapcsolat – NTLM esetében a kiszolgáló *(akihez az ügyfél fordult)* a tartományvezérlő segítségével hitelesítette az ügyfelet *(ez volt a „pass through")*. A Kerberos használatával a kiszolgáló önmaga hitelesítheti az ügyfelet, nincs szükség a tartományvezérlő közbenjárására, így csökken a hálózat terhelése. Ezen kívül – az NTLM-mel ellentétben – elég, ha az ügyfél egyszer beszerzi az adott kiszolgálóhoz szükséges hozzáférési lehetőséget, amit a kapcsolat további részében igény szerint akár többször is felhasználhat.

Kölcsönös hitelesítés – Az NTLM-et alkalmazó rendszerekben a kiszolgáló azonosította az ügyfelet, az ügyfél *(vagy akár egy másik kiszolgáló)* azonban nem lehetett biztos abban, hogy a kiszolgáló valóban az, akinek azt ő hiszi. A Kerberos protokoll ilyen értelemben nem tesz különbséget: kiszolgálónak számít mindenki, aki egy szolgáltatást mások által elérhetővé tesz, és ügyfél mindenki, aki egy ilyen szolgáltatáshoz szeretne hozzáférni. A Kerberos hitelesítés mindkét résztvevője biztos lehet abban, hogy a kapcsolat másik végén az található, akinek az mondja magát.

Meghatalmazásos hitelesítés – A Windows szolgáltatások, miközben egy ügyfél nevében tevékenykednek, egyfajta „megszemélyesítést" alkalmaznak, így a szolgáltatásnak mindenhez pontosan annyi joga van, mint a hozzá kapcsolódó ügyfélnek. A helyi számítógépen belüli ilyen megszemélyesítéseket mind az NTLM, mind a Kerberos lehetővé teszi. Csak a Kerberos képes viszont a „megszemélyesítés" jellegű hitelesítésre, amire a mai, háromrétegű alkalmazások esetén sokszor szükség van – nevezetesen, hogy az alkalmazás középső rétegén található kiszolgáló az alsó réteg kiszolgálójával *(például egy adatbáziskiszolgálóval)* az ügyfél nevében és az ő jogaival felruházva tartja a kapcsolatot. Az NTLM nem támogatja a hasonló megoldásokat.

A tartományok közötti megbízotti kapcsolatok kezelése egyszerűbb – a kölcsönös hitelesítésnek köszönhető, hogy a Windows 2000 tartományok kapcsolatai alapértelmezésben kétirányúak és tranzitívak *(tranzitív: ha A tartomány megbízotti viszonyban áll B tartománnyal, B pedig C-vel, akkor A automatikusan megbízotti viszonyba kerül C-vel is)*. A Windows 2000 tartományi fába rendezett tartományok kapcsolatai sokkal egyszerűbbek és átláthatók. A fa bármely tartománya, illetve tartományi erdő esetén az erdő bármely fájának bármely tartománya által kiadott hitelesítés az adott fában illetve erdőben érvényes.

Együttműködés – A Windows 2000 Kerberos implementációja az Internet Engineering Task Force *(IETF)* által ajánlott szabványokon alapul. A szabványos megvalósítás jó alapot teremt más, ugyancsak Kerberos Version 5 hitelesítési protokollt használó rendszerekkel való együttműködésre.

A Kerberos hitelesítési protokoll szabványai

A Kerberos hitelesítési protokollt a Massachusetts Institute of Technology *(MIT)* szakemberei az úgynevezett „Project Athena" projekt keretében már több, mint tíz éve fejlesztik. A protokoll első nyilvános változata már a negyedik verzió volt. A v4 áttekintése és széles körű felhasználása során összegyűlt tapasztalatok alapján fejlesztették ki a jelenlegi, ötödik verziót, amely jelenleg szabványjavaslatként található meg. A Windows 2000 Kerberos az RFC1510 implementációja, a Kerberos segítségével átadott biztonsági tokenek *(a felhasználó hozzáférési jogosultságai az egyes erőforrásokhoz)* pedig az RFC1964 Internet szabványjavaslatot *(GSS API)* követik.

A Kerberos bővítményei

A Kerberos protokoll alapvetően szimmetrikus *(közös kulcsú)* titkosítási algoritmuson alapul, ezért a Windows 2000 az eredeti Kerberos protokoll kibővített változatát használja. Természetesen ez a bővítmény is egy IETF szabványtervezet megvalósítása. Ez teszi lehetővé, hogy a Kerberos hitelesítési folyamat korai szakaszában kihasználhassuk az aktív memóriakártya *(„smart card")* és egyéb nyílt kulcsú infrastruktúrák előnyeit. A Kerberos és a nyílt kulcsú hitelesítés kapcsolatáról részletesebben a cikk második felében ejtünk szót.

A protokoll áttekintése

A Kerberos hitelesítő protokoll egy kommunikációs kapcsolat résztvevőinek kölcsönös azonosítását teszi lehetővé olyan környezetben, ahol az átvitt adat illetéktelenek által látható, módosítható, egyszóval a biztonságos adatátvitel nem garantálható. Ez a környezet – természetesen – nagyon hasonlít napjaink Internetéhez, ahol az illetéktelen behatoló átveheti akár az ügyfél, akár a kiszolgáló szerepét, és így értékes információkhoz juthat. Feladatunk tehát egy olyan biztonságos kommunikáció megvalósítása, amely az első kapcsolatfelvételtől kezdve kizárja az illetéktelen hozzáférést.

Alapok – avagy legyen egy közös titkunk

A Kerberos protokoll alapja a közös titkon alapuló hitelesítés. Az elv egyszerű: ha egy titkot csak két ember ismer, azok a közös titok segítségével könnyedén azonosíthatják egymást.

Vegyük példaként az angolszász világban oly népszerű párost, Alicet és Bobot. Tegyük fel, hogy Alice gyakran küld üzeneteket Bobnak, Bob viszont szeretne biztos lenni abban, hogy az Alice-tól kapott leveleket valóban Alice írta. Elhatározzák, hogy együtt választanak egy jelszót, amit nem árulnak el kettőjükön kívül senkinek. Ha Alice üzenetén valahogy látszik, hogy Alice ismeri a közös titkot, Bob biztos lehet abban, hogy a levél valóban Alicetől származik.

Adódik azonban egy probléma: honnan derül ki, hogy Alice valóban ismeri a titkot? Ha Alice beleírná azt az üzenetbe, akkor a nevető harmadik, a kis gonosz Carol elég, ha csak egy ilyen üzenetet elkap a hálózaton, a titok máris nem titok többé. Valami más megoldásra van szükség: anélkül kell jeleznünk a jelszó ismeretét, hogy azt beleírnánk az üzenetbe.

A Kerberos ezt közös kulcsú *(más néven titkos kulcsú, vagy szimmetrikus)* titkosítás segítségével oldja meg. A szimmetrikus titkosítás lényege, hogy egy adott, közös kulcs segítségével titkosított üzenetet ugyanazzal a kulccsal tudunk csak megfejteni. A kommunikáció résztvevői pedig ezt a kulcsot használják közös titokként. A titok ismerete egyszerűen bizonyítható: egyik részről az üzenet titkosítása, másik részről a megfejtése a bizonyíték, hiszen a közös kulcs ismerete nélkül ezek egyike sem lehetséges.

Hitelesítők *(authenticators)*

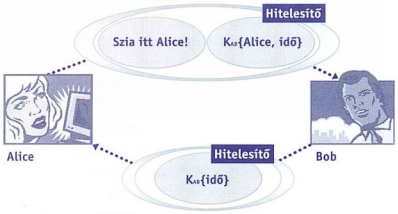
Vegyünk egy egyszerű, titkos kulcson alapuló hitelesítési protokollt. A kapcsolatfelvétel elején valaki áll a virtuális ajtó előtt és bebocsátást kér. A belépéshez egy, a közös kulccsal titkosított adatcsomagot, úgynevezett hitelesítőt nyújt át. Az adatcsomag eredeti tartalmának minden esetben másnak kell lennie, különben a hálózaton elkapott hitelesítőt később bárki illetéktelenül felhasználhatná. A kapu őre átveszi a hitelesítőt, majd az általa ismert titkos kulccsal megpróbálja megfejteni. Ha ez sikerült, az őr biztos lehet abban, hogy az ajtó előtt kívül olyasvalaki áll, aki ismeri a közös kulcsot. *(Márpedig azt csak ketten ismerik: az őr és a belépésre jogosult felhasználó.)*

Ha a hitelesítés kölcsönös, az őrnek is hitelesítenie kell önmagát a felhasználó előtt. A teljes üzenetet nem küldheti vissza, hiszen azt a kulcs ismerete nélkül is bárki megtehetné. Inkább a megfejtett üzenet tartalmát valahogy megváltoztatja, majd a módosított adatcsomagot titkosítja, és hitelesítőként visszaadja az ajtó előtt álló idegennek. Az a saját kulcsával megfejti az üzenetet, ellenőrzi a tartalmát, és ha mindent rendben talált, biztos lehet abban, hogy az ajtónálló is rendelkezik a megfelelő kulccsal: hiszen képes volt megfejteni az üzenetet, megváltoztatni a tartalmát, majd titkosítva visszaküldeni azt. Ebben a pillanatban az őr és az idegen azonosították egymást.

Lássuk ugyanezt kicsit részletesebben, Alice és Bob példáján. Alice a felhasználó, aki szeretne hozzáférni Bob egy szolgáltatásához.

1. Alice egy üzenetet küld Bobnak, amely a saját nevéből, valamint egy, a közös kulccsal titkosított hitelesítőből áll. Ebben a példában a hitelesítő két adatmezőt tartalmaz: az egyik valami, ami azonosítja Alicet, mondjuk a neve; a másik pedig az Alice számítógépén mért pontos idő.
2. Bob megkapja Alice üzenetét. Az üzenetből azonnal látszik, hogy olyasvalaki küldte, aki Alicenak mondja magát. Bob a közös kulccsal megfejti a hitelesítőt, és így hozzáfér Alice valódi nevéhez, valamint az Alice számítógépén mért időhöz. A protokoll működéséhez fontos, hogy Bob és Alice számítógépének órái szinkronizálva legyenek. Tegyük fel, hogy a két óra közötti eltérés soha nem nagyobb, mint öt perc. Bob tehát összehasonlítja a hitelesítőben kapott időt és a saját óráját. Ha az eltérés kevesebb, mint öt perc, Bob majdnem biztos lehet abban, hogy az üzenetet Alice küldte. Azért csak majdnem, mert lehetséges, hogy Alice hitelesítőjét valaki a hálózaton elkapta, majd változatlanul, de öt percen belül újrakülte. Ez a megismételt hitelesítő a fentiek alapján még érvényesnek számítana. Bob ezt úgy kerülheti el, hogy megjegyzi az Alicetől utoljára megérkezett hitelesítő időpontját, és ha annál régebbi, vagy azzal megegyező időpontú hitelesítőt kap, legalábbis gyanakodni kezd. Ha azonban minden rendben van, immár biztos lehet abban, hogy az üzenet Alicetől érkezett.
3. Most Bobon a sor: kiveszi az időpontot az Alicetől kapott hitelesítőből, azt a közös kulccsal újra titkosítja, majd visszaküldi Alicenek.

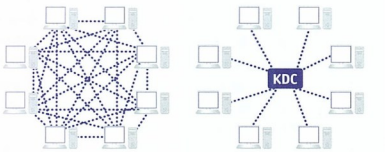
Figyeljük meg, hogy Bob nem a teljes hitelesítőt küldte vissza – hiszen arra bárki képes lenne –, hanem annak csak egy részét. Alice így biztos lehet abban, hogy Bob rendelkezik a közös kulcs egy példányával, hiszen képes volt megfejteni az üzenetet, módosítani a tartalmát, majd újra titkosítani azt. Alice megkapja Bob üzenetét, visszafejti a titkosított adatokat, majd a kapott időpontot összehasonlítja az ő általa küldött adattal. Ha a két időpont megegyezik, akkor Alice biztos lehet benne, hogy üzenetét Bob kapta meg, hiszen a közös kulcsot csak ők ketten ismerik.



Alice és Bob kölcsönös azonosítása (K_{AB} a közös kulcs, K_{AB}{adat} a közös kulccsal titkosított adat jele)

Központosított kulcskezelés *(Key Distribution)*

Ez azonban még korántsem jelent megoldást minden problémára. Mindenekelőtt feltételezzük, hogy Alice és Bob előzőleg valamilyen biztonságos csatornán *(mondjuk, fülbesúgással)* megegyeztek a közös titokban. A számítógépek azonban viszonylag ritkán sugdosnak egymás fülébe, másrészt pedig ha nem kettő, hanem mondjuk nyolc egyed szeretne biztonságosan kommunikálni, bizony nagy lenne a sugdolózás. A súgás problémáját egyelőre hagyjuk el, tegyük fel, hogy van mód arra, hogy biztonságosan megállapodhassunk a közös kulcsban. A bábeli zűrzavart pedig úgy kerülhetjük el, ha kijelölünk egy központi egyedet, akinek mindenki megsúgja a saját kulcsát, és ha valaki kommunikálni szeretne, ugyancsak tőle kéri el a címzettnek szóló kulcsot. Az alábbi ábrán jól látszik, hogy mennyivel kisebb lesz a hangzavar.



A központosított kulcskezelés lényegesen kevesebb kapcsolatot igényel

Itt jön a képbe a görög mitológiából Kerberos *(pontosabban Cerberus)*, Hádes – az alvilág kapuját őrző – háromfejű kutyája. A modern Kerberos három feje a következő: az ügyfél, aki a szolgáltatást igénybe szeretné venni; a kiszolgáló, aki a szolgáltatást nyújtja; valamint a harmadik fél, aki segít abban, hogy az ügyfél és a kiszolgáló végül egymásra találjon. Ez a harmadik fél, mint a fenti ábrán is látszik, a Key Distribution Center *(KDC)*. A KDC egy megbízható, biztonságos kiszolgálón fut, ő kezeli a Kerberos tartomány *(realm)* biztonsági adatbázisát. A Kerberos tartomány egyébként egyenrangú a Windows 2000 tartománnyal (domain), tehát minden Windows 2000 tartományban található legalább egy, a helyi feladatokkal foglalkozó KDC. A KDC és az ügyfél egymás közti kommunikációjuk során az ügyfél úgynevezett hosszú távú kulcsát *(long-term key)* használják, amit előzőleg általában a felhasználó jelszavából állítanak elő. A KDC természetesen a tartomány minden felhasználójának és számítógépének ismeri a hosszú távú kulcsát.

Ha tehát egy ügyfél szeretné felvenni a kapcsolatot egy kiszolgálóval, előbb szól a KDC-nek, aki az ügyfél és a kiszolgáló jövőbeli kapcsolatára egy eldobható, úgynevezett rövid távú kulcs *(session key)* formájában áldását adja. Minden új ügyfél-kiszolgáló kapcsolathoz új rövid távú, más néven szakaszkulcs tartozik. A KDC ezt a rövid távú kulcsot az ügyfél hosszú távú kulcsával titkosítva elküldi az ügyfélnek; a kiszolgáló hosszú távú kulcsával titkosítva elküldi a kiszolgálónak, azok megfejtik az üzenetet és az új, közös szakaszkulcs segítségével máris indulhat a társalgás.

Azaz csak indulhatna, de mi történik, ha a KDC csomagja a kiszolgálóhoz valamilyen okból nem jut el? Az ügyfél hiába kopogtat a kiszolgálónál, süket fülekre talál. Ha az ügyfélhez nem jut el a kulcs, a kiszolgáló várna hiába, és számítástechnikai körökben – mint tudjuk – még a várakozás sincs ingyen.

Szakaszkulcsok *(Session Tickets)*

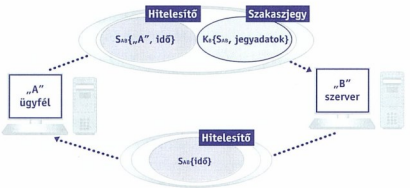
A való világban ez kicsit másként működik. A szakaszkulcsot a KDC csak az ügyfélnek küldi el, mégpedig úgy, hogy a visszaküldött szeretetcsomagba – az ügyfél hosszú távú kulcsával titkosított szakaszkulcs mellé – beletesz egy, a kiszolgáló számára titkosított adatstruktúrát, amely a szakaszkulcs mellett még más hasznos információkat is tartalmaz *(amit persze az ügyfél se visszafejteni, se módosítani nem képes)*.

A KDC működés közben (K_A az ügyfél, K_B a kiszolgáló hosszú távú kulcsa, S_{AB} a KDC által a jövendő kapcsolathoz generált szakaszkulcs)

Ezt a beágyazott adatcsomagot hívjuk szakaszjegynek *(session ticket)*, mégpedig azért, mert az ügyfél ezt a jegyet bemutatva tudja majd felvenni a kapcsolatot a kiszolgálóval. A KDC nem foglalkozik azzal, hogy az üzenetek valóban eljutnak-e a címzetthez, ugyanis semmi baj nem történik, ha a válasz elveszik – legfeljebb az ügyfél később újabb kulcsot kér.

Lássuk, mihez kezd az ügyfél a visszakapott csomaggal. Egyrészt, a saját hosszú távú kulcsával kicsomagolja a szakaszkulcsot, és biztonságos helyre teszi. A szakaszjegyet a

kulcs mellé helyezi. Amikor a konkrét kapcsolatfelvételre sor kerül, a szakaszkulccsal előállít egy hitelesítőt *(authenticatort)*, majd ezt és a címzetthez érvényes szakaszjegyet elküldi a kiszolgálónak. A kiszolgáló a csomagból kiveszi a neki szóló szakaszjegyet, a saját hosszú távú kulcsával megfejti azt, kiveszi belőle a szakaszkulcsot és jegy egyéb adatait. Ha az adatok alapján úgy dönt, hogy szóba áll az ügyféllel, a szakaszkulcs segítségével kicsomagolja a kapott hitelesítőt, ellenőrzi az időt, törli a hitelesítő egy részét, a maradékot a szakaszkulccsal visszacsomagolja, és visszaküldi az ügyfélnek. Az ügyfél a visszakapott hitelesítőt visszafejti, ellenőrzi a tartalmát és ha minden rendben ment, a szakaszkulcs segítségével már indulhat a titkosított kommunikáció. Ráadásul a hitelesítő visszaküldésével az ügyfél és a kiszolgáló azonosították egymást.



Kerberos azonosítás (S_{AB} a KDC által a kapcsolathoz generált szakaszkulcs, K_B a kiszolgáló hosszú távú kulcsa)

A megoldás másik előnye, hogy az egyes kiszolgálóra szóló jegyek újra felhasználhatók, tehát az ügyfélnek nem kell minden kapcsolat előtt a KDC-hez fordulnia. A biztonság érdekében azonban minden jegy csak egy adott ideig érvényes *(ez az idő alapértelmezésben általában 8 óra)*, melyet a jegy adatmezőjében tárolunk. Az érvényességi idő lejárta után az ügyfél kénytelen lesz megújítani a jegyet. Ha az ügyfél kilép a rendszerből, a számítógépén tárolt jegyek elvesznek, tehát belépéskor mindenképpen szükség lesz új szakaszjegyek kérésére.

Jegy a jegyekhez (Ticket-Granting Ticket – TGT)

A felhasználók hosszú távú kulcsát a jelszóból állítják elő, mégpedig egy egyirányú titkosító, úgynevezett „hash" algoritmus segítségével. A szabvány szerint minden Kerberos implementációnak ismernie kell a DES-CBC-MD5 kódolást *(ez egy kriptográfiai módszer a hash előállítására)*.

A KDC adatbázisában megtalálható az összes tartományi felhasználó és számítógép hosszú távú kulcsa. Amikor Alice a munkaállomásán bejelentkezik, az általa beírt jelszóból előáll egy kulcs, amit a bejelentkezéshez használni fog. A KDC ugyanezt a kulcsot a saját adatbázisából veszi, így lesz képes azonosítani a felhasználót.

A hosszú távú kulcsot egy munkamenetben mindössze egyszer állítják elő, mégpedig akkor, amikor a felhasználó először bejelentkezik a rendszerbe. A bejelentkezés után a felhasználó kap egy speciális szakaszjegyet, amit azután mindennemű kapcsolat felépítéséhez és kezdeményezéséhez használhat *(és használnia is kell)*.

Mint mondtuk, a KDC az, aki az ügyfél és a kiszolgálók közötti azonosítást lehetővé teszi. Az ügyfél tehát, mielőtt a kiszolgálóhoz menne, a KDC segítségére szorul *(szakaszjegyet kell kérnie tőle a kiszolgálóhoz)*, ezért a legelső szakaszjegy, amire az ügyfélnek szüksége van, magához a KDC-hez szól, hogy ezentúl ne kelljen a hosszú távú kulcsot használva azonosítania magát.

Ez a speciális szakaszjegy a Ticket-Granting Ticket *(TGT)*, azaz a jegy a jegyekhez. Az ügyfél a bejelentkezés után e jegy bemutatásával kérhet további szakaszjegyeket a KDC-től. A TGT egyébként teljesen hasonló az általános szakaszjegyekhez, hiszen tulajdonképpen ez is csak egy jegy egy szolgáltatáshoz *(mégpedig a KDC jegyárusához)*.

Amikor tehát a felhasználó azonosítása megtörtént, a KDC a további jegyek kiadásához generál Alice számára egy szakaszkulcsot *(ez a bejelentkezési szakaszkulcs, logon session key)*. Ezt a szakaszkulcsot a felhasználóra vonatkozó adatokkal együtt becsomagolja és a saját hosszú távú kulcsával titkosítja. Ez a titkosított adatcsomag maga a TGT. Látható, hogy a felhasználó a TGT-t se elolvasni, se módosítani nem tudja, hiszen azt a KDC titkosította, önmagának. A KDC ezután a szakaszkulcsot betitkosítja a felhasználó hosszú távú kulcsával is *(hiszen a kulcshoz azért neki is hozzá kell férnie)*, és némi, a TGT-re vonatkozó információ mellett az egész csomagot visszaküldi a felhasználónak.

Az ügyfél tehát válaszul kap egy csomagot, ami két részből áll: az első rész a felhasználó saját hosszú távú kulcsával titkosított információ és a szakaszkulcs egy példánya, amit a KDC-vel való további kapcsolatok titkosítására majd felhasználhat. A második rész a KDC kulcsával titkosított TGT, amivel pedig azonosíthatja magát. Az ügyfél a dekódolt szakaszkulcsot, az adatokat, és magát a TGT-t biztos helyre menti.

A felhasználó jelszavát és hosszú távú kulcsát ekkor el is felejthetjük, hiszen a KDC felé a TGT, más kiszolgálók felé pedig a majdan KDC-től kapott szakaszjegyek segítségével jöhet létre a kapcsolat.

Hitelesítés tartományok között

A KDC feladata kettős: egyrészt, a hitelesítő szolgáltatás *(Authentication Service)* TGT-ket, másrészt a jegykiszolgáló *(Ticket-Granting Service)* a már TGT-vel jelentkező ügyfelek részére szakaszjegyeket gyárt. Ez a kettősség teszi lehetővé, hogy a Kerberos azonosítás tartományi határokat átlépve is működhessen. Lehetséges, hogy az ügyfél az egyik tartomány KDC-jétől kapott TGT segítségével egy másik tartomány KDC-jétől kérjen szakaszjegyet egy szolgáltatáshoz. A KDC szakaszjegyeket mindig csak a saját tartományába tartozó szolgáltatásokhoz adhat, hiszen csak a saját tartományában található felhasználók és számítógépek hosszú távú kulcsával rendelkezik.

A dolog megértéséhez kezdjük a legegyszerűbb esettel: legyen két tartományunk, Pest és Buda. Ha azt szeretnénk, hogy a két tartomány között létrejöhessen Kerberos hitelesítés, a tartományok között is meg kell osztanunk egy közös titkot, egy speciális kulcsot. Ez a tartományok közötti kulcs, az inter-domain key.

A Windows 2000 a tartományok közvetlen megbízotti viszonyának kialakításakor ezt a tartományok közötti Kerberos kulcsot is létrehozza. A Windows 2000 tartományok között megbízotti viszony automatikusan alakul ki, amikor egy tartományi fában a meglévő tartomány*(ok)* mellé újat hozunk létre – természetesen csak a „szomszédos" tartományok között, hiszen a megbízotti viszonyok átlátszósága, tranzitív jellege

miatt – néhány kivételtől eltekintve – a további közvetlen megbízotti viszonyok *(értsd: mindenki mindenkivel)* létrehozása nem szükséges. Az egymással közvetlen megbízotti viszonyban álló *(szomszédos)* Windows 2000 tartományok tehát rendelkeznek közös tartományok közötti kulccsal.

Gondoljunk csak bele, mit jelent ez a plussz egy kulcs: mindössze annyit, hogy egy tartomány KDC-je a tartományi fiókokon kívül még egy valakinek a hosszú távú kulcsát is ismeri: ez pedig a másik tartomány KDC-je. Magyarán, Pest KDC-jétől ugyanúgy kérhetünk szakaszjegyet Buda KDC-jéhez, mint ahogy bármilyen más szolgáltatáshoz.

Főhősünk, Fű Benő Pesten ül számítógépe előtt, és szeretné elérni Gipsz Jakab számítógépének CD-meghajtóját, aki a Rózsadombon *(Budán)* lakik. Benő Pest KDC-hez fordul a kérésével. Pest KDC felismeri, hogy a kérés Buda KDC-re tartozik, ezért Benő tőle csak egy hivatkozó jegyet (referral ticket) kap, aminek segítségével Buda KDC-nél érdeklődhet. *(Ezt a jegyet a tartományok közötti kulccsal titkosították.)* Benő a jeggyel átússza a Dunát, és jelentkezik Buda KDC-nél, aki az általa is ismert tartományok közötti kulccsal ellenőrzi a jegy helyességét, majd egy hagyományos, immár a rózsadombi Gipsz Jakab szolgáltatásához szóló szakaszjegyet ad Benőnek. Benő ezzel a jeggyel azután bátran jelentkezhet a Rózsadombon, hiszen az pontosan olyan, mintha azt más, tőzsgyökeres budai felhasználó kapta volna.

Ha kettőnél több tartományunk van, bonyolódik a helyzet. Általában nincs értelme annak, hogy minden tartományt minden tartománnyal megbízotti viszonyba hozzunk, hiszen a Windows 2000 tartományok közötti megbízotti viszony tranzitív, átlátszó. Ha az ügyfél és az általa igényelt szolgáltatás olyan két különböző tartományban található, amelyek nem szomszédosak *(azaz nem állnak egymással közvetlen megbízotti viszonyban)*, akkor az ügyfél az előző esethez hasonló módon „végigutazhatja" a fát, míg eléri a céltartományt.

Lássunk egy példát: adott három tartomány, Debrecen, Pécs és Budapest. A fa csúcsán Budapest található, azaz Debrecen és Pécs egymással nem áll közvetlen megbízotti viszonyban *(Debrecennek és Pécsnek tehát nincs közös tartományok közötti kulcsa)*.

Fű Benő pécsi előadása alkalmával szeretné elérni az azóta Debrecenbe költöző, Gipsz Jakab számítógépét:

1. Benő Pécs KDC-től jegyet kér a debreceni Gipsz Jakab számítógépéhez. Pécs KDC erre csak egy, a Budapest KDC-hez szóló, a Pécs-Budapest közös kulccsal titkosított hivatkozási jeggyel válaszol *(mással nem is tudna, mert Debrecennel nincs közös kulcsa)*.
2. Benő a jeggyel Budapest KDC-hez fordul, aki ismét csak egy hivatkozási jegyet tud adni: a jegy ezúttal Debrecen KDC-hez szól *(a Budapest-Debrecen kulccsal titkosítva)*.
3. Benő továbbutazik, és az utoljára kapott jegyet bemutatja Debrecen KDC-nek, aki mosolyogva átnyújtja Benőnek a Gipsz Jakabhoz érvényes szakaszjegyet.
4. A szakaszjeggyel Benő a következő nyolc órában már vigan és közvetlenül elérheti Gipsz Jakab számítógépét.

Alprotokollok

A Kerberos protokoll valójában három alprotokollból áll:

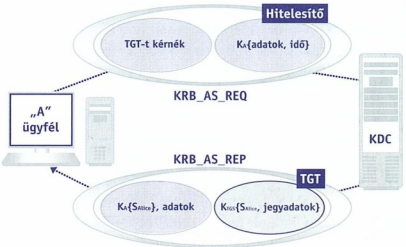
1. Authentication Service Exchange *(AS)*: hitelesítő szolgáltatás, ez azonosítja a felhasználót és állítja elő számára a TGT-t
2. Ticket-Granting Service Exchange *(TGS)*: a jegykiszolgáló szolgáltatás, ami az érvényes TGT-ket bemutató ügyfelek számára más szolgáltatásokhoz érvényes szakaszjegyeket készít
3. Client/Server Exchange *(CS)*: az ügyfél és a kiszolgáló közötti azonosítás, a TGS-től kapott szakaszjegy alapján

AS Exchange

Az első két protokoll a kapcsolatot kezdeményező ügyfél és a KDC, az utolsó pedig az ügyfél és az általa áhított szolgáltatást nyújtó kiszolgáló között zajlik. A protokollok megértéséhez vegyük ismét Alice-t és Bob-ot. Alice bejelentkezik a hálózatba és szeretne hozzáférni Bob egy szolgáltatásához. Mindegyik protokoll kérésből *(KRB_xxx_REQ)* és válaszból *(KRB_xxx_REP)* áll.

Az AS kérés és válasz tulajdonképpen Alice bejelentkezése. A kérésben Alice elküldi a KDC-nek a saját adatait, az igényelt szolgáltatás leírását *(TGT-t szeretne)*, valamint a jelszóból képzett hosszú távú kulccsal (K_A) titkosított hitelesítőt. A KDC megkapja a csomagot, a titkosítatlan adatokból kiderül számára, hogy Alice próbál meg bejelentkezni. Az adatbázisából megkeresi Alice hosszú távú kulcsát (K_A) és megpróbálja dekódolni a hitelesítőt, ezzel azonosítja Alice-t. Ha sikerült, és a benne található adatok (például az idő) érvényesek, a KDC generál egy bejelentkezési szakaszkulcsot (S_{Alice}), és előkészíti a választ:

A KDC a szakaszkulcsot (S_{Alice}) és néhány egyéb hasznos adatot saját (K_{TGS}) kulcsával betitkosítja – ez a TGT. A szakaszkulcsot Alice hosszú távú kulcsával (K_A) is betitkosítja, hogy Alice is hozzáférjen a szakaszkulcshoz. Az egészhez hozzáragaszt még némi nyílt információt a TGT-vel kapcsolatban (hiszen a TGT-ben tárolt adatokhoz Alice nem jut hozzá), és válaszként elküldi Alice-nak.

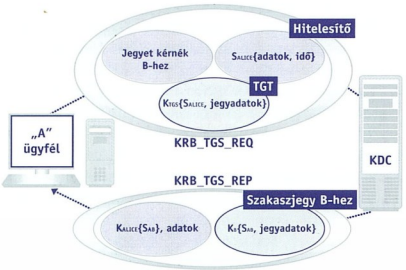


Az AS (Authentication Service) protokoll (S_{Alice} a KDC által a TGT-hez generált szakaszkulcs, K_A Alice; K_{TGS} a KDC hosszú távú kulcsa)

Amikor Alice megkapja a pozitív választ, a saját hosszú távú kulcsával (K_A) dekódolja a szakaszkulcsot (S_{Alice}), majd az adatokkal a komplett TGT-vel együtt elmenti.

TGS Exchange

Az AS protokoll segítségével Alice jogot formált arra, hogy jegyet vásárolhasson a tartomány szolgáltatásaihoz. Ezután a szakaszjegyek megváltása következik:

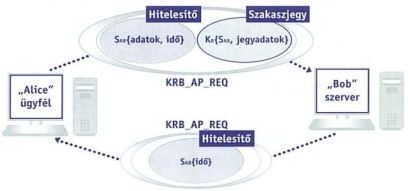


A TGS (Ticket-Granting Service) protokoll (S_{Alice} a KDC által a TGT-hez generált szakaszkulcs, S_{AB} Alice és Bob kapcsolatához generált szakaszkulcs, K_B Bob; K_{TGS} a KDC hosszú távú kulcsa)

Alice a TGS kérésben meghatározza a kívánt szolgáltatást, amihez a jegyet kéri. Alice a bejelentkezési szakaszkulccsal *(S_{Alice})* előállít egy hitelesítőt, és az egészhez mellékeli az elmentett TGT-t. A KDC a saját titkos kulcsával *(K_{TGS})* dekódolja a TGT-t és így hozzájut Alice bejelentkezési szakaszkulcsához *(S_{Alice})*. Ezzel a kulccsal már ki tudja nyitni a hitelesítőt, és a szokásos módszerrel ellenőrzi a kérés hitelességét. Ha az ellenőrzés során sikerrel járt, az adatbázisában megkeresi a kívánt szolgáltatás *(Bob)* hosszú távú kulcsát *(K_B)* és előállít egy egyedi szakaszkulcsot, amit majd Alice és Bob fog használni a későbbi párbeszédeik során *(S_{AB})*.
Az új szakaszkulcsot *(S_{AB})* titkosítja az Alice által küldött szakaszkulccsal *(S_{Alice})*, és a titkosított kulcshoz hozzácsap még némi információt a következő szakaszjegyről. A szakaszjegy az új szakaszkulcsból *(S_{AB})* és számos, Bob számára hasznos információból áll *(pl. Alice jogai, csoporttagsága, stb.)* – mindez természetesen Bob hosszú távú kulcsával *(K_B)* titkosítva, hogy csak ő tudja elolvasni.
Amikor Alice megkapja a választ, a bejelentkezési szakaszkulcsával *(S_{Alice})* kibontja a Bob felé használható új szakaszkulcsot *(S_{AB})*. A szakaszkulcsot, a szakaszjegyet, és a rá vonatkozó adatokat ugyancsak elmenti.

CS Exchange

Alice ezután már felveheti a közvetlen kapcsolatot Bob-bal. Alice először is, a Bob-féle szakaszkulccsal *(S_{AB})* előállít egy hitelesítőt, majd azt a Bob-hoz érvényes szakaszjeggyel együtt elküldi Bob-nak.
Bob megkapja a csomagot, saját hosszútávú kulcsával *(KB)* kinyitja a szakaszjegyet, és megszerzi belőle a szakaszkulcsot *(S_{AB})*, valamint hasznos információkhoz jut Alice-ről. A szakaszkulcs segítségével Bob már képes kibontani és ellenőrizni a hitelesítőt is.



A CS (Client-Server) protokoll (S_{AB} Alice és Bob kapcsolatához generált szakaszkulcs, K_B Bob hosszú távú kulcsa)

Ha a kölcsönös hitelesítést elvárjuk, akkor utolsó lépésben Bob-nak is bizonyítania kell Alice felé, hogy ő valóban Bob. Ezt a klasszikus módon teszi: fogja az Alice által kapott hitelesítőt, megváltoztatja a tartalmát *(például kiveszi az adatokat és csak az időt hagyja benne)*, majd a szakaszjeggyel *(S_{AB})* titkosítva visszaküldi.
Amikor Alice megkapja ezt az üzenetet, biztos lehet benne, hogy azt Bob küldte, hiszen ki tudta nyitni – és vissza tudta csomagolni – a szakaszkulccsal titkosított hitelesítőt. A szakaszkulcshoz viszont csakis a szakaszjegyből juthatott hozzá, ami viszont Bob saját hosszú távú kulcsával volt titkosítva. Konklúzió: aki a hitelesítőt így vissza tudta küldeni, az ismeri Bob hosszú távú kulcsát, következésképpen vagy Bobbal, vagy a KDC-vel állunk szemben *(de a KDC nem tenne ilyen gonoszságot ;-))*.

Fülöp Miklós
mick@netacademia.net

Irodalom

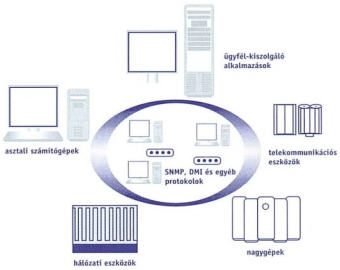
Miller, S., Neuman, C., Schiller, J., and J. Saltzer, „Section E.2.1: Kerberos Authentication and Authorization System," MIT Project Athena, Cambridge, MA, December 1987.
Kohl, J., and C. Neuman, „The Kerberos Network Authentication Service (V5)," RFC 1510, Sept. 1993.
Tung, B., Neuman, C., Wray, J., Medvinsky, A., Hur, M., and J. Trostle, „Public Key Cryptography for Initial Authentication in Kerberos," draft-ietf-cat-kerberos-pk-init.

Windows Management Instrumentation

A Microsoft és más, vele együttműködő cégek több olyan célkitűzést fogalmaztak meg, melyek elősegítik a nagykiterjedésű PC-hálózatok tulajdonlási költségének *(TCO)* csökkentését. A TCO messze meghaladja egy elosztott PC-hálózat hardver- és szoftvereszközeinek beszerzési árát. A TCO-ba beletartoznak az üzembehelyezés, a karbantartás, a rendszerfelügyelet, a képzés, a telefonos és helybeni terméktámogatás költségei, valamint a hardver- és szoftverfrissítésekkel kapcsolatos kiadások. E költségcsökkentő kezdeményezések közül az egyik legfontosabb a WBEM *(Web-Based Enterprise Management – nagykiterjedésű rendszerek Web-alapú felügyelete)*, amely a felügyeleti infrastruktúrára vonatkozó szabványokat fogalmaz meg, és lehetővé teszi többféle hardver- és szoftverfelügyeleti rendszerből származó információk összegzését. Ez az információ aztán felhasználható a rendszerfelügyeleti alkalmazásokban. A WBEM a CIM *(Common Information Model – közös információ-modell)* sémán alapul, amely a DMTF *(Distributed Management Task Force – elosztott felügyelet munkacsoport)* által létrehozott szabvány. A Microsoft Windows Management Instrumentation *(WMI, Windows felügyeleti eszközkészlet)* WBEM-kompatibilis, és a Windows 2000 operációs rendszer beállításairól, állapotáról és működéséről összefüggő, gazdag információkkal szolgáló modellt biztosít. A Windows 2000 más rendszerfelügyeleti szolgáltatásaival együtt használva a WMI leegyszerűsítheti az integrált felügyeleti alkalmazások fejlesztését, lehetővé téve a gyártóknak skálázható, hatékony felügyeleti rendszerek készítését.
Cikkünkben először a WBEM áttekintésével és kialakulásával, majd a WBEM szabványos összetevőivel, és a WBEM-kompatibilis Windows felügyeleti architektúrával foglalkozunk. Végül a WMI funkciók és más Windows felügyeleti szolgáltatások együttműködésére mutatunk példákat.

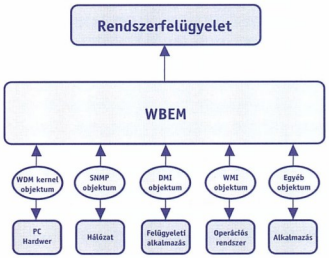
A WBEM áttekintése

A WBEM a nagykiterjedésű hálózatok felügyeleti információinak elérését és megosztását szolgáló szabványos, de nem szabadalmazott *(vagyis nyílt)* megoldások kifejlesztését célzó kezdeményezés. A WBEM olyan megoldásokat fog eredményezni, melyek lehetővé teszik a különböző forrásokból származó felügyeleti adatok összegyűjtését, összekapcsolását és összesítését, és ezzel gazdagabb és pontosabb képet adhatnak a nagyvállalati környezetről. A WBEM kezdeményezés célkitűzései között szerepel a nagykiterjedésű hálózatok állapotáról szóló, és azok felügyeleti adatainak összegyűjtésével kapcsolatos problémák megoldása. Ezek a hálózatok számos hardvergyártó eszközeit, többféle protokollt és operációs rendszert, és rengeteg elosztott alkalmazást tartalmazhatnak.



Nagykiterjedésű hálózatban megtalálható tipikus protokollok és eszközök.

A nagykiterjedésű hálózatok felügyelete általában különféle eszköztípusok változatos protokolljaihoz és csatolófelületeihez kötődik *(például a Simple Network Management Protocol (SNMP) a hálózatok felügyeletére, a Desktop Management Interface (DMI) az asztali gépek felügyeletére szolgál)*. A WBEM szerint a nagykiterjedésű hálózatok felügyeletéhez olyan eszközökre van szükség, melyek együttműködnek annak érdekében, hogy a felügyeleti információk összegyűjtésére egy egyszerű, közös modellt alkossanak. A WBEM biztosítja ezt a közös modellt és adatforrást, melyek bővíthetők, hogy képesek legyenek együttműködni a meglévő hálózati összetevőkkel, eszközökkel és protokollokkal.



A WBEM architektúra.

A WBEM nem böngészőalapú eszköz, nem felhasználói felület, nem adattároló, nem hálózatfelügyeleti protokoll, nem objektummodell, és nem is regisztrációs adatbázist, címtárat, vagy fájlrendszert helyettesítő eszköz. A WBEM egy kezdeményezés, mely a nagykiterjedésű hálózatok felügyeletéről szóló szabványgyűjtemény létrehozását tűzi ki célul. A felügyeleti szabványok:

- Leírják a felügyelt objektumokról szóló információk eléréséhez szükséges struktúrákat és konvenciókat.
- Támogatják az információk központosítását, így a külön-

féle ügyfelek és felügyeleti eszközök képesek adatokat szolgáltatni, visszakeresni és elemezni.
- Támogatják, hogy hitelesítés után bárhonnan a hálózatról hozzá lehessen férni a felügyelt objektumokhoz, így azok elemezhetők és manipulálhatók.

A WBEM kialakulása

A WBEM ajánlást 1996-ban a Microsoft, a Compaq Computer, a BMC Software, a Cisco Systems és az Intel vezetésével több cég hozta létre. Az elképzelés olyan nyílt felügyeleti környezet kialakítása volt, melyben a lehető legtöbb meglévő technológia és szabvány felhasználásával az összes felügyeleti rendszer és alkalmazás elérheti, szabályozhatja és megoszthatja egymással és az eszközt felügyelő ügynökkel *(agent)* a felügyeleti információkat. Sok szempontból ez a meghatározott cél a World Wide Web tapasztalatait vette alapul, ahol az Interneten levő eszközök az információk forrásaiként és használóiként anélkül szerepelhetnek, hogy tudnák egymásról, hogy a másik milyen környezetben működik. A nyílt felügyeleti rendszer létrehozásában a webes technológiák és a megszokott felügyeleti eszközök együttes használatának lehetősége miatt lett a kezdeményezés neve Web-Based Enterprise Management *(WBEM)*.

Az alapítótagok a Distributed Management Task Force-szal *(DMTF)* együttműködve kifejlesztették a bármely felügyeleti eszköz *(például SNMP, DMI)* definiálását és elérését szolgáló környezetfüggetlen eszközkészlet részletes leírásának első verzióját. Ennek fő eleme egy adatleíró eljárás volt, amely később – immár a DMTF szabványaként - Common Information Model *(CIM)* néven vált ismertté.

Az eredetileg HyperMedia Management Schema *(HMMS)* projekt néven futó CIM specifikáció megadta a modellezési nyelvet, a névadási és hozzárendelési eljárásokat, melyeket az adatszolgáltatóktól és más felügyeleti modellekből származó információk összegyűjtésére és átvitelére használtak. A CIM séma szolgáltatja az érvényes modell-leírásokat és az információs keretrendszert, megad egy tulajdonságokkal és asszociációkkal rendelkező osztálykészletet, ezzel lehetővé teszi a felügyelt környezet információinak rendezését.

A DMTF tulajdonát képezi a CIM specifikációja és a CIM séma, melyeket a hálózatfelügyeleti adatok elérésének és megosztásának iparági szintű szabványaként határoztak meg.

Az 1996-tól 1998-ig tartó időszakban a Microsoft kidolgozta a WBEM Windows-os megvalósítását. Ebbe a munkába beletartozott a WBEM szoftverfejlesztő készlet *(SDK)*, a különböző CIM összetevők és CIM-kompatibilis adatszolgáltatók kifejlesztése is.

1998. júniusában a DMTF bejelentette, hogy átvette a WBEM kezdeményezést az alapító vállalatoktól. Jelenleg a DMTF a WBEM kezdeményezés megvalósítására irányuló erőfeszítések központja, és szervezeti keretet biztosít a WBEM-kompatibilis eljárások és szabványok fejlesztésében való széleskörű iparági részvételhez. A WBEM-alapú szabványok egyedi megvalósítása *(például a Microsoft Windows Management Instrumentation SDK (volt WBEM SDK))* továbbra is az őket kifejlesztő gyártók feladata. A WBEM kezdeményezés átvételekor a DMTF beleegyezett, hogy a meglevő WBEM eljárásokat, például a Microsoft CIM megvalósítását referencia-példaként használja. A DMTF beleegyezett abba is, hogy fenntartja a WBEM ígérte platformfüggetlenséget, és elzárkózik minden megvalósítási függés *(például adott programnyelv használata)* megadásától bármely követelményben.

A szabványos WBEM összetevők

Jelenleg két fő részből áll a WBEM (további szabványok várhatók, például az XML használata a CIM objektumok platformfüggetlen megosztására):
- A CIM specifikáció, mely meghatározza a WBEM megvalósítás követelményeit.
- A CIM séma, mely meghatározza az adattároló tartalmát.

Alapvetően a WBEM a CIM megvalósítását tűzi ki célul. A CIM a felügyelt objektumok objektumorientált sémája. Ezek a felügyelt objektumok a rendszererőforrások megjelenítési formái, és a séma minden általuk szolgáltatott adathoz egyszerű adatleíró eljárást nyújt. A WBEM meghatározza, hogyan kell az adatokat megjeleníteni, és tartalmaz egy folyamatszabványt, ami meghatározza az összetevők egymásra hatását.

A CIM séma a központi *(core)* modellből, és a közös *(common)* modellekből áll. A központi modell az összes felügyeleti tartományra érvényes, a közös modellek különböző felügyeleti tartományok *(számítógépek, hálózat, adatbázis, alkalmazás, egyéb eszközök)* közös információit adja meg. Maga a séma bővíthető, a bővítő-sémák a közös séma technológiafüggő kiegészítései.

A Microsoft WBEM megvalósítása—a WMI

A WMI *(Microsoft Windows Management Instrumentation – Microsoft Windows felügyeleti eszközkészlet)* WBEM-kompatibilis, és CIM alapokon támogatja az egységes rendszer- és alkalmazásfelügyeletet. A WMI a Microsoft Windows felügyeleti szolgáltatások egyik fő összetevője. A Windows felügyeleti szolgáltatások közé tartoznak az Active Directory helymeghatározó és házirendszolgáltatásai, a Microsoft Management Console *(MMC)* megjelenítési szolgáltatásai, és a Windows Scripting Host *(WSH)* automatizálási képességei is.

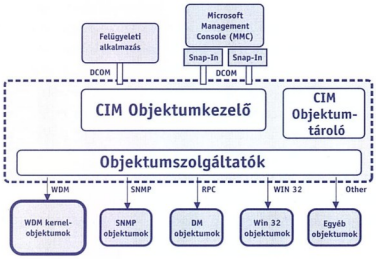
A WMI segítségével csökkenthetők a rendszerek összetevőinek felügyeleti költségei, és a karbantartásukra fordított idő. A WMI a következőket biztosítja:
- Gazdag és összefüggő információmodellt a Windows 98 és Windows 2000 működéséről, állapotáról és beállításairól *(a Windows NT 4.0-hoz és a Windows 95-höz is letölthetők a WMI fő összetevői)*.
- COM API-t, amely egyetlen felületen elérhetővé teszi az összes felügyeleti információt.
- Együttműködést más Windows 2000 felügyeleti szolgáltatásokkal. Ez az eszközök gyártóinak egyszerűbbé teszi az integrált felügyeleti alkalmazások készítését.
- Rugalmas architektúrát, mely biztosítja a gyártóknak az információmodell kiterjesztését új eszközökhöz, alkalmazásokhoz, vagy más fejlesztésekhez írt kódmodulokkal *(WMI szolgáltatókkal)*.
- Részletes eseményarchitektúrát, mely lehetővé teszi a felügyeleti információk változásainak azonosítását és összesítését, más felügyeleti információkkal való összehasonlítását és összekapcsolását, valamint a helyi vagy távoli felügyeleti alkalmazásokhoz való továbbítását.
- Gazdag lekérdezőnyelvet, mely biztosítja az információs modell részletes lekérdezését.
- Scriptelhető API-t, melynek segítségével a felügyeleti

alkalmazások fejlesztéséhez használható a Microsoft Visual Basic vagy a Windows Scripting Host *(WSH)*.

A távoli események kezelésének és az információmodell gazdag lekérdezőnyelvének együttes használata például biztosíthatja az összetett felügyeleti problémák megoldását. Az a lehetőség, hogy ezek a megoldások Visual Basic vagy WSH parancsfáljok használatával elkészíthetők, új távlatokat nyitnak a Windows NT felügyeletéhez.

A WMI architektúra

A WBEM három részből álló architektúrával biztosítja a felügyeleti adatok összegyűjtését és elosztását. A WMI-ben ez az architektúra a szabványos objektum-definíciókból *(CIM-kompatibilis objektum-tároló)*, a felügyeleti adatok elérésének és elosztásának szabványos protokolljából *(COM/DCOM; de más protokollok használata is lehetséges)*, és egy vagy több Win32 WMI adatszolgáltatóként működő dll-ből áll. A WMI szolgáltató eszközkészlet adatokat biztosít a CIM séma részeinek.



A WMI architektúra

A WMI funkcionalitást biztosító folyamat *(process)* a WinMgmt.exe. Ez a futtatható fájl biztosítja a WMI-t alkotó CIM objektumtároló *(object repository)*, CIM objektumkezelő *(Object Manager)*, és API-k működését.

A CIM objektumkezelő

A CIM objektumkezelő a Microsoft WBEM megvalósításának egyik fő része. A WBEM fő célja az adatok egységes megjelenítése, és azok objektum-orientált módon való beágyazása a CIM objektum-tárolóba. A CIM objektumkezelő a tárolóban tárolt, felügyelt objektumokhoz gyűjtési és kezelési felületet tartalmaz. A CIM objektumkezelő nem éri el közvetlenül az információkat, hanem a WMI szolgáltatók gyűjtik össze az erőforrástól *(felügyelt objektumtól)* azokat, majd a WMI API-n keresztül a felügyeleti alkalmazások számára elérhetővé teszik őket.

A WMI szolgáltatók

A WMI szolgáltatók *(WMI provider)* közvetítőként működnek a CIM objektumkezelő, és egy vagy több felügyelt objektum között. Amikor a CIM objektumkezelőtől a felügyeleti alkalmazás egy, a tárolóban nem elérhető információt, vagy általa nem támogatott eseményekről szóló értesítéseket kér, továbbítja a kérést a szolgáltatóhoz. Ezután a szolgáltató biztosítja a kért információt, vagy az eseményről szóló értesítést.

A WMI az alábbi szolgáltatókat tartalmazza:

- Win32 szolgáltató
- WDM szolgáltató
- Eseménynapló szolgáltató
- Rendszerleíró adatbázis *(registry)* szolgáltató
- Teljesítményfigyelő *(Performance Counter)* szolgáltató
- Active Directory szolgáltató
- Windows Installer szolgáltató
- SNMP szolgáltató

Más gyártók a WMI SDK alkalmazásával készíthetnek egyedi szolgáltatókat, melyek együtt tudnak működni a saját környezetük egyedi felügyelt objektumaival.

A WMI eljárásai nem a meglévő felügyeleti szabványok *(például SNMP, DMI, CMIP)* lecserélését szolgálják, és alkalmazásuk nem jelenti a szabadalmazott vagy platformfüggő keretrendszerek *(például NDS)* kiirtását. A WMI tulajdonképpen kiegészíti ezeket azzal, hogy integrációs felületet biztosít, melyen keresztül bármely forrásból származó adat elérhető.

Az események kezelése

Az eseményekre való figyelmeztetés fontos lehetőség a WMI-ben, mert ez biztosítja, hogy az összetevők felderítsék a hardverrel vagy szoftverrel kapcsolatos eseményeket és/vagy hibákat. Ezután az esemény keresztülhalad a WMI architektúrán a megfelelő felügyeleti összetevőhöz, mely végrehajthatja a helyesbítő eljárást.

A WMI-ben az esemény egy történés, amely vagy előre megadott, valós világban létrejövő feltételeknek *(kívülről jövő esemény)*, vagy a CIM tárolóban bekövetkezett változásoknak *(belülről jövő esemény)* felel meg. Egy esemény megtörténte után egy eseményszolgáltató értesíti a CIM objektumkezelőt, mely továbbítja az eseményt egy vagy több bejegyzett címzettnek, azaz az eseményfigyelőknek *(event consumer)*. Az eseményfigyelők bejegyezhetik a CIM objektum-kezelőben, hogy adott típusú értesítéseket kapjanak, az eseményszolgáltatók pedig bejegyezhetők, hogy bizonyosfajta értesítéseket küldjenek. Az eseményfigyelők az eseményszolgáltatóktól független működése érdekében a CIM objektumkezelő közvetítőként szerepel, megfelelteti egymásnak a bejegyzett figyelőket és a megadott szolgáltatókat, és továbbítja a megfelelő eseményeket.

Az eseményfigyelők anélkül is bejegyezhetik magukat, hogy tudnák, mi szolgáltatja az eseményeket és értesítéseket. A bejegyzéshez ezek a figyelők egy szűrőt adnak meg, mely a WQL *(WMI Query Language – WMI lekérdezőnyelv)* használatával készül el. Ez írja le azokat a feltételeket, melyek létrejöttekor a figyelő értesítést akar kapni.

A WMI lekérdezőnyelv

A WQL az SQL olyan „nyelvjárása", mely eseményekről szóló értesítéseket, és más WBEM kompatibilis funkciókat támogató bővítményeket tartalmaz. Amikor a figyelők bejegyzik magukat az eseményekről szóló értesítések címzettjeként, tulajdonképpen egy lekérdezést adnak meg, ami leírja az esemény típusát és a feltételeket, melyek teljesülésekor értesítést kapnak. A WMI SDK-ban definiált WQL használható értesítésszűrők készítésére is.

WBEM-kompatibilis scriptprogramozás

A WMI Script felületei CIM objektumkezelővel együttműködő parancsfájl és Visual Basic alkalmazások fejlesztésére használhatók. A WMI az alábbi nyelvek támogatását biztosítja:

- Microsoft Visual Basic
- Visual Basic for Applications
- Visual Basic, Scripting Edition *(VBScript)*
- Microsoft JScript
- Perl

A CIM objektumkezelő scriptelési felületei annyiban különböznek a COM felületektől, hogy ezeket eleve a Visual Basic, Visual Basic for Applications, Visual Basic Scripting Edition (VBScript) és más parancsnyelvekkel való együttműködésre tervezték. A Microsoft WBEM-kompatibilis scriptelés az alábbi előnyöket biztosítja:

- Adatközpontú megközelítést alkalmaz, a CIM-et. A CIM modellt biztosít az eltérő információk kezeléséhez, a script API pedig elszigeteli egymástól az alkalmazásokat, és a különböző adatforrások összetett rendszerét.
- Biztosítja a rendszer-, hálózat-, és alkalmazásinformációk széleskörű lefedését. A Microsoft megvalósítás a Win32, SNMP, regisztrációs adatbázis, Windows Driver Model *(WDM)*, Performance Monitor, eseménynapló, és ADSI szolgáltatókat teszi elérhetővé. Más gyártók, például az Intel, a Compaq, a Hewlett-Packard és a BMC Software szintén fognak készíteni termékeikhez szolgáltatókat, hogy lehetővé tegyék gyártófüggő eszközkészletek használatát *(ez így lesz a Microsoft Systems Management Serverrel is)*. A Microsoft egyéb szolgáltatói jelenleg fejlesztés alatt állnak.
- A szolgáltató eszközkészletek könnyen bővíthetők. Az eszközök, a minták és a bővíthető szolgáltató architektúra teljes leírása megtalálható a Microsoft WMI SDK-ban. Emellett a szolgáltatók fejlesztése széleskörű iparági támogatást élvez.
- Egyszerű új parancsfájlokat írni. A Microsoft WBEM-kompatibilis API-t egyszerű használni, a séma pedig böngészhető és bővíthető.

WDM Szolgáltató

A Microsoft a WDM szolgáltatót az operációs rendszer kernel kezelésére fejlesztette ki. A WDM eszközkészlet a Windows Driver Model *(WDM)* architektúra része, de széleskörű felhasználhatósága miatt másfajta meghajtóprogramokkal is együtt tud működni *(például SCSI és NDIS)*. A WMI a WDM szolgáltatót használja az adatok közzétételére, az eszközök beállítására, és az eszközmeghajtók eseményeiről szóló értesítések szolgáltatásához.

A WMI WDM részei az alábbi adatokat osztják szét:

- **Közzétett adatok:** szabványos adatkészletet találunk a Windows 2000-rel szállított meghajtókban.
- **Egyedi adatok:** Az OEM-ek és független gyártók meghajtóbővítései szolgáltatják ezeket.
- **Biztonsági adatok:** A Windows 2000 biztonsági leírói szolgáltatják ezeket.
- **Erőforrásigényes adatok *(opcionális)*:** Néhány adatgyűjtő tevékenység jelentősen befolyásolhatja a meghajtóprogram teljesítményét, ezért ezeket az adatokat csak akkor kell gyűjteni, amikor egy felügyeleti alkalmazás külön kéri ezt. Amikor az adatokat használó utolsó alkalmazás is befejezi a működést, a WMI jelzi a meghajtóprogramnak, hogy fejezze be az adatgyűjtést. Azt, hogy mi az erőforrásigényes adat, nem a WMI, hanem a meghajtó készítője dönti el, egy egyszerű eljárás segítségével
- **Eseményekről szóló értesítések:** Az eseményekről szóló értesítés a WMI egyik fő funkciója, mert ez teszi lehetővé, hogy a meghajtók felderítsék a hardverrel kapcsolatos eseményeket és/vagy hibákat. A hardvereseményekről szóló értesítéseket az eseményszűrők és a CIM objektumkezelő kezeli.

A WMI azt is lehetővé teszi, hogy a felügyeleti alkalmazás beállításokat végezzen egy eszközön.

A WMI működés közben - parancsfájl példák

Végül lássunk néhány egyszerű példát a WMI működéséről. Nem állítjuk, hogy ezek alapján bárki is képes lesz saját, teljes körű rendszerfelügyeleti parancsfájlokat készíteni, célunk inkább az, hogy az eddigi elméletet egy kis gyakorlattal támasszuk alá. Tekintettel a CIM objektumok óriási számára meg sem próbálunk átfogó képet adni, csupán a legkézenfekvőbb objektumok lekérdezésére vállalkozunk.

A példák kódját egy .vbs kiterjesztésű szövegfájlba írva lehet kipróbálni. A futtatás: cscript fájlnév.vbs. Így a kimenet a standard out-ra megy, ezáltal könnyen át lehet irányítani fájlba a cscript fájlnév.vbs >kimenet.txt segítségével.

Az első példánkban felépítünk egy általános eljárást, amivel bármelyik WMI osztályt meghívhatjuk és megfigyelhetjük annak összes jellemzőjét. Konkrét alkalmazás esetén persze nem kötelező végigmenni minden egyes tulajdonságon, elég csak a kiválasztottakat közvetlenül elérni.

A minden jellemzőt listázó eljárás:

```
Sub CallWMI(Provider, Output)
Set PropSet = _
GetObject("winmgmts:{impersonationLevel="&_
"impersonate}").InstancesOf(Provider)

For Each Props In PropSet
  For Each Prop In Props.Properties_
    Message = Message & Prop.Name & " : "

    If IsArray(Prop) Then
      For Each Va In Prop.Value
        If Not IsNull(Va) Then
          Value = CStr(Va)
        Else
          Value = "Null"
        End If
        Message = Message + Value & "; "
      Next
    Else
      If Not IsNull(Prop.Value) Then
        Value = CStr(Prop.Value)
      Else
        Value = "Null"
      End If
        Message = Message + Value & "; "
    End If
    Message = Message + vbCrLf

  Next
```

```
Next

OutputString = Message

End Sub
```

Az eljárás létrehoz egy példányt a Provider paraméterben megadott WMI osztályból, és végiglépked annak összes jellemzőjén, összegyűjtve azokat szöveges formában az output paraméterbe.
Távoli gép eléréséhez a GetObject-et kell átírni a következő módon:

```
GetObject("winmgmts://gépnév")
```

Nézzük meg az eljárásunk használatát!
Alapinformációk lekérése egy számítógépről:

```
CallWMI "Win32_ComputerSystem", OutputString
WScript.Echo OutputString
```

A kimenetből néhány érdekesebb sor:

```
CreationClassName : Win32_ComputerSystem;
CurrentTimeZone : 60;
Description : AT/AT COMPATIBLE;
Domain : NETACADEMIA;
DomainRole : 4;
InfraredSupported : True;
Name : PLATAN;
NumberOfProcessors : 1;
SystemStartupDelay : 30;
SystemType : X86-based PC;
És még további 100 sor...
```

Látható, hogy a Win32_ComputerSystem osztály segítségével a legalapvetőbb információkat lehet lekérni egy számítógépről. Részletesebb információkat további WMI osztályok meghívásával lehet elérni. Nézzük meg például, mit lehet megtudni a hálózati kártyákról.
Hálózati kártyák adatainak lekérdezése:

```
CallWMI "Win32_NetworkAdapter", OutputString
```

Kimenet:

```
AdapterType : Ethernet 802.3;
Description : Intel 21041 Based PCI Ethernet
Adapter;
MACAddress : 00:80:48:EA:75:8A;
Manufacturer : Intel;
PNPDeviceID : PCI\VEN_1011&DEV_0014&SUBSYS_
00000000&REV_11\3&61AAA01&0&58;
PowerManagementSupported : False;
...
```

Mit lehet tudni a videokártyáról?

```
CallWMI " Win32_VideoController", OutputString
```

Kimenet:

```
AdapterDACType : S3 SDAC;
AdapterRAM : 4194304;
Caption : S3 Inc. Trio3D/2X Display Driver Version
3.30.10;
CurrentBitsPerPixel : 24;
CurrentHorizontalResolution : 1024;
CurrentRefreshRate : 85;
CurrentVerticalResolution : 768;
DriverVersion : 4.1024.330.0010;
MaxRefreshRate : 85;
MinRefreshRate : 43;
... És még további 60 jellemző.
```

Ezek a példák csak a hardverrel foglakoztak. Azonban hihetetlenül sok szoftverinformáció is elérhető a WMI-n keresztül. Ízelítőül és házi feladat gyanánt a következőket javaslom kipróbálásra:
Win32_Process
Win32_Service
Win32_Share
Win32_Directory
Ráadásul ezeken az osztályokon keresztül nemcsak információkat érhetünk el, hanem vannak metódusaik is, amelyeket meghívva az adott objektummal kapcsolatos műveleteket is el lehet végeztetni. Például a Win32_Directory-nak van Compress metódusa, amivel egy kiválasztott állományt vagy könyvtárat be lehet állítani tömörítettre. Vagy a Win32_Process-nek van egy Terminate metódusa, amivel egy processzt ki lehet lőni. Ha mindehhez hozzávesszük azt a tényt, hogy a WMI működik távoli gépre is, akkor valóban fantasztikus távvezérlési lehetőség van a kezünkben.
Megjegyzés: a Wscript.Echo metódus hibaüzenettel elszáll, ha a lekérdezés eredménye túl sok szöveget ad vissza. Ez a hiba eltűnik, ha a kimenetet fájlba irányítjuk át.

További információk

A Windows 2000 Server felügyeleti infrastruktúrájáról szóló legfrissebb információk a [1] címen találhatók meg.
A Windows Management Instrumentation-ről (WMI) bővebb információ az MSDN-en a [2] címen lelhető fel.
WBEM-ről és a DMTF-ről további információk a DMTF webhelyén találhatók: [3].
A Win32-es platformon elérhető osztályokról és azon tulajdonságairól a [4] címről kiindulva lehet részletes segítséget találni.

A cikkben található URL-ek:

[1] http://www.microsoft.com/windows/server/technical/management/default.asp
[2] http://msdn.microsoft.com/downloads/sdks/wmi/default.asp
[3] http://www.dmtf.org/
[4] http://msdn.microsoft.com/library/psdk/wmisdk/clascomp_3d4j.htm

Microsoft Network Monitor *(III. rész)*

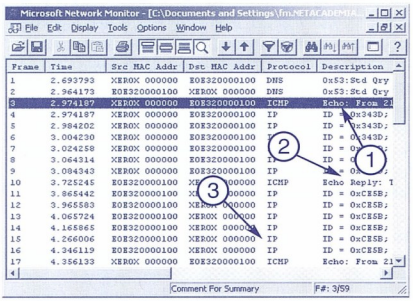
PING, PING, PING

Mai élveboncolásunk első alanya egy jó nagy ping lesz. Méretét tekintve akkora, hogy semmiképpen sem fér be egyetlen, maximálisan 1514 bájt méretű keretbe. A kísérlet azért fontos számunkra, mert a való világ valós adatainak többsége lényegesen nagyobb, mint 1514 bájt, így – mint csepp a tengerben – e jókora pingen felfedezhető ugyanaz az átalakítási eljárás, ami minden nagyobb fájl esetén megtörténik, s amit a FONTOS.XLS is elszenved, ha hálózaton keresztül továbbítjuk – vagyis darabokra bomlik a feladónál, és ezekből áll össze a fogadónál. Ezt kapd el:

```
PING –l 10000 www.netacademia.nat
```

Az itt elemzett PING letölthető az újság weblapjáról, neve: BIGPING.CAP.

Először madártávlatból vessünk egy pillantást az elkapott hálózati forgalomra:



A nagy ping látképe

Jól megfigyelhető, hogy az ICMP Echo ① és ICMP Echo Reply ② közé további IP csomagok ékelődtek, melyekről – legalábbis első ránézésre – nem látszik azonnal, hogy a PING-hez tartoznának. Közönséges IP csomagok volnának?? A korábbi cikkeimben leírtaknak megfelelően egy beérkező hálózati csomag determinisztikus úton jut el a megfelelő feldolgozóegységhez, hisz minden egyes réteg pontosan tudja, hogy a számára feldolgozhatatlan „Number of data bytes remaining" adatokat hova továbbítsa. Az Ethernet keretben az Ethernet Type mező mutatja meg, hogy mit szállít az adott keret, míg például az IP-ben a Protocol mező végzi ugyanezt a feladatot. Nézzük tehát meg az egyik „kósza" IP csomagot ③, hogy vajon mit tud magáról:

```
IP: ID = 0x343D; Proto = ICMP; Len: 1500, Frag.
    Offset = 1480 (0x5C8)
    ....
  IP: Total Length = 1500 (0x5DC)
  IP: Identification = 13373 (0x343D)
  IP: Flags Summary = 201 (0xC9)
    IP: .......1 = More fragments in datagram
    after this one
    IP: ......0. = May fragment datagram if
    necessary
  IP: Fragment Offset = 1480 (0x5C8) bytes
  IP: Time to Live = 128 (0x80)
  IP: Protocol = ICMP - Internet Control Message
   IP: Fragmented Datagram Data: Number of data
    bytes remaining = 1480 (0x05C8)
```

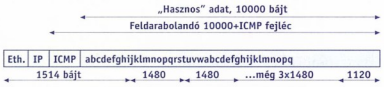
7
5
4
6

Elég sok újdonságot mutat ez a néhány, természetesen gondosan megritkított sor. Az IP fejléc Protocol mezőjéből leolvasható ④, hogy az itt szállított adat tulajdonképpen ICMP, vagyis annak maradéka. Honnan tudjuk, hogy maradék? Egyfelől onnan, hogy a Fragment bit ⑤ be van állítva, ami tulajdonképpen nem azt jelzi, hogy az éppen nagyítónk alá került csomag egy töredék, hanem azt, hogy további töredékek várhatók *(More fragments in datagram after this one)*. Másfelől onnan, hogy a következő olvasható a „hasznos" adatok előtt ⑥: Fragmented Datagram Data: Number of data bytes remaining.

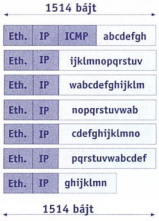
Ami pedig a NetMon lustaságát illeti *(hogy csak odafigyeléssel deríthető ki, hogy ez a ping folytatása)* érthető, hiszen ICMP fejlécet nem találunk ebben a csomagban. Minek is kellene mind a hét utazó csomagban megismételni, hogy ez egy ICMP Echo? A csomagok anélkül is utat találnak felfelé a protokollveremben. A kérdés csak az, hogyha a gép egyszerre két nagydarab, töredezett pinget kap, akkor a töredékeket hogyan tudja megfelelően osztályozni, hogy honnan jöttek? Erre szolgál az Identification *(=13373 (0x343D))* mező ⑦. Minden beérkező IP csomagban, és töredékeiben azonos ez a random érték. Így nincs más teendője az IP feldolgozónak, mint mindaddig ugyanoda dobálni a beérkező töredékeket, amíg azok Identification mezője megegyezik, és be nem fut az utolsó töredék, ahol a Fragment bit már nulla *(Last fragment in datagram)*.

Tördelés

Most vessünk egy pillantást a tördelés általános menetére. Az alábbi ábra a PING feldarabolásának sematikus vázlata. Összeáll a teljes, tízezer bájtos PING, és ezt valaki csomagok sorozatára bontja úgy, hogy az ICMP adattartalom lesz a trancsírozás áldozata – de az IP fejléc épségben megismétlődik!



A 10000 bájtos ICMP Echo darabolás előtt...



...és a darabolás után

A teljes átvitt adatmennyiség a fejlécekkel együtt pedig 6x1514+1162=10246 bájt, ami már az Ethernet és IP fejlécek méretét is tartalmazza. Ki végzi vajon a tördelést? Az előzőek alapján az IP protokollra gyanakodhatunk. Más alkalmazásokra gondolva beláthatjuk, hogy nem az alkalmazás darabol. Az biztos, hogy nem az Excel fogja szorgosan felaprítani a FONTOS.XLS-t adatátvitel előtt, hisz pontosan arra találták ki a rétegzett hálózatni architektúrákat, hogy például a hardver által okozott kellemetlenségek rejtve maradjanak a magasabb szintű rétegek, s főleg az alkalmazások elől. Jelen esetben az „alkalmazás" az ICMP, előle rejtve marad a darabolás. De vajon miért pont az IP végzi ezt a munkát? Miért nem az Ethernet meghajtóprogramja? Hisz mégiscsak közelebb áll az 1514 bájtos korlátot okozó Ethernet kártyához?

A kérdés jó, a válasz pedig az, hogy az Ethernet fejléc ugyan „közelebb" van a hardverhez, de épp emiatt kevésbé számíthatunk rá: amikor egy IP csomag útválasztó-hegyeken át jut el egy távoli kiszolgálóig, akkor az Ethernet fejléc tartóssága enyhén szólva is megkérdőjelezhető, hisz a legelső útválasztó lebontja, és teljesen más *(például Token-Ring)* fejlécet illeszt a csomag elé. Így a végponttól végpontig történő címzés nem bízható az Ethernetre, csak egy minden matatást és útválasztást túlélő rétegre – s az IP pont erre való. Minden töredezett csomag elején ott a helye az IP fejlécnek, hogy a töredékek mindegyike utat találjon a célba! Az ICMP fejlécről ugyanez a nagyfokú hasznosság már nem mondható el, emiatt az már a tördelés áldozatául esik. Ha majd rátérünk a TCP csatorna elemzésére, látni fogjuk, hogy ott már a TCP fejléc is beleszól a szállításba, így világos, hogy az sem fog belekerülni a darálóba. A következő kérdés önként adódik: honnan tudja az IP, vagy majd a TCP, hogy mekkorára kell tördelni? A hálózati kártya által elfogadott legnagyobb keret méretét, valamint az összes réteg fejlécigényét megfelelő API hívásokkal le lehet kérdezni, s a Windows 2000 ezt éppúgy elvégzi menet közben, mint annyi minden mást. De hogy egy kicsit a múltba révedezzünk...

Maximum Transmission Unit

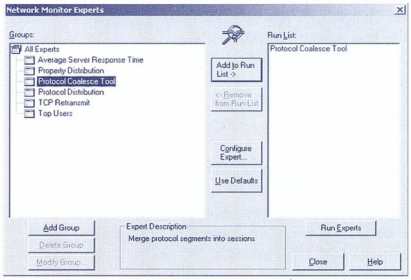
Hajdanában-danában, amikor az operációs rendszerek még feleennyit tudtak, bizony néha szükség lehetett a nem *(igazán)* támogatott kártyák 3rd party meghajtóit egy kis INI-fájl matatással megsegíteni a hatékony működés érdekében. Volt idő, amikor a tördelési határt kézzel kellett beállítani, s ennek a letűnt korszaknak állít emléket a regisztrációs adatbázis MTU kulcsa, ahol mind a mai napig meg lehet változtatni kézzel a tördelési határt.

```
Kulcs neve: MTU, alapértelmezésben nincs ott
Helye: HKLM\SYSTEM\CurrentControlSet\Services\
  Tcpip\Parameters\Interfaces\<interface-name>
Adattípus: REG_DWORD
Tartomány: 0x44 – dinamikusan felismert MTU vagy
    0xFFFFFFFF
Alapértelmezett érték, ha a kulcs hiányzik:
    0xFFFFFFFF (=Autodetect)
```

Ennek a gyakorlatban még sohasem vettem hasznát, egyszerűen csak érdekes. Még érdekesebb, hogy míg IP szinten a helyi hálózati kártya MTU-jára vagyunk kíváncsiak, addig a TCP réteg a két kommunikáló fél között a teljes útvonalon érvényesíthető MTU-ra kíváncsi, s ezt le is kérdezi az útvonal teljes hosszán – ha tudja.

Protocol Coalesce Tool

A széttöredezett PING remek lehetőséget nyújt egy másik nagyon hasznos eszköz bemutatására, amellyel nemcsak a végponti kommunikáló felek, hanem a hálózati forgalmat figyelő köztes személy is képes a töredékeket egyesíteni, s ezzel értékesebb adatokhoz jutni. Sajnos az egyesítő *(Coalesce)* eszköz az ingyenes, beépített Network Monitorban le van tiltva, így csak vastagabb pénztárcájú olvasóim figyeljenek jól. A vagyonosabbak ugyanis egy szakajtóderéknyi kiváló szakértőt *(Expert)* is kapnak a NetMonnal, akik különféle elemzéseket végezhetnek a már elkapott/elmentett hálózati forgalom alapján. A Tools->Experts menüpontból indulunk *(figyelem, a NetMon menüi változnak annak függvényében, hogy melyik ablakban: a fő vagy a részletes nézetben ácsorgunk éppen!)* ahol elénk tárul egy csomó fura szerzet, amelyekkel részletesebben majd egy későbbi cikkben foglalkozom, most a töredezett ping összevonására összpontosítunk.



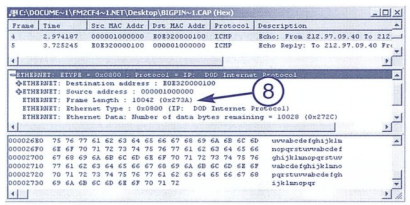
A Protocol Coalesce Tool

A „Protocol Coalesce Tool"-t az „Add to Run..." gombbal áthajítottam a jobboldalra, a lefuttatandó feladatok közé, s a „Run Experts" megnyomása után máris gyönyörködhetünk, no nem az eredményben, hanem abban az útvonalban, ahova a szakértés eredménye került, ami roppant szív-

derítő látvány, különösen, ha a Desktopra teszi ;)

```
C:\Documents and Settings\fm.NETACADEMIA\
↳ Desktop\bigping(Coalesced)02.cap
```

Innen már csak egy kis egérfutkosás, és megnyitható a készítmények, az összevont pingek fájlja. Vessünk egy pillantást egy összevont ICMP Echo-ra. Mekkora az Ethernet keret mérete?



Összevont bigping

Amint az a fenti ábráról leolvasható, a keretméret 10042 bájt ⑧, ami sokszorosa a maximális 1514 bájtnak. Az összevonás eredménye többé soha nem tuszkolható ki az Ethernet hálózatba, bármennyire is szabványos keretnek látszik első pillantásra, s bármilyen csábító is a pénzes NetMon „Transmit Frame" menüpontja. Number of data bytes remaining? 10028=10000 betű + 20 bájt IP fejléc + 8 bájt ICMP fejléc. Pont, mint azon az előző ábrán, ahol a ping darabolás előtti állapotát rajzoltam le. Ugyanezzel az eszközzel lehet egy, akár megabájtos keretbe összeszedni a hálózati forgalomban tördelve utazó dokumentumokat. De nem akarok további hackertippeket adni, haladjunk a korral, és lássunk más érdekességeket az ICMP háza tájáról.

Tracert

Gondolkodott-e már valaki azon, hogy a tracert *(Trace Route)* parancs honnan a csudából tudja két távoli végpont között az útvonalat? Talán le lehet kérdezni az útválszatókat? Lehet, hogy le lehet, de honnan tudjam, hogy melyiket, hisz az IP-ben éppen a szabad útválasztás az egyik legnagyobb találmány a világon, azaz akár minden egyes IP csomagom más útvonalon juthat el a célállomásig. Vajon van-e esély arra, hogy hálózati forgalmam nem a tracert által kiírt/megjósolt irányba fog menni? Ha alternatív útvonalak vannak a két végpont között, és az útválasztók útvonaltáblája is lehetővé teszi, akkor bizony jó esély van rá! Aki esetleg nem emlékezne rá, a tracert így működik:

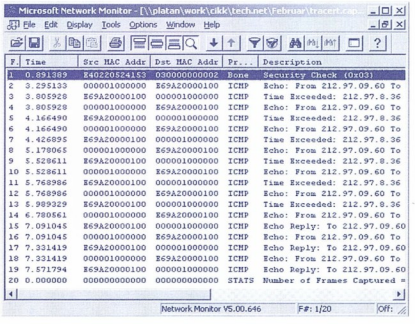
```
C:\>tracert www.netacademia.net

Tracing route to www.netacademia.net [212.97.8.36]
over a maximum of 30 hops:

1    510 ms    361 ms    260 ms
     ↳ as5200-0.ahol.com [212.97.9.1]
2    351 ms    240 ms    221 ms
     ↳ core-if.router0.hu.ahol.com [212.97.8.1]
3    310 ms    240 ms    241 ms
     ↳ hofeherke.netacademia.net [212.97.8.36]
Trace complete.
```

Sajnos túl közel ültem a webkiszolgálónkhoz, így a lista rövidre sikeredett, de a világ más tájairól a www.netacademia.net bizony akár 40 routernyi távolságban is lehet! A válaszban soronként láthatjuk a köztes útválasztókat, valamint azok *(3 darab)* válaszidejét. A válaszidők átlaga talán többet mondana, de ez a jószág három különböző mérésének eredményét nem képes kiátlagolni, hanem szépen kiírja mindet egymás mellé. Például az as5200-0.ahol.com [212.97.9.1] nevű gép először 510 ms, másodszor 361 ms, végül a harmadik mérésre 260 ms alatt válaszolt. Ha a mért idők helyén csillagokat látunk, akkor az adott válaszidő több volt, mint 1 másodperc.

Idejében elindított NetMonnal szerencsésen elkaptam a tracert hálózati forgalmát, hogy megvizsgálhassuk, vajon melyik az a protokoll, amivel routereket lehet lekérdezni.

Igen, az ICMP-ről van szó már megint. Említettem, hogy az ICMP-nek rengeteg változata van, ezek közül most ismerkedjünk meg az Echoval... Na neeeeeeee! Már megint a ping?

Igen, a ping fog segíteni, mert olyan érdekes módon fogjuk használni, hogy a router kénytelen lesz válaszolni, azért mert a csomag TTL-je - gonosz módon – még a célba jutás előtt lejár. A múltkori cikkben emlékeztem meg az IP fejléc mezőiről, s akkor esett szó a TTL szerepéről, ami megakadályozza, hogy egy csomag a végtelenségig keringjen hibás útválasztású hálózatodon. Általában egy IP csomag elég magas TTL-lel indul útjára *(128)*, ami elegendően nagy ahhoz, hogy a földet akár háromszor is körbeutazza. A tracert parancs azonban a legelső csomagot 1-es TTL-lel adja fel, amely az első útválasztón azonnal halálra ítéltetik. Mit tesz ilyenkor a router? Akár csendben el is temethetné a csomagot, de visszaszól, mégpedig – és ez tényleg új ICMP szerep – ICMP Time Exceeded üzenettel *(időtúllépés)*. Ennek a válasznak a megfordulási idejéből számítódik és íródik ki a képernyőre a legelső válaszidő. Ezután a tracert még két ilyen TTL=1-es csomagot kiküld, hogy meglegyen a három mérési érték, utána három darab TTL=2 küldése következik, majd TTL=3 és így tovább, amíg a TTL elég nagy nem lesz ahhoz, hogy a csomag eljusson a végállomásig, s az Echo-ra megérkezhessen az Echo Reply. Madártávlatból a folyamat így néz ki:



A tracert forgalma

S most vizsgáljuk meg az egyes csomagokat! Vajon a feladott Echo-kban mi az IP cím? Mindig a legközelebbi routerre mutat? Szó sincs róla, mindig a végcél IP címe van benne, csak

a csomag *(az alacsony TTL miatt)* nem jut el a célba. A Time Exceeded csomag beltartalma, részenként a következő:

```
IP: ID = 0x8214; Proto = ICMP; Len: 56
  ...
    IP: Precedence = Internetwork Control
```

A Precedence értéke megváltozott Routine-ről *(0x00)* Internetwork Controlra *(0xC0)*.

```
IP: Data: Number of data bytes remaining = 36
(0x0024)
ICMP: Time Exceeded: 212.97.8.36  (See frame 2)
  ICMP: Packet Type = Time Exceeded
  ICMP: Time Exceeded Code = Time To Live
↳Exceeded In Transit
    ICMP: Data: Number of data bytes remaining =
      ↳ 28 (0x001C)
```

Innentől pedig megismétlődik az eredeti IP csomag, hogy az időtúllépési üzenetet fogadó oldalán fel lehessen ismerni, mi nem ért célba:

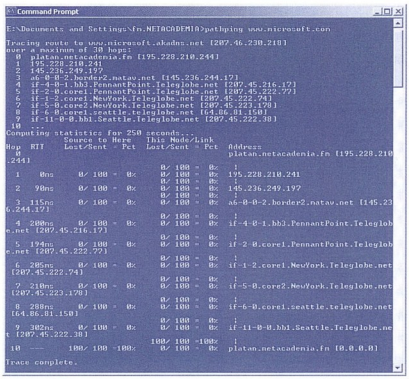
```
ICMP: Description of original IP frame
ICMP: (IP) Version = 4 (0x4)
… eredeti IP cím, checksum stb.
…
```

A hibaüzenetet feladó gép IP címe alapján a feladó még elvégez egy fordított DNS lekérdezést *(reverse lookup)*, hogy megtudja annak a routernek a DNS nevét *(ha van)*, ahol a csomag elhunyt, majd kiírja a képernyőre. Az én esetemben a fordított lekérdezés azért nem látszik az elkapott hálózati forgalomban, mert többször is lefuttattam ugyanazt a parancsot, s míg legelőször még szükség volt a reverse lookupra *(de akkor még nem futtattam a NetMont)*, a továbbiakban már nem, mert a Windows 2000 ügyféloldali DNS gyorsítótára elmentette a válaszokat, amit IPCONFIG / DISPLAYDNS-sel le is ellenőrizhetünk.

Egy kérdés maradt hátra: vajon biztosan a tracert által kiírt útvonalon fognak haladni csomagjaink? Mint az a madártávlati képből látszott, itt egymástól teljesen független IP csomagok jöttek-mentek, az egyiknek kisebb volt a TTL-je, a másiknak nagyobb, de a feladó és a cél IP címén kívül más közös nem volt bennük. Ad abszurdum még az is elképzelhető, hogy minden egyes tracert csomag más úton jut el a számára kijelölt kivégzési pontig *(ahol elfogy a TTL)*, így az útvonalinformációt – nagyobb távolságokon - igencsak fenntartásokkal illik kezelni.

Pathping

Ha valaki esetleg veszi a fáradságot és körülnéz a Windows 2000 SYSTEM32 könyvtárában, talál ott egy pathping.exe parancsot, amely nagyjából ugyanazt tudja, mint a tracert azzal kiegészítve, hogy képes leellenőrizni az útvonal teljes hosszán az RSVP protokoll támogatását. Az RSVP a Resource Reservation Protocol rövidítése, és garantált sávszélesség *(QoS, szolgáltatásminőség)* lefoglalására használható olyan útvonalakon, ahol minden router támogatja ezt a lehetőséget. Csínján bánjunk a Pathpinggel, mert elég alapos/erőszakos jószág! Míg a tracert beéri routerenként 3 pinggel, addig ez utóbbi valóságos pingáradatot ereszt meg a célgép felé *(routerenként 100 darab ICMP Echo!!)*, igaz, mérési eredményei jóval pontosabbak a tracert-től kapott adatoknál, hisz száz ping esetén már van mit statisztikázni: százalékosan meg tudja mondani, melyik útválasztó mennyit hibázik. Az alábbi ábra azt mutatja, hogy milyen kiváló vonalaink vannak a Microsoft.com-ig ;)



A pathping eredménye

Ennek a mérésnek a hálózati forgalma 1000 *(ezer!)* darab ICMP Echo-t, és Replyt jelentett, erre bizonyíték az elkapott forgalom utolsó „keretében", a statisztikacsomagban olvasható:

```
STATS: Number of Frames Captured = 2048
STATS: Elapsed Time = 5 Minutes  18 Seconds
↳ 439786 MicroSeconds
STATS: Total Frames Captured = 2048 (0x800)
STATS: Total Bytes Captured = 262073 (0x3FFB9)
STATS: MAC Frames Received = 1875
STATS: MAC CRC Errors = 0
STATS: MAC Bytes Received = 0x000000000009C036
STATS: MAC Frames Dropped due to No Buffers = 0
STATS: MAC MultiCasts Received = 30
STATS: MAC BroadCasts Received = 26
STATS: MAC Frames Dropped due to HardWare
↳ Errors = 0
```

A statisztikacsomagot mindig a NetMon Trace utolsó csomagjában, az itt felhasznált fájlokat *(BIGPING.CAP, tracert.cap, pathping.cap)* pedig a weben találják Lelkes Olvasóim.

Fóti Marcell
marcellf@netacademia.net

Használjunk USB-t és 1394-et

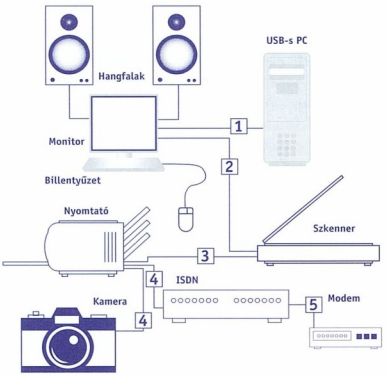
Közeleg az idő, amikor PC-ink hátuljáról eltűnnek a soros és a párhuzamos portok csatlakozói, megszűnik a kábeláradat. Nem lesz PS/2-es port, nem lesz külön hangkártya- és modemkimenet, nem fog fájni a fejünk az elfogyott IRQ-k miatt. Lesz viszont egy vagy két *(néhol négy)* USB konnektor, egy IEEE-1394 csatlakozó, egy monitorcsatlakozó és egy helyi hálózati kimenet. Semmi több.

Mi is az a*(z)* USB?

Mint a neve is mutatja az USB *(Universal Serial Bus)* egy busz, melyre maximum 126 *(!)* eszköz csatlakoztatható. Figyelem, ezek az eszközök összesen egyetlen egy megszakítási címet fognak elhasználni! Az eszközök bármikor csatlakoztathatók vagy leköthetők a buszra*(-ról)*, az operációs rendszer automatikusan érzékeli és „beállítja" ezeket. Nincs jumperelés, nincs újraindítás, minden Plug and Play.

Hogyan kell használni?

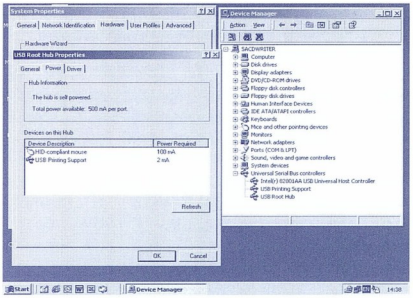
Kábelezési szabályok: A kábelek legfeljebb 5 méter hosszúak lehetnek, melyek segítségével maximum egy 5 kábelből álló láncolatot hozhatunk létre a számítógéptől a legutolsó még használható eszközig. *(Tehát a maximális távolság a géptől 25 m, de a kábelhosszokat hiába vesszük kisebbre, a láncolat akkor is csak öt lépésből állhat.)* A csomópontokban természetesen lehetnek elágazások, ha az eszközök több USB porttal rendelkeznek vagy az eszköz maga egy USB HUB. *(Így lehetséges az elméleti 126-os határ elérése.)* Az alap USB kábelnek különböző a két vége, ezért lehetetlen nem működő „hurkot" létrehozni.



Az ábrán jól látható a maximum 5 láncszem

Portok?

Egy USB port lehet aktív vagy passzív. Az aktív port maga biztosítja a rácsatlakoztatott eszköz számára az energiát *(aktív portja természetesen olyan eszköznek lehet, amely az áramot nem a buszról veszi)*. A buszt vezérlő szoftver gondoskodik arról, hogy ha „elfogyott a buszról az összes energia" *(maximum 500 mA)* akkor több eszköz ne akarjon áramot felvenni.



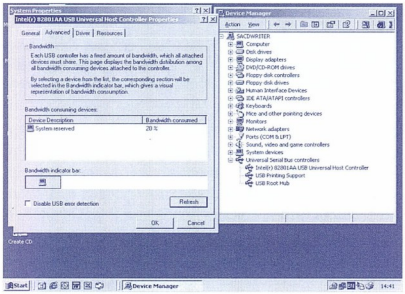
Az egérkénk passzív és sokat fogyaszt, a nyomtató viszont aktívan szerénykedik

Ennek feloldására való az aktív port. Ha egy aktív portot tartalmazó eszközt kikapcsolunk, akkor onnantól lefelé a láncban lévő eszközök nem biztos, hogy működni fognak, minden az energiafelvételen múlik. Honnan tudhatjuk, hogy egy eszköz mennyi energiát használ *(lásd fent)* vagy milyen sebességgel működik *(high/low)*? Erre szolgál a Windows 98 CD-n egy kis programocska: (\tools\reskit\diagnose\usbview.exe).

Az USBVIEW Windows 2000 alatt is megy...

Az USB a következő operációs rendszerekben támogatott: Windows 95 OSR2.1B; Windows 98; Windows 2000.
És ha öregecske a gép?
USB porttal nem rendelkező számítógépet *(pl. régebbi Pentium modellek)* is bővíthetünk olyan PCI-os kártyával, amely

USB csatolót tartalmaz. USB segítségével akár hálózatot is kiépíthetünk, erre is léteznek már eszközök.



Az 1.1-es USB teljes sávszélessége hamar elfogyhat

USB 2.0

Az USB 2.0-s verziójában *(melyet olyan cégek dolgoztak ki, mint a Compaq, HP, Intel, Lucent, Microsoft, NEC és a Philips)* a sebesség közel negyvenszeresére, azaz 480 Mbit/s-ra emelkedik majd! Jó hír, hogy a gyorsabb adatforgalom a „régi" USB kábeleken zajlik, valamint a meglévő eszközök is ráköthetőek az új buszra *(ez alól kivételek a HUB-ok, amelyek szerepét a már háromféle sebességre képes - 480, 12, 1,5 Mbit/s - 2.0-sok veszik át)*. Ami viszont szükséges: egy USB 2.0-s PCI csatolókártya, vagy egy új alaplap, ami már 2.0-s USB-t támogat, de sajnos ezek megjelenésére még várnunk kell egy pár hónapot. *(Az Intel már az I815-ös chipsetben ígérte a támogatást, de később ez elhalasztódott.)* A szoftveres támogatásra is várni kell, jelenleg még nem létezik 2.0-s USB patch a windows-okhoz. *(A Whistler, „akarom mondani" a Windows XP már támogatni fogja.)*

Technológia	Maximális sebesség (elméletben)
Soros port	230 Kbit/s
USB 1.1 Low	1,5 Mbit/s
USB 1.1 High	12 Mbit/s
FireWire	400 Mbit/s
USB 2.0 High	480 Mbit/s
Ultra SCSI-3	160 Mbyte/s

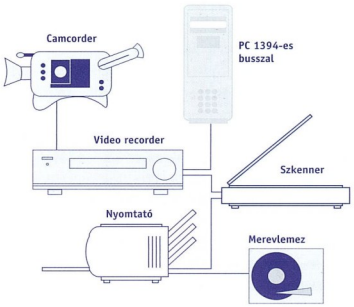
Fire-Wire

Mivel az USB 2.0 még nem elérhető, egy lehetséges alternatíva lehet az IEEE 1394. Ez tulajdonképpen az Apple által kitalált Fire-Wire, amelynek szabványát 1990-ben kibővítették, s ennek eredményeképpen jött létre az IEEE 1394 specifikáció.

Sebesség

A 1394-es busz két sebességen működhet: 200 Mbit/s-on *(S200)* és 400 Mbit/s-on *(S400)*, ez utóbbi már 50 Mbyte/s-ot jelent. *(Jobb, mint egy UW SCSI!)* A nagy adatátviteli sebesség igénye főleg multimédiás applikációknál jelentkezik, tehát azon eszközök lesznek *(vannak)* 1394-es csatlakozóval ellátva, amelyek ebben szerepet játszanak: camcorderek, videofelvevők, merevlemezek. Persze jelenleg is léteznek olyan eszközök *(pl. videograbber kártyák)*, melyekkel mondjuk a videófeldogozás elvégezhető, de az 1394 jóval olcsóbb megoldást jelent.

Az 1394 segítségével könnyedén létrehozhatunk otthon is egy kis videóstúdiót, hiszen az 1394-es buszra csatlakoztatott videókamera és a merevlemez között közvetlen adatforgalom jön létre, a „film" egy szoftverrel megszerkeszthető, majd tárolható az ugyancsak a buszra csatlakozható videofelvevőn.



Az én kis házistúdióm...

Kábelezés

Az IEEE 1394-es buszra maximum 63 eszköz csatlakoztatható egyszerre, a maximális kábelhossz *(két eszköz között)* négy és fél méter lehet. Legfeljebb 16 ilyen kábel fűzhető láncba, tehát a számítógép és a legmesszebb lévő eszköz maximális távolsága 72 méter. Nincs szükség terminálásra, ID jumperelésre *(SCSI)*, az eszközök menet közben csatlakoztathatók vagy leköthetők *(hot plug)*, minden rákapcsolt eszköz automatikusan címződik. Csakúgy, mint az USB esetében, a csillagpontos elrendezéshez HUB-okat használhatunk, de minden olyan eszköz, amelynek egynél több 1394-es csatlakozója van már HUB-ként is funkcionál. Az eszközök áramfelvétele is hasonló az USB-nél látottakhoz, azaz vannak aktív és passzív portok. A passzív porttal ellátott eszköz közvetlenül a buszról veszi fel a működéséhez szükséges energiát, az aktív port pedig a passzív eszközöket tudja ellátni maximum 3 Watt energiával. A használt kábel 6 eres: 2 kell az „áramhoz", 2 kábelen megy az adat, kettőn pedig a szinkronjel. *(Erről többet itt most inkább nem írnék...)*

Csatlakozó

A megfelelő csatlakozó keresésekor a következő szempontokat vették figyelembe: legyen strapabíró, legyen megfelelően árnyékolt és persze olcsó! A választás a Nintendo Gameboy csatlakozójára esett...

Azoknak sem kell elkeseredniük, akiknek a gépén nincs 1394-es csatlakozó, hiszen vásárolható olyan kártya *(általában PCI buszos)*, amely rendelkezik ilyennel.

Végezetül csak érdekességként jegyzem meg, hogy az SCSI-3 specifikáció felveti az IEEE 1394 kábelek használatának alkalmazását is *(hiszen ez jóval olcsóbb)*.

Tóth László
tothl@montana.hu
MCSE, Compaq ASE

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, megszűnnének 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

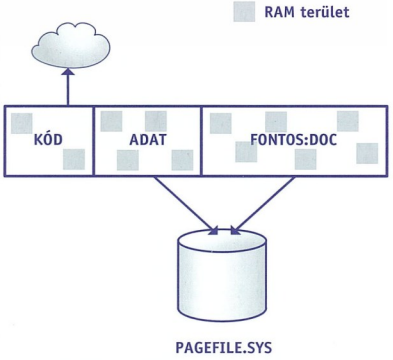
Először gyakorló rendszergazdaként vettem észre, hogy valami nem stimmel a világgal. Volt egy hálózatom több száz olyan munkaállomással, melyeknek merevlemezére az Office egyszerűen nem fért fel. Mit tesz ilyenkor a rendszergazda? Elolvassa a dokumentációt, és felfedezi, a SETUP.EXE /A módszert, mellyel 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.

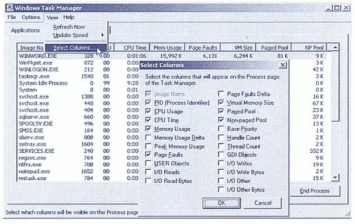


Egy alkalmazás kilapozása memóriahiány esetén. A szürke négyzetek a fizikai memóriában vannak – a többi terület nincs ott!

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 memóriaméricskélő eszközünk 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.

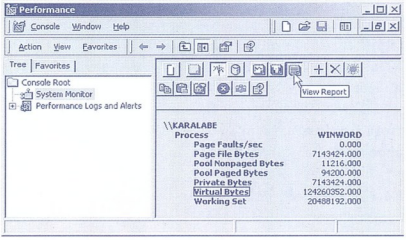


A Task Manager lehetőségei

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!

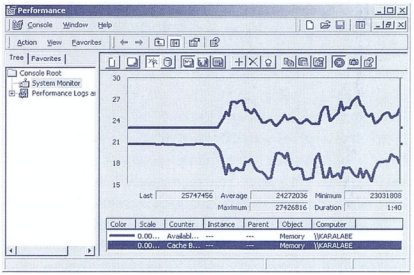
Ez a Windows 2000-nél még elég bután megy, mert - bár minden egyes alkalmazás mindig ugyanúgy indul - a 2000 nem adja neki oda az indulólapokat, az EXE lassan „befaultolódik". Ez magyarázza, hogy például egy vacak kis CALC.EXE közvetlenül indítás után miért mutat 334 Page Faultot, pedig senki sem bántotta!

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

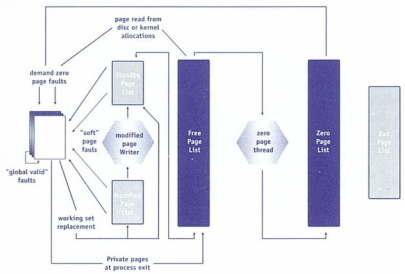
Mintha látszólag minden bájt, mely lefoglalódik, a cache területre kerülne, és fordítva. Mintha soha semmi nem kerülne át például a programok hatáskörébe. *(Persze átkerül, ha indítgatunk és leállítgatunk programocskákat, de ha csak a meglévőkkel manipulálunk, ez ritkán látszik.)* Mi valójában az „Available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

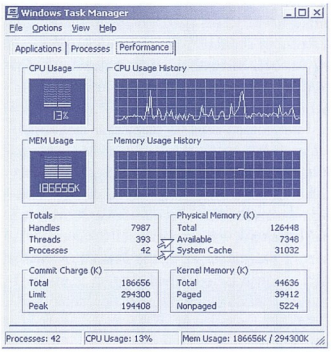
Cache Bytes

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

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:

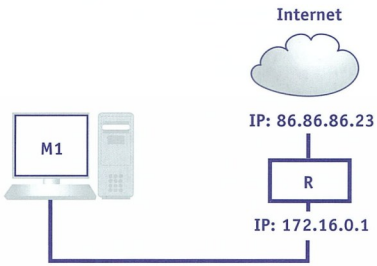
1. **szabály:** Minden gép csak a saját hálózatára küldhet csomagot.
2. **szabály:** Ha mégis távoli hálózatra kellene küldeni a csomagot, akkor az első szabály lép életbe.

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 :-)

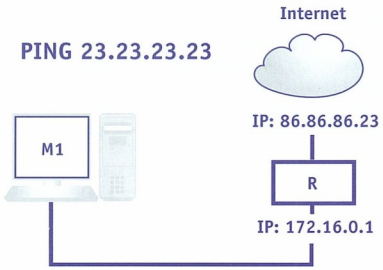
Az őrök „segítsége", közreműködése nélkül nincs lehetőség külső kapcsolatra. Ugyanez a helyzet az IP-vel. Amint külső hálózatra forgalmazunk, rá kell vennünk a fegyőrt, hogy segítsen. Mindenképpen szükségünk van egy segédre, egy postásra, egy útválasztóra. Egyszerű esetben alhálózatonként egyetlenegy „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.

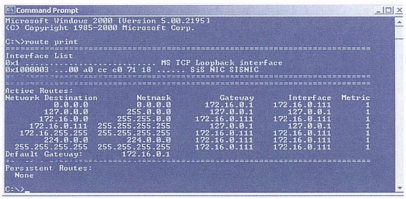
Itt jegyzem meg, hogy manapság minden Subnet Mask balról csupa egyes, jobbról csupa nulla, például:

```
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 belső, vagy külső. 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?

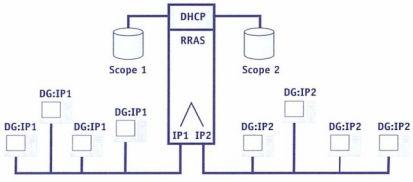
A kiemelt sor értelmezése ezek szerint: a 172.16.0.111 címre „induló" *(valójában érkező)* csomagokat a 127.0.0.1 *(SELF)* címre kell továbbítani, tehát be a gépbe. De az is kiolvasható, hogy ha a célgép IP címe mondjuk 193.193.193.193, akkor ... hogyis? Meg kell keresnünk a 193.193.193-as subnetet a 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

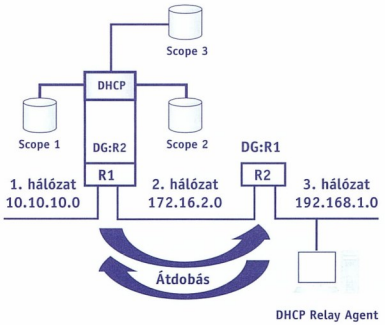
Még egyszerűbb a hálózat gépeinek felkészítése, ha az RRAS-ra felteszünk egy DHCP Servert, és mindkét kártya IP címtartományára felveszünk egy scope-t *(IP cím készletet)*, ahol a DG mindkét szkópban a megfelelő RRAS lábra mutat. Többkártyás gépen ennél többre nincs is szükség, mert a DHCP Server meg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

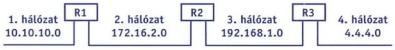
A fenti ábrán a DHCP Servernek már három szkópja *(IP címkészlete)* van, de csak kettő alhálózat gépei képesek közvetlenül címet kérni tőle. A harmadik hálózaton ilyen esetekben el lehet helyezni egy DHCP Relay Agentet, mely onnan a DHCP címkérési broadcastokat unicasttá alakítva átdobja a routeren túlra,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

Szálak *(II. rész)* Időzítés és prioritás

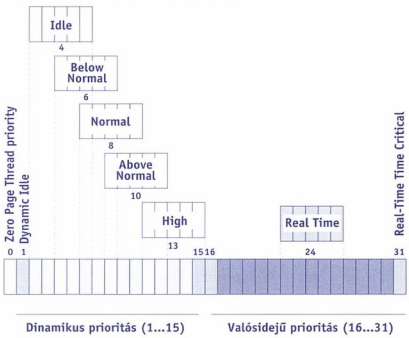
Időzítés és prioritás

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

Egy kis ismétlés

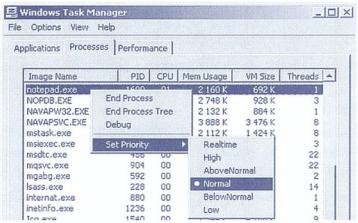
Foglaljuk össze az előző számban leírtakat: a Windows NT/2000 *(továbbiakban csak Windows, különös tekintettel arra, hogy nem a Windows 9x családról van szó)* operációs rendszer szinten végrehajtási szálakat futtat. Minden végrehajtási szál valamilyen processzhez tartozik *(egy processzhez akár több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

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



A Windows 32 prioritási szintje és a prioritásosztályok elhelyezkedése

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



Processz prioritásának módosítása futás közben a Task Manager segítségével

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

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

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

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

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

Alap és pillanatnyi prioritás

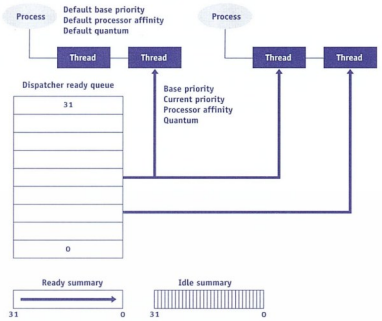
Egy végrehajtási szál pillanatnyi prioritásértéke két részből áll össze:

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

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

A dispatcher várakozási sor-adatbázisa

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



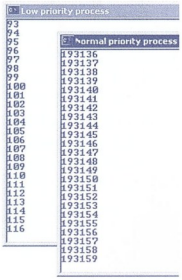
A Dispatcher várakozási sor-adatbázisa

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

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

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

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



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

Az ábrán látható tesztalkalmazás egyvalamire képes: számol. Teszi ezt addig, amíg a CTRL+C billentyűkombináció lenyomásával meg nem állítjuk. Az st.bat a start parancs segítségével gyors egymásutánban kétszer elindítja ugyanazt az alkalmazást, először egy alacsonyabb, majd egy magasabb prioritással. Az eredmény önmagáért beszél. Ha kipróbáljuk, érdemes megfigyelni, hogy a futásidő eloszlása korántsem sima: amikor néha végre az alacsonyabb prioritású végrehajtási szál is szóhoz jut, gyors egymásutánban 15-20 sort ír a képernyőre, majd újabb hosszú szünet következik. Látható, hogy a magasabb prioritású processz *(és az ő egyetlen végrehajtási szála)* több nagyságrenddel többet futhat, mint az alacsonyabb prioritású változat. Az érdekes az, hogy ez utóbbi is szóhoz jut néha... nem mond ez ellent a pár sorral ezelőtt említetteknek? Nos, nem: az alacsonyabb prioritású processz nyilván akkor futhatott, amikor a magasabb prioritású éppen nem volt futásra kész állapotban *(például mert I/O műveletre várt)*, de az is lehet, hogy az operációs rendszer mentette meg az „éhhaláltól" – lásd később.

A működő rendszer egyik kulcsa a várakozás

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

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

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

Priority Boost I/O műveletek után

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

Priority Boost az előtérben futó végrehajtási szálnak

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

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

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

Többprocesszoros rendszerek

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

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

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

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

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

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

Végül...

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

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:

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

XML-gessünk *(II. rész)*

Bevezetés

Két hónappal ezelőtti számunkban elindultunk az XML-hez kapcsolódó technológiák felidézésével. Ebben a számban utánajárunk az XML dokumentumok strukturális leírására szolgáló sémáknak, majd rátérünk az Internet Explorer XML képességeire. Ezen belül elmélyedünk az XML dokumentumok formázásában az XSL és a CSS alkalmazására. Az itt leírt alapok megteremtik azt a tudást, amely segítségével már ügyfél és kiszolgáló oldalon is tudunk XML dokumentum alapú weboldalakat építeni – melynek részleteit a következő számban olvashatják.

XML Stratégia

Az XML technológia vívmányai, akár tetszik, akár nem, a nyakunkon vannak, és mielőbb alkalmaznunk kell tudni őket. Havonta 4-5 oldalban alig lehet valamit átadni ezekből a forradalmi módszerekből, mert ha alapozással akarnánk kezdeni, akkor minimum a következő 10 cikk csak fogalmak tisztázásával menne el, és még mindig nem tudnánk használni az XML-t, csak értenénk, hogy mit takarnak azok az X-szel kezdődő néhány betűs rövidítések. Emiatt a cikksorozatomban hibrid stratégiát vezetek be. Minden szám elején lesz körülbelül egy oldalnyi bevezető, amelynek célja néhány fogalom tisztázása az XML háza tájáról. A maradék oldalakat mindig valamilyen kézzelfogható, gyakorlatban azonnal alkalmazható példakódokkal fogom megtölteni. A két rész nem feltétlen fog szorosan összetartozni, de egy közös kapocs lesz közöttük: mindkettő XML-ről fog szólni. Emiatt előfordulhat, hogy lesz a cikkekben egykét fogalom, amit előtte nem definiáltam, de ilyenkor mindig hivatkozni fogok egy-egy URL-re, ahol a fogalomról részletes információ kapható. Vágjuk hát bele!

Sémaleírás

Adott az XML, mint általános, egyszerű és hatékony adatleíró nyelv. Önleíró, azaz egy XML dokumentumot minden külső információ nélkül értelmezni lehet, be lehet járni a benne található csomópontokat, kiolvasva az elemek értékét és attribútumait. Azonban irányított kommunikáció esetén, akár gépek közötti információcserénél, akár gép-ember kapcsolatban sokszor szabályokat kell alkotnunk a küldendő illetve fogadandó XML dokumentum szerkezetére vonatkozóan. Ekkor már csak azokat az XML dokumentumokat fogadjuk el érvényesnek, amelyek szerkezete megfelel a formális leírásban szereplő feltételeknek. Például egy megrendelést leíró XML dokumentumra valószínűleg kiköténk, hogy benne kell legyen a megrendelő neve, címe, adószáma satöbbi. Ha a kapott megrendelés.xml-ben nem szerepel minden kívánatos adat, akkor visszadobjuk a megrendelést, mert nem érvényes.

Az XML dokumentum szerkezetének, más néven sémájának leírására több módszer is a rendelkezésünkre áll, melyek fokozatosan, a használat során fejlődtek ki. Nézzük meg milyen hárombetűs rövidítések segítenek bennünket az XML dokumentumok sémájának leírásában:

- DTD, Document Type Definition: **[1]** a legelső eszköz volt az XML dokumentumok struktúrájának leírására. Több sebből is vérzik, de a legnagyobb problémája: nem XML formátumú. Ha már egyszer kitalálták az XML-t, nagyon gyors és hatékony feldolgozót *(parser)* írtak hozzá, akkor miért nem lehet az elemzendő XML dokumentum sémáját is XML-ben leírni? Emellett nincsenek benne adattípusok, mindent csak szövegként lehet definiálni, valamint nem támogatja a névtereket, amely nélkül szó sem lehet több forrásból összefűzött adatok ellenőrzésére. Ezek igen súlyos hiányosságok, amely miatt a DTD hamarosan ki fog halni, de egyenlőre sajnos egyedül ez az egyetlen, végleges, elfogadott sémaleíró nyelv.
- XDR, XML Data-Reduced: **[2]** a nagyreményű Microsoft DTD utód - volt. A Microsoft gyorsan lemondott a DTD használatáról, és gyors ütemben belekezdett egy XML formátumú sémaleíró nyelv kidolgozásába, amelyet a Word Wide Web konzorciumnak is elküldött szabványosításra. Megszületett az XDR. Amellett, hogy XML formátumú még jóval flexibilisebb is, mint a DTD. A DTD-ben leírt struktúrának maradéktalanul meg kell felelni egy XML dokumentumnak. Az XDR is tud ilyen szigorú lenni, de emellett elő lehet azt is írni, hogy az ellenőrizendő dokumentum egyes részeiben lehetnek további elemek is, amelyet a séma nem ír le. Például egy személyről szóló XML adatlapban kötelezővé tesszük a név, születési dátum és az anyja neve elemeket, de ezen felül megengedjük, hogy egy buzgó gazdi például a kutyája nevét és haja színét is beleszerkessze a dokumentumba. A hangsúly nem azon van, hogy előírhatunk opcionális elemeket, hanem azon hogy megengedhetünk olyan elemeket is, amelyekről a séma készítésekor még nem is tudtuk, hogy lesznek. Ehhez kapcsolódó szolgáltatás, hogy XDR segítségével le lehet szabályozni a dokumentum egy részét is, nemcsak a teljes egészet. DTD-vel természetesen csak az egész dokumentumra lehet szabályokat definiálni.

  Emellett az XDR bővíthető, azaz az igények megváltozásakor nem kell a sémát kidobni, csak egy másik névtér bevezetésével kiegészíteni a meglevőt. Utolsó, de nagyon fontos szolgáltatás az XDR-ben, hogy az elemek és attribútumoknak meg lehet adni a típusát *(egész szám, dátum satöbbi)*. A DTD-ben minden szövegként van deklarálva. A legtöbb, a közeli múltban fejlesztett Microsoft termék XDR-t használ sémaleírásra. Azonban már a kapuban dörömböl az
- XSD, XML Schema Definition **[3]**, amely a cikk írásának pillanatában a leginkább aktuális sémaleíró nyelv. A Biztalk Server, illetve a .NET XML osztályok már tudják – tudni fogják ezt a sémaleírást is kezelni *(természete-*

sen az XDR mellett). A Visual Studio 7 egyik alapszolgáltatása az XSD sémák grafikus szerkesztése, konverziója XDR-ből XSD-be, XML dokumentumból XSD generálása stb. *(A Visual Studio 7 illetve a .NET stratégia fejlesztői szemmel nézve több, mint szenzációs. Újságunkban is rövidesen megkezdjük az alapok lefektetését.)*

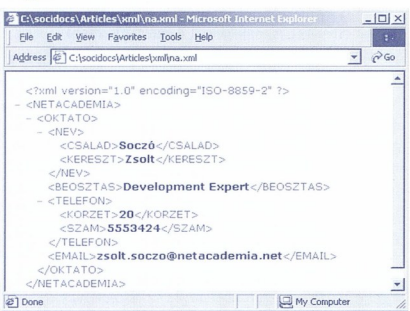
Az XML és a Web

A Microsoft nagy erőkkel ráállt az XML technológiában rejlő lehetőségekre, és ez az elkötelezettség meglátszik az utóbbi 3 év összes Microsoft termékén is. Az összes terméket áthatja az XML, legyen szó konfigurációs állományokról *(.NET-ben minden konfigurációs adat XML fájlokban van, hello registry – végre!)*, kiszolgáló termékekről, vagy az Internet Explorer-ről. Még azt sem tudtuk mi az az XML, de az IE4 már támogatta, persze az akkori szabványnak megfelelően még eléggé gyerekcipőben, de már működtek benne nagyon hasznos funkciók. A Microsoft Internet Explorer 5 már igen hatékony XML támogatást tartalmaz, amely segítségével az információk megjelenítéshez kapcsolódó funkciók jelentős részét ügyféloldalon meg lehet oldani, csökkentve ezzel a szerver terhelését. Ezt mondják Amerikában, ahol tényleg agyonterhelik a webkiszolgálókat a szörfözők. Ezzel szemben nálunk, ahol örül egy webszájt, ha van napi 5000 találata, inkább a lassú modemes kapcsolat miatti lecsökkent szerverhez fordulás az, ami miatt érdemes kihasználni az IE-ben rejlő lehetőségeket.

Persze általában az Internetes környezetben nem köthetjük ki a látogatóknak, hogy használjanak IE 5.5SP1-et, mert csak az tudja kihasználni az általunk megírt funkciót. Azonban heterogén böngészőkörnyezetben sem kell lemondani az XML kínálta előnyökről, csak buta böngésző esetén az XML-lel kapcsolatos funkciókat a kiszolgálón kell implementálni, és csak a kész HTML kódot leküldeni a böngészőknek. Okos böngészőnek megy az XML, butának a HTML. Az okos keveset fordul a webszerverhez, a buta sokat.

IE és XML, a két jó barát

Csapjunk bele az IE5 XML szolgáltatásaiba. Ha egy XML állományt adunk meg a böngészőnek, akkor ő azt közvetlenül megjeleníti.



Egy közönséges XML állomány az IE5-ben [4]

Figyeljük meg, az <?xml version="1.0" encoding="ISO-8859-2" ?> sort! Egyrészt leírja, hogy a dokumentum az XML1.0-s szabványra épül **[5]**, valamint azt, hogy a benne található karaktereket az ISO-8859-2-es kódtábla alapján kell értelmezni. Ha ezt nem tesszük meg, az összes értelmezőprogram, beleértve azt is, ami az IE mögött is van, kiborul az első ékezetes karakternél. Az IE5.5 így dohog:

```
An Invalid character was found in text content.
Line 6, Position 19
        <CSALAD>Socz?SALAD>
-------------------^
```

Jó, de ez így minden, csak nem felhasználóbarát. Alakítsuk át ezt HTML-é, amelyen keresztül már szépen, formázottan láttathatjuk az információkat. Két módszer kínálkozik erre, illetve egy harmadik, ami az első kettő keveréke.

1. Írjunk egy Cascaded Style Sheet-et *(CSS)* **[6]**, amiben leírjuk, hogy melyik elem hogyan jelenjen meg a böngészőben. Ez működő, de igen erősen behatárolt módszer, mert csak nagyon korlátozott lehetőségeink vannak a cél HTML dokumentum formátumának meghatározásában. Egyszerűen azért, mert a CSS nem erre van kitalálva. Arra nagyon jó, hogy leírjuk vele egy adott HTML elem megjelenését, de itt XML elemeket kellene transzformálni valamilyen, általunk definiált HTML formátumba, és erre a feladatra nem alkalmas a CSS. Viszont, erre van kitalálva az
2. Extensible Style Language, röviden XSL **[7]**. Az XSL pont arra van kitalálva, hogy tetszőlegesen bonyolult módon jelenítsünk meg egy XML dokumentumot, ami nevezhetnénk adatforrásnak is, kihangsúlyozva, hogy az nem hordoz semmiféle megjelenítési információt. XSL segítségével könnyedén megváltoztathatjuk az adatok, elemek eredeti sorrendjét, megszűrhetjük a benne található adatokat, ismétlődéseket vihetünk be, stb. Egyszóval XSL segítségével olyan HTML kimenetet generálunk, amilyet csak szeretnénk. Tulajdonképpen az eredeti XML dokumentumot bármi mássá is átkonvertálhatjuk vele, például egy másik XML dialektussá, azaz egy más struktúrájú XML dokumentummá, vagy éppen szöveges állománnyá! Amivé csak akarjuk. Nem véletlenül fejlődött ki az XSLT **[8]**, az XSL Transformations szabvány az XSL-ből. Az XSL esetén a kiinduló cél az XML adatok meg jelenítésének szabályozása volt - alternatíva a CSS mellé. Az XSLT már kifejezetten annak fényében készült, hogy az XML dokumentumokat valamilyen más formátumúra akarjuk átalakítani,transzformálni.

Van XSLT-nk, amivel pompásan át lehet alakítani a megjelenítendő XML dokumentumot HTML-é. Azonban egy bonyolultabb XSLT transzformáció igen áttekinthetetlen tud lenni, és ha még ebbe képeket, formázó parancsokat, stílusokat is beleszövünk, akkor egy olyan XML-HTML-formázó_parancsok spagettit kapunk, ami karbantarthatatlan lesz. Ekkor jön a nyerő ötlet. XSLT-vel transzformáljuk át a kívánatos HTML-é a forrás XML-t, és a formázásokra csak hivatkozunk benne, de ne fejtsük ki őket az XSLT-ben. Ehelyett az összes formázó utasítást rakjuk ki CSS-be, és máris ötvöztük az XSLT flexibilitását a CSS szuper formázási képességeivel. Mindeközben a grafikus bármikor nyugodtan átgyúrhatja a CSS-t, nem kell szembesülnie *(és elszaladnia :)* az XSLT-vel.

Az XSLT közbelép

Most, hogy kipletykáltuk a grafikust, készítsünk egy XSL dokumentumot, és adjuk meg a forrás XML-ünknek, hogy amikor a böngésző megjeleníti, használja az XSL-ünket. Ehhez az XML forrás elején megadjuk az XSL fájl elérését *(na.xsl)*:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<?xml-stylesheet type="text/xsl" href="na.xsl"?>
<NETACADEMIA>
...
```

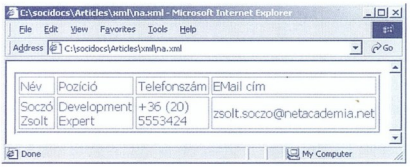
A böngésző az XML állomány betöltésekor megnézi, hogy van-e XSL hozzárendelve. Ha van, akkor letölti azt is, és végrehajtja az abban leírt műveleteket, majd megjeleníti a végeredményt. Nézzünk egy egyszerű transzformációt:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet xmlns:xsl="uri:xsl">
 <xsl:template match="/">                          (1)
   <HTML>
   <HEAD>
   <link rel="stylesheet"
    type="text/css" href="na.css" />               (0)
   </HEAD>
     <BODY>
       <TABLE CLASS="OktatokTabla">
         <TR>
           <TD>Név</TD>
           <TD>Pozíció</TD>
           <TD>Telefonszám</TD>
           <TD>EMail cím</TD>
         </TR>
         <xsl:for-each                             (3)
              select="NETACADEMIA/OKTATÓ>
           <TR>
             <xsl:apply-templates/>                (4)
           </TR>
         </xsl:for-each>
       </TABLE>
     </BODY>
   </HTML>
 </xsl:template>                                   (2)

 <xsl:template match="NEV">                        (5)
   <TD><xsl:value-of select="CSALAD"/>
   <xsl:value-of select="KERESZT"/></TD>
 </xsl:template>

 <xsl:template match="BEOSZTAS">                   (6)
   <TD><xsl:value-of select="."/></TD>
 </xsl:template>

 <xsl:template match="TELEFON">                    (7)
   <TD>+36 (<xsl:value-of select="KORZET"/>)
   <xsl:value-of select="SZAM"/></TD>              (9)
 </xsl:template>

 <xsl:template match="EMAIL">                      (8)
   <TD><xsl:value-of select="."/></TD>
 </xsl:template>

</xsl:stylesheet>
```

Hogyan hajtja vége a böngésző az imént felvázolt transzformációt? A azt jelzi, hogy a match-ben megjelölt elemre, ami jelen esetben a gyökér elem *(<NETACADEMIA>)* hajtsa végre a záró tagig leírt műveleteket. Mit jelent a végrehajtani? A nem XSL tagokat egyszerűen bemásolja a kimenetbe, így egészen a pontig minden bekerül a végeredménybe. Jól látható, hogy itt egy HTML táblázatot hoztam létre, egy fejléccel. Az XML forrásból származó tartalom generálása a körül történik. Az xsl:for-each arra utasítja az XSL értelmezőt, hogy a select után megadott XPath **[9]** kifejezés által kiválasztott elemeken menjen végig, és mindegyikre hajtsa végre az utasítás törzsében leírt parancsokat. A NETACADEMIA/OKTATO XPath kifejezés az összes, a NETACADEMIA elem alatt létrehozott OKTATO elemekre ad egyezést, azokat választja ki *(jelen esetben csak egyet, de ha több ismétlődő OKTATO elem lenne a forrásban, akkor mindegyiket)*. Mi történik a kiválasztott elemekkel? Generálódik hozzájuk egy <TR> </TR> páros, ami HTML-ben a táblázat egy sorát írja le. És a lényeg, a forrásadatok a kifejezés jóvoltából kerülnek bele a kimenetbe. Az xsl:apply-templates elem azt jelenti, hogy az éppen feldolgozás alatt álló elemre nézze meg, hogy van-e egyező sablon létrehozva. Ha van, akkor azt végrehajtja, és annak a kimenete beleszövődik a kifejezés helyébe. Esetünkben a paranccsal lefúrtunk az XML forrásunk <OKTATO> elemében, így az sablonok már erről a szintről indulnak, azaz a match attribútumokban megadott XPath kifejezések kiegészülnek így: /NETACADEMIA/OKTATO/TELEFON, mondjuk a sablon esetén. Másképpen fogalmazva a forrás XML-ben a /NETACADEMIA/OKTATO/TELEFON elérési úttal jelzett elem elérésekor az XSL processzor meghívja a sablont. A sablonon belül az xsl:value-of elem kiválasztja a select="KORZET" által kijelölt elem értékét, azaz a <KORZET></KORZET> közötti szöveget. Néhány egyéb formázó karakter és a SZAM elem értékének felolvasása után a sablon véget ér. Ezután az XSL motor megnézi, hogy van-e még más sablon, ami egyezést mutat valamelyik elemmel. Esetünkben mind a négy sablon szóhoz jut minden egyes oktatóhoz.

Miután végigment az összes elemen, kiíródik a táblázat vége, és véget ér a végrehajtás a ponton.

Aki ezt az utat lelkiismeretesen végigkövette, az megérdemli, hogy megnézze a végeredményt:



Mitől lett ez ilyen díszes és tarka *(mi lesz ebből a szürke újságban?! :)* ? Hát attól, hogy az XSL által generált HTML

kimenet kapott még egy CSS-t is a -val megjelölt sorban. Anélkül csak egy fehér táblázatot látnánk. A teljesség kedvéért itt az na.css tartalma:

```
<STYLE>
BODY
{
    BACKGROUND-COLOR: snow
}
TABLE.OktatokTabla
{
    BORDER-RIGHT: groove;
    BORDER-TOP: groove;
    BORDER-LEFT: groove;
    COLOR: mediumblue;
    BORDER-BOTTOM: groove;
    FONT-FAMILY: Tahoma;
    BACKGROUND-COLOR: lightgoldenrodyellow
}
TD
{
    BORDER-RIGHT: thin groove;
    BORDER-TOP: thin groove;
    BORDER-LEFT: thin groove;
    BORDER-BOTTOM: thin groove
}
</STYLE>
```

Az előbbi XSL transzformáció egy nagyon egyszerű példa volt, ennek ellenére már ez is elég átláthatatlan és bonyolult lett. Az XML technológia bonyodalmaiba akkor kóstolunk bele igazán, amikor az XSLT nyelvet kezdjük el használni. A gond az, hogy az XSLT nem procedurális nyelv, hanem alapvetően sablonok egyeztetésén alapul. Ez a fajta gondolkodás igen szokatlan egy Basic vagy C nyelven felnőtt programozónak.

További nehézség, hogy mit tehetünk akkor, ha egy jó bonyolult XSLT transzformáció nem úgy működik, ahogy azt elvárjuk tőle? Nem formátumbeli hibákra gondolok, azt kiszűrik a transzformáló komponensek. Akkor leszünk meleg helyzetben, ha „csak" logikai hiba van az XSLT-nk működésében, és nem azt a transzformációt hajtja végre, amit mi módoltunk ki. Ilyenkor jönnek képbe a nyomkövető vagy debugger programok, amelyek segítségével lépésenként lehet végrehajtani a programunkat, ebben az esetben az XSL transzformációt. Ez egy kemény feladat, de szerencsére az Activestate cégnek - amely elsősorban a Win32-es Perl eszközeiről híres – megjelent a Visual XSLT 1.0 **[10]** programja, ami egy plug-in a Visual Studio.NET 7-hez, és amelynek segítségével szerkeszthetjük és nyomkövethetjük az XSL transzformációinkat. Szenzációs termék! A debugger ablakban láthatjuk a transzformáló XSLT-t, benne az éppen végrehajtás alatt álló XSL paranccsal. A Watch ablakban láthatjuk a forrás XML éppen feldolgozás alatt álló elemeit, az Output window-ban pedig a transzformált végeredményt.

A program jelenleg Beta 1 állapotban van, hasonlóan a Visual Studio.NET 7-hez. Ennek ellenére aki komolyan, a jelenben és a jövőben élve szeretne XML-lel foglalkozni, mindenképpen szerezzen be egy példányt a Visual Studio.NET 7-ből, és töltse le a Visual XSLT 1.0-t. Mindkettő Must Have!

Újdonságok

Az XML-t alkalmazó alkalmazásokat programozók legalapvetőbb fegyvere a parser, XML értelmező, amely az XML Document Object Model-en keresztül programozottan elérhetővé teszi az XML dokumentumok belső szerkezetét. A parser-ek lehetővé teszik a felolvasandó XML dokumentumok ellenőrzését egy megadott sémával szemben, amely jelen pillanatban – Microsoft-os platformon – DTD és XDR-el történhet. Az aktuális parser a Microsoft-tól a 3-as verziónál jár, ez az, amit éles környezetben is lehet használni.

Azonban nincs megállás. Áprilisban a Microsoft kiadta a Microsoft XML Parser *(MSXML)* 4.0 Technology Preview Release-t. Ezt még nem ajánlják produkciós környezetbe, de mindenképpen érdemes tanulmányozni. Miért? Azért, mert végre támogatja az XSD-t! A parser-ben implementált séma validáló kód a Wide Web Consortium *(W3C)* XML Schema, 2001. március 30-ikai ajánlásában leírt módon működik, azaz olyan friss, hogy még meleg! Mivel ez még mindig nem a végleges XSD szabvány, így várható, hogy az valamennyit még változni fog a végleges kiadásig. Az XML parser 4 természetesen követni fogja a változásokat, újabb és újabb kiadásokkal.

Mi van még a csomagban az XSD mellett? Megjelent benne egy új objektum, az MXHTMLWriter, amely segítségével a SAX API által felolvasott XML dokumentumból kapott adatfolyamot lehet HTML-lé transzformálni. Ez egy alternatív megoldás lesz az XML --> XML DOM --> XSLT --> HTML feldolgozásra, kifejezetten nagyteljesítményű webalkalmazásokhoz.

Az új verziót fel lehet telepíteni olyan gépre is, amin már van MSXML3-as parser, így az alkalmazáaok a verzió függő ProgID-k használatával tudják használni mind a 3-as, mind a 4-es változatot. Arra viszont vigyázzunk, hogy ne rakjuk fel produkciós szerverre, mert a verziófüggetlen objektumlétrehozást alkalmazó programok az új verzióból fognak egy példányt kapni, ami meglepetéseket okozhat.

Zárszó

Aki lelkiismeretesen végiglépkedett a példa XSL-en és XML-en, annak már van fogalma az XML alapú fejlesztések alapköveiről, látja a jéghegy csúcsát. De az igazi csemege még a víz alatt van. A következő számban a transzformációt már nem fogjuk az Internet Explorer-re bízni, hanem a kezünkbe vesszük az XML DOM-ot, és azzal hajttatjuk végre az XSL-ünket, mind kiszolgáló oldalon, mind ügyfél oldalon. És ha már ott vannak az adatok a böngészőben, miért ne lehetne rögtön szűrni illetve sorbarendezni őket, anélkül, hogy a szerverhez kellene fordulni.? Miért is ne? A júniusi számban megmutatom hogyan.

Soczó Zsolt MCSE, MCSD, MCDBA
Zsolt.Soczo@netacademia.net

A cikkben szereplő URL-ek:

[1]: DTD szabvány http://www.oasis-open.org/cover
[2]: XDR szabvány http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm
[3]: XSD http://www.w3.org/XML/Schema
[4]: Példakódok http://technet.netacademia.net/feladatok/xml
[5]: XML1.0 szabvány, magyarázattal! http://www.xml.com/axml/testaxml.htm
[6]: CSS szabvány http://www.w3.org/Style/CSS
[7]: XSL szabvány http://www.w3.org/Style/XSL
[8]: XSLT szabvány http://www.w3.org/TR/xslt
[9]: XPath szabvány http://www.w3.org/TR/xpath
[10]: Visual XSLT 1.0 Beta
http://www.activestate.com/ASPN/Downloads/VisualXSLT

XML-gessünk *(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 lóerővel 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 elhományosította 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 jelenti.
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 záloga.)*
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 zseniális 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 befoglaló 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>
```

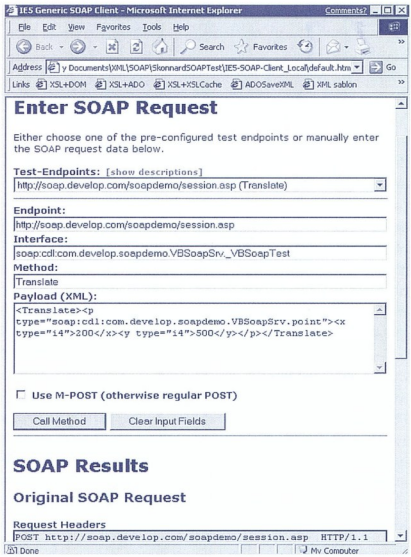
Láthatjuk a szabványos HTTP státuszkódot *(sikeres kiszolgálás - 200)*, és a borítékba zárt választ, amely ugyanabban a névtérben van, mint a kérés. A SOAPMethodName fejlécre most nincs szükség, ezért nincs is jelen.

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 környezetben 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

MIME – levelezés az interneten *(III. rész)*

Az elmúlt két számban megismerkedtünk a MIME üzenetek néhány fontos tulajdonságával. Tudjuk, milyen korlátokat szab az SMTP protokoll, ismerjük a MIME karakterkészletekre és tartalomtípusokra vonatkozó bővítményeit; most végre következhetnek a több részből álló üzenetek, amelyek a MIME igazi erejét adják. Az egyes RFC-k címét nem írjuk le, minden RFC ugyanúgy, az **[1]** módon leírt címmel érhető el.

A MIME Browser

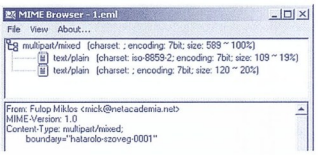
Aki Windows 2000 operációs rendszerrel rendelkezik, a **[2]** címről letöltheti a MIME Browser nevű programocskát, ami MIME formátumú levelek tartalmának vizsgálatára használható demonstrációs eszköz. A telepítés után futtassuk a programkönyvtárban található regshell.vbs scriptet, ekkor az .eml fájlok *(elmentett MIME formátumú levelek)* kiugró menüjében megjelenik a „Browse MIME Structure" menüpont, amire kattintva elindul a MIME Browser, és megkísérli betölteni az adott állományt. A cikk olvasása során ez az eszköz folyamatosan nagy segítséget nyújt majd a levélformátumok megértésében, ellenőrzésében. Annak sem kell izgulnia, aki Windows 2000 híján nem tudja használni a programot, mert az újság hasábjain ábrákkal megpróbáljuk szemléltetni a látnivalókat. Ugyaninnen, a **[2]** címről lehet letölteni a cikkben található példa-leveleket is, amelyeket bárki – még a MIME Browser hiányában is – kipróbálhat. Az .eml fájlokra kettőt kattintva az alapértelmezett levelezőprogram megjeleníti a levél tartalmát, ha pedig Notepad-del megnyitjuk őket, tanulmányozhatjuk a forráskódjukat.

Több részből álló üzenetek

A MIME üzenet több részből is állhat. Az SMTP szabvány szempontjából persze továbbra is egyetlen, csupaszöveg állományról van szó, a levél részekre bontását a szövegben található határolókarakterek *(vagy inkább „határolómondat")* segítik. Egy nagyon egyszerű, mindössze két szöveges részt tartalmazó MIME levél egyszerűsített forráskódja így néz ki *(1.eml)*:

```
From: Fulop Miklos <mick@netacademia.net>
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="hatarolo-szoveg-0001"

Ez egy magyarazo szoveg nem MIME programok
reszere. Ha ezt latja, ideje haladni a korral!

--hatarolo-szoveg-0001
Content-Type: text/plain

Ez az elso resz torzse

--hatarolo-szoveg-0001
Content-Type: text/plain

Ez a masodik resz torzse

--hatarolo-szoveg-0001--
```

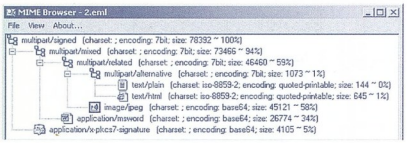
Ez az üzenet a demonstrációt leszámítva semmire sem használható, mert itt csak a MIME megértéséhez szükséges fejléceket tüntettük fel. A szokásos és elhagyhatatlan „MIME-Version" fejléc mellett feltűnik egy eddig ismeretlen tartalomtípus-főcsoport *(Content-Type)*, ez pedig a multipart. Ez a tartalomtípus jelzi a program számára, hogy az üzenet tartalma több részből áll, azt fel kell darabolni és egyenként, mint üzenetrészeket *(angolul bodypart)* kezelni. Az üzenetrészekről még később szólunk, előbb lessük meg a darabolás módját: A Content-Type fejlécnek van egy boundary paramétere *(esetünkben ennek értéke „hatarolo-szoveg-0001")*. Ahol ez a szöveg feltűnik az üzenetben *(mindig egy sorban, két mínuszjel után, lásd ①)*, ott képzeletben fogjunk egy ollót, és nyisszantsuk el a szöveget. A legelső előfordulás előtti részt *(②)* mi, MIME-ul tudók eldobjuk. Aki viszont nem ismeri a MIME-ot, annak ez fog legelőször megjelenni, ezért ez a hely ideális a régebbi levelezőprogramoknak szóló kedves üzenet elhelyezésére. A több *(nem feltétlenül két-)* részből álló üzenet utolsó részének végén található határolószöveg annyiban különbözik a többitől, hogy annak a végén is találunk két mínuszjelet *(③)*.



A levél a MIME Browserben. Jól látszik, hogy a levél két részből áll; és a részek text/plain típusúak

Az üzenetrészek

A szétszabdalt üzenet részei külön-külön egy-egy igazi, „nagy" SMTP levélhez hasonlítanak. Mindegyik üzenetrész *(bodypart)* fejrészt *(fejléceket)* és törzset tartalmaz, a két fő összetevőt pedig egy üres sor választja el. Az üzenetrész fejlécei között most persze nem a teljes levélre, csak az adott üzenetrészre vonatkozó MIME-fejlécek találhatók *(tartalomtípus, karakterkészlet, kódolás, stb.)*. Miután minden üzenetrész saját fejlécekkel rendelkezik, semmi akadálya annak, hogy az üzenet egyik része mondjuk cirill betűs szöveg, a másik pedig mondjuk egy magyar nyelvű HTML dokumentum legyen. Sőt! Az üzenetrész önmaga is lehet egy több részből álló *(multipart)* üzenetrész, saját, egyedi határolószöveggel – ilyenkor fogjuk a kisollót, és az üzenetrészt tovább daraboljuk. Akármilyen bonyolult MIME-fát építhetünk, álljon itt egy valóságból kölcsönzött, bár nem az egyszerűbbek közül való példa *(2.eml)*:



HTML formátumú, beágyazott képet és egy Word csatolt állományt tartalmazó, végül digitálisan aláírt levél felépítése

Gyakorlatlanok ne próbálják az ábrát rögtön, részletekbe menően megérteni – a migrén fájdalmas lehet, és ráadásul lassan is múlik el, – egyelőre a hierarchiát próbáljuk meg átlátni. Figyeljük meg a négy egymásba ágyazott multipart üzenetrészt, és azt, hogy ki kinek a gyermeke. A második „szinten" található multipart/mixed üzenetrész például egy Word dokumentumot és másik multipart üzenetrészt tartalmaz, ami egy újabb multipart-ból és egy JPEG képből áll – és ez még csak a hierarchia fele!

Üzenetrész-fejlécek

A multipart tartalomtípus is „csak" egy főcsoport, és ennek a főcsoportnak is több alcsoportja létezik. Mielőtt azonban a multipart-alcsoportokra rátérnénk, bemutatjuk a leggyakoribb, multipart-alcsoporttól függetlenül használt fejléceket.

A Content-Type üzenetrész-fejléc

Ez talán a legfontosabb információ: az üzenetrész által hordozott adat típusát határozza meg *(ami ugye lehet valamelyik hagyományos, az előző számban már bemutatott tartalomtípus, vagy akár egy másik multipart üzenetrész is)*. Szöveges tartalomtípus esetén a legtöbb esetben ez a fejléc rejti az üzenetrészben használt kódtábla azonosítóját *(charset)*:

```
Content-Type: text/plain;
        charset="iso-8859-2"
```

A name paraméterben pedig gyakran hordozza az üzenetrész által tartalmazott fájl eredeti nevét, például:

```
Content-Type: application/msword;
        name="doc1.doc"
```

Végül ne feledkezzünk el arról, hogy a multipart tartalomtípus esetén itt kell megadnunk a darabolás helyét jelző szöveget *(boundary)*:

```
Content-Type: multipart/mixed;
        boundary="hatarolo-szoveg-0001"
```

A Content-Type fejlécnek e háromnál sokkal többféle paramétere lehet, ezeket a paramétereket mindig maga a tartalomtípus határozza meg. A különböző multipart tartalomtípusok paramétereiről a megfelelő helyen mi is említést teszünk, egyéb esetben az RFC-k böngészése segíthet.

A Content-Transfer-Encoding üzenetrész-fejléc

Csakúgy, mint a teljes üzenetre vonatkozóan *(lásd a teljes leírást az előző számban)*, üzenetrész-fejlécként használva is az üzenetrész-tartalom 7 bitre kódolásához használt módszert jelzi. Üzenetrészenként használva megoldható, hogy az üzenet szöveges részeit a Quoted-Printable, míg a bináris részeket *(mondjuk egy csatolt word dokumentumot)* az ilyen tartalomnál hatékonyabb Base64 kódolással vigyük át. Fut még a 7bit kódolás is, ami a nemkódolást jelenti *(ASCII szöveg esetén)*. Példák:

```
Content-Transfer-Encoding: quoted-printable
Content-Transfer-Encoding: base64
Content-Transfer-Encoding: 7bit
```

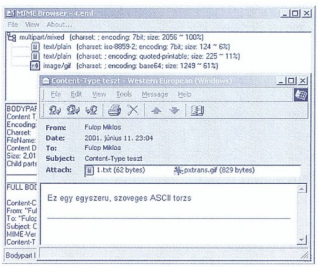
A Content-Disposition üzenetrész-fejléc

No, ez a fejléc már teljesen új. Az üzenetrészre aggatott Content-Disposition fejléc jelzi, hogy az egy csatolt állomány *(ez esetben értéke attachment)*, vagy a levélben valahol felhasznált komponens *(ilyenkor inline)*. Az inline típusú üzenetrészt a levelezőprogramok elvileg nem jelenítik meg, mint csatolmányt. Próbáljuk csak ki, nyissuk meg a 3.eml példát! A levelezőprogram a három üzenetrészből az elsőt, a levél törzsét megjeleníti; a másodikat csatolt fájlként jelzi *(kattintással megnyithatjuk, ha akarjuk)*, míg a harmadik ismét, felhasználói közbeavatkozás nélkül látható *(hiszen „része" az üzenetnek)*. Ha a Content-Disposition értéke attachment, akkor a fájl későbbi kezelésére vonatkozó információs paramétereket is tartalmazhat: mindenekelőtt a fájl nevét *(filename)*, esetleg a méretét, vagy a létrehozásának dátumát, például:

```
Content-Disposition: attachment;
        filename="1.txt"
```

Ebből tudja a levelezőprogram, mi volt a csatolt fájl eredeti neve. Ha ez a paraméter hiányzik, a program kénytelen a hasára csapni; ilyenkor kapunk ilyen fájlneveket, hogy: att00891.txt vagy hogy att00915.gif.
Hogy a kiterjesztést honnan tudja mégis? Ennek megfejtése házi feladat!

Nos körülbelül ennyit az általános üzenetrész-fejlécekről. A továbbiakban a különféle multipart-tartalomtípusokról, és az általuk használt, speciális üzenetrész-fejlécekről értekezünk. Mielőtt beleugranánk a mélyvízbe, ellenőrizzük az általános üzenetrész-fejlécekről összegyűjtött tudásunkat a 4.eml példa segítségével!



Egyszerű, csatolt fájlokat tartalmazó levél – figyeljük meg az üzenetrészeknél használt különféle kódolási módszereket (encoding)

A multipart-alcsoportok

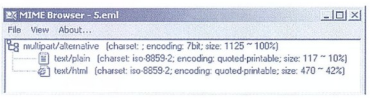
Futólag már említettük, hogy a multipart tartalomtípus-főcsoport különféle alcsoportokat tartalmaz. Az rendben van, hogy az üzenet több részből áll, de van-e a részeknek valami közük egymáshoz? Az alcsoport szerepe, hogy megjelölje az általa tartalmazott üzenetrészek egymáshoz való viszonyát. Mindegyik tartalomtípusnak különleges szerepe van, és a legtöbb saját üzenetrész-fejléceket is használ.

A multipart/mixed tartalomtípus

A legáltalánosabb, legegyszerűbb és talán leggyakoribb típus *(RFC 1521)*. A részeknek nincs szorosabb közük egymáshoz, hacsak nem az, hogy egy üzenetbe kerültek. Tipikus példa az egyszerű, csatolt állományokat tartalmazó üzenet *(mint az eddigi példák többsége, például a legutolsó 4.eml is)*.

A multipart/alternative tartalomtípus

Ez egy nagyon érdekes típus *(RFC 1521)*; szerepe a HTML levelek megjelenésével vált egyre fontosabbá. Az alternatív üzenetrészek ugyanazt tartalmazzák, különböző formátumban; közülük elég azt megjelenítenünk, ami a szívünknek kedves. Lássuk csak egy egyszerű, tisztán szöveget tartalmazó, de HTML levél felépítését *(5.eml)*:

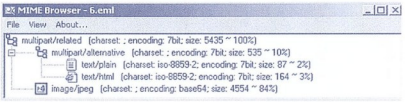


HTML levél, text alternatívával

Egy jólnevelt HTML levél egy text/plain és egy text/html részt tartalmaz, ahol a text/plain üzenetrész a HTML rész szöveges kivonatát tartalmazza. Mi dönthetünk: ha nem akarjuk a HTML-t megjeleníteni, ott a szöveges verzió. Ez persze azt jelenti, hogy a levél tartalma redundáns, viszont levelezőprogram-barát. Látható, hogy az üzenet teljes terjedelmének 42%-át a HTML üzenetrész teszi ki, a plusz szöveges rész csak további 10% *(a 100-ból fennmaradó részt a levél SMTP fejlécei képezik; ez az arány persze levelenként eltérő)*. A többlet terhelés tehát jelen van, de általában nem számottevő.
Fontos megjegyezni, hogy az alternatív tartalomtípus sem csak két verziót tartalmazhat; előfordulhat akár egy text-rtf-word6-word2000 csomag is. Az RFC 1521 mindössze annyit kér, hogy a legegyszerűbb formátum *(esetünkben a sima szöveg)* legyen a levélben legelöl, majd növekvő bonyolultsági sorrendben következhetik a további verziók. Így sokkal könnyebb a feldolgozás azok számára, akik csak valamely „gyengébb" formátumot ismerik.

A multipart/related tartalomtípus

Egymáshoz kapcsolódó üzenetrészek *(általában az egyikben hivatkozás történik a másikra)* azonosítója *(RFC 2112)*. Egyszerűbb megérteni, ha magunk elé képzelünk egy HTML üzenetet, ami egy képet tartalmaz *(tehát nem csatolt állományként, hanem beillesztve)*. A kép is az üzenetben utazik, a HTML kódban pedig speciális hivatkozás szerepel a helyén. Ha megnyitunk egy ilyen levelet, a kép az üzenet belsejében jelenik meg, és nem érjük el, mint csatolt állományt. Természetesen maga a HTML rész sem mint tiszta HTML, hanem mint komplex multipart/alternative rész utazik *(lásd az előző bekezdést)*; egy beillesztett képet tartalmazó HTML levélben tehát már felismerhetjük az első hierarchikus jellemzőket:



HTML levél, melyben kép is van

Egy kérdés nyitott maradt: arról volt szó, hogy a HTML tartalom hivatkozik a levélbe ágyazott képre. Pontosan hogyan is történik ez? Lássuk a 6.eml forrásának erre vonatkozó részeit *(mondjuk a HTML üzenetrészt teljes egészében ④, és a kép üzenetrészének elejét ⑤)*:

```
...
--hatarolo_szoveg_0001                                    ④
Content-Type: text/html
Content-Transfer-Encoding: 7bit

<HTML><HEAD></HEAD>
<BODY>
<IMG src="cid:content_id_0001">
</BODY></HTML>                                            ⑦

--hatarolo_szoveg_0001--

--hatarolo_szoveg_0000                                    ⑤
Content-Type: image/jpeg;
        name="MSGIRL2.jpg"
Content-Transfer-Encoding: base64
Content-ID: <content_id_0001>                             ⑥

/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAAYEBQYF...
FhYaHSUfGhsjHBYWICwgIyYnKSopGR8tMC0oMCUo...
...
```

A határoló szövegeket próbáljuk meg magunk értelmezni, de ehhez a teljes kódot látni kell! *(Annyit segítek, hogy a _0001 végű a multipart/alternative, a _0000 végű pedig a multipart/related határolószövege.)* A lényeg most az üzenetbe ágyazott elemek „címzése". Figyeljük meg, hogy a képet tartalmazó üzenetrész tartalmaz egy Content-ID fejlécet ⑥:

```
Content-ID: <content_id_0001>
```

A fejléc értéke az adott üzenetrész azonosítója *(a kacsacsőrök nem tartoznak az azonosítóhoz)*.
A HTML kódban pedig a kép forrásaként nem internetes URL *(például http://...)*, hanem egy speciális címzés szerepel *(egyébként ez is egyfajta URL)* ⑦:

```
<IMG src="cid:content_id_0001">
```

A megjelenítéskor a levelezőprogram a cid: bevezetőből tudja, hogy a hivatkozott elemet az üzeneten belül, a megadott Content-ID alapján kell megkeresnie.
A multipart/related tartalomtípusnak van egy kötelező type paramétere, ami a „gyökérként" feldolgozandó üze-

netrész tartalomtípusát tartalmazza *(annak az üzenetrésznek, amelyre nem hivatkoznak, hanem ő hivatkozik)*; esetünkben ez a multipart/alternative:

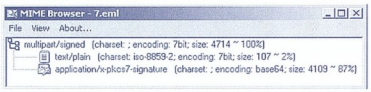
```
Content-Type: multipart/related;
        type="multipart/alternative";
        boundary="hatarolo_szoveg_0000"
```

A tartalomtípus többi paramétere opcionális, akit érdekel, olvassa el a témáról szóló RFC 2112-t.

A multipart/signed és multipart/encrypted tartalomtípusok

Digitálisan aláírt *(signed)* / titkosított *(encrypted)* üzenetek *(RFC 1847)*. Kezdjük a végéről: a multipart/encrypted, azaz titkosított többrészes üzenet, amely mindig pontosan két részből áll. Az első a titkosításhoz szükséges kontrollinformáció, a második pedig maga a titkosított adatfolyam, amely application/octet-stream *(8 bites adatfolyam)* tartalomtípust hordoz. A manapság használatos titkosítások esetén általában a multipart/encrypted tartalomtípus nem szerepel, az S/MIME levelezőprogramok elintézik a dolgot egyetlen önhordó *(még csak nem is multipart)* titkosított leveléllel *(amelynek tartalomtípusa nemes egyszerűséggel application/x-pkcs7-mime)*.

Viszont az aláírt levelek! A 7.eml egy egyszerű digitálisan aláírt szöveges üzenet. A multipart/signed is mindig pontosan két üzenetrészt tartalmaz.



Digitálisan aláírt szöveges üzenet

Ebből az első a digitálisan aláírt, „védett" üzenetrész *(bármi lehet, egyszerű szövegtől az összetett multipart üzenetrészekig)*; a második pedig az aláírás maga, tartalomtípusa az aláírás algoritmusától függ. Ha belenézünk a forráskódjába *(most már ideje lenne letölteni azt a MIME Browsert, nem?)*, a fő tartalomtípus fejlécénél ezt látjuk:

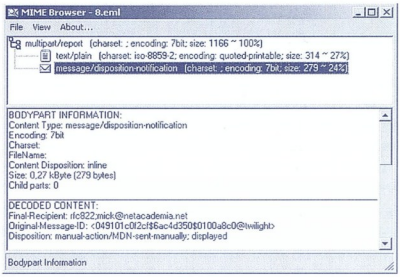
```
Content-Type: multipart/signed;
        protocol="application/x-pkcs7-signature";
        boundary="hatarolo-szoveg-0000";
        micalg=SHA1
```

Jól látható a tartalomtípus három kötelező paramétere. Az egyértelmű határolószöveg *(boundary)* mellett meg kell adni a digitális aláírás protokollját *(protocol, ez lesz a második üzenetrész tartalomtípusa is)*, valamint az ellenőrzőösszeg képzéséhez használt algoritmus *(Message Integrity Check Algorythm, micalg)* nevét. A fejléc tanulsága szerint a példában látható digitális aláírás az SHA1 algoritmus segítségével készült. Ha az első, védett üzenetrészben bármit *(akár egy üzenetrész-fejlécet is)* megváltoztatunk, a digitális aláírás azonnal „elromlik" és kiderül a turpisság.

A multipart/report tartalomtípus

Bizonyára mindannyian kaptunk már valamelyik levelezőkiszolgálótól a levelünk sikertelen vagy akár sikeres kézbesítésére vonatkozó válaszlevelet, esetleg visszajelzést arról, hogy a címzett elolvasta a levelet. Azt már kevesebben tudják, hogy ezeknek a leveleknek speciális formátumuk van, elősegítve ezzel az ilyen üzenetek automatikus feldolgozását.

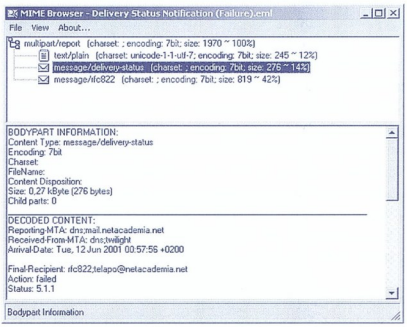
Válasszuk ketté a multipart/report *(RFC 1892)* tartalomtípust! Nézzük meg először, hogyan néz ki egy elolvasást visszaigazoló üzenet *(read receipt, 8.eml)*:



Read receipt

A multipart/report első tagja egy egyszerű szöveges üzenet arról, hogy elolvastam a magamnak küldött levelet. A második tag pedig egy message/disposition-notification tartalomtípusú üzenetrész *(RFC 2298)*, aminek tartalma elárulja az eredeti üzenet azonosítóját *(a levelezőprogramunk tudná követni, hogy melyik levélről érkezett visszajelzés!)*, hogy végülis ki kapta meg az üzenetet *(például ha átirányítás történt)*, illetve a visszajelzés elküldésének körülményeit *(nem automatikusan, hanem az én megkérdezésemmel történt)*.

Lássunk egy kézbesítési üzenetet *(Delivery Report)*, amelyben a levelezőkiszolgálónk értesít róla, hogy a Télapónak egyelőre nem a NetAcademia-hoz kell címezni a leveleket:



A Télapó nem nálunk dolgozik...

Az első rész itt is egy szöveges üzenet a felhasználónak. Az üzenetben szerepel még a szóban forgó levél egy-az-egyben *(message/rfc822, lásd RFC 1521)*, valamint maga a kézbesítésről szóló üzenet *(message/delivery-status, RFC 1894)*.

Most lapozzon vissza a 30. oldalra és tekintse meg még egyszer azt a bizonyos HTML levelet, amely beágyazott képet és egy csatolt Word dokumentumot tartalmaz, és a végén még jól alá is írtuk. Ugye, hogy mégsem annyira bonyolult?

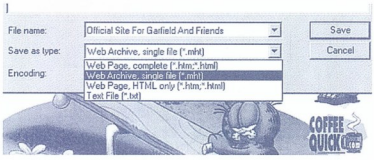
Egy csendes megjegyzés: a MIME annyira jól ki van találva, hogy az is képes a leveleket olvasni, aki kénytelen csupán karakteres (linux/unix) környezetben dolgozni. Még a fenti, igencsak összetett példa is tartalmaz tisztán szöveges összetevőt! Mindössze egy rendes, MIME-kompatibilis levelezőprogram kellene hozzá...

Az MHTML dokumentum

Egy érdekességet szeretnék mutatni a MIME-ról szóló cikksorozat végén, aminek érdekes módon a levelezéshez semmi, a MIME-hoz viszont annál több köze van. A most leírtakat az Internet Explorer 5.0 vagy újabb változatának felhasználói már ismerhetik *(illetve ha még nem ismernék, akkor kipróbálhatják)*, aki nem rendelkezik ezzel a verzióval, annak itt az ideje frissíteni. Kezdjük az alapproblémával: egy HTML oldal tartalmát szeretnénk elmenteni, későbbi olvasgatásra. A klasszikus módszer valami ilyesmit eredményez:

Ha egy weblapot elmentünk, a HTML fájl mellett a „sallangok" egy külön könyvtárba kerülnek

Van eset *(a Garfield például pont ilyen :-))*, hogy nem gond, sőt előny, ha egy weblapon található képek, grafikák külön-külön a lemezre íródnak. Máskor azonban kifejezetten macera a fájl és a hozzá tartozó könyvtár együttes másolgatása *(próbáljuk ki, és hozzunk létre egy könyvtárat és egy HTML fájlt, mint a fenti példában; azután töröljük ki a HTML fájlt, és csodák csodájára a Windows a megkérdezésünk nélkül törli a (szerinte) hozzá tartozó könyvtárat is .. khm..)*. Egyszóval ezzel sokszor csak a baj van.
Nem lehetne összecsapni az egészet egyetlen fájlba? *(Na jó, nem a tömörítésre gondolok...)*. Dehogynem. Az Internet Explorer 5.0-tól kezdve a Save As... ablakban egy eddig ismeretlen, új formátumot is kiválaszhatunk:

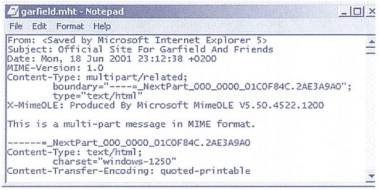


IE5-től kezdve Garfield-ot .mht-ba is menthetjük

„Web Archive, single file" *(webarchívum, egyetlen fájl)* – mondja a magyarázat az .mht kiterjesztés mellett. Nosza, próbáljuk ki! A mentés végén valóban egy fájl keletkezik *(néhányat a [2] címről is letölthet a lelkes Olvasó)*. Hurrá! Semmi sallang!

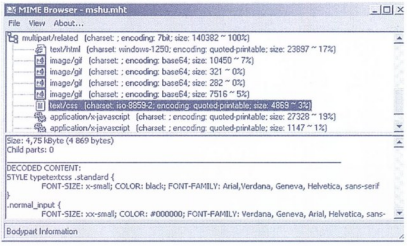
Na de mi köze ennek a MIME-hoz?!

Rögtön meglátjuk, ha egy .mht fájlt a jó öreg MIME Browser-be vontatunk *(persze akkor is kiderül a turpisság, ha egyszerűen megnyitjuk, mondjuk NotePad-del)*. Az .mht fájl *(MHTML dokumentum)* nem más, mint egy MIME formátumú HTML „levél", beágyazott objektumokkal.



Az .mht fájl valójában egy HTML levél, az M betű a MIME-ot jelképezi: MHTML Document

Még feladója, feladási dátuma és témája is van! És hogy mi van belül? Rögtön meglátjuk, ha megnyitunk egy ilyen fájlt a MIME Browserrel:

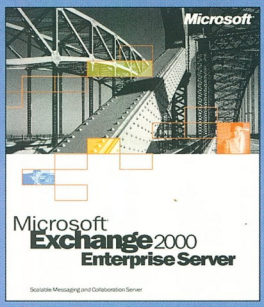


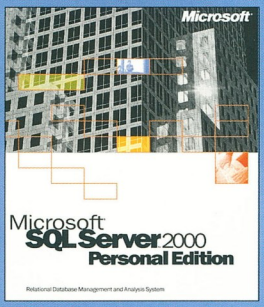
A Microsoft Magyarország elmentett weblapja a MIME Browserben

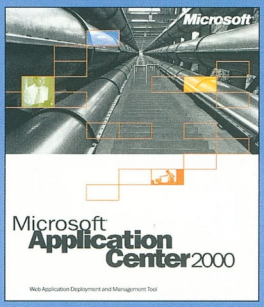
Az ábrán a Microsoft Magyarország .mht-ba mentett főlapjának kicsit kozmetikázott képe látható *(az itt láthatónál sokkal több kép van a fájlban, de ez ne zavarjon meg senkit)*. Az üzenet típusa értelemszerűen multipart/related *(aki nem érti miért pont ez, lapozzon vissza)*; az első elem természetesen maga a HTML kód, majd azt követik szépen a hozzá kapcsolódó üzenetrészek: képek, stíluslapok, javascript-darabkák.

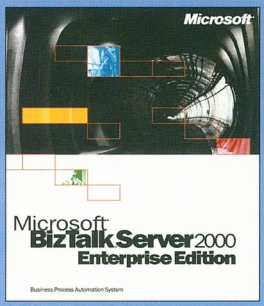
Fülöp Miklós
mick@netacademia.net

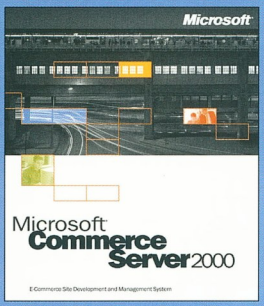
A cikkben található URL-ek:
[1] RFC xxxx: http://www.ietf.org/rfc/rfcxxxx.txt
[2] http://technet.netacademia.net/download/MIME

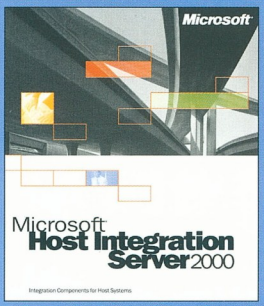


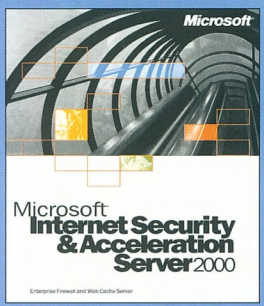


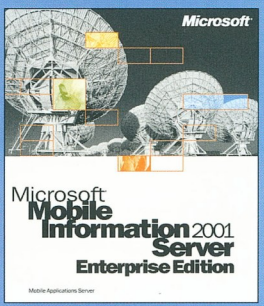


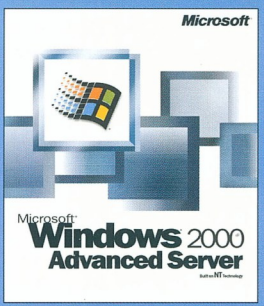


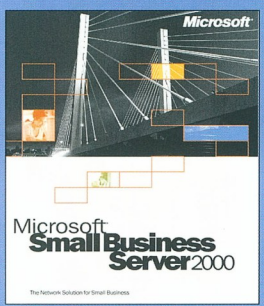


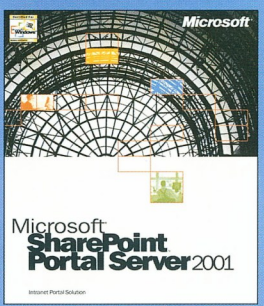


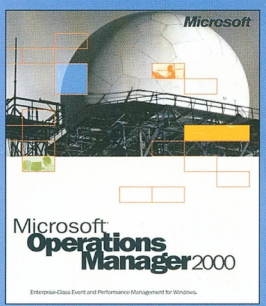












Microsoft TechNet

Microsoft