Microsoft Windows NT 4.0

Cu cateva luni in urma Microsoft a distribuit al doilea beta a unei noi versiuni a celui mai apreciat sistem de operare al sãu, Windows NT. Noua versiune se va numi Windows NT 4.0 si va combina interfata frumoasã, flexibilã si usor de utilizat specificã Windows 95 cu stabilitatea, puterea si viteza cu care am fost obisnuiti în vechile versiuni de Windows NT. Window NT 4.0 este cu mult mai mult decât o simplã aranjare cosmeticã a codului, versiunea aducând o serie de facilitãti suplimentare, îndelung asteptate de cãtre utilizatorii de Windows.

Înainte de a intra în subiectul propriu-zis, si anume prezentarea noii versiuni, sã precizãm încã o datã care este viziunea Microsoft asupra rolului sistemelor sale de operare. Cele trei sisteme principale de la Microsoft sunt Windows, Windows NT Workstation si Windows NT Server. Microsoft a avut ambitia sã dezvolte un set de API-uri comune pentru toate aceste trei sisteme de operare, în asa fel încât sã putem scrie o aplicatie o singurã datã si sã o putem rula pe oricare dintre aceste sisteme de operare. Telul a fost cu atât mai greu de atins cu cât Windows NT ruleazã atât pe procesoare Intel cât si pe procesoare RISC precum procesoarele MIPS, Alpha sau PowerPC.

Dincolo de aceste API-uri comune, între cele trei sisteme de operare existã diferente majore, mai ales în ceea ce priveste implementarea internã a acestor API-uri. Windows 95 se bazeazã pe un nucleu propriu în timp ce Windows NT Server si Windows NT Workstation au la bazã un alt nucleu de sistem de operare. Diferentele între ultimele douã sisteme de operare sunt mai ales în zona de facilitãti cu care vine sistemul în mod nativ si în zona de parametrii de configurare impliciti ai sistemului. Pe moment,

nucleul Windows 95 este mai putin stabil si flexibil dar Microsoft promite pentru anul 1999 trecerea întregii familii pe nucleul NT.

Încã în acest an, Microsoft sperã sã lanseze versiunea finalã pentru Windows NT 4.0 Server si Workstation si o versiune actualizatã OEM de Windows 95 care sã continã printre altele Internet Explorer 3.0. Spre sfârsitul lui 96 va fi lansat si Internet Add-on pentru Windows 95 si Windows NT (numele de cod al acestui produs este acum Nashville) care va fi în principal o integrare între shell si browser. În fine, în anul 1997 va fi lansat Cairo ca un continuator al lui Windows NT si Memphis ca un continuator a lui Windows 95. De altfel, o parte dintre tehnologiile preconizate pentru Cairo au fost introduse în avans în Windows NT 4.0, de exemplu Distributed COM (vezi ilustratia „Urmãtorii pasi în dezvoltarea sistemelor Windows“ din pagina urmãroare).

În ceea ce priveste Internetul, Microsoft are de data aceasta o viziune foarte clarã: în prezent avem aplicatii standalone si pagini Web statice. În viitor, va fi dezvoltatã o tehnologie care sã ne permitã sã creãm obiecte comune celor douã abordãri fãcându-le pe acestea sã poatã comunica între ele. Primul pas este desigur tehnologia ActiveX care ne permite crearea de obiecte (controale OLE) care pot fi folosite atât în dialogurile aplicatiilor standalone cât si în paginile Web.

Ce mai e nou

Principalele noi trãsãturi în Windows NT 4.0 sunt urmãtoarele: s-au executat modificãri asupra shell-ului, s-au îmbunãtãtit mecanismele de listare, s-a mãrit performanta sistemului grafic, s-au modificat unele API-uri si unele mecanisme necesare driverelor.

Unul dintre principalele scopuri declarate ale proiectantilor noii versiuni a fost acela de a îmbunãtãti performantele sistemului. Pentru aceasta, s-a mers pânã la unele restructurãri chiar în interiorul nucleului Windows NT. În plus, s-a ajuns la o

compatibilitate foarte bunã între API-ul Windows 95 si API-ul Windows NT. Desigur, shell-ul implementat în Windows NT este superior celui din Windows 95 prin faptul cã are suport complet pentru Unicode si toate celelalte trãsãturi specifice Windows NT care lipsesc din Windows 95. Printre acestea, enumerãm extensiile proprietãtilor fisierelor de tip .ink, suport îmbunãtãtit pentru variabilele sistem, profiluri pe grupuri de masini, interfatã pentru comprimarea fisierelor, securitate suplimentarã pentru cosurile de gunoi (Recycle Bin), un convertor de fonturi Type 1, si un manager extins de procese care permite o manipulare suplimentarã în cazul aplicatiilor care nu îsi creazã propriile ferestre.

În ceea ce priveste robustetea si performanta noului shell, acestea vin din extensii precum o utilizare mult mai intensã a firelor de executie, un suport mai bun pentru oprirea aplicatiilor care s-au agãtat. Microsoft anuntã cã au fost eliminate multe scame din shell-ul Windows 95 si zone de memorie uitate alocate au fost eliberate.

În ceea ce priveste schimbãrile legate de mecanismele de listare, aceste sunt legate în primul rând de faptul cã a fost introdusã noua interfatã de tip Windows 95. În noul Windows NT directoarele de imprimante se pot partaja între calculatoare permitându-se astfel administrarea de la distantã a imprimantelor. Se poate merge chiar pânã la instalarea de la distantã a unui driver pe un calculator cu o arhitecturã diferitã fatã de calculatorul de pe care se face instalarea. În plus, Windows NT 4.0 adaugã o interfatã utilizator comunã tuturor driverelor de imprimantã usurând astfel sarcina proiectantilor de drivere si utilizarea driverelor în acelasi timp. În fine, pentru metafisierele extinse din Windows prelucrãrile necesare imprimãrii se pot executa la distantã, pe calculatorul pe care este instalatã imprimanta.

În ceea ce priveste sistemul grafic, schimbãrile se referã în primul rând la noile drivere care au fost introduse în sistem pentru WD

ThinkPad, Matrox, Millennium, Trident, Number 9 Imagine, Cirrus. În plus, programul SETUP al Windows NT 4.0 tine acum cont si de eventualele drivere de la terti prezente în timpul instalãrii. În fine, a fost introdus un nou API: DirectDraw 2.0 si suport pentru schimbarea dinamicã a rezolutiei ecranului (fãrã repornirea sistemului). În acelasi timp a fost extinsã si implementarea de OpenGL existentã în Windows NT pentru conformanta cu standardul OpenGL 1.1. Noua implementare ruleazã complet în spatiul utilizator si existã surse comune pentru implementarea de Windows 95 si Windows NT.

La nivelul API-urilor, modificãrile au dus în primul rând la faptul cã noul API Windows NT este un superset al API-ului Windows 95 (au fost implementate toate functiile specifice Windows 95). În plus, a fost adãugat suport pentru aplicatiile foarte mari în ceea ce priveste firele usoare de executie (lightweight threads sau fibers), si câteva apeluri noi precum: SwitchToThread, SignalObjectAndWait, QuequeUserApc, afinitate la procesor bazatã pe proces si timere dupã care se poate astepta.

În zona de drivere, s-au extins driverele pentru CD-uri, SCSI si Enhanced IDE, a apãrut începutul unui suport Plug and Play (doar extendere de bus Hal). Sportul complet pentru Plug and play a fost anuntat pentru începutul lui 97.

Alte extensii ale Windows NT Workstation includ API-ul de telefonie, API-ul de criptografie, Windows Messaging Subsystem (servicii de postã Microsoft si Internet), suport pentru profiluri hardware la pornirea sistemului, care permit startarea a diverse configuratii de rezolutii video, servicii si dispozitive instalate. Pe platformele RISC, a apãrut un emulator 486 care permite rularea aplicatiilor de 16 biti care au nevoie de modul 386 extins sã ruleze pe masini RISC.

În ceea ce priveste Windows NT Server, acesta oferã în plus serverul de informatii Internet (IIS), PPTP (Point to point tunneling

protocol), un router multiprotocol (TCP/IP, AppleTalk si IPX/SPX), performantã superioarã, integrare între DNS (Domain Name Server) si WINS, serviciul RPL care permite statiilor Windows 95 fãrã disc sã booteze de pe serverul NT, extensii ale servicii lor de acces la distantã (RAS). În plus, au fost introdusi o serie de vrãjitori (wizzards) administrativi care trebuie sã ajute la operatiile mai des executate, precum ar fi: Add User Accouts Wizzard, Group Management Wzzard, Managing File and Folder Access Wizzard, Add Printer Wizzard, Add/Remove Programs Wizzard, Install new Modem Wizzard, Network Client Administrator Wizzard, Licence Wizzard.

Schimbãri arhitecturale

În vechile versiuni de nucleu Windows NT, inclusiv 3.51, subsistemul Win32 rula în mod utilizator, cu alte cuvinte în afara spatiului nucleului propriu-zis. Ratiunea acestui mod de lucru era aceea cã sistemul era mult mai stabil datoritã faptului cã driverele de dispozitive grafice, aflate în interiorul subsistemului Win32, rulau în mod utilizator. Aceste drivere nu erau în general dezvoltate de Microsoft ci de parteneri terti si puteau crea probleme în cazul în care contineau erori. (vezi figura „Detalii ale vechiului Subsistem Win32").

Din pãcate, acestã arhitecturã a dus la costuri mari ale apelurilor cãtre modulele USER si GDI, la o implementare complexã de sistem cache si batch pentru executia acestor apeluri în interiorul sistemului, consum mare de resurse, greutãti suplimentare la întretinere si depanare, scãderi de performantã si, în sfârsit, dificultate crescutã la implementarea trãsãturilor specifice Windows 95.

La constructia versiunii 4.0, proiectantii au luat hotãrârea sã mute modulele USER si GDI în executivul NT sub formã de drivere încãrcate dinamic. Facilitãtile expuse de executiv cãtre aceste module sunt deci facilitãtile expuse de cãtre executiv în mod

normal cãtre un driver de dispozitiv sau cãtre un sistem de fisiere. Se evitã astfel accesul direct al modulelor USER si GDI la structurile de date interne ale executivului, acest acces fãcându-se prin interfete bine definite (vezi figura „Detalii ale noii arhitecturi Win32").

Microsoft sustine cã acestã schimbare nu va afecta stabilitatea sau modularitatea sistemului desi alte pãreri sustin contrariul. Cert este cã îmbunãtãtirile de performantã graficã sunt vizibile în Windows NT 4.0. În noua arhitecturã, aplicatiile nu pot accesa sau corupe datele interne ale subsistemului Win32, erorile în subsistemul Win32 (drivere) pot fi controlate si fixate în acelasi mod ca si erorile din executiv sau sistemul de fisiere, iar în timpul rulãrii aplicatiilor se întâmplã mult mai putine schimbãri de context procesor ceea ce duce la o scalabilitate superioarã a masinilor multiprocesor simetrice.

Arhitectura de retea

În Windows NT 4.0 Microsoft a anuntat numeroase extensii în zonele de internetworking, acces la distantã, TAPI, NDIS 4.0, Windows Sockets, DNS si securitate. Sã vedem mai detaliat câteva dintre aceste extensii.

În serverul Windows NT 4.0 rutarea a fost integratã prin introducerea unui suport multiprotocol. Acest suport lucreazã cu RIP/IP, RIP/IPX si RTMP/Appletalk. Suportul multiprotocol permite extensii prin introducerea de drivere specifice unor noi medii de transfer sau protocoale.

NDIS 4.0 a fost si el extins. Principalele trãsãturi nou adãugate sunt optimizarea cãilor de transmitere si receptie prin transmiterea/primirea de mai multe pachete cu un singur apel API, interfete pentru suportul unor medii aditionale, scalabilitate de multiprocesor pentru driverele miniport, detectia automatã a

cartelelor de retea PCI si EISA ale cãror drivere sunt în biblioteca de drivere a sistemului.

Noua versiune de Windows NT va implementa deasemenea API-ul Windows Sockets 2.0. Aceastã nouã versiune de Windows Sockets va contine o independentã fatã de transport (TC/IP, IPX/SPX, Appletalk...), o integrare superiarã cu Win32 si servicii superioare pentru multimedia. Extensiile de performantã ale Windows Sockets 2.0 sunt cele care permit transmisia unui fisier întreg în Internet (se eliminã astfel tranzitiile multiple între nucleu si aplicatie în cazul unor tranzactii lungi Internet) si modificãri aduse apelului AcceptEx care reduc numãrul de operatii necesare pentru acceptarea unei conexiuni Internet.

În noua sa versiune, Windows NT devine o platformã care oferã servicii multiple, printre care servicii ale retelei Windows, servicii de informatii Internet (IIS ruleazã mult mai performant sub Windows NT 4.0 si întreaga sa administratie a fost mutatã sub formã de pagini Web), servicii pentru retele NetWare (server de fisiere, listare, directory management), servicii pentru Macintosh (server de fisiere, listare), servicii NFS (necesitã achizitionarea unor produse de la terti).

În ceea ce priveste serviciile specifice retelelor Windows, Microsoft anuntã cã performanta transferurilor este mult îmbunãtãtitã, în special în cazul mediilor de 100Mb. Se pot accesa servere de oriunde din Internet pe baza adresei de Internet (datã de un server DNS). De altfel serviciile DNS si WINS sunt concepute pentru a lucra împreunã si a îsi rezolva problemele reciproc. În plus, Microsoft a anuntat extensii pentru integrarea statiilor Windows 95 cu serverele NT în ceea ce priveste performanta operatiilor client-server si a listãrii pe imprimante NT.

Calculatoarele Macintosh conectate în retele Windows NT pot beneficia de serviciile serverelor NT 4.0 fãrã nici un efort special. Serviciile de fisiere si imprimare sunt disponibile direct prin

software-ul standard Appleshare iar conturile Macintosh sunt conturi standard Windows NT. Sistemul de fisiere NTFS, cu spatii de nume multiple, poate memora nume de fisiere Macintosh iar iconurile aplicatiilor create cu un PC sunt afisate corect datoritã extensiei de asociere de tipuri care se poate instala pe Macintosh. În fine, calculatoarele Macintosh pot tipãri fisiere de orice fel, inclusiv Postscript la orice imprimantã conectatã la Windows NT.

Viitoarele domenii de interes pentru Microsoft în zona retelelor vor fi standardele de transmisie în retea pentru multimedia (ATM, RSVP, RTP), cresterea usurintei de configurare, integrare superioarã cu Internetul, cresterea performantei si a scalabilitãtii.

COM Distribuit

Poate cã acest nume vi se pare necunoscut, dar el nu este în fapt decât o redenumire pentru vechiul tel Microsoft prevãzut pentru Cairo si anume Network OLE. În fapt, cei dintre dumneavoastrã care stiu ce este si cum functioneazã COM (Component Object Model) stiu deja si cum va functiona viitorul DCOM: este un COM cu reteaua activatã.

Actualele utilitare de constructie a aplicatiilor distribite sunt extrem de greu de utilizat si destul de limitate. RPC necesitã cunostinte foarte multe despre retele si este suportat de putine utilitare de dezvoltare. Alternativa HTTP/CGI este o comunicatie într-o singurã directie, foarte greu de depanat. În plus, un OLE transpus în retea ar avea avantajul cã permite pãstrarea cunostentelor si aplicatiilor deja existente.

DCOM va permite o comunicatie în douã directii, simetricã si o performantã acceptabilã. În plus, scalabilitatea va fi mult mai bunã. Toate aplicatiile existente, scrise în COM vor putea fi portate fãrã nici o modificare spre COM distribuit. DCOM va fi optimizat în principal pentru protocoale interactive (mult mai eficiente, mai ales peste UDP), definirea usoarã de API-uri de obiecte care pot fi

extinse, integrare cu tehnologia ActiveX, trãsãturi de securitate elaborate.

Microsoft, foarte mândru de tehnologiile OLE în general si de COM si DCOM în special face un efort sustinut de portare a acestei tehnologii pe alte platforme. Astfel, firma a fãcut un contract cu Software AG pentru a porta DCOM pe diverse platforme Unix. Digital Equipment Corporation lucreazã si el la integrarea DCOM cu CORBA/ObjectBroker si introducerea infrastructurii DCE pe acelasi platforme Unix. Primele rezultate ale portãrilor se vor vedea la sfârsitul anului 1996. În plus, Microsoft face eforturi sã integreze COM cu clasele de obiecte create cu ajutorul limbajului Java în noua sa masinã virtualã Java (care va fi disponibilã împreunã cu Visual J++).

Microsoft si Internetul

Microsoft, chiar dacã putin mai târziu decât ar fi trebuit, a realizat în sfârsit cã PC-urile împreunã cu Internetul vor reprezenta principala atractie în viitor. De aceea în interiorul firmei au fost lansate câteva proiecte de primã urgentã care sã ducã la reducerea decalajului. Aceste proiecte intrã în douã mari grupuri: tehnologia de navigare si tehnologia de server de informatii.

Microsoft spune cã strategia sa Internet se poate rezuma astfel: îmbrãtiseazã, extinde, adaugã valoare. A îmbrãtisa înseamnã la Microsoft preluarea standardelor de succes în Internet si integrarea acestora în produsele de bazã. Extinderea înseamnã îmbunãtãtirea protocoalelor, standardelor si produselor pentru satisfacerea superioarã a clientilor. În sfârsit, adãugarea de valoare înseamnã impunerea noilor idei înapoi în Internet pentru a oferi o functionalitate superioarã într-un mod deschis.

Întrebarea de bazã pe care si-a pus-o Microsoft este aceea dacã aplicatiile standalone, specifice PC-urilor trebuie sã fie pentru totdeauna despãrtite de paginile Web statice specifice Internetului.

Amândouã aceste solutii oferã interfete grafice, componente software, aplicatii client-server, multimedia, 3D si conferinte interactive. De ce nu ar putea fi unificate aceste douã abordãri? De ce n-ar putea fi create aplicatii care sã preia ce este mai bun din Web si PC pentru a crea aplicatii interactive distribuite si comunicatii interactive multimedia între indivizi? În aceastã situatie, Microsoft a decis sã creeze o strategie care sã ducã la integrarea produselor din Microsoft BackOffice cu Web-ul.

Împreunã cu Windows NT 4.0 beta 2 vine integrat Internet Explorer 2.0, navigatorul gratuit construit de Microsoft ca rãspuns la rãspândirea extraordinarã a lui Nescape Navigator. Probabil cã în produsul final va fi integrat deja Internet Explorer 3.0 care contine tehnologia ActiveX si suport pentru appleturi Java. Internet Explorer 3.0 este disponibil deja în versiune beta 2. Principala trãsãturã pe care Internet Explorer o opune concurentului sãu de la Netscape este suportul pentru tehnologia ActiveX. Aceastã nouã tehnologie mult lãudatã de Microsoft are urmãtoarele trãsãturi: este independentã de limbaj, ruleazã pe platforme multiple, este rapidã, compatibilã cu controalele OLE, appleturile Java pot fi vãzute automat ca si controale ActiveX, se poate utiliza în pagini Web sau în aplicatii standalone, este suportatã de foarte multe utilitare de dezvoltare (toate care includ suport pentru OCX).

Noua tehnologie suportã desigur script-urile pentru minimizarea traficului între client si server. În principal, ActiveX suportã VBScript si JavaScript dar existã si suport pentru dezvoltarea unor noi limbaje de cãtre terti. Suportul multimedia în ActiveX este foarte dezvoltat, incluzând Direct3D (graficã de mare performantã, construitã direct pe API-ul DirectX), ActiveVRML pentru crearea de lumi virtuale si ActiveX Movie pentru aplicatii sofisticate care necesitã video sau audio.

Cel de-al doilea punct de interes la Microsoft este serverul de informatii pentru Internet (IIS), care este în esentã un server Web.

Serverul vine integrat în Windows NT 4.0 Server si contine urmãtoarele componente: servicii FTP, Gopher, HTTP, documentatii multiple on-line, suport pentru standarde industriale de extensie precum CGI, ISAPI (Information Server API), interfatã de conectare la baze de date (Internet Database Connector), aplicatii de administrare.

Serviciile Web contin trãsãturi excelente, precum ar fi servere virtuale (serverul poate fi vãzut sub mai multe nume), parolele sunt transmise criptat, existã un API (ISAPI) care oferã o cale usoarã de extindere a functionalitãtii serverului, conexiune cu dazele de date care permite încãrcarea informatiilor dintr-o bazã de date în paginile Web si SSL care oferã posibilitãti de comunicare criptatã. Cu serverul IIS se poate limita traficul în retea, se poate controla multimea clientilor care au acces la server, se pot crea fisiere de logare a traficului (vezi figura „Arhitectura serverului de informatii Internet").

Interfata ISAPI permite crearea de extensii ale serverului Web într-un mod standard. Aceste extensii beneficiazã de o perfomantã ridicatã rulând chiar în interiorul procesului si îsi pãstreazã starea de la o cerere la alta. Cu ajutorul aplicatiilor ActiveX se pot crea machete, se pot procesa informatii sau se poate genera dinamic continutul unei pagini. Cu ajutorul filtrelor ActiveX se pot monitoriza cererile, se pot genera statistici, se poate extinde mecanismul de autentificare, se pot executa translatãri ale informatiilor.

Conectorul de baze de date oferã integrare între serverul Web si serverul SQL sau altã sursã de date ODBC. Acest conector este de fapt o extensie ISAPI ceea ce îl face foarte rapid. Configurarea este foarte usoarã si nu necesitã programare. Se pot executa cereri dinamice si statice complexe sau actualizãri si se pot apela proceduri memorate în baza de date. În fine, conectorul vine în mod standard cu ODBC 2.5 si driverul pentru SQL Server. Modul

în care gândeste Microsoft extinderea BackOffice pentru suportul Internet este prezentat în figura alãturatã.

Administrare

Principalele teluri ale noului sistem de administrare al Windows NT 4.0 sunt: o administrare mai usoarã a sistemului pentru cei care nu sunt obisnuiti cu sistemul si o administrare usoarã a retelelor eterogene de Windows NT si Windows 95.

În primul rând Microsoft a construit o multime de vrãjitori (pe care i-am enumerat deja) care sã ajute la executia operatiilor celor mai comune pentru un administrator sau utilizator. În al doilea rând, au apãrut noi utilitare corespunzãtoare noilor facilitãti aflate în sistem. A apãrut astfel un monitor de retea, un administrator de DNS, un editor de politicã a sistemului, utilitare de administrarea a Windows NT de pe un sistem Windows 95, un administrator pentru configurarea DCOM.

În fine, managementul domeniilor si utilizatorilor este mult îmbunãtãtit pentru a putea controla mai bine drepturile si preferintele fiecãrui utilizator conectat la un sistem NT.

Performantã

Microsoft anuntã cã noul sãu server si-a îmbunãtãtit mult performantele generale. Performantele serverului de fisiere par sã fi crescut cu peste 100% datoritã redirectorului modificat, schimbãrilor din SMP si din serverul de fisiere propriu-zis. Performanta se obtine mai ales în retele de 100 MB/s pentru care a fost optimizat Windows NT 4.0. În ceea ce priveste serverul de aplicatii, acesta si-a îmbunãtãtit scalabilitatea, a definit noi API-uri pentru aplicatiile sofisticate precum afinitatea soft (la un anumit procesor), dezactivarea cresterii dinamice de prioritate (dynamic priority boosting), crearea unei operatii atomice de semnalizare si asteptare sau atasarea conditionalã a unei sectiuni critice.

Microsoft anuntã cã serverul sãu SQL rulează sub Windows NT mult mai repede acum si bate un server Oracle rulând sub UnixWare sau Solaris ca sã nu mai vorbim de Sybase sau Oracle pentru NT. Ultimele douã rãmân în urmã, dupã spusele lui Microsoft, datoritã unei proaste implementãri vis-a-vis de facilitãtile sistemului. În ceea ce priveste IIS, Microsoft afirmã cã acesta merge de 3-4 ori mai repede decât competitorul sãu Netscape NetSite.

Nuclee monolitice

De la aparitia ideii de micro-nucleu, aceasta a suscitat un enorm entuziasm printre cercetãtori si industriasi. Tehnica promitea sã rezolve elegant o multime de probleme din proiectarea sistemelor de operare, si sã permitã scrierea de sisteme distribuite cu mare usurintã. În mod paradoxal însã, la aceastã datã, toate sistemele de operare de uz general au mai curînd o arhitecturã monoliticã. Chiar si despre Windows NT, un cal pe care multã lume serioasã pariazã ca învingãtor în cursa sistemelor de operare, se rîde adesea: „a plecat ca un micro-nucleu, dar s-a umflat pînã a ajuns mai mare ca un macro-nucleu".
Am spus mai sus „sisteme de operare de uz general". Toate consideratiile arhitecturale pe care le prezint sînt valabile pentru majoritatea sistemelor de operare existente la zi. Consideratiile despre eficientã, care sînt cruciale în supravietuirea comercialã a unui sistem, sînt însã semnificative numai pentru sistemele de operare pentru calculatoare obisnuite. Prin contrast, sistemele de operare specializate (de exemplu, sistemele de timp real pentru controlul proceselor, sau pentru masini electronice de jocuri) sînt într-adevãr micro-nuclee, si îsi fac foarte bine treaba lor. Cheia este însã aceasta: treaba lor este într-adevãr foarte specializatã; o masinã SEGA de jocuri electronice nu are nici disc, nici retea, nici periferice prea multe, asa cã sistemul de operare este special scris. Atentia noastrã se apleacã mai ales asupra sistemelor tip Unix/Windows (3.1/NT/95)/ VMS, care sînt

concepute sã permitã rularea unei varietãti nelimitate de aplicatii si partajarea resurselor între programe care nu stiu unul despre celãlalt, adesea în medii „deschise" (retele). O sã procedez pe parcursul acestui articol indirect: voi atinge tot felul de probleme care aparent nu au mare legãturã cu subiectul central, dupã care, în final, într-o sectiune sumarã, voi arãta cum consecintele faptelor pe care le-am tot însiruit, si pe care le socotesc nu lipsite de interes în sine, se adunã întru concluzia indicatã mai sus, care promite dominatia sistemelor monolitice.

Apeluri de sistem

În aceastã sectiune voi face o scurtã recapitulare a modului de functionare a nucleului, pentru a întelege de unde izvorãsc toate problemele. Pentru cã am vorbit aiurea despre aceste lucruri mai pe larg, aici voi fi oarecum succint. Cititorul interesat poate gãsi o descriere a functionãrii unui nucleu de sistem de operare în articolul meu publicat în serial, în PC Report, în septembrie/octombrie 1996 [Pentru cei care nu au cumpãrat revista, articolele la care fac referintã sînt disponibile în postscript din pagina mea de web.]. Nucleul unui sistem de operare se poate asemui cu o bibliotecã de functii care sînt puse la dispozitia proceselor utilizatorilor. Practic întreg accesul la perifericele conectate este mediat de nucleu, din motive de reutilizare a codului, eficientã si, mai ales, securitate [Se înregistreazã fireste si tendinte de a permite accesul proceselor direct la periferice, cum ar fi, de exemplu, în tehnologia Unet, în care procesele pot scrie direct pe placa de retea; deocamdatã sistemele comerciale însã nu au mers atît de departe.]. Pentru utilizatorul normal acest lucru se manifestã prin prezenta unei colectii de functii gata fãcute, cu care el poate manipula perifericele (terminal, disc, fisiere, retea, etc.). Un exemplu tipic este functia **write()** din Unix, prin care se pot trimite date spre un periferic. Functiile puse la dispozitie de cãtre nucleu se numesc apeluri de sistem (SYSTEM CALLS). O altã functie importantã a nucleului, vizibilã utilizatorului prin apeluri

de sistem pentru crearea, distrugerea si manipularea proceselor, este cea de management al proceselor. Un proces este un program care se executã. Nucleul permite mai multor programe independente sã fie „încãrcate" în memorie, puse în executie, oprite si terminate. O functie care este mai rar sub controlul utilizatorului este cea de „planificare" (scheduling) a proceselor: oprirea proceselor care s-au executat prea mult si pornirea celor care tînjesc dupã putinã activitate. Nucleul implementeazã de asemenea notiunea de spatiu de adrese, folosind sistemul de memorie virtualã. În arhitecturile „clasice", fiecare proces are impresia cã posedã în întregime memoria calculatorului. Acest truc este realizat folosind translatarea adreselor (address mapping): pentru fiecare proces nucleul mentine o listã a zonelor de memorie care-i sînt vizibile, iar orice referintã la memorie a unui proces este recalculatã si tradusã într-o referintã într-una din zonele care i-au fost alocate. Astfel, adresa 5 („adresã virtualã") va indica o locatie diferitã de memorie în RAM („adresã fizicã") pentru fiecare proces. Vom vedea un pic mai jos cã sistemul de memorie virtualã permite cîteodatã vizibilitatea unei zone de memorie mai multor procese, pentru usurarea comunicãrii între ele. Nucleul însusi este protejat folosind memoria virtualã. Pentru a putea apela serviciile nucleului, el trebuie sã fie cumva vizibil proceselor. Dar zona de memorie în care se aflã nucleul devine accesibilã numai atunci cînd procesele invocã serviciile nucleului, fiind invizibilã sau inaccesibilã în mod normal.

Structura unui apel

Sã vedem cum poate un proces ordinar beneficia de serviciile nucleului, pãstrînd totusi nucleul inaccesibil. (Nucleul posedã o grãmadã de structuri de date, despre toate procesele, asa încît citirea lor ar putea reprezenta o periculoasã scurgere de informatii. Cu atît mai mult scrierea în zona de memorie fizicã în care se aflã nucleul trebuie sã fie prohibitã în mod normal). Atîta vreme cît un proces se executã, el foloseste Unitatea Centralã într-un mod

neprivilegiat (user mode). Procesul „vede" din memoria fizicã numai portiunea care i-a fost alocatã de nucleu, cam ca în figura „Mod utilizator”. Sã presupunem cã procesul vrea sã cheme un apel de sistem (**write()**, ca sã fim concreti). (Un scenariu asemãnãtor este valabil pentru cazul survenirii unei întreruperi sau executãrii unei operatii ilegale.) Pentru acest scop procesul cheamã o functie de bibliotecã oferitã de fabricantii sistemului, care împachetează argumentele în niste registri, iar într-un registru convenit (de pildã AX) codul apelului de sistem (write() sã zicem cã are codul 3), dupã care executã o instructiune specialã a microprocesorului. Aceastã instructiune are un efect dramatic: cauzează trecerea procesorului în mod privilegiat (kernel-mode), dupã care sare la o rutinã specialã. Aceastã rutinã în primul rînd transformã modul în care se face translatarea adreselor, „aducînd" nucleul în spatiul de adrese al procesului curent. (Aceastã „aducere" se poate face automat prin faptul cã zonele de memorie ale nucleului pot fi accesibile numai în modul privilegiat; depinde de caracteristicile unitãtii de management a memoriei si procesorului prin ce detalii anume se obtine vizibilitatea.) Cert este cã, subit, imaginea aratã ca în figura „Mod privilegiat”. Deodatã toate structurile de date si codul nucleului au devenit vizibile. Apoi rutina specialã (care tocmai se executã) se uitã în registrii conveniti pentru a depista apelul fãcut (în exemplul nostru gãseste în AX un 3). Apoi în functie de acesta cheamã una sau alta din procedurile de tratare din codul nucleului (de-multiplexează apelul, de obicei folosind o tabelã care pentru fiecare apel contine o adresã în nucleu: **call syscall[AX]**). Mai departe, procedura de tratare a apelului de sistem, care este specificã pentru apelul nostru (write) cautã în locurile convenite argumentele (de obicei tot în registri), verificã validitatea lor si începe executarea apelului. Un apel de sistem de genul lui **write()** roagã nucleul sã transfere date spre un periferic. În cazul lui write datele sînt indicate prin adresa virtualã a unui buffer si mãrimea lui: write(periferic, buffer, mãrime). În mod normal, nucleul trebuie sã copieze continutul întregului buffer în interiorul nucleului pentru prelucrare. De ce? Pentru cã acest

proces va fi suspendat acum, în asteptarea terminării executării apelului de sistem (în general, interactiunea cu perifericele este foarte lentã si cauzeazã suspendarea proceselor). Ori dacã acest proces este suspendat, un altul va fi pornit. Dar acest lucru va schimba modul în care este translatat spatiul de adrese virtuale, deci adresa virtualã a buffer-ului indicat nu va mai avea aceeasi semnificatie pentru nucleu! Fãcînd tot felul de trucuri uneori nucleul reuseste sã evite copierea datelor [De pildã, nucleul poate retine adresa fizicã a buffer-ului, si poate marca în tabelele interne acea zonã ca fiind „tabu" (pentru a nu fi modificatã de algoritmii de paginare, care refolosesc memoria fizicã) pînã continutul ei a fost prelucrat.]. Pentru anumite operatii însã, copierea datelor în interiorul nucleului nu poate fi evitatã: de exemplu, cînd datele trebuie sã plece în retea ele trebuie împachetate si sparte în bucãti mai mici, sau atunci cînd merg spre disc trebuie re-aliniate si mutate în cache. De asemenea, cînd datele se duc spre un alt proces (de pildã printr-un pipe în Unix) ele trebuie din nou copiate în interiorul nucleului, pentru a lãsa procesul care face write sã continue sã folosească buffer-ul fãrã a modifica datele deja trimise (dupã scrierea într-o „teavã" (pipe) procesul care face scrierea de obicei îsi continuã executia, dar datele sînt pãstrate pînã cînd un proces de la celãlalt capãt al „tevii" le citeste. Pãstrarea se face în nucleu).

Comutarea proceselor

Sã ne uitãm acum si la operatiile care însotesc comutarea executiei de la un proces la altul, pentru cã acest cost este foarte important în micro-nuclee. Comutarea proceselor implicã salvarea stării procesului curent si încãrcarea stării procesului care urmează pentru executie. Pentru cã cea mai mare parte din stare este continutã în tabele aflate în memorie, schimbarea se poate face relativ simplu încãrcînd valoarea unui pointer spre noua cãsutã din tabelã care se va folosi (pentru a comuta de la procesul 3 la procesul 5, nucleul va pune în pointerul spre cãsuta din tabel cu

datele procesului curent, valoarea 5 în locul lui 3). În general însã, trebuie luate în calcul mai multe operatii. Anume trebuie fãcute urmãtoarele operatiuni, nici una foarte complicatã:

Faza 1: salvarea stãrii:

1. Salvarea registrilor curenti;
2. Salvarea stãrii coprocesorului matematic;
3. Salvarea registrilor de depanare, dacã procesul curent era depanat;
4. Salvarea registrilor si stãrii unitãtii de management a memoriei; salvarea tabelei de translatare a adreselor, a procesului curent (tabela de translatare indicã modul în care se interpretează adresele virtuale pentru procesul curent);
5. Modificarea contoarelor si ceasurilor de executie pentru a reflecta timpul consumat de procesul care se opreste;

Faza 2: comutarea:

6. Rularea algoritmului de planificare (scheduling), care parcurge cozile de procese gata de executie în ordinea prioritãtilor, alegînd pe cel mai urgent;
7. Golirea cache-urilor de translatare a adreselor --- Translation Lookaside Buffer [vedeti si articolul meu despre cache-uri din PC Report din martie 1997 pentru o discutie a TLB.] TLB este un cache care retine felul în care se traduc adresele virtuale cele mai des folosite pentru procesul curent; din moment ce semnificatia adresei virtuale 5 pentru noul proces va fi alta, vechea ei asociere trebuie stearsã si din TLB.

Faza 3: încãrcarea noului proces:

8. Încãrcarea tuturor registrilor salvati (de pasii 1--3), cu valorile lor pentru noul proces;

9. Încãrcarea tabelei de translatare a adreselor, a noului proces si a registrilor unitãtii de management a memoriei. În acest fel noul spatiu de adrese virtuale devine vizibil si cel vechi invizibil;
10. Pentru cã în timp ce noul proces „dormea" s-au putut întîmpla evenimente interesante pentru el (de exemplu, în Unix i-a fost trimis un semnal), acum este momentul de a lua actiuni speciale (în cazul semnalelor Unix, se construiesc cadre pe stivã pentru procedurile de tratare a semnalelor, sau procesul este omorît);
11. Schimbarea pointerilor spre a puncta spre noul proces.
Dupã cum vedeti, sînt totusi o sumedenie de operatii de fãcut. Nuclee foarte sofisticate pot avea operatiile de comutare a proceselor chiar mai complicate decît cele descrise aici. Sã observãm cã în comutarea proceselor mai existã cel putin un cost ascuns, implicat de operatia de schimbarea localitãtii de adresare: pentru cã începem rularea unui nou proces, care va folosi un spatiu de adrese complet diferit, cache-ul microprocesorului va genera foarte multe rateuri pentru început, fiind încãrcat cu date din spatiul vechiului proces. De asemenea, TLB a fost golit (în pasul 7 mai sus), deci pentru a-l umple din nou cu traducerea adreselor în noul proces, va trebui sã fie consultatã tabela de traducere a adreselor pentru noul proces, operatie costisitoare, deoarece implicã accese suplimentare la memorie. Un alt posibil cost va fi plãtit pînã noul proces îsi aduce de pe memoria secundarã (disc) paginile de memorie din setul de lucru (working set); datoritã faptului cã paginile de memorie îndelung ne-folosite sînt de obicei scoase afarã pe disc, s-ar putea ca procesul care tocmai porneste sã trebuiascã sã si le ia de acolo. Aducerea unei pagini este o operatie extrem de costisitoare, care implicã, pe lîngã accesul la disc, si oprirea procesului care cere pagina pînã la venirea acesteia, ceea ce înseamnã încã o comutare de procese!

Plasarea serviciilor Existã în mod logic trei locuri unde poate fi implementat un serviciu:
1. În spatiul procesului care îl foloseste, ca o bibliotecã de functii;

2. În interiorul nucleului, accesat printr-un apel de sistem;
3. În gestiunea unui proces separat, numit „server".
(Un al patrulea loc, mai putin uzual, va fi de asemenea discutat.)
Figura „Unde sunt serviciile” prezintã cele trei cazuri. Le vom analiza pe fiecare pe scurt. Pentru un serviciu dat, foarte adesea proiectantul sistemului are la dispozitie toate cele 3 posibilitãti. Modul dominant în care sînt plasate serviciile (adicã locul majoritãtii serviciilor) dã si clasificarea unui sistem în taxonomia sistemelor de operare. Practic, fiecare sistem de operare va avea servicii în toate cele trei pãrti, astfel încît diferenta este mai curînd una de grad decît de naturã. Astfel, sistemele care aleg varianta 2 pentru majoritatea serviciilor se numesc monolitice, pentru cã tind sã aibã un nucleu foarte mare, cu o grãmadã de cod. Sistemele care optează pentru varianta 3 se numesc prin contrast „micro-nuclee", pentru cã nucleul avînd putine servicii devine foarte mic. În fine, sisteme în care majoritatea serviciilor sînt plasate în spatiul proceselor însele, în functii de bibliotecã, sînt relativ putin rãspîndite. Vom vedea însã niste candidati putin mai jos.

În biblioteci

Cu servicii plasate în biblioteci, este obisnuit orice programator care a folosit un limbaj de genul C sau Pascal. O bibliotecã este o colectie de functii gata scrise, la care programele utilizatorilor se pot „lega", si pe care le pot folosi. Legarea (linking) la functiile din biblioteci se poate face, fie atunci cînd programul este creat (la sfîrsitul compilãrii), si atunci se numeste „legare staticã" (static linking), fie abia dupã ce programul a fost pornit în executie, fiind atunci numitã legare dinamicã (dynamic linking). Cert este cã se face doar odatã, asa încît costul legãrii se „amortizează" cînd functiile din bibliotecã sînt folosite intens. „Costul" unui astfel de serviciu este extrem de scãzut; cel mai scãzut posibil probabil, pentru cã implementarea unui apel de functie în termenii microprocesorului este foarte ieftinã. Trebuie însã sã observãm cã natura codului din bibliotecile partajate care se încarcã dinamic

[Este vorba de faptul cã acest cod este „independent de pozitie" (POSITION INDEPENDENT CODE; PIC).] îl face cîteodatã mai ineficient decît codul obisnuit, cu factori cuprinsi între 1% si 30%. În caseta „Exemple: servicii în biblioteci” sunt prezentate câteva exemple faimoase de servicii plasate în biblioteci.

În nucleu

Al doilea loc unde poate fi plasat un serviciu este în nucleu. Sistemele de operare monolitice pun în nucleu mai toate serviciile care au o utilizare frecventã. Sisteme monolitice tipice sînt (si veti recunoaste toate sistemele dominante pe piatã): Windows 3.1, Windows 95, Unix, VMS, JavaOS [Informatiile mele în legãturã cu JavaOS nu sînt foarte ample, dar cred cã poate fi categorisit ca monolitic.]. Windows NT este considerat în continuare un sistem micro-nucleu, desi contine în nucleu servicii care la Unix (un monolit tipic) sînt în afara nucleului, cum ar fi sistemul de ferestre sau serverul de fisiere. Dupã cum vedeti, granitele sînt difuze între categorii ... Sistemele monolitice sînt comercial cele mai rãspîndite. Pentru ilustrare, sã vedem care sînt categoriile de servicii oferite de un sistem Unix tipic:

Operatii cu procese (creare, distrugere, etc.)*;
Depanarea si mãsurarea (profiling) proceselor*;
Planificarea si executia proceselor*;
Accounting (contabilitate) si tarifare dupã consumul resurselor*;
Operatii cu fisiere;
Comunicatie inter-proces:* tevi-conducte (pipes), semnale, memorie partajatã, semafoare, mesaje;
Protocoale de comunicatie în retea (TCP/IP);
Gestiunea memoriei virtuale*;
Alocarea si eliberarea memoriei*;
Timere si alarme;
Mecanisme de protectie si securitate*;

Legarea dinamicã;
Managementul perifericelor.

Am marcat cu asterisc serviciile care în orice implementare a unui sistem de operare, fie ea monolit sau micro-nucleu, trebuie sã fie oferite de nucleu. (Cum se descurcã exokernel-ul fãrã ele, mie personal nu îmi este foarte clar.) Sistemele monolitice sînt destul de greu de scris si relativ inflexibile: o schimbare a serviciilor oferite se poate face în mod traditional doar oprind sistemul si recompilînd o imagine a nucleului [Sistemele moderne Unix pot încãrca dinamic unele portiuni de nucleu.]. Existã însã o cantitate considerabilã de experientã în folosirea si manipularea acestor sisteme. Cea mai dezirabilã trãsãturã a acestor sisteme (comparate cu bibliotecile) este separatia netã între spatiul de adrese al nucleului si cel al proceselor utilizator. Aceasta permite nucleului sã aibã un control foarte strîns [Cel putin teoretic; faptul cã se raporteazã mereu noi bug-uri în securitatea sistemelor Unix nu zguduie de loc încrederea adeptilor în aceastã tezã.] asupra operatiilor care pot fi efectuate de procese, si îi permite sã forteze cu usurintã respectarea politicilor de folosire a resurselor.

În server(e)

În fine, putem lua o resursã partajatã si o putem depune în bratele unui proces, care sã aibã grijã de ea; procesul care ne „serveste" cu aceastã resursã se va numi „server". Nucleul trebuie sã punã la dispozitia proceselor o metodã eficace prin care sã comunice între ele; de îndatã ce au aceastã metodã la dispozitie, procesele care au nevoie de resursa detinutã de server devin „clientii" lui, trimitîndu-i un mesaj cu cererea lor. Serverul le rãspunde clientilor cu datele cerute. Marele avantaj al acestei formule este cã teoretic se poate aplica si în cazul în care clientul si serverul sînt pe masini diferite si comunicã printr-o retea. Într-adevãr, majoritatea covîrsitoare a aplicatiilor din retea folosesc aceastã arhitecturã. De aici apar însã si problemele, dupã cum vom vedea într-o sectiune ulterioarã

consacratã în mod special sistemelor micro-nucleu, care tind sã exploateze tehnologia client-server. Pentru a ilustra diferenta de performantã, pe acelasi calculator pe care am fãcut mãsurãtorile anterioare, un nucleu experimental extrem de simplu (PicOS --- implementat de autor), fãrã memorie virtualã, cu procese integral rezidente în RAM, permite schimbarea a circa 5000 de mesaje/sec între douã procese pe aceeasi masinã. Asta înseamnã deja 200 de microsecunde pentru un mesaj, adicã 400 pentru un apel complet cerere/rãspuns. Comparati cu performanta unui apel de sistem si cu a unui apel de procedurã. Caseta „Exemple: Servere si resurse” ilustreazã cîteva exemple reale de folosire a serverelor pentru gestiunea resurselor.

Resurse fãrã servere

Sã notãm în treacãt cã toate cele trei solutii citate dau fiecare resursã pe seama cuiva: o bibliotecã, un nucleu, un proces. Existã si o solutie „democraticã", în care nimeni nu posedã un obiect; acest stil de proiectare se numeste „fãrã servere" (serverless). Din pãcate, algoritmii folositi pentru acest fel de probleme sînt în general putin robusti si extrem de complicati, si trãdeazã adesea tocmai cauza pentru care erau creati: evitarea unei „gîtuituri" (bottleneck) în accesul la resursã, reprezentatã de serverul care o gestioneazã. Sã notãm totusi o aplicatie a acestui gen de algoritmi în sistemele de calcul paralele si distribuite, mai ales a celor care implementeazã ceea ce se numeste memorie distribuitã partajatã (Distributed Shared Memory, DSM). Ideea centralã este de a avea pentru toate calculatoarele dintr-o retea un singur spatiu de adrese urias, în care toate scriu si citesc, dar care nu este memorat fizic în vreun loc fixat, ci ale cãrui „locatii" se „plimbã" dupã necesitãti între masinile care le folosesc. Existã si sisteme de fisiere implementate dupã aceastã schemã, dar progresele comerciale sînt (încã) slãbute. Nici noi nu o sã le consacrãm deci prea mare importantã în acest articol, care iar a început sã ia proportii mai mari decît cele anticipate initial de autor.

Semantica operatiilor

„Unde sã plasãm un serviciu, în care din cele trei posturi?" Aparent rãspunsul la aceastã întrebare depinde doar de consideratii de eficientã si esteticã, precum si de extravaganta design-erului. Dar lucrurile nu stau chiar asa! Vom vedea cã anumite însusiri ale unui serviciu depind esential de plasarea sa, si cã fiecare din cele trei scheme are proprietãti speciale, care nu pot fi simulate nicicum în întregime de celelalte. (Vom ignora asemenea consideratii elementare cum ar fi cã nu putem plasa servicii de creare a proceselor într-o bibliotecã, pentru cã atunci cine creazã procesul în care se aflã biblioteca?) Lista de diferente care urmeazã nu este în nici un caz exhaustivã, ci doar vrea sã ilustreze prin exemple problema.

Diferente semantice bibliotecã-nucleu

Deosebirea dintre bibliotecã si nucleu este cu sigurantã familiarã oricãrui programator în C care, frustrat, a încercat sã-si depaneze programele punînd printf()-uri pe ici-colo, dar care nu dãdeau nici un efect! Explicatia este simplã: printf() scrie, dupã cum am vãzut, într-un buffer, care este golit numai în anumite circumstante. Dacã o eroare survine înainte ca apelul de sistem **write()** sã fie executat si procesul moare, continutul din buffer este definitiv pierdut! (Solutia este, fireste, sã fortãm golirea buffer-ului folosind functia fflush().) Asa ceva nu se va întîmpla dacã folosim direct **write()**, pentru cã odatã apelul de sistem executat, datele au fost copiate de nucleu si vor ajunge pînã la urmã la perifericul-destinatie. Diferenta esentialã între cele douã cazuri este de duratã de viatã a informatiei: dacã informatia este într-o bibliotecã, localã unui proces, atunci ea nu poate supravietui mortii procesului [Fireste, o bibliotecã partajatã poate servi drept depozit de informatie]. Nucleul în schimb supravietuieste tuturor proceselor (teoretic), deci poate mentine în sigurantã informatiile globale pentru întregul sistem. Un alt exemplu faimos este implementarea protocoalelor de retea în biblioteci, care pune infernale dificultãti (de altfel, acesta este unul dintre motivele atacurilor la exokernel, care, vã amintiti,

este o bibliotecã mare): de exemplu, standardul TCP/IP impune ca dupã închiderea unei conexiuni de retea, unul din capete sã retinã identificatorul conexiunii pentru o vreme nefolosit („30 de secunde" scrie la carte), în asa fel încît sã nu fie însusit de o altã conexiune (în acel caz, pachete întîrziate ale conexiunii precedente care mai rãtãcesc pe retea ar putea fi incorect primite pe conexiunea nouã). Puteti vedea în Unix lista tuturor conexiunilor cu comanda netstat. Cele care sînt în starea TIME_WAIT sînt conexiuni de acest gen: terminate, dar memorate. Pe un server de web trebuie sã fie o multime de astfel de conexiuni la un moment dat [Dacã nu aveti la îndemînã un server de web, creati o conexiune cu telnet localhost, vedeti ambele capete cu netstat, închideti conexiunea cu exit si apoi vedeti informatia rãmasã din nou cu netstat.]. Ei bine, pe un proces care are protocoalele de comunicatie implementate în biblioteci îl vor trece toate sudorile sã mentinã identificatorul conexiunii dupã moartea procesului.

Diferente semantice nucleu--server

O deosebire de acelasi gen de semanticã (semnificatie), de acelasi gen al serviciilor subzistã între serviciile oferite de nucleu si cele oferite de servere aflate la distantã: în mod normal, pe o masinã, ori merge nucleul si procesele, ori, dacã nucleul nu merge, nu merge nimic. Cu alte cuvinte, dacã procesele pot cere servicii de la nucleu sînt sigure cã gestionarul lor este sãnãtos. Acest lucru nu mai este adevãrat în cazul proceselor care cer servicii de la distantã: cînd arunci niste informatie în retea si nu primesti rãspuns este greu de zis dacã informatia a ajuns si rãspunsul nu, sau informatia s-a pierdut, sau a ajuns si serverul a murit înainte sau dupã ce a primit-o. De aici si complexitatea enormã a protocoalelor de retea si a algoritmilor distribuiti. O altã diferentã, greu de mascat între solutia unei probleme cu nucleu si solutia cu servere este în încredere. Un nucleu stie cã toate procesele care ruleazã pe masina lui au fost create de el si sînt „legitime". Pe de altã parte, cînd un server primeste o cerere de la distantã (sau chiar pe aceeasi masinã), el nu are la dispozitie mijloacele nucleului de verificare. Un mecanism complet diferit trebuie inventat pentru a asigura

serverul de identitatea clientilor, un lucru nenecesar pentru un serviciu plasat în nucleu.

Promisiunile micro-nucleelor

Iatã care sînt unele din avantajele (reale sau fictive) ale arhitecturii micro-nucleu; unele din aceste merite sînt atribuite arhitecturii client-server, altele arhitecturii de micro-nucleu, dar am vãzut cã a doua o implicã pe prima. Modularitate. Într-un nucleu monolitic diferitele pãrti comunicã foarte adesea folosind variabile globale, ceea ce ridicã probleme dificile privitoare la corectitudinea codului. Mai ales pe multiprocesoare, unde mai multe programe pot actiona simultan asupra aceleiasi structuri de date, se pot ivi tot felul de comportamente ciudate. Prin contrast, în solutia cu servere, un server gestioneazã o cantitate redusã de resurse, iar interactiunea cu alte programe se face prin interfete foarte bine precizate (mesaje). E clar, micro-nucleele încurajeazã modularitatea. Scalabilitate. Vrei un serviciu mai puternic: mai adaugi niste servere sau niste clienti, înlocuiesti serverele cu altele mai performante. O masinã este prea încãrcatã: muti din servere pe altã masinã. Toate acestea sînt aspecte ale cresterii incrementale a unui sistem, sau crestere în raport cu resursele si necesitãtile disponibile. Distribuire facilã. Dacã primitiva ta de bazã este trimite_mesaj(), atunci esti încurajat sã dezvolti solutii pentru programe în care nu conteazã unde se aflã server-ul. Mediile de calcul distribuit (DCE: Distributed Computing Environment al lui OSF: Open Software Foundation) sînt un exemplu de standard de scriere a unei aplicatii independent de numãrul de calculatoare pe care se executã. Adaptabilitate (customizability). Nucleul nu poate fi schimbat decît în micã mãsurã, cu mare grijã, si arar fãrã a opri sistemul. Pe de altã parte, serverele sînt simple procese, care pot fi create si omorîte dinamic, fãrã a avea nevoie de privilegii administrative extraordinare. Cu alte cuvinte fiecare utilizator al unui micro-nucleu ar putea sã-si construiascã mediul care îi convine; cine nu foloseste fisiere nu porneste nici un server de

fisiere, si are mai multe resurse pentru alte ocupatii. Dimensiune redusã. Vîrful de lance al creatorilor de micro-nuclee este: plãtesti numai pentru ce folosesti. Nu ai nevoie de ceva: nu primesti. Un nucleu monolit contine toate serviciile, fie cã le vrei, fie cã nu. Comunicatie puternicã inter-proces. Aceasta este o conditie necesarã pentru viabilitatea unui micro-nucleu. Comunicatia trebuie sã fie eficientã, pentru cã fiecare serviciu implicã cel putin un schimb de douã mesaje (cerere/rãspuns), si flexibilã pentru a permite transmiterea unei variate game de mesaje (de la un numãr, la un fisier de megaocteti cu imagini).

RPC

Orice discutie serioasã despre arhitectura client-server trebuie sã atingã mãcar în trecere subiectul apelului procedurilor la distantã (remote procedure calls). Subiectul este fascinant si meritã o tratare mult mai amplã. Observati cã în cazul folosirii serverelor nu numai cã am mutat un serviciu într-un proces separat, dar am schimbat si natura modului în care serviciul este invocat: înainte era printr-un apel de functie (sau de sistem), dar acum este prin trimiterea unui mesaj. E aceeasi diferentã de perspectivã ca între o procedurã si un fisier; pentru un programator paradigma mesajului este mai incomodã. Din cauza asta a fost inventatã împachetarea mesajelor în proceduri. Practic programatorul cheamã o procedurã (dintr-o bibliotecã), care procedurã construieste mesajul, îl trimite, asteaptã rãspunsul si se întoarce în mod uzual. În acest fel programatorul opereazã din nou cu conceptul familiar de procedurã. O astfel de procedurã chematã la distantã se numeste în englezã „remote procedure call", prescurtat RPC. Un pachet RPC face multe lucruri: dintr-o specificare de nivel înalt a procedurii oferite de server[Într-un limbaj de descriere a interfetei, „interface definition language", prescurtat IDL.] el construieste automat functiile pentru client si server. Functiile acestea care manipuleazã mesajul se numesc în englezã STUB. Misiunea lor este de a împacheta argumentele procedurii într-un mesaj (operatie numitã marshalling

), de a trimite mesajul si a despacheta valorile la receptia unui mesaj. Functionarea este prezentatã schematic în figura RPC. Procedurile stub sînt generate de compilatoare speciale din simpla descriere a interfetei lor (tipul argumentelor si al rezultatelor). O altã problemã rezolvatã de un pachet RPC este de a localiza serverele care oferã servicii. Cînd un client porneste, el stie doar cã undeva ar trebui sã se afle un server care exportã serviciile care îl intereseazã. Operatia de descoperire a server-ului se numeste tot „legare", dar în englezã foloseste termenul BINDING, care este un sinonim partial pentru „linking" (corespunzînd acestei faze din compilarea traditionalã). O calitate a unui pachet RPC bun este cã permite invocarea de proceduri pe masini diferite arhitectural de cea pe care se aflã clientul. Pentru asta procedurile stub de împachetare trebuie sã foloseascã un standard comun de reprezentare a datelor, astfel încît calculatoare care folosesc reprezentãri diferite (ex. big/little endian) diferite sã se înteleagã totusi între ele. Existã o multime de probleme cu RPC, care scad oarecum din meritele metodei; în principal, o serie de diferente între semantica unui apel de procedurã obisnuitã si al unei proceduri distante sînt aproape de nedepãsit. De exemplu, cum trimiti un pointer la distantã si la ce-i foloseste server-ului, care are alt spatiu de adrese?

Problema micro-nucleelor

Din cît am divagat, probabil cã problema sare-n ochi: într-un micro-nucleu costul unui serviciu este prea ridicat. Asta pentru cã transmiterea unui mesaj implicã:
1. Apeluri de sistem pentru trimitere si receptie de mesaje, atît de partea clientului cît si a serverului;
2. Cel putin o comutare de procese;
3. Pentru RPC împachetarea si despachetarea argumentelor;
4. Copierea argumentelor din spatiul de adrese al clientului, în nucleu, eventual prin retea, si apoi în spatiul serverului;
5. Crearea unui thread în server pentru a trata cererea;

6. Dacã mesajul trece prin retea este posibil sã fie copiat de mai multe ori: de pildã din nucleu pe placa de retea si invers la receptie; 7. Copierea rãspunsului pe traseul invers server ---> client.

O altã observatie interesantã este cã, în general, în micro-nuclee tendinta este de a fragmenta un serviciu oferit de un nucleu monolit în MAI MULTE servicii oferite de servere diferite. De exemplu, apelul open(fisier) din Unix de obicei devine un apel pentru a traduce numele de fisier într-un identificator unic, executat de serverul de directoare (sau, mai rãu, traducerea fiecãrei pãrti din „cãrare" (path) independent, printr-un apel separat al unui alt server!), si apoi „deschiderea" fisierului la serverul de fisiere propriu-zis (figura 6 ilustreazã acest fapt). În acest caz, ceea ce initial era un singur apel de sistem poate deveni o suitã de zeci de mesaje schimbate cu mai multe servere!

În loc de FINAL: arhitectura NT

Vom încheia acest articol urmãrind cîteva din trucurile fãcute de proiectantii sistemului Windows NT în lupta cu microsecundele. Trebuie spus din capul locului cã pentru a pãstra compatibilitatea cu atîtea sisteme existente (Windows 95, OS/2, etc.) ei nu aveau de ales între prea multe variante arhitecturale, si cã solutia cu serverele din figura 4 este cea mai flexibilã.

Local procedure call (LPC)

Un pachet RPC face o grãmadã de operatii de care nu este nevoie în cazul în care clientul si serverul sînt pe aceeasi masinã. De exemplu, conversia datelor într-un format standard si înapoi este curatã sinucidere, din moment ce procesorul este acelasi. Pe de altã parte o cantitate impresionantã de mesaje în NT tind sã fie între programe locale; de exemplu, tot ce tine de afisare pe ecran se plimbã între unul din serverele de emulare si serverul Windows 95, care singur are în mînã gestiunea ecranului. Din cauza asta între NT 3.54 si NT 4.0 serverul de ecran a coborît în nucleu, cum apare si în figura din caseta „Arhitectura lui Windows NT”. Mai apar si

alte mesaje locale, de exemplu între un program Unix si serverul de emulare Unix. Proiectantii NT au optimizat în mod special acest gen de comunicare, numind-o „apel de procedurã localã" (Local Procedure Call, LPC). Cînd apelul unei functii dintr-un server este pe aceeasi masinã, majoritatea operatiilor costisitoare sînt pur si simplu evitate. Din pãcate, nici atîta nu este suficient pentru a atinge performanta necesarã. Iatã alte douã trucuri care sînt folosite cu succes în cazul NT, pentru servere locale.

Memoria partajatã

Dacã se eliminã operatiile care transformã datele (marshalling), atunci cel mai mare cost nu este dat nici de apelul de sistem, nici de comutarea proceselor, ci de copierea argumentelor mari. Foarte adesea serverele de fisiere sînt implementate folosind RPC; operatii tipice vor transfera cantitãti mari de date din/spre fisier. Copierea datelor din client în nucleu si apoi în server pentru un **write()** (sau invers la citirea unui fisier) este o risipã majorã (overhead) fãrã nici un beneficiu real. Proiectantii lui Windows NT au folosit un mod special de transmitere a datelor între un client si un server, care foloseste mecanismul de memorie virtualã oferit de nucleu. Astfel, clientul alocã o zonã de memorie partajatã (pagini din RAM care sînt vizibile în mai multe spatii virtuale) si trimite server-ului doar o descriere a zonei partajate. De pildã, pentru un **write()** clientul alocã zona (printr-un apel special de sistem), scrie datele în zonã, face un LPC **write()** la server, dînd descrierea zonei partajate. Serverul poate accesa acum direct memoria comunã cu clientul. Nici un octet nu a fost mutat în bufferele din nucleu si înapoi! Singura operatie este modificarea tabelelor de translatare a adreselor pentru a face zona vizibilã si în server. Zona partajatã este fãcutã vizibilã în spatiul server-ului în timp ce serverul executã apelul, dupã care devine din nou invizibilã pentru server. Aceeasi tehnologie, a memoriei partajate, este folositã si în anumite implementãri ale serverelor de ferestre X Windows, în

care clientii locali pot da direct zone de memorie, si nu continutul lor.

Prealocarea resurselor

Amintiti-vã, din sectiunea dedicatã serviciilor oferite de servere, cum functioneazã un server în NT: face „pui" (thread-uri) pentru fiecare nouã cerere, care mor dupã executia cererii. Aceastã creare de thread-uri pentru fiecare serviciu este de asemenea o sursã importantã de ineficientã. De aceea, Windows NT mai încalcã încã odatã regulile bunelor maniere si face o optimizare specialã. Proiectantii spun cã aceastã optimizare este atît de urîtã încît nici nu este accesibilã utilizatorului obisnuit; ea este folositã numai între serverele de emulare si modulul grafic. Pe scurt, între un client foarte activ si un server se stabileste o cale extrem de rapidã de comunicare la cererea clientului. Astfel, server-ul alocã un thread special pentru acel client, care va executa numai cererile acestui client. Mai existã o zonã partajatã alocatã permanent pentru uzul celor doi, si o pereche de „evenimente" (event pair). Perechea de evenimente este de fapt un întrerupãtor prin care clientul si thread-ul alocat din server îsi semnalează reciproc (mai exact îi semnalează planificatorului din nucleu) cînd au nevoie de serviciile celuilalt. Acest mecanism simplificã multe operatii: thread-ul nu mai este creat/distrus, tabela de pagini pentru memoria partajatã nu mai este modificatã (pentru cã zona partajatã va fi folositã pentru totdeauna de douã thread-uri fixate), planificatorul (scheduler-ul) nu mai are de fãcut nici o decizie de alegere, pentru cã va rula server-ul în cuanta de timp neconsumatã a client-ului.

Concluzii

Microsoft pare sã fi fãcut un efort substantial la constructia noii versiuni de Windows NT, pentru a pregãti o loviturã decisivã concurentilor sãi (în principal Unix-uri). Si, dupã cum se pare acum, bãtãlia se va da în principal pe douã câmpuri: bazele de date

si Internetul. Dacã tot ceea ce si-a propus Microsoft va fi rezolvat cu adevãrat, Windows NT va deveni un concurent serios al serverelor Unix care stãpânesc încã în marile companii.