CALCULATOARE PERSONALE

4 / 92
(15)
150 Lei
ISSN 1220-1529

Revista editată de Micro ATCI S.R.L. Tîrgu Mureş

Ultima oră!

S-a înfiinţat POLICOM S.A., societate comercială multinaţională cu capital privat, avînd principal acţionar Ab Mod S.N.C.

Noua firmă vă oferă un sortiment larg de produse:

- Calculatoare personale compatibile IBM
- Imprimante matriciale
- Dispozitive grafice
- Floppy - dischete
- Produse software originale
- Copiatoare Toshiba
- Truse şi accesorii pentru întreţinere

Nu uitaţi!

POLICOM S.A. un partener potenţial pentru afacerile dumneavoastră.

CLUJ: B-dul 22 Decembrie nr. 135 tel. 95-156350

Noutăţi

Vă prezentăm

Multitasking

Reţele

Baze de date

Grafică

Laborator

Cursuri

Fundamente

Hardware

Practică

Rubrici


... nu-i nimic, zic eu. Încarc sistemul de pe drive-ul A, la rece", asta pentru siguranță, după care încerc din nou o citire pe hard. Iarăși eșuează. E clar, nu e virus. Atunci ce e, mă întreb. Ei bine, iubito, știi până la urmă cine era de vină?
CMOSUL!, dragă, CMOSUL!"

if

revistă de informatică
editată de firma **Micro ATCI**

Director: ing. Dumitru Dunca

Redacţia:
Redactor:
ing. Cristian Nagy
Secretariat redacţie:
Doina Ceşa
Colaboratori:
Antonescu Valentin - caricatura
mat. Cozac Nelu
ing. Iacob Ioan
ing. Kallo Tibor
ing. Guţică Mirela
ing. Maier Romulus
ing. Malide Cristian
ing. Pantea Mircea

Tiparul: Tiporex S.R.L.
Tiraj: 8.000 ex.
Preţ: 150 lei

Manuscrise originale sau listinguri de programe sînt primite cu plăcere de redacţie, cu condiţia să nu fi fost publicate şi în altă parte. Prin expedierea unui manuscris pe adresa redacţiei, autorul consimte implicit la publicarea materialului său în cadrul revistei. Onorariul se negociază cu redacţia. Materialele nepublicate nu se înapoiază şi nu se reţin.

Revista noastră vă oferă spaţiu pentru reclamă şi publicitate. Pentru amănunte vă rugăm să luaţi legătura cu redacţia.

Cei care doresc să anexeze revistei pliante publicitare tipărite în regie proprie, sînt rugaţi şi ei să se adreseze redacţiei.

Adresa şi telefonul redacţiei:
Micro ATCI, redacţia "if",
RO-4300 Tîrgu Mureş,
C.P. 64, O.P. 1,
tel./fax 954/31660

Despărţire

Îi anunţăm pe cititorii noştri fideli că în redacţia revistei "if" s-a produs o importantă schimbare. Colectivul redacţional care s-a ocupat de conceperea şi editarea acestei reviste, adică doamna Ingrid Maier, domnul Romulus Maier şi domnul Iosif Fettich, au fondat o nouă firmă - HotSoft şi nu mai lucrează în cadrul firmei Micro ATCI, nici la redacţia "if".

Cei care doresc să menţină legătura cu ei o pot face la adresa: HOTSOFT C. P. 172, O.P. 1, Tîrgu Mureş sau la unul din telefoanele: 954 - 41882, 41417.

Noul redactor le mulţumeşte pentru ţinuta şi prestigiul pe care le-a dobîndit revista "if" datorită eforturilor depuse de ei încă de la primul număr.

Mesaj

"Grea misie, misia de ... redactor" - mai ales atunci cînd este vorba de un nou redactor (nou numai pentru cititorii vechi ai revistei). Grea, mai ales privit prin prisma prestigiului dobîndit de "if" de-a lungul timpului. Ar trebui ca acest prestigiu să nu aibă de suferit, iar cel care se va ocupa de acum de revistă este decis să îl menţină.

Chiar dacă acest număr nu pare a fi chiar cea mai fericită ilustrare a bunelor mele intenţii, vă rog să nu vă descurajaţi. Eventual puteţi încerca să-l priviţi ca pe un număr de "tranziţie". Şi cum orice tranziţie originală se face cu unele mici sacrificii, mai amînaţi puţin aluziile la demisie.

În altă ordine de idei, o revistă care să fie citită de mulţi informaticieni trebuie să fie scrisă de mulţi informaticieni, adică şi de către cei care o citesc. Sînt convins că mulţi dintre cei care urmăresc revista noastră ar avea ceva important de adus la cunoştinţa colegilor de breaslă. Nu ezitaţi să o faceţi ! Dacă bănuiţi că eforturile dumneavoastră de a pune la punct o aplicaţie sau de a face să funcţioneze un program rebel v-au condus la rezultate ce i-ar interesa şi pe alţii, scrieţi-ne.

O notă specială pentru cei nemulţumiţi: dacă există ceva care vă deranjează în revista noastră, nu ezitaţi să ne scrieţi. Critica este foarte utilă, chiar dacă nu constituie totdeauna cea mai plăcută surpriză.

În general, am fi foarte interesaţi să ştim ce vă place şi (eventual) ce nu vă place în "if", ce aţi mai dori să găsiţi în paginile noastre.

Cristian Nagy

Noutăţi AST

Firma americană AST Research anunţă două noi apariţii:

- Un notebook AST Premium Exec 386SX/25C realizat în tehnologie FSTN (Film Supertwist Nematic Technology).
  Caracteristici: procesor AMD 386SXL-25, 4MB RAM, floppy 3,5", hard disk de 60 sau 80 MB, display VGA color (640x480 puncte cu 16 culori), preţ orientativ 6000$. Noul notebook a primit premiul "Inovation of the Year" în martie 1992, la Hanovra.
- Un sistem de transmisie - pe linie telefonică normală - a imaginilor video fixe numit "Photophone Image Communication System". Sistemul este deja utilizat: completarea şi actualizarea bazelor de date complexe (Royal Canadian Mounted Police), asistenţă tehnică complexă la distanţă (Pratt & Whitney, Hughes Aircraft).

CLUB

O ştire de ultimă oră:

Firmele LOGIC din Sibiu şi ROMSOFT din Bucureşti au format un grup de iniţiativă avînd ca scop fondarea primului Club al utilizatorilor de produse Borland.

Cei interesaţi sînt rugaţi să adreseze întrebările sau sugestiile privind organizarea şi funcţionarea clubului, problematica şi statutul, cota de participare pe adresa domnului Gavrilescu Gavril, Centrul Naţional de Difuzare a produselor Informatice - ROMSOFT S.A., bul. Mareşal Averescu nr. 8-10, sect. 1, Bucureşti, tel/fax: 90 - 666229.

(LOGIC News)

DELL COMPUTER

A lansat, în luna mai 1992, calculatorul 450 DE/2 DG. Noul sistem este echipat cu procesor 80486 la 50 MHz, procesor grafic, 96Mb RAM, 1.4 Mb HDD, 2 Gb tape backup. Acest sistem este realizat în tehnologia numită "Processor - Direct Graphics" dezvoltată de DELL împreună cu INTEL. Avantajul cuplării directe a sistemului grafic constă în evitarea "gîtuirii" transferurilor pe magistrala EISA, oferind semnificative performanţe grafice la un cost incredibil de redus. De asemenea, realizarea cu ASIC (Application Specific Integrated Circuit) a controloarelor de memorie şi I/0 îmbunătăţeşte atît performanţa cît şi fiabilitatea calculatorului.

EPSON- Tehnologie care impune

Cuplarea imprimantelor la Host prin cablu coaxial

EPSON pune la dispoziţia noilor imprimante matriciale, cu jet de cerneală şi laser, interfeţele din seria B, care pot fi montate într-un singur conector al imprimantei. Interfaţa pentru cablu coaxial permite emularea imprimantelor IBM 3287, 3262, 3268, 4214 şi 4224 şi dispune de un test al funcţiilor logice ale imprimantei, de intrare de la PC pentru Intelligent Printer Sharing şi de funcţii pentru tipărirea automată a şirurilor de caractere.

Pentru o adaptare optimă la sistem, interfaţa mai dispune de seturi de caractere specifice pentru 24 de ţări, posibilitate de configurare de la Host sau de la PC, Transmission - şi Printer - Dump, precum şi de patru funcţii Bypass (utilizarea directă a proprietăţilor imprimantei).

Cuplare imprimantelor la Host prin cablu Twinax

Interfaţa Twinax #C82D42 permite cuplarea noilor imprimante EPSON la Host-uri IBM/3X şi AS/400. Sînt emulate imprimantele IBM 5256, 5224, 5225 şi 4210/-14/-24. Interfaţa oferă aceleaşi facilităţi ca şi cea pentru cablu coaxial, prezentată mai sus.

(EPSON - Germania)

Port paralel mai rapid

După părerea lui Bill Cott, directorul programului de cercetare a pieţei de imprimante pentru InfoCorp/Computer Intelligence, viteza sporită a portului paralel de imprimantă este rezultatul unui utilitar soft care "sare" peste BIOS-ul PC-ului. Utilitarul va fi oferit de Microsoft ca un produs de sine stătător, dar Gott prezice că fabricanţii de PC-uri îl vor include în viitoarele lor versiuni de BIOS.

Driver-ele software bidirecţionale vor elimina o sursă de necazuri ale utilizatorilor - trimiterea de caractere la imprimantă şi descoperirea, mult mai tîrziu, a faptului că imprimanta nu are hîrtie sau toner. Un astfel de driver va permite imprimantei să afişeze informaţia pe ecranul utilizatorului în loc să trimită numai un mesaj criptic de genul "Printer error".

Mesajele de eroare a imprimantei mai lungi şi mai detaliate îi vor ajuta pe utilizatori să diagnosticheze problemele de tipărire "deoarece trăim în reţea, (şi) imprimantă nu este amplasată fizic (aproape de) persoana care efectuează listarea".

(PC Week)


We draw on your imagination.
A Bold New Shape.
A Fast New Pace.
Take a bold first step to reshape plotting at your firm. Find out more today: Call 800-932-1212. In Canada, call 416-635-9010. We're expecting to hear from you.
A & C INTERNATIONAL S.A.
AUTHORIZED DEALER
Constantin Tanase Str., No. 15, Sect. 2, Bucharest, 73299 Romania
Tel: 40-0-53.53.15
Service - "Hot Line": 40-0-12.77.74
Fax: 40-0-12.77.74
Telex: 10674 MAXIM R
CalComp
A Lockheed Company

Microsoft şi HP se unesc pentru îmbunătăţirea listării în LAN

Microsoft Corp. şi Hewlett - Packard Co. au format o echipă comună într-un efort de a da formă viitorului listării în reţea.

Cele două firme dezvoltă un software ce va "împinge" rata de ieşire a unui port paralel de la 150 Kb/sec. la 1 Mb/sec., precum şi driver-e care permit imprimantei să comunice anumite situaţii de eroare, de exemplu că a rămas fără hîrtie.

În plus, Microsoft definitivează un cartuş pentru imprimantele HP Laserjet seriile II şi III care va include un driver de imprimantă GDI (Graphics Device Interface) pentru Windows.

Cartuşul va permite aplicaţiilor Windows să tipărească documentele utilizînd comenzi GDI în loc să le convertească în limbajul de comandă "nativ" al imprimantei. Se estimează că eliminarea acestui pas va mări viteza de listare a lui Windows de mai mult de zece ori.

Ideea GDI a Microsoft - ului este cea mai nouă tentativă de a-şi aşeza tehnologia pe poziţia de rival al PostScript-ului lui Adobe Systems Inc, urmînd eşecului înregistrat de limbajul de descriere a paginii True-Image al lui Microsoft.

Atît porturile paralele bidirectional de mare viteză cît şi driverele soft bidirecţionale sînt aşteptate să apară în viitoarea generaţie de imprimante laser de la HP.

Prima imprimantă din această familie, care va fi lansată spre sfîrşitul verii, va folosi o nouă "maşinărie" de la Canon Corp. pentru a lista 8 pagini pe minut la o rezoluţie de 600 puncte pe inch. Ea va include Nivelul 1 Post Script, cu Nivelul 2 ca opţiune.

Maşina va costa între 1995$ şi 2500$ în funcţie de cantitatea de memorie pe care o include.

(PC Week)

OS/2 Versiune 2.0

Viena, 27 mai 1992 - Distribuitorii de IBM Personal Systems din Europa răsăriteană au instalat şi pot face demonstraţii cu OS/2. Versiunea 2.0 - un sistem de operare puternic şi totuşi, uşor de utilizat.

OS/2 2.0 oferă trei medii de operare: DOS, Windows şi OS/2 într-un singur pachet, oferind o flexibilitate unică.. În acest moment există, pentru 16 biţi, peste 17000 de aplicaţii DOS, 4900 pentru Windows şi 2500 pentru OS/2.

Sub OS/2, utilizatorii au acces la un spectru imens de programe.

Pentru a se asigura că OS/2 2.0 va rula pe platforme compatibile IBM PC, firma IBM a creat un important laborator de testare a echipamentelor la Boca Raton, SUA, pe lîngă cel existent la Basingstoke, Marea Britanie. Pînă acum, testele de compatibilitate OS/2 2.0 au fost trecute de peste 100 de modele, inclusiv echipamente provenind de la Compaq, Dell, CompuAdd, AST şi Tandy.

(Serviciul de presă IBM - Eastern Europe)

BORLAND C++ 3.0

A apărut versiunea 3.0 a compilatorului de C++ produs de BORLAND International.

După afirmaţiile firmei, această versiune are cîteva noutăţi importante:

- Optimizări globale pentru C şi C++ - compilatorul oferă acum optimizări globale. Optimizatorul dispune de facilităţi ca eliminarea codului mort, sub-expresii comune şi globale, compactarea buclelor, transmiterea parametrilor prin regiştri, permiţînd minimizarea mărimii aplicaţiilor sau maximizarea vitezei lor de excuţie.
- Mediu de dezvoltare Windows - se pot edita, compila şi rula aplicaţii de sub Windows. Mediul de dezvoltare integrat Windows (IDE) include un puternic editor multi-window, ObjectBrowser care permite vizualizarea grafică a relaţiilor între obiecte şi permite "navigarea" rapidă prin codul sursă, SpeedBar pentru accesul rapid la funcţiile uzuale şi un help "online" care include referinţe complete pentru Windows API.

Cerinţe minime pentru sistem:

- Familia IBM sau Compaq de PC-uri şi echipamente compatibile 100%.
  PC-DOS (MS-DOS) 3.0 sau versiuni ulterioare compatibile 100%.
  Microsoft Windows 3.0 sau versiuni ulterioare compatibile 100% pentru programare sub Windows.
  Opţional - Microsoft SDK.
  Este necesar hard disc.
- Borland C++: Intel 286 sau superior cu 1 Mb de memorie extinsă. Pentru programarea sub Windows sînt necesari 2 Mb de memorie extinsă.

(Borland International)

Vă prezentăm:

IBM Personal System/1 Pro 386SX

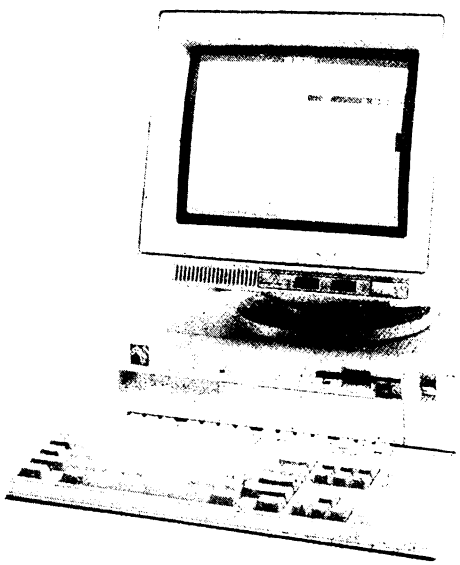
La sfîrşitul lunii aprilie 1992, "big blue" a anunţat calculatorul IBM PS/1 Pro 386SX Model 2123.

Acesta este menit să lărgească spectrul produselor IBM cu sisteme mai puternice şi mai uşor de extins. Modelul se adresează aplicaţiilor de tip:

- lucru adus acasă de la birou
- conducerea unor firme mici
- şcolarizarea pentru PC
- învăţămînt

Calculatorul IBM PS/1 Pro 386SX Model 2123 este livrat în trei variante:

- □ Model E41 - dischetă 1,44 MB, disc fix 40MB
- □ Model E81 - dischetă 1,44 MB, disc fix 80 MB
- □ Model E31 - dischetă 1,44MB, disc fix 129 MB.



Descriere

Configuraţia hardware constă din:

- Sistem cu:
  - procesor 80386SX la 20 MHz.
  - memorie de 2MB (85) pe placa de bază
  - două socluri suplimentare SIMM pe placa de bază
  - circuit ceas
  - interfaţă grafică VGA pe 16 biţi
  - dischetă de 1,44MB
  - disc fix de 40MB, 80MB sau 129MB
  - trei conectori de bus AT
  - interfaţă paralelă
  - interfaţă serială
  - port tastatură
  - port mouse
- Monitor color VGA de 14":
  - rezoluţie maximă
    - APA 640x480
    - AIN 720x400
  - frecvenţă verticală cadre maximă: 70Hz, neîntreţesut
  - frecvenţă de baleiaj vertical maximă: 31,5KHz
  - număr maxim de nuanţe de gri: 64
  - număr maxim de culori: 256
  - număr maxim de culori ale paletei: 262 144
  - moduri standard
    - text: 25 linii x 80 coloane
    - grafic: 480 linii x 640 coloane.
- Tastatură cu 102 taste
- Mouse cu 2 butoane

Pe discul fix sînt instalate, de către producător, următoarele produse software:

- IBM DOS 5.0
- Microsoft Works pentru DOS 2.0
- Microsoft Windows 3.0
- Microsoft Productivity Pack for Windows.

Cerinţe de programare

Sistemul de operare iniţial pentru IBM PS/1 Pro 386SX este DOS 5.0. Echipamentul este compatibil cu:

- PC/DOS 5.00
- PC/DOS+WINDOWS 3.0
- Operating System/2 2.0.

ROM-ul suportă Compatibility BIOS (CBIOS), POST şi BASIC.

Compatibilitate

Calculatorul IBM PS/1 Pro 386SX Model 2123 este compatibil cu sistemele IBM Personal Computer, Personal Computer XT, Personal Computer AT şi PS/2 cu bus AT la nivelul interfeţei BIOS şi cu cele mai multe interfeţe hardware. Datorită nivelului înalt de integrare, nu pot fi utilizate extensiile de memorie şi adaptoarele grafice pentru IBM PC-XT, At şi PS/1.

Limitări

Spaţiul maxim de memorie care poate fi adresat pe 24 de biţi (386 SX) pe IBM PS/1 Pro Model 2123 este de 16MB.

Din acest spaţiu, sistemul are nevoie de 1MB pentru utilizare proprie, permiţînd beneficiarilor să utilizeze un spaţiu de memorie de maximum 15MB.

Produse suportate

Produsele hardware compatibile cu IBM PS/1 Pro 386SX Model 2123 sînt:

- module memorie PS/2
- adaptoare de reţea IBM
- adaptoare Token Ring IBM
- disc fix de 129 MB IBM
- ecrane color 8513, 8514, 8515, 8518 sau monocrom 8503
- imprimante IBM Personal Printer, Laser Printer, Pro Printer
- imprimante Epson LQ 2550, HP Laserjet III sau Paintjet XL
- adaptoare 3Com Ethernet II, X25/Communication

Compatibilitate software

Testele de compatibilitate software au fost efectuate de către EMEA Compatibility Test Group de la Basingstoke, Scoţia în perioada 6.02.92 la 26.03.92.

Următoarele produse sînt compatibile cu IBM PS/1 Pro şi vor funcţiona foarte apropiat de specificaţiile din documentaţii:

- Sisteme de operare
  - DOS 5.00
  - DR DOS 6.0
  - OS/2 ES$ 1.0, LS 2.0 şi SE 2.0
- Produse de comunicaţii
  - 3270 Emulation Program Entry Level 2.00
  - Novell Netware Lite 1.0

PC 3270 2.00

- Windows
  - PC 3270 Windows 2.00
  - Procom Plus 1.1
- Produse pentru business/productivitate
  - Aldus PageMaker (Windows) 4.00 şi (OS/2 PM) 3.01
  - Corel Draw (Windows) 2.01 şi (OS/2 PM) 2.00
  - Lotus 123 (Windows) 1.0 şi 123/G(OS/2 PM) 1.00
  - Lotus Freelance (DOS) 4.0 şi (Windows) 1.0
  - Microsoft Excel (Windows) 3.00
  - Microsoft Windows 3.0
  - Microsoft Word (Windows) 1.10 şi 1.11
  - Microsoft Flight Simulator 4.0
  - Word Perfect (DOS) 5.1

Următoarele produse soft au fost testate pe modele S.U.A. cu sisteme de operare S.U.A.

- Enhanced 52550 Emulation Program 2.2
- IBM Personal Communications /3270 2.0
- IBM CAD şi CAD/Plus (DOS şi OS/2) 3.0
- IBM System Manager 2.0
- Story Board Plus 2.0

- Software de reţea IBM
  - IBM PC Lan Program 1.34
  - IBM OS/2 Lan Server Entry 2.0
  - IBM OS/2 Lan Server Advanced 2.0
- Software de reţea Novell
  - NetWare Lite 1.0
  - NetWare 2.2 şi 3.11
  - Novell Print Server 1.21
  - NetWare Link/X.25 1.0

Următoarele produse soft au fost testate pe IBM PS/1 Pro de un laborator independent de testare utilizînd modele şi sisteme de operare S.U.A.

Firma	Descriere	Vers.	DOS 5.0	OS/2 2.0
Aldus	PageMaker	4.0	x	
Ashto-Tate	dBASE III Plus	1.10	x	
	dBASE IV	1.10	x	
Autodesk, Inc.	AutoCad	10C	x	x
	AutoSketch	2.0	x	
Borland International, Inc.	PARADOX	3.5	x	
	PARADOX for OS/2	3.0		x
	Quattro Pro	3.0	x	
	TURBO C++	2.0	x	
Central Point Software	FastBack Plus	2.1	x	
	PC Tools Deluxe	7.1	x	
Fox Software, Inc.	Fox Pro	1.02	x	
Lotus Development Corporation	Freelance Plus	3.01	x	
	Lotus 1-2-3	3.0	x	x
Micro Focus	COBOL/2 Compiler + Workbench + Toolset	2.4		x
Microsoft Corporation	Basic Compiler	7.1		x
	Excel	3.0	x	
	Macro Assembler	5.1		x
	Windows	3.0	x	
	Word for Windows	1.1	x	
Oracle Corporation	ORACLE for OS/2	5.1C		x
Peter Norton Computing	Norton Utilities Advanced	6.0	x	
QuarterDeck, Inc.	DesqView 386	2.3	x	
Software Publishing	Harvard Graphics	2.301	x	
Word Perfect Corporation	WordPerfect	5.1	x	
Wordstar International	WordStar 2000 Plus	6.0	x	
	WordStar	6.0	x	

(IBM - Eastern Europe)

Multitasking Cooperativ în C++

Programarea sistemelor incluse în instalaţii complexe ne pune în faţa unor probleme interesante. O aplicaţie tipică poate implica "jonglarea" - pe un sistem foarte mic - cu o varietate de funcţii de eşantionare, procesoare, control şi comunicaţie ce se desfăşoară simultan. Astfel de aplicaţii sînt, inevitabil, paralele şi se implementează cel mai bine prin mai multe taskuri mici care lucrează împreună. Din nefericire, pentru un multitasking normal este necesar un suport din partea sistemului de operarea, comutarea contextelor poate fi lentă, iar partajarea resurselor poate fi complicată. Pentru a evita aceste neajunsuri se poate adopta soluţia creerii unor obiecte de multitasking cooperativ în C++.

Multitasking cooperativ şi multitasking forţat

În multitasking-ul forţat, o întrerupere de ceas lansează periodic un program executiv. Acsta determină dacă taskul curent a rulat prea mult timp şi - dacă este aşa - forţează comutarea contextului la următorul task din lista de aşteptare. Dacă taskul mai dispune încă de timp, el actualizează un contor şi revine din întrerupere. Deoarece o întrerupere poată să survină şi să forţeze o comutare de context în orice punct al programului, sistemul trebuie să salveze şi să refacă starea complată a procesorului. Partajarea resurselor de către taskuri impune rutine care să blocheze şi să deblocheze accesul la structurile de date comune. Mai mult, apelurile la sistemul de operare trebuie să fie reentrante (ceea ce nu se întîmplă la MS-DOS).

De cealaltă parte, multitaskingul cooperativ nu are un executiv şi nu se bazează pe întreruperi. De cîte ori un task este gata să cedeze controlul, el apelează o subrutină (*pause*) care comută contextul la următorul task. Deoarece punctul de comutare este întotdeauna acelaşi (rutina *pause*) este necesar să se salveze numai indicatorul vîrfului stivei şi cîţiva regiştri. Aceasta determină o comutare de context simplă şi - potenţial - foarte rapidă. Multitaskingul cooperativ simplifică mult partajarea resurselor deoarece nu se pot "strecura" alte taskuri în timpul actualizării structurilor comune de date. Apelurile de sistem pot fi efectuate sub orice sistem de operare deoarece, în timpul apelului, nu poate surveni nici o comutare neaşteptată de context.

Programul

Nucleul de multitasking cooperativ prezentat mai jos este implementat utilizînd un obiect task din C++. Obiectul task şi fişierele "header" sînt prezentate în listingurile Unu şi Doi. A mai fost inclus şi un program demonstrativ (Listing Trei) care prezintă utilizarea nucleului de multitasking. Programul demonstrativ are nevoie de obiectul fereastră text şi de fişierele "header" prezentate în Listingurile Patru şi Cinci. Interfaţa cu nucleul constă din rutinele *InitTasking*, *fork* şi *pause*. Rutina *InitTasking* iniţializează sistemul multitasking şi ea trebuie apelată înainte de orice alt apel. Rutina *fork* generează noi taskuri, iar *pause* efectuează comutarea contextului la următorul task. Obiectul task ăn sine are o metodă de a modifica ceea ce execută un obiect task - *Activate* şi o metodă *Show* pentru afişarea variabilelor stării curente în scopul depanării. Deşi programul este scris în Borland C++, el ar trebui să fie portabil şi în alte medii, dar cu următoarele neajunsuri:

Programul demonstrativ utilizează cîteva funcţii de lucru cu ecranul specifice bibliotecii Borland standard, iar obiectele task fac acces direct la regiştrii procesorului via "pseudovariabilele" Borland. Referinţele la aceste pseudovariabile se pot înlocui uşor cu secvenţe scrise în limbaj de asamblare. Se poate folosi orice tip de procesor dar, desigur, trebuie modificată partea de salvare a regiştrilor. Deşi aceste obiecte au fost concepute pentru modelul de memorie *small*, este necesară numai o mică modificare pentru modele mai mari.

Comutarea contextului

Convenţia de apel din C, utilizată şi în C++, realizează apelul unei proceduri depunînd pe stivă toţi parametrii transmişi, urmaţi de adresa de întoarcere. Cînd procedura apelată se termină, ea revine la adresa preluată din stivă lăsînd programului apelant sarcina de a îndepărta din stivă parametrii transmişi. Regiştrii BP, SI, DI şi indicatorul vîrfului stivei (SP) păstrează starea completă a unui program Turbo C++ atunci cînd se apelează o funcţie. De aceea, pentru a salva starea unui task, este necesar să se depună numai aceşti trei regiştri pe stivă şi să se memoreze indicatorul vîrfului stivei. Următorul obiect task începe cu preluarea indicatorului salvat al vîrfului stivei, extragerea regiştrilor din stivă, şi revine la adresa de unde a fost lansat noul task. Acum noul task rulează iar apelantul pune stiva în ordine. Un exemplu de diagramă de execuţie pentru două taskuri poate fi observat în Figura 1. Se

remarcă faptul că instrucţiunea *pause* din interiorul buclei *while* a taskului 1 se execută de două ori. Fiecare execuţie abordează taskul 2 într-un alt punct, în funcţie de locul în care taskul 2 a cedat controlul la execuţia anterioară.

Obiecte task

Rutina *fork* din Listingul 2 crează un nou obiect task şi iniţializează obiectul pentru a executa funcţia ce i-a fost transmisă. Fiecare obiect conţine propria sa zonă de stivă, indicatorul vîrfului stivei ce a fost salvat, un număr de identificare pentru depanare şi se leagă cu alte obiecte task. Numărul de obiecte task nu etse limitat decît de memoria disponibilă. Procedeul de construire a taskurilor leagă noul obiect într-o listă circulară de alte obiecte task (Figura 2).Mărimea zonei de stivă proprie este transmisă constructorului. Mărimea implicită a stivei este, în acest program, destul de mare din cauza unor funcţii de bibliotecă de genul lui *printf* care utilizează un spaţiu apreciabil din stivă. Valoarea zero pentru mărimea stivei determină constructorul să asigneze noul obiect stivă la stiva sistemului. Numai un singur obiect task poate să deţină stiva sistemului. Programul crează acest task în timpul executării funcţiei *InitTasking* apelată la începutul programului. În cazul modelului de memorie *small*, funcţia *malloc* (care alocă zona de stivă) utilizează indicatorul curent al vîrfului stivei pentru a determina dacă are spaţiu disponoibil. Pentru ca procesele fiu să poată genera taskuri suplimentare, funcţia *fork* "împrumută" stiva sistemului înainte de noua operaţie. Acum oricare din taskuri pot genera taskuri suplimentare după necesităţi, utilizînd funcţia *fork*. Metoda *Activate* iniţializează stiva taskului astfel încît să refacă valori fictive în SI, DI, BP şi iniţializează adresa de întoarcere astfel încît să execute funcţia transmisă lui *fork*. De asemenea, *Activate* pune pe stivă adresa rutinei *terminate*. Dacă funcţia transmisă se termină şi se întoarce, execuţia trece la *terminate*, care şterge obiectul task şi face o comutare de context la taskul următor. Fiecare obiect task are cîte un pointer înainte şi unul înapoi, astfel încît metoda *destructor* poate să extragă uşor un task din lista înlănţuită.

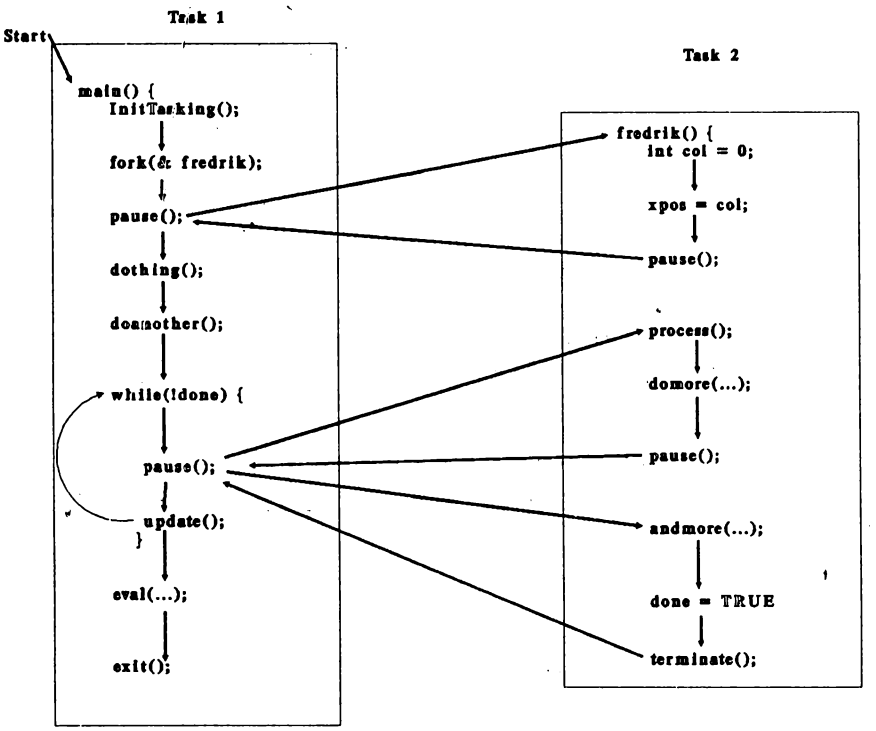


Figura 1: Execuţia pentru un sistem cu două taskuri

Comutare

Atunci cînd taskul aflat în execuţie este gata să cedeze controlul execută funcţia *pause*. Această rutină trimite un mesaj *Switch* către obiectul task aflat în execuţie curentă. *Switch* depune pe stivă cei trei regiştri, salvează indicatorul vîrfului stivei, şi se mută la următorul task din lista înlănţuită. Metoda *switch* utilizează o mică "şmecherie" pentru salvarea regişitrilor. Pentru fiecare metodă a unui obiect, compilatorul generează automat o instrucţiune PUSH BP la începutul şi instrucţiunea complementară POP BP înainte de instrucţiunea de întoarcere.

În mod normal compilatorul utilizează SI şi DI pentru variabile registru şi generează instrucţiuni PUSH numai dacă se utilizează variabile registru. Asignînd, la începutul rutinei, pe SI lui DI, vom forţa compilatorul să salveze regiştrii SI şi DI generînd PUSH SI, PUSH DI, precum şi instrucţiunile POP corespondente la sfîrşitul rutinei. După executarea instrucţiunilor PUSH, toţi regiştrii necesari sînt pe stivă, astfel încît indicatorul vîrfului stivei este memorat în obiect. Funcţia extrage indicatorul vîrfului stivei pentru noul tassk din lista înlănţuită şi, la ieşirea din rutină, extrage din stivă regiştrii noului task şi se întoarce la adresa anterioară de execuţie a noului task. Codul, în limbaj de asamblare, generat de *switch* este prezentat în Figura 3.

Această tehnică de a forţa compilatorul să salveze regiştrii importanţi funcţionează în multe alte compilatoare. Deşi rutina ar putea fi optimizată în limbaj de asamblare, a fost mai uşor să se păstreze codul la nivelul înalt şi să nu recurgă la programare direct în limbaj de

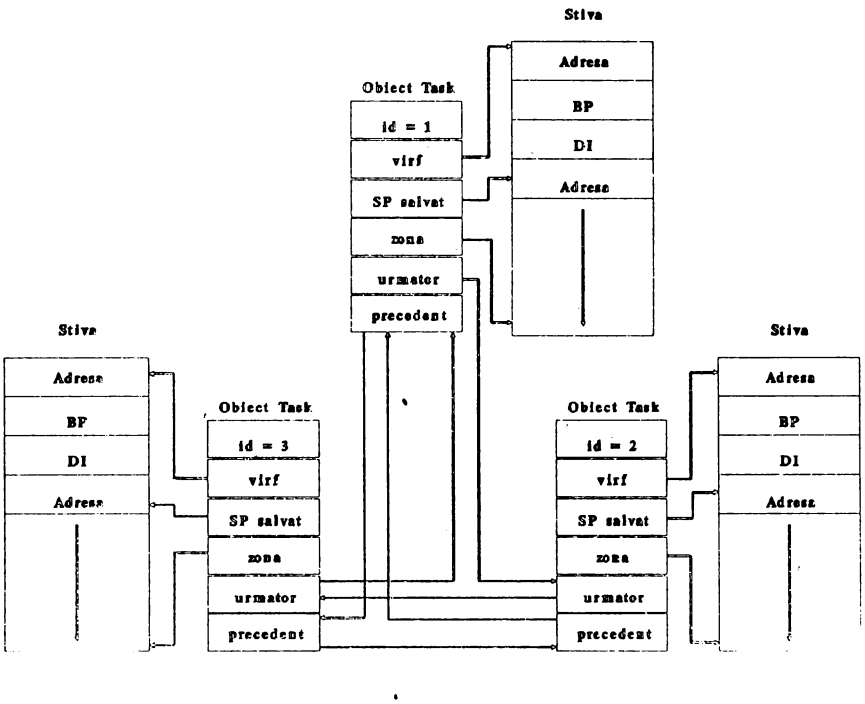


Figura 2: Obiecte task în lista înlănţuită

asamblare.

Programul demonstrativ

Programul demonstrativ crează şase taskuri care rulează fiecare în fereastra sa de text (programul este compatibil cu orice display standard IBM). Programul crează cinci dintre taskuri executînd *fork*; al şaselea este taskul "principal", care utilizează stiva sistemului. Cînd programul demonstrativ rulează se pot crea taskuri temporare suplimentare apăsînd tasta M. Chiar şi cu un 8088 programul poate lansa un mare număr de taskuri înainte de a se încetini apreciabil. La programarea acestui nucleu multitasking cooperativ este important să nu uităm să plasăm o instrucţiune *pause* ori de cîte ori o funcţie este pasibilă de un consum mare de timp. De exemplu, rutina de intrare de la tastatură a fost scrisă în aşa fel încît să cicleze periodic prin *pause* în timp ce aşteaptă apăsarea unei taste, iar întîrzierile sînt realizate prin monitorarea ceasului BIOS şi executarea instrucţiunii *pause* pînă la scurgerea numărului dorit de tacturi. În aplicaţiile "înglobate" rutinele de intrare/ieşire execută *pause* între eşantionări sau în timp ce aşteaptă ca datele să fie disponibile. Rutinele de comunicaţie execută *pause* pînă cînd codul de tratare a întreruperii a complrtat un buffer de intrare sau a terminat o transmisie.

Concluzie

Multitaskingul cooperativ oferă unele din avantajele unor sisteme de operare multitasking puternice, în absenţa multor complicaţii. Pentru sisteme înglobate în alte echipamente, cînd resursele sînt adesea reduse, lipsa unui executiv şi a supervizorului asociat fac multitaskingul cooperativ să fie ideal. Pentru sisteme mici, se poate elimina chiar şi alocarea dinamică de memorie pentru a se obţine cod compact şi rapid. Mai mult chiar, pînă cînd sistemele de operare multitasking se vor răspîndi la o arie mai largă de utilizatori, multitaskingul cooperativ le permite celor care sînt nevoiţi să folosească taskuri multiple să scrie efectiv aplicaţiile în timp ce aşteaptă cu nerăbdare detronarea MS-DOS-ului.

C. N.

```
Task:: Switch ()
    push bp; salveaz{ cadrul stiv{
    mov bp, sp
    push si; salveaz{ registrul SI
    push di; salveaz{ registrul DI
    mov si, di; instrac'iunc dummy
...
_SI = _DI; //instruc'iune pentru a for'a salvarea SI, DI
saved = (int *)_SP; //memoreaz{ SP
Current Task = next; //fixare pe noul task
_SP = (int) Current Task - saved; //utilizeaz{ SP al noului task
...
    pop di; reface DI din noul task
    pop si; reface SI din noul task
    pop bp; reface cadrul stiv{ din noul task
    ret; revine la adresa de execu'ie a noului task
```

Figura 3: o parte din codul în limbaj de asamblare generat pentru Task:: Switch.

Listing Unu

```
// Fisier: TASK.H -- Obiecte task
// Fiecare obiect task contine propria sa stiva, legaturi cu obiectele task
// urmator si precedent din lista inlantuita, indicatorul salvat al
// virfului stivei si informatii pentru depanare

class Task {
    int *area;    // baza zonei stiva
    int id;    //numarul task-ului - pentru depanare
    Task *next;  // task urmator
    Task *prev;  // task precedent
    int *saved;   // locatia stivei salvata in timpul lui pause()
    int *top;// virful stivei
public:
    void  Activate( void (*)() );// functie de lansare
    int GetId()  // intoarce numarul task-ului
        {return id; }
    Task *GetNext() // intoarce pointer la task-ul urmator
        { return next; }
    int *GetSP()      // intoarce SP salvat
        { return saved; }
    void SetNext(Task *t) // fixeaza pointer la task-ul urmator
        { next = t; }
    void SetPrev(Task *t) // fixeaza pointer la task-ul precedent
        { prev = t; }
    void  Show();     // afiseaza date pentru depanare
    void  Switch();   // comuta contextul la task-ul urmator
    Task(int);        // constructor
    ~Task();          // distrugator
};

Task    *fork(void (*)() );
Task    *InitTasking();  // initializeaza nucleul multitasking
void    pause();       // comuta la task-ul urmator

extern int totalTasks;// numar total de task-uri - pentru depanare
```

Listing Doi

```
// Fisier: TASK.CPP // Multitasking cooperativ in C++
#include <conio.h>
#include <stdlib.h>
#include "task.h"

// Prototipuri
void    terminate(); // iesire din task la terminare

// Definiri
#define STACKSIZE 0x200  // marimea stivei fiecarui task

// Variabile globale
Task    *CurrentTask = 0;  // task-ul in executie
Task    *SystemTask; // task-ul principal care utilizeaza stiva sistemului
int totalTasks = 0; // contor de task-uri -- pentru depanare

// Task.Aactivate - pregateste stiva unui obiect task astfel incit, atunci
// cind se comuta la acest task sa se execute functia transmisa
void    Task::Activate( void (*f)() )
{
    saved = top;     // reset stiva
    *(--saved) = (int) &terminate; // "ucide" functia taskului la iesire
*(--saved) = (int) f; // plaseaza functia pentru adresa de revenire
    *(--saved) = _BP;    // salveaza toti registrii pentru comutare
    *(--saved) = _SI;    // salveaza SI
    *(--saved) = _DI;    // salveaza DI }

// Task.Show - afisarea informatiilor pentru depanare
void    Task::Show()
{
 cprintf("Task: %4i  area: %04X\n\r", id,area);
 cprintf(" top: %04X  saved: %04X\n\r",top,saved);
 cprintf("prev: %04X  next: %04X",prev,next); }

// Task.Switch - comuta contextul la noul obiect task
// din lista inlantuita. Salveaza indicatorul curent al virfului
// stivei, preia indicatorul virfului  stivei pentru task-ul urmator
// (dupa ce l-a facut task curent) si revine
void    Task::Switch()
{
 _SI = _DI;  // forteaza compilatorul sa salveze SI,DI
saved = (int *)_SP;  // memoreaza indicatorul virfului stivei
CurrentTask = next;
_SP = (int)CurrentTask-saved;// reface SP pentru noul task
}

// Task.Task - initializeaza noul obiect task. Plaseaza obiectul in lista
// inlantuita cu celelalte task-uri. Daca marimea este 0, nu aloca spatiu
// pentru stiva ci foloseste stiva sistemului.
Task::Task(int size)
{
static int      newid = 0;  // identificator unic pentru fiecare task
id = newid++;
totalTasks++;
if (size)
    {               // se creaza un task operator ?
    if ((area = (int *) malloc(size * 2)) == 0) //Nu, deci aloca
        {
        cprintf("Memorie insuficienta pentru a crea task-ul %i\n",id);
        exit(1);
        }
    top = area + size;   // fixeaza virful absolut al stivei
    saved = top;
    next = CurrentTask-GetNext(); // se leaga in lista task-urilor
    prev = CurrentTask;
    prev-SetNext(this);
    next-SetPrev(this);
    }else {             // nu aloca stiva
        top = (int *)_SP; // in schimb, preia stiva sistemului
        saved = top;
        next = this;       // deoarece este primul task, il fac sa
        prev = this;       // puncteze spre mine
    }
}
```

```
// Distrugatorul de task-uri - reda sistemului toata memoria alocata
Task:: ~Task()
{
 totalTasks--;
prev-SetNext(next);       // scoate acest task din lista inlantuita
next-SetPrev(prev);

 CurrentTask = next;     // noul rask curent
if (area)              // nu elibereaza daca nu este alocata stiva
    free(area);  // elibeareaza stiva obiectului
}
// fork - creaza un task nou care sa execute functia transmisa.
// La terminarea/ functiei task-ul este distrus automat.
// fork intoarce un pointer la noul obiect task sau NULL
// daca nu are spatiu de memorie
Task    *fork( void (*f)() )
{
 Task    *newtask;          // pointer la noul obiect task
// In modelele de memorie "small", malloc utilizeaza indicatorul
// virfului stivei pentru a determina daca exista memorie libera.
// Pentru a permite ramificarea din sub-task-uri, se "imprumuta" stiva
// sistemului pentru operatia malloc.
int  temp = _SP;              // salveaza SP curent
 _SP = (int)SystemTask-GetSP() - 20; // imprumuta stiva sistemului
// creaza noul obiect task
if ( (newtask = (Task *) new Task(STACKSIZE)) )
    newtask-Activate(f); // pregateste noul task sa execute functia
_SP = temp;                   // reface stiva initiala return newtask
return newtask;
}

// InitTasking - Initializeaza totul inainte de inceperea
// multitasking-ului. Aceasta functie trebuie apelata inainte
// de ramificarea oricarui alt task.
 // Ea creaza task-ul "sistem" cooptind stiva sistemului
// intr-un obiect task. (task nr. 0) si face CurrentTask sa puncteze
// la noul task operator.
Task    *InitTasking()
{
CurrentTask = (Task *) new Task(0); // creaza stiva
sistem SystemTask = CurrentTask; // fixeaza indicatorul
        // virfului stivei
return SystemTask; }

// pause - interfata non-obiect cu contextul comutat
void     pause()
{
 CurrentTask-Switch();   // comuta contextul in afara task-ului
}

// terminate - elimina task-ul curent la terminarea lui fork.
//  Acesata nu // nu este o metoda, dar adresa lui este plasata
// pe stiva initiala si daca  functia task-ului se intoarce, terminate
// va fi urmatoarea adresa de executie
void     terminate()
{
 _DI = _SI;
delete CurrentTask; // "ucide" task-ul
curent _SP = (int) CurrentTask-GetSP();// fixare pe stiva task-ului
        // urmator si revenire in noul task
}
```

Listing Trei

```
// Fisier: DEMO.CPP
// Program demonstrativ pentru obiecte de multitasking cooperativ

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>.h
#include <time.h>
#include <conio.h>
#include <string.h>
#include <bios.h>
#include <ctype.h>
#include <dos.h>

#include "task.h"
#include "twindow.h"

//Prototipuri pentru functiile demonstrative
void endlessCount();
void fiveseconds();
void funwindow();
void msdelay(int);
int newgetch();
void periodic();
void quicky();
void status();
void wallclock();

main()
{
 /* Initializeaza nucleul multitasking. Creaza task-ul tata sistem */
InitTasking();          // initializeaza task, coopteaza stiva sistem
 /* -- initializare ecran -- */
textattr(WHITE);    // text "normal - alb pe negru
clrscr();       // sterge ecranul
_setcursortype(_NOCURSOR); // fara cursor

 /* -- lanseaza citeva task-uri -- */
fork( &endlessCount );   // lanseaza task-ul de numarare infinita
fork( &wallclock );   // lanseaza task-ul ceas
fork( &periodic );    // lanseaza lansatorul periodic de task-uri
fork( &funwindow );  // lanseaza o fereastra stranie
fork( &status );  // lanseaza functia de numarara a task-urilor

 /* -- creaza fereastra principala -- */
TextWindow myWindow(1,20,80,25, (LIGHTGRAY<4)+BLUE);
gotoxy(20,1);
cputs("*** Multitasking Cooperativ - Demonstratie ***\r\n");
cputs("  Fiecare fereastra este un obiect task separat ");
cputs("care executa o functie C++.\r\n");
cputs("Toate ruleaza 'concurent' utilizind\r\n");
cputs("rutina de comutare a contextului pause().");
```

```
gotoxy(2,6);
cputs("Comenzi: [C]reerea unui nou task, [I]esire");

/* -- asteapta o tasta si o proceseaza -- */
for(;;)
    switch ( toupper( newgetch() ) )
    {
        case 'I': // iesire - sterge ecranul si iese
            window(1,1,80,24);
            textattr(WHITE);
            clrscr();
            _setcursortype(_NORMALCURSOR);
            return(0);
        case 'C': // genereaza un nou task
            fork(&quicky);
            break;
        default: // caracter ilegal
            sound(500);
            msdelay(160);
            nosound();
            break;
    }
}

// endlessCount - deschide o fereastra si numara fara oprire
void endlessCount()
{
TextWindow myWindow( 40,7,80,9,(CYAN<4)+RED);
cprintf("Acest task nu se termina niciodata!");
long count = 0;
for(;;)
    {
    // numara, dar nu uita sa faci pause()
    myWindow.Activate();
    gotoxy(2,2);
    cprintf("Numarator: %li", count++);
    pause();      // lasa si alte task-uri sa ruleze
    }
}

// fiveseconds - deschide o fereastra, numara 5 secunde si iese
void fiveseconds()
{
TextWindow myWindow(5,5,25,8,(GREEN<4)+RED);
cprintf(" Task temporar care");
gotoxy(2,2);
cprintf("dureaza 5 secunde");

time_t t;          // citeste ora curenta
t = time(NULL);

int i = 10000;    // decrementeaza de la 10000
while( difftime(time(NULL), t) < 5 ) {
    // continua decrementarea
    myWindow.Activate(); // pina cind difftime este 5 secunde
    gotoxy(6,3);      // sau mai mult
    textcolor(BLACK);
    cprintf("%5i",i--);
    pause();          // lasa si alte task-uri sa ruleze
    }
}

// funwindow - afiseaza un caracter mobil in fereastra
void funwindow()
{
TextWindow myWindow(60,12,79,12, (CYAN<4) + YELLOW);
for( int i = 0; ; i = ++i % 20 )
    { //muta i de la 0 la 19
    myWindow.Activate();
    cputs(" ");
    gotoxy( abs( ((i/20) * 20) + i) + 1, 1);
    textcolor( rand() % 15 );
    cputs("$");
    msdelay(100);    // asteapta 100 ms
    }
}
// msdelay - intirziere cu un numar de milisecunde
// cu rezolutie de 55 ms
void msdelay(int delay)
{
long ticksplus = biostime(0,0L) + delay / 55;
while (biostime(0,0L)  ticksplus )// asteapta pina trece timpul
    pause();  // lasa si alte task-uri sa ruleze
}

// newgetch - face acelasi lucru ca si getch, cu exceptia asteptarii
 // unei taste
int newgetch()
{
while ( !kbhit() )
    pause();
return getch();
}

// periodic - lanseaza periodic un alt task
void periodic()
{
TextWindow myWindow(1,10,65,2,(LIGHTGRAY<4)
    +MAGENTA);
cputs("Lanseaza un task temporar la fiecare 10 secunde");
for(;;)
    {
    for( int i = 0; i  10; i++ )
    {   // bucleaza de 10 ori inainte de ramificare
        myWindow.Activate();
        gotoxy(20,2);
        cprintf("%i",i);
        msdelay(1000);  // intirziere de o secunda
    }
    fork(&fiveseconds);  // lanseaza noul task ce moare dupa 5 secunde
    }
}

// quicky - deschide ferestre, ramine activ citeva secunde si iese
void quicky()
{
```

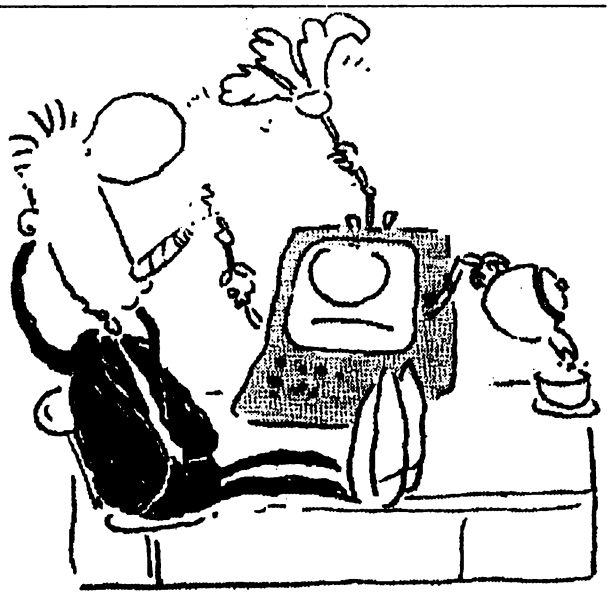
```
static int xpos = 3;   // pozitia x a ferestrei noului task
static int ypos = 3;   // pozitia y a ferestrei noului task
TextWindow myWindow(xpos+1,ypos+1,xpos+8,ypos+6,
        (RED<4)+WHITE);
xpos = (xpos+3) % 64;
ypos = ++ypos % 7;

for( int i = 0; i  10; i++ )
    {   // decrementeaza 10 secunde
    myWindow.Activate();
    cprintf("Mort in 10 sec.: %i",i);
    msdelay(1000);        // intirziere de 10 secunde
    }
}

// status - afiseaza numarul de taskuri care ruleaza
void status() { TextWindow myWindow(1,1,18,1, (MAGENTA<4) +
YELLOW);
for(;;) { myWindow.Activate();
Tasks);
    msdelay(2000);        // asteapta 200 milisecunde } }

// wallclock - afiseaza permanent ora void wallclock()
{ TextWindow myWindow(54,1,80,1,(RED<4) + BLUE);
time_t t;              // va mentine ora
char buf[40];     // buffer temporar pentru a elimina \n
for(;;)
    {             // actualizarea orei se face permanent
    myWindow.Activate();
    t =time(NULL); // preia adresa sirului de caractere cu ora
    strcpy(buf,ctime(&t)); // copiaza sirul in temp
    buf[24] = '\0';       // elimina \n
    cprintf("%s",buf);
    msdelay(1000);        // asteapta o secunda
    }
}
```

Listing patru

```
// Fisier: TWINDOW.H -- obiecte fereastra pentruprogramul
    // demonstrativ
// Presupune functiile de biblioteca Borland C++

class TextWindow {
    int attrib;    // atribut pentru modul text
    int left, top; // pozitii x,y de inceput
    int right, bottom;     // pozitii x,y finale

public:
    void  Activate();      // activare
    TextWindow(int,int,int,int,int); // constuctor
    ~TextWindow();     // distrugator
}
```

Listing cinci

```
// Fisier: TWINDOW.CPP - obiecte feresatra text

#include "twindow.h"
#include <stdio.h>
#include <conio.h>

// activarea unei ferestre
void TextWindow::Activate()
{

window(left,top,right,bottom); // utilizeaza biblioteca C++
 textattr(attrib);
}

// constructor de ferestre - memoreaza coordonatele si sterge fereastra
TextWindow::TextWindow(int l,int t,int r,int b,int a)
{cp

left = l;
top = t;
right = r;
bottom = b;
attrib = a;
 Activate();      // activeaza fereastra
clrscr();      // sterge fereastra
}

// distrugatorul de ferestre - sterge ferestrele si le face fundalul negru
TextWindow::~TextWindow()
{

 Activate();
textattr( (BLACK < < 4) + WHITE);
clrscr();
}
```



PC-urile ca sisteme de achiziţie de date în reţea

Reţelele locale de calculatoare (Local Area Network - LAN), au început să capete o mai largă răspîndire doar în ultimii ani. Cuvinte "cheie" ca Workstation, File Server, Token Ring, NETBIOS, ETHERNET, ARCNET etc., au început să facă parte din jargonul zilnic al utilizatorilor de PC-uri. Calculatorul personal (PC), este "forţa majoră" care singură, este responsabilă pentru aceste fapte.

PC-ul, a fost conceput iniţial pentru a fi un înlocuitor ieftin al minicalculatoarelor, deoarece era mult mai accesibil majorităţii utilizatorilor. Pe măsură ce numărul de utilizatori creştea, aceştia simţeau tot mai mult nevoia de a-şi comunica datele între ei.

Datorită cantităţii PC-urilor IBM XT/AT şi compatibile existente, tot mai mulţi producători au început să dezvolte soft şi hard pentru a-şi putea realiza aplicaţiile lor în reţea. Astfel, un număr mare de companii au produs interfeţe montabile în PC, ce realizau toate funcţiile controllerelor de reţea (Ethernet, Arcnet, Token Ring) şi tot mai multe versiuni de soft de reţea apăreau în mod regulat.

PC-urile erau iniţial utilizate în birouri pentru a realiza task-uri (sarcini) cum ar fi: editare şi prelucrare de texte, gestionarea de baze de date şi calcul tabelar, apoi au început să-şi găsească drumul spre laboratoare şi întreprinderi industriale, pentru a efectua măsurători şi a controla procese în timp real. Reţelele azi, urmăresc acelaşi drum.

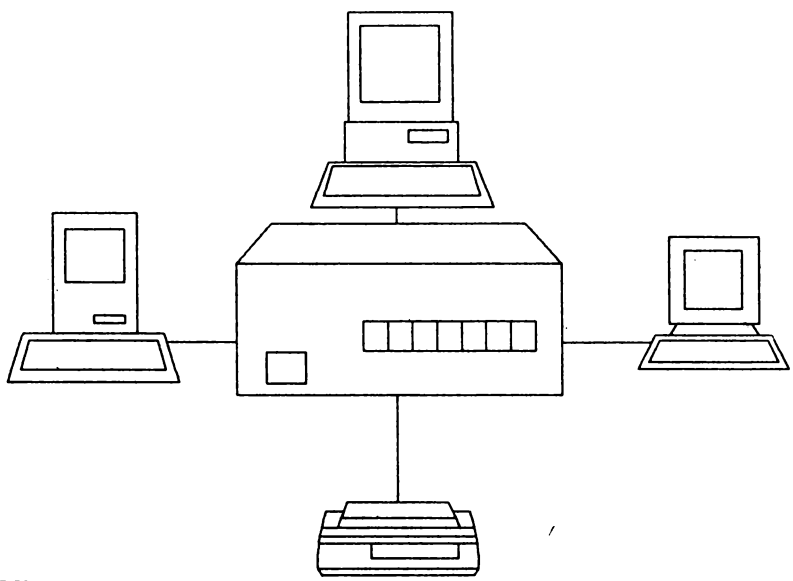
Încrederea obţinută în sistemele de control ale proceselor bazate pe PC-uri se bazează atît pe produsele hardware cît şi pe cele software apărute. Utilizatorii din industrie caută soluţii de a conecta calculatoarele industriale, utilizînd atît tehnica reţelelor DISTRIBUITE cît şi pe cea a controlului unui proces de la un calculator de proces central GAZDĂ - HOST, chiar şi în cazul în care punctele de măsură ce trebuie monitorizate şi controlate sînt răspîndite pe o arie geografică mare.

De asemenea, pentru a putea colecta datele de la punctele de măsură aflate la distanţă, inginerii de proces studiază posibilitatea de a controla un anumit proces dintr-un punct aflat la o distanţă oarecare faţă de locul unde se află calculatorul de proces. Informaţia derivată din proces, (calitate, cantitate, etc.) este folosită la un nivel superior, făcînd parte integrantă din fluxul total informaţional al întreprinderii.

În multe aplicaţii, calculatoarele IBM PC-XT/AT/386 şi compatibile ce lucrează sub sistemul de operare MS-DOS au reprezentat focarul tuturor aplicaţiilor bazate pe măsură şi control, iar calculatoarele microVAX şi VAX sînt mai mult utilizate ca SUPERVIZOR şi sistem de management informaţional în reţea.

În aplicaţii de control de sine stătătoare (ce nu lucrează în reţea), s-au realizat un număr de aplicaţii soft excelente pentru achiziţii de date şi control pentru laborator şi industrie. Dintre acestea cele mai populare pachete de programe includ LABTECH NOTEBOOK pentru achiziţii de date şi LT/CONTROL precum şi GENESIS pentru controlul proceselor.

Recent, proiectanţii acestor pachete de programe, le-au dezvoltat mai departe pentru a permite lucrul



în reţele distribuite (PEER - TO-PEER) cît şi în reţele IERARHICE de tip GAZDĂ-LA-CLIENT - HOST-TO-CLIENT.

Ofertele firmei Laboratory Technologies Corporation, se bazează atît pe pachetul de programe LT/CONTROL ce rulează la nivelul calculatorului GAZDĂ din camera de control, cît şi pe LT/CONTROL sau LABTECHNOTEBOOK, ce rulează în nodurile din reţeaua întreprinderii.

Firma ICONICS, producătorul pachetului de programe GENESIS, a anunţat recent programul GEN-NET, care este un sistem distribuit ce rulează în fiecare nod de reţea, oferind posibilitatea de comunicare între noduri, pe baza unei legături ce poate fi Ethernet, Arcnet sau Token Ring, operînd sub NETBIOS.

Ambele opţiuni de reţea, LT/CONTROL şi GENESIS permit ca un program ce rulează într-un calculator să folosească date dintr-un alt program ce rulează într-un alt calculator. Reţelele distribuite permit, ca datele achiziţionate în timp real, fişierele şi mesajele de alarmă să poată fi transmise la toate nodurile reţelei.

Reţele distribuite

Reţelele distribuite, se folosesc în general în aplicaţii în care există mai multe puncte de control, care centralizează informaţii dintr-o întreprindere sau dintr-un proces complex şi datele generate într-un punct afectează luarea unei decizii într-un alt punct. În aceste aplicaţii, în fiecare nod se folosesc de obicei calculatoare de aceeaşi putere de calcul. Un inginer de proces va avea nevoie de posibilitatea afişării vizuale pe un monitor în fiecare nod al reţelei. Această configuraţie este de asemenea utilizată în aplicaţii care cer o independenţă în fiecare nod, astfel ca în cazul defectării unui nod, acest lucru să nu afecteze alte noduri ale reţelei.

Pentru acest tip de aplicaţii sînt necesare pachete soft complete, pentru a adresa punctele de intrare/ieşire (I/O) din fiecare nod. Aceste pachete trebuie să dea inginerului, posibilitatea de a-şi dezvolta propriile sale strategii de control, folosind algoritmi standard pentru a controla intrările şi ieşirile analogice şi numerice.

Algoritmii trebuie să includă funcţii ca PID, filtrare, Leadleg şi Deadtime. Pachetul va trebui de asemenea să poată efectua operaţii trigonometrice, matematice şi loğice. În timpul rulării, posibilitatea de a memora date de la diferite puncte de măsură preselectate, este obligatorie. Azi, aceste programe trebuie să fie capabile să lucreze şi în reţea.

Tendinţele de perspectivă ale acestor pachete software sînt de a include din ce în ce mai multe funcţii, ceea ce însă, de cele mai multe ori, are ca efect creşterea nevoii de memorie din ce în ce mai mare. Adaptoarele grafice cel mai des utilizate sînt EGA şi VGA. Necesităţile de stocare se vor concretiza în utilizarea unui harddisk de maximum 40 MBytes. În spiritul unei eventuale dezvoltări ulterioare, este bine ca sistemul ales să fie bazat pe procesorul 80286 sau 80386.

Datorită necesităţii memoriei adiţionale cerute de soft-ul de reţea, sistemele trebuie să fie prevăzute cu memorie expandată sau extinsă. Este de dorit utilizarea unei memorii între 1 şi 4 MBytes.

Genesis - în reţele distribuite

Pachetul de programe GEN-NET al firmei ICONICS, oferă posibilitatea lucrului în reţele distribuite, permiţînd ca datele culese în timp real de la diferitele puncte de măsură, fişierele şi mesajele de alarmă să poată fi transmise în reţea între multiplele noduri GENESIS - compatibile. Fiecare nod GENESIS - compatibil, este capabil să funcţioneze INDEPENDENT, în timp real şi să asigure în continuare funcţionarea reţelei în cazul căderii unui nod.

GEN-NET este compatibil cu reţelele bazate pe NETBIOS sau ARCNET. NETBIOS - un protocol de comunicaţii pentru reţele distribuite dezvoltat de IBM - este un standard DE FACTO şi permite realizarea unor conexiuni virtuale între PC-urile reţelei. NETBIOS este livrabil pentru toate reţelele standard (Ethernet, Arcnet şi Token Ring) ca şi pentru alte tipuri de reţele industriale.

GEN-NET este integrat în pachetul GENESIS, prin tehnicile sale bazate pe CAD, iconuri şi uşurinţa de folosire. Nodurile noi ale reţelei, se pot adăuga sau modifica cu aceeaşi uşurinţă ca şi modificarea funcţiilor de I/O. GEN-NET include un soft de reţea şi nu necesită alte pachete de soft pentru gestionarea reţelei.

LT/Control în reţele distribuite

Spre deosebire de GENESIS, LT/CONTROL nu are inclus şi soft-ul de reţea şi de aceea nu poate opera într-o reţea distribuită decît dacă foloseşte un pachet de software de reţea standard ca de exemplu IBM PC LAN, 10 NET sau LANTASTIC. Pachetul LANTASTIC, elaborat de firma Artisoft Inc., are faţă de celelalte avantajul că foloseşte foarte puţină memorie RAM şi are un preţ de cost scăzut.

Reţelele distribuite au un preţ de cost scăzut, permiţînd posibilitatea interschimbării datelor între calculatoarele din nodurile reţelei, ce controlează procesul. Performanţele nu sînt chiar atît de bune ca şi în cazul reţelelor construite pe principiul IERARHIEI, dar oricum aceasta se datorează mai mult sistemului de operare MS-DOS, care constituie o frînă pentru transferul datelor la viteze mari. De asemenea, trebuie acordată o mai mare atenţie securităţii datelor, pentru că

acest lucru este mai greu de făcut în reţelele distribuite.

Reţele locale ierarhice

Într-o reţea locală ierarhică, se utilizează un FILE SERVER pentru a pune la dispoziţia staţiilor de lucru soft-ul necesar realizării funcţiilor de achiziţie de date în reţea. Fiecare staţie de lucru funcţionează independent, dar este legată la un calculator GAZDĂ - HOST, care rulează un program bazat pe strategia citirii intrărilor de la staţiile din nodurile reţelei. Staţiile de lucru din noduri nu pot comunica între ele în mod direct, decît prin intermediul FILE SERVER-ului.

FILE SERVER-ul şi calculatorul GAZDĂ pot fi un acelaşi calculator, sau pot fi două calculatoare diferite. Datorită faptului că performanţa întregului sistem depinde de caracteristicile FILE SERVER-ului, este foarte important alegerea unui FILE SERVER cu performanţe ridicate, în special dacă el şi calculatorul GAZDĂ sînt realizate în acelaşi PC.

În configuraţiile ierarhice, performanţele pe care trebuie să le aibă calculatoarele din noduri sînt minimale. În majoritatea situaţiilor acestea pot fi XT-uri fără harddisk prevăzute cu un monitor, necesar pentru a afişa mesaje unui operator.

LT/Control

în reţele ierarhice

În aplicaţiile ierarhice, LT/CONTROL poate opera în calculatorul GAZDĂ, definind strategia folosirii în timp real a datelor provenite de la staţiile din nodurile reţelei. Un monitor ce serveşte ca interfaţă utilizator poate fi ataşat la calculatorul GAZDĂ, pentru a da posibilitatea inginerului de proces de a avea un control activ asupra întregii reţele.

Soft-ul ce rulează pe staţiile din nodurile reţelei, are cerinţe mici. El trebuie să acceseze punctele de măsură analogice, să le digitizeze şi să trimită valorile la calculatorul GAZDĂ, în timp real. Acest program trebuie de asemenea să accepte date de la calculatorul GAZDĂ şi să le dirijeze la modulele de ieşiri analogice şi numerice. "Închiderea buclei" se realizează la nivelul calculatorului GAZDĂ. Staţiile din nodurile reţelei funcţionează ca nişte dispozitive de I/O.

Cerinţele soft ale calculatoarelor din nodurile reţelei se realizează uşor cu versiunile RUN-TIME ale pachetului de programe LABTECH NOTEBOOK. Acesta este un pachet de programe de I/O ce poate dirija în timp real datele la, sau de la, reţea. Transferul de date către calculatorul GAZDĂ se poate face cu viteze de pînă la 10 MB/sec.

NOTEBOOK poate rula atît în modul HIGH SPEED - viteză ridicată (fără monitor, datele fiind trimise cu viteze mari) cît şi în modul NORMAL (datele colectate transmiţîndu-se cu viteze mai mici, dar datele în timp real putînd fi monitorizate pe un monitor conectat la o staţie dintr-un nod oarecare).

Controlul întregii reţele este realizat de către FILE SERVER, în care se instalează un sistem de operare de reţea cu rolul de a dirija traficul în reţea. Piaţa oferă astăzi un număr mare de asemenea sisteme de operare în reţea dintre care însă cel mai popular este NOVELL NETWARE, ce reprezintă un standard DE FACTO. Acest sistem de operare are diferite versiuni în funcţie de numărul de utilizatori (de la 4 la 100).

Genesis în reţele ierarhice

Pachetul de programe GENESIS, poate de asemenea lucra în configuraţii ierarhice, utilizînd diverse drivere soft de reţea ca de exemplu, DACNET, dezvoltat de firma ADAC Corporation pentru interfeţele ARCNET. În această configuraţie, GENESIS se lansează în calculatorul GAZDĂ, stabilind o strategie de preluare a datelor obţinute de la staţiile de lucru din noduri (ce pot fi configuraţii hardware cu preţ de cost scăzut), fiecare dintre ele putînd fi bazate pe PC-uri configurate ca sisteme de achiziţie de date în care rulează un program relativ simplu de colectare a datelor.

În orice nod, programul ADLI-BI/O, scanează continuu toate intrările, atît cele analogice, cît şi cele numerice şi memorează datele rezultate în matrici de memorie de intrare locale. Nodul poate de asemenea accepta date de la calculatorul GAZDĂ, pe care le memorează în matricile de memorie locale de ieşire. La nivelul calculatorului GAZDĂ, dacă GENESIS necesită o dată de intrare de la un nod, transmite o cerere de date de intrare via driver-ul DACNET. Nodul răspunde prin trimiterea datelor cerute prin reţea la GAZDĂ. Calculatorul GAZDĂ operează asupra datelor folosind algoritmi de control din pachetul de programe GENESIS generînd un fişier de ieşire pentru ieşirile analogice sau numerice.

Astfel, sistemul lucrează folosind o strategie bazată pe scanare şi cerere. Datele nu sînt puse în reţea decît dacă ele sînt cerute de calculatorul GAZDĂ, eliminînd astfel eventualele coliziuni ale acestora.

Topologia este viabilă doar în cazul în care atît calculatorul GAZDĂ cît şi staţiile de lucru din noduri sînt funcţionale. În cazul în care calculatorul GAZDĂ "cade", acest lucru este detectat de soft-ul de reţea care pune întreaga reţea sub controlul unui program aflat în memoria sau pe harddisk-ul uneia dintre staţiile de lucru dintr-un nod.

Kallo Tibor

Performanță în grafica 3-D:

AutoCAD Release 11

Pentru a desena obiecte care să arate ca în realitate aveți nevoie într-adevăr de un sistem în trei dimensiuni. Desenele de arhitectură par să prindă viață cînd le construiești în 3-D și adaugi umbriri naturale elementelor constructive. Construirea unui model tridimensional al unui automobil vă permite să vă rotiți în jurul acestuia avînd posibilitatea să-l priviți din orice unghi doriți. Este mult mai eficient să priviți dintr-o poziție mai îndepărtată la o imagine 3-D a obiectului, așa cum este el perceput în realitate, decît să priviți trei reprezentări disjuncte în 2-D ale acestuia. Orice proiectant poate beneficia de posibilitatea de a privi imagini multiple ale obiectelor fără să fie nevoit să creeze desene pentru fiecare perspectivă în parte, economisind astfel multe ore în procesul proiectării. Instrumentele pentru generarea de suprafețe reale, precum și "plimbări" animate prin structuri arhitecturale vă pot ajuta să comunicați mai eficient ideile Dvs. și probabil să vă puneți semnătura mult mai rapid pe un proiect finalizat.

În cei zece ani de existență, AutoCAD a stabilit multe standarde industriale pentru proiectarea și desenarea pe calculatoare sub DOS. AutoCAD continuă să aibă o poziție foarte puternică, deoarece este cea mai larg folosită și ușor integrabilă aplicație CAD. Împreună cu cea mai recentă versiune, AutoCAD Release 11, Autodesk furnizează opțiunea Advanced Modeling Extension (AME) pentru a adăuga AutoCAD-ului facilitatea de a modela obiecte solide 3D. AME este integrat pe deplin în AutoCAD și puteți folosi același set de comenzi pentru crearea și manipularea obiectelor solide. Pentru sarcini mai complexe, cum ar fi operații booleene, analiză finită și dinamică, și detecția de interferențe, comenzile AME sînt indispensabile. Indiferent că începeți cu forme solide de bază sau rotiți și extrudați obiecte 2D, puteți realiza foarte repede ascunderea liniilor și generarea suprafețelor. Pentru ca AutoCAD-ul să obțină modele care să arate ca în realitate și în culori naturale, cu parametri de iluminare și texturi, este nevoie de extensia opțională AutoShade; cumpărînd versiunea 2.0 a acesteia beneficiați și de înalta tehnologie inclusă de pachetul RenderMan. Această opțiune vă permite să controlați caracteristicile suprafețelor, incluzînd gradul de opacitate, textură, reflectivitate, precum și un număr nelimitat de surse de lumină. Se pot importa imagini scanate care pot deveni astfel texturi pentru suprafețe. Modelele astfel realizate pot fi animate, existînd și posibilitatea de a le studia prin deplasarea unei camere de luat vederi prin interiorul structurilor proiectate.

Avînd la dispoziție redarea cu 16,7 milioane nuanțe cromatice într-un sistem organizat pe 24 de biți veți fi limitați doar de structura hardware, capacitatea memoriei și de propria dumneavoastră imaginație.

Arhitectura deschisă a programului AutoCAD vă permite flexibilitatea de a-l adapta pentru aplicații diferite. Această adaptabilitate este una din caracteristicile remarcabile ale pachetului. AutoCAD are un limbaj de programare integrat, AutoLisp, care vă dă posibilitatea să rulați aplicații scrise în AutoLisp realizate de terțe firme.

Cu ajutorul interfeței ADS (AutoCAD Development System) puteți lucra cu limbaje de programare de nivel înalt cum este limbajul C pentru a

CE FACE SOFT-UL DE CAD TRIDIMENSIONAL?

Automatizează procesul de realizare a desenelor complexe și ușurează efectuarea reviziilor. Produsele performante furnizează instrumente pentru generarea suprafețelor reale și animarea parametrică.

AVANTAJE

Desenele finale arată foarte bine și sînt ușor de actualizat. Folosind mai degrabă programe 3-D decît 2-D aveți posibilitatea să generați vederi noi fără a fi necesar să realizați un nou desen.

DEZAVANTAJE

Aceste pachete de soft au o curbă de învățare abruptă. Introducerea informației 3-D cu ajutorul unui dispozitiv de intrare 2-D poate să nu fie la îndemîna oricui.

RECOMANDĂRI

AutoCAD 386 este standardul industrial din motive bine întemeiate:

- este sistemul CAD cel mai răspîndit printre utilizatori (peste 400.000 instalări în lumea întreagă)
- este utilizat de ingineri și proiectanți în aplicatii specifice pentru arhitectură, constructii civile, inginerie mecanică, electrică și electronică, chimică, sisteme informatice geografice (GIS), facilități de conducere și planificare, desktop publishing și ilustrare tehnică, ș.a..
- rulează pe o gamă largă de platforme de calcul: PC/MS-DOS, OS/2 și SCO Xenix 386, Apple Macintosh II, Sun Microsystem, DEC și Apollo.
- există suport pentru sute de dispozitive periferice.
- are o arhitectură deschisă - se poate foarte ușor adapta de către utilizator
- există peste 600 pachete de aplicații dedicate
- modelează tridimensional prin muchii (wire-frame) sau prin suprafete (corp solid).
- are facilități pentru schimbul de fișiere cu alte programe în multiple formate (DXF, IGES, HPGL, WPG, RIB, ASCII, coordonate. x,y,z)
- vizualizare dinamică cu perspectivă.
- rețele extinse de centre de instruire și grupuri de utilizatori independente.

face un pas mai departe în adaptarea programului AutoCAD.

Structura flexibilă de comenzi a programului, capabilă să satisfacă orice preferință personală, vă permite să alegeți comenzi din meniurile situate în partea dreaptă sau în partea superioară a ecranului, meniuri fereastră (disponibile cu anumite drivere pentru ecran), meniuri pentru tablete grafice și meniuri de tip buton.

Un model complex poate fi un teren bogat pentru compararea instrumentelor din pachetul de bază AutoCAD cu cele introduse prin extensia sa AME. Să presupunem că într-un desen de arhitectură o coloană a unui templu are 25 de caneluri sau corpuri de rotație. Printr-o abordare cu modelare de corp solid specifică AME putem construi doar 24 de caneluri, dar cu comenzile de bază AutoCAD am putea desena 25.

Folosind însă Coons Patch (o pînză de suprafață complexă reprezentată printr-o serie de noduri sau puncte) se pot ascunde intersecțiile dintre elementele unei construcții 3-D.

Pentru desenarea unei cupole folosindu-se modelarea de corp solid, prin operații booleene de scădere, se poate tăia o sferă la nivelul intersecției cu pereții construcției.

AutoCAD poate rula într-o configurație cu unul sau două monitoare (al doilea monitor afișează linia de comandă și meniurile de comenzi). În modul protejat pe 32 de biți AutoCAD folosește un manager de memorie virtuală pentru a accesa direct toată memoria din sistemul dumneavoastră fără să utilizeze memoria expandată. AutoCAD este compatibil de asemenea cu majoritatea rețelelor bazate pe calculatoare personale.

După revista "PC Magazine"

Vincențiu Turică

Probleme în grafica 3-D

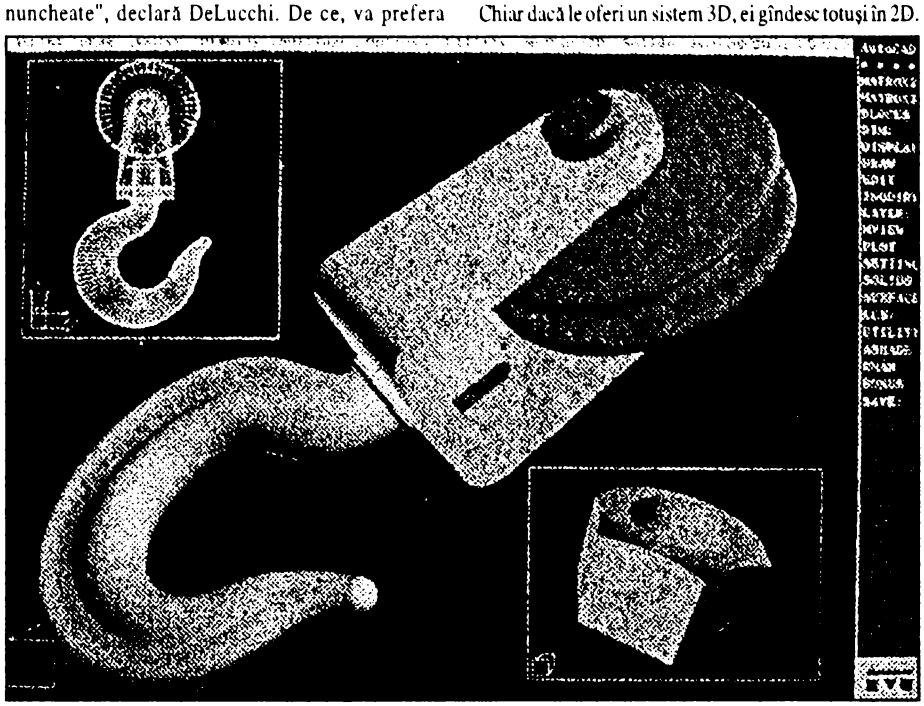
Cel mai popular pachet de programe pe piața de microCAD este AutoCAD al firmei Autodesk. Într-un anumit sens, deși nu tocmai complet, este cel mai performant din zona CAD pe PC. Majoritatea utilizatorilor îl consideră un program dificil de utilizat, dar este un rău necesar și merită să depășești dificultățile de învățare. Chris DeLucchi este președintele AutoCAD User's Group din San Diego și autorul seriei "The AutoCAD Cookbook". Acesta declară: "AutoCAD poate să nu fie cel mai bun, dar este foarte ușor adaptabil după preferințe și este parte a unei infrastructuri foarte largi - mulți producători au dezvoltat pe acest nucleu extensii dedicate". Există literalmente sute de aplicații dezvoltate pe acest nucleu, cu ajutorul cărora AutoCAD poate face orice vă puteți imagina fără a fi necesar un efort de programare în AutoLisp. Alte pachete de programe destinate proiectării asistate au limbaje de programare proprii, dar niciunul nu are aceeași popularitate ca limbajul Autolisp. AutoCAD este disponibil pe mai multe platforme decît orice altă aplicație CAD: PC, MacIntosh, diferite stații grafice sub sistemul de operare Unix. Totuși, în grafica 3-D, "pentru generarea unor modele puțin mai complexe, toate aplicațiile de nivel mediu, incluzînd AutoCAD, sînt îngenuncheate", declară DeLucchi. De ce, va prefera cineva să utilizeze AutoCAD dintre toate pachetele de programe disponibile pe PC?

"Mai întîi, datorită adaptabilității sale. Mulți gîndesc că nu este plăcut și nici ușor de utilizat, dar pentru a elimina această piedică ca să-l faceți mai prietenos, îl puteți modifica foarte ușor. AutoCAD este un program foarte îndrăgit.

Al doilea motiv ar fi infrastructura acestuia - grupuri organizate de utilizatori (User Groups), cărți, reviste, dealeri, școli, precum și operatori experimentați".

Dave Sander, proiectant experimentat, consultant în probleme de CAD, declară: "Partea cea mai costisitoare a realizării unui desen nu este nici calculatorul și nici softul, ci salariul pe care instituția îl plătește operatorului pentru ceea ce face. Alegeți programul care are cea mai adecvată interfață cu activitatea Dvs. Pe de altă parte, asigurați-vă că aveți destulă putere de calcul ca să nu fiți nevoit să așteptați după computer".

Sfatul lui Sander? "Întrebați-vă ce face în prezent departamentul din care faceți parte. Dacă e vorba despre proiecte simple sau care nu implică comunicația cu alte instituții, puteți alege un sistem CAD mai puțin performant. Pe măsură ce activitatea Dvs se dezvoltă, ușurința în utilizare devine mai puțin importantă. Asta înseamnă că aveți mai multe persoane care folosesc softul în mod regulat, tinzînd astfel să devină mai experimentați. Problema nu este că ei trebuie să piardă trei luni învățînd să lucreze într-un sistem. Problema cu programele 3D este că oamenii nu sînt obișnuiți să gîndească tridimensional. Desenatorii au fost antrenați ani de zile cu reprezentările bidimensionale pe hîrtie. Chiar dacă le oferi un sistem 3D, ei gîndesc totuși în 2D.



FOXPRO 2.0
Tehnologia Rushmore

Reacţia faţă de tehnologia Rushmore a lui Fox a generat două păreri opuse. Unii o compară cu faimosul citat al lui Arthur Clarke: "Orice tehnologie suficient de avansată nu poate fi distinsă de magie". Ceilalţi răspund: "Orice tehnologie suficient de avansată nu poate fi distinsă de o demonstraţie dichisită". Care este adevărul?

Era foarte greu de dat un răspuns după privirea fugară de la prezentare. Preşedintele lui Fox Software, dr. David Fulton, a deschis o bază de date cu un milion de articole şi a tastat comanda "COUNT FOR Street = Main Street' AND State = 'CA'. În mai puţin de o secundă interogarea s-a executat, auditoriul a fost buimăcit, iar Fulton a jubilat.

La un studiu mai atent, se descoperă că unele clauze utile, cum ar fi SET ORDER, pot înrăutăţi performanţele lui Rushmore.

Deci, locul ocupat de Rushmore în intervalul dintre magia autentică şi iluzia deşartă depinde de necesităţile aplicaţiei, ceea ce va încerca să clarifice şi articolul de faţă.

Deşi este posibil ca, în viitor, Fox să îmbunătăţească această tehnologie, un lucru este sigur: Rushmore este rapid. Este uimitoare viteza cu care, după o interogare iniţială se desfăşoară interogările ulterioare. Fox a avut dintotdeauna algoritmi de "cache" foarte buni, dar aceasta nu poate fi singura explicaţie. De exemplu:

```
COUNT FOR Sex = 'M' AND
Age >100

COUNT FOR Sex = 'M' AND
Age < 10
```

Dacă prima interogare, pe o bază de date cu un milion de articole, se face leneş - în patru secunde, cea de-a doua interogare se termină înainte de a apuca să-ţi priveşti ceasul. Este limpede că Rushmore memorează un fel de rezultate intermediare ale primei interogări. Această tehnică face ca al doilea COUNT să nu o ia de la început. Rushmore poate să memoreze rezultatele unor rutine pentru cazul în care se doreşte rularea acelor rutine mai tîrziu.

Un alt aspect interesant al tehnologiei Rushmore este "incompatibilitatea" cu date ordonate. Alegerea unei ordonări diferită de ordinea fizică a articolelor poate, pentru baze de date mari, să facă un BROWSE FOR încet ca un melc.

Misterul pare să se dezlege dacă reflectăm asupra limitării de 500000 de articole pentru unele utilizări - limitare motivată de Fox prin "motive tehnice întemeiate". Unii utilizatori susţin că această limitare este legată de indecşi. S-a emis ipoteza că Rushmore utilizează hărţi de biţi pentru a memora rezultatele fiecărei expresii ce a corespuns unui cîmp indexat, cu cîte un bit pentru un articol. Aceasta ar comprima considerabil informaţia necesară găsirii articolelor ce satisfac condiţiile unei interogări. De asemenea, explică limitarea la 500000 de articole pentru utilizarea lui Rushmore în versiunile non - 386: limitarea segmentelor la 64K în arhitecturile 8086 şi 80286 ar putea explica această limită, deoarece în 64K nu se pot păstra hărţi de biţi mai mari de 512000 de elemente. În modul protejat 386, mărimea segmentelor este mult superioară, ceea ce ar însemna că Rushmore ar putea lucra cu baze de date ce se apropie de 32 miliarde de articole.

Dar cum crează şi cum utilizează Rushmore hărţile de biţi? Să luăm exemplul următor:

```
LOCATE FOR Charges >
100000 AND MONTH(In-
cur_date) = 3
```

Dacă există doi indecşi, după expresiile Charges şi MONTH (Incur-Date), se vor crea două hărţi de biţi prin parcurgerea celor doi indecşi. Dacă baza de date conţine opt înregistrări, vom putea vedea două hărţi de biţi memorate sub forma 10000010 şi 00100011. Cele două hărţi de biţi sînt comparate logic cu operatorul AND, obţinîndu-se 00000010. Bitul "1" are un offset de 7 biţi, deci FoxPro extrage a şaptea înregistrare. Dacă apoi executăm:

```
LOCATE FOR Charges >
100000 AND Age >= 18
```

Rushmore va căuta numai în indexul Age pentru a crea harta de biţi 11011100 pentru Age >=18 şi va face operaţia AND cu harta de biţi pentru Charges - care există deja, obţinînd valoarea 10000000 care desemnează prima înregistrare. Este foarte important de reţinut că această hartă de biţi este păstrată în ordinea fizică a înregistrărilor. Dacă dorim să vedem

rezultatele ordonate după un anumit index, trebuie să revenim la index. Accesul lui Rushmore ordonat după index poate fi semnificativ încetinit în funcţie de mărimea fişierului, mărimea memoriei disponibile pentru "cache" şi de gradul de fragmentare a datelor.

Totuşi, Rushmore foloseşte cîteva "scurtături" iscusite. El poate să preia inversa unei hărţi de biţi în loc să parcurgă din nou indexul. Dacă există o hartă de biţi care garantează că nu se vor găsi înregistrări, Rushmore nu se va mai deranja să creeeze celelalte hărţi de biţi. Un exemplu ar putea fi:

Interogarea 1:

```
COUNT FOR Age = 300
```

(0 înregistrări)

Interogarea 2:

```
COUNT FOR Age = 300 AND
(Sex = 'M' OR Diagnosis
= '650')
```

În acest caz nu se vor crea hărţi de biţi pentru Sex sau Diagnosis.

Impactul tehnologiei Rushmore asupra stilului

Abia am explicat cîteva chestiuni şi apar întrebările. Este nevoie să ne schimbăm stilul de programare? Tehnica Rushmore este la fel de utilă în aplicaţii cu mai mulţi utilzatori ca în interogări ad-hoc ale unui singur utilizator?

Între anumite limite, Rushmore permite să se scrie mai puţin riguros şi să se obţină, încă, performanţe bune.

Tehnologia este proiectată pentru a extrage seturi de articole. La celălalt pol, comanda FoxPro SEEK este optimizată pentru găsirea unui singur articol. Dacă avem nevoie să găsim un singur articol, un SEEK va fi mai rapid decît LOCATE FOR. Clauza FOR activează Rushmore, care va construi un set de hărţi de biţi în eventualitatea că dorim să vedem mai mult decît primul articol ce corespunde condiţiei. Aceasta cere timp. Posibilitatea de a observa diferenţa de viteză depinde de mărimea fişierului raportată la viteza maşinii.

SET ORDER poate pune Rushmore la grea încercare dacă fişierele sînt mari. Tehnologia Rushmore a fost proiectată să creeze hărţi de biţi neordonate şi să returneze răspunsurile rapid, dar SET ORDER forţează FoxPro-ul să revină la indecşi şi la baza de date pentru a determina ordinea mulţimii rezultat. (Notă: SET ORDER nu afectează operaţia COUNT sau operaţia SELECT din SQL).

S-au efectuat teste pe tabele mari (de la 300000 la 2 milioane articole), modificînd numărul de înregistrări din mulţimea rezultat. Pe măsură ce mulţimea rezultat devine mai mică, timpul de execuţie creşte - uneori chiar considerabil. Această tendinţă s-a manifestat atît pe sisteme cu memorie multă (24 Mb), cît şi pe sisteme cu memorie minimală (640 Kb).

Crearea din mers a unui index condiţional temporar pare să dureze mult prea mult ca să merite să fie făcută (Rushmore nu măreşte viteza lui INDEX FOR). În unele cazuri pare realizabilă crearea unei copii sortate a mulţimii rezultat cu comanda SELECT din SQL. Cealaltă opţiune ar fi să nu se utilizeaze Rushmore în aceste situaţii, ci să se folosească propriile tehnici anterioare de programare. Totuşi, ele ar putea să vi se pară lente după ce aţi văzut ce poate face Rushmore în interogări neordonate.

Un ultim avertisment pentru cei neatenţi: deschiderea oricărui fişier .IDX va seta implicit ordinea. Pentru a evita o mulţime de apeluri la asistenţa tehnică în legătură cu o "scamă" în viteza lui Rushmore încercaţi un `USE <numedbf> INDEX <numeidx> ORDER 0`.

O altă plîngere obişnuită survine cînd se încearcă utilizarea lui Rushmore în interogări ce nu sînt optimizabile. Fox defineşte expresia optimizabilă ca *<expresie cu index> <operator relaţional> <expresie cu constante>*. Un exemplu simplu ar fi:

```
Firstname >= 'Kevin'
```

unde există un fişier index - după cîmpul Firstname - deschis. Dacă avem, totuşi, numai un index după Lastname+Firstname, nu se poate optimiza expresia Firstname = 'Kevin'. Este necesar ca Rushmore să găsească un index care corespunde expresiei de interogare. În cazul expresiilor cu caractere, el compară valorile indecşilor caracter cu caracter de la sînga la dreapta. Viteza de căutare în index ar fi puternic redusă dacă ar trebui să caute fiecare octet al indexului pentru a vedea care chei au o anumită valoare "ascunsă" în interiorul lor.

Astfel se explică motivul pentru care nu se poate utiliza curent operatorul de căutare de subşiruri $. Totuşi, FoxPro ar putea să facă o căutare octet cu octet dacă ar folosi indecşi după o singură cheie. Aceasta ar reduce volumul de date ce trebuie parcurse într-un fişier de indecşi multipli sau într-o bază de date.

S-ar părea că nu mai este necesar să-i atenţionăm pe utilizatorii "înrăiţi" de FoxPro ca Rushmore nu va creşte ca prin farmec viteza desenărilor pe ecran, a manipulării tastaturii, a GET-urilor, a APPEND BLANK-urilor, sau a altor funcţii de acest gen, dar aureola ce însoţeşte tehnologia Rushmore pare să-i fi determinat pe unii să se aştepte la tot felul de însuşiri miraculoase. Nu vă amăgiţi: Rushmore este mai ales o metodă de căutare şi localizare de articole.

Pe ce dăm banii?

Înainte de a trece la interogări mai elaborate, să ne oprim puţin la COUNT. De obicei, demonstraţiile publice cu FoxPro 2.0 includ evidenţierea vitezei lui Rushmore executînd COUNT pe baze de date mari. Din nefericire, cei care dezvoltă programe nu utilizează COUNT prea mult. Iată cîteva su-

gestii în legătură cu modul în care se poate profita de avantajele celei mai rapide comenzi Rushmore.

Tabelele încrucişate de frecvenţă constituie o primă estimare foarte bună atunci cînd ne analizăm datele. Ele ne ajută să "simţim" numerele şi să localizăm rapid potenţiale nepotriviri în date. Comanda

```
SELCT    Diagnosis,
COUNT    (*)
FROM     Hospital
GROUP    BY    Diagnosis
```

ne poate ajuta să evităm ore de depanare ce ar putea surveni ulterior. Variaţiuni pe această temă sînt:

```
SELECT      Diagnosis
DISTINCT    FROM    Hospital
```

sau

```
SELECT       COUNT(DISTINCT
Diagnosis)
FROM     Hospital
```

Dacă aveţi o bază de date cu un inventar, poate fi mult mai rapid să faceţi un COUNT pentru toate înregistrările "vîndute" unui client decît un SUM pentru subtotalurile dintr-o bază de date cu comenzi. De fapt, toate celelalte comenzi care utilizează Rushmore impun citirea din baza de date la un anumit moment. Nu vă aşteptaţi ca ele să fie la fel de rapide ca şi COUNT, care - de obicei - citeşte numai indexul.

Unii programatori au sugerat că bătrîna strategie manuală SEEK / WHILE va fi mai rapidă decît o condiţie FOR optimizată ad hoc. Aceasta era valabil pentru versiunile anterioare ale programelor de test pentru FoxPro 2.0. Dar vremurile s-au schimbat. S-au testat diferite proceduri de extragere a datelor variind spaţiul de memorie, înregistrările extrase, gradul de dispersie a datelor şi modul de sortare al datelor de ieşire. Rezultatele pot fi văzute în tabelul 1.

În general, SEEK/WHILE este optim numai atunci cînd se ştie că mulţimea obţinută ca rezultat va fi mică. În toate celelalte cazuri Rushmore pare să fie la fel de bun, sau chiar mai bun. În unul din cazuri - un rezumat (AVERAGE, SUM, MAX) al unei submulţimi mari în condiţiile unui spaţiu de memorie redus - a fost cu un ordin de mărime mai rapid. Altfel, viteza a fost numai de două sau trei ori mai mare.

Dacă nu se pune problema performanţei, utilizarea construcţiei FOR de către Rushmore face ca programarea şi depanarea să fie mai simple decît pentru SEEK/ WHILE, care cere urmărirea stării curente a lui SET ORDER în cod. De asemenea, construcţia FOR nu are corespondent SEEK/WHILE atunci cînd există OR în condiţie (de exemplu nu există căi de a crea un index concatenat pentru a căuta Field1 = x OR Field2 = y).

Rushmore şi SQL vor face ca şi programarea relaţională să fie mai atrăgătoare. Sistemele complexe ce depind de date redundante şi totaluri de rulare pot fi înlocuite cu baze de date normalizate mai simple. SUM-urile executate "din zbor" vor fi, uzual, mult mai uşor de făcut.

Aceasta ne îndeamnă să studiem în continuare o altă noutate Fox: interogări care citesc numai fişierele index, dar nu şi bazele de date. În mod obişnuit, operaţiunile COUNT şi SELECT <fields> DISTINCT sînt singurele operaţiuni ce rulează cu viteze mici. O altă interogare uzuală ce ar putea beneficia de citirea exculsivă a fişierelor index este sumarea pe un singur cîmp. De exemplu:

SUM Charges For Charges >1000

cere numai citirea unui index al acelui cîmp, fără să se atingă de baza de date.

RQBE - SQL-ul lui FoxPro 2.0 şi Rushmore constituie o combinaţie excelentă pentru analiza datelor. Se pot crea şi executa interogrări atît de repede şi de simplu, încît ne putem permite să fim creativi. Acolo unde, poate, v-aţi decis să nu mai exploraţi noi zone deoarece ar fi durat prea mult, puteţi acum să simulaţi diverse scenarii. Deoarece FoxPro salvează rezultate interme-

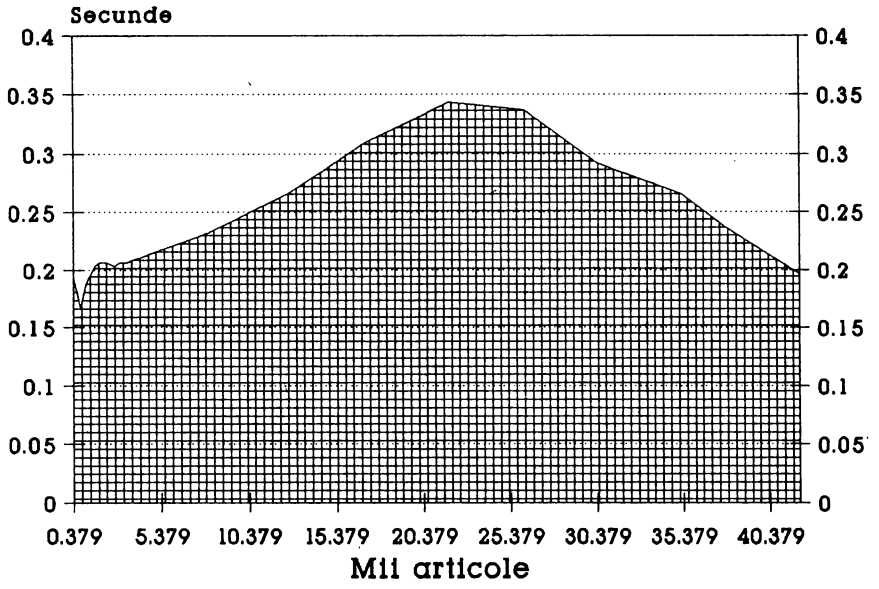


Figura 1. Operaţiunea lui FoxPro 2.0 de căutare în index pentru construirea unei hărţi de biţi este foarte eficientă. După găsirea a aproximativ jumătate din înregistrări, timpul de execuţie scade.

diare şi are un "cache" excelent, se pot rafina întrebările - pentru un număr dublu de interogări nu este nevoie de un interval dublu de timp.

Cîteva obiecţii

Dacă bazele de date sînt utilizate atît pentru interogări ad-hoc cît şi pentru tranzacţii, este posibil să nu se dorească indexarea fiecărui cîmp, deşi aceasta ar permite accelerarea interogărilor de către Rushmore. De fiecare dată cînd se modifică sau se adaugă un articol, FoxPro are de efectuat suplimentar actualizarea acelor indecşi. Aceasta este o decizie ce se ia în funcţie de situaţie.

Pentru un sistem de colectare de date, indecşii construiţi pe măsură ce se adaugă articole unul cîte unul pot deveni foarte fragmentaţi. O anume valoare cheie care nu este unică poate fi localizată într-o regiune puternic dispersată a unui fişier index. Valorile distribuite aleator în fişierul bază de date şi în fişierul index fac "viaţa grea" pentru hard disc.

Fragmentarea indexului este o chestiune controversată numai dacă este total ignorată. Timpul necesar pentru ca Rushmore să creeze o hartă de biţi creşte spectaculos dacă arborele index devine foarte fragmentat. Dacă se folosesc indecşi compuşi, problema se multiplică. Cel mai simplu leac este indexarea periodică a bazei de date.

Utilizarea intensă a lui Rushmore sau adăugările frecvente de pachete de articole pot face ca indexarea să devină nefolositoare. Dacă nu se poate face reindexarea suficient de des, se poate lua în considerare împărţirea indexului compus în mai multe fişiere compacte de index după o singură cheie. Astfel se pierde avantajul important al indecşilor structurali compuşi (noutate în FoxPro) deschişi automat împreună cu baza de date.

De asemenea, Rushmore nu este direct sensibil la SET DELETED ON. Deoarece starea DELETED este memorată în articol, ea este în esenţă un alt cîmp. Se poate crea un index după acel "cîmp" indexînd după funcţia DELETED(). În multe aplicaţii, o soluţie mai bună este să se pună blanc fără a mai fi necesar să se adauge NOT DELETED() în toate interogările.

Tabela 1

Rushmore contra SEEK / WHILE

Tabela 1.A Rushmore contra SEEK/WHILE pe date dispersate

	Memorie multă	Memorie minimală
Mulţime rezultat mică	Egalitate	SEEK/WHILE
Mulţime rezultat medie	RUSHMORE	Egalitate
Mulţime rezultat mare	RUSHMORE/Egalitate*	RUSMORE

Tabela 1.B Rushmore contra SEEK/WHILE pe date ordonate

	Memorie multă	Memorie minimală
Mulţime rezultat mică	SEEK/WHILE	SEEK/WHILE
Mulţime rezultat medie	Egalitate	Egalitate
Mulţime rezultat mare	Egalitate*	RUSMORE/Egalitate*

* Rushmore are avantajul de a extrage datele în ordine fizică

În general, utlizarea lui SEEK/WHILE este optimă numai atunci cînd se ştie că mulţimea rezultat va fi mică. În toatae celelalte teste Rusmore pare să fie mai bun sau egal.

Rushmore în Reţea

Lucrul cu date partajate într-o arhitectură file server - cum ar fi FoxPro LAN, a fost dintotdeauna o decepţie după ce aţi văzut ce performanţe atinge FoxPro cu ceva memorie pentru "cache". Problema fundamentală este cum poate utilizatorul de la o staţie de lucru să actualizeze o copie a articolelor sale care se află în memoria "cache" de pe o altă staţie. Pentru toate bazele de date file server de pe piaţă răspunsul este: nu poate. Pentru toate bazele de date care utilizează această arhitectură soluţia este să nu utilizeze "cache" pentru datele partajate. La prima vedere, s-ar părea că Rushmore trebuie să urmeze aceleaşi reguli, devenind ineficace în LAN.

Dar nu de data aceasta. Fox foloseşte "cache" pentru hărţile de biţi Rushmore şi indecşii folosiţi la construirea lor, şi, pur şi simplu, "simte" că un alt utilizator modifică un index. Dacă se dă ulterior o comandă care foloseşte o hartă de biţ, se parcurg din nou indecşii modificaţi. De exemplu, dacă o hartă de biţi reprezintă toate articolele care satisfac (Age > 10) şi cineva modifică valoarea unei anumite înregistrări din 4 în 5, harta de biţi nu se va schimba - şi totuşi se va face o nouă parcurgere. Fenomenul poate fi observat comparînd timpii de execuţie.

Această abordare ne poate determina să preferăm să avem mai multe fişiere index în lodul unui singur fişier cu indecşi multipli pentru

a obţine performanţe optime pentru Rushmore în reţea. De ce să permitem ca modificarea unei singure chei să forţeze o nouă parcurgere a numeroşilor indecşi memoraţi într-un fişier de indecşi multipli utilizaţi într-o interogare Rushmore ? Este mai bine să limităm noua parcurgere la o singură cheie prin păstrarea fiecărei chei în propriul ei fişier index.

Indecşi în general

Într-adevăr, studierea modului în care Rushmore realizează - în general - parcurgerea indecşilor conduce la concluzia că este mai bine să se folosească fişiere index individuale şi compacte.

Pentru a se testa modul în care Rushmore parcurge indecşii, s-a plasat un fişier index pe un disc proaspăt formatat şi s-a observat afişajul Smartvu care indică pistele la care se face acces. Apoi s-a testat comanda COUNT FOR pe diverse tipuri de indecşi pentru a vedea cît se parcurge din index şi în ce ordine.

În principiu, există trei tipuri de indecşi în FoxPro 2.0. Primul tip testat a fost un fişier .idx de tip FoxPro 1, nefragmentat. Comanda `COUNT FOR Fld1 = 'A'` a parcurs întregul index deşi nodurile asociate acestei valori erau situate la începutul fişierului. Căutarea a fost, în cea mai mare parte, secvenţială cu cîteva salturi în diferite secţiuni ale fişierului.

Al doilea test a avut în obiectiv un fişier .idx FoxPro 2.0 nefragmentat. Acest test nu a citit fiecare pistă şi a făcut acces la piste aproape secvenţial.

FoxPro 2.0 are implementate şi fişiere de indecşi multipli (.cdx). Un fişier .cdx poate conţine mai mulţi indecşi, numiţi etichete; etichetele .cdx sînt indecşi compactaţi. Acest ultim test a folosit o bază de date de coduri poştale cu un fişier .cdx indexînd toate cele trei cîmpuri: cod, oraş şi stat. COUNT FOR Cod = '3' a parcurs numai porţiunea care conţinea eticheta cod poştal a fişierului .cdx. Din nou, parcurgerea a fost - în majoritate - secvenţială, dar a sărit unele piste şi a revenit la ele mai tîrziu.

Cînd indecşii sînt fragmentaţi, procesul de căutare nu mai este secvenţial. În cazul fişierelor .cdx, FoxPro nu a mai căutat numai în prima parte a fişierului. Deoarece eticheta era dispersată de-a lungul întregului fişier, Rushmore a trebuit să facă salturi mari. Uneori a trebuit să sară înainte şi înapoi între piste aflate la mare distanţă una de alta. Din acest motiv - şi din altele, este bine să se utilizeze fişiere separate de indecşi compacţi.

Din motive de performanţă, Fox ar trebui să adauge facilitatea de deschidere automată a fişierelor index .cdx şi la indecşii compacţi separaţi. Într-un singur antet s-ar putea memora toate numele indecşilor compacţi separaţi care ar trebui deschişi automat.

Concurenţă

Rushmore are cîteva neajunsuri mai deranjante decît eşecul accidental în furnizarea performanţelor maxime. Dacă a fost construită o hartă de biţi Rushmore pentru o comandă ca BROWSE FOR, este posibil ca ea să nu mai fie evaluată dacă alţi utilizatori modifică datele care intră sub incidenţa criteriului de selecţie. O soluţie ar putea fi utilizarea unei comenzi `SHOW WINDOW <nume fereastra browse> REFRESH` pentru a actualiza fereastra BROWSE. Dar acest lucru nu se face automat. Din acest motiv unii vor prefera să folosească, pentru fişierele partajate, comanda BROWSE KEY care este mai puţin flexibilă (şi care nu utilizează Rushmore).

O problemă similară se manifestă şi pentru SCAN FOR / ENDSCAN. Mulţimea rezultat Rushmore se crează atunci cînd se întîlneşte primul SCAN FOR în program şi ea nu va mai fi modificată, indiferent de ce se petrece în interiorul buclei SCAN. În orice situaţie în care mulţimea rezultat obţinută cu Rushmore poate fi afectată în timpul executării comenzii, fie se va utiliza clauza NOOPTIMIZE în comandă pentru a dezactiva Rushmore, fie se vor utiliza metode tradiţionale - cum ar fi SEEK/WHILE - pentru a rezolva problema.

Concluzii

Operaţiunea de explorare a indexului în vederea construirii unei hărţi de biţi este extrem de eficientă. După cum se vede din Figura 1, după ce s-au găst circa jumătate din articole, timpul de execuţie scade. Aceasta sugerează că Fox determină, la începutul explorării arborelui index, care cale (condiţia căutată sau inversa ei) va conduce mai repede la răspuns. Mai mult, timpul necesar pentru a efectua un COUNT FOR pentru un număr dublu de articole nu este de două ori mai lung. Noile tipuri de indecşi scurtează acest timp mai mult decît vechile fişiere .idx.

Dacă nu vă lăsaţi ademeniţi să faceţi mai multe cu Rushmore, veţi putea observa un trafic mai redus în reţele. Şi totuşi, este de presupus că mulţi vor fi seduşi de performanţele unor proceduri pe care nu le-au mai încercat pînă acum şi vor mări traficul. Rămîne de văzut în ce măsură se poate implementa un FoxPro 2.0 pe un sistem mare cu mai mulţi utilizatori care să poată trata atît procesarea de tranzacţii, cît şi interogările ad hoc.

Utilizat corect, Rushmore poate rezolva probleme pe care puţini le-au crezut realizabile pe un PC. Mai trebuie încă să se lucreze la finisarea programelor în domeniul performanţelor de viteză, dar mult mai puţin ca înainte. Probabil că nu există un alt produs mai rapid pînă cînd ne hotărîm să dăm mult mai mulţi bani pe sisteme de gestiune a bazelor de date pe mini şi/sau mainframe.

(C. N.)

BAZE DE DATE DISTRIBUITE

Distribuirea datelor constituie o idee atrăgătoare din mai multe motive. Ea oferă posibilitatea de a exploata MIP-urile ieftine de pe PC-uri, staţii de lucru şi borduri UNIX, de a putea configura operaţiunile pe computer ale organizaţiilor care au birouri răspîndite în ţară. În pofida acestor avantaje, cel mai bun sfat care poate fi dat utilizatorilor de calculatoare care au în vedere o abordare distribuită a bazei de date este - de obicei - "Nu faceţi aşa ceva".

Totuşi, temeinicia acestui sfat depinde de ce se intenţionează să se facă. "Baza de date distribuite" are înţelesuri diferite pentru diferiţi oameni.

Pe de o parte, cei care vînd sisteme de gestiune a bazelor de date pot să pretindă în ofertă posibilităţi de distribuire pur şi simplu pentru că sînt permise citirea şi accesul pentru actualizare de la distanţă într-o reţea, în timp ce, pe de altă parte, teoreticienii insistă asupra faptului că adevărata distribuire este asigurată numai dacă aplicaţiile şi utilizatorii pot trata o bază de date ca un singur depozit logic de date, indiferent de gradul de împrăştiere în reţea.

Între aceste două poziţii, se găsesc diverse nivele de sotisficare, prezentînd opţiuni utile şi diverşi coeficienţi de risc.

Citiri şi actualizări la distanţă

Aproape toţi cei care comercializează sisteme de gestiune a bazelor de date oferă facilităţi de citire şi de acces pentru actualizare de la distanţă ca un prim pas către distribuirea bazelor de date. Această facilitate permite unui utilizator să manipuleze de la distanţă tabele într-o reţea, dar accesul poate fi limitat la o bază de date la un anumit moment de timp în cadrul aceleiaşi tranzacţii. Trebuie să remarcăm că există două moduri de gestionare a accesului la date într-o reţea. Primul mod constă în controlul local, cu fiecare acces fizic efectuat prin reţea. Acest mod este ineficient şi poate duce la timpi de răspuns inacceptabili, din cauza consumului de timp al reţelei care se adaugă celui necesar fiecărui acces fizic.

O altă soluţie este controlul accesului la baza de date către un proces "server" de pe maşina situată la distanţă, care procesează un acces logic - cum ar fi o interogare SQL, şi întoarce rezultatul prin reţea. Acest mod de operare este esenţial pentru bazele de date distribuite şi cele mai multe produse funcţionează astfel.

Accesul la distanţă face posibile configuraţiile "client-server", deşi aceste configuraţii nu implică în mod necesar distribuirea datelor. Avabtajul evident este că serverele de date pot fi ajustate pentru asigurarea unei deserviri rapide a datelor. Dacă un singur server de date se dovedeşte insuficient, se pot adăuga altele, şi astfel se face primul pas către baze de date distribuite.

Baze de date distribuite

Imediat de există mai mult de o bază de date, viaţa începe să devină mai complicată, chiar dacă ele se găsesc pe acelaşi calculator. Este foarte posibil ca, de exemplu, o tabelă cu clienţi să fie ţinută într-o bază de date şi o tabelă cu comenzi într-o alta. De aceea ar fi raţional să se unească cele două tabele pentru a lista clienţii şi comenzile pe care le-au depus. Surprinzător, nu toate produsele de baze de date suportă aceasta, deşi majoritatea o permit inclusiv produsele cele mai utilizate ca Ingres, Sybase şi Oracle.

Totuşi, dacă se amestecă diferite produse de baze de date, este din ce în ce mai puţin posibil să se poată realiza astfel de "uniri". În această situaţie devenim dependenţi de 4GL pentru a putea trata problema via "porţile" pe care le oferă către diverse baze de date. În orice caz, dacă intenţionaţi să implementaţi o bază de date distribuită, este bine să nu amestecaţi produsele de baze de date deoarece fiecare oferă posibilităţi diferite şi concilierea lor poate deveni un coşmar.

Prima regulă a distribuirii datelor este că datele se vor distribui numai pentru motive de viteză. Este mult mai bine să existe o singură bază de date care serveşte toate aplicaţiile rulate de o organizaţie. Totuşi, aceasta nu este totdeanua practic, deoarece chiar şi maşinile foarte puternice, cum ar fi Model 204 de la Computer Corp of America, sînt limitate ca volum de date şi număr de tranzacţii pe care le pot manipula. De asemenea, mai există şi o limită a puterii calculatorului.

Primul pas este, de obicei, să se divizeze resursa masivă de date în cîteva baze de date astfel încît nici o aplicaţie să nu aibă nevoie să apeleze mai mult de o bază de date, cu excepţia unor accesări ocazionale de citire. Aceasta este strategia utilizată de multe amplasări mari şi ea produce rareori probleme.

Cu toate acestea, unele instalări au nevoie să distribuie datele într-un mod mai sofisticat. De exemplu, o companie de distribuire poate avea cîteva depozite în locuri diferite, fiecare fiind servit de un computer local. Într-un asemenea sistem, tranzacţiile cu mişcările stocurilor pot cuprinde mai multe calculatoare. În acest caz, poate să existe o necesitate autentică pentru o bază de date distribuită.

Idealul pentru bazele de date distribuite ese să se lucreze logic cu o singură bază de date, eventual răspîndită peste mai multe noduri ale reţelei. Implementarea va consta în cîteva baze de date fizice, dar din punctul de vedere al utilizatorilor şi al aplicaţiilor există o singură bază de date. Atingerea acestui ideal nu este deloc simplă, iar interogările şi actualizările pot ridica probleme semnificative.

Interogări distribuite

Atunci cînd există două baze de date în noduri diferite ale unei reţele şi cînd o interogare uneşte tabele din fiecare din ele, softul de gestionare trebuie să decidă unde să "gospodărească" tabela rezultat. Unirea a două tabele va genera o tabelă virtuală, şi întrebarea care se pune este unde să se depună această tabelă virtuală. Este posibil, de exemplu, să se selecteze două articole dintr-o tabelă şi cinci mii din cealaltă, astfel încît prin reţea vor călători fie două articole, fie cinci mii. În mod evident, este mai bine să se transmită numai două, dar decizia trebuie să o ia softul de gestiune.

Aici se pune problema unei optimizări distribuite şi foarte puţine produse oferă vreo sofisticare în această zonă. Cele cîteva care o fac includ Ingres, Informix şi NonStop SQL al lui Tandem, dar chiar utilizarea unuia din aceste produse nu rezolvă în mod necesar problema. Dacă există mai multe baze de date şi mai multe noduri implicate, problema se compune. Dacă bazele de date sînt distribuite fără a ţine cont de interogările ce ar putea fi executate, atunci este foarte posibil ca performanţă să devină inacceptabilă indiferent de flexebilitatea optimizării interogărilor. O posibilitate ar fi duplicarea unor baze de date sau a unor submulţimi ale bazelor de date în diferite noduri ale reţelei, astfel încît interogările să fie nevoite să cuprindă mai multe noduri cît mai rar. Atît timp cît softul de control ştie de existenţa acestor "server-e de interogare", problema se reduce.

Totuşi, duplicarea datelor trebuie planificată adecvat, astfel încît datele duplicate să poată fi actualizate ori de cîte ori este necesar. În funcţie de sistem, este necesar ca datele duplicate să fie la zi, sau poate fi acceptabil ca ele să fie cu o zi "în urmă". În ambele cazuri duplicarea datelor trebuie gestionată adecvat şi, în mod evident, este de dorit să existe posibilitatea duplicării automate. Din nou, există numai cîteva produse care asigură această necesitate, printre care Adabas, Datacom:DB, Rdb şi NonStop SQL.

Posibilităţile de duplicare ar trebui să includă planificarea momentului efectuării duplicării şi capacitatea de a transmite numai datele modificate pentru a reduce la minimum traficul în reţea. Pe lîngă utilitatea în realizarea interogărilor, utilizarea acestei metode mai prezintă şi alte avantaje. De exemplu, într-un sistem de vînzare cu amănuntul ar putea folosi la transmiterea săptămînală sau zilnică a listelor de preţuri curente. De asemenea, ar putea fi util la extragerea de date pentru spreadshreet-uri, programe de grafică, şi altele.

Actualizări distribuite

Dacă, dintr-un motiv sau altul, o interogare distribuită eşuează, aceasta nu va afecta integritatea datelor. Dar dacă eşuează o actualizare distribuită, afirmaţia anterioară nu mai este adevărată. Pentru a asigura integritatea datelor în cadrul actualizărilor distribuite, este necesar să se asigure executarea în două faze. Acest proces se asigură că toate bazele de date implicate într-o tranzacţie sînt gata să recepţioneze datele, efectuează tranzacţia, apoi verifică dacă actualizarea s-a efectuat corect în toate bazele de date implicate în proces. Dacă, dintr-un motiv oarecare, actualizarea a eşuat în unul dintre noduri, se va şterge tranzacţia, revenind la situaţia anterioară. Există puţine produse care oferă o execuţie în două faze, printre care Datacom: DB, Rdb, Ingres, Empress, Sybase, iar mulţi alţi producători sînt hotărîţi să ofere şi ei această facilitate.

Totuşi, faptul că este disponibilă o astfel de posibilitate, nu ar trebui să fie o încurajare în implementarea de tranzacţii distribuite. Sistemele care se întind peste mai multe noduri sînt vulnerabile la căderea oricăreia dintre ele şi sînt, deci, mai puţin sigure. De asemenea, tranzacţiile care implică mai mult de un nod al unei reţele sînt mai lente decît cele care accesează date locale şi există posibilitatea - cea mai rea dintre toate - de apariţie a unor blocaje globale.

Un blocaj global este un conflict de blocare ("zăvorîre") a înregistrărilor care cuprinde mai mult de un nod al reţelei. Aceasta este o situaţie asemănătoare unei îmbrăţişări mortale, în care două tranzacţii nu se pot termina deoarece s-au "zăvorît" una pe alta. Fenomenul poate avea un efect vast, paralizînd cîteva din nodurile reţelei. Din aceste motive, tranzacţiile distribuite ar trebui să fie folosite numai acolo unde nu există o altă opţiune satisfăcătoare.

Transparenţă

Din ceea ce am spus pînă acum, s-ar părea că implementarea unei baze de date distribuite este realizabilă dacă se dispune de optimizare distribuită, duplicare automată şi execuţie în două faze a actualizărilor. Acest lucru este aproape adevărat, dar există o altă caracteristică care este utilă - alţii ar spune chiar imperios necesară - transparenţa.

Transparenţă înseamnă că aplicaţia nu este nevoită să cunoască locul unde sînt plasate fizic datele. Transparenţa completă a localizării datelor implică un S.G.B.D. capabil să gestioneze traseele actualizărilor în cadrul reţelei şi capabil să recupereze totul în urma eşecului în oricare din noduri. Nu există pe piaţă produse capabile să satisfacă aceste cerinţe, în special din cauza condiţiilor de comportare în caz de eşec.

Oricum, structura reţelei şi amplasarea fizică a bazelor de date pot fi definite uşor, iar cererile de date pot fi "cartografiate" la definirea lor, ceea ce asigură un grad rezonabil de transparenţă a amplasării. Această posibilitate este prezentată la cîteva produse din care amintim Ingres, Sybase, Informix, Unify şi DB2.

Avantajul transparenţei amplasării constă în posibilitatea de a reorganiza datele în reţea sau de a modifica chiar reţeaua fără a fi nevoie să se adapteze aplicaţia. Probabil că astfel de manevre nu se fac frecvent, dar dacă ori de cîte ori este nevoie de astfel de modificări se ajunge la schimbări în codul programului, devin evidente virtuţile unui anumit nivel de transparenţă.

Aspectul - probabil - cel mai curios al bazelor de date distribuite în constituie competiţia vînzătorilor de a le oferi. Este ca şi cînd ar exista mii de organizaţii care îşi doresc cu disperare să îşi distribuie datele şi care de-abia aşteaptă apariţia produsului potrivit. Şi totuşi, nu este cazul.

Cele mai multe firme care trec dincolo de o simplă configuraţie client-server, către o bază de date complet distribuită, sfîrşesc prin a regreta acest paas şi a reveni în scurt timp. De aceea o recomandăm rareori. Este mai uşor să nu înveţi pe calea cea mai grea.

N. C.

Cele douăsprezece porunci pentru baze de date distibuite

Cele douăsprezece "porunci" pe care trebuie să le îndeplinească o bază de date distribuită transparent - enunţate de C. J. Date - amintesc de versurile celebrului cîntec "Crazy He Calls Me" al lui Billy Holiday:

Ce-i greu o să fac chiar acum;
Ce-i imposibil o să dureze puţin mai mult.

Vă prezentăm în continuare enunţul celor douăsprezece legi aşa cum au fost ele enunţate de Date. Care este părerea dumneavoastră ?

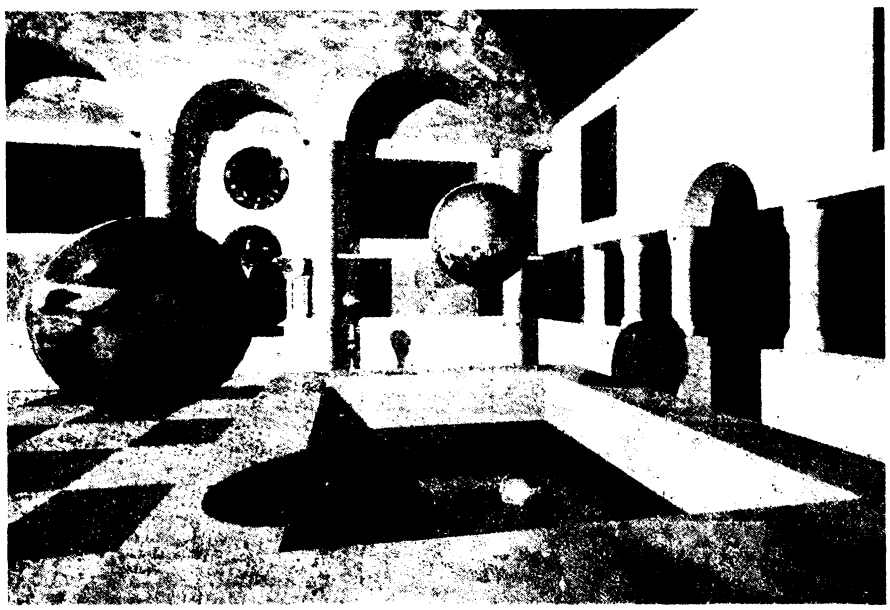
- Regula 1. Autonomia locală: Datele locale sînt deţinute şi gestionate local, inclusiv responsabilitatea şi securitatea: nici un post nu depinde de altele pentru a funcţiona.
- Regula 2. Toate posturile sînt egale: Nici un post nu se bazează pe un post central.
- Regula 3: Funcţionare continuă: O oprire planificată nu trebuie să fie niciodată necesară, deci instalările efectuate la un post nu afectează funcţionarea celorlalte; bazele de date pot fi create sau distruse fără a opri vreo componentă.
- Regula 4. Transparenţa amplasării: Utilizatorii nu au nevoie să ştie unde sînt amplasate datele pentru a le extrage.
- Regula 5. Transparenţa fragmentării: Relaţiile între elementele datelor pot fi fragmentate pentru stocare, dar acest lucru este transparent pentru utilizator.
- Regula 6. Transparenţa duplicării: Relaţiile şi fragmentele sînt reprezentate fizic prin copii multiple, stocate separat, dar transparent pentru utilizator.
- Regula 7. Procesarea interogărilor distribuite: Activitatea de intrare/ieşire se poate desfăşura la mai multe posturi, permiţînd optimizarea interogărilor atît local cît şi global.
- Regula 8. Actualizări distibuite: Tranzacţiile singulare pot executa codul la mai multe posturi.
- Regula 9. Independenţa hardware: Toate maşinile participă ca parteneri egali.
- Regula 10. Independenţa faţă de sistemul de operare: Sînt suportate mai multe sisteme de operare.
- Regula 11. Independenţa faţă de reţea: Sînt suportate mai multe reţele.
- Regula 12. Independenţa faţă de bazele de date: Se asigură suportul pentru diverse baze de date cu interfeţe de acelaşi fel.

O privire asupra graficii animate

O privire rapidă şi atentă asupra spoturilor publicitare şi videoclipurilor actuale ne arată că animaţie pe calculator şi-a făcut deja intrarea în afacerile zilei. Oricît de uşor ar părea rezultatul final, construcţia fiecărei imagini în parte într-un sistem grafic necesită mult timp, este foarte complicată, şi de aceea foarte scumpă.

Odată cu interfaţa Renderman se naşte un nou standard. Articolul ce urmează oferă o privire de ansamblu asupra tehnicilor fundamentale şi tendinţelor actuale din domeniu.

Apariţia unei noi tehnici pentru grafica computerizată, Rendering-ul, va influenţa designul viitoarelor calculatoare, sisteme economice şi aplicaţii. Rendering-ul constă în diverse tehnologii: programe de tip C.A.D. şi efectele speciale din industria de film. Rendering-ul este cu adevărat pretenţios: chiar şi facilităţile de excepţie ale staţiilor de lucru cu unităţi centrale rapide; cu procesoare de virgulă mobilă, cu accelatoare de grafică şi alte extensii par a fi atunci cînd pe ele se rulează un program de Rendering, extrem de lente. Programele actuale de Redering creează imagini grafice deosebite, în timp ce software-ul mai are un drum lung de parcurs.



Aplicaţiile grafice de astăzi vor fi folosite în programe de CAD a căror structură de bază va fi constituită din imagini. Programele de Rendering fac posibilă utilizatorului controlarea amănunţită a compoziţiei unei imagini: umbre, saturarea culorilor, efecte de lumină, culori, reflexii şi alte elemente.

Rendering-ul se diferenţiază de "modelarea solidului", dar ambele metode sînt strîns legate între ele şi se dezvoltă în paralel. Scopul original al CAD-ului era desprinderea procesului de calcul de procesul de construcţie al unei clădiri sau de finisare industrială a unui obiect. De-a lungul timpului a ieţit în evidenţă că introducerea formaţiilor complete, tridimensionale despre un obiect măreşte flexibilitatea algoritmilor de calcul şi astfel face posibilă folosirea calculatorului pentru a vedea cum obiectele interacţionează între ele.

Pentru a construi o imagine realistă calculatoruul are nevoie de informaţii foarte detaliate asupra proprietăţilor suprafeţelor exterioare ale obiectelor şi de o descriere suficientă a luminilor şi a informaţiilor de de fundal, informaţii de care "modelarea solidă" nu ţine cont.

Cererea din ce în ce mai mare de fidelitate a imaginii va conduce la folosirea calculatoarelor în artă şi în industria de film. Diferenţa faţă de "modelarea solidă" este că obiectele sînt înconjurate de lumini, umbre şi alte elemente de compoziţie. Prin colaborarea cu oamenii de ştiinţă ce studiază grafica pe calculator, constructorii au făcut deja paşi importanţi în crearea unor imagini cît mai naturale. Programele CAD posedă tehnologia fundamentală de care au nevoie cercetătorii şi ilustratorii, dar le lipseşte facilitatea de a poziţiona şi dimensiona corect obiectele.

Imagini fotorealiste

Există o serie de motive care fac Redering-ul interesant. Rendering-ul este pentru grafica pe calculator ceea ce a fost suprarealismul pentru arta abstractă. Tehnologia a ajuns la un punct de convergenţă; oamenii de ştiinţă şi artiştii lucrează împreună la experimente ce se pot concentra în mai multe domenii pentru a construi algoritmi grafici puternici şi eficienţi: ca răspuns la cererea anterioară de sisteme de tip Rendering, industria de calculatoare a început să construiască sisteme asemănătoare cu sistemele Mainframe şi introducerea puterii grafice în server-e şi

a le da în mîinile utilizatorilor evoluaţi. Luate împreună algoritmii, hardware-ul şi presiunea pieţei de CAD au dus la dezvoltarea explozivă din domeniul Renderingului. Prin Rendering o imagine devine într-un mod simplu o imagine plină de conţinut. Principala muncă în domeniul "modelării solide" constă în dezvoltarea modelelor simple, schematice, şi apoi ataşarea oricărui obiect component a unei culori unice. Rendering-ul transformă imaginea CAD în imagini grafice fotorealiste. Utilizatorii programelor de Rendering au în principal două probleme: hardware-ul şi modul de descriere al graficii. Renderingul funcţionează cel mai bine cu un hardware de 24 biţi la care, pentru obţinerea celor trei culori fundamentale, roşu, verde, albastru, se rezervă cîte 8 biţi (la o reprezentare color pe 32 biţi am mai avea disponibili încă 8 biţi pentru informaţii de control). Pentru rulări simple sînt necesari cel puţin 4 MB de memorie iar pentru rulări complexe necesarul devine de 8 MB. Unităţile centrale rapide (cel puţin 25 MHz) şi coprocesoarele grafice ajută la accelerarea procesului.

Dar chiar şi pe cele mai puternice sisteme timpul rămîne un factor restrictiv. E nevoie de cîteva minute pentru un obiect ce ocupă doar a opta parte din ecran. Anexarea poligoanelor sau întrebuinţarea Anti-Aliasing-ului măreşte timpul necesar construcţiei unei imagini complete. Poate dura 20 sau 30 min construcţia unei imagini care este "zidită" de la bază şi pînă în momentul în care utilizatorul poate determina dacă lumina sau culoarea sînt bine plasate. Modificări minimale în compoziţia unei imagini pot duce la modificarea proporţiilor, ceea ce măreşte tentaţia şi necesitatea de a experimenta din nou. Timpul necesar construirii unei imagini poate îngrădi foarte mult şi chiar îndepărta utilizatorul. Doar cei ce folosesc sisteme foarte puternice au posibilitatea reală de a experimenta. Toţi ceilalţi pot face acest lucru doar în cadrul unor sesiuni limitate în timp.

Principalele programe CAD pot da imaginilor prin această metodă un efect tridimensional, prin aceea că suprafeţelor mari li se pot ataşa texturi diferite sau culori diferite. Prin Rendering suprafeţele exterioare sînt împărţite în părţi foarte mici, ce pot fi umbrite individual.

Deşi poate fi important ca un singur obiect, de exemplu un automobil, să poată fi considerat ca fiind compus din mai multe obiecte luate împreună, la fel ca şi în cazul scenelor interesante (un automobil rulînd pe o stradă, un elefant fugind printr-un ţinut de porţelan) reflexele de lumină ale obiectelor singulare în astfel de scene sînt complicate şi foarte greu de construit corect. Transpunerea unui scenariu tridimensional întrr-o imagine bidimensională ridică şi alte probleme de genul:

- care părţi ale obiectelor se văd în imaginea construită?
- ce culoare au părţile vizibile ale fiecărui obiect?

La rezolvarea acestor probleme se folosesc două tehnici principale: tehnica Z-buffering şi tehnica Raytracing. Metoda ceea mai simplă de a lucra cu probleme de genul "ce se va vedea?" este metoda Z-buffering cunoscută şi ca metoda "scan-line". În tehnica Z-buffer obiectele sînt prelucrate unul după celălalt. Pentru fiecare pixel separat algoritmul Z-buffer "priveşte" înapoi în scenariu pentru a vedea dacă obiectul studiat poate fi văzut din acest punct. Dacă poate fi văzut apare "verificarea Z-buffer". Z-buffer-ul este de fapt o zonă de memorie în care pentru fiecare pixel al construcţiei se păstrează imaginea următoare pînă cînd obiectul este complet construit. Cînd obiectul curent se află în apropiere se ia în considerare că toate imaginile construite anterior vor fi umbrite de acesta. Umbrirea este de fapt procesul de alegere a culorii pe care o va avea punctul dat. Există trei metode principale de umbrire care se folosesc împreună cu Z-buffering-ul:

- umbrirea Lambert - cea mai simplă şi mai rapidă tehnică. Obiectul care va trebui umbrit este împărţit în poligoane şi fiecărui poligon i se atribuie o culoare în funcţie de proporţiile lui şi de poziţia faţă de sursa de lumină. De aici rezultă o reprezentare cu faţete:
- umbrirea Gourand - lucrează tot cu poligoane. Culoarea va fi împrăştiată pe toată suprafaţa unui poligon, pornind din colţuri. Procesorul de mediere a culorii îmbunătăţeşte rezultatul dar prelungeşte semnificativ timpul de lucru.
- umbrirea Phong - se calculează lungimea suprafeţei exterioare a obiectului tridimensional relativă la poligon şi se umbreşte cu această valoare. Cu această metodă se obţin rezultate şi mai bune decît cu metoda Gourand dar se înrăutăţeşte şi mai tare timpul de răspuns.

Umbrirea costă timp

Fiecare tehnică are slăbiciunile ei. Umbrirea Lambert se vede întotdeauna că e generată pe calculator. La umbrirea Gourand acest lucru nu mai e atît de vizibil dar se mai pot încă recunoaşte urmele unora dintre poligoane, mai ales în zonele cu un grad mare de curbare. Chiar şi la umbrirea Phong se mai cunosc marginile poligoanelor dar acest lucru poate fi evitat prin specificarea unui număr mai mare de poligoane în perimetrele curbate puternic, ceea ce va duce la o construcţie de lungă durată. Z-buffering-ul este o tehnică de forţă brută care conduce la imagini artificiale. Cînd fiecare obiect este generat separat umbrele, reflexiile unui obiect asupra celorlalte transparenţa,

răămîn nefolosite. Prin urmare Anti-Aliasiny-ul este greu de obţinut. Această artificialitate a reprezentării ştiinţifice nu reprezintă un impediment pentru că foarte des este nevoie de descrierea împreună a unor obiecte separate şi interacţiunea acestora şi mai puţin de obţinerea unei imagini realiste.

Pe de altă parte Z-buffering-ul oferă facilitatea de a avea un control puternic asupra diverselor efecte Z-buffering-ul este astăzi sprijinit în diverse moduri direct de hardware. Un Z-buffer este în esenţă o zonă de memorie rapidă analog cu un Frame-buffer, conceput special pentru asta, pentru a memora informaţiile intermediare în timpul construcţiei unei imagini. Z-buffer-ul îmbunătăţeşte Rendering-ul în mod esenţial la preţuri acceptabile.

Ray-tracing-ul nu este un leac universal

Ray-tracing-ul este un algoritm complicat şi conduce la cele mai bune rezultate. El este unul dintre cele mai vechi şi cele mai bune metode de rendering. Obiectele pentru Ray-tracing-ul unui scenariu trebuie să fie toate încărcate în memorie. După aceasta o rază va fi trasată dintr-un punct fix înspre ecran. Dacă raza întîlneşte un obiect sînt memorate informaţii despre iluminarea în punctul de întîlnire şi se construieşte un arbore de raze. Raza se reflectă apoi din obiect şi întîlneşte apoi un alt obiect. Cînd raza părăseşte scena sau după un anumit număr de reflectări se porneşte o nouă rază dintr-un alt pixel. Acest proces continuă pînă cînd toţi pixelii din ecran au fost adresaţi şi arborele este complet. Arborele va fi apoi prelucrat pînă cînd pentru fiecare punct al ecranului va fi calculată o anumită umbrire. Ray-tracing-ul poate lucra şi cu lumină foarte puternic reflectată, umbrirea se face corect şi poate fi continuată pînă cînd se obţine Anti-Aliasing-ul. Cu toate acestea interacţiunea umbrelor cu obiectele transparente poate fi tratată fals şi interacţiuni de lumină subtile de exemplu, în condiţiile de iluminare în spaţii închise, pot să nu fie mulţumitoare. Ray-tracing-ul permite rezultate de bună calitate dar este de zece ori mai încet decît tehnicile de scan-line. El se încetineşte exponenţial cînd numărul obiectelor din scenariu se măreşte şi asta cu toate că această tehnică pare a fi cea mai puternică. Nici una din tehnicile de Rendering descrise pînă acum nu tratează prea bine lumina slabă. Cînd vă puteţi găsi în interiorul unei clădiri priviţi odată înspre un colţ. Suprafeţele luminoase şi întunecoase dau nenumărate reflexii ale luminii între suprafeţele exterioare, care sînt foarte greu de imitat. O tehnică de preprocesoare, care prelucrează aceste reflexe luminoase slabe, se numeşte "radiozitate". Radiozitatea împarte obiectele nu numai în poligoane cărora le ţine culoarea, dar ţine cont şi de efectele de lumină care se vor reflecta de la un poligon la altul. Prin aceasta se vor putea prelucra efecte de culoare fină cum ar fi reflexia roz care vine de la un perete roşu. Cu Radiozitatea vor fi calculate doar efectele pe care le au obiectele singulare asupra altor obiecte. După aceea vor fi folosite acele tehnici de rendering pentru a obţine imaginea completă. Deşi se obţine rezultatul final ca la un Rendering, tehnica radiozităţii are nevoie de o mulţime de timp. Radiozitatea este foarte importantă pentru obţinerea unor imagini de înaltă calitate astfel încît producătorii de staţii de lucru fac eforturi mari pentru a sprijini această metodă.

Chiar şi cu cele mai bune tehnici şi cele mai economicoase, Rendering-ul nu e suficient pentru ochiul uman. Ochiul uman este foarte sensibil la lumini fine, uneori contează o singură dungă întunecoasă sau luminoasă. Rezolvarea acestei probleme constă în introducerea mai multor poligoane. Aceasta ar putea ajuta sau nu în funcţie de de detaliile fine ale luminii şi de poziţionare. Un Rendering efectiv necesită o grămadă de cunoştinţe şi de prelucrări ale luminii, şi mult timp liber. Producerea unei imagini realiste generată pe calculator este determinată de trei factori: calitatea obiectului geometric de bază dintr-un pachet CAD, calitatea suprafeţelor exterioare care au fost alese pentru fiecare obiect, şi de exactitatea fizică a Rendering-ului. De aceea, fotorealismul va fi mereu îmbunătăţibil şi va fi în puterea acestor factori să se obţină rezultate din ce în ce mai bune.

"Imaginile atrăgătoare" sînt o răspaltă meritorie, dar dacă se află şi în mişcare, ele devin minunate. Luaţi de exemplu, mai multe imagini care diferă între ele foarte puţin într-o secvenţă oarecare şi veţi obţine o animaţie. O asemenea animaţie nu poate fi generată în timp real - amintiţi-vă că poate dura pînă la 30 min. producerea unei singure imagini pe un calculator de birou cu performanţe slabe. Pe de altă parte putem genera animaţia construind imaginile una după alta şi memorîndu-le pentru a ale afişa ulterior.

Este necesară răbdarea

Animaţia şi Rendering-ul au dus spoturile publicitare şi industria filmului la cea mai înaltă calitate. Aceste filme au depăşit domeniul experimentalului şi au intrat în domeniul ştiinţei, în aplicaţiile militare sau în alte domenii importante.

Cu toate că animaţia prin Rendering consumă foarte mult timp există totuşi cîteva trucuri cu care se poate micşora numărul de ore necesar. De exemplu, animarea doar a unor mici părţi dintr-o scenă micşorînd astfel zona ce va trebui reconstituită. Calculatorul permite astfel de trucuri, ca de exemplu "intermedierea" metodă în care animatorul generează figura de început şi de sfîrşit iar calculatorul generează poziţiile intermediare. Dar "intermedierea" generează ex-

clusiv metodele schematice şi rendering-ul este cel care generează mai departe umplerea fiecărei imagini.

O altă problemă a animaţiei este spaţiul de memorie. Un ecran întreg de 640x480 pixeli şi cu o culoare reprezentată pe 8 biţi are nevoie de aproximativ 300KB. La 30 de imagini (cadre) pe secundă avem nevoie pentru un minut de animaţia de 540MB de memorie şi aceasta este mai mult decît pot avea majoritatea, utilizatorilor. Deci, utilizatorii ce vor să folosească animaţia şi Rendering-ul trebuie să genereze mai întîi cîteva cadre, să l;e memoreze, apoi să genereze următoarele cadre. Toate acestea trebuie salvate pe bandă. Din acest motiv filmul nu mai este la fel de uşor de editat şi pe diskett-ă.

Firma Pixar a început ca un compartiment al firmei Lucas Film, avînd sarcina să creeze efecte speciale pentru filme cu ajutorul grafiicii pe calculator. Acum cîţiva ani firma a fost cumpărată de Steve Jobs, ca întemeietorul firmei Apple şi întemeietorul NeXT. Pixar lucrează independent faţă de NeXT în unele domenii chiar cu rezultate mai bune. Pînă nu demult Pixar a fost cunoscută prin efectele speciale din film, Recent, filmul său de scurt metraj "Tin Toy" a cîştigat premiul Oscar. Pixar a mai produs cunoscutul efect de geneză pentru filmul "Star Trek II": "mînia lui Khan". În acest timp Pixar a descoperit cel puţin două lucruri importante: un extraordinar algoritm de Rendering şi o interfaţă cu ajutorul căreia alte programe pot comunica cu algoritmul. Algoritmul este cunoscut sub numele de "REYES" prescurtare de la "Renders Everythink You Ever Saw". Interfaţa se numeşte "Interfaţa Renderman". REYES a fost dezvoltată de Loren Carpeuter cu ajutorul unor specialişti de la SGI. Algoritmul atinge calităţile Ray-tracing-ului. Pixar susţine că viteza de generare a lui REYES la imaginile cele mai simple este mai slabă decît a Ray-tracing-ului aproape la fel de rapidă pentru imagini mai complicate, cu multe obiecte. Astfel spus Renderman este o interfaţă între un sistem de Rendering, ca de exemplu REYES. Ea este independentă de algoritm, hardware, sau viteze de execuţie.

Tot ce e necesar pentru generarea unei imagini cade în sarcina comenzilor RIB (Renderman Interface Bifte stream). Restul este făcut de sisetemul Rendering. Există astăzi Kit-uri de produse de dezvoltare pentru PC, cu Macintosh şi pe alte staţii de lucru. Producătorii de soft au cumpărat licenţele de la Pixxar, şi folosesc interfaţa Renderman.

Pe această cale Pixar a devenit standard universal pentru Rendering. Pachetele software care conţin în ele metoda Rendering oferă împreună cu ele şi interfaţa Renderman. Pentru PC interfaţa a fost folosită deja de "Cadkey", FastCAD şi mai tîrziu de AutoCAD. Pentru Macintosh este utilizat de Statavision 3D şi de swivel 3D.NeXT a promis că interfaţa Renderman va fi disponibilă şi în noul său sistem color. Formatul RIB a devenit noul standard de comunicare date, date care trebuie eventual folosite la Rendering aşa cum EPS este un standard pentru date date ce trebuie eventual tipărite la imprimantă. Astfel s-a născut o metodă simplă de comunicaţie între PC, Macintosh şi marii giganţi precum Cray. Transmiterea rapidă în reţea a informaţiei de imagine a devenit foarte uşor de practicat.

Stadiul actual al Rendering-ului aminteşte de stadiul fotografiei la începutul acestui secol. Experţii au acces la această tehnologie şi în curînd, şi mare masă de oamnei. Imaginile în mişcare sînt posibile azi doar pe calculatoarele cele mai performante, dar mîine vor fi accesibile pentru toţi utilizatorii. Ne bucurăm deja cînd hardware-ul are putere din ce în ce mai mare şi odată cu acesta mult şi posibilităţile Rendering-ului.

Programare VGA - Animaţie

În primul articol despre programare VGA în 256 culori am arătat cum se obţine o rezoluţie de 320 x 240 pixeli. Am amintit şi avantajele pe care le prezintă acest mod faţă de celelalte moduri. În acest articol vom prezenta tehnica de paginare a memoriei ecran, precum şi un program de animaţie care va folosi această tehnică.

Ştim că în modul 320x240x256 memoria ecran este organizată pe patru plane, fiecare plan de memorie putînd fi referit ca o matrice de 240 linii şi 80 coloane. Rezultă că o imagine de 320x240 pixeli ocupă în total 76800 octeţi, cîte 19200 octeţi în fiecare plan de memorie. Dar capacitatea unui plan de memorie este de 65536 octeţi, deci această imagine ocupă mai puţin de o treime din memoria ecran.

```
Port[$3D4]:=12;
Port[$3D5]:=Hi(Adr);
Port[$3D4]:=13;
Port[$3D5]:=Lo(Adr);
```

Este important de reţinut faptul că această adresă este aceeaşi pentru toate cele patru plane de memorie. Iniţial (adică după setarea modului grafic), se afişează conţinutul zonei care se află la începutul memoriei ecran.

Iată un exemplu de paginare pentru modul 320 x 240 x 256:

- pagina 0: începe la adresa relativă 0;
- pagina 1: începe la adresa relativă 19200;
- pagina 2: începe la adresa relativă 38400;

Rămîn în această situaţie 7936 octeţi nefolosiţi pentru fiecare din cele patru plane de memorie. Ajunşi în acest punct observăm că procedurile scrise pînă acum nu ne mai sînt de folos, confirmînd parcă o concluzie extrasă dintr-un tratat de murphologie: "Dacă constructorii ar fi construit clădiri aşa cum scriu programatorii programe, atunci prima ciocănitoare care ar fi venit ar fi distrus civilizaţia". Să mai facem încă o observaţie: procedurile XWritePixel şi XReadPixel sînt apelate foarte des, şi de aceea ar fi de dorit ca ele să fie scrise în limabj de asamblare, pentru ca viteza de desenare a imaginii să fie cît mai mare.

Iată de ce în acest articol reluăm în întregime unit-ul Xgrafica, îmbogăţit cu două proceduri noi: XSetPActive şi XSetPVideo (ele corespund funcţional procedurilor SetActivePage şi SetVideoPage din unit-ul Graph). De asemenea, procedurile XWritePixel şi XReadPixel sînt scrise în limbaj de asamblare. De data aceasta ne-am luat cîteva măsuri de precauţie, deoarece în articolul următor dorim să utilizăm mouse-ul în modul grafic 320x240x256. Atunci vom vedea la ce folosesc cele cîteva instrucţiuni care în acest moment par inutile.

Vom explica în continuare cîteva principii folosite la scrierea programului Xanim. Acesta prezintă un pătrat care se mişcă în sens trigonometric pe o traiectorie circulară, pînă ce se apasă o tastă. Pentru început se desenează fundalul în primele două pagini. Pentru a ascunde fazele pregătitoare, şi deci pentru a arăta privitorului doar rezultatul final (adică animaţia), pentru toate culorile utilizate în program am apelat procedura XSetColor, astfel că iniţial este afişat pe display un fond albastru. După ce am pregatit fundalul în ambele pagini se setează în mod corespunzător culorile utilizate şi se trece la animaţie. Totul se petrece ca la cinematograf: iniţial se fac toate pregătirile în cabina de proiecţie, după care se stinge lumina şi se derulează filmul. Animaţia este realizată după regulile următoare:

- în tabelul tridimensional Pozitii avem memorate coordonatele succesive ale pătratului;
- în fiecare moment avem o pagină de lucru (setată de procedura SetPActive), şi o pagină de afisare (setată de procedura SetPVideo);
- pagina de lucru nu coincide cu pagina de afişare;
- în pagina de lucru ştergem pătratul de pe poziţia dinainte, şi desenăm pătratul cu două poziţii în faţă (nu cu una!); pătratul va fi desenat cu o poziţie în faţă în cealaltă pagină;
- desenarea pătratului într-o poziţie este precedată de salvarea porţiunii respective a fundalului; ştergerea pătratului se face prin refacerea fundalului în poziţia respectivă;
- după ce am desenat pătratul în noua poziţie, pagina de lucru devine pagină de afişare, iar pagina de afişare devine pagină de lucru.

De ce am folosit această tehnică de programare, care la prima vedere pare cam sofisticată? De ce nu am folosit o tehnică mai simplă, folosind o singură pagină de memorie video? În cazul în care animaţia

este ceva mai complexă, operaţiile respective (refacerea fundalului şi desenarea zonei mobile în noua poziţie) pot duce la efecte mai puţin plăcute (suprapunere treptată a imaginii noi peste cea veche etc). În cazul folosirii mai multor pagini de memorie video astfel de efecte nu mai apar, deoarece privitorul vede doar pagina de afişare, în care în acel moment nu se întîmplă nimic. În momentul în care în cealaltă pagină de memorie s-a construit complet noua imagine, cele două pagini de memorie îşi schimbă în acelaşi timp rolurile (pagina de afişare devine pagină de lucru, iar pagina de lucru devine pagină de afişare).

Am văzut la început că în modul 320x240x256 pot fi folosite trei pagini de memorie ecran. În general o animaţie se realizează cu rezultate bune folosind două pagini de memorie (cum am văzut mai sus). La ce putem folosi a treia pagina de memorie în acest mod grafic? Să presupunem că avem diferite secvenţe de animaţie care se derulează succesiv (fiecare secvenţă terminîndu-se la apariţia unui eveniment extern: apăsarea unei taste, a unui buton mouse, etc.). Evident, pregătirea unei secvenţe cere oarecare timp. Pentru a nu oferi privitorului o imagine monocromă (cum este cazul programului prezentat), se poate desena un cadru ăn pagina a treia, care va fi afişat pe durata pregătirii acestei secvenţe. Acest cadru va fi static, dar se poate rezolva şi acest neajuns folosind întreruperea de timp (în revista "If" nr. 6/1990 este prezentat un model de program pe această temă). Procedura conectată la ăntreruperea de timp poate afişa în poziţii aleatoare pixeli de diferite culori. Atenţie! pentru a realiza acest lucru se apelează procedura XSetPActive cu argumentul 2, după care trebuie restaurată pagina de lucru dinaintea apelului. O metodă similară se foloseşte la televiziune în pauzele dintre două emisiuni.

Să facem cîteva observaţii în legătură cu procedurile XGetImg şi XPutImg. Apelînd procedurile XReadPixel şi XWritePixel se face de fiecare dată comutarea de pe un plan de memorie pe altul, operaţie care este consumatoare de timp. Dacă se structurează corespunzător tabloul Zons, astfel încît să se memoreze separat fiecare plan de memorie, procedurile XGetImg şi XPutImg pot fi scrise astfel încît să opereze mai întîi numai în planul 0 de memorie, apoi numai în planul 1, s.a.m.d. Lăsăm cititorului plăcerea de a rezolva acest exerciţiu de optimizare.

mat. Nelu Cozac

XGRAFICA.ASM

```
data    segment word public
        extrn   adrpagina:word
data    ends

code    segment byte public
        assume cs:code,ds:data

calcule proc    near
        cmp     cx,0
        js      calcer
        cmp     cx,320
        jae     calcer
        cmp     dx,0
        js      calcer
        cmp     dx,240
        jae     calcer
        mov     ax,0a000h
        mov     es,ax
        mov     ax,dx
        mov     bx,80
        imul    x
        add     ax,adrpagina
        mov     bx,cx
        shr     bx,1
        shr     bx,1
        add     bx,ax
        and     cl,3
        ret
calcer:
        stc
        ret
calcule endp

        public  xwritepixel
xwritepixel     proc far
        push    bp
        mov     bp,sp
        mov     cx,[bp+10]
        mov     dx,[bp+8]
        call    calcule
        jc      xwriter
        mov     ax,0102h
        shl     ah,cl
        mov     dx,3c4h
        pushf
        cli
        out     dx,ax
        mov     al,byte ptr [bp+6]
        mov     es:[bx],al
        popf
xwriter:
        pop     bp
        ret     6
xwritepixel     endp

        public  xreadpixel
```

```
xreadpixel      proc far
        push    bp
        mov     bp,sp
        mov     cx,[bp+8]
        mov     dx,[bp+6]
        call    calcule
        mov     al,0
        jc      xreader
        mov     ah,cl
        mov     al,4
        mov     dx,3ceh
        pushf
        cli
        out     dx,ax
        mov     al,es:[bx]
        popf
xreader:
        xor     ah,ah
        pop     bp
        ret     4
xreadpixel      endp
code    ends

        end
```

XGRAFICA.PAS

```
unit xgrafica; {$D-,L-,S-}

interface uses dos;
var adrpagina:word;

procedure xsetgraph;
procedure xwritepixel(x,y:word; p:byte);
function xreadpixel(x,y:word):byte;
procedure xsetcolor(c,r,g,b:byte);
procedure xsetpactive(nrp:byte);
procedure xsetpvideo(nrp:byte);
procedure xterminate;

implementation

var regs:registers; i,j,k:word;
    me:array[0..239,0..79] of byte absolute $a000:0;
const valori:array[1..10] of word=
    ($0d06,$3e07,$4109,$ea10,$ac11,
    $df12,$0014,$e715,$0616,$e317);

procedure xsetgraph;
begin
    regs.ax:=19; intr(16,regs);
    portw[$3c4]:=$604;
    portw[$3c4]:=$100;
    port[$3c2]:=$e3;
    portw[$3c4]:=$300;
    port[$3d4]:=$11;
    port[$3d5]:=port[$3d5] and $7f;
    for i:=1 to 10 do portw[$3d4]:=valori[i];
    portw[$3c4]:=$f02;
    fillchar(me,240*80*3,#0);
    adrpagina:=0;
end;

{$l xgrafica}

procedure xwritepixel(x,y:word; p:byte); external;
function xreadpixel(x,y:word):byte; external;

procedure xsetcolor(c,r,g,b:byte);
begin
    with regs do
        begin
        ah:=16; al:=16; bh:=0; bl:=c;
        ch:=g; cl:=b; dh:=r;
        end;
    intr(16,regs);
end;

procedure xsetpactive(nrp:byte);
begin
    adrpagina:=nrp*240*80;
end;

procedure xsetpvideo(nrp:byte);
var k:word;
begin
    k:=nrp*240*80;
    port[$3d4]:=12; port[$3d5]:=hi(k);
    port[$3d4]:=13; port[$3d5]:=lo(k);
end;

procedure xterminate;
begin
    regs.ax:=3; intr(16,regs);
end;

end.
```

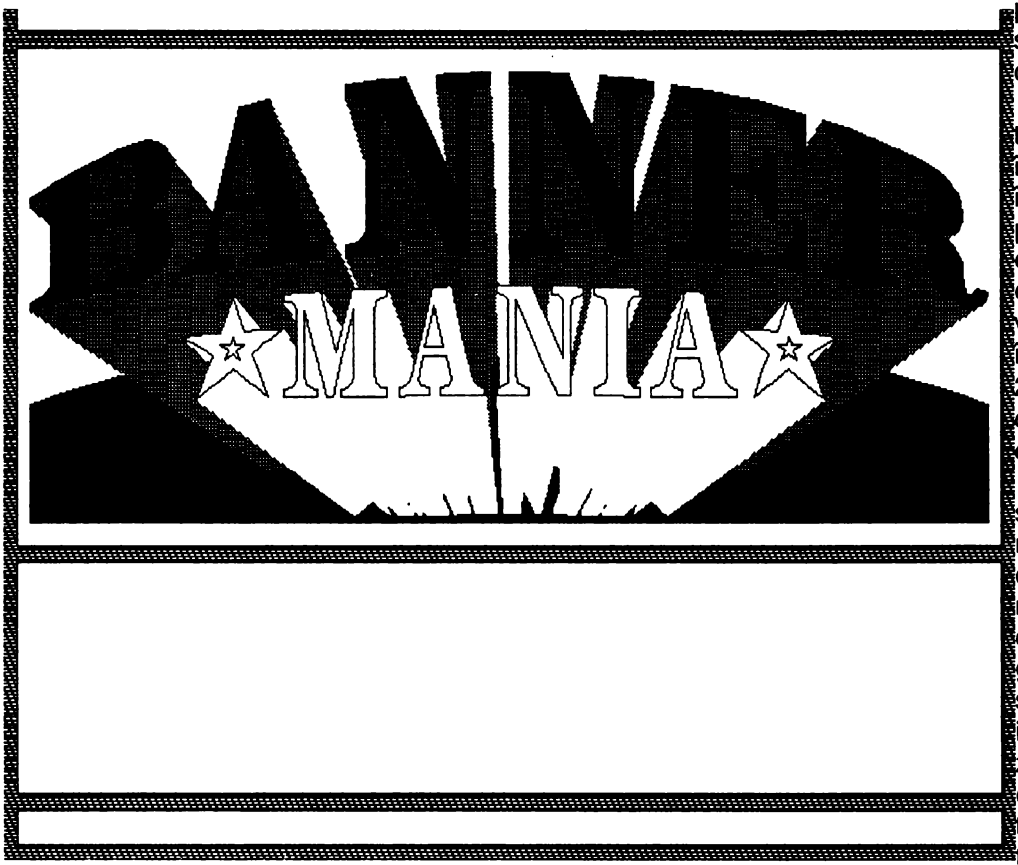
XANIM.PAS

```
uses xgrafica,crt;
const lgecran=240*80; nrpozitii=90;
var i,j,k,l,p,q,r:integer; x,y,z:real;
    pozitii:array[0..nrpozitii-1,1..2] of integer;
```

```
    zons:array[0..1,0..15,0..15] of byte;

procedure xgetimg(t:integer);
begin
    k:=pozitii[t,1];
    l:=pozitii[t,2];
    for i:=0 to 15 do for j:=0 to 15 do
        zons[p,i,j]:=xreadpixel(k+i,l+j);
end;

procedure xputimg(t:integer);
begin
    k:=pozitii[t,1];
    l:=pozitii[t,2];
    for i:=0 to 15 do for j:=0 to 15 do
        xwritepixel(k+i,l+j,zons[p,i,j]);
    end;

begin
    for i:=0 to nrpozitii-1 do
    begin
        z:=pi*i*2/nrpozitii;
        x:=100*cos(z); y:=100*sin(z);
        pozitii[i,1]:=round(x+152.0);
        pozitii[i,2]:=round(112.0-y);
    end;
    xsetgraph;
    xsetcolor(0,0,0,63);
    xsetcolor(1,0,0,63);
    xsetcolor(2,0,0,63);
    xsetcolor(4,0,0,63);
    xsetcolor(5,0,0,63);
    xsetcolor(6,0,0,63);

    for p:=0 to 1 do
    begin
        xsetpactive(p);
        for i:=0 to 319 do for j:=0 to 239 do
        xwritepixel(i,j,1);
        for i:=0 to 319 do
            begin
            for j:=0 to 1 do
                xwritepixel(i,j,2);
            for j:=119 to 120 do
                xwritepixel(i,j,2);
            for j:=238 to 239 do
                xwritepixel(i,j,2);
            end;
        for j:=0 to 239 do
        begin
            for i:=0 to 1 do
                xwritepixel(i,j,2);
            for i:=159 to 160 do
                xwritepixel(i,j,2);
            for i:=318 to 319 do
                writepixel(i,j,2);
        end;
        for i:=2 to 118 do
            for j:=0 to 2 do
                begin
                xwritepixel(i+j,i,4);
                xwritepixel(319-i-j,i,5);
                end;
            for i:=237 downto 121 do
                for j:=0 to 2 do
                begin
        xwritepixel(239-i+j,i,5);
        xwritepixel(i+j+78,i,4);
        end;
    end;

    xsetcolor(1,0,0,63);
    xsetcolor(2,0,63,0);
    xsetcolor(4,63,0,0);
    xsetcolor(5,63,0,63);
    xsetcolor(6,63,63,0);
    p:=0; xsetpactive(p); xgetimg(0);
    p:=1; xsetpactive(p); xgetimg(1);
    p:=0; q:=0;

    repeat
        xsetpactive(p); xsetpvideo(1-p);
        r:=(q+2) mod nrpozitii;
        xputimg(q); xgetimg(r);
        k:=pozitii[r,1];
        l:=pozitii[r,2];
        for i:=0 to 15 do for j:=0 to 15 do
            xwritepixel(k+i,l+j,6);
        q:=(q+1) mod nrpozitii; p:=1-p;
        delay(25);
    until keypressed;

    xterminate;
end.
```

BANNER

Programul BANNER a fost des utilizat în ultima vreme pentru realizarea unor anunţuri publicitare de dimensiuni mari, în forme estetice implicînd efecte grafice deosebite ca: evidenţierea contururilor semnelor folosite, suprapuneri prin dublare sau chiar triplare însoţite de "colorări" cu diferite nunaţe de gri, aspect de perspectivă sau spaţial, tensionări sau chiar torsionări ale înscrisurilor, încadrarea semnelor în alte forme prestabilite (ca de exemplu în balonaşe) etc.



Dat fiind că programul permite prelucrarea a doar (cel mult) două rînduri de semne - pe fiecare rînd putîndu-se afla cel mult 64 - el nu poate fi considerat sistem de editare propriu-zis. Totuşi îl apreciem ca exemplu de program bine realizat, datorită dimensiunilor reduse (nucleul său este de aproape 400 K şi nu necesită drivere speciale), portabilităţii deosebit de extinse, precum şi perspectivele pe care le deschide.

Pentru afişarea pe monitor şi imprimarea pe hîrtie programul foloseşte aceleaşi fişiere-font, pe care le recunoaşte după extensia FNT. În momentul lansării în execuţie se face o recunoaştere a acestor fişiere-font ce se află în acelaşi directory ca şi programul executabil BANNER.EXE. Aşadar utilizatorul poate crea noi fişiere-font sau îşi poate utiliza propriile fişiere-font după dorinţă.

Înainte de a trece la descrierea structurii unui astfel de fişier-font, să facem cîteva consideraţii preliminare.

1) În fiecare fişier-font original se găsesc descrierile a 123 de semne, care corespund:

- tastelor normale (95 de semne, pentru codurile ASCII între 32 şi 126 inclusiv);
- combinaţiilor `ctrl + tasta`, care dau coduri între 0 (`ctrl + @`) şi 27 (`ctrl + ]`); acestora le corespund deci 28 de semne speciale, care în fonturile originale sînt triunghiuri (în diverse poziţii), buline, stele, săgeţi diverse, semne-cărţi de joc etc.

2) Se presupune că toate semnele descrise în font au aceeaşi înălţime. Pentru a avea posibilitatea de a face comparaţii care să implice lanţurile semnelor, vom considera că înălţimea comună este 256 (împărţită în *ascender*=192 şi *descender*=64).

3) Semnele sînt descrise prin contururi, formate din segmente şi curbe Bezier. "Umplerea" zonelor plane este controlată de către program, prin opţiunile alese. Informaţii despre interpolarea de tip Bezier pot fi găsite în orice carte recentă de proiectare asistată de calculator (Computer Aided Design). Programul BANNER foloseşte doar curbe Bezier bazate pe 4 noduri.

4) Transformările imaginii grafice nu constituie o surpriză pentru cei familiarizaţi cu funcţiile complexe (şi transformările conforme); re-

comandarea celor interesaţi studierea cursurilor elementare de teoria funcţiilor complexe.

Structura unui fişier-font `nume.FNT` este următoarea:

I. Zona header, conţinînd denumirea de afişare a fontului. Această denumire, şi nu cea a fişierului-font, este preluată în program şi folosită pentru selectarea fontului.

bytes	conţinut
0 - 3	AB CD 00 05
4	lungimea L a denumirii de afişare (ca string)
5 - L + 4	denumire de afişare (string)

II. Zona de adrese relative. Această zonă conţine 250 bytes, organizaţi ca 125 de întregi (high-low).

Primii 124 ne dau adresele relative ale semnelor în zona descrieri rectoriale (începînd cu semnul 32 - blancul - a cărui adresă relativă este 00 00; menţionăm că după adresa semnului 126 urmează adresa primului semn special, corespunzător lui `ctrl + @`); cel de-al 125-lea ne indică lungimea zonei de descrieri vectoriale.

III. Zona de descrieri vectoriale. Aici, fiecărui semn descris în font i se precizează lăţimea şi conturul, într-o succesiune de bytes al cărei început şi sfîrşit pot fi aflate uşor consultînd adresele relative succesive din zona anterioară.

Descrierea semnului începe printr-un byte (unsigned), în care i se indică lăţimea, urmat de secvenţe de comenzi. Aceste secvenţe pot fi de patru tipuri (Tabelul 1).

Pentru interpretarea corectă a secvenţelor de comenzi, trebuie să ţinem seamă de următoarele. Iniţial, înainte de a prelua descrierea semnului, programul BANNER.EXE setează un flag şi rezervă loc unui "punct" special Q. În orice caz, prima comandă este de forma 60 P0 (salt la punctul iniţial P0). Înainte de executarea unei comenzi de tip polisegment

2n P_1 P_2 ... P_n, (n - cifră hex)

se testează dacă P_1 coincide cu P_0, iar în caz afirmativ se desetează flagul şi se atribuie lui Q valoarea P_0. Prin executarea comenzii se trasează segmentele P_0-P_1, P_1-P_2, ... P_{n-1} - P_n, apoi se modifică $P_0=P_n$.

Prin executarea unei comenzi

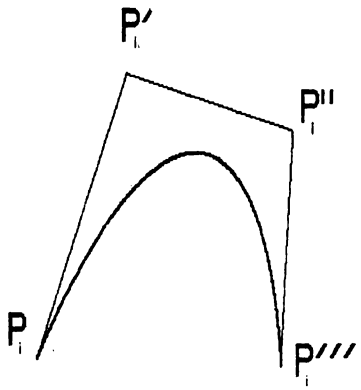
4m P'_1 P''_1 P'''_1 ... P'_m P''_m P'''_m, (m - cifră hex)

se trasează m curbe Bezier cu nodurile P_i, P'_i, P''_i, P'''_i ($1 < i < M$), unde $P_1=P_0$ iar $P_{i+1}=P'''_i$ (pentru $i < m$). În final se modifică $P_0=P'''_m$.

Înainte de executarea unei comenzi E0 sau 60 P_0, se verifică dacă flagul este desetat; în caz afirmativ se setează şi se trasează segmentul Q-P_0, apoi se execută comanda.

Sperăm că datele prezentate mai sus vor permite doritorilor să creeeze după dorinţă fonturi tip "BANNER".

Romulus Maier



Tabelul 1

Tip	Conţinut	Interpretare
1	2n x1 y1 ... xn yn	polisegment, format din segmente implicînd punctul anterior şi punctele P1,...,Pn de coordonate Pi = (xi,yi). Aici xi şi yi sînt consideraţi octeţi fără semn, iar n este cifră hex
2	4m x'1 y'1 x''1 y''1 x'''1 y'''1 ... x'm y'm x''m y''m	polilinie formată din curbe Bezier (al căror număr este dat de cifra hex m)
3	60 x y	salt la punctul P de coordonate (x,y). Aici x şi y sînt consideraţi, ca şi mai sus, octeţi fără semn.
4	E0	terminarea descrierii semnului

Cîteva probleme de logică rezolvate şi comentate în limbalul TURBO PROLOG V2.0

Să enunţăm alte probleme care pot fi rezolvate în mod facil cu ajutorul limbajului susmenţionat.

PROBLEMA 1:

Într-un bloc cu 4 etaje locuiesc familiile K, L, M, N la etaje diferite. Ştim că:

(a) familiile K şi L nu locuiesc la etaje alăturate.

(b) familia M locuieşte la unul din etajele aflate sub familia K.

(c) familia N locuieşte la unul din etajele dintre familiile L şi M.

Întrebări:

(I) Dacă familia M locuieşte la etajul 3, care este ordinea familiilor de jos în sus?

(II) Care dintre afirmaţiile de mai jos poate fi adevărată în condiţiile (a),(b) şi (c)?

(A) M locuieşte la etajul 1
(B) M locuieşte la etajul 2
(C) M locuieşte la etajul 4
(D) K locuieşte la etajul 1
(E) N locuieşte la etajul 1

Există mai multe moduri în care putem rezolva problema. Vom aborda pentru început calea cea mai simplă: vom scrie trei predicate care să conţină condiţiile din enunţ. Predicatele vor avea drept argumente patru variabile de tip "caracter", care vor simboliza cele patru familii. Poziţia unui caracter în lista de argumente determină etajul la care locuieşte familia respectivă. Predicatele vor avea numele "a","b" şi "c", conform condiţiilor din enunţ.

```
PREDICATES
      a(CHAR,CHAR,CHAR,CHAR)
      b(CHAR,CHAR,CHAR,CHAR)
      c(CHAR,CHAR,CHAR,CHAR)
rezultat

CLAUSES
      a('K',_,'L',_).          %1
      a('K',_,_,'L').          %2
      a(_,'K',_,'L').          %3
      a('L',_,'K',_).          %4
      a('L',_,_,'K').          %5
      a(_,'L',_,'K').          %6

      b('M','K',_,_).          %7
      b('M',_,'K',_).          %8
      b('M',_,_,'K').          %9
      b(_,'M','K',_).          %10
      b(_,'M',_,'K').          %11
      b(_,_,'M','K').          %12

c('L','N','M',_).              %13
c('L','N',_,'M').              %14
c('L',_,'N','M').              %15
c(_,'L','N','M').              %16
c('M','N','L',_).              %17
c('M','N',_,'L').              %18
c('M',_,'N','L').              %19
c(_,'M','N','L').              %20

rezultat:-
a(A,B,C,D),                    %21
b(A,B,C,D),                    %22
c(A,B,C,D),                    %23
write(A,B,C,D),nl,             %24
fail.                          %25

GOAL    rezultat.
```

Clauzele %1..6 ale predicatului "a", au fost scrise cu argumentele 'K' şi 'L' aşezate în aşa fel încît să satisfacă condiţia pusă la punctul (a). Pentru aceasta am scris toate variantele în care 'K' şi 'L' nu sînt alăturate.

Clauzele predicatelor "b" şi "c" vor fi scrise la fel, în

mod corespunzător enunţului.

În final am definit predicatul "rezultat", care foloseşte clauzele anterioare pentru a găsi răspunsul. Observăm că predicatele %7, %8 şi %9 au argumentele cu acelaşi nume aşezate în aceeaşi poziţie. Această aranjare a argumentelor va folosi instanţierea, adică legarea variabilelor, iniţial libere, de valori care să fie acceptate concomotent de predicatele "a","b" şi "c". Instanţierea se va face prin testare, pînă cînd va rezulta o soluţie. Mai precis, se ia prima clauză şi se atribuie valoarea 'K' pentru primul argument şi valoarea "L' pentru al treilea argument. Argumentele care nu ne interesează în acest moment au fost reprezentate prin variabila anonimă "_". Deoarece clauzele sînt declaraţii ale unei baze de date, compilatorul va genera un mesaj benign de avertizare, prin care ne spune că toate argumentele anonime vor fi legate în mod automat la o valoare de referinţă. Apoi se va încerca instanţierea argumentelor din prima clauză a predicatului "b". Acest lucru nu va reuşi deoarece argumentul 'K' nu figurează în aceeaşi poziţie în cele două clauze. În această situaţie vor fi testate următoarele clauze pînă cînd nu se va găsi un conflict între argumente. Observăm că prima clauză nu poate realiza o instanţiere cu nici o clauză a predicatului "b". Ca urmare prima clauză a predicatului "a" va fi abandonată şi va fi abordată cea de-a doua clauză. Acest mecanism continuă pînă cînd se identifică acele clauze ale predicatelor "a","b" şi "c", care, în mod simultan, nu intră în contradicţie între ele. Aceasta va fi prima soluţie, respectiv

MKNL

rezultată din clauzele %3, %7 şi %19. Predicatul %24 va scrie cele patru variabile pe ecran şi va trece la rînd nou.

Predicatul %25, "fail", va sili reluarea instanţierii, pentru a găsi şi alte soluţii.(Reamintim faptul că predicatul "fail" raportează "eşec" în orice situaţie.) Într-adevăr va mai fi găsită şi soluţia :

LNMK

rezultată din clauzele %5,%12şi %13.

Din cele două soluţii, răspunsurile vor fi

(I) LNMK

(II) (A)

Acest program rezolvă în mod satisfăcător problema noastră. Observăm că este destul de uşor să greşim la scrierea clauzelor sau să omitem anumite variante oferite de aceeaşi condiţie a problemei. Înainte de a găsi o rezolvare mai elegantă a aceleiaşi probleme, să mai exersăm cu o problemă asemănătoare, pe care nu o mai comentăm:

PROBLEMA 2:

Obiectelor K, L, M, N, O, P trebuie să li se atribuie notele 1, 2, 3, 4, 5, 6, 7 conform unei liste de instrucţiuni. Din această listă s-a pierdut o parte, rămînînd condiţiile:

(a) K are nota 2

(b) N are notă mai mare decît K

(c) R are notă mai mică cu 1 decît N

(d) L şi O au note mai mari decît R

(e) P are o notă cu 6 mai mare sau mai mică decît M

(f) toate obiectele au note diferite între ele

(I) Care din afirmaţiile de mai jos este cu certitudine adevărată?

(A) P are nota 1

(B) L are nota 6

(C) R are nota 6

(D) O are nota 6

(E) K are notă mai mică decît P

(II) Dacă ştim în plus că O are nota 6, care este nota lui L?

(A) 2

(B) 3

(C) 5

(D) 4

(E) nu se poate determina

(III) Dacă ştim în plus faşă de condiţiile (a)....(f) că R are notă mai mare decît P, cîte moduri de atribuire a notelor mai rămîn posibile?

(A) 0

(B) 1

(C) 2

(D) 4

(E) nu se poate determina

(IV) Care din afirmaţiile de mai jos este cu certitudine adevărată în condiţiile (a)...(f)?

(A) M are nota 2

(B) M are nota 6

(C) M are nota 7

(D) O are nota 1

(E) O are o notă cu 1 mai mare sau mai mică decît L

Şi rezolvarea:

```
code=1200

PREDICATES
b(STRING,STRING,STRING,STRING)
c(STRING,STRING,STRING,STRING,
STRING)
d(STRING,STRING,STRING,STRING,
```

```
STRING)
   e(STRING,STRING)
raspuns1

CLAUSES
% conditiile a si f vor fi conti-
nute in toate predicatele b,c,d
si e.
b("N",_,_,_).
b(_,"N",_,_).
b(_,_,"N",_).
b(_,_,_,"N").

c("R","N",_,_,_).
c(_,"R","N",_,_).
c(_,_,"R","N",_).
c(_,_,_,"R","N").

d("R","N","L","O",_).
d("R","N","L",_,"O").
d("R","N",_,"L","O").
d("R","N","O","L",_).
d("R","N","O",_,"L").
d("R","N",_,"O","L").
d(_,"R","N","L","O").
d(_,"R","N","O","L").

e("P","M").
e("M","P").

raspuns1:-
b(D,E,F,G),
c(C,D,E,F,G),
d(C,D,E,F,G)
,e(A,G),
write(A,"K",C,D,E,F,G),nl,fail.

GOAL    raspuns1.
```

Pentru variaţie am folosit variabile de tip STRING în loc de CHAR.

Soluţiile furnizate de program vor fi următoarele:

PKRNLOM
MKRNLOP
PKRNOLM
MKRNOLP

Din aceste soluţii putem vedea că răspunsurile sînt următoarele:

(I) (c)
(II) (c)
(III) (c)
(IV) (e)

După cum am menţionat anterior, acest mod de rezolvare, deşi este simplu, nu este şi cel mai elegant deoarece trebuie să găsim noi toate combinaţiile posibile în funcţie de condiţiile din enunţ. Pentru a evita această sursă posibilă de erori, vom încerca să găsim un algoritm care caută singur toate combinaţiile posibile, din care se vor selecta ulterior doar soluţiile care satisfac condiţiile din enunţ. Să enunţăm a treia problemă pe care să o rezolvăm în acest mod şi după aceea să vedem cum se poate adapta acastă metodă pentru rezolvarea problemelor enunţate anterior.

PROBLEMA 3:

Un inspector trebuie să viziteze patru clase K,L,M,N în cursul unei zile. El doreşte ca:

1) să inspecteze clasa K înaintea claselor L şi M
2) să inspecteze clasa L înaintea clasei N
3) a treia clasă inspectată să fie M

În cîte moduri pot fi vizitate clasele şi care sînt aceste moduri?

Rezolvare:

```
DOMAINS
LISTA=STRING*

DATABASE
variante_posibile(LISTA)
variante_corecte(LISTA)

PREDICATES
    generez_variantele
    aranjamente(LISTA,LISTA)
    aranjamente1(STRING,LISTA,
LISTA)
    aflu_ponderea(LISTA,STRING,
INTEGER)
    append(LISTA,LISTA,LISTA)
    lungimea(LISTA,INTEGER)
    conditii
```

```
CLAUSES
aranjamente([],[]).

aranjamente([X|L],M)      :-
aranjamente(L,N),
aranjamentel(X,N,M).

aranjamentel(X,[A|L],[A|M]):-
    A<>X,
    aranjamentel(X,L,M).

    aranjamentel(X,L,[X|L]).

generez_variantele:-
L=["K","L","M","N"],
aranjamente(L,L1),
    assertz(variante_posibile(L1),
fail.

generez_variantele.

aflu_ponderea([],_,0).

aflu_ponderea(Lista,El,Nr):-
append(X,[El|_],Lista),
lungimea([El|X],Nr).

append([],X,X).

append([H|Tail],L,[H|Tail2]):
-append(Tail,L,Tail2).

lungimea([],0).
lungimea([_|Tail],Len):-
lungimea(Tail,T_len),Len=T_len+1.

conditii:-
variante_posibile(Lista),
    aflu_ponderea(Lista,"K",Nrk),
    aflu_ponderea(Lista,"L",Nrl),
    aflu_ponderea(Lista,"M",Nrm),
    aflu_ponderea(Lista,"N",Nrn),
    Nrk<Nrl,   Nrk<Nrm,   %conditia   1
    Nrl<Nrn,   %conditia   2
Nrm=3,   %conditia   3
retract
(variante_posibile(Lista)),
assertz
(variante_corecte(Lista)),fail.

conditii:-variante_corecte(X),
write(X),nl,
fail.

conditii.

GOAL      generez_variantele,
conditii.
```

Pentru a obţine toate variantele posibile de aranjare a elementelor unei liste am definit predicatul "aranjamente". Acesta va prelua lista şi va returna o listă cu acelaşi număr de elemente, aranjate în alt mod. Observăm că are o structură aproape identică cu predicatul de sortare a unei liste (vezi articolul "Folosirea listelor în limbajul PROLOG"), diferenţa constînd din modificarea condiţiei A>X, în A<>X. Condiţia A<>X poate lipsi atunci cînd lista nu are duplicate.

Predicatul "generez_variantele" defineşte lista pe care o prelucrăm, apelează predicatul "aranjamente" şi reţine în baza de date o soluţie cu ajutorul predicatului "assertz". Constatăm din nou prezenţa predicatului "fail" la sfîrşitul definiţiei. Acesta, după cum ne este cunoscut deja, va forţa găsirea tuturor soluţiilor, respectiv va determina clauza să reia căutarea soluţiilor pînă la epuizarea tuturor variantelor posibile. După reţinerea în baza de date din memorie a tuturor combinaţiilor, predicatul "generez_variantele" va abandona prima clauză şi o va executa pe a doua, care nu face nici o acţiune, dar asigură ieşirea din predicat.

Pentru a putea pune condiţiile din enunţ, va trebui în prealabil să definim predicatul "aflu_ponderea", care permite să aflăm poziţia oricărui element dintr-o listă. Prima clauză a acestui predicat va returna zero pentru listele vide. A doua clauză va folosi predicatul "append", care, apelat în modul

"append(X,[Element|Y],Lista)"

cu argumentele "Lista" şi "Element" cunoscute, va rupe "Lista" în punctul în care întîlneşte "Element" şi va returna lista X care conţine toate elementele de la începutul listei pînă la "Element" şi lista Y care conţine restul de elemente (adică cele de după "Element", pînă în capăt). Acum nu ne rămîne decît să numărăm elementele listei X, la care adăugăm şi "Element". Pentru aceasta am definit predicatul "lungimea" care returnează numărul de elemente dintr-o listă. Şi acest predicat a fost discutat într-un articol anterior.

Reţinem faptul că predicatul

aflu_ponderea(Lista ,Element,Nr)

cu argumentele Lista şi Element cunoscute, va returna Nr, care arată al cîtelea element este "Element"

în "Lista".

Din acest moment putem defini predicatul "conditii" pentru a selecta doar acele răspunsuri care satisfac enunţul problemei.

Din baza de date "variante_posibile", se extrag înregistrările care îndeplinesc condiţiile din enunţ şi se reţin în baza de date "variante_corecte". Astfel condiţia 1) impune ca elementul K să fie aşezat în faţa elementelor L şi M, adică :

Nrk<Nrl şi Nrk<Nrm.

În mod asemănător sînt puse şi celelalte condiţii. Din nou predicatul "fail" forţează parcurgerea întregii baze de date.

A doua clauză a predicatului va afişa conţinutul bazei de date "variante_corecte" adică:

```
["K","L","M","N"]
```

Acesta este răspunsul dorit.

Cu mici modificări se pot rezolva şi problemele anterioare.

□ Pentru problema nr.1 se fac modificările:

```
Nrk<>Nrl+1,Nrk<>Nrl-1,     %cond    a
Nrm<Nrk,                     %cond b
sau(Nrl,Nrn,Nrm)             %cond c
```

în prima clauză a predicatului "condiţii". În plus se va defini un predicat numit "sau" pentru condiţia "c" astfel:

```
PREDICATES

sau(INTEGER,INTEGER,INTEGER)

CLAUSES

sau(L,N,M):-N<l,N>M.
sau(L,N,M):-N>L,N<M.
```

□ Pentru problema nr. 2 se fac modificările:

prima clauză a predicatului "generez_variante" va fi rescris astfel:

```
generez_variante:-
L=["K","L","M","N","O","P","R"],
aranjamente(L,L1),
L1=[_,"K",_,_,_,_,_],
%conditia    (a)
assertz(variante_posibile(L1)),
fail.
```

În primul rînd a fost modificată lista L conform enunţului şi în al doilea rînd a fost introdusă condiţia (a) în acest loc deoarece aceasta evită de la început reţinerea soluţiilor care cu certitudine nu duc la rezultate corecte. Un alt motiv pentru care condiţia (a) a fost introdusă aici şi nu în predicatul "conditii" este acela că se preîntîmpină generarea unei baze de date de 7! elemente (adică 5040 de elemente) şi se vor genera doar 6! elemente (adică 720).

În prima clauză a predicatului "condiţii" se fac modificările:

```
aflu_ponderea(Lista,"K",Nrk),
aflu_ponderea(Lista,"L",Nrl),
aflu_ponderea(Lista,"N",Nrn),
aflu_ponderea(Lista,"O",Nro),
aflu_ponderea(Lista,"R",Nrr),
Nrn>Nrk,              % cond b
Nrr=Nrn-1             % cond c
Nrl>Nrr,Nro>Nrr       % cond d
sau(Lista),           % cond e
retract
(variante_posibile(Lista)),
assertz
(variante_posibile(Lista)),
fail.
```

Şi se va defini predicatul "sau" în felul următor:

```
PREDICATES

sau(LISTA)

CLAUSES

sau(["P",_,_,_,_,_,"M"]).
sau(["M",_,_,_,_,_,"P"]).
```

Modificările sînt relativ minore şi nu afectează în mod esenţial logica generală a programului iniţial. Folosind acest program ca bază de plecare, putem rezolva o gamă largă de probleme de logică. Mai mult, anumite predicate definite aici pot fi folosite fără modificări (doar eventual la declararea tipurilor de date) în alte programe.

ing. Mircea Pantea

Metode de regăsire cu multiatribut:

Metoda semnăturii.

Metodele de regăsire cu multiatribut sînt cunoscute în literatura de specialitate sub denumirea de metode ale semnăturii datorită modului de organizare şi extracţie a informaţiei. Aceste metode au fost folosite separat sau în combinaţie cu metodele de inversie pentru obţinerea de chei secundare de regăsire (Knuth 1973, Knott 1975, Nievergeld 1984). Cele mai multe dintre ele se bazează pe suprapunerea codurilor.

Fiecare cuvînt cu semnificaţie pentru regăsire este codificat printr-o anumită metodă, pentru ca apoi să fie stocat într-un fişier separat, numit fişier semnătură. Stocarea codurilor unul după altul va da performanţe maxime la regăsire dar, titimpul de acces, în cazul fişierelor semnătură mari, va fi deasemenea mare. De aceea, aceste coduri pot fi suprapuse cu SAU logic (BOOLEAN), pentru a obţine un cod (o semnătură) pentru mai multe atribute. Metoda suprapunerii codurilor a fost propusă pentru prima dată de C.N.Mooers în 1949. El a inventat un ingenios echipament mecanic care era capabil să rezolve foarte rapid cereri de regăsiri pentru o bază de date bibliografică. Extracţia cuvintelor cheie se realiza manual şi pentru hash-funcţie era utilizată o tabelă. De la început s-a pus problema minimizării probabilităţii de a avea rezultate false (false drops). Stiassny (1960) a ajuns la concluzia că probabilitatea de a avea rezultate false este minimă dacă într-o semnătură numărul de bits 1 este egal cu numărul de bits 0. În 1964 Kautz şi Singleton au încercat găsirea unei metode de proiectare a unui sistem de semnături care nu dau rezultate false. Acest lucru este posibil prin crearea unei tabele de control, dar metoda, teoretic interesantă, practic nu este realizabilă.

Bourne (1963) a ilustrat pentru prima dată această metodă. Să presupunem o înregistrare care conţine trei cuvinte cheie: "text", "metodă", "codificare". Fiecare cuvînt cheie este codificat printr-o hash-funcţie rezultînd un şir de F bits dintre care m sînt 1. Aceste şiruri sînt suprapuse cu funcţia SAU logic obţinîndu-se "semnătura" înregistrării. După cum arătam mai sus, aceste semnături pot fi stocate în fişiere semnătură sau, astfel formate, pot fi utilizate în cazul metodei de inversiune a termenilor, pentru determinarea locaţiei înregistrării.

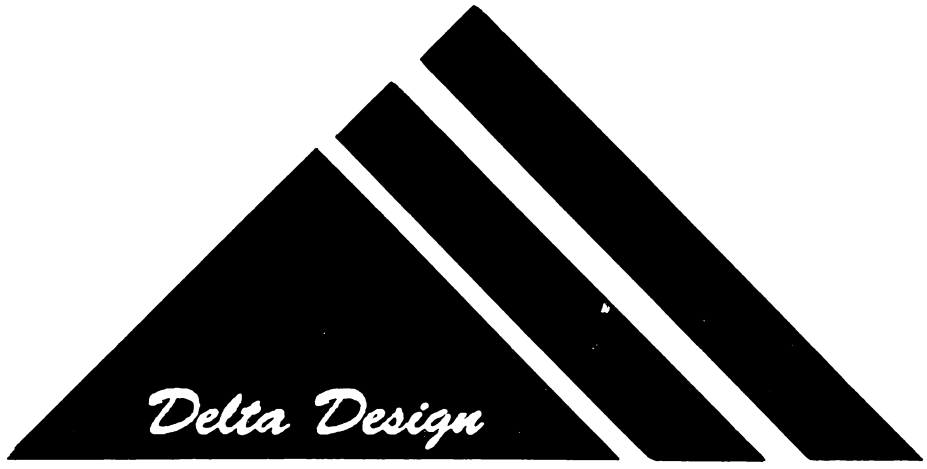
Fig. 1. Exemplu de suprapunere a codurilor (F=16, m=4):

text	001	100	001	010
metodă	000	101	011	000
codificare	001	000	001	110
semnătură	001	101	011	110

1. Modalităţi de folosire a metodei semnăturii

Pentru folosirea semnăturilor, primul care a dat o soluţie interesantă pentru determinarea locaţiei înregistrării, a fost Gustafson în 1971. El a presupus că atributele sînt cuvintele cheie care descriu un document şi o înregistrare corespunde unui document. Bineînţeles, numărul de atribute per document trebuie să fie fix (Gustafson a propus 6). Să presupunem o hash-funcţie H care face să corespundă unui cuvînt w un număr întreg H(w) în intervalul L0..15D. Semnătura unui cuvînt este un şir de 16 bits dintre care toţi sînt 0 cu excepţia bit-ului de pe poziţia H(w). Semnatura înregistrării, obţinută printr-o operaţie SAU logic (OR) va conţine k < 6 bits setaţi de valoarea 1. În concluzie, vor fi comb(16, 6) = 8008 semnaturi distincte, deci tot atîtea înregistrări. Putem reprezenta aceste 8008 posibilităţi într-o hash-tabelă cu 8008 noduri. Interesant la această metodă este faptul că numărul de căutări descreşte foarte repede (aproape exponenţial) în funcţie de numărul de termeni dintr-o cerere de regăsire (query) conjunctivă. Astfel, o cerere pentru un singur cuvînt necesită comb(15, 5) = 3003 încercări (se parcurg 3003 noduri ale hash-tabelei), una pentru două cuvinte comb(14, 4) = 1001 încercări, etc.

Mai tîrziu, au fost studiate şi propuse metode pornind de la metoda lui Gustafson (Rothnie şi Lozano în 1974, Rivest în 1976, Aho şi Ullman în 1979, Lloyd în 1980). Aceştia au propus ca semnătura unei înregistrări să fie creată prin concatenarea semnăturilor atributelor. Această idee crează probleme în special la regăsiri pentru că şi o cerere simplă necesită a fi expandată într-una disjunctivă, ceea ce înseamnă timp mare de răspuns.



Delta Design S.A., Hotel Bucureşti, 63-81 Victoria Street,
Wing D - Agencies, Stair E 2, 4th Floor, Sect. 1, Bucharest,
Phones: 15:42.26 / 40 015.45.80 Ext.2404 / Fax: 40 13.60.40.

99.94% of the users fully satisfied of the delivered products

Authorised distributors for the following companies:
Tulip Computers, Apple Computer Inc., Microsoft, Autodesk,
Ashton Tate, Claris, Corel, Delta Design International etc.

Presupunînd că avem 6 cuvinte cheie/document, şi căutăm documentul ce conţine cuvîntul "informaţie", cererea pentru regăsirea acestui cuvînt va trebui formulată:

cuv - cheie1 = "informaţie" sau
cuv - cheie2 = "informaţie" sau
....
cuv - cheie6 = "informaţie".

Mai elegantă, metoda lui Gustafson ridică următoarele probleme:

- Performanţele descresc cu creşterea fişierului;
- Numărul de cuvinte cheie dintr-un document este mare şi, în acest caz, hash-tabela trebuie să fie foarte mare; în situaţia în care este formulată o cerere cu 3-4 cuvinte, va fi parcursă o mare porţiune a bazei de date.. De asemenea, cererile care nu sînt conjunctive, crează dificultăţi.

Ideea lui Gustafson a fost preluată şi de alţi cercetători care au încercat să o facă aplicabilă, dar nici unul nu a găsit o soluţie optimă. Knott (1971) a propus utilizarea unei hash-funcţii adaptive, care se schimbă în funcţie de mulţimea şi distribuţia termenilor inseraţi. Aceste funcţii utilizează o structură de arbore suplimentară, nodurile acestei structuri fiind pointeri către grupuri de informaţie aflată pe un mediu de stocare secundar. Codul unui atribut se obţine prin parcurgerea acestui arbore. Larson (1978) a dezvoltat această idee folosind o pădure de arbori ca structură auxiliară de date. Martin (1979) sugerează o codificare în "spirală" bazată pe o funcţie exponenţială (neuniformă). Lloyd şi Ramamohanarao (1982) utilizează o cheie secundară, etc.

Aceste metode nu au dat pînă acum rezultate practice, convenabile. Cea de-a doua idee, cea a stocării semnăturilor într-un fişier separat, fişierul semnătură, este mult mai interesantă.

În acest caz nu apare restricţia legată de numărul fix de cuvinte cheie dintr-un docuemnt, pentru că documentul poate fi divizat în blocuri logice care vor conţine un număr constant de cuvinte cheie. În 1969, Files şi Huskey au aplicat această metodă pentru o bază de date bibliografică. Ei au folosit o listă de cuvinte ce se exclud de la codificare şi o procedură de evitare a acestora. Pentru codificare, au utilizat o procedura numerică drept hash-funcţie.

Tsichritzis şi Christodoulakis (1983), urmaţi de Larson (1983) au propus utilizarea fişierului semnătură fără a întrebuinţa suprapunerea codurilor. În felul acesta, poziţia informaţiei este păstrată, pentru că sînt stocate semnăturile fiecărui cuvînt, una după cealaltă (concatenate). În 1984, Rabitti şi Zizka au atras atenţia că metoda prin concaternare implică legături mai greoaie cu CPU (unitatea centrală) decît cea a suprapunerii codurilor.

Pflatz (1980), a sugerat utilizarea unui descriptor indexat de fişiere (în terminologia acestuia), de fapt un sistem de mai multe nivele de fişiere semnătură. Semnătura din nivelul i este obţinută prin suprapunerea semnăturilor de pe nivelele i - 1. Semnătura înregistrărilor de pe primul nivel este obţinută prin concatenarea semnăturilor corespunzătoare cuvintelor cheiei din înregistrare.

Fig. 2. Fişiere semnătură pe mai multe nivele:

Fişier de date			descriptor nivel 1	descriptor nivel 2
Nume	Sex	Salariu		
Smith, J.	M	40000	010 0 01	010 1 11
Smith, M.	M	50000	010 0 10	100 0 11
Smith, P.	F	20000	010 1 00	
Wong, B.	M	15000	100 0 00	
Wong, M.	M	60000	100 0 11	
Wong, T.	M	50000	100 0 10	

Roberts (1979) a folosit o metodă cu un singur nivel de fişier semnătură pentru o aplicaţie tip agendă telefonică. El a propus următoarele idei interesante:

- Stocarea semnăturilor în fişierul semnătură se face în maniera "bit-slice", adică primul bit al tuturor semnăturilor este stocat consecutiv, apoi al doilea, şi aşa mai departe. Dacă îngreunează actualizările, această metodă reduce foarte mult timpul de regăsire.
- Crearea semnăturilor în aşa fel încît termenii care apar frecvent în cererile de regăsire să fie trataţi într-un mod special.

În 1985, Faloutsos a făcut o analiză matematică a modului de organizare a fişierului semnătură pentru evitarea rezultatelor false la regăsire, şi a stabilit un sistem de reguli ce trebuiesc aplicate la proiectarea unei unui fişier semnătură pentru a putea reduce cu 50% rezultatele false (false drops).

Principalele dezavantaje sînt legate de timpul de răspuns mare în cazul atunci cînd fişierul semnătură este mare şi bineînţeles, probabilitatea răspunsurilor false.

2. Metode de proiectare a fişierului semnătură şi de extracţie a semnăturilor

Semnătura fiecărui cuvînt (Word Signature)

Vom nota în continuare această metodă cu WS. Fiecare cuvînt al unui document (cu excepţia cuvintelor comune: şi, sau, etc.) este codificat într-un şir de F

bits (nu intervine un număr fix de bits 1) şi aceste coduri sînt concatenate pentru a forma semnătura documentului. Pentru regăsire, este codificat cuvîntul cerut şi apoi, este parcurs întreg fişierul semnătură, pas cu pas.

Fig. 3. Ilustrare a metodei WS

	text	metodă	codificare
Semnătura cuvîntului	0000	0100	0111
Semnătura documentului	0000	0100	0111

Suprapunerea codurilor
(Superimposed Coding)

Ne vom referi la această metodă, în continuare, prin SC. Ea a fost propusă de Christodoulakis şi Faloutsos în 1984 şi se bazează pe metoda propusă de Moores în 1949. Fiecare document este divizat în blocuri logice, conţinînd fiecare D cuvinte distincte şi necomune. Fiecare cuvînt este codificat într-un şir de F bits, dintre care m sînt 1, apoi aceste coduri sînt suprapuse cu SAU logic pentru a obţine semnătura blocului (vezi Figura 1). Alegerea valorilor pentru D,F şi m se face în aşa fel încît numărul de bits 1 din semnătura blocului să reprezinte jumătate din numărul total de bits, în felul acesta realizîndu-se optimizarea performanţelor. Pentru căutarea unui cuvînt, se formează mai întîi semnătura acelui cuvînt. Să presupunem că va avea bits 1 în propoziţiile 1, 5, 7, 9. Vor fi examinate pe rînd semnăturile blocurilor şi, dacă în propoziţiile 1, 5, 7, 9 va exista 1, acele blocuri vor fi considerate ca avînd inclus acest cuvînt.

În ideea de a avea posibilitatea de regăsire a termenilor trunchiaţi, pentru a fi codificare şi stocare, cuvintele sînt divizate în triplete, avînd marcat începutul şi sfîrşitul.

Fig. 4. Exemplu pentru codificare dacă se doreşte regăsire de termeni trunchiaţi.
I = "Început", E = "Sfîrşit"
Sigw(data)=Sig(I,d,a) OR Sig(d,a,t) OR Sig(a,t,a) OR Sig(t,a,E)

Compresia blocurilor de bits
(Bit-block compression)

Această metodă va fi referită prin BC. Propusă pentru prima dată de Faloutsos (1985), metoda BC presupune utilizarea pentru codificare metoda SC, ca apoi vectorul de bits corespunzător unui bloc să fie comprimat separat. Comprimarea se poate face în mai multe feluri, Faloutsos a propus memoratea poziţiei bits 1 în cadrul blocului.

Codificarea pe întreaga lungime
(Run - length encoding)

McIlroy, în 1982, a aplicat pentru prima dată această metodă, la care ne vor referi cu iniţialele RL. Este o metodă de compresie, McIlroy comprimînd cu ajutorul ei un dicţionar de 30000 de cuvinte. Fiecare cuvînt este codificat într-un vector de bits conţinînd un număr fix de bits 1. Ceea ce se memorează este numărul de 0 dintre două apariţii succesive de 1. Avantajul acestei metode este legat de compresia excelentă pe care o realizează, dar, timpul de răspuns la interogare este foarte încet. Deasemenea, vectorul trebuie să conţină zone largi de bits 0, pentru ca compresia să fie eficientă.

Codificarea în ordinea poziţiei
(Bit - Slice)

Metoda BS, este, după cum se va vedea foarte eficientă la regăsiri, dar pune probleme serioase operaţiilor de actualizare sau ştergere. Codificarea cuvintelor se face cu o hash-funcţie, şi memorarea lor se realizează în felul următor: prima dată sînt memoraţi bits de pe prima poziţie, apoi cei de pe a doua, ş.a.m.d. Se poate aplica această metodă în combinaţie cu metoda SC, obţinîndu-se semnături ale blocurilor care sînt stocate aşa cum a fost arătat mai sus.

Fig. 5. Exemplu pentru metoda BS.

Să presupunem că avem semnăturile a 4 blocuri, F=5:
(01001), (01101), (11101), (00110)
Memorarea acestora în fişierul semnătură se va face:
(0010), (1110), (0111), (0001), (111))
Să presupunem că se doreşte regăsirea cuvîntului a cărui semnătură este Sig(q) = (00101). Algoritmul de verificare va cerceta doar poziţiile 3 şi 5 (acolo unde este 1) şi rezultă pentru poziţia 3 blocurile 2,3,4 şi pentru poziţia 5 blocurile 1, 2, 3. Rezultatul se obţine prin intersecţia celor două răspunsuri, deci cuvîntul căutat se va afla în blocurile 2 şi 3.

3. Comparaţie între metode.

Toate variantele discutate ale metodei semnăturii pot da răspunsuri false ("fale drops"), pentru că este posibil ca o semnătură să corespundă unei întrebări, dar de fapt cuvîntul respectiv să nu existe în textul respectiv. Notăm probalitatea de a avea un răspuns fals cu Fd.

Fd = probabilitatea ca o semnătură să rezulte în urma aplicării metodei că aparţine unui text şi de fapt să nu aparţină.

Faloutsos şi Christodoulakis au studiat în 1984 Fd pentru metodele WS şi SC.

În cazul WS, Fd este:

$$Fd,ws = 1 - (1 - 1/Smax)^{D} \quad (1)$$

Ecuaţia (1) poate fi justificată într-un mod intuitiv. Ea indică faptul că Fd este independentă de dimensiunea vocabularului, a bazei de date sau de frecvenţa apariţiilor unor cuvinte.

În cazul SC, Fd este:

$$Fd,SC = (1/2)^{mopt} \quad (2)$$

$$mopt = F*ln2/D \quad (3)$$

Aceste valori au fost determinate statistic pentru o bază de date cu intrări bibliografice de 3,3 MegaByte. Ecuaţia (3) implică concluzia că Fd este minim atunci cînd jumătate din numărul de bits din blocul semnătură este format din bits 1.

În 1985, Faloutsos a stabilit formule aproximative din care poate rezulta variaţia Fd în funcţie de creşterea dimensiunii F a semnăturii.

$$\log 2Fd,ws = \log2D - Fws/D \quad (4)$$

$$\log 2Fd,SC = -FSC / (D*log2e) =$$

$$= -FSC / D*0,693 \quad (5)$$

$$\log 2Fd,RL = n*(1+log2log 2e) - FRL/D=$$

$$= 1,528*n - FRL/D \quad (6)$$

$$\log 2Fd,BC = n*(1+log2e-log 2log 2e)-$$

$$- FBC/D = 1,913*n - FBC/D \quad (7)$$

(Nu am găsit o formulă pentru metoda BS)

În aceste ecuaţii:

$F_{d,xx}$ = F_d pentru metoda XX

F_{xx} = Dimensiunea blocului semnătură pentru metoda XX

D = Numărul de cuvinte noncomune din bloc

S_{max} = Numărul maxim de semnături distincte (WS)

f = Dimensiunea semnăturii cuvîntului în bits (WS)

m = Numărul bits 1 din semnătura blocului (SC)

m_{opt} = Valoarea optimă pentru m (SC)

B = Dimensiunea vectorului (BC, RL)

n = Numărul bits 1 corespunzători unui cuvînt (BC, RL)

b = dimensiunea blocului de bits (BC)

b_{opt} = valoarea optimă pt. b (BC)

Concluziile ce rezultă din aceste formule şi din studii experimentale şi statistice sînt:

- Pentru metode bazate pe compresie, cele mai bune rezultate se obţin pentru n=1.
- Pentru toaye metodele, curbele devin linii drepte pentru diemnsiuni mari ale semnăturilor.
- Cel mai bun timp la regăsire îl oferă BS şi apoi SC.
- RL este cea mai economicoasă metodă în privinţa spaţiului folosit.
- BC reprezintă un compromis între SC şi RL.
- WS păstrează ordinea apariţiilor cuvintelor în texte.
- Metoda BS ridică probleme la actualizare.
- Dimensiunea fişierului semnătură este, pentru metodele SC, BS, WS aproximativ 10% din documemntele originare (baza de date), în comparaţie cu un fişier invers ce ocupă aprox. 300%.

În momentul cînd, în urma studiului materialelor din bibliografie, am construit structura de date abstracte Signature pentru arhivare şi regăsire de date documentare prin metoda semnăturii, am ales metodele SC şi BC pentru implementare ca avînd cele mai bune performanţe şi fiind uşor de implementat. În cazul unei aplicaţii ce foloseşte această structură de date abstracte, în funcţie de tipul acesteia (se cere un timp de răspuns foarte bun sau nu, se fac deseori actualizări sau nu, etc.) se alege una din soluţii.

Pentru metoda SC am ales:

```
F = 256;
D = 40;
```

atunci

$$mopt = F*ln2/D = 4,58$$

De aici, am ales

$$m = 5.$$

Problema metodei semnăturii rămîne încă o problemă de cercetare, dezvoltare şi optimizare viitoare.

aprilie, 1992

ing. Mirela Guţică

Bibliografie:

- Access Methods for Text - C. Faloutsos - 1985;
- Signaturverfahren fur Sekundarschlussel-Zugriff - H. Eirund - Oldenburg - W. Deutschland, 1991

MODULELE MULTICIP

UN NOU CONCEPT PENTRU ÎNCAPSULAREA DE MARE DENSITATE

Introducere

În ultima decadă se vorbeşte tot mai mult despre circuitele integrate specifice pe aplicaţii, deoarece aceste ASIC (fie ele reţele de porţi sau circuite complet la comandă) au evoluat rapid dintr-o alternativă de proiectare neobişnuită şi costisitoare către metoda preferată (cea mai eficientă) de proiectare a sistemelor.

Datorită progreselor tehnologiilor semiconductoare şi perfecţionărilor majore în domeniul proiectării şi producţiei asistate de calculator, problemele de proiectare şi realizare structuri au fost rezolvate, dar se ivesc altele: interconectare, încapsulare, montare în echipament. Tehnologia clasică, ce a fost utilizată cu succes pentru încapsularea dispozitivelor LSI, nu mai poate satisface cerinţele VLSI. De fapt, multe din beneficiile cîştigate de VLSI pe cip se pierd acum la încapsulare. Plăcile de cablaj multistrat şi circuitele hibride cu straturi groase [1], cu dimensiunea minim absolută a traseelor de 125/μm (pas de 250/μm), sînt inadecvate pentru interconectarea corespunzătoare a acestor dispozitive complexe, cu număr foarte mare de intrări şi ieşiri. Este necesară o tehnologie care să accepte o densitate de încapsulare sporită şi viteze mai mari ale semnalelor.

La vitezele de lucru ridicate la care s-a ajuns în prezent apar certe limitări în privinţa capsulelor. Astfel, PQFP (capsulele plate de plastic) actuale, cu montare pe suprafaţă, merg foarte bine la frecvenţa de 50 Mhz şi poate chiar 80 MHz (necesitînd terminale suplimentare de masă şi de alimentare pentru ecranarea laterală împotriva diafoniei), dar nu şi la peste 100 MHz.

Mai importantă este căldura disipată generată de circuitele ultrarapide actuale, chiar şi în tehnologia CMOS. Unele tipuri depăşesc 1 W, apropiindu-se de limita de 1,2 W a capsulei PQFP. Înseamnă că sînt necesare noi materiale de încapsulare, cu conductivitate termică mare, sau noi soluţii constructive de PQFP, care să încorporeze radiatoare sau "vias" termice.

Încapsularea în sistemul TAB (Tape Automated Bonding), extensia logică pentru QFP, permite densităţi mai mari de interconectare (cercetarea de laborator arată că se pot chiar depăşi 1000 de terminale, dar în viitorul apropiat, aplicaţiile uzuale, eficiente ale TAB vor fi pentru circuite cu 300 - 400) însă ridică probleme la "implementare".

Procesul clasic de lipire prin retopire (solder reflow) permite, la majoritatea fabricanţilor, montarea capsulelor cu pas de 0,63 ... 0,5 mm. Dar, pentru mai mult de 400 de terminale, este necesar, conform tabelului de mai jos, un pas mai mic de 0,4 mm, care pune în pericol montarea subsistemelor, deoarece probabilitatea formării scurtcicuitelor (punţi de aliaj de lipit) şi a golurilor (circuite deschise) devine foarte mare. Deasemenea, la un pas de 0,4 mm, trebuie utilizat un sistem de recunoaştere automată a formelor, pentru a plasa cu precizie componentele ASIC pe placă. Adoptarea tehnologiei TAB cu pas fin înseamnă ca producătorii de sisteme trebuie să treacă la o combinaţie de retopire convenţională şi sudură a terminalelor exterioare (OLB), aşa cum a fost pusă la punct pentru TAB.

Tabelul 1 . Relaţia între numărul de terminale şi pasul necesar:

Nr. pini	200	300	400	500	600	700	1000
Pas(mm)	0.6	0.5	0.4	0.3	0.3	0.1	0.04
	clasică		AB OLB				

Adoptarea deciziei de trecere la TAB duce la unele consecinţe inevitabile:

- un echipament de producţie pentru sudura conexiunilor exterioare pentru TAB poate costa peste 100.000$. Sudura OLB este un proces secvenţial, relativ lent comparat cu abordarea clasică pentru circuitele ASIC cu montarea pe suprafaţă: robot de plasare tip pick-and-place şi cuptor de retopire, deci o prelucrare în paralel, pe loturi. În loc de mai multe componente pe secundă, durează mai multe secunde interconectarea unui dispozitiv;
- inginerii producători de sisteme trebuie să înveţe să decupeze si să plieze terminalele intr-o forma adecvata locului de amplasare. Pînă acum, operaţia era efec-

tuată pentru ei de fabricantul de dispozitive la cerere. Se obţine astfel un avantaj prin faptul ca se minimizează manevrarea şi transportul dispozitivelor cu terminalele formate, care sînt fragile. Se reduce astfel deplasarea terminalelor şi apariţia de necoplaneităţi;

- apar considerente de proiectare termică. În tehnologia TAB cu pas fin, inginerii de sistem vor avea de făcut faţă unor densităţi de putere disipată de 4 - 10 ori mai mare. Volumul mare al circuitului integrat, atunci cînd era încapsulat de către producătorii de ASIC, putea disipa mult mai multă căldură decît o face structura neîncapsulata cu care lucrează proiectanţii în tehnologia TAB. Dacă proiectanţii de sisteme vor utiliza circuite ASIC cu pas fin direct pe placă, ei trebuie să înţeleagă şi să rezolve problemele termice suplimentare ce apar, atît pe placă cît şi la nivel de sistem;
- în fine, costurile iniţiale asociate tehnologiei TAB sînt ridicate. Producătorii de echipamente mai ambiţioşi ce doresc să treacă la tehnologia de mare densitate, să facă tranziţia de la tehnica tradiţională, de montare a componentelor în găuri la varianta de montare pe suprafaţă cu pas fin, trebuie să investească între 3 şi 10 milioane de dolari, sau chiar mai mult, în funcţie de volumul de producţie şi dezvoltările implicate.

Noua soluţie - MCM

Mai mult ca oricînd, pentru îmbunătăţirea performanţelor ASIC sînt esenţiale metodologii de încapsulare noi. Limitările de interconectare (rutare) şi fabricaţie ale cablajelor imprimate convenţionale creează o cerere mare pentru plăci cu densitate de linii mai mare fără un preţ considerabil sporit. La fel, menţinerea densităţii actuale a plăcilor şi dezvoltarea unor noi scheme de încapsulare a componentelor ar putea satisface cerinţele noilor produse. Între timp, o tehnologie evoluţionară de încapsulare, de departe asemănătoare cu circuitele hibride, prezintă soluţii clare pentru aceste cerinţe de ASIC cu pas fin care tocmai apar. Modulele multicip (MCM), ce conţin o putere de calcul considerabilă sub forma de VLSI în cadrul a numai cîteva componente, apar acum în capsule obişnuite, precum DIL sau QFP. Ca sistem selfconsistent, MCM-urile prezintă proiectanţilor o capsulă cu un număr mai redus de terminale/ pini, ce pot fi manipulate în mod obişnuit de echipamentele de producţie deja existente.

Principiul constă în utilizarea interconectărilor cu straturi subţiri pe siliciu. În structurile de siliciu este inerentă o interconectare cu densitate foarte mare de trasee, mult superioară celei ale cablajelor imprimate clasice sau modulelor multistrat pe ceramică (MCA). Această capabilitate se aplică la MCM-uri pentru a produce substraturi pe baza de siliciu, în fabrici de plachete depăşite moral. Uzura morală a fabricilor de VLSI fiind foarte ridicată actualmente, reprofilarea lor pe efectuarea de interconexiuni pentru modulele multicip constituie o "valorificare superioară" ce nu-i de neglijat. Se pot realiza curent interconexiuni pe substraturi de siliciu, care sînt cu cel puţin un ordin de mărime mai avansate decît substraturile clasice (cablajele sau circuitele hibride pe ceramică).

Un substrat de Si activ pentru MCM reduce deasemenea puterea totală disipată, ceea ce înseamnă o reducere majoră de preţ. Cînd se ataşează mai multe cipuri la un substrat de modul, liniile / căile de semnal sînt mai scurte, ceea ce reduce puterea de comandă necesară pe cip. Ori, se ştie că la CMOS disipaţia statică este foarte redusă, iar cea dinamică depinde de frecvenţa de lucru, tensiunea de alimentare şi capacităţile de sarcină. Liniile mai scurte înseamnă capacităţi mai reduse. Cînd substratul de siliciu este populat cu elemente active şi pasive precum tranzistori de comandă, rezistoare de sarcină (adaptare, terminatoare de linie, tragere "pull-up" la alimentarea pozitivă), condensatoare, puterea degajată în modulul respectiv se reduce cu circa 20%. Acest nivel este semnificativ pentru minimizarea costului sistemelor de înaltă performanţă bazate pe circuite ASIC.

La nivel elementar, MCM de astăzi combină 4 - 10 elemente diferite pe un substrat, fie cu straturi subţiri, fie cu straturi groase. O derivată a MCM, denumită placă funcţională, apare acum ca o tehnologie mai avansată ce adăposteşte tehnologii LSI complexe. Cele mai multe aplicaţii pentru plăcile funcţionale au fost fixate să utilizeze un conector clasic cu 60 de pini la capăt sau un conector cu 120 de pini pe laterală.

Iniţial, acest MCM particular a debutat ca o extensie de memorie (pe placă) în sistemele de calcul (add- in). Pe placa de bază a calculatorului pot fi incluşi 2, 4 sau 16 Mb de memorie. Acum, datorită cipurilor pe bandă TAB, plăcile funcţionale pot include modemuri, procesore grafice, coprocesoare matematice sau de comunicaţii, controllere pentru interconectarea în reţeaua locală (LAN).

Un astfel de modul multicip avansat poate conţine, la dimensiuni de numai 5 x 5 cm, pînă la 45 de cipuri de integrate LSI. De exemplu, memoria tridimensională produsă de Texas Instruments conţine 8 EEPROM-uri de 256 K pe 9 biţi şi 32 de cipuri de RAM statice de 1 M a 4 biţi, plus 2 circuite specifice (ASIC) şi 3 de prelucrare digitală a semnalelor (DSP). Folosind organizarea tridimensională, substraturile de Si şi interconectarea flip-TAB în interiorul modulului, exteriorul se prezintă ca o capsulă "clasică" pentru montarea pe suprafaţă cu pas fin. Într-adevar, Quad - Flatpack-ul respectiv are 308 terminale cu pas de 0,635mm şi poate fi montat cu echipamentele actuale, în vreme ce în

interior sînt 800 de conexiuni, interconectate cu un pas foarte fin.

Producătorii de sisteme trebuie să constate care tehnică de încapsulare corespunde cel mai bine nevoilor unor sisteme particulare. Trebuiesc evaluate cu grijă mai mulţi parametri de selecţie, ce includ costul, înăltimea, disipaţia termică, nivelul de performanţe, materialele si factorul de formă. Deşi flip - TAB- ul merge bine, pot exista şi alte abordări, precum urmează:

1. În metoda devenită clasică, cipuri pe placă interconectate TAB (TAB COB), structurile sînt pe cablaj cu bumbii în sus, iar terminalele exterioare ale grilei TAB sînt pliate astfel încît să emuleze o capsulă convenţională pentru montarea pe suprafaţă, cu terminale în aripa de pescăruş. Această variantă este aplicată astăzi în multe sisteme comerciale, fiind relativ ieftin de implementat, iar factorii de formă sînt similari celorlalte capsule.

2. Dacă minimizarea spaţiului pe verticală este de o importanţă deosebită, cum ar fi aplicaţiile pentru "module inteligente" ce necesită o grosime minimă (cărţi de credit inteligente), atunci este mai bine de utilizat metoda cu cipuri în placa cu TAB cu "buzunar" (P-TAB CIB). In aceasta variantă (Chip In Board, deci cipuri în interiorul plăcii) cipurile sînt plasate în cavităţi ale substratului (buzunare) astfel încît înălţimea totală se reduce.

3. În fine, dacă evacuarea puterii disipate este principala problemă a sistemului (dispozitive de putere, circuite ultrarapide), se utilizează cipuri răsturnate pe placă, interconectate TAB. Cipurile sînt cu faţa cu bumbii de contact în jos, iar spatele este disponibil pentru ataşarea la un sistem de disipare termică adecvat, care să asigure performanţele termice dorite.

Perspective

În viitorul apropiat, MCM pe substrat de siliciu vor permite proiectarea şi realizarea unor funcţii foarte puternice în module de numai 5 x 7,6 cm. Exemplul reluat de mai multe reviste occidentale prezintă un cablaj special (cu miez de Invar) cu componente în tehnologia montării pe suprafaţă, suporturi de structuri cu şi fără terminale, cu paşi fini, de 0,65 şi 0,5 mm. Placa aceasta, "varianta veche", are o suprafata de 15,5 x 15,5 cm^2, iar echivalentul funcţional "nou" în MCM (cu componente flip - cip TAB) are dimensiuni 6 ori mai reduse. Această abordare nu este încă în producţia de serie, dar în laborator se iveşte deja următoarea generaţie de MCM-uri. Folosind pe substrat structuri flip-cip, se pot reduce în continuare dimensiunile, cu preţul unor complicaţii la montaj. Substratul multicip poate fi apoi înglobat într-o capsulă clasică sau ataşat direct pe placa de cablaj. În următorii doi ani (91 - 93), tehnologia MCM se va infiltra în aplicaţiile de sistem de înaltă performanţă, atît în domeniul comercial, aerospaţial cît şi militar. Desigur, în special armata va exploata reducerile de dimensiuni şi greutate pe care le asigura MCM-urile. Sistemele existente vor fi îmbunătăţite (upgraded) amplasînd funcţii şi posibilităţi suplimentare într-o cutie de aceleaşi dimensiuni.

MCM avansate

Etapa ulterioara (1993 - 95) va consta din module multicip care încep să-şi adauge blocuri constructive tridimensionale (spaţiul de deasupra plăcii de interconectare adăposteşte circuitele integrate. Într-o încapsulare 3D, dispozitivele active sînt stivuite şi montate perpendicular pe planul de interconectare ce distribuie semnalele şi tensiunile de alimentare între dispozitive. Este cea mai densă dintre tehnicile de împachetare cunoscute - dacă puterea disipată nu ridică probleme, încapsularea tridimensională promite o îmbunătăţire a densităţii de memorie de pînă la 250 - 300 de ori faţă de DIP-uri sau SOIC-uri. Spre jumătatea anilor '90 modulele de sistem vor fi bazate pe conceptul de placă funcţională. Vor folosi substraturi de siliciu şi vor beneficia de avantajele celei de-a treia dimensiuni. Aceasta se realizează în prezent în laborator cu sistemele de montaj ce folosesc tehnologiile TAB şi flip-cip.

În plus, topologia de substraturi adaptata aplicaţiei rezolvă problemele de interconectare între numeroasele cipuri.

Substraturile speciale de siliciu şi MCM-urile stivuite disipă suficientă căldură ca să accepte densităţile mari de putere ce rezultă din cipurile strîns împachetate. Siliciul are o bună conductivitate termică, iar acolo unde este necesar se utilizează disipatoare de beriliu. Un avantaj suplimentar al substraturilor active de siliciu este posibilitatea de á încorpora circuite funcţionale complexe şi/sau componente pasive.

Cristian Malide - ICCE Bucureşti

Bibliografie

1. Brevetul US 4.551.788 din 5 nov. 1985, pe numele lui Daniel, Larry Yglesia intitulat "Multi-Chip Carrier Array".

2.Brevetul US 4.764.846 din 16 aug. 1988, atribuit lui Go, T.G. de la Irvine Sensors, SUA, "High Density Electronic Package Comprising Stacked Sub-Modules",

3. Rossi, R.D. (National Starch Chemical), "High Density Interconnection Packaging of I.C.'s", HCT sept.'89, pag. 35.

4.Barbou, R. (Sextant Avionics), "Hybrids + ASIC's= A High Density Packaging Concept", HCT nov.'89, pag. 11.

5.Buschbom, M.L. (Texas Instruments), "ASIC Designs Decisions Demand Broader Perspective", Electronic Design, 11 Jan.'90, pag. 111.

MIFIX

GESTOC

REPARATII

METROLOGIE

ISCIR

CASA

BANCA

SALARII

FACTURARI

PLAY.ASM

Vă propun un program care crează un fond sonor interesant. Acest program se instalează rezident în memorie şi lucrează cu trei întreruperi: cea de disc (int 13H), cea de tastatură (int 09H) şi cea de timp (1CH).

Funcţionare: orice acces la disc va trece printr-o procedură care, dacă accesul reprezintă o scriere/citire/formatare, va emite un sunet a cărui frecven1ă depinde de cilindrul şi sectorul accesat. (Pentru asta există vectorul "tabela", care asociază '1' unei asemenea funcţii şi '0' unei funcţii de resetare/interogare etc. - codul funcţiei este primit în registrul AH). Procedura care lucrează pe întreruperea de timp va opri semnalul sonor după circa 0.33 secunde, pentru a nu deveni supărător. Procedura pentru tastatură va comuta (sunet sau nu) după cum este sau nu activă tasta Scroll Lock.

Programul este absolut original.

ing. fiz. Cristian Paris

PLAYASM

```
;
.RADIX 16
;           Constante:
PORT_A  equ     60h         ; adresa port A 8255
PORT_B  equ     61h         ; adresa port B 8255
TIMER   equ     40h         ; adresa port timer 8253
;
;           Macro def:
;           ----------
get_set_vect        MACRO n  ; citeste un vector de intrerupere
            mov     ax,35&n  ; si scrie in loc un nou vector
            int     21
            mov     cs:oldo&n,bx
            mov     cs:olds&n,es
            mov     ax,25&n
            mov     dx,offset newint&n
            int      21
 ENDM
;-------------------------

cseg        SEGMENT
            ASSUME          cs:cseg

            ORG     2c
env_seg label           word    ; adresa context program

            ORG     100
start:
            jmp     entry       ; punct de intrare in cod
;
;           Sectiune de date
;           ----------------------
oldvec1c    label   dword       ; adresa originala procedura timer
oldo1c      dw      ?
olds1c      dw      ?
oldvec13    label   dword       ; adresa originala procedura disc
oldo13      dw      ?
olds13      dw      ?
oldvec09    label   dword       ; adresa originala procedura keyboard
oldo09      dw      ?
olds09      dw      ?
tabela      db      0,0,1,1,1,1,0,0,0,0,0,1,1,0,0,1,1
            db      10 dup(0)
contor      db      0           ; daca < 6,are voie sa cinte
steag       db      1           ; comandat de la tatstatura:
                                ; steag = = 1,are voie sa cinte
                                ; steag = = 0,nu are voie
;-------------------------------------------
; Procedura apelata de 18.2 ori pe secunda ( INT 1Ch )
;
newint1c    PROC    far
            pushf
            call    cs:oldvec1c ; executa procedura originala
            cmp     byte ptr cs:steag,1 ; are voie sa cinte ?
            jne     short gata  ; nu
            inc     cs:contor   ; contor+ +
            cmp     byte ptr cs:contor,4    ; contor = = 6 ?
            jne     short gata ; nu,lasa sa cinte
            push    ax          ; da,opreste sunet...
            in      al,PORT_B
            and     al,0fch
            out     PORT_B,al
            pop     ax
            mov     byte ptr cs:contor,0    ; ...si reseteaza contor
gata:
            iret
newint1c    ENDP
;-------------------------------------------
; Procedura de tratare intrerupere disc (INT 13h)
;
newint13    PROC    far
            cmp     ax,0ff00    ; pentru autorecunoastere
            jne     short ok
            xchg    ah,al
            iret
ok:
            cmp     byte ptr cs:steag,1     ; are voie sa cinte ?
            jne     short gata2             ;nu
            push    ax
            push    bx
            lea     bx,tabela
            mov     al,ah       ; codul functiei pus in AL
            xlat    tabela
            cmp     al,1        ; AL = = 1 ? da,sa sune
            jne     short gata1             ; nu,sa nu sune
;
; Construieste frecventa sunetului dupa formula
; (empirica): (NrSector*16) OR ((NrCap+1)*128)
;
            push    cx
            push    dx
            and     cl,3f       ; mascheaza bitii de pista
            xor     ch,ch
            mov     bx,cx       ; NrSector in BX
            mov     cl,4
            shl     bx,cl       ; NrSector*16
            xor     dl,dl       ; DX va contine NrCap*256...
            inc     dh          ; ... +256
            shr     dx,1        ; .../2
            or      bx,dx       ; BX contine frecventa sunetului
;
; Cinta pe frecventa din AX
            mov     ax,34ddh    ; contor timp = 1234ddh
            mov     dx,0012
            div     bx          ; imparte (DX:AX) la BX
            mov     bx,ax
            in      al,PORT_B
            test    al,3        ; difuzor pornit ?
            jne     short go    ; da,continua
            or      al,3        ; nu,porneste difuzor
            out     PORT_B,al
```

```
        mov     al,0b6      ; selecteaza TIMER 2
        out     TIMER+3,al  ; scrie TIMER MODE REG
go:
        mov     al,bl
        out     TIMER+2,al          ; scrie LSB
        mov     al,bh
        out     TIMER+2,al          ; scrie MSB
        mov     byte ptr cs:contor,0  ; reseteaza contor
        pop     dx
        pop     cx
gata1:
        pop     bx
        pop     ax
gata2:
        jmp     cs:[oldvec13] ; executa procedura originala
newint13  ENDP
;-----------------------------------
; Procedura de tratare intrerupere keyboard
;
newint09  PROC far
        push    ax
        in      al,PORT_A
        cmp     al,46       ; a fost ScrollLock ?
        jne     short iese  ; nu,continua
        xor     byte ptr cs:steag,1   ; da,schimb steag
        in      al,PORT_B             ;opreste sunet
        and     al,0fc
        out     PORT_B,al
iese:
        pop     ax
        jmp     cs:[oldvec09]  ; executa procedura originala
newint09  ENDP
; ---------------------------------------
;       Zona de cod tranzient:
; ---------------------------------------
last_adr  label byte ; sfirsit cod rezident
mess    db      "Play installed,press ScrollLock to (com)mute."
        db      0d,0a,"$"
mess2   db      "Play already installed.",0d,0a,"$"
;
entry:
        mov     ax,0ff00    ; autorecunoastere
        int     13
        cmp     ax,0ff      ; deja instalat ?
        jne     short inst  ; nu,instaleaza
        mov     ah,09       ; da,gata
        lea     dx,mess2
        int     21
        mov     ax,4c00     ; termina
        int     21
inst:
        IRP     int_no,c,13,09
        get_set_vect        int_no
        ENDM

        lea     dx,cs:mess
        mov     ah,09
        int     21
        mov     es,cs:env_seg  ; elibereaza blocul de context
        mov     ah,49
        int     21
        mov     dx,offset cs:last_adr  ; marime cod rezident...
        mov     cl,4        ; ...transformata in paragrafe
        shr     dx,cl
        inc     dx          ; ... +1 pentru siguranta
        mov     ax,3100     ; termina rezident
        int     21
cseg    ENDS

        END     start
```

Program autoreproducător

Referitor la articolul "Cine ridică mănuşa?" apărut în "if" nr. 8/91, în care aţi tratat problema conceperii unui program autoreproducător, vă trimit un program scris în TURBO PASCAL 5.0 care afişează pe ecran în urma compilării şi execuţiei copia exactă a sursei sale.

Ideea programului este de a afişa fiecare rînd din partea de declaraţii a programului şi în acelaşi timp instrucţiunea prin care se realizează afişarea este concatenată la un şir ce va conţine în final toate aceste instrucţiuni, separate prin perechea de caractere CR/LF. Această dublă acţiune este realizată de procedura q. Ulterior, şirul S este tipărit prin procedura r, care afişează şi sfîrşitul programului.

Am folosit în locul caracterelor codurile ASCII ale acestora pentru a evita dublarea semnelor apostrof ce delimitează un şir de caractere.

mat. Cristian Nemeş

```
program p;
var S:string;
procedure q(T:string);
begin
      writeln(T);
      S:=S + #113#40#39 + T + #39#41#59#10#13
end;

procedure r;
begin
      write(S, #114#10#13#101#110#100#46)
end;

begin
      q('program p;');
      q('var S:string;');
      q('procedure q(T:string);');
      q('begin');
      q('writeln(T);');
      q('S:=S+#113#40#39+T+#39#41#59#10#13');
      q('end;');
      q('procedure r;');
      q('begin');
      q('write(S,#114#10#13#101#110#100#46)');
      q('end;');
      q('begin');
      r
end.
```

Meniuri verticale în PARADOX

Înainte de a vedea cum se rezolvă problema meniurilor verticale în sistemul de gestiune a bazelor de date PARADOX, trebuie să precizăm cîteva aspecte.

Astfel, în primul rînd trebuie subliniat faptul că acest S.G.B.D. încurajează foarte mult lucrul în propriul său "stil". Aceasta implică - de fapt - că programele trebuie să fie uşor de scris, scurte şi rapide. Bineînţeles că, folosind comenzile şi funcţiile PAL (Programming Application Language - limbajul de programare propriu PARADOX), se poate devia de la stilul impus; în acest caz trbuie să facem un compromis şi să acceptăm faptul că este nevoie de un cod mai complex care, implicit, poate să ruleze mai încet.

Creerea de meniuri verticale este un astfel de caz în care se deviază de la "stilul PARADOX". Interfaţa cu utilizatorul a PARADOX-ului este proiectată în jurul meniurilor orizontale. În prezent, interfeţele grafice cu utilizatorul (GUI = Grafic User Interface) utilizează meniurile verticale, modă ce a fost introdusă de lansarea pe piaţă a Windows-ului, şi ca exemple ar putea fi citate majoritatea IDE (Integrated Development Environment), majoritatea utilitarelor - PC Tools, Norton Commander, etc.

Meniurile orizontale se pot genera simplu în PARDOX utilizînd comanda SHOWMENU cu următoarea sintaxă:

```
Showmenu
        Optiunea1: Descriere1;
        .
        .
        OptiuneaN:DescriereaN
        default optiuneaX
to numevariabila
```

unde `Optiunea1, ..., OptiuneaN` sînt şiruri ce reprezintă opţiunile meniului care va apare pe prima linie a ecranului, iar `Descriere1, ..., DescriereaN` sînt explicaţiile corespunzătoare opţiunilor meniului ce apar pe a doua linie a ecranului în momentul în care "bara de rulare" (scrolling-bar) este poziţionată pe opţiunea corespunzătoare.

Programul prezentat în continuare a fost conceput în speranţa că îşi va găsi o utilitate pentru cititorii noştri utilizatori ai acestui S.G.B.D., dat fiind faptul că PARADOX-ul nu oferă nici o primitivă pentru generarea, respectiv utilizarea meniurilor verticale.

Descrierea programului

Programul conţine o procedură numită *Meniu_Vertical()*, proiectată pentru a fi utilizată ca o procedură standard. Ea dimensionează automat mărimea meniului pe ecran ţinînd cont de numărul de opţiuni pe care le-aţi specificat, opţiuni care pot fi în număr de cel mult 20. Script-ul continuă cu un mic program de aplicaţie care demonstrează modul în care funcţionează procedura. Rutina a fost scrisă folosind comenzile PAL de poziţionare a cursorului, de colorare a unor porţiuni pe ecran cu o anumită culoare, de scriere a unui text, etc. Dintre aceste comenzi amintim:

`@ x,y` - poziţionarea cursorului

`?? text` - afişarea unui text la poziţia curentă a cursorului

`Text - EndText` - afişarea unui text pe mai multe rînduri

`Paintcanvas` - schimbarea culorii pentru ceea ce este afişat pe ecran

`Style` - precizează culoarea ce se va utiliza pentru afişare

Procedura *Meniu_Vertical* este declarată cu şapte parametri formali ce controlează poziţia şi tipul chenarului precum şi culoarea meniului. Această procedură utilizează comanda `Paintcanvas`, una dintre cele mai puternice comenzi PAL. `Paintcanvas` permite afişarea textului şi modificarea culorii pe ecran, cu un control explicit al locaţiei, mărimii, culorii, umplerii şi chenarului zonei ce va fi desenată.

"Umbra" meniului se desenează prin simpla schimbare a culorii, astfel încît ceea ce exista înainte pe ecran nu va fi şters sau suprascris. Dar atunci cînd avem nevoie să desnăm "cutia" meniului, `Paintcanvas` ne permite să umplem aria cu spaţii (cod ASCII 32). Această manevră conduce efectiv la suprascrierea imaginii precedente, creînd o zonă opacă.

Opţiunea `Border` a comenzii `Paintcanvas` este folosită pentru desenarea chenarului în jurul meniului. PARADOX va utiliza caracterele din şirul opţiunii Fill aşa cum trebuie fără a mai fi nevoie să facă deplasarea cursorului în jurul meniului pentru desenarea chenarului şi "pictarea" fiecărui element al său în parte.

În prima parte a procedurii, prompt-ul (textul din partea superioară a ecranului) şi opţiunile meniului sînt desenate folosind simple scrieri pe ecran. A doua parte este constituită din ciclul de procesare a tastelor apăsate. Funcţia *getchar()* aşteaptă un caracter (apăsarea unei taste) de la utilizator şi îl memorează într-o variabilă.

Mişcarea barei de evidenţiere a opţiunilor este însoţită de modificarea unei variabile locale, *case.alegere*, care conţine opţiunea curentă a meniului (opţiunea scoasă în evidenţă în momentul respectiv). Înaintea cererii caracterului de la utilizator, se scoate în evidenţă opţiunea printr-o culoare specială utlizînd comanda Paintcanvas. După ce s-a tastat un caracter, se readuce la culoarea normală opţiunea curentă.

Tastările se procesează în mod tradiţional, verificînd fiecare tastă validă şi ajustînd bara de scoatere în evidenţă în mod corespunzător. Tastele valide pentru ieşire sînt `Esc` şi `Enter` .

Deoarece opţiunile meniului sînt conţinute într-un tablou, variabila case.alegere indexează tabloul; de asemenea ea conţine opţiunea selectată cînd utlizatorul apasă `Enter` . În exemplul nostru, odată ce opţiunea a fost selectată cu `Enter` , se afişează un mesaj ce arată opţiunea aleasă. Într-o aplicaţie reală, se vor apela diferite proceduri - în funcţie de opţiunea selectată din meniu.

Extinderea exemplului

Pentru cei interesaţi în extinderea exemplului de meniu vertical, prezentăm în continuare cîteva idei care merită să fie încercate:

- În funcţie de deplasarea barei între opţiuni, să se afişeze o descriere a acestora pe a doua linie a ecranului. (Utilizaţi un tablou suplimentar în care se memorează descrierile, indexarea lui făcîndu-se tot cu ajutorul variabilei de selecţie).
- Schimbaţi meniul într-o listă de opţiuni (oricît de multe) care să poată defila în fereastra de pe ecran:
  - Folosiţi o tabelă pentru a memora elementele listei. Declaraţi un tablou cu un număr de elemente egal cu numărul articolelor din tabelă şi citiţi elmentele în tablou la intrarea în procedură.
  - Cînd se apasă o tastă, decideţi dacă mişcarea implică schimbarea poziţiei barei sau dacă mişcarea implică şi defilarea listei de opţiuni. Implementaţi o procedură separată pentru fiecare tip de mişcare.
  - Urmăriţi elementul din listă aflat în partea de sus a ferestrei şi opţiunea curentă relativ la opţiunea din partea de sus. Folosind aceste variabile, opţiunea curentă este egală cu opţiunea de sus + opţiunea curentă - 1.

ing. Ioan Iacob

MENIU.SC

```
proc meniu_vertical (rind.sus,  ; rindul coltului stinga sus
      coloana.sus,   ; coloana coltului stinga sus
      lungime.max,   ; marimea celei mai lungi optiuni
      culoare.meniu, ; culoarea meniului si optiunilor
      culoare.optiune,; culoarea pentru evidentiere
      tip.chenar     ; afisarea chenarului cu linie dubla sau simpla
      )

private cite, x, care.alegere

cite = arraysize (alegeri)
care.alegere = 1
cursor off

paintcanvas attribute 7
      rind.sus + 1,coloana.sus + 2,rind.sus + cite + 2,coloana.sus +
      lungime.max + 5

paintcanvas fill " "
      attribute culoare.meniu
      rind.sus,coloana.sus,rind.sus + cite + 1,coloana.sus +
      lungime.max + 3

if tip.chenar = "s" then ;desenarea chenarului cu linie simpla
      paintcanvas border
      fill "" + fill ("",lungime.max + 2) + "" + fill ("",cite*2) +
      "" + fill ("",lungime.max + 2) + ""
      attribute culoare.meniu
      rind.sus,coloana.sus,rind.sus + cite + 1,coloana.sus +
      lungime.max + 3
endif
if tip.chenar = "d" then ;desenarea chenarului cu linie dubla
      paintcanvas border
          fill "" + fill ("",lungime.max + 2) + "" + fill ("",cite*2) +
          "" + fill ("",lungime.max + 2) + ""
          attribute culoare.meniu
          rind.sus,coloana.sus,rind.sus + cite + 1,coloana.sus +
          lungime.max + 3
endif

style attribute culoare.meniu

for x from 1 to cite  ;ciclu pentru scrierea optiunilor meniului pe ecran
      @ rind.sus + x, coloana.sus + 2  ?? alegeri[x]
endfor

while true     ;Ciclu de procesare a tastelor apasate de utilizator
      paintcanvas attribute culoare.optiune
      rind.sus + care.alegere,coloana.sus + 2,rind.sus + care.alegere,
      coloana.sus + lungime.max + 1
      ;Prin aceasta comanda se "picteaza" optiunea cu culoarea
      de evidentiere (diferitade culoarea meniului)
      retval = getchar ()  ;se citeste tasta apasata
      paintcanvas attribute culoare.meniu
          rind.sus + care.alegere,coloana.sus + 2,
          rind.sus + care.alegere,coloana.sus + lungime.max + 1
      ;Prin aceasta comanda se readuce optiunea anterioara
1     a culoarea meniului

      switch
          case retval = -72 :    ; UP
              if care.alegere = 1 then
                  care.alegere = cite
              else
                  care.alegere = care.alegere - 1
              endif
          case retval = -80 :      ; DOWN
              if care.alegere = cite then
                  care.alegere = 1
              else
                  care.alegere = care.alegere + 1
              endif
          case retval = -75 : return          ; LEFT
          case retval = -77 : return          ; RIGHT
          case retval = -71 :                 ; HOME
              care.alegere = 1
          case retval = -79 :                 ; END
              care.alegere = cite
```

```
        case retval = 27 :    ; ESC
            ;Daca se apasa ESC programul stocheaza in variabila
            ;de sistem retval un numar egal cu numarul optiunilor
            ;meniului plus unu
            return cite+1
        case retval = 13 :        ; ENTER
    ;Daca se apasa ENTER programul intoarce numarul optiunii
    ;curente din meniu
            return care.alegere
        otherwise     :
            beep
    endswitch
endwhile
endproc

;Program pentru demonstrarea modului de lucru al procedurii
clear
array alegeri[3]
alegeri[1]="Introducere date"
alegeri[2]="Actualizare date"
alegeri[3]="Consultare"
while true
meniu_vertical(3,30,16,78,30,"d")
switch
    case retval = 1 :
        message "Ati ales optiunea ",retval
    case retval = 2 :
        message "Ati ales optiunea ",retval
    case retval = 3 :
        message "Ati ales optiunea ",retval
    case retval = 4 :
        message "Ati apasat ESC" sleep 1500 quitloop
endswitch
sleep 1500
endwhile
```

Mai multe comenzi într-o linie DOS

Vă propunem un scurt program care permite să se dea mai multe comenzi într-o singură linie de comandă MS-DOS.

Programul se numeşte "E". Pentru executarea lui se tastează E urmat de comenzi. Comenzile se termină cu caracterul "#" şi sînt separate prin spaţii. Caracterele de redirectare şi "piping" se vor substitui astfel:

! în loc de |
{ în lloc de <
} în loc de >

Programul a fost testat sub MS-DOS 5.0 cu Turbo C Versiunea 2.0.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include .h

void stringcat( char x[],char y[] ), stringchg( char x[],char c,char d );
char *delblank( char *x );

void stringcat( char x[],char y[] )
{
 int i,j;

 i = j = 0;
 while( x[i] != '\0' )
       i++;
 if( i >= 255 )
       return;
 while( (x[i++] = y[j++]) != '\0' )
       ;
}


void stringchg( char x[],char c, char d )
{
 int i;

 i = 0;
 while( x[i] != '\0' )
       {
       if( x[i] == c )
            x[i] = d;
       i++;
       }
}


char *delblank( char *x )
{
 char *ptr;

 if( x[0] != ' ' )
       return( x );
 do
       {
            ptr = strpbrk( x," ");
            if( ptr != NULL )
                  x++;
            } while( (ptr != NULL) && (x[0] == ' ') );
 return( x );
}


void main( int argc,char *argv[] )
{
 char *ptr, *tok;
 char string[255];
 int  i, numbargs;

 string[0] = '\0';
 numbargs = argc - 1;
 for( i = 1; i < numbargs; i++ )
       {
       stringcat( string,argv[i] );
       if( i  numbargs )
            stringcat( string," " );
       }
 stringchg( string,'!','|' );
 stringchg( string,'{','<' );
 stringchg( string,'}','>' );
 ptr = strpbrk( string,"#" );
```

```
if( ptr != NULL )
      tok = "#";
else
      tok = ";";
ptr = strtok( string,tok );
printf( "\n" );
puts( ptr );
if( system( ptr ) != 0 )
      perror( strerror );
while( ptr != NULL )
      {
      ptr = strtok( NULL,tok );
      if( ptr = = NULL )
          break;
       printf( "\n" );
       ptr = delblank( ptr );
       puts( ptr );
      i = system( ptr );
      }
}
```

Mulţime Mandelbrot

Iată o nouă variantă de program de desenare a unei mulţimi Mandelbrot.

Programul QMANDEL.ASM este scris pentru PC-uri pe 32 de biţi (80386 sau 80486) cu interfaţă grafică VGA.

Se asamblează cu TASM şi se link-editează cu TLINK dîndu-se comanda:

TLINK /t qmandel.obj

pentru a se obţine un fişier de tip .COM.

Pe un calculator cu 80386 SX la 16MHz programul a desenat un ecran complet în circa 62 secunde.

QMANDEL.ASM

```
; Program: qmandel.asm
; Functie: desenare pe PC386 cu VGA
; Limbaj: TASM, TLINK /t

cseg   SEGMENT PARA 'CODE'
       ASSUME CS:cseg, DS:cseg, SS:cseg, ES:cseg
       ORG 100h
start  PROC FAR
       .386
       mov   ax,0012h
       int   10h
       mov   esi,xl
_13:
       mov   edi,yo
_11:
       mov   eax,esi
       mov   ebx,edi
       mov   bp,iter
_12:
       dec   bp
       je    SHORT nopt
       mov   ecx,eax
       imul  ebx
       xchg  eax,ecx
       imul  eax
       xchg  eax,ebx
       imul  eax
       mov   edx,ebx
       add   edx,eax
       cmp   edx,max
       jg    SHORT pixel
       sub   ebx,eax
       mov   eax,ecx
       sar   eax,7
       add   eax,edi
       sar   ebx,8
       add   ebx,esi
       xchg  eax,ebx
       jmp   SHORT _12
pixel:
       mov   cx,si
       mov   dx,WORD PTR cs:yo
       sub   cx,WORD PTR cs:xl
       sub   dx,di
       xor   bh,bh
       mov   ax,bp
       and   al,0fh
       mov   ah,0ch
       int   10h
nopt:
       dec   edi
       cmp   edi,yu
       jne   _11
       inc   esi
       cmp   esi,xr
       jne   _13
       mov   ah,8
       int   21h
       mov   ax,0003h
       int   10h
       mov   ax,4c00h
       int   21h
start  ENDP

xl     DD    -480
xr     DD    160
yo     DD    240
yu     DD    -240
max    DD    500000
iter   DW    32

cseg   ENDS

       END   start
```

Cod	Nume pachet	Nr. dischete	Format	Capacitate	Limba
Sisteme de operare					
29	DR - DOS 6.0	1	3,5"	1,4M	E
Proiectare circuite electronice					
41	Addi-Data: Signal Tools	1	5,25"	1,2M	E
42	MicroSim Corporation: Design Center Demo	1	5,25"	1,2M	E
43	ORCAD	1	5,25"	1,2M	E
44	ORCAD: PC Board Layout Tools	1	5,25"	360K	E
Aplicaţii					
12	CAPP	1	5,25"	1,2M	D
2	Buch,Trio	2	5,25"	360K	D
39	BALPRO - balanţă contabilă	1	5,25"	360K	R
40	UNO - completare formulare	1	5,25"	360	R
45	Hamor Soft: hMISS - secretariat, hCONT - contabilitate, hGEN&hUTIL - dezv. aplicaţii	1	5,25"	360	R
Utilitare, biblioteci					
14	Borland Windows Tools	1	5,25"	360K	D
17	KAO BTX-Dekoder + Demo	1	5,25"	360K	D
36	QuickStep Tools	2	5,25"	1,2M	D
6	Spindrift Libraries Demo	1	5,25"	360K	E
7	BKS Demo	1	5,25"	360K	D
47	Verificator - Comprex S.R.L.	1	5,25"	1,2M	R
Baze de date					
10	Paradox 3.0	1	5,25"	360K	D
46	Personal Database Demo - AvantGarde Soft.	1	5,25"	1,2M	R
31	Foxpro 2.0	2	3,5"	720K	D
18	SOS - Software Tools Datenbank	1	5,25"	360K	D
24	Advanced Revelation	1	3,5"	720K	E
25	Rbase 3.1	1	3,5"	720K	D
32	WindowBase	1	5,25"	1,2M	D
33	Address One	1	5,25"	360K	D
37	Superbase 4	2	3,5"	720K	D
Grafică, DTP, CAD					
11	Time Works Publisher	2	5,25"	360K	D
15	MEGA CAD	1	5,25"	360K	D
27	Aldus Page Maker 4.0	1	3,5"	720K	E
30	Draw Perfect 1.1 si Office 3.0	1	3,5"	1,4M	D
9	Micrografx	1	5,25"	360K	D
Cursuri					
5	DOS - Curs IBM	1	5,25"	360K	D
Procesoare de texte, OCR					
28	Word Perfect for Windows	1	3,5"	1,4M	D
35	Scout OCR	1	5,25"	360K	D
40	Ekta Editor V1.5	1	5,25"	360K	E
Măsură şi control					
3	Labtech Notebook Demo	1	5,25"	1,2M	E
4	Labtech Control	1	5,25"	1,2M	E
34	Scope	1	5,25"	360K	D
Jocuri, diverse					
8	Pearl Agency	1	5,25"	360K	D

Un nou serviciu pentru cititorii noştri:

SHARE - if

Pentru o mai bună circulaţie a informaţiei între ofertanţi şi solicitanţi de programe (şi nu numai ...), vă oferim posibilitatea să:

- economisiţi timpul necesar introducerii programelor mai lungi, al căror listing a apărut în "**if**".
- preluaţi pe dischetă, pentru eventuale multiplicări, textul articolele din "**if**" pe care aţi vrea să le aveţi disponibile şi în această formă
- preluaţi acele programe şi texte de articole care ne-au fost trimise spre publicare, dar pe care, din variate motive (cel mai adesea, lipsa de spaţiu) - nu ajungem să le publicăm. (Desigur, aceste materiale le difuzăm numai atunci cînd există acordul autorului - încă nu am găsit o variantă de recompensare a autorului pentru această variantă de difuzare) O listă incompletă: criptare fişiere, completare formulare, o parolă solidă, nucleu de multitasking, "catalog" de dischete, etc.
- faceţi cunoştinţă cu acele programe din care ni s-au trimis demo-uri, respectiv **să difuzaţi programele dumneavoastră demonstrative** prin intermedierea noastră. Acest din urmă punct sperăm să intereseze în mod deosebit pe toţi autorii noştri de programe, cărora le este destul de greu să-şi facă publicitatea de care au, evident, nevoie.

Cei dornici să includem demo-urile lor în oferta noastră sînt rugaţi să achite, odată cu expedierea dischetei sursă, suma de 500 lei în contul nostru: Cont nr. 40729-96010402, Banca Română pt. Dezvoltare Mureş.

(Cei care trimit materiale informativ/didactice, şi nu publicitare, nu au de achitat nimic.) Preţul pe care îl practicăm pentru o dischetă este, din păcate, greu de definit în avans, din motive cunoscute de toată lumea. De aceea vă rugăm să vă interesaţi - telefonic sau în scris - în momentul lansării comenzii. În principiu, preţurile sînt identice cu preţurile cu care, la rîndul nostru, am achiziţionat dischetele, la care se adaugă 50 lei pentru munca noastră şi, suplimentar, taxa de expediere. Desigur, cei care ne trimit dischetele lor, au de achitat doar taxa de copiere şi expediere, iar cei care trec sau trimit pe cineva pe la redacţie, vor avea de achitat doar cei 50 lei, taxa de copiere.

Vă rugăm ca pe comanda pe care ne-o trimiteţi să specificaţi pe îngă numele pachetului şi capacitatea şi formatul dischetelor pe care doriţi să vă furnizăm produsul.

Pentru fanii Fox

În urma apariţiei reclamei pentru produsul Fox Pro 2.0 au sosit pe adresa redacţiei noastre cîteva cereri pentru dischete demonstrative.

Îi rugăm pe toţi cei interesaţi să trimită talonul respectiv pe adresa:

DARIAN ROM SUISSE SRL

3400 CLUJ-NAPOCA

str. Observatorului nr 146/15

tel: 95-123611

fax: 95-124567

Noi am transmis taloanele primite destinatarului de drept.

În curînd va apare

"if" nr. 5 / 1992

Important

În curînd va apare cartea

Programarea în limbajul Pascal

editată de Micro ATCI Tîrgu Mureş

Manualul este o traducere din limba germană şi este destinat atît celor care vin în contact pentru prima oară cu acest limbaj de programare, cît şi celor care doresc să se perfecţioneze învăţînd noi metode şi tehnici de programare. Volumul este bogat în exemple, toate programele fiind testate cu compilatorul Turbo Pascal 6.0 al firmei BORLAND International.

Tirajul fiind limitat, asiguraţi-vă că nu veţi pierde lucrarea expediind o comandă fermă pe adresa:

Micro ATCI, C.P. 64, O.P. 1
4300 Tîrgu Mureş

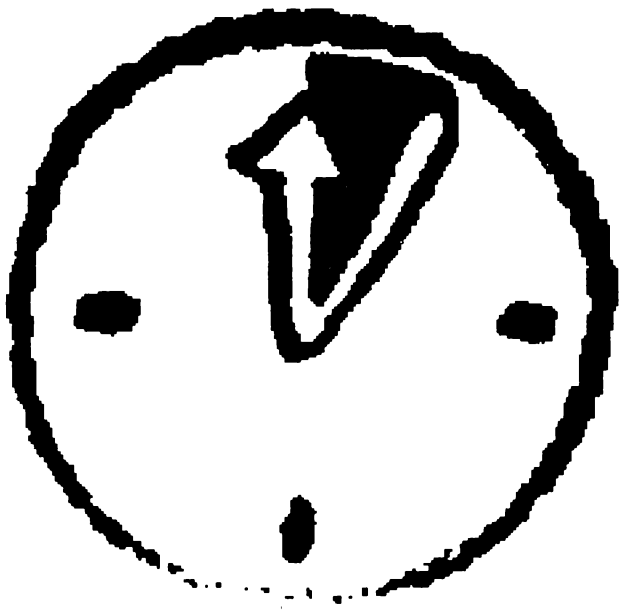
Talon de comandă-abonament

Subsemnat(a/ul).....
doresc să-mi trimiteţi |_____| exemplare din fiecare număr al revistei **If**,
începînd cu numărul |_____| , pe adresa:
.....
.....
.....
Telefon: acasă ..... la servici. .....
Pentru aceasta am expediat prin mandat poştal/virament/cec .....lei în contul nr. 40729-96010402 deschis la Banca Română pt. Dezvoltare Mureş. Abonamentul funcţionează pînă la epuizarea sumei; în cazul că banii pe care i-am trimis nu mai ajung pentru un alt număr, voi fi anunţat de acest lucru în timp util, astfel încît nu risc să pierd nici un număr. Dacă mă răzgîndesc şi renunţ la abonament, mi se returnează toată suma rămasă.

If 4 / 92 **Data:** .....

Semnătura .....

Five minutes and you're in business



În viaţă, majoritatea
lucrurilor sînt dificile la
început,
iar primele cinci minute sînt
cele mai grele.
De aceea, IBM a realizat un
calculator cu care vă puteţi
înţelege din primele cinci
minute.

IBM Personal System PRO

În cinci minute, "sînteţi în
business".
În cîteva zile îl stăpîniţi.
După o săptămînă, vă gîndiţi
deja la viitor.

The new IBM PS PRO 386 SX

386 SX, 20 MHz, AT BUS, 2 MB RAM, 80 MB HDD, 1,44 MB FDD,
monitor VGA
color 14", mouse. Oferta standard include şi pachetele de programe:
DOS 5.0, Windows 3.0, Works 2.0.
Modelul PS PRO 386 SX poate fi obţinut cu uşurinţă, la un preţ foarte
accesibil de la

Micro ATCI Tîrgu Mureş

IBM authorized remarketer

str. Gheorghe Doja nr.36

Tel / Fax: 954 - 31660