

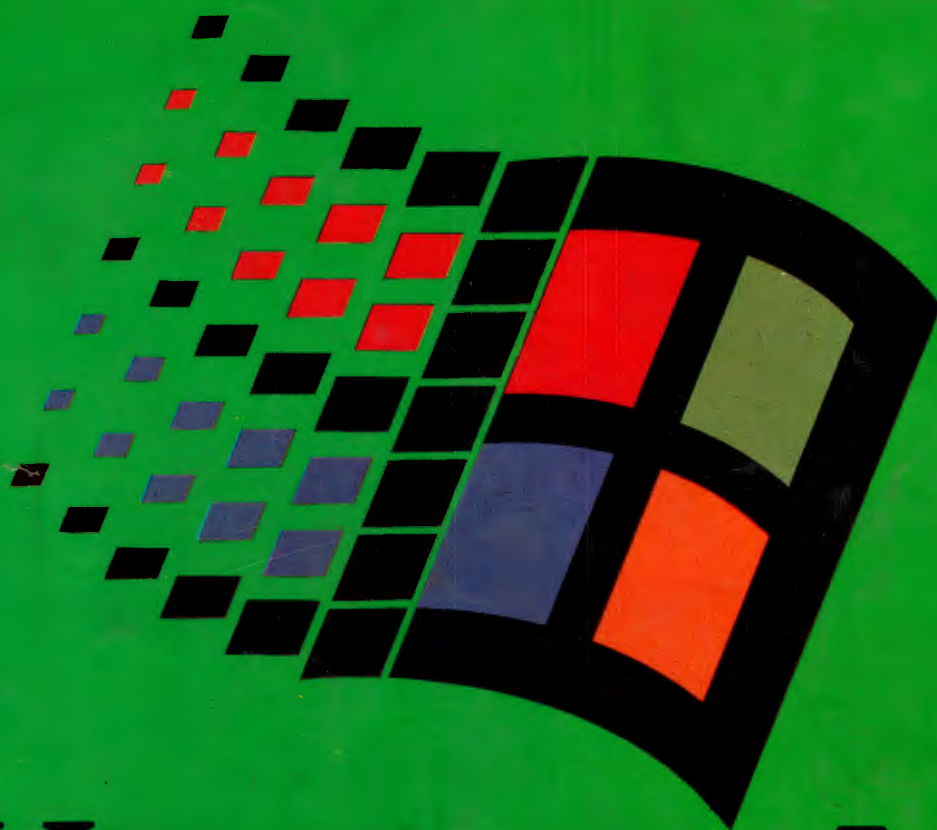
实用第一,智慧密集 Practicability First, Intelligence Intensive

创刊号

# 电脑编程技巧与维护

COMPUTER PROGRAMMING SKILLS & MAINTENANCE

1994 年 7 月



Microsoft **WINDOWS NT**™

**Microsoft®**



# 中国电脑有希望



北京希望电脑公司成立于1985年1月,隶属于中国科学院,是经济体制改革以来中国大陆新兴的高技术企业之一,采用市场为主导的运行机制,并向股份有限公司过渡。

公司总部位于北京中关村,是“北京新技术产业开发试验区”首批认定的“高技术企业”,享有各项优惠政策。

公司现有职工160余人,工程技术人员占68%,高级技术人员占11.5%,总部除管理部门外,设有CAD部、SGI部,系统技术部,电源产品部、软件部和技术资料部等业务部门;公司在上海、南京、广州、成都设有独资子公司,分别负责华东、华南和西南的市场推广工作;在香港设有合资子公司,负责公司的产品开发和国际推广工作。公司各业务部门和子公司在全国设有100多个分销点,长期客户超过1000家,构成一个有效的全国销售网。

北京希望电脑公司的经营规模在八年间稳步发展,自1988年以来,希望公司一直在“北京新技术产业开发试验区”的排名表中列十名之内。人均历年平均产值超过50万,在全国同类企业中也名列前茅,是一个在全国享有知名度的高技术企业。

☆ 汉化微机CAD工作站、SGI工作站。

☆ “宝合”UPS电源。

☆ HOPE-126无线寻呼中心系统。

☆ UC DOS、dBASE IV 2.0中文版。

AUTO CAD 汉字环境等各种软件。

☆ 计算机技术资料。

地址:北京海淀路82号

电话:2567387、2562329、2565884、

2567819、2579873

信箱:北京8721信箱

邮政编码:100080

电挂: 0755 传真: 2561057

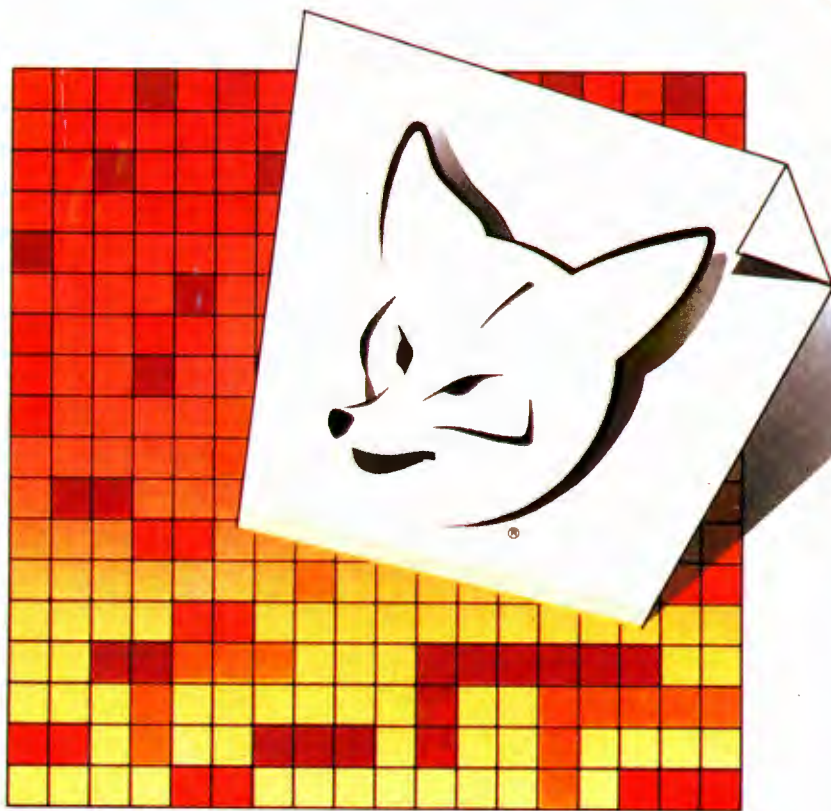


北京希望

电脑公司



Unlock  
the power of  
Microsoft FoxPro  
for MS-DOS and  
Windows.



# Microsoft FOXPRO LIBRARY CONSTRUCTION KIT

*For MS-DOS and Windows™*



# 希望汉字系统

汉字系统的希望 UC DOS 3.0

为什么我们总是人们追赶的目标？

※ 支持直接写屏,英文制表符自动识别

西文软件无需汉化即可支持汉字,采用独一无二的英文制表符识别技术,充分保持原版西文软件的神奇风采

※ 国内唯一真正实现零内存的汉字系统

386 以上微机,只要有一定的扩充内存,系统在启动时就可自动将所有程序和数据放入扩充内存,不占用任何 DOS 基本内存,不受 DOS 版本限制;对 286 或没有扩充内存的微机,可利用独家提供的“(虚拟内存管理器 VMS)”为应用程序提供最大的基本内存

※ 经香港金山公司授权,系统配备 WPS 进行文字处理

UCDOS 3.0 提供的 WPS 可同时使用 26 种高精度矢量字库进行模拟显示、打印输出,并在网络环境中首次真正实现共享打印,模拟显示和打印速度可提高 2-3 倍

※ 真正实现网络共享

网络版 UC DOS 3.0 无工作站(包括无盘工作站)数目限制;彻底解决网络中远程终端间的通讯问题;显示字库可存放于服务器上,为各站点保留更多的低端内存

※ 完善的汉字输入方法

系统提供全拼、简拼、双拼、五笔、普通和电报码等多种汉字输入方法,各种输入方法均带有大量词组;独创“记忆词组”,成功地解决了局部词组和专业性词汇输入困难的问题;支持屏幕取词功能,各输入法共享自定义词组

※ 强大的打印功能

国内唯一将点阵字库和矢量字库(26 种)有机结合的汉字系统,保证了低点阵汉字的质量;支持针打、喷打和激打,在任何软件中均可直接打印 2048×2048 点阵以内的任意点阵汉字;独特的打印字库还原技术,还原速度可与硬件媲美,打印速度超过硬字库

※ 特殊显示功能

可在屏幕上显示任何点阵的汉字,大小仅受屏幕尺寸限制;支持点、线、圆、椭圆、扇形、矩形及图形填充等多种作图功能;利用控制字符可实现对简谱文件的后台演奏;提供图像动态保存,所有特显功能均可用于各种图形模式并可在 FoxPro、Borland 系列等直接写屏型软件中使用

※ 彻底支持 DOS5.0、DOS6.0 和 DRDOS6.0

※ 彻底支持鼠标功能

※ 本系统以纯软件方式提供,是便携机用户的最佳选择

敬告用户:

1. UC DOS 及 PT DOS 老用户可凭用户卡进行优惠升级,单用户版 300 元/套,网络版 600 元/套
2. UC DOS 3.0 统一零售价单用户版 880 元/套,网络版 2000 元/套
3. 欢迎广大用户向我公司及各地经销单位索取 UC DOS 3.0 简版,15 元/套

## 北京希望电脑公司软件部

地址:北京 海淀路 82 号

信箱:北京 8721 信箱

邮码:100080

电话:2579873、8422024、  
8422025

传真:2561057

## 上海希望电脑公司

地址:(200052)上海新华路 416 号

电话(传真):(021)2521656、  
2569929

## 成都希望电脑公司

地址:(610015)成都市新南路  
四维村街 6 号(磨子桥附近)

电话(传真):(028)5589787、  
5556333

## 广州希望电脑技术公司

地址:(510620)广州天河体  
育西路育蕾三街二号

电话:(020)7505151、7505152、  
7505153  
传真:(020)7500275

## 南京希望电脑技术公司

地址:(210005)南京市中山南  
路 105 号

电话:(025)4410794、4410549  
传真:(025)4410548



# 祝贺《电脑编程技巧与维护》杂志创刊

《电脑编程技巧与维护》月刊是当今计算机及其应用技术大力发展时代新面世的实用第一,智慧密集的新型电脑专业刊物。它以内容新颖、实用性强和技术先进为宗旨,力求为广大电脑工作者提供一个学习、交流国内外电脑编程技术、技巧及电脑维护技能的新天地。

从其特色鲜明的刊名《电脑编程技巧与维护》即可看出它所蕴含的价值。毋庸置疑,软件是电脑的灵魂,没有软件,特别是没有好的编程语言、系统软件和应用软件,即使再好的电脑也不能充分发挥其所有效能;同时没有实效良好的保障措施——软硬件维护技术,也不能保障电脑的正常运行。

《电脑编程技巧与维护》月刊,正是瞄准了加快发展我国电脑事业的两大关键技术——编程技术、技巧与维护保障技术,以最经济、最实惠的期刊与软盘作为专业知识、技术的传播媒介,内容集中了国内外广大从事和学习电脑编程、操作与维护人员的智慧和精技,以期共同提高,促进我国的电脑开发技术和应用水平,尤其是编程技术、技巧与维护技术的提高和升华,创造出优秀的电脑软硬件,使我国电脑开发技术与应用水平进一步地提高,增强我国计算机应用与开发人员的综合能力,从而,加速我国经济建设的发展。

我国现有电脑从业人员数百万,大家在各自不同的岗位上从事着电脑系统的开发和维修保障工作,为我国电脑事业的发展做出了积极的贡献。作为一名老一代电脑工作者,也是创办本刊的发起人之一,深知未来美好的世界是年青人的,开创电脑事业的新境界永远靠的是年青人。我真诚地希望广大电脑工作者和编辑们共同精心耕耘好《电脑编程技巧与维护》月刊这块泛域沃土,使其成为我们汲取电脑技术和展现自己聪明才智的理想园地,成为大家的益友,受到广大读者的欢迎。

本刊与国外著名软硬件公司如 Microsoft 等有着广泛的联系,为支持本刊的工作,它们已经并将继续提供世界最先进的技术信息(这些信息均是由这些公司授权,无任何版权争议)。我们将陆续地把这些最新的前沿技术快速地报道给本刊的广大读者。更希望国际同仁友好不吝积极赐稿,以求共同提高。

最后,受编辑部之托,对本刊创办期间给予大力帮助的领导、同仁友好,特别是给予资助的北京希望电脑公司,致以崇高的敬意和衷心地感谢!

郭平欣

编者  
导读

这是一个崭新的开始,对于您,也对于我们。作为编者,我们希望您能喜爱它,希望它能成为您的好伙伴……。

那么,怎样才能使它充分发挥“才华”,为您做出最大的贡献呢?让我们来给您介绍几个小窍门吧。

提起窍门,那可真是本刊的一大特色了。到目前为止,国内的各种计算机类报刊真是纷纷扬扬、各领风骚了,可要说集“窍门”于一身,专营“窍门”的却寥寥无几。所以,知道了本刊的这一特色,您也就基本上会利用本刊了。

再有,本刊为了把这些“窍门”完全彻底地贯彻到读者那里,还实行了一个别出心裁、行之有效的措施,即同时提供实用程序软盘,而这也就成了本刊的另一大特色。您要是能掌握这一关键信息,那就达到充分利用本刊的目的了。为什么这么说呢,因为“窍门”的精华即在于实用程序,而程序的利用真是既繁琐(录入困难),又容易出错(录入时难免出错而导致调试不通),而我们所提供的实用程序软盘则都是经过实际运行通过的,既有源程序,又有编译后的可执行程序;使您既具有自己进行再加工的正确基础,又能够拿来即用,可谓鱼和熊掌兼而得之。

另外,我们还特别为您设置了服务卡及意见咨询卡,请您随时提出要求,以便本刊越办越好,您所能得到的东西也会越来越多。这样,您就达到了使本刊充分发挥潜能,与本刊共呼吸的最高境界了。

## 实用第一,智慧密集

## 电脑编程技巧与维护月刊

## (创刊号)

## 软平台

- COMMAND.COM 模拟程序 SHELL ..... (5)
- DIR 命令的妙用 ..... (7)
- 巧用 DOS 的 I/O 重定向功能 10 例 ..... (9)
- 高密软盘快速复制 ..... (10)

## 图形图象处理

- 彩色图象的抖动打印 ..... (11)
- 设置各种 VGA 卡高分辨率图形模式的 BIOS 接口 ..... (13)
- 一种实用的 EGA/VGA 屏幕放大技术 ..... (16)
- 用 C++ 实现交互式实时动画技术 ..... (18)
- XENIX 下的图形软件包 CGI ..... (22)

## 汉字处理

- 西文环境下图形与汉字同屏显示的方法 ..... (25)

## 编程语言

- 完善的监视器类型测试程序 ..... (26)
- 用 Turbo C 实现 SHELL 功能 ..... (29)
- 混合语言编程时慎用 OFFSET ..... (30)
- Borland C++ 与汇编语言的接口技术 ..... (31)
- 汇编语言的递归调用和过程自调用 ..... (34)

## 数据库

- 巧用 FoxPro2.5 屏幕生成器 ..... (35)
- 用 Browse 使编程量减少一半 ..... (36)

**Practice First, Intelligence Intensive**

# Computer Program Skills & Maintenance

用 FBASE 扩充 FoxPro 的图文声象功能 .....	(38)
FoxBASE+通用立体投影下拉式菜单程序设计 .....	(40)

## 计算机安全

预测未来病毒 .....	(44)
CPAV 免疫功能剖析 .....	(48)
银行系统中全信息记录操作痕迹 .....	(50)
一种新的程序跟踪方法 .....	(52)
Gene 病毒的检测与清除 .....	(54)

## 软件维护

AutoCAD 12 版的问题及其解决方法 .....	(55)
给 DOS 增加一个分页命令 .....	(59)
数据备份盘的随机恢复技术 .....	(60)
修改 FORMAT 使得无系统启动时的提示更合理 .....	(51)

## 实用软件

Microsoft MS-DOS 6.2 简体中文版 .....	(64)
FoxPro 的过去、现在和未来 .....	(67)

## 电脑博士信箱

DOS 与 Windows .....	(68)
---------------------	------

## 新产品

一种新型的网络系统向宇对等网系统 .....	(70)
华光 VI 型电子出版系统推出 .....	(58)

顾问	郭平欣 刘洪昆
名誉社长	周明陶
社长	毕研元
副社长	袁保佑
总编	秦人华
执行总编	秦长久
副总编	王路敬 孙毅
编辑	开元 程钧 易言 艾蒙 田艾 肖贻 苏古 阮薇 温达
公关部主任	吕莉 伊梦
出版部主任	祁连顺
发行部主任	孙茹萍
编辑出版	《电脑编程技巧与维护》杂志社
地址	北京市劳动人民文化宫内
邮编	100006
编辑部电话	5123823
公关部电话	5232502
照排	《电脑编程技巧与维护》杂志社 电脑排版部
印刷	北京市地质矿产局印刷厂
订阅处	全国各地邮局
国内总发行	山东省潍坊市邮电局
邮发代号	24-106
刊号	CN11-3411/TP
每期定价	3.80 元
出版日期	7 月 8 日

**本刊全部内容的版权,归本社所有,未经同意,不得转载。**

# 欢迎您

## 成为本刊的撰稿人

**作**为一个电脑编程、应用与维护技术人员,无不想使自己的聪明才智得以充分发挥,那么,《电脑编程技巧与维护》杂志正好是您的一个机会。想想您的经验所得,再看看它的栏目设置,不要再犹豫,把您的心血贡献出来吧,它必会给您带来满意的回报。

《电脑编程技巧与维护》杂志是一个专门为从事电脑编程和系统应用与维护人员创办的专业性和实用性都很强的专业技术杂志,它的主要栏目有:

**软平台** 该栏目以我国应用面广、发展潜力大的操作系统如 DOS、Windows、PWIN、Unix 以及大型的应用系统,各种建立于 DOS 及 Windows 基础上的汉字系统如 UC DOS、金山 DOS、2.13 等为基础,介绍这些系统的新功能、新特性,以及利用这些功能和特性进行开发、应用的技术和技巧。

**图形图象处理** 该栏目主要介绍图形图象软件的编制技术、技巧和编程的具体方法。它包括软件界面的开发、鼠标器的驱动、各种图形图象格式的介绍及应用等。

**汉字处理** 该栏目主要介绍汉字处理的编程实用技术,它包括:汉字的直接写屏、各种字库的组织结构,以及汉字的放大、平滑、旋转等多方面的内容。

**编程语言** 该栏目主要介绍 C、C++、汇编以及 BASIC 等编程语言的编程技术、技巧以及它们所

提供的函数的灵活应用技术,该栏目的文章将主要以实用程序说明应用的方法、技术与技巧。

**数据库** 该栏目主要介绍 FoxBASE、dBASE、FoxPro 以及 ORACLE 等数据库的新功能、新用法,以及利用这些数据库系统进行开发工作的经验、编程技术、技巧和编程方法。

**计算机安全** 该栏目主要介绍加密技术、反跟踪技术、反病毒技术,以及具体病毒的检测和消除技术等。

**网络与通讯** 该栏目主要介绍网络的应用技术,包括网络操作系统和在网络环境下的应用开发技术、技巧。

**软件维护** 该栏目主要介绍软

件功能的用户扩充或扩展,及软件故障的具体现象和原因,并给出合理的恢复方法。

**实用软件** 该栏目主要介绍如系统维护工具、压缩工具、调试工具(WORD、PCTOOLS、NORTON、DUP、PKZIP、ARJ 等)的应用技术与技巧。

**硬件维护** 该栏目介绍计算机硬件系统、部件的维护与维修经验,提供用户自身修复硬件系统故障的实例。

在以上这些方面有经验所得的同仁请不吝赐稿。稿件一旦录用即寄样刊及稿酬,本刊每年都将举办优秀撰稿人评选活动,评选出的优胜者,本刊将免费赠送下一年的《电脑编程技巧与维护》杂志。

### 投 稿 须 知

来稿请寄(100006)北京市劳动人民文化宫内《电脑编程技巧与维护》编辑部。稿件请用方格稿纸书写整齐或用打印机清楚地打印出来,所附程序也请打印清楚。作者姓名、单位以及通信地址、邮编请用整齐的字迹写在醒目的位置。来稿必须将有关稿件的程序完整地附上,以便读者能直接应用。本刊鼓励软盘投稿(同时也请附上一份打印稿),磁盘投寄的稿件一旦录用,稿费将加倍考虑磁盘的成本费和邮费。稿件不能一稿两投,本刊在收到稿件后两个月内发录用通知,在此之后没收到录用通知的作者,稿件可另行处理。



## 栏目编辑的话

本栏目以目前我国应用面广、发展潜力大的操作系统如 DOS、Windows、PWIN、Unix 以及大型的应用系统,各种建立在 DOS 及 Windows 基础之上的汉字系统如 UC DOS、Super DOS、CCDOS、2.13 等为基础,介绍这些系统的新功能、新特性,以及利用这些功能和特性进行开发、应用的技术和技巧。随着软平台功能的日臻全面与完善,用户要将这些系统的潜力及其所提供的实用编程方法与技术充分利用起来,需要精心地研究和长期的工作。软平台这一栏目将国内乃至国外计算机应用界人士的经验与精华汇集起来,使读者能通过这一栏目充分了解软平台发展的最新技术,从而在软平台上的开发应用工作更加得心应手。

在此创刊号里,我们选用了 COMMAND.COM 模拟程序 SHELL、DIR 命令的妙用等四篇文章。

尊敬的广大读者朋友们:我们诚恳地期望着大家将您最真诚、最宝贵的意见、指正和建议填写于征询卡里,或写信给我们。谢谢!对几位作者致以敬意和感谢。欢迎赐稿。

# COMMAND.COM 模拟程序 SHELL

□王 辰



所周知,COMMAND.COM 是 DOS 的命令解释程序,虽然它是系统程序的一部分,但它又具有普通程序的特点,它的功能大致可分为三部分:(1)从键盘接收命令;(2)分析命令;(3)执行命令。

只要了解了 COMMAND.COM 的功能,我们完全可以写一个新命令解释程序来替换它以满足我们的特殊需要。

为了帮助读者消除对 COMMAND.COM 的神秘感,笔者编写了一个小程序 SHELL.ASM,该程序在编译、连接后,通过 EXE2BIN 转换成 .COM 文件便可以替换

COMMAND.COM 进行工作。本程序同样将其所执行的命令分作内部命令和外部命令两部分。

要使用本程序,只要在 CONFIG.SYS 中增加以下命令即可:

SHELL=SHELL.COM

这样在系统启动后,就出现本程序的提示符 SHELL>,这时,就可执行 SHELL 的内部及外部命令。因为本程序只是一个示范程序,因此其内部命令只有 CLS 和 EXIT 两种,而且必须是大写。

下面是 SHELL.ASM 的源程序:

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
ORG 002CH
ENVIRONMENT LABEL WORD
ORG 0080H
PARAMETER LABEL BYTE
ORG 100H
BEGIN:
JMP INITIZER
START:
GO _LP:
CALL PROMPT ;显示提示符
CALL GET _CMD ;接收命令
LEA DX,CRLF
CALL DISP _MESS
CALL INNER ;检查是否为内部命令,是则执行之
JNC GO _LP
CALL SEARCH _CMD ;查找是否外部命令
JNC EXEC _IT
LEA DX,BAD _CMD ;失败时,显示"Bad command or file name"
CALL DISP _MESS
JMP GO _LP
EXEC _IT:
CALL EXEC _CMD ;执行外部命令
MOV AH,4DH
INT 21H ;取返回码
MOV EXIT _CODE,AX ;保存返回码
JMP GO _LP
BAD _CMD DB 'Bad command or file name $'
CRLF DB 0DH,0AH,' $ '
CMD _BUF DB 50,?,50 DUP(?)
EXIT _CODE DW ?
BIG _SMBL DB 'SHELL> $ '
COM DB 'COMEXEBAT'
PARBLK DW 0
PARCMD DW OFFSET CMDTAIL
DW ?
DD -1
DD -1
CMDTAIL DB 0,0DH
ERRMSG DB 'Execute Failure!' '0dh,0ah' '$ '
INNER _CMDS EQU $ ;内部命令表
DB 'CLS',0
DW OFFSET CLS _CMD
DB 'SCD',0
DW OFFSET SCD _CMD
DB 'SREN',0
DW OFFSET SREN _CMD
```

```

        DB      'EXIT',0
        DW      EXIT __CMD
        DB      0
DISP __MESS PROC
        MOV     AH,9
        INT     21H
        RET
DISP __MESS ENDP
PROMPT      PROC ;该过程显示提示符
        LEA     DX,CRLF
        CALL    DISP __MESS
        LEA     DX,BIG __SMBL
        CALL    DISP __MESS
        RET
PROMPT      ENDP
GET __CMD   PROC ;该过程从键盘接收命令
        LEA     DX,CMD __BUF
        MOV     AH,10
        INT     21H
        MOV     CL,CMD __BUF+1
        CMP     CL,0
        JA      G __OK
        CALL    PROMPT
        JMP     GET __CMD
G __OK:
        RET
GET __CMD   ENDP
INNER      PROC ;该过程处理内部命令
        CLD
        LEA     SI,INNER __CMDS
INN1:
        CMP     BYTE PTR[SI],0
        JNZ     INN2
        STC
        RET
INN2:
        LEA     DI,CMD __BUF+2
        MOV     CL,CMD __BUF+1
INN3:
        CMP     BYTE PTR[DI],' '
        JNE     INN4
        INC     DI
        JMP     INN3
INN4:
        MOV     AL,[SI]
        OR      AL,AL
        JZ      INN5
        CMP     AL,[DI]
        JNZ     INN6
        INC     SI
        INC     DI
        JMP     INN4
INN5:
        CMP     BYTE PTR[DI],0DH
        JZ      INN7
        CMP     BYTE PTR[DI],' '
        JNZ     INN6
INN7:
        CALL    WORD PTR[SI+1]
        CLC
        RET
INN6:
        LODSB
        OR      AL,AL
        JNZ     INN6
        ADD     SI,2
        JMP     INN1
INNER      ENDP
SEARCH __CMD PROC ;该过程查找是否存在可执行文件
        LEA     DI,CMD __BUF+2
        MOV     CL,CMD __BUF+1
        XOR     CH,CH
        ADD     DI,CX
        MOV     AL,'.'
        STOSB
        MOV     CX,3
        LEA     SI,COM
S __LP:
        PUSH    DI
        PUSH    CX
        MOV     CX,3
        REP     MOVSB
        MOV     BYTE PTR[DI],0
        LEA     DX,CMD __BUF+2
        XOR     CX,CX
        MOV     AH,4EH
        INT     21H
        POP     CX
        POP     DI
        JNC     S __OK
        LOOP    S __LP
        STC
        RET
S __OK:
        CLC
        RET
EXEC __CMD  PROC ;该过程执行外部命令
        LEA     DX,CMD __BUF+2
        LEA     BX,PARBLK
        CALL    EXEC
        RET
EXEC __CMD  ENDP
STKSEG      DW      ?
STKPTR      DW      ?
;
EXEC         PROC
        PUSH    DS
        PUSH    ES
        MOV     CS,STKSEG,SS
        MOV     CS,STKPTR,SP
        MOV     AH,4BH
        MOV     AL,0
        INT     21H
        MOV     SS,CS,STKSEG
        MOV     SP,CS,STKPTR
        POP     ES
        POP     DS
        JNC     EXEC1
        PUSH    AX
        LEA     DX,ERRMSG
        CALL    DISP __MESS
EXEC1:
        RET
EXEC         ENDP
CLS __CMD   PROC ;内部命令 CLS 的执行过程
        MOV     AX,600H
        MOV     BH,7
        MOV     CX,0
        MOV     DX,184FH
        INT     10H
        CALL    LOCATE00
        RET
CLS __CMD   ENDP
EXIT __CMD  PROC
        MOV     AH,4CH
        INT     21H
        RET
EXIT __CMD  ENDP
LOCATE00    PROC
        MOV     DX,0
        CALL    LOCATE
        RET
LOCATE00    ENDP
LOCATE      PROC
        PUSH    BX
        PUSH    CX
        MOV     BX,0
        MOV     AH,2
        INT     10H
        POP     CX
        POP     BX
        RET
LOCATE      ENDP
INITIZER:
        MOV     CS:[16H],CS
        LEA     DX,G0-LP
        MOV     CS:[0AH],DX
        MOV     CS:[0CH],CS
        MOV     AX,2522H
        INT     21H
        MOV     [PARCMD+2],CS
        LEA     BX,INITIZER
        MOV     CL,4
        SHR     BX,CL
        ADD     BX,5
        MOV     AH,4AH
        MOV     BX,400
        INT     21H
        CALL    DISP-COPYRIGHT
        JMP     START
COPYRIGHT    DB      'Simple SHELL to test COMMAND INTERPRETER!',10,13
        DB      'Copyright NCI,AI 1991',13,10,'$'
DISP __COPYRIGHT PROC
        LEA     DX,COPYRIGHT
        CALL    DISP __MESS
        RET
DISP __COPYRIGHT ENDP
CODE        ENDS
END         BEGIN

```

# DIR 命令的妙用

□ 常 久

**D**IR 命令是 DOS 各版本提供的一个内部命令,它用于文件的显示,我们通常知道 DIR 命令的两个参数/P 和/W。/P 用于分页显示,/W 用于横向显示,如果我们按日期、扩展名、文件名排序、文件大小、文件建立时间显示,以及显示子目录,DOS5.0 以下的版本则不能实现,这对于用户很不方便,本文采用一个批处理程序完善 DOS5.0 以下版本的 DIR 命令,并介绍 DOS5.0 以上版本的 DIR 命令的参数。

## 一、DOS5.0 以下版本 DIR 命令的完善

这里给出一个批处理文件 DDIR.BAT,该文件用于完善 DOS5.0 以下版本的 DIR 命令,该批处理文件的内容如下:

其中参数:

/A 按日期显示  
/D 显示子目录  
/E 按扩展名分类显示  
/N 按名字分类显示  
/S 按文件的大小显示  
/T 按文件的建立时间显示

```
@echo off
IF /A==%1 GOTO DATE
IF /D==%1 GOTO DIR
IF /E==%1 GOTO EXT
IF /N==%1 GOTO NAME
IF /S==%1 GOTO SIZE
IF /T==%1 GOTO TIME
:HELP
ECHO DDIR
ECHO
ECHO SORT & FIND MUST EXIST IN DOS SUBDIRECTORY
ECHO /A= SORT BY DATE
ECHO /D= SORT AND LIST ONLY DIRECTORY
ECHO /E= SORT BY EXTENSION
ECHO /N= SORT BY NAME
ECHO /S= SORT BY SIZE
ECHO /T= SORT BY TIME
ECHO
ECHO SYNTAX:DDIR SWITCH[DIRECTORY]
GOTO END
:DATE
ECHO %2 DIRECTORY: SORT BY DATE
DIR %2|C:\DOS\SORT /+24|C:\DOS\FIND/V"e"
GOTO END
:DIR %2
ECHO %2 DIRECTORY:LIST OF SUBDIRECTORY
DIR %2|C:\DOS\SORT|C:\DOS\FIND"DIR)"
GOTO END
:EXT
ECHO %2 DIRECTORY: SORT BY EXTENSION
```

```
DIR %2|C:\DOS\SORT/+10|C:\DOS\FIND/V"e"|C:\DOS\FIND/V
"DIR)"
GOTO END
:NAME
ECHO %2 DIRECTORY: SORT BY NAME
DIR %2|C:\DOS\SORT|C:\DOS\FIND/V"e"
GOTO END
:SIZE
ECHO %2 DIRECTORY: SORT BY SIZE
DIR %2|C:\DOS\SORT /+16|C:\DOS\FIND/V"e"|C:\DOS\FIND /V
"DIR)"
GOTO END
:TIME
ECHO %2 DIRECTORY: SORT BY TIME
DIR %2|C:\DOS\SORT /+34|C:\DOS\SORT /+39|C:\DOS\FIND /V"e"
GOTO END
:END
ECHO ON
```

## 二、DOS5.0 以上版本 DIR 命令参数介绍

DOS5.0 以上版本对 DIR 命令提供了较为丰富的参数,下面我们对有关参数举例介绍:

DOS5.0 版本的两个重要的开关参数/A 和/O,与这两个开关参数联用的相关参数如下表:

/A 开关参数

选择	功能介绍
—	使正常参数功能反向
A	显示档案文件
D	只显示目录
H	只显示隐含文件
R	只显示只读文件
S	只显示系统文件

/O 开关参数

选择	功能介绍
—	使正常参数功能反向
D	按日期和时间显示
E	按扩展名显示
G	先列子目录再显示文件
N	按名字显示
S	按文件大小显示

下面我们举例加以说明:

用 DIR 命令显示 C 盘根目录如下:

```
COMMAND          COM          47845    05-11-93    3:46p
DOS               <DIR>          05-06-93    9:18a
BETATEST          <DIR>          02-21-94    11:08a
```



CONFIG	SYS	237	02-08-94	9:08a
WINWORD	<DIR>		02-21-94	10:49a
HGW	<DIR>		12-07-93	8:46a
WINDOWS	<DIR>		11-17-93	2:47p
AUTOEXEC	BAT	136	02-21-94	11:04a
SMARTDRV	SYS	8323	04-29-91	5:20a
BETATEST	INI	295	01-12-94	2:21a
BETATEST	INI	295	01-12-94	2:21p
FOXPROW	<DIR>		02-07-94	11:30a
EXCEL	<DIR>		11-17-93	3:30p
M-6403	EXE	10733	04-19-92	12:56a
DD	<DIR>		02-16-94	10:28a
M-6403	<DIR>		10-04-93	4:12p
SUN	<DIR>		03-04-94	3:34p
JJJ	TXT	2093	03-31-94	10:30a
SCRNFILE	IMG	39376	03-31-94	10:12a
TEST	<DIR>		04-04-94	10:12a

同样的内容用 DIR/A 显示如下:

MSDOS	SYS	37394	05-11-93	3:46p
COMMAND	COM	47845	05-11-93	3:46p
DOS	<DIR>		05-06-93	9:18a
BETATEST	<DIR>		02-21-94	11:08a
CONFIG	SYS	237	02-08-94	9:08a
WINWORD	<DIR>		02-21-94	10:49a
HGW	<DIR>		12-07-93	8:46a
WINDOWS	<DIR>		11-17-93	2:47p
AUTOEXEC	BAT	136	02-21-94	11:04a
SMARTDRV	SYS	8323	04-29-91	5:20a
386SPART	PAR	12570624	04-08-94	11:13a
BETATEST	INI	295	01-12-94	2:21p
FOXPROW	<DIR>		02-07-94	11:30a
EXCEL	<DIR>		11-17-93	3:30p
M-6403	EXE	10733	04-19-92	12:56a
D	<DIR>		02-16-94	10:28a
M-6403	<DIR>		10-04-93	4:12p
SUN	<DIR>		03-04-94	3:34p
JJJ	TXT	2093	03-31-94	10:30a
SCRNFILE	IMG	39376	03-31-94	10:12a
TEST	<DIR>		04-04-94	10:12a

用 DIR/AA/O-D 显示如下:

386SPART	PAR	1257624	04-08-94	11:13a
JJJ	TXT	2093	03-31-94	10:30a
SCRNFILE	IMG	39376	03-31-94	10:12a
AUTOEXEC	BAT	136	02-21-94	11:04a
CONFIG	SYS	237	02-08-94	9:08a
BETATEST	INI	295	01-12-94	2:1p
COMMAND	COM	47845	05-11-93	3:46p
M-6403	EXE	10733	04-19-92	12:56a
SMARTDRV	SYS	8323	04-29-91	5:20a

用 DIR/OD 显示如下:

SMARTDRV	SYS	8323	04-29-91	5:20a
M-6403	EXE	10733	04-19-92	12:56a
DOS	<DIR>		05-06-93	9:18a
COMMAND	COM	47845	05-11-93	3:46p
M-6403	<DIR>		10-04-93	4:12p
WINDOWS	<DIR>		11-17-93	2:47p

EXCEL	<DIR>		11-17-93	3:30p
HGW	<DIR>		12-07-93	8:46a
BETATEST	INI	295	01-12-94	2:21p
FOXPROW	<DIR>		02-07-93	11:30a
CONFIG	SYS	237	02-08-94	9:08a
DD	<DIR>		02-16-94	10:28a
WINWORD	<DIR>		02-21-94	10:49a
AUTOEXEC	BAT	136	02-21-94	11:04a
BETATEST	<DIR>		02-21-94	11:08a
SUN	<DIR>		03-04-94	3:34p
SCRNFILE	IMG	39376	03-31-94	10:12a
JJJ	TXT	2093	03-31-94	10:30a
TEST	<DIR>		04-04-94	10:12a

用 DIR/O-D 显示的结果如下:

TEST	<DIR>		04-04-94	10:12a
JJJ	TXT	2093	03-31-94	10:30a
SCRNFILE	IMG	39376	03-31-94	10:12a
SUN	<DIR>		03-04-94	3:34p
BETATEST	<DIR>		02-21-94	11:08a
AUTOEXEC	BAT	136	02-21-94	11:04a
WINWORD	<DIR>		02-21-94	10:49a
DD	<DIR>		02-16-94	10:28a
CONFIG	SYS	237	02-08-94	9:08a
FOXPROW	<DIR>		02-07-94	11:30a
BETATEST	INI	295	01-12-94	2:21p
HGW	<DIR>		12-07-93	8:46a
EXCEL	<DIR>		11-17-93	3:30p
WINDOWS	<DIR>		11-17-93	2:47p
M-6403	<DIR>		10-04-93	4:12p
COMMAND	COM	47845	05-11-93	3:46p
DOS	<DIR>		05-06-93	9:18a
M-6403	EXE	10733	04-19-92	12:56a
SMARTDRV	SYS	8323	04-29-91	5:20a

DOS5.0 以上版本除提供 DIR 命令常用的参数/P、/W 以及我们介绍的/A 和/O 命令外,还提供了/L(以小写字母显示目录信息)、/B(只显示文件名和目录名)、/S(显示现行目录下的所有文件及其下所有子目录中的文件)参数。DIR 命令参数的综合使用将大大有利于用户显示磁盘信息,如:我们要以横向、分页显示全盘中的档案文件,并且最新文件在前依次到最老的文件,可用如下命令:

```
DIR C:\A:A/O:-D/S/W/P
```

### 三、DOS5.0 以上版本 DIR 命令的妙用

如果我们要在每次 DIR 时以字母顺序显示并且满屏暂停,我们可以在 AUTOEXEC.BAT 中加入如下命令:

```
SET DIRCMD=/O:N/P
```

如果要恢复 DIR 命令的默认功能则可键入:

```
SET DIRCMD
```

如果我们要在全盘搜索一个我们需要的命令,则可配合 DOSKEY 命令加以使用,如:

```
DOSKEY GET=DIR\$/S
```

\$1 需要用户输入文件名。

# 巧用 DOS 的 I/O 重定向功能 10 例

□朱 猛

**D**OS 中规定了标准的输入输出(I/O)设备,默认状况下,DOS 将键盘作为标准输入,将屏幕显示作为标准输出。PC-DOS 2.00 以上版本为用户提供 I/O 重定向功能,允许标准输入输出的重定向:输出转向“>”将命令输出到一个文件或设备;输出转向“>>”将命令的输出附加在某个文件末尾;输入转向“<”将命令输入改为一个文件或非标准输入设备。改向符“>”、“>>”或“<”以及管道操作符“|”的解释和执行是由 COMMAND.COM 模块完成的,管道操作使一个命令的输出成为第二个命令的输入,实际上是标准输入、输出重定向的组合。

灵活巧妙地运用 DOS 提供的 I/O 重定向功能,能解决许多实际问题,收到事半功倍的效果。笔者在实际运用中积累了一些 I/O 重定向(>、>>、<)的使用经验,现整理成文献给读者,以求起到抛砖引玉的目的。

## 1. 消除不必要信息的显示

在 DOS 命令后面加上 >NUL 输出转向命令,可将该命令的输出重定向到一个不存在的设备 NUL,实际上是什么也不输出,这样就可以解决批文件中 ECHO OFF 命令所不能解决的多余信息输出问题。例如在批文件中有一条 COPY 命令,用 ECHO OFF 不能消除它所生成的“1 File(s) copied”信息,而在 COPY 命令后面加上输出转向 >NUL 即可消除。如:

```
copy filename1 filename2>NUL
```

## 2. 拷贝加密盘文件

对一些加密盘用 DISKCOPY 和 PC-TTOOLS 均提示有坏的磁道,无法完成拷贝工作。这时用病毒检测软件检查,记下盘上要拷贝文件的文件名,然后巧妙利用输出转向 > 和 TYPE 命令结合可以完成文件的拷贝。假如源盘插入 A 驱动器中,目标盘插入 B 驱动器中,那么拷贝的命令格式为:

```
A:>TYPE 文件名>B:文件名
<Enter>
```

注意:拷贝的同时可以改变文件名。这种方法并不能适用所有的加密软件。

## 3. COM 和 EXE 型文件的调试

我们知道,为了查看、修改 COM 和 EXE 型源程序,一般采用 DEBUG 调试软件。但对于初学者来说,用 DEBUG 很不方便。下述方法可以实现用编辑功能很强的、用户最熟悉的 EDLIN 或 WORDSTAR 软件来获取 COM 和 EXE 型源程序。

(1) 利用 DOS 输出重定向功能,将 COM 或 EXE 型文件定义到某一文件中,如:

```
C:>debug command.com>ABC <Enter>
```

```
-U CS: 0 50;对 CS:0 到 50 反汇编
```

```
-Q ;返回
```

值得注意的是:此程序并不在屏幕上显示,而是定向到文件 ABC 中,操作时只能看到驱动器指示灯闪烁,因此要注意输入正确。

(2) 执行 EDLIN ABC 或进入 WORDSTAR,采用 D 命令,调出文件 ABC 进行分析。

## 4. 简便的计数器

如果需要统计某个程序的运行次数或了解计算机的使用频度,可以编个程序解决计数问题。但实际上有简便易行的一条语句就能实现计数器作用的方法:利用输出转向 >> 功能,将语句 ECHO + >> ABC 加进 .BAT 文件中,那么它运行一次,文件 ABC 中就增加了一个“+”号,因此,文件 ABC 也就实现了计数器的作用。

另外,如果想要记录每次使用计算机的日期和时间,以便加强用机管理,可以采用如下方法:在自动批处理文件 AUTOEXEC.BAT 中加入下列命令:

```
(1)ECHO +>>temp.abc
(2)ECHO +>>temp.abc
(3)TYPE temp.abc|DATE>>ABC
(4)TYPE temp.abc|TIME>>ABC
(5)DEL temp.abc
```

其中,第 1 行和第 2 行将执行 DATE 和 TIME 命令时要手工键入的“回车”重定向到一个临时文件 temp.abc 中,第 3 行和第 4 行则利用 DOS 管道功能将 DATE 和 TIME 命令的输出信息重定向到文件 ABC 中。这样在每次启动计算机

以后可以执行命令 C:>TYPE ABC 来查看每次使用计算机的日期和时间。

## 5. 文本文件的分屏显示

TYPE 命令显示文件时不能分页显示,由于机器显示的速度太快,所以不得不频繁地按暂停键来控制显示速度,这是很不方便的。为了实现文本文件的分屏显示,可以编制一段程序实现,很多报刊上都曾介绍过这种方法。实际上采用 DOS 的重定向功能和 MORE 命令可以方便地实现文本文件的分屏显示。方法为:

```
C:>MORE<文本文件名 <Enter>
```

注意:使用上述方式显示文件时,在当前目录下必须要有 MORE.COM。MORE 命令的用途是从标准的输入设备读入数据,并将一组满屏数据送入标准的输出设备,然后停顿,并给出信息“——MORE——”,按任一键就可以继续显示下一屏数据,直到全部文件显示完毕为止。

利用 DOS 管道操作“|”及 TYPE 命令也能实现文本文件的分屏显示。方法为:

```
C>TYPE 文本文件名 |
MORE <Enter>
```

## 6. 实现后台作业功能

有时,对于需要大批量占用机器执行时间的程序如大量的数据计算和输出打印等,在已知操作过程的情况下,可先将具体操作控制步骤存放于一个磁盘文件中,然后一切操作中的输入控制过程,重定向于由磁盘文件当中直接读取,从而减少键盘输入和人的干预。在操作过程结束,磁盘控制文件读取完成了之后,按 Ctrl+Break 键连同操作过程全部退出,操作员可继续干其他事情。例如有一程序 FILE.EXE 操作过程为:在主程序下选“04”回车为打印功能;再选“01”回车为打印人员工资表;再选打印科室为“08”回车;选“10”回车为打印月份,开始打印;打印完了之后再返回打印功能,等待再选其他打印功能……,如还是打印人员工资表,可按如下过程建立磁盘文件 ABC:

# 高密软盘快速复制

□易言

286、386 机的标准配置为 1.2M 5.25" + 1.44M 3.5" 软驱, 或 1.2M + 360K 5.25" 软驱, 并有 4M 或 2MRAM。这类机器的 1.2M 或 1.44M 磁盘复制由于只有一个同类高密软驱, 源盘和目标盘只能使用同一软驱, 目前各版 MS DOS 和各种工具软件都是先读源盘内容, 然后提示插入目标盘, 再进行写盘操作, 完成复制过程最少反复三次这个过程, 若要复制多份备份, 还要反复读源盘, 相当繁琐。究其原因 DOS 的常规内存最大为 640K, 所用拷盘程序只使用常规内存而没能利用机器提供的扩充和扩展内存。

DR DOS 6.0 的 DISKCOPY 命令充分发挥了这类机器的强大资源优势。它利用 XMS、EMS 或在磁盘上建立临时文件对高密软盘进行快速复制, 而不必多次换盘, 速度提高三倍以上, 极大地提高了工作效率。

根据机器的配置, 在 CONFIG.SYS 中设置 XMS 或 EMS 驱动程序 (HIDOS.SYS 或 EMS386.SYS), 用 DISKCOPY 拷贝高密软盘时, 自动利用 XMS 或 EMS 存储源盘内容, 一次完成拷贝。

在 DR DOS 6.0 环境下, 无 XMS 和 EMS 的机器, 借助在硬盘上生成临时文

件也能实现快速拷盘。例如:

```
C:\>DISKCOPY A: C:\DISKA.IMG
```

```
C:\>DISKCOPY C:\DISKA.IMG A:
```

使用参数 /M 可实现一次读入多次拷贝, 拷贝完成时, 询问是否拷贝其他软盘, 直到完成所需数目; 参数 /A 用音响提醒用户拷贝完成或需要换软盘。

```
C:\>DISKCOPY C:\DISKA.IMG
A:/M/A
```

若在 MS DOS 下运行 DR DOS 6.0 的 DISKCOPY 命令, “系统并不提示 ‘Incorrect DOS Version’, 而是在读源盘时报告 ‘Error reading from the disk’”。

可以是大概记得的一些特殊字符及字符串等。

## 10. 特征文件的批量处理

特征文件的处理, 如删除磁盘中所有的 .BAK 文件、清理众多的 .DBF 文件中的数据、拷贝所有 .DBF 文件结构、打印所有的 .PRG 文件等等, 都可以利用 DOS 重定向 > 将所有特征文件定向到某个文件中去, 然后建立 .BAT 文件实现批量处理的目的。下面以一次性删除磁盘中所有 .BAK 文件为例加以说明。

首先, 在 dBASE III 系统下建立一个名为 SCBAK.DBF 的库文件, 结构为:

字段名	类型	宽度
X	C	80

进入 DOS 系统, 作如下操作:

```
C:\>CHKDSK/V|FIND ".BAK">
ABC.TXT
```

这样就将所有 .BAK 文件的特征信息重定向到文件 ABC.TXT 中。

进入 dBASE III, 建立以下程序:

```
USE SCBAK
APPE FROM ABC.TXT SDF
REPL ALL X WITH "DEL"
+SUBS(X,1,8)
```

```
COPY TO AUT.BAT SDF
```

执行批文件 AUT.BAT 即可一次性删除所有 .BAK 文件。

上述操作过程可以建成批处理文件, 限于篇幅, 本文不给出具体文件。

```
C:\>COPY CON ABC <Enter>
```

```
04<      ;选打印功能
```

```
01<      ;打印人员工资表
```

```
08<      ;打印 08 科室
```

```
10<      ;打印 10 月份, 开始打印
```

```
<      ;打印完了, 回车返回
```

```
01<      ;打印人员工资表
```

```
09<      ;打印 09 科室
```

```
10<      ;打印 10 月份, 开始打印
```

```
<      ;打印完了, 回车返回
```

```
.....
```

然后执行 C:\>FILE<ABC, 在控制文件 ABC 读取完后, 机器继续等待, 键盘控制不起作用。按 Ctrl+Break 键退出运行程序, 返回提示符 C:\>, 这时可以继续通过键盘控制干其他事情。这样就巧妙地利用输入重定向 < 实现了后台作业功能。

## 7. 文件名的解密

一些用户经常使用半个汉字或空区位码的方法给文件名加密, 实际上用 DOS 的现有功能可以方便地进行解密。如果加密文件名中有一个或几个能看清楚的字或汉字等标志字, 可以用具有通配符 ? 或 \* 的 COPY 命令将其复制成新文件名的方法解密。如没有标志字, 可以利用 DOS 重定向 > 进行解密, 方法为: 先执行 C:\>DIR>ABC.BAT, 再调用 EDLIN.COM 对文件 ABC.BAT 进行编辑, 删去不必要的信息, 仅留下加密行中的加密文件名, 然后插入相应的 DOS 命令, 如在加密文件名前插入 “REN” 和一

格字符, 后面插入一空格字符和 “ZHM”, 然后存盘。执行 C:\>ABC 即可将加密文件名改为 ZHM, 达到了解密的目的。

这种方法同样适用于目录名的解密, 只要在编辑取得的目录名前插入 “CD” 和一空格符, 然后存盘执行, 即可进入加密目录。

## 8. 目录和文件的遍历

CHKDSK 加上参数 “V” 能显示盘中所有文件和路径, 而不管它们是否隐藏以及如何隐藏。但执行中显示的信息过多过快, 不便于查看, 这时可以利用输出重定向 > 将显示信息重定向到某一文件中慢慢去看。即 C:\>CHKDSK/V>ABC <Enter>, 则生成的文件 ABC 可以利用 EDLIN 和 WORDSTAR 软件打开, 查看所有的目录和文件, 且每个文件前都有它完整的路径信息。

## 9. 文件快速定位

DOS 层次目录结构的实现, 给用户带来很大的方便, 但同时也使得查找不在当前目录下的文件变得复杂费时。利用 DOS 重定向 > 和命令 CHKDSK 可以实现复杂的层次目录结构中方便快速地定位文件, 方法如下:

```
C:\>CHKDSK/V>ABC <Enter>
```

```
C:\>FIND 文件名 ABC <Enter>
```

输出结果不仅给出了要找的文件名, 还给出了路径信息。上述命令中文件名必须大写, 它可以是已知的完整文件名, 也



## 栏目编辑的话

近年来,随着视频显示技术的发展,图形软件有了很大的发展,人机界面及人机交互产生了革命性的变革,以图形模式为基础编制软件已成为软件开发和应用的时尚,尤其是 Microsoft 的 Windows 的推出对把这一潮流推上了一个新的高峰,广大计算机软件开发人员和应用人员紧随这一时尚对编制图形软件产生了极大的兴趣,他们迫切需要了解图形软件的编制技术、技巧,并希望通过适当的环境交流各自工作中的心得、体会和经验。图形、图象处理栏目正是为满足这些读者的愿望而开设的,它向读者介绍图形软件的编制技术、技巧和编程的具体方法。它包括图形软件界面的开发、鼠标的驱动等等。以后本杂志将顺潮流而动,将增加在图形、图象处理技术方面的技术信息,我们真诚地希望得到广大读者的支持和帮助。

## 彩色图象的抖动打印

□计学荣

随着计算机和扫描仪的普及使用,人们利用彩色图象的领域越来越多,如人事档案管理、商标管理等领域。由于彩色打印机的出现和普及使用,人们希望利用彩色打印机打印彩色照片,但是一般彩色打印机只提供 Windows 下的驱动,而不提供 DOS 下的驱动,这给 DOS 用户带来不便。用户使用本文给出的彩色打印技术可以编写自己的打印驱动。

## 一、抖动的概念

抖动(DITHER)是将彩色图象还原为一组黑白斑点(DOTS)的算法过程,这种方法能应用于显示及打印输出,使用抖动的直接原因在于输出的灰度级没有图象的灰度级高,比如显示的灰度为 64 级,而图象本身为 256 级,则就要使用抖动方法,将 256 级灰度变成 64 级再显示,打印输出也是如此。具体实现抖动的方法有很多种,下面给出简要介绍:

## 1. RYLANDER 4×4 模式法

该方法是构造一个 4×4 的矩阵,每个 4×4 矩阵都能表示 4×4+1=17 级灰度级,灰度级数对应 4×4 矩阵中均匀分布的点数。

0 级	1 级	2 级
0 0 0 0	1 0 0 0	1 0 0 0
0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 0	0 0 0 0	0 0 1 0
0 0 0 0	0 0 0 0	0 0 0 0

3 级	4 级	5 级
1 0 1 0	1 0 1 0	1 0 0 0
0 0 0 0	0 0 0 0	1 0 1 0
0 0 1 0	1 0 1 0	1 0 1 0
0 0 0 0	0 0 0 0	0 0 0 0

6 级	7 级	8 级
1 0 1 0	1 0 1 0	1 0 1 0
1 0 0 0	1 0 1 0	1 0 1 0
1 0 1 0	1 0 1 0	1 0 1 0
0 0 1 0	0 0 1 0	1 0 1 0

9 级	10 级	11 级
1 1 1 0	1 1 1 0	1 1 1 1
1 0 1 0	1 0 1 0	1 0 1 0
1 0 1 0	1 0 1 1	1 0 1 1
1 0 1 0	1 0 1 0	1 0 1 0

12 级	13 级	14 级
1 1 1 1	1 1 1 1	1 1 1 1
1 0 1 0	1 1 1 0	1 1 1 0
1 1 1 1	1 1 1 1	1 1 1 1
1 0 1 0	1 0 1 0	1 0 1 1

15 级	16 级
1 1 1 1	1 1 1 1
1 1 1 1	1 1 1 1
1 1 1 1	1 1 1 1
1 0 1 1	1 1 1 1

首先将原图 256 级灰度变成 16 级灰度(如除以 16),再根据得到的灰度级对应相应的 4×4 矩阵,如果对应的矩阵点为 1 就打印,为 0 就不打印。具体使用时既可以一个图象点对应 4×4 矩阵,也可以 4×4 矩阵对应 4×4 矩阵,后者是将 4×4 矩阵中的 16 个点的灰度值相加取平均值再用抖动。

## 2. RYLANDER 2×2 模式法

2×2 矩阵能形成 2×2+1=5 级灰度,同理构成 2×2 矩阵:

0 级	1 级	2 级	3 级	4 级
0 0	1 0	1 0	1 1	1 1
0 0	0 0	0 1	0 1	1 1

首先将 256 级灰度变成 4 级灰度(如 256/64),再根据灰度级对应相应的 2×2 矩阵,如果点为 1 打印,点为 0 则不打印。具体使用时既可以一个图象点对应 2×2 矩阵,也可以 2×2 矩阵对应 2×2 矩阵,后者是将 2×2 矩阵中的 4 个点的灰度值相加取平均值再用抖动。

## 3. RYLANDER 1×1 模式法

1×1 方法就是将图象变成 2 级灰度,以 128 为界,灰度级大于或等于 128 就打印,小于 128 就不打印。

## 4. BAYER 高频振动法

BAYER 矩阵:

0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

首先读原始图象块(4 \* 4),将每点的 256 级灰度变成 256/16=16 级,这样得到一个 4 \* 4 矩阵,其中每个元素值均在 0 到 16 之间,再分别与 BAYER 方阵中的元素作比较,若前者大于后者则打印,否则不打印。

## 二、误差传递

直接使用以上抖动方法打印图象,如果没有误差传递,效果将很差,人眼只能看到色块,缺乏层次感。

误差是经过抖动计算产生的,例如原始图象某点的灰度值为 150,变成 16 级灰度时,该点的误差为  $150\%16=6$ ,变成 4 级灰度时该点的误差为  $150\%4=22$ ,变成 2 级时该点的误差为  $150-255=-105$ ,可见误差是不能忽略的。

传递误差是指将每点的误差按一定比例顺序向右向下传递,主要有以下方法:

### 1. FLOYD—STEINBERG 滤波器

X 7  
3 5 1

含义是将 X 处的误差向右向下传递,其中向相邻右边点传递误差的 7/16,向左下方点传递误差的 3/16,向正下方传 5/16,向右下方传 1/16,各点依次类推,直至图象全部处理完毕。

该方法速度快,效果稍微差一些。

### 2. STUCKI 滤波器

X 8 4  
2 4 8 4 2  
1 2 4 2 1

该方法各比例数相加为 42,即向 X 右方点传递误差的 8/42,再向右边点传 4/42,其他点依此类推。

该方法速度慢,但效果好一些。

### 3. BURKES 滤波器

X 8 4  
2 4 8 4 2

该方法各比例数相加为 32,即向 X 右方点传递误差的 8/32,再向右边点传 4/32,其他点依此类推。

该方法速度比 STUCKI 快,效果比 FLOYD 好一些。

使用以上介绍的抖动和误差传递方法,已经能够编写打印灰度图象的程序了,但彩色图象的打印要复杂得多。

## 三、彩色图象打印

目前彩色打印机有很多种,有的高性能打印机能够直接混色打印,将 RGB 或 CMYK 各色值(0—255)直接送打印机,如 SHARP 7000,但大部分打印机不能完全混色,而只能使用这种抖动方法,而且在彩色叠加上也要特殊处理。

大部分打印机都使用黄(Y)品(M)青(C)黑(K)作为彩色打印输出。从 RGB 到 CMKY 的转换公式是:

$$M=255-G$$

$$Y=255-B$$

$$C=255-R$$

$$BK=L-(H-L)/k$$

$$\text{其中: } L=\min\{C,Y,M\}$$

$$H=\max\{C,Y,M\},$$

$$k=1\ldots 20$$

BK 为黑色成分值。

我们使用的打印输出方法是将彩色图象中的各点分别取其黄品青黑四色,而这四色分别作抖动和误差传递,分别生成要打印的数据,再分别按这四种颜色送往打印机。要注意的是有的打印机不支持三色以上混色打印(如 SHARP JX-735),处理方法是如果有三色同时打印的点,就用黑点代替,同时该点的其他三色清 0,这样该点打印为黑色。另一个问题是要不要黑色和其他三色同时作抖动和误差传递,经过实验,黑色可以不参加抖动和误差传递,而只是在三色均为 1 的点打印黑色。

## 四、关于彩色图象的无级缩放

我们一般都希望打印出来的图象与扫描的原稿一样大小,但一般情况下,用户扫描的分辨率与打印分辨率不同,这样打印出来的图象与原稿大小就不会相同。解决的方法是根据打印分辨率和图象分辨率作缩放,比如原图分辨率为 100DPI,而打印分辨率为 180DPI,则要将原图横纵向均放大  $180/100=1.8$  倍,再打印输出,这样打印的图象与原稿大小就完全相同。

实现无级缩放的方法可采用差值法(线性差值或三次差值),为了提高速度,可采用近似的靠点法。具体作法是在纵向上,当要在两行中间插入一行时,判断要插入的行与哪一行更近一些,然后该行就取与该行更近一些的那一行的值,在横向上,当要在两点中间插入一点时,判断要插入的点与哪一点更近一些,然后该点就取与该点更近一些的那一点的值。

在实际实现时,由于运算量大,要避免浮点数计算,可用长整形数的计算来代替。

## 五、总 结

我们使用以上介绍的方法实现了一些彩色打印机的彩色打印输出,特点是可以指定任意倍数的缩放,而且做到了边缩放边打印。经过比较,采用 RYLANDER 1×1 和 BAYER 矩阵方法打印效果最好,而用 RYLANDER 4×4 和 2×2 矩阵法会有锯齿出现,误差传递方法可以采用 FLOYD 方法。

RYLANDER 1×1 和 BAYER 矩阵方法打印输出的效果分别与 PHOTOSTYLER 打印中的 DIFFUSION 和 DISPERSED 方法打印输出的效果相似,而且速度上达到或超过了它的水平。

目前,本文介绍的彩色打印方法已经应用到 ITbase 通用图文数据库系统中。

# 设置各种 VGA 卡 高分辨率图形模式的 BIOS 接口

□张建明

目前,国内市场上 VGA 图形显示卡的种类越来越多,不仅有标准 VGA 卡和各种兼容 VGA 卡,还有种类众多的 TVGA 卡和 SVGA 卡,它们支持的图形分辨率可以达到 1024 \* 768,支持的图形颜色可以达到从 262144 种颜色中任选 256 色,为了充分利用它们所具备的强大的图形功能,就必须对其所提供的编程接口进行了解,但许多编程人员都苦于无法收集到全面的有关各种 VGA 卡编程接口的信息。

在此,我们分别针对各种 VGA 卡,详细给出设置各种其所支持的高分辨率图形模式的 BIOS INT 10H 中断的扩展功能调用接口,这些接口对于我们开发具有高通用性和可靠性的应用程序,具有相当的实用价值。

其中需要说明的是,如果您的 VGA 卡上的视频缓冲区容量,少于下表中列出的所需视频缓冲区容量,则可能会有若干高分辨率的显示模式无法设定,AX, BX, CX 和 DX 分别表示在进行 BIOS INT 10H 中断调用时,各相应寄存器应被设定的值。

## 1. 显示卡名称:AST

主芯片集名称:PARADISE

所需视频缓冲区容量:256KB

- 1 分辨率320 \* 200,256 色/262144 色模式:  
\_AX=0x13, \_BX=0, \_CX=0, \_DX=0;
- 2 分辨率640 \* 400,256 色/262144 色模式:  
\_AX=0x5e, \_BX=0, \_CX=0, \_DX=0;
- 3 分辨率640 \* 480,16 色/64 色模式:  
\_AX=0x10, \_BX=0, \_CX=0, \_DX=0;
- 4 分辨率640 \* 480,16 色/64 色模式:  
\_AX=0x12, \_BX=0, \_CX=0, \_DX=0;
- 5 分辨率800 \* 600,16 色/262144 色模式:  
\_AX=0x58, \_BX=0, \_CX=0, \_DX=0

## 2. 显示卡名称:AST VGA Plus

主芯片集名称:PARADISE

所需视频缓冲区容量:512KB

- 1 分辨率320 \* 200,256 色/262144 色模式:  
\_AX=0x13, \_BX=0, \_CX=0, \_DX=0;
- 2 分辨率640 \* 400,256 色/262144 色模式:  
\_AX=0x5e, \_BX=0, \_CX=0, \_DX=0;
- 3 分辨率640 \* 480,256 色/262144 色模式:  
\_AX=0x5f, \_BX=0, \_CX=0, \_DX=0;
- 4 分辨率640 \* 350,16 色/64 色模式:  
\_AX=0x10, \_BX=0, \_CX=0, \_DX=0;
- 5 分辨率640 \* 480,16 色/64 色模式:  
\_AX=0x12, \_BX=0, \_CX=0, \_DX=0;
- 6 分辨率800 \* 600,16 色/262144 色模式:  
\_AX=0x58, \_BX=0, \_CX=0, \_DX=0.

## 3. 显示卡名称:Compaq

主芯片集名称:PARADISE

所需视频缓冲区容量:512KB

- 1 分辨率320 \* 200,256 色/262144 色模式:  
\_AX=0x13, \_BX=0, \_CX=0, \_DX=0;
- 2 分辨率640 \* 400,256 色/262144 色模式:  
\_AX=0x5e, \_BX=0, \_CX=0, \_DX=0;
- 3 分辨率640 \* 480,256 色/262144 色模式:  
\_AX=0x5f, \_BX=0, \_CX=0, \_DX=0;
- 4 分辨率640 \* 350,16 色/64 色模式:  
\_AX=0x10, \_BX=0, \_CX=0, \_DX=0;
- 5 分辨率640 \* 480,16 色/64 色模式:  
\_AX=0x12, \_BX=0, \_CX=0, \_DX=0;
- 6 分辨率800 \* 600,16 色/262144 色模式:  
\_AX=0x58, \_BX=0, \_CX=0, \_DX=0

## 4. 显示卡名称:Dell

主芯片集名称:PARADISE

所需视频缓冲区容量:512KB

- 1 分辨率320 \* 200,256 色/262144 色模式:  
\_AX=0x13, \_BX=0, \_CX=0, \_DX=0;
- 2 分辨率640 \* 400,256 色/262144 色模式:  
\_AX=0x5e, \_BX=0, \_CX=0, \_DX=0;
- 3 分辨率640 \* 480,256 色/262144 色模式:  
\_AX=0x5f, \_BX=0, \_CX=0, \_DX=0;
- 4 分辨率640 \* 350,16 色/64 色模式:  
\_AX=0x10, \_BX=0, \_CX=0, \_DX=0;
- 5 分辨率640 \* 480,16 色/64 色模式:  
\_AX=0x12, \_BX=0, \_CX=0, \_DX=0;
- 6 分辨率800 \* 600,16 色/262144 色模式:  
\_AX=0x58, \_BX=0, \_CX=0, \_DX=0

## 5. 显示卡名称:Everex EV-673

主芯片集名称:EVEREX

所需视频缓冲区容量:256 KB

- 1 分辨率320 \* 200,256 色/262144 色模式:  
\_AX=0x13, \_BX=0, \_CX=0, \_DX=0;
- 2 分辨率640 \* 350,256 色/262144 色模式:  
\_AX=0x70, \_BX=0x13, \_CX=0, \_DX=0;
- 3 分辨率640 \* 400,256 色/262144 色模式:  
\_AX=0x70, \_BX=0x14, \_CX=0, \_DX=0;
- 4 分辨率512 \* 480,256 色/262144 色模式:  
\_AX=0x70, \_BX=0x15, \_CX=0, \_DX=0;
- 5 分辨率640 \* 350,16 色/64 色模式:  
\_AX=0x10, \_BX=0, \_CX=0, \_DX=0;
- 6 分辨率640 \* 480,16 色/64 色模式:  
\_AX=0x12, \_BX=0, \_CX=0, \_DX=0;
- 7 分辨率800 \* 600,16 色/262144 色模式:  
\_AX=0x70, \_BX=0x02, \_CX=0, \_DX=0

## 6. 显示卡名称:Genoa 5200

主芯片集名称:TSENG



所需视频缓冲区容量:512KB

- 1 分辨率320\*200,256色/262144色模式:  
\_\_AX=0x13,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 2 分辨率640\*350,256色/262144色模式:  
\_\_AX=0x2d,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 3 分辨率640\*480,256色/262144色模式:  
\_\_AX=0x2e,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 4 分辨率640\*350,16色/64色模式:  
\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 5 分辨率640\*480,16色/64色模式:  
\_\_AX=0x12,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 6 分辨率800\*600,16色/262144色模式:  
\_\_AX=0x29,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 7 分辨率1024\*768,16色/262144色模式:  
\_\_AX=0x37,\_\_BX=0,\_\_CX=0,\_\_DX=0
7. 显示卡名称:Orchid Designer  
主芯片集名称:TSENG  
所需视频缓冲区容量:512KB
- 1 分辨率320\*200,256色/262144色模式:  
\_\_AX=0x13,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 2 分辨率640\*350,256色/262144色模式:  
\_\_AX=0x2d,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 3 分辨率640\*480,256色/262144色模式:  
\_\_AX=0x2e,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 4 分辨率800\*600,256色/262144色模式:  
\_\_AX=0x30,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 5 分辨率640\*350,16色/64色模式:  
\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 6 分辨率640\*480,16色/64色模式:  
\_\_AX=0x12,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 7 分辨率800\*600,16色/262144色模式:  
\_\_AX=0x29,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 8 分辨率1024\*768,16色/262144色模式:  
\_\_AX=0x37,\_\_BX=0,\_\_CX=0,\_\_DX=0
8. 显示卡名称:Paradise VGA Pro  
主芯片集名称:PARADISE  
所需视频缓冲区容量:512KB
- 1 分辨率320\*200,256色/262144色模式:  
\_\_AX=0x13,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 2 分辨率640\*400,256色/262144色模式:  
\_\_AX=0x5e,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 3 分辨率640\*480,256色/262144色模式:  
\_\_AX=0x5f,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 4 分辨率640\*350,16色/64色模式:  
\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 5 分辨率640\*480,16色/64色模式:  
\_\_AX=0x12,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 6 分辨率800\*600,16色/262144色模式:  
\_\_AX=0x58,\_\_BX=0,\_\_CX=0,\_\_DX=0
9. 显示卡名称:Sigma VGA/H  
主芯片集名称:TSENG  
所需视频缓冲区容量:64KB
- 1 分辨率320\*200,256色/262144色模式:  
\_\_AX=0x13,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 2 分辨率640\*350,16色/64色模式:  
\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 3 分辨率640\*480,16色/64色模式:  
\_\_AX=0x12,\_\_BX=0,\_\_CX=0,\_\_DX=0;
  - 4 分辨率800\*600,16色/262144色模式:  
\_\_AX=0x29,\_\_BX=0,\_\_CX=0,\_\_DX=0
10. 显示卡名称:STB VGA/EM  
主芯片集名称:TSENG  
所需视频缓冲区容量是:512KB
- 1 分辨率320\*200,256色/262144色模式:  
\_\_AX=0x13,\_\_BX=0,\_\_CX=0,\_\_DX=0;

- 2 分辨率640\*350,256色/262144色模式:  
\_\_AX=0x2d,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 3 分辨率640\*480,256色/262144色模式:  
\_\_AX=0x2e,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 4 分辨率640\*350,16色/64色模式:  
\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 5 分辨率640\*480,16色/64色模式:  
\_\_AX=0x12,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 6 分辨率800\*600,16色/262144色模式:  
\_\_AX=0x29,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 7 分辨率960\*720,16色/64色模式:  
\_\_AX=0x36,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 8 分辨率1024\*768,16色/262144色模式:  
\_\_AX=0x37,\_\_BX=0,\_\_CX=0,\_\_DX=0

#### 11. 显示卡名称:Tecmar

主芯片集名称:TSENG

所需视频缓冲区容量:256 KB

- 1 分辨率320\*200,256色/262144色模式:  
\_\_AX=0x13,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 2 分辨率640\*350,256色/262144色模式:  
\_\_AX=0x1a,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 3 分辨率640\*400,256色/262144色模式:  
\_\_AX=0x1b,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 4 分辨率640\*350,16色/64色模式:  
\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0
- 5 分辨率640\*480,16色/64色模式:  
\_\_AX=0x12,\_\_BX=0,\_\_CX=0,\_\_DX=0

#### 12. 显示卡名称:Tecmar VGA-AD

主芯片集名称:TSENG

所需视频缓冲区容量:512KB

- 1 分辨率320\*200,256色/262144色模式:  
\_\_AX=0x13,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 2 分辨率640\*350,256色/262144色模式:  
\_\_AX=0x1a,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 3 分辨率640\*400,256色/262144色模式:  
\_\_AX=0x1b,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 4 分辨率640\*480,256色/262144色模式:  
\_\_AX=0x1c,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 5 分辨率800\*600,256色/262144色模式:  
\_\_AX=0x1d,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 6 分辨率640\*350,16色/64色模式:  
\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 7 分辨率640\*480,16色/64色模式:  
\_\_AX=0x12,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 8 分辨率800\*600,16色/262144色模式:  
\_\_AX=0x29,\_\_BX=0,\_\_CX=0,\_\_DX=0;
- 9 分辨率1024\*768,16色/262144色模式:  
\_\_AX=0x37,\_\_BX=0,\_\_CX=0,\_\_DX=0

#### 13. 显示卡名称:Vega V7

主芯片集名称:CIRRUS

所需视频缓冲区容量:64KB

- 1 分辨率320\*200,256色/262144色模式:  
\_\_AX=0x6f05,\_\_BX=0x13,\_\_CX=0,\_\_DX=0;
- 2 分辨率640\*350,16色/64色模式:  
\_\_AX=0x6f05,\_\_BX=0x10,\_\_CX=0,\_\_DX=0;
- 3 分辨率640\*480,16色/262144色模式:  
\_\_AX=0x6f05,\_\_BX=0x12,\_\_CX=0,\_\_DX=0;
- 4 分辨率752\*410,16色/262144色模式:  
\_\_AX=0x6f05,\_\_BX=0x60,\_\_CX=0,\_\_DX=0;
- 5 分辨率720\*540,16色/262144色模式:  
\_\_AX=0x6f05,\_\_BX=0x61,\_\_CX=0,\_\_DX=0;
- 6 分辨率800\*600,16色/262144色模式:  
\_\_AX=0x6f05,\_\_BX=0x62,\_\_CX=0,\_\_DX=0

#### 14. 显示卡名称:Video 7 V7 VRAM

主芯片集名称:VIDEO 7

所需视频缓冲区容量:512 KB

- 1 分辨率 320 \* 200,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x13,\_\_CX=0,\_\_DX=0;
- 2 分辨率 640 \* 400,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x66,\_\_CX=0,\_\_DX=0;
- 3 分辨率 640 \* 480,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x67,\_\_CX=0,\_\_DX=0;
- 4 分辨率 720 \* 540,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x68,\_\_CX=0,\_\_DX=0;
- 5 分辨率 800 \* 600,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x69,\_\_CX=0,\_\_DX=0;
- 6 分辨率 640 \* 350,16 色/64 色模式:  
\_\_AX=0x6f05,\_\_BX=0x10,\_\_CX=0,\_\_DX=0;
- 7 分辨率 640 \* 480,16 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x12,\_\_CX=0,\_\_DX=0;
- 8 分辨率 752 \* 410,16 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x60,\_\_CX=0,\_\_DX=0;
- 9 分辨率 720 \* 540,16 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x61,\_\_CX=0,\_\_DX=0;
- 10 分辨率 800 \* 600,16 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x62,\_\_CX=0,\_\_DX=0;
- 11 分辨率 1024 \* 768,16 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x65,\_\_CX=0,\_\_DX=0;

15. 显示卡名称:Video 7 Fastwrite

主芯片集名称:VIDEO 7

所需视频缓冲区容量:256KB

- 1 分辨率 320 \* 200,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x13,\_\_CX=0,\_\_DX=0;
- 2 分辨率 640 \* 400,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x66,\_\_CX=0,\_\_DX=0;
- 3 分辨率 640 \* 350,16 色/64 色模式:  
\_\_AX=0x6f05,\_\_BX=0x10,\_\_CX=0,\_\_DX=0;
- 4 分辨率 640 \* 480,16 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x12,\_\_CX=0,\_\_DX=0;
- 5 分辨率 752 \* 410,16 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x60,\_\_CX=0,\_\_DX=0;
- 6 分辨率 720 \* 540,16 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x61,\_\_CX=0,\_\_DX=0;
- 7 分辨率 800 \* 600,16 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x62,\_\_CX=0,\_\_DX=0;

16. 显示卡名称:Video 7 Fastwrite AD

主芯片集名称:VIDEO 7

所需视频缓冲区容量:512KB

- 1 分辨率 320 \* 200,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x13,\_\_CX=0,\_\_DX=0;
- 2 分辨率 640 \* 400,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x66,\_\_CX=0,\_\_DX=0;
- 3 分辨率 640 \* 480,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x67,\_\_CX=0,\_\_DX=0;
- 4 分辨率 640 \* 350,16 色/64 色模式:  
\_\_AX=0x6f05,\_\_BX=0x10,\_\_CX=0,\_\_DX=0;
- 5 分辨率 640 \* 480,16 色/64 色模式:  
\_\_AX=0x6f05,\_\_BX=0x12,\_\_CX=0,\_\_DX=0;
- 6 分辨率 720 \* 540,16 色/64 色模式:  
\_\_AX=0x6f05,\_\_BX=0x62,\_\_CX=0,\_\_DX=0;
- 7 分辨率 800 \* 600,16 色/64 色模式:  
\_\_AX=0x6f05,\_\_BX=0x62,\_\_CX=0,\_\_DX=0;
- 8 分辨率 1024 \* 768,16 色/64 色模式:  
\_\_AX=0x6f05,\_\_BX=0x65,\_\_CX=0,\_\_DX=0;

17. 显示卡名称:Video 7 VRAM

主芯片集名称:VIDEO 7

所需视频缓冲区容量:512KB

- 1 分辨率 320 \* 200,256 色/262144 色模式:  
\_\_AX=0x6f05,\_\_BX=0x13,\_\_CX=0,\_\_DX=0;
- 2 分辨率 640 \* 400,256 色/262144 色模式:

\_\_AX=0x6f05,\_\_BX=0x66,\_\_CX=0,\_\_DX=0;

3 分辨率 640 \* 480,256 色/262144 色模式:

\_\_AX=0x6f05,\_\_BX=0x67,\_\_CX=0,\_\_DX=0;

4 分辨率 640 \* 350,16 色/64 色模式:

\_\_AX=0x6f05,\_\_BX=0x10,\_\_CX=0,\_\_DX=0;

5 分辨率 640 \* 480,16 色/64 色模式:

\_\_AX=0x6f05,\_\_BX=0x12,\_\_CX=0,\_\_DX=0;

6 分辨率 720 \* 540,16 色/64 色模式:

\_\_AX=0x6f05,\_\_BX=0x61,\_\_CX=0,\_\_DX=0;

7 分辨率 800 \* 600,16 色/64 色模式:

\_\_AX=0x6f05,\_\_BX=0x62,\_\_CX=0,\_\_DX=0;

8 分辨率 1024 \* 768,16 色/64 色模式:

\_\_AX=0x6f05,\_\_BX=0x65,\_\_CX=0,\_\_DX=0

18. 显示卡名称:Video 7 VEGA Deluxe

主芯片集名称:VIDEO7

所需视频缓冲区容量:256 KB

1 分辨率 640 \* 350,16 色/64 色模式:

\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0;

2 分辨率 640 \* 400,16 色/64 色模式:

\_\_AX=0x6f05,\_\_BX=0x12,\_\_CX=0,\_\_DX=0;

3 分辨率 720 \* 410,16 色/64 色模式:

\_\_AX=0x6f05,\_\_BX=0x60,\_\_CX=0,\_\_DX=0

19. 显示卡名称:ATI VGA Wonder

主芯片集名称:ATI

所需视频缓冲区容量:512 KB

1 分辨率 320 \* 200,256 色/262144 色模式

\_\_AX=0x13,\_\_BX=0,\_\_CX=0,\_\_DX=0;

2 分辨率 640 \* 400,256 色/262144 色模式:

\_\_AX=0x61,\_\_BX=0,\_\_CX=0,\_\_DX=0;

3 分辨率 640 \* 480,256 色/262144 色模式:

\_\_AX=0x62,\_\_BX=0,\_\_CX=0,\_\_DX=0;

4 分辨率 800 \* 600,256 色/262144 色模式:

\_\_AX=0x63,\_\_BX=0,\_\_CX=0,\_\_DX=0;

5 分辨率 640 \* 350,16 色/64 色模式:

\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0;

6 分辨率 640 \* 480,16 色/64 色模式

\_\_AX=0x12,\_\_BX=0,\_\_CX=0,\_\_DX=0;

7 分辨率 800 \* 600,16 色/262144 色模式:

\_\_AX=0x54,\_\_BX=0,\_\_CX=0,\_\_DX=0;

8 分辨率 1024 \* 768,16 色/262144 色模式:

\_\_AX=0x65,\_\_BX=0,\_\_CX=0,\_\_DX=0

20. 显示卡名称:TVGA 8900

主芯片集名称:TRIDENT

所需视频缓冲区容量:1024 KB

1 分辨率 320 \* 200,256 色/262144 色模式:

\_\_AX=0x13,\_\_BX=0,\_\_CX=0,\_\_DX=0;

2 分辨率 640 \* 400,256 色/262144 色模式:

\_\_AX=0x5c,\_\_BX=0,\_\_CX=0,\_\_DX=0;

3 分辨率 640 \* 480,256 色/262144 色模式:

\_\_AX=0x5d,\_\_BX=0,\_\_CX=0,\_\_DX=0;

4 分辨率 800 \* 600,256 色/262144 色模式:

\_\_AX=0x5e,\_\_BX=0,\_\_CX=0,\_\_DX=0;

5 分辨率 1024 \* 768,256 色/262144 色模式:

\_\_AX=0x62,\_\_BX=0,\_\_CX=0,\_\_DX=0;

6 分辨率 640 \* 350,16 色/262144 色模式:

\_\_AX=0x10,\_\_BX=0,\_\_CX=0,\_\_DX=0;

7 分辨率 640 \* 480,16 色/262144 色模式:

\_\_AX=0x12,\_\_BX=0,\_\_CX=0,\_\_DX=0;

8 分辨率 800 \* 600,16 色/262144 色模式:

\_\_AX=0x5b,\_\_BX=0,\_\_CX=0,\_\_DX=0;

9 分辨率 1024 \* 768,16 色/262144 色模式:

\_\_AX=0x5f,\_\_BX=0,\_\_CX=0,\_\_DX=0;

10 分辨率 1024 \* 768,4 色/262144 色模式:

\_\_AX=0x60,\_\_BX=0,\_\_CX=0,\_\_DX=0;

11 分辨率 768 \* 1024,16 色/262144 色模式:

\_\_AX=0x61,\_\_BX=0,\_\_CX=0,\_\_DX=0

# 一种实用的 EGA/VGA 屏幕放大技术

□ 朱磊莹

在许多应用程序中,程序员往往用各种方法来美化屏幕,以求画面的生动活泼。将汉字或西文放大显示就是其中的一项重要内容。

对屏幕的实时放大是指从屏幕读取每个象元的值,经过处理后再写回屏幕。利用汇编语言对硬件能完全控制的特点,我们可实现这一想法。途径有两种:一是通过 BIOS 调用,写 INT 10H 0DH 号功能读像素,0CH 号功能写像素。这种方法可移植性好,但速度很慢,对于大量的点操作,简直不胜其繁;二是对显示缓冲区直接读、写,先设置好一些锁存寄存器,再用 MOV 指令即可实现。这种方法是比较理想的,其前提是对显示卡的构造相当了解。

本文介绍的技术是对显示缓冲区的直接读、写,用于 EGA 或 VGA 卡。在此先大略谈一下 EGA/VGA 显示缓冲区。单色、彩色显示卡 MDA 和 CGA 的显示缓冲区在主机中是固定的,与屏幕像素有着——对应的关系。而 EGA/VGA 由于像素信息量大,多至 256K,主机内存不能满足,所以采用一种“位平面压缩”的方式,主机内存从 A0000H~BFFFFH (共 128K)分页对应着 EGA/VGA 卡的 256K 内存。以 EGA/VGA 的模式 10H 为例,主机的 128K 内存使用 A0000H~B0000H 的 64K,分成 2 个页面,每个页面 32K,对应显示缓冲区的 0,1,2,3 四个位面,代表亮色,红色、绿色和蓝色。(即主存的 1 个字节在显存则是 4 个字节映射),具体访问哪个位平面则由图形控制寄存器的映象屏蔽寄存器(用于读)和位平面选择寄存器(用于写)来控制。在模式 10H,显示分辨率为 640×350,每个像素占 1 位(1 Bit),屏幕上一行 640 个像素则需要 80 个字节。下面的图 1 反映了主存(A0000H~B0000H,第 0 页)与像素的对应关系。注意在一个字节中,最高位(Bit 7)表示第 1 个点,如图 2 所示。

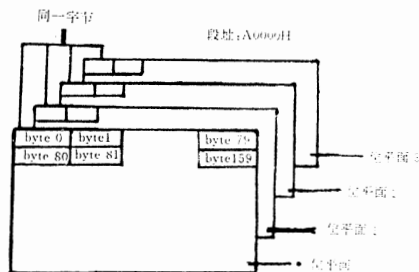


图 1 显存与主存的对应

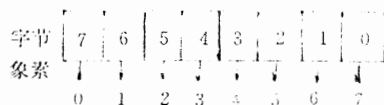


图 2 像素与字节的对应

本文给出了一个演示程序,将“合山矿务局”放大 4 次(其实是每次将 1/4 的屏幕放大至整屏),这样横、纵扩均为 16 倍(可放  $2^n$  倍,  $n$  为次数)。程序中的读、写像素子程序可适用于对屏幕的任一区域,只要修改读、写显示缓冲区的起始地址和写点数目即可。EGA/VGA 有 6 组,共 73 个控制寄存器,本程序只使用了其中的几个,便可达到屏幕放大的效果,显得简单而实用。这几个寄存器均属图形控制寄存器组,读写时(EGA 一般只能写)先向地址寄存器 3CEH 送入偏移量,再向 3CFH 送数即可。用 OUT 指令实现送数。程序的纵扩用 EGA/VGA 的写方式 1(可将屏幕的图形从一处复制到另一处,但必须以一字节为界),将第几行复制到第  $2n$  和  $2n+1$  行。横扩则需要将平行的像素全部读出,再将每个像素写两次,写满一行,使用 EGA/VGA 的读方式 0 和写方式 2。

## 下面是程序清单:

```
程序名:ENLARGE.ASM
用 MASM 编译, LINK 链接
STACK SEGMENT STACK 'STACK'
    DB 200H DUP(?)
STACK ENDS
DATA SEGMENT PUBLIC 'DATA'
    TIT-0          DB '合山矿务局', 24H
    DAT-ADD        EQU offset DAT-BUF
    DAT-BUF        DB 320 * 175 DUP(0)
```

```
ROW-COUNT DB 175
COL-COUNT DB 40
LARG-COUNT DB 4
DATA ENDS

;
CODE SEGMENT PUBLIC 'CODE'
    ASSUME CS, CODE, DS, DATA, SS, STACK
START:
    PUSH DS
    MOV AX, 0
    PUSH AX
; set EGA mode 10H
    MOV AH, 0
    MOV AL, 10H
    INT 10H
; display tit-0
    MOV DX, 0
    MOV BX, 0
    MOV AH, 2
    INT 10H
    MOV AX, DATA
    MOV DS, AX
    MOV SI, OFFSET TIT-0
    CALL DIS-0E
    MOV AH, 0
    INT 16H
BEG-LAR:
    NOP
    CALL PIX-IN      ;读屏幕像素
    CALL X-LA        ;横扩
    MOV AH, 0
    INT 16H
    CALL Y-LA        ;纵扩
    MOV AH, 0
    INT 16H
    MOV AX, DATA
    MOV DS, AX
    MOV AL, 175
    MOV DS:[ROW-COUNT], AL
    MOV AL, 40
    MOV DS:[COL-COUNT], AL
    DEC BYTE PTR DS:[LARG-COUNT]
    JNZ BEG-LAR      ;重复 4 次
    MOV AL, 0
    CALL WRITE-MODE
    MOV AH, 0
    MOV AL, 3
    INT 10H          ;将屏幕方式设置回字符方式
    MOV AH, 4CH
    INT 21H
;
DIS-0E PROC NEAR      ;显示字符串子程序
    NOP
DIS-STR:
    LODSB
    CMP AL, 24H
    JZ FINI5
    MOV BL, 04H
    MOV AH, 0EH
    INT 10H
    JMP DIS-STR
FINI5:
    RET
```



```

DIS-0E ENDP
PIX-IN PROC NEAR ;子程序,将 1/4 的屏幕
;像素读进缓冲区
    MOV AL,0
    CALL READ-MODE
    MOV DI,DAT-ADD
    MOV AX,DATA
    MOV ES,AX
    MOV AX,0A000H
    MOV DS,AX
    MOV SI,0
READ-DATA:
    CALL READ-PIX
    INC SI
    DEC BYTE PTR ES:[COL-COUNT]
    JNZ READ-DATA
    ADD SI,80
    SUB SI,40
    DEC BYTE PTR ES:[ROW-COUNT]
    JZ FINI1
    ADD BYTE PTR ES:[COL-COUNT],40
    JMP READ-DATA
FINI1:
    RET
PIX-IN ENDP
;
X-LA PROC NEAR
;写像素,使用已读进的数据,一个字节写两次
;use write mode 2
    MOV AL,2
    CALL WRITE-MODE ;写方式 2
    MOV AX,DATA
    MOV DS,AX
    MOV AL,80
    MOV DS:[COL-COUNT],AL
    MOV AL,175
    MOV DS:[ROW-COUNT],AL
    MOV SI,DAT-ADD
    MOV AX,0A000H
    MOV ES,AX
    MOV DI,0
WRITE-BYTE:
    CALL WRITE-PIX
    INC DI
    DEC DS:[COL-COUNT]
    JNZ WRITE-BYTE
    DEC DS:[ROW-COUNT]
    JZ FINI2
    MOV AL,80
    MOV DS:[COL-COUNT],AL
    ADD DI,80
    SUB DI,80
    JMP WRITE-BYTE
FINI2:
    RET
;
X-LA ENDP
;
Y-LA PROC NEAR
;纵扩子程序,用写方式 1,将第 N 行复制到第 2N,
;2N+1 行
    MOV AX,0A000H
    MOV ES,AX
    MOV AL,1
    CALL WRITE-MODE
    MOV BH,175
COPY-ROWS:
    MOV AL,BH
    DEC AL
    MOV CX,80
    MUL CL
    MOV SI,AX
    MOV DI,SI
    ADD DI,SI
STORE-2ROW:
    MOV AL,ES:[SI]
    MOV ES:[DI],AL
    MOV ES:[DI+80],AL
    INC DI
    INC SI
    LOOP STORE-2ROW
    DEC BH
    JNZ COPY-ROWS
    RET
Y-LA ENDP
;
READ-PIX PROC NEAR
;读像素,一次读 8 个像素,放入 8 个字节
    MOV CX,8
    MOV BH,10000000B
R1:
    MOV BL,0
    MOV AH,4
R2:
    MOV DX,3CEH ;寄存器地址选择
    MOV AL,4 ;偏移 4 为映象选择寄存器
    OUT DX,AL
    INC DX
    MOV AL,AH
    DEC AL
    OUT DX,AL ;放入位屏蔽数
    MOV AL,DS:[SI]
    SHL BL,1
    AND AL,BH
    JZ R3
    OR BL,00000001B
R3:
    DEC AH
    JNZ R2
    MOV ES:[DI],BL
    INC DI
;
SHR BH,1
LOOP R1
RET
READ-PIX ENDP
;
WRITE-PIX PROC NEAR
    MOV CX,4 ;写 4 次,8 个像素
    MOV BH,11000000B ;位屏蔽数
W1:
    MOV DX,3CEH
    MOV AL,8
    OUT DX,AL ;偏移 8 为位屏蔽寄存器
    INC DX
    MOV AL,BH
    OUT DX,AL ;bit mask
    MOV AL,ES:[DI]
    LODSB
    MOV ES:[DI],AL
    SHR BH,1
    SHR BH,1
    LOOP W1
    RET
WRITE-PIX ENDP
;
WRITE-MODE PROC NEAR
;AL=写方式.0,1,2 或 3
    PUSH AX
    MOV DX,3CEH
    MOV AL,5
    OUT DX,AL
    INC DX
    POP AX
    OUT DX,AL
    RET
WRITE-MODE ENDP
;
READ-MODE PROC NEAR
;AL=读方式.0 或 1
;要设置图形方式寄存器的位 3
    CMP AL,1
    JNE SET-BIT3
    MOV AL,08H
SET-BIT3:
    PUSH AX
    MOV DX,3CEH
    MOV AL,5
    OUT DX,AL
    INC DX
    POP AX
    OUT DX,AL
    RET
READ-MODE ENDP
CODE ENDS
END START

```

(上接第 25 页)

```

for j:=1 to 16 do
    begin
        w:=memw[segaddr+offsetaddr];
        w:=swap(w);
        offsetaddr:=offsetaddr+2;
        for k:=1 to 16 do
            begin ah:= $c;
                if(hi(w)and $80)<>0 then
                    al:=15
                else al:=0;
                intr($10,regs);
                inc(cx);
                w:=w shl 1;
            end;
            inc(dx);
            cx:=y;
        end;
        y:=y+16+space;
        until i>length(s);
    end;
end;
BEGIN
    clrscr;
    gd:=9;
    gm:=2; {VGA 方式}
    initgraph(gd,gm,'');
    hz(5,10,10,'4752,4230,2626,5554');
    {从(10,10)处开始显示“显示汉字”}
    {.....}
    {其他作图语句或显示汉字语句}
    {.....}
    readln;
    closegraph;
END.

```

# 用 C++ 实现交互式实时动画技术

□丁 弘 汪国庆

## 一、引言

近年来,随着三维实体造型设计、三维动画设计及多媒体技术的发展,计算机图形学进入了一个空前繁荣的崭新阶段。看到 3D—Studio 等动画制作软件强大的三维建模功能灵活的动画编辑功能及平滑逼真的视觉效果,我们惊叹的同时,更希望自己的工作中也能使用这些先进的图形技术。

通常 3D—Studio 等软件主要是用于影视等动画的制作及广告的创意等,对于科学研究而言,则存在相当大的局限性。科学研究对图形功能的要求,是科学计算的可视化和各种微观物理现象的动态模拟。需要对数据的迅速反应,且动画时,物体的运动应符合一定的物理规律(或规律的简化描述),同时图形软件应具有更多的专业性和知识性。

我们从事的分子动力学模拟,是用计算机来模拟物质的微观运动,进而得出一些有意义的物理景观。我们用 C++ 语言开发并逐步完善的图形软件,强化了该研究的图形功能。该软件能够直接使用和组织原始数据文件中的数据,形成三维实体和完成动画过程,并通过一组三维建模程序、消隐程序、光线跟踪程序等,产生效果逼真的三维实体与动画。它其中还包括一个交互式实时动画模块,能在计算过程中交互实时地观察物体的运动情况。本文旨在介绍实时动画的原理和实现方法,并给出一个二维交互式实时动画示例程序的全部 C++ 源代码。

## 二、实时动画的概念、

### 原理及特点

在个人机上产生计算机动画的方法,从基于硬件支持的角度考虑,可分为帧动画和实时动画两类。

帧动画,也称为页动画。动画程序先

建立许多帧全屏幕图象,并将每帧图象存入单独的页缓冲区中(可以是 RAM,也可以是磁盘),当所有图象全部建完以后,再用特定的程序将这些图形页,以适当的速率和顺序,“放电影”般地一一显示到屏幕上,产生动画效果,此时微处理器集中进行动画处理,3D—Studio 的动画,采用的便是帧动画的方式。这种方法的动画速度快,动画可以制作得十分精致,细节的表达也可十分清楚。但是这类动画处理的是许多帧全屏幕的图象,占用内存和磁盘空间十分庞大(例如一个 3DS 制作的 20 秒的电视广告片,占用的磁盘空间可达 10MB 以上),动画制作的时间也较长,更为主要的是这种动画,一经制成,画面便不能有丝毫的改动,不能对数据的变化实时地作出反应,更不具有交互的性能。

与帧动画在启动动画序列之前绘制并保存好所有的图形不同,实时动画是在动画过程中实时绘制图象的。在实时动画过程中,微处理器的时间划分为图象建立时间和图象动画时间。它至少需要两个图形页(page0 和 page1),两个图形页分别被设置为活动页(active page,所有绘制图形的任务,均在活动页中进行)和显示页(Visual page,用于在显示器上显示图形),在 C 和 C++ 中,可用 setactivepage() 和 setvisualpage() 这两个函数分别设置两个图形页。当 Page0 设置为活动页绘制图形时,Page1 被设置为显示页显示,当 Page0 中的图形绘成后,将 Page0 转换为显示页显示图形,而将 Page1 设置为活动页绘制新的图形。如此通过一幅幅图形的实时绘制与显示便可实现实时动画的效果。由此可见,实时动画的速度远不及帧动画,其速度取决于图象的复杂程度,所以实时动画一般用于不太复杂的图形。对于三维的构型,通常只显示三维物体的网络框架,而不进行表面着色,或只进行简单的着色。

另外,对于背景复杂的动画,可以建立一个通用的背景页,动画的每一帧均可选择背景页的内容复制到隐藏页的图象中,然后将完整的图象转换到屏幕上显示,这种方法亦可提高实时动画的速度。最后可适当降低显示分辨率来提高动画速度。

实时动画的速度虽慢,但却是最为灵活的方法,可交互式地产生动画。这种交互性体现在三个方面:

1. 动画过程中,计算机可以实时地进行实体采样,在动画过程中,观测者并不知道下一时刻图形会发生什么样的变化,产生如同观测实验般的效果。
2. 动画过程中,可以通过键盘、鼠标器等来移动、旋转物体。
3. 动画过程中,可以通过键盘、鼠标器等来移动摄像机(视点),以看清各个角度及位置物体的景观。

## 三、用 C++ 语言开发图形软件

C++ 语言是一种仿真语言,它依赖于面向对象的程序设计(OOP),由于 C++ 语言与 C 语言完全兼容,且扩展了库函数和关键字,及增加了面向对象编程的高效性能,正逐渐成为软件开发,尤其是图形软件开发中流行的程序设计语言。

封装,是最为体现 C++ 面向对象编辑特色的概念。它认为所谓对象,即是指由一组数据和对数据进行的操作所组成的实体。封装便是指定义这一对象的结构类型,这种结构类型在 C++ 中称为类(class),类和 C 中的 struct、union 不同,在 C++ 中,数据和对数据进行操作的函数,均可成为类的成员,C++ 将数据和函数同等看待,而不考虑它们之间的差别。每一对象,可以有许多不同的实例,C++ 为这些不同的实例,提供了同一可执行的代码(这一代码甚至可以为该类的继承类所重用,有关继承的概

念,亦是 C++ OOP 编程的特色之一)。封装隐藏了许多复杂性,它意味着可在实体外进行处理,包括编程,调试和修改,这也是最合乎人类处理问题思维习惯的编程方式。例如,我们在驾驶汽车时,操纵的是方向盘、刹车闸和油门,既不直接操纵齿轮和曲轴,也不关心它们在驾驶过程中发生了哪些变化。(有关类的操作,请详见示例程序。)在示例程序中,定义了两个类 Physical Display 类和 Viewport 类。Physical Display 用来管理屏幕,包括设置图形模式、设置画笔颜色、填充模式、以及页操作等。Viewport 类提供低级图形输出,包括画直线、圆、多边形,视口操作(以上示例程序中不用)以及位块传输操作,缺省字体输出

```

/* =====
示例程序 ANIMA.CPP,支持 VGA 和 EGA 显示适配器,
显示交互式实时动画效果
===== */
/* =====常量===== */
const int dFAIL=0; //判别量
const int dEMPTY=0;
const int dYES=1; const int dNO =0;
const int dLT __ARROW=75; //扩展键代码
const int dRT __ARROW=77; const int dDN__ARROW=80;
const int dUP __ARROW=72; const int dALT__X=45;
const int dF1=59; const int dF2=60
const int dF3=61; const int dF4=62
const int dF5=63; const int dF6=64
const int dF7=65;
/* =====include 文件===== */
#include <alloc.h>
#include <bios.h>
#include <process.h>
#include <iostream.h>
#include <dos.h>
#include "LIB.HPP"
/* =====函数原型===== */
static void dStartup(void); //初始化图形模式
static void dKeyboard(void); //检测键盘响应
static void dSetBitMem(void); //申请位块内存
static void dDrawSphere(void); //画球
static void dQuit__Pgm(void); //中止程序
static void dPurge(void); //清除键盘缓冲区
static void dSound(int, double); //误操作警告
static void InteractiveAnimation(); //交互式动画
/* =====全局变量声明===== */
static int CharWidth=0, CharHeight=0; //字符参数
static char KeyCode=0, KeyNum=0; //击键参数
static char SolidFill[]={255,255,255,255,255,255,
255,255}; //100%填充
static int X__Res=0, Y__Res=0; //屏幕分辨率
static char Prompt1[]="F1-F7=Change Color, Arrow Key--Control
Moving";
static char Prompt2[]="ALT+X=Quit";
static float xSCent, ySCent, R=15.0, DeltaDist=4.0;
char far * Array; //指向位图所占内存区
static int ActPage=0, VisPage=1;
static float xObj, yObj;
static int SphereColor=4;
PhysicalDisplay Display;
//创建—PhysicalDisplay 类实例
Viewport Viewport(&Display);
//创建—Viewport 类实例
/* =====函数定义=====
=====主函数===== */

```

等。如果为了构筑逼真的三维实体(如球体,圆柱体等),可以建立一个三维建模类 3DModel,它可以提供坐标变换,典型实体建模,实体的旋转、移动、伸缩等操作,以及更为复杂的平滑阴影着色功能。还可以建立 Animation 类,它提供符合各种物理规律的运动函数,以产生真实的动画效果。用户可以在一个框架上,方便地由局部到整体,由简单到复杂,开发自己潜力无穷的 C++ 图形应用软件。

总之,利用 C++ 的封装、继承等技术、形成一种崭新的、合乎人类思维习惯的程序设计方式,加上 C++ 其他的扩展功能,以及流行的 C++ 软件功能强大的编译器和工具箱,使 C++ 语言

日益深入人心,显示出旺盛的生命力。

#### 四、示例程序介绍

示例程序 ANIMA.CPP 在 Borland C++ 2.0 上编制,必须采用 Large 模式编译。编译时,打开 Project/New,按 Ins 键加入 ANIMA.CPP 程序和类实现文件 LIB.CPP,按 Ctrl+F9 进行 Build ALL。

该示例程序的运行:按 F1~F7 改变园球的颜色,按 → ← ↑ ↓ 键可实时控制动画过程中园球移动的方向,按 ALT+X 键退出。按所有未定义键,将发出蜂鸣声。

本程序在 486DX2-50 机上运行,实时动画效果很好。

```

main()
{
dStartup();
Display.Init2D(0,0,X__Res-1,Y__Res-1);
Display.SetHue(9);
Display.SetFill(SolidFill,9);
Viewport1.DrawPand(0,0,X__Res,-1,Y__Res-1);
Viewport1.PutText(4,15,58,Prompt1);
Viewport1.PutText(6,34,58,Prompt2);
Display.SetHue(15);
Viewport1.DrawBorder(0,0,X__Res-1,Y__Res-1);
Display.SetHue(7);
Display.BackUp(); //背景保存至隐藏页
dSetBitMem();
xSCent=X__Res/2; ySCent=Y__Res/2;
InteractiveAnimation();
dQuit__Pgm();
}
/* =====画球===== */
static void dDrawSphere(void)
{ Viewport1.PutPSET(xObj-R,yObj-R,Array);
Display.SetWritePage(ActPage);
Display.SetDisplayPage(VisPage);
Display.SetFill(SolidFill,SphereColor);
fillellipse(xSCent,ySCent,R,R);
ActPage=VisPage; VisPage=ActPage;
xObj=xSCent; yObj=ySCent;
Display.Restore(); //恢复隐藏页内容
return;
}
/* =====交互式动画===== */
static void InteractiveAnimation(void)
{ PRIMARY__LOOP:
bKeyboard();
if(KeyCode==0) goto PRIMARY__LOOP;
if(KeyCode!=2){ dSound(300,.2);
dPurge(); goto PRIMARY__LOOP; }
EXTENDED__KEY:
switch(KeyNum)
{ case dF1: SphereColor=0; break;
case dF2: SphereColor=1; break;
case dF3: SphereColor=2; break;
case dF4: SphereColor=3; break;
case dF5: SphereColor=4; break;
case dF6: SphereColor=5; break;
case dF7: SphereColor=6; break;
LT__LOOP: xSCent-=DeltaDist;
if(xSCent<=0) xSCent=xObj;
dDrawSphere(); dKeyboard();
if(KeyCode==0) goto LT__LOOP;
go to EXTENDED__KEY;
case dRT__ARROW:

```

```

RT __ LOOP; xScnt += DeltaDist;
    if (xScnt >= X __ Res) xScnt = xObj;
    dDrawSphere(); dKeyboard();
    if (KeyCode == 0) goto RT __ LOOP;
    goto EXTENDED __ KEY;
case dUP __ ARROW;
UP __ LOOP; yScnt -= DeltaDist;
    if (yScnt <= 0) yScnt = yObj;
    dDrawSphere(); dKeyboard();
    if (KeyCode == 0) goto UP __ LOOP;
    goto EXTENDED __ KEY;
case dDN __ ARROW;
DN __ LOOP; yScnt += DeltaDist;
    if (yScnt >= Y __ Res) yScnt = yObj;
    dDrawSphere(); dKeyboard();
    if (KeyCode == 0) goto DN __ LOOP;
    goto EXTENDED __ KEY;
case dALT __ x; return;
default; dSound(300, .2);
    dPurge(); goto PRIMARY __ LOOP;
}
/* ===== 初始化位图内存 ===== */
static void dSetBitMem(void)
{
    unsigned Size;
    Size = imagesize(300, 200, 300 + 2 * R, 200 + 2 * R);
    Array = (char far *)farmalloc(Size);
    getimage(300, 200, 300 + 2 * R, 200 + 2 * R, Array);
    return;
}
/* ===== 中止程序 ===== */
static void dQuit __ Pgm(void)
{
    Display.ShutDownGraphics(); exit(0);
}
/* ===== 图形模式 ===== */
static void dStartup(void)
{
    int Mode = Display.SetupMode();
    if (Mode != 0)
    {
        // EGA __ ECD __ mode; 640 * 350 * 16 __ clolr, 80 * 43 char
        X __ Res = 640; Y __ Res = 350;
        CharWidth = 8; CharHeight = 8; return;
    }
    cout << "\n This C++ program requires a"
    cout << "\n VGA or EGA graphic adapter."
    exit(-1); // 中止程序, 提示信息
}
/* ===== 键盘申请 ===== */
static void dKeyboard(void)
{
    union Anyname
    {
        int RawCode; char Code[3]; } Keystroke;
        char TempKey = 0;
        if (bioskey(1) == dEMPTY) { KeyCode = 0; return; }
        Keystroke.RawCode = bioskey(0);
        TempKey = Keystroke.Code[0];
        if (TempKey != 0) { KeyCode = 1; KeyNum = TempKey; return; }
        if (TempKey == 0)
        {
            KeyCode = 2; KeyNum = Keystroke.Code[1];
            return;
        }
    }
}
/* ===== 清除键盘缓冲 ===== */
static void dPurge(void)
{
    do dKeyboard(); while (KeyCode != 0);
    return;
}
}
/* ===== 声音警告 ===== */
static void dSound(int Hertz, doubled Dur)
{
    sound(Hertz); delay(Dur * 1000);
    nosound(); return;
}
}
/* ===== end of file ANIMA. CPP ===== */
/* =====
LIB. CPP 源代码, PhysicalDisplay 类
和 Viewport 类的实现文件
===== */
/* ===== 常量 ===== */
const int dFAIL = 0; const int dYES = 1;
const int dNO = 0;

```

```

#include <dos.h>
#include <alloc.h>
#include <mem.h>
#include "LIB. HPP"
/* ===== PhysicalDisplay 类: 构造函数 ===== */
PhysicalDisplay::PhysicalDisplay()
{
    ObjectReady = dYES; RAMpage = dNO;
    TextWidth = 0; TextHeight = 0;
    static char SolidFill[] =
        {255, 255, 255, 255, 255, 255, 255, 255};
    Left = 0; Top = 0;
    Right = 0; Bottom = 0;
    Length = 0;
}
/* ===== PhysiacalDisplay 类: 析构函数 ===== */
PhysicalDisplay::~PhysicalDisplay()
{
    ObjectReady = dNO;
}
/* ===== 初始化图形 ===== */
int PhysicalDisplay::SetupMode(void)
{
    int graphics __ adapter = EGA, graphics __ mode = EGAHI;
    int ErrorCode = graphresult();
    if (ErrorCode != grOK) return 0;
    initgraph(&graphics __ adapter, &graphics __ mode, "");
    settextstyle(0, 0, 1);
    return dYES;
}
void PhysicalDisplay::Init2D(int x1, int yf1,
                             int x2, int y2)
{
    Left = x1; Top = yf1; Right = x2; Bottom = y2;
    Length = 28000; TextWidth = 8; TextHeight = 8;
    return;
}
void PhysicalDisplay::ShutDownGraphics(void)
{
    if (RAMpage == dYES) RAMpage = dNO;
    cleardevice(); closegraph(); return;
}
/* ===== 设置画笔 ===== */
void PhysicalDisplay::SetHue(int hue)
{
    setcolor(hue); return;
}
void PhysicalDisplay::SetFill(char * pat, int hue)
{
    setfillpattern(pat, hue); pat ++;
    if (* pat == SolidFill[1])
        setfillstyle(SOLID __ FILL, hue); return;
}
/* ===== 页操作 ===== */
void PhysicalDisplay::SetwritePage(int page)
{
    Setactivepage(Page); return;
}
void PhysicalDisplay::SetDisplayPage(int Page)
{
    setvisualpage(page); return;
}
/* ===== Viewport 类: 析构函数 ===== */
Viewport::Viewport(physical(Display * Caller)
{
    GenDisplay = Caller;
    ViewportReady = dYES;
}
}
/* ===== Viewport 类: 析构函数 ===== */
Viewport::~Viewport()
{
    ViewportReady = dNO;
}
/* ===== 低级作图函数 ===== */
void Viewport::DrawBorder(int x1, int yf1,
                           int x2, int y2)
{
    rectangle(x1, yf1, x2, y2); return;
}
void Viewport::DrawPanel(int x1, int yf1,
                           int x2, int y2)
{
    bar(x1, yf1, x2, y2); return;
}
/* ===== 位图操作函数 ===== */
void Viewport::PutPSET(int x, int y,
                        char far * blk)
{
    putimage(x, y, blk, COPY __ PUT; return;
}
/* ===== 文本操作函数 ===== */
void Viewport::PutText(int row, int col,
                        int clr, char * tptr)

```

```

{ int TCrow, TCcol, tempclr;
  TCcol=GenDisplay->TextWidth * (col-1);
  TCrow=GenDisplay->TextHeight * (row-1)
  tempclr=getcolor(); setcolor(clr);
  outtextxy(TCcol, TCrow, tptr);
  setcolor(tempclr); return;
}
/* === 隐藏页操作 === */
char far * PhysicalDisplay::InitHiddenPage(Void)
{ char far * vptr;
  vptr=(char far *)farmalloc(Length);
  return vptr;
}
void PhysicalDisplay::BackUp(void)//EGA __ mode
{ outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)0);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)1);
  movedata(0xa000, 0x0000, 0xa800, 0x0000, Length);
  outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)1);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)2);
  movedata(0xa0000, 0x0000, 0xa800, 0x0000, Length);
  outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)2);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)4);
  movedata(0xa000, 0x0000, 0xa800, 0x0000, Length);
  outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)3);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)8);
  movedata(0xa000, 0x0000, 0xa800, 0x0000, Length);
  outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)0);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)15); return;
}
void PhysicalDisplay::Restore(void)
{ outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)0);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)1);

  movedata(0xa800, 0x0000, 0xa000, 0x0000, Length);
  outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)1);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)2);

  movedata(0xa800, 0x0000, 0xa000, 0x0000, Length);
  outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)2);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)4);

  movedata(0xa800, 0x0000, 0xa000, 0x0000, Length);
  outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)3);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)8);

  movedata(0xa800, 0x0000, 0xa000, 0x0000, Length);
  outportb(0x03ce, (unsigned char)4);
  outportb(0x03cf, (unsigned char)0);
  outportb(0x03c4, (unsigned char)2);
  outportb(0x03c5, (unsigned char)15); return;
}

```

```

/* === end of file LIB.CPP === */
/* ===== */
LIB.HPP 源代码, PhysicalDisplay 类
和 Viewport 类的声明文件
=====
*/
#if ! defined(__LARGE__)
#error MUST USE LARGE MEMORY MODEL
#endif
#include<graphics.h>
class PhysicalDisplay//对显示器操作
{private:
  int ObjectReady;//判明对象是否初始化
  int RAMpage;//判明隐藏页是否在 RAM 中
  char SolidFill[8];//填充模式
  int TextWidth;//字符宽, 高
  int TextHeight;
  int Left;//视区坐标
  int Top;
  int Right;
  int Bottom;
  unsigned int Length;//位图所占字长
  char far * BitMap0;//保存隐藏页至 RAM
  char far * BitMap1;
  char far * BitMap2;
  char far * BitMap3;
  union//计算地址的联合体
  {struct{unsigned int A1;
    unsigned int A2;}Address;
    char far * FarAddress;}U1;
  //隐藏的位图之段地址与偏移地址
  unsigned int Seg1;//
  unsigned int Seg2;
  unsigned int Seg3;
  unsigned int Seg4;
  unsigned int off1;
  unsigned int off2;
  unsigned int off3;
  unsigned int off4;
  char far * InitHiddenPage(void);
  //初始化隐藏页
public:
  PhysicalDisplay();//构造函数
  ~PhysicalDisplay();//析构函数
  friend class Viewport;//友元类
  void Init2D(int, int, int, int);
  int SetupMode(void);//建立图形模式
  void ShutDownGraphics(void);//返回文本模式
  void setHue(int); //设置画笔颜色
  void SetFill(char *, int); //设置填充模式
  void BackUp(void);
  void Restore(void);
  void SetWritePage(int);//设置活动页
  void SetDisplayPage(int);//设置显示页
};
class Viewport//对视区和低级图形的操作
{private:
  int ViewportReady;
  PhysicalDisplay * GenDisplay;
public:
  Viewport(PhysicalDisplay *)
  ~Viewport();
  void DrawBorder(int, int, int, int);//画边框
  void DrawPanel(int, int, int, int);//填充一区域
  void PutText(int, int, int, char *)//显示文本
  void PutPSET(int, int, char far *);
};
/* end of file LIB.HPP */

```



# XENIX 下的图形软件包 CGI

□ 张 荣

XENIX 系统是 UNIX 系统在以 intel 芯片为主 CPU 的微机上的实现,是一个功能很强的多用户、多任务操作系统,由于它能充分发挥高档微机的软硬件资源,具有安全保密性好、无病毒等优点,近年来在国内得到了广泛应用。然而,人们发现 XENIX 下的应用软件中一般都不具有绘图功能,甚至 BASIC 下的 preset 语句也不能使用。这给在 XENIX 下进行二次开发的用户带来了很大困难。

Sco 公司提供的 XENIX 下的图形界面软件 CGI (Computer Graphics Interface) 提供了独立于设备的驱动程序,隐藏了低级图元操作和设备特性,为 XENIX 下的用户作图提供了方便。

## 一、CGI 的运行环境及主要文件

CGI 运行的硬件环境:

CPU: intel80286或80386及其以上;

显示器: CGA,EGA,VGA 等;

打印机等输出设备;

软件环境:XENIX System V(在 Sco XENIX System V Release 2.2.2以下版本中使用,需在安装显示器驱动程序前先安装 Sco XENIX System V Link Kit)。

使用 CGI 的用户必须在其 .profile 文件(sh)或 .login 文件(csh)中对运行环境进行设置(以下假定在标准 sh 下的设置):

(1)设置设备驱动程序所在目录的路径

```
CGIPATH=/usr/lib/cgi
```

```
export CGIPATH.
```

(2)设置同设备驱动程序相对应的设备逻辑名

```
CGIDISP=驱动程序名(如 ega vga16...)
```

```
CGIPRNT=驱动程序名(如 epson100)
```

(在 CGI1.0 中 CGIPATH 以 VDIPATH 代替,CGIDISP 以 DISPLAY 代替,CGIPRNT 以 print\_driver 代替)。

如果 CGIDISP 中指定的设备是一个打印机也可用定义变量 CGIPRNT 来代替。如果运行图形文件时选用了 -P 选项,则要用到 CGIPRNT 环境变量。

CGIPRNT 环境变量也可指定显示设备,但必须与图形文件中 Savin[11]~Savin[18]中的定义保持一致。

其他一些环境变量,如定义缓冲区大小变量、字型环境变量等一般不用,在此不做介绍。

图形界面软件包 CGI 主要包括以下几类文件:

显示器、打印机、绘图机等设备驱动程序;

测试程序;

公共变量及函数库程序;

字型设置程序等等。

CGI 软件包安装后,其大部分文件抽取到 /usr/lib 目录下。

## 二、CGI 的坐标系统

CGI 是基于 VDC 系统(虚拟设备坐标系统)设计的。在理论上 VDC 空间是一个二维笛卡尔坐标系,可以有任意的精度并无限扩展。而实际中应用的是它的一个子集,其范围可用 16 位整型数字表示为 -32768 到 32767。一般 CGI 要将 VDC 系统上的点转换到合适的设备坐标上才能正确地显示出我们所需的图形。这种坐标变换方式有四种:

方式 0 全屏方式:将 0\_32767 映射到 X 和 Y 轴上,屏幕左下角坐标为 (0,0)。如果设备本身在 X 轴和 Y 轴上不具有相同数目的像素,则图形会发生畸变,如:将圆显示成椭圆。因此,这种方式对设备的依赖性较强。

方式 1 保持比率方式:将 0\_32767 映射到显示设备的长轴上;而将 VDC 空间的一个子集映射到短轴上,具有同样 VDC 长度的线沿 X 轴和 Y 轴映射后,其物理长度相同,而不依赖于设备特性。

方式 2 设备单位方式:所有坐标都以设备的物理单位说明,依赖于设备的特性。

方式 3 短轴方式:同方式 1 相似。但将 0\_32767 映射到设备的短轴上。

CGI 使用设备驱动程序的返回信息转换 VDC 坐标成设备的物理坐标。

## 三、CGI 的函数

CGI 是一个独立于图形设备的接口软件。它提供了 133 个图形函数。具体分为以下几类:

控制函数:共 31 个,主要完成对 CGI 基本设备的控制、图形设备的初始化及结束图形操作。

位映像函数:共有 4 个,提供了处理光栅图象的能力,主要完成对位映像的创建,拷贝选择及删除。

输入输出函数:输入函数共 10 个,主要完成设置输入范围,激活输入设备等操作;输出函数共 17 个,主要完成对图形原语及各种 CGI 错误信息的输出。

属性函数:共有 37 个,主要完成对图形原语的颜色、大小、宽度、模式等属性控制的功能。

询问函数:共有 34 个。主要获取包括设备能力和当前属性在内的图形当前状态信息。

使用高级语言(如 C)或汇编语言调用 CGI 提供的这五类函数,便可以得到所需的各种图形。例如统计分析用的折线图,直方图和园饼图等,使用起来方便灵活,因此在实际中有很高的实用价值。

## 四、CGI 的正文

CGI 可以方便灵活地创建和操作正文。其正文方式有：原始正文、图形正文和光标正文三种。原始正文可在指定位置显示正文串，可产生正式文档，适用于开发图形和正文混合的应用程序；图形正文也可在指定位置显示正文串，能够旋转和缩放，并有多钟字型，适用于控制象其它图形属性的正文属性；光标正文只能在固定矩形区显示正文串，不能同图形混合使用，适用于开发屏幕菜单和表格应用软件。

CGI 对正文的控制方式有两种：图形方式和光标寻址方式。在图形方式下，CGI 将正文视为图形原语进行处理；在光标寻址方式下，正文为典型的 24 \* 80 的屏幕格式。

## 五、利用 CGI 函数作图的基本步骤

假定已正确设置了 CGI 的运行环境。

### (1) 设置图形控制参数

利用 CGI 函数作图应首先定义 CGI 函数中常用的变量，例：devhandle, savin[18], savary[66] 等等；然后打开 CGI 工作站，定义所用设备类型。此操作在需要时装入设备驱动程序，返回有关设备能力的各种参数。打开工作站之前必需视自己的需要对 savin[18] 数组赋值，定义坐标变换模式、线的类型、线的颜色序号、图形正文文字型号及颜色序号、填充内部式样，填充式样序号及填充颜色序号。

savin[11] \_\_ savin[18] 中存放工作站标识名，如“CGIDISP”，必须与 CGI 环境变量的设置保持一致。devhandle 中存放返回的和工作站标识名相关的设备描述符，savary 数组存放返回的有关设备能力的各种参数。

### (2) 定义图形原语的属性

在打开工作站时已设置了图形原语的属性，但在实际应用中往往需要改变某些属性，以便使所作图形更加美观、醒目。下面分别介绍几种常用属性的设置。

#### ① 色彩的设置

色彩的设置包括背景色与图形原语色的设置两类，色彩的可选值范围受所用显示设备及设备驱动程序的限制。例如：若采用 CVGA 显示卡，启动 vga16 驱动程序，则色彩一般取值为 0~7，当然，也可根据灰度值设置颜色表增加色彩的种类，最多可达 256 种。图形原语色彩的设置比较灵活，可统一设置，也可分别设置线、填充区、正文等图形原语的色彩。

#### ② 填充区属性的设置

填充区的属性除色彩外，还包括内部式样和填充式样两种。CGI 填充内部式样有 5 种：空心、实心、图案、影线和位映像，在选择图案方式下，填充式样有 33 种，在影线方式下，填充式样有 6 种供选用。

#### ③ 线型、线宽的设置

CGI 对所有设备提供了 6 种预定义线型，也可由用户自行定义；线的宽度以 VDC 坐标值来表示。

CGI 提供的属性设置函数非常丰富，除以上介绍的三种之外，还包括对标记、正文等多种属性的设置，在此不一一介绍。

### (3) 图形输出

在图形输出设备（如显示器）上输出图形，可以通过 CGI 提供的图形原语输出函数来完成。点的位置由 VDC 坐标系中的 (x, y) 坐标来确定。任何一种图形原语，都需要确定其关键参数，例如：画一个圆，需要指定园心和半径；椭圆则需指定

中心点、长、短轴半径；矩形区域由对角线端点的坐标来定义。CGI 提供的图形原语还包括弧、椭圆弧、扇形片、折线、长条、标记、网格及正文等等，通过对色彩和填充区方式等属性的不同设置可以由这些基本图元组合成各种各样所需要的复杂图形。

### (4) 编译链接

编写好的 CGI 作图程序需经过编译链接，将 C 语言联编库 Libccgi.a 链入。libccgi.a 有三种模式：大模式 Llibccgi.a、中模式 Mlibccgi.a 和小模式 Slibccgi.a。一般 286 机器采用大模式，386 机器采用小模式。C 库 Libccgi.a 必须在目录 /usr/lib 下。例如：在 386 系统下编译作图程序 tj.c 的命令为：

```
cc -i -O tj tj.c -lccgi
```

编译正确，则产生可执行的作图文件 tj。

## 六、CGI 作图示例

笔者利用 CGI 函数进行财务状况分析，绘制直方图、折线图、园饼图，并取得了良好效果。限于篇幅，这里仅给出作统计直方图的程序，并加入适当注释。程序主要包括以下四部分：

- (1) 系统初始化。定义变量及图形控制参数；取得数据并作相应处理，以实际数据中的最大值为最大坐标值将其转换为 VDC 坐标系的数值；
- (2) 绘制带刻度的网状坐标平面；
- (3) 绘制直方图；
- (4) 关闭工作站，结束图形操作。

```
#include <stdio.h>
#define int      short
#define BOTTOM 32767                                // 屏幕上端最大
                                                    // VDC 坐标值

#define CENTER 12000
#define M      3000                                // 坐标原点的横坐标
#define B      1000
int devhandle, savin[18], savary[66];
main ()
{
    FILE *fp, *fopen ( );
    int coun[60], j, n, k, i;
    float tj[60], cj[60], s, s1, s2;
    int ptsin[10];
    char sg[60][6], cstrng[80];
    savin[0]=0;                                       // 定义图形控制参数
    savin[1]=1;
    savin[2]=1;
    savin[3]=3;
    savin[4]=1;
    savin[5]=1;
    savin[6]=1;
    savin[7]=0;
    savin[8]=0;
    savin[9]=1;
    savin[10]=1;
    savin[11]='C';
    savin[12]='G';
    savin[13]='I';
    savin[14]='D';
    savin[15]='I';
    savin[16]='S';
    savin[17]='P';
    savin[18]=' ';
    fp=fopen("tj.txt", "r");                        // 打开数据文件
    i=0;
    s=0.0;
    while (!feof(fp))                               // 取得数据并取得最大值
    {
```

```

fscanf(fp, "%s%f\n", sg[i], &tj[i]);
if (tj[i] > s)
{
    s = tj[i];
    k = i;
};
i = i + 1;
};
s2 = 0.1;
coun[k] = 0;
while (tj[k] > (s2 * 10.0))
{
    s2 = s2 * 10.0;
    coun[k] = coun[k] + 1;
};
n = i;
s1 = s / 32;
s = 31000 / s;
for (i = 0; i <= n; i++) // 转换为 VDC 坐标
    cj[i] = tj[i] * s;
    v_opnwk(savin, &devhandle, savary); // 打开工作站

    ptsin[0] = M - 80; // 定义线端点坐标
    ptsin[1] = B;
    ptsin[2] = M - 80;
    ptsin[3] = 32200;
    v_pline(devhandle, 2, ptsin); // 绘制横坐标轴
    ptsin[2] = 32000;
    ptsin[3] = B;
    v_pline(devhandle, 2, ptsin);
    vst_color(devhandle, 2);
    v_gtext(devhandle, 0, B, "0"); // 标记坐标原点
    ptsin[0] = M - 80;
    ptsin[1] = 32200;
    ptsin[2] = 32000;
    ptsin[3] = 32200;
    v_pline(devhandle, 2, ptsin);
    // 标记最大刻度值
    v_gtext(devhandle, 0, 32000, ecvt((double)(int)tj[k], coun[k]));
    if (tj[k] < 1.00)
        v_gtext(devhandle, 0, 32000, "0");
    if (tj[k] - (int)tj[k] > 0.001)
    {
        v_gtext(devhandle, 1400, 32000, ".");
    }
    if ((tj[k] - (int)tj[k]) * 10 < 1.0)
    {
        v_gtext(devhandle, 1700, 32000, "0");
        if (tj[k] * 100.0 - (int)(tj[k] * 100.0) < 1.0)
            v_gtext(devhandle, 2100, 32000, "0");
        else
            v_gtext(devhandle, 2100, 32000, ecvt((double)(tj[k] - (int)tj[k]), 1));
    }
    else
        v_gtext(devhandle, 1700, 32000, ecvt((double)(tj[k] - (int)tj[k]), 2));
};
ptsin[0] = 32000;
ptsin[1] = B;
v_pline(devhandle, 2, ptsin);

vsl_color(devhandle, 6); // 定义线的颜色
vsl_type(devhandle, 7); // 定义线型
vsl_width(devhandle, 0); // 定义线宽
for (i = 1; i <= 32; i++) // 绘制网格横向坐标线
{
    ptsin[0] = M;
    ptsin[1] = (int)(31000.0 / 32.0 * i + B;
    ptsin[2] = 32000;
    ptsin[3] = ptsin[1];
    v_pline(devhandle, 2, ptsin);

    if ((i % 5) == 0)
    {
        s2 = 0.1;
        coun[i] = 0;
        while ((s1 * i) > (s2 * 10.0))
        {
            s2 = s2 * 10.0;
            coun[i] = coun[i] + 1;
        };
        s2 = s1 * i;
        // 标记 Y 轴刻度
        v_gtext(devhandle, 0, ptsin[1], ecvt((double)(int)s2, coun[i]));
        if (s2 < 1.00)
            v_gtext(devhandle, 0, ptsin[1], "0");
        if ((s1 - (int)s1) > 0.001)
        {
            v_gtext(devhandle, 1400, ptsin[1], ".");
        }
        if ((s2 - (int)s2) * 10 < 1.0)
        {
            v_gtext(devhandle, 1700, ptsin[1], "0");
            if (s2 * 10.0 - (int)(s2 * 10.0) < 1.0)
                v_gtext(devhandle, 2100, ptsin[1], "0");
            else
                v_gtext(devhandle, 2100, ptsin[1], ecvt((double)(s2 - (int)s2), 1));
        }
        else
            v_gtext(devhandle, 1700, ptsin[1], ecvt((double)(s2 - (int)s2), 2));
    }
};
};
if ((32000 - M) / n < 1000) // 定义直方图长条宽度
    s = (32000 - M) / n;
else
    if ((32000 - M) / n < 2000)
        s = (32000 - M) / n;
    else
        s = 2000;
for (i = 0; i < n; i++) // 绘制网格纵向坐标线
{
    ptsin[0] = (int)(s * (float)i) + M;
    ptsin[1] = B;
    ptsin[2] = ptsin[0];
    ptsin[3] = 32200;
    v_pline(devhandle, 2, ptsin); // 输出纵向坐标线
    v_gtext(devhandle, ptsin[0], 0, sg[i]); // 标记 X 轴
};
k = 0;
for (i = 0; i < n; i++) // 绘制直方图
{
    k = k + 1;
    if (k > 5) k = 1;
    j = (i + 1) % 7 + 1;
    vsf_interior(devhandle, 3); // 设置填充区内部式样
    vsf_style(devhandle, k); // 设置填充式样
    vsf_color(devhandle, j); // 设置填充区色彩
    ptsin[0] = (int)(i * s) + M; // 定义直方图长左下角
    ptsin[1] = B + 100; // 和右上角的 VDC 坐标
    ptsin[2] = ptsin[0] + (int)s;
    ptsin[3] = (int)cj[i] + B + 100;
    v_bar(devhandle, ptsin); // 输出直方图长条
};
ptsin[0] = 0;
ptsin[1] = 0;

// 图形滞留屏幕,
// 等待按任一键退出
vrq_string(devhandle, 80, 0, ptsin, cstrng);
v_clswk(devhandle); // 关闭工作站
}

```

## 栏目编辑的话

汉字处理是我国计算机应用以及软件开发的重要课题,它包括许多具体的技术如直接写屏技术,汉字的立体显示技术及放大、旋转、平滑技术等等,本栏目将以汉字处理的实用技术为核心,全面介绍汉字处理的编程实用技术、技巧,它包括:汉字的直接写屏,各种字库的组织结构,以及汉字的放大平滑等多方面的内容。通过这一栏目读者可了解甚至掌握汉字处理的新型而实用的技术,栏目中提供的实用程序用户可以直接上机使用。

## 西文环境下图形与汉字同屏显示的方法

□陈德伍

**在**实际应用过程中,有时要求利用高级语言提供的丰富的作图语句或过程在屏幕上作图,同时在屏幕上任意位置显示汉字,如显示程序框图中的汉字,画面的文字说明,具有特殊要求的表格或表达式等。对中文操作系统支持的软件来说,这些都是不成问题的,如汉字系统 2.13H 就提供了不少这类特殊显示的功能调用,本文介绍的方法适用于西文操作系统环境下显示少量汉字的场合,由于既不需对西文操作系统支持的应用软件进行汉化处理,又可充分利用应用软件所提供的良好编程环境,因此编程简单,使用起来也比较方便。

**主要功能:**在图形方式下,在屏幕的任意位置上显示汉字串,(含英文、数字、制表符等)汉字与汉字之间的空隙可任意调节,每行汉字间的间距可任意调节,可从左向右显示,也可从上向下、从左上向右下、从左下向右上显示。

**程序设计主要思想:**根据汉字的区位码计算出汉字的机内码,用中断调用的方法得到汉字字模点阵在内存中的段地址,再通过调用 INT 10H 写象点的方法将汉字点阵写到屏幕上指定的位置。

**硬件环境:**286、386 及兼容机,VGA 彩色显示器,程序稍作改动也可用于 IBM PC/XT 兼容机及其它类型显示器。

**软件环境:**DOS 3.3,Turbo PASCAL 5.0,汉字 2.13H 中的 HZK16(16×16 点阵显示字库)、FILE1A.COM(一级字库驻留内存模块)。

**操作过程如下:**

**第一步:**开机进入西文操作系统环境。

**第二步:**将 HZK16 和 FILE1A.COM 两文件拷贝到 C 盘的根目录或任一子目录下。

**第三步:**在根目录或上述两文件所在的子目录下,键入 FILE1A 及回车,汉字库 HZK16 中的一级字库驻留内存,同时设置中断 INT 7EH,当 DX = 汉字内码时,通过调用 INT

7EH,DX = 该汉字字模在内存中的段地址。

**第四步:**转 Turbo PASCAL 所在子目录,输入如下所示的源程序后,若编译正确无误,执行该程序即可得到所期望的显示效果。

若想得到屏幕的硬拷贝,可在第三步中运行一次汉字 2.13H 中 SEGP.COM 或 SEG.COM。

本文所介绍的方法主要缺点是需要输入汉字的区位码,这对需要显示大量汉字的场合显得不太方便。因此,如需显示的汉字量较大,可利用中文操作系统提供的功能,将用其它方法输入的汉字串转变为相应的区位码存盘,再在源程序中加入通过读文件对 hz 过程中的值参,即表示汉字串的区位码字符串 s 赋值的语句的办法来解决。

为了简单起见,只给出了由左向右显示汉字的子程序,读者只需稍加改动就能完成其他显示功能。

```
program display;
uses crt,graph,dos;
var gd,gm:integer;
procedure hz(space,x,y:word;s:string);
var i,j,k,code:integer;
regs:registers;
offsetaddr,segaddr,w:word;
c:array[1..4] of char;
begin
  with regs do
    begin
      i:=1;
      repeat
        fillchar(regs,sizeof(regs),0);
        for j:=1 to 4 do
          begin c[j]:=s[i];i:=i+1;
            end;
          i:=i+1;
          val(c[1]+c[2],dh,code);
          val(c[3]+c[4],dh,code);
          dx:=(dx+$2020)or $8080;
          intr($7f,regs);
          segaddr:=dx;offsetaddr:=0;
          dx:=x;cx:=y;
```

(下转第17页)

## 栏目编辑的话

编程语言如 C、C++、汇编、BASIC、PASCAL、COBOL 等是用户开发应用系统的工具,灵活地应用计算机语言,充分理解编程语言所提供的编程特性是顺利、圆满地完成开发任务的基础。本栏目将广大计算机开发人员应用具体语言的心得体会集中起来,奉献给广大的读者,同时我们还将通过和编程语言开发厂商的联系,及时地向读者介绍新出现的编程语言的功能特点,并追踪编程语言的版本更新,向读者介绍具体编程语言的最新扩充的功能及其在程序开发中的应用,使广大的计算机开发人员能够通过本栏目,掌握或更深入地了解各种语言的编程应用技术与技巧,以期提高开发人员的工作效率,为开发人员制定自己的开发计划和在开发中充分、顺利地应用语言的功能奠定基础。编程语言栏目将重点介绍 C 语言、C++ 语言、汇编语言以及 BASIC 语言的编程技术、编程技巧以及它们所提供的函数的灵活应用技术,栏目中的文章将主要以实用程序说明应用的方法、技术与技巧,使读者能直观地掌握这些语言。

## 完善的监视器类型测试程序

□王 洋

**在**开发软件时,为了提高软件的适应性,使之在装有不同类型监视器的机器上均能运行,同时也为了能够充分发挥系统监视器硬件的性能,就需要对系统中所安装的监视器的类型进行测试。

关于测试监视器类型的方法,已有许多报纸、杂志介绍过这方面的文章,但这些文章有一个共同的缺点,就是考虑均不够全面,或多或少地遗漏了某些必要的测试过程或步骤。

在 Turbo C 中,有一个用于测试监视器类型的函数,这个函数就是 detectgraph。经过对 Turbo C 的 detectgraph 函数的跟踪分析,证实了该函数的确考虑十分全面,很有借鉴价值,因此笔者将其整理出来,供读者参考。

本文所附程序 DETECTGRAPH.ASM 是一个独立的模块,过程 DETECTGRAPH 为其主过程,应用程序只要调用该过程就可获得监视器的类型信息,该过程的用法为:

入口:无

出口:AX=-2 —— 系统中没有安装图形设备

AX=1	——	CGA
AX=2	——	MCGA
AX=3	——	EGA
AX=4	——	EGA64
AX=5	——	EGAMONO
AX=6	——	IBM8514
AX=7	——	HERCMONO
AX=8	——	ATT400
AX=9	——	VGA
AX=10	——	PC3270

将该过程的出口值与 Turbo C 的 detectgraph 函数的返回值进行比较,就会发现该出口值实际上就是 Turbo C 的 detectgraph 函数的返回值。

detectgraph 过程不但能够检测出常用的单色、CGA、EGA、VGA 监视器,而且能检测出较少使用(甚至没有用过)的 IBM8514、ATT400、PC3270。本文的另外一个程序 SAMPLE.ASM 演示了如何使用 DETECTGRAPH。

要生成运行文件,需要首先编译 DETECTGRAPH.ASM 和 DETECTG.ASM 程序,之后将 DETECTG 与 DETECTGRAPH 连接在一起。生成 DETECTG.EXE 文件,最后用 EXE2BIN 将 DETECTG.EXE 转换成为 DETECTG.COM 文件。

下面为生成运行文件 SAMPLE.COM 的过程:

- (1)MASM DETECTGRAPH;
  - (2)MASM SAMPLE;
  - (3)LINK SAMPLE+DETECTGRAPH;
  - (4)EXE2BIN SAMPLE SAMPLE.COM
- SAMPLE.ASM 的源程序如:

```
CR      =          13
LF      =          10
TAB     =           9
EXTRN   DETECTGRAPH: NEAR
CODE    SEGMENT    PUBLIC 'CODE'
        ASSUME     CS:CODE,DS:CODE
        ORG        100H

START:
        JMP        BEGIN
COPYRIGHT DB CR,LF,TAB,'Sample Program Using DETECTGRAPH Version 1.0'
        DB CR,LF,TAB,'Author:WZX',
        DB CR,LF,TAB,'BEIJING'
        DB CR,LF,TAB,'Tel:2017661',' '$'
CRLF    DB CR,LF,' '$',
NO-GRAPH-MON DB CR,LF,TAB,'No Graphics Monitor Found! ',CR,LF,' '$'
MON-TYPE-IS DB CR,LF,TAB,'Monitor Type Is $'
MON-TYPE-MESS DB 'CGA '
MON-TYPE-LEN EQU $-MON-TYPE-MESS
        DB 'MCGA '
        DB 'EGA '
        DB 'EGA64 '
        DB 'EGAMONO'
        DB 'IBM8514'
        DB 'HERCMONO'
        DB 'ATT400'
        DB 'VGA '
        DB 'PC3270'

BEGIN:
        LEA        DX,COPYRIGHT
        MOV        AH,9
        INT        21H
```



```

LEA    DX,MON-TYPE-IS
MOV     AH,9
INT     21H
CALL    DETECTGRAPH
CMP     AL,0
JA      GRAPH-MON-OK
1LEA    DX,NO-GRAPH-MON
JMP     DISP-EXIT
GRAPH-MON-OK:
DEC     AL
MOV     BL,MON-TYPE-LEN
MUL     BL
LEA     DX,MON-TYPE-MESS
ADD     DX,AX
MOV     AH,40H
MOV     BX,1
MOV     CX,MON-TYPE-LEN
INT     21H
LEA     DX,CRLF
DISP-EXIT:
MOV     AH,9
INT     21H
MOV     AH,4CH
INT     21H
ENDS
END     START

MOV     MON-TYPE,2
LBL-5:
POP     ES
RET
IS-HERC:
CALL    TEST-EGA
JB       LBL-3
CALL    TEST6
AND     AL,AL
JZ       LBL-6
MOV     MON-TYPE,7
POP     ES
RET
LBL-6:
MOV     SI,0B800H
MOV     ES,SI
XOR     SI,SI
MOV     AX,ES:[SI]
NOT     AX
NOT     Word Ptr ES:[SI]
NOP
NOP
CMP     AX,ES:[SI]
JNZ     LBL-7
MOV     MON-TYPE,1
LBL-7:
POP     ES
RET
TEST-EGA PROC
MOV     AX,1200H
MOV     BL,10H
MOV     BH,0FFH
MOV     CL,0FH
INT     10H
CMP     CL,0CH
JGE     LBL-8
CMP     BH,1
JG       LBL-8
CMP     BL,3
JG       LBL-8
STC
RET
LBL-8:
CLC
RET
TEST-EGA ENDP
TEST2 PROC
PUSH    ES
MOV     MON-TYPE,4
CMP     BH,1
JZ       LBL-9
CALL    TEST5
JB       LBL-10
AND     BL,BL
JZ       LBL-10
MOV     MON-TYPE,3
CALL    TEST4
JB       LBL-11
MOV     BX,0C000H
MOV     ES,BX
MOV     BX,0039H
CM      Word Ptr ES:[BX],345AH
JNZ     LBL-10
CMP     Word Ptr ES:[BX+02],3934H
JNZ     LBL-10
LBL-11:
MOV     MON-TYPE,9
LBL-10:
POP     ES
RET

```

因为 DETECTGRAPH. ASM 为一个独立的模块,因此在编写主程序(即调用 DETECTGRAPH 过程的程序,比如本文的 SAMPLE. ASM)时必须注意以下问题:

(1)代码段的名字必须为 CODE;

(2)在主程序中必须作如下声明:

```
EXTRN    DETECTGRAPH;NEAR
```

(3)代码段必须声明为 PUBLIC

DETECTGRAPH. ASM 的源程序如下:

```

CODE      SEGMENT    PUBLIC 'CODE'
ASSUME    CS:CODE,DS:CODE
PUBLIC    DETECTGRAPH
MON-TYPE  DB         ?
TEST-MON  PROC
PUSH      ES
MOV       AH,0FH
INT       10H
CMP       AL,7
JZ        IS-HERC
CALL      CLEAR-AL
JNB       LBL-1
MOV       MON-TYPE,6
POP       ES
RET
LBL-1:
CALL      TEST-EGA
JNB       LBL-2
LBL-3:
CALL      TEST2
POP       ES
RET
LBL-2:
CALL      TEST3
AND       AX,AX
JZ        LBL-4
MOV       MON-TYPE,10
POP       ES
RET
LBL-4:
MOV       MON-TYPE,1
CALL      TEST4
JNB       LBL-5

```

```

POP     ES
RET
TEST-EGA PROC
MOV     AX,1200H
MOV     BL,10H
MOV     BH,0FFH
MOV     CL,0FH
INT     10H
CMP     CL,0CH
JGE     LBL-8
CMP     BH,1
JG       LBL-8
CMP     BL,3
JG       LBL-8
STC
RET
LBL-8:
CLC
RET
TEST-EGA ENDP
TEST2 PROC
PUSH    ES
MOV     MON-TYPE,4
CMP     BH,1
JZ       LBL-9
CALL    TEST5
JB       LBL-10
AND     BL,BL
JZ       LBL-10
MOV     MON-TYPE,3
CALL    TEST4
JB       LBL-11
MOV     BX,0C000H
MOV     ES,BX
MOV     BX,0039H
CM      Word Ptr ES:[BX],345AH
JNZ     LBL-10
CMP     Word Ptr ES:[BX+02],3934H
JNZ     LBL-10
LBL-11:
MOV     MON-TYPE,9
LBL-10:
POP     ES
RET

```

```

LBL-9:
    POP     ES
    MOV     MON=TYPE,5
    RET

TEST2  ENDP
TEST5  PROC
    CMP     CL,2
    JB      LBL-12
    CMP     CL,6
    CMC
    JNB     LBL-12
    CMP     CL,8

LBL-12:
    RET

TEST5  ENDP

TEST4  PROC
    MOV     AX,1A00H
    INT     10H
    CMP     AL,1AH
    JNZ     LBL-13
    CMP     BL,7
    JZ      LBL-14
    CMP     BL,8
    JZ      LBL-14
    MP      BL,0BH
    JB      LBL-13
    CMP     BL,0CH
    JA      LBL-13

LBL-14:
    STC
    RET

LBL-13:
    CLC
    RET

TEST4  ENDP

CLEAR-AL PROC
    XOR     AL,AL
    RET

CLEAR-AL ENDP

TEST6  PROC
    MOV     DX,03BAH
    XOR     BL,BL
    IN      AL,DX
    AND     AL,80H
    MOV     AH,AL
    MOV     CX,8000H

LBL-16:
    IN      AL,DX
    AND     AL,80H
    CMP     AL,AH
    JZ      LBL-17
    INC     BL
    CMP     BL,0AH
    JNB     LBL-18

LBL-17:
    LOOP    LBL-16
    XOR     AL,AL
    RET

LBL-18:
    MOV     CX,8000H

LBL-19:
    IN      AL,DX
    AND     AL,30H
    CMP     AL,10H
    JNZ     LBL-20
    LOOP    LBL-19

MOV     AL,2
RET

LBL-20:
    MOV     AL,1
    RET

TEST6  ENDP
TEST3  PROC
    MOV     AL,6
    XOR     CX,CX
    XOR     DX,DX
    MOV     AH,30H
    INT     10H
    MOV     AX,CX
    OR      AX,DX
    JZ      LBL-22
    PUSH    DS
    MOV     DS,DX
    MOV     BX,DX
    MOV     AL,[BX+02]
    POP     DS
    OR      AL,AL
    JZ      LBL-21
    CMP     AL,2
    JNZ     LBL-22

LBL-21:
    MOV     DX,0188H
    IN      AL,DX
    TEST    AL,4
    JZ      LBL-22
    MOV     AX,1
    RET

LBL-22:
    XOR     AX,AX
    RET

TEST3  ENDP
TEST-MON ENDP
DETECTGRAPH PROC
    ;      INPUT:NO
    ;      OUTPUT:AL=GDRIVER
    ;      AH=GMODE
    PUSH    BX
    PUSH    CX
    PUSH    DX
    MOV     MON=TYPE,0FFH
    CALL    TEST-MON
    MOV     AL,MON=TYPE
    CMP     AL,0
    JG      VALID-G11
    MOV     AL,-2

VALID-G11:
    XOR     AH,AH
    POP     DX
    POP     CX
    POP     BX
    RET

DETECTGRAPH ENDP
CODE      ENDS
END TEST-MON

```

我们说 DETECTGRAPH 较为完善,是将其与其他的监视器测试程序相比较而言的,但从另外一角度来讲,DETECTGRAPH 仍有其不完善之处,这就是它只能测试单色、CGA、EGA 和 VGA 监视器,而对目前使用日益广泛的各种 Super VGA(包括 TVGA)却不能进行测试,这不能不说是一种遗憾,但是可以相信,Turbo C 以后一定会弥补这种遗憾。

# 用 Turbo C 实现 SHELL 功能

□徐建钢

**S**HELL 功能的作用就是在一个应用软件运行环境中,通过输入 SHELL 命令或选择相应菜单,可以退到 DOS 环境,运行 DOS 命令或其它程序和软件,然后在 DOS 提示符下键入 EXIT 命令返回原应用软件的运行环境,继续原运行软件。

这种 SHELL 功能十分有用,它可以让用户在原应用软件的运行过程中,查看暂时遗忘的文件名,运行某些特定的配置程序,多屏幕编辑不同的 ASCII 码文件,中途生成原软件运行所必需的数据文件等等。如果用户在自己编制的程序中动态地运用这种 SHELL 技术,则可实现某些原本难以实现的功能,或收到意外的惊奇

效果。

下面介绍用 TURBO C 语言实现这种 SHELL 技术的方法。

TURBO C 语言的函数库中,有一个 SYSTEM 函数,其中

调用方式: `int system(char * string)`

原型文件: `stdlib.h`

函数功能:把由 `string` 所指向的字符串作为命令传递给 DOS,并且返回命令的退出状态。例如 `system("dir")` 语句用于在程序运行过程中在屏幕上列出当前目录的文件; `system("cls")` 语句用于清屏,在程序中相当于 `clrscr()` 函数。

上述调用方法只能执行一条 DOS 命令,且并不退到 DOS 环境。

若以一个空字符串为参数调用 `system` 函数,也即使用 `system("")`; 语句,就可以实现 SHELL 功能。

下面是实现 SHELL 的一个例程:

```
#include "stdio.h"
#include "stdlib.h"
main()
{
    puts("Now you are in your program.");
    puts("Press a key to DOS enviroment.");
    puts("use EXIT command to return.");
    getch();
    system("");
    puts("Now in your program again.");
    getch();
    exit(0);
}
```

## 第一代打印机共享器的换代产品

### — 挂接式全自动打印机共享器

### 你听说过吗?

98 元

省一台打印机  
—— 绝对不是神话!

适用于各类打印机,激光印字机,绘图仪

**北京向宇计算机公司**

地址:北京复兴路甲 20 号 27 分号 312,314 邮编:100036

电话:8263441 2063366 呼 1511 开户银行:工商行北京翠微路城市信用社 帐号:07900-36

诚征全国代理

# 混合语言编程时慎用 OFFSET

□ 王 祥

**现**在使用高级语言与汇编语言混合编程的越来越多,在编写汇编语言程序时有一个需要注意的问题,就是要注意 OFFSET 的使用。这个问题在单独使用汇编语言编程时也存在,但混合语言编程使得该问题更加突出,读完本文后就会明白的这一点。先看下面的例子:

disp. c 调用汇编语言模块 display. asm, 而 display. asm 显示字符串“Demo of Usage of OFFSET!”。

disp. c 源程序如下:

```
#include <io. h>
#include <stdio. h>
extern display();
main()
{
display();
}
```

DISPLAY. ASM 源程序如下:

```
STACK SEGMENT STACK PAPA'STACK'
    DW 100 DUP(0)
STACK ENDS
__DATA SEGMENT PUBLIC WORD'DATA'
SAMPLE DB 'Demo of Usage of OFFSET!',13,10,'$'
__DATA ENDS
DGROUP GROUP STACK, __DATA
    PUBLIC __display
__TEXT SEGMENT PUBLIC WORD'CODE'
    ASSUME CS: __TEXT, DS: DGROUP, ES: DGROUP, SS: DGROUP

__display PROC
    PUSH BP
    MOV BP, SP
    PUSH DS
    PUSH ES
    PUSH DX
    PUSH AX
    MOV AX, DGROUP
    MOV DS, AX
    MOV ES, AX
    MOV DX, OFFSET SAMPLE
    MOV AH, 9
    INT 21H
    POP AX
    POP DX
    POP ES
    POP DS
    POP BP
    RET
__display ENDP
__TEST ENDS
END
```

执行如下操作: (1)cl/c disp. c; (2)masm display; (3)link disp+display; 形成运行文件 disp. exe, 执行 disp, 得到如下显示结果:

MS Run - Time Library-Copyright (c) 1987, Microsoft  
Crop Demo of Usage of OFFSET!

可见执行结果与预期不同,其原因就在于 display. asm 中

的语句 mov dx, offset sample。

OFFSET 指令用于返回变量或标号的偏移量,这在不使用 GROUP 伪指令的情况下,可以得到正确结果,但是如果程序中使用了 GROUP 伪指令,OFFSET 返回的仅仅是变量或标号所在段的偏移量,而不是其所在的组(GROUP)的偏移量,这样,OFFSET 返回的并非变量或标号在组中的偏移量,如此则得不到正确的结果。

在使用高级语言与汇编语言混合编程时,均使用了组(比如:在 C 语言中将 -DATA, BSS, CONST 和 STACK 均放在组 DGROUP 中),这样在混合语言编程中使用 OFFSET 将不能得到正确的结果。在前面的例子中,送给 DX 的并非 SAMPLE 在组 DGROUP 中的真正的位置,因此显示结果不正确。

要得到变量或标号的真正偏移量,可以在变量或标号之前加上组名来实现,如下面的例子:

```
DGROUP GROUP DATA CODE
DATA SEGMENT
F00 DB 0
OFFINDATA DW F00
OFFINDGROUP DW DGROUP: F00
DATA ENDS

CODE SEGMENT
    ASSUME CS: CODE, DS: DGROUP
    ...
    MOV DX, OFFSET F00
    MOV BX, OFFSET DGROUP: F00
    ...
CODE ENDS
```

因此,在前面所举的例子中,将语句: MOV DX, OFFSET SAMPLE 改为 MOV DX, OFFSET DGROUP: SAMPLE 即可得到正确结果。

其实还有一种获取变量或标号的方法,就是使用 LEA 指令,这种指令的用法为:

LEA REGISTER, 变量或标号, 该指令将变量或标号的偏移量送入寄存器 REGISTER, 该方法比 OFFSET 方法有许多优点:

1. LEA 指令为 3 字节指令, 而 OFFSET 方法为 4 字节指令, 因此该方法节省空间。

2. LEA 指令周期数比 OFFSET 指令周期数少, 因此 LEA 方法速度快。

3. 使用 LEA 方法, 在编程时书写简单。比较: LEA DX, SAMPLE 和 MOV DX, OFFSET SAMPLE。

另外 LEA 方法还有一个优点就是 LEA 所取得的偏移量为变量或标号在组中的偏移量, 而不是它在自己段中的偏移量。在所举的例子中, 将语句: MOV DX, OFFSET SAMPLE 改作 LEA DX, SAMPLE 即可得到正确结果。

综上所述, 可以看出: LEA 方法与 OFFSET 方法相比, 可以说只有优点而无缺点, 因此建议在获取变量或标号的偏移量时不要使用 OFFSET 方法而改用 LEA 方法。

# Borland C++与汇编语言的接口技术

□张兴滔

**B**orland C++(以下简称C++)是世界著名的软件公司 Borland international 推出的最新面向对象程序设计 OOP(Object-Oriented Programming) 软件开发系统,它与目前的 Turbo C 高度兼容,并且是一种比 C 语言更进一步发展的面向对象的高级编程工具。但作为一种高级语言,在需要调用低级功能时需要较高的运行速度时,就需要用到与汇编语言进行交互式调用或者是进行各种数据的相互传递。本文就谈谈这两种语言间的接口技术和实现方法。

## 一、C++调用汇编子过程的技术

### 1. C++源程序的模式

在调用汇编子过程的 C++源程序的文件开始处,应该用 extern 标志符加以说明,例如:

```
extern int min(int v1,int v2).
```

这是由于编译器不能自动识别 C++源程序以外的其它子过程,否则将被看作程序内的变量而进行处理;如果有返回值,必须与 C++源程序中定义的不同,1个字节的返回值返回到 AL 寄存器中,2个字节值返回到 AX 寄存器中,4个字节的返回值返回到 DX:AX 寄存器中,而3个或大于5个字节的返回值放入一个静态数据区,然后返回指向那个地址的指针(小数据模式时在 AX 寄存器中,大数据模式的在 DX:AX 寄存器中),因此被调用的子过程必须将返回的值(或指针)拷贝到指针所指向的地方。

### 2. 汇编语言源程序的模式

在编写汇编语言源程序时,应该遵守几个约定:确保连接器可以得到必要的信息;确保文件格式和用户的 C++程序使用的存储模式一致。

一般说来,汇编语言模块由三部分组成:代码、初始化数据和非初始化数据,每组信息被组织在自己的段中,段名取决于

用户的 C++程序使用的存储模式,而 Turbo 汇编(TASM)提供了三种简化这几个段的指令(.CODE、.DATA 和 .DATA?),这样就可以用来定义这些段;另外还可以使用伪指令.MODEL 来定义存储模式,这也决定了对汇编子过程的调用属性是远程调用还是近程调用,例如使用小存储模式:

```
.MODEL small
.CODE
...代码段...
.DATA
...初始化数据段...
.DATA?
...非初始化数据段...
```

对汇编子过程来说,这种调用属于外部调用,因此要对过程名赋以公有属性(public)的定义;还有由于汇编子过程不是主过程,所以程序结束处的 END 伪指令后面不能加起始标号,以免连接器连接时产生错误,另外编译器在目标模块中保存外部标识符时,会在该标识符前面自动地添加一个下划线,这就说明应该在汇编语言模块中对想从 C++程序中引用的任何标识符的前面添加一个下划线。汇编语言子过程被调用时,如果带有参数,那么程序开始处的编写格式一般如下:首先应该保存 BP 寄存器值,赋堆栈指针 SP 给基址指针 BP,然后保存所使用的寄存器,再通过基址指针 BP 从栈中获取由 C++调用时传递来的实参。在结束时调整堆栈指针 SP,使之越过 C++此次调用的实参,然后恢复 SP 寄存器,最后恢复 BP 寄存器并返回。

## 二、从汇编语言子过程中对 C++程序的调用

### 1. 从汇编程序中对 C++函数的调用

从汇编语言模块中调用 C++函数,首先必须使 C++函数在汇编语言模块

中可见,可以通过使用 near 或 far 修饰符来说明为 EXTRN,对于在汇编语言程序中用.MODEL 命令定义存储模式的汇编模块来说,也可以只用关键字 PROC 来说明它为 EXTRN,使要说明的 C++函数对汇编语言模块是可见的。例如用户已经编写了下面这个 C++函数:

```
long docalc(int * para1,int para2,
int para3);
```

为了简便起见,假设正在使用 tiny、small 或 compact 存储模式,这样在汇编语言模块中就可以这样说明它:

```
EXTRN _docalc:near
或者是
```

```
.MODEL small
EXTRN _docalc:PROC
```

对于使用 medium、large 或 huge 存储模式,可如发炮制地将它说明为:

```
EXTRN _docalc:far
或者是
```

```
.MODEL large
EXTRN _docalc:PROC
```

假设你想将结果存在一个名字为 ans 的 32 位长的地方,C++中等价的调用是 ans=docalc(&xval,imax,421),其中 docalc 被调用时带有三个参数:

名字为 xval 的位置的地址;  
存于名字为 imax 位置的值;  
一个常数值 421(+进制)。

要使汇编程序能够按照 C++中定义的形参形式进行正确地调用,那么实参的传递方式必须严格规定,首先在栈中压入 421,接着是 imax,然后是 xval 的地址,最后调用 docalc,当它返回时,必须清除栈空间中额外的 6 个或 8 个(对远程调用而言)字符,然后将结果送到 ans 和 ans+2 中。下面为相应的源代码:

```
MOV AX,421;AX 被赋值,并压入栈中
PUSH AX
PUSH IMAX ;imax 压入栈中
```



```
LEA    AX,XVAL
PUSH   AX          ;xval的地址值压
                    ;入栈中
CALL   _DOCALC     ;调用C++函数
ADD    SP,6        ;清除堆栈
MOV    ANS,AX      ;对ANS进行赋值
MOV    ANX+2,DX
```

## 2. 从汇编模块对C++中变量的调用

同样使用 EXTRN 语句可以使用户的汇编语言模块引用为C++程序中定义的变量,为了引用数据,应该在用户的数据段中适当地放置 EXTRN 语句,定义格式如下:

```
EXTRN  vname;size
```

其中 vname 为变量名,size 表示变量的大小,其可能取值是:

```
BYTE—1个字节;
WORD—2个字节;
DWORD—4个字节;
QWORD—8个字节;
TWORD—10个字节。
```

所以如果C++程序中有下面的全局变量:

```
int i,array[10];
char ch;
long result;
```

用户可以使用下面的语句使它们在你的汇编模块中成为可引用的变量:

```
EXTRN  -i; WORD,-array;
WORD,-ch;BYTE,-result;DWORD
```

其中,如果使用 huge 存储模式,EXTRN 语句必须在任何段以外,这同样适用于上面所讲的C++函数。

## 三、入口参数的传递方式和顺序

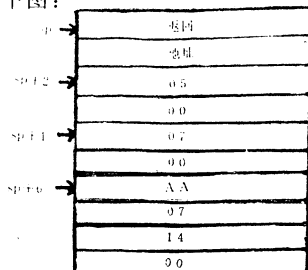
C++程序是通过堆栈向汇编语言子过程传递入口参数的。在调用子过程时,先按顺序将各个参数压入栈中,然后再压入返回时的地址。例如已经说明了下面的函数原型:

```
void func(int p1,int p2,int p3);
```

当此函数被调用时,参数要按照从右向左的顺序(p1,p2,p3)压入栈中,然后在栈中压入返回地址,所以如果进行了下面这个调用:

```
main()
{
int i,j;
long k;
...
i=5;
j=7;
k=0x1407AA;
func(i,j,k);
```

...  
}
当刚刚调用 func 函数时,栈中的参数顺序如下图:



堆栈图

由于栈是从高地址向低地址生成的,所以此时在返回地址下面的参数首先为 i,然后是 j,k,即 i,j,k 三个参数中,参数 i 应该在栈顶。另外被调用的过程无需要知道有多少参数被压到了栈中,更重要的是被调用过程不需要将参数弹出栈,因为调用过程将完成这个工作。例如:编译程序从这个 main 函数的源代码中产生的汇编语言是这样的:

```
mov WORD PTR [bp-8],5      ;置 j=5
mov WORD PTR [bp-6],7;置 j=7
mov WORD PTR [bp-2],0014H
                        ;置 k=0x1407AA
mov WORD PTR [bp-4],07AAH
push WORD PTR [BP-2]
                        ;把 k 的高字压入栈中
push WORD PTR [BP-4]
                        ;把 k 的低字压入栈中
push WORD PTR [BP-6]
                        ;把 j 压入栈中
push WORD PTR [BP-8]
                        ;把 i 压入栈中
call NEAR PTR func
                        ;调用函数,同时压入返回地址
add sp,8              ;调整堆栈指针
```

由上面最后一条指令: add sp,8,可知编译器在这点是知道有多少参数被压入栈中的,它也知道由调用 func 函数而压入的返回地址已经被 func 结束处的 ret 指令弹出栈。在汇编子过程中获得参数而无需从栈中弹出东西,只需保存基址指针 BP,将堆栈指针 SP 送给基址指针,然后通过它直接检索栈中的值,以获取各种参数值。注意:当用户将 BP 寄存器压入栈中时,参数的相对偏移量增加了2个字节,因为栈中多了 BP 值的两个字节。

在被调用函数中,函数的开头和结尾可以采用如下所示的格式:

```
MOV    BP,SP
SUB    SP,N
PUSH   DI
PUSH   SI
.....
POP    SI
POP    DI
MOV    SP,BP
POP    BP
```

采用这种格式以后,根据后面将要讲的寄存器使用原则,在函数体内就可以对所有的寄存器作任意的修改和使用。这里借助 BP 作地址寄存器到堆栈中寻找操作数,[BP+偏移量]用来取出调用者传来的参数,[BP+偏移量]用来存取函数体内开辟的局部变量区,也就是相当于C++语言中的自动变量区。当控制权由被调用函数返回到调用函数以后,由发出调用的函数从堆栈中把参数清除掉,这一点无论是C++语言调用汇编语言,还是汇编语言调用C++语言,都应该遵守这个参数传递原则。

## 四、返回结果值的处理

一般为了接收函数的返回结果,C++程序要定义与返回结果同类型的变量,将函数值赋予该变量。C++对返回值的处理有一定的规定:对16位(2个字节)值(char,short int,enum 和 near 指针),使用 AX 寄存器;对于32位(4个字节)值(包括 far、huge 指针等)还要使用 DX 寄存器,其中高字(对指针而言是段地址)在 DX 中,低字在 AX 中,float double 和 long double 值返回在80×87栈顶 TOS (Top Of Stack)寄存器 ST(0)中,如果使用80×87仿真库,值被返回到仿真库的 TOS 寄存器中。调用函数必须将这个值拷贝到需要它的地方。由上可见,1个字节的结构返回到 AL 寄存器中,2个字节值返回到 AX 寄存器中,4个字节值返回到 DX:AX 寄存器中,而3个或者是大于5个字节的值放入一个静态数据区,然后返回指向那个地址的指针(小数据模型时在 AX 寄存器中,大数据模型在 DX:AX 寄存器中),因此被调用的子过程必须将返回的值(或指针)拷贝到指针所指向的地方。例如:C++中有下面的函数原型:

```
extern int min(int v1,int v2);
它返回传递给它的两个值中的最小值,
min 的格式是:
```

```
PUBLIC  -min
-min PROC NEAR
PUSH   BP,SP      ;保存 BP 压入栈中
MOV    BP,SP      ;把 SP 赋给 BP
MOV    AX,[BP+4];参数 V1给 AX
CMP    AX,[BP+6];参数 V1与 V2比较
JLE    EXIT       ;如果 V1>V2
MOV    AX,[BP+6]   ;V2值给 AX
EXIT:
POP    BP         ;恢复 BP
RET     ;返回结果,退出 min
-min ENDP
```

上面 min 一例中,所处理的值全部为16位长,所以只需将返回结果放在 AX 寄存器中。下表列出了各类返回值的存放原则。

# 用“宝合”UPS 电源, 再不怕电网突然掉电



PH-500S



PH-1000

当外电网突然掉电时, 微电脑靠其内部的储能电容通常能维持10MS左右的工作时间, 为了避免数据丢失, 磁盘和磁头遭受损失, 造成严重后果, 这就需要有一种电源系统, 不仅要有不间断供电性能, 而且还要有抑制电网噪音, 抗电网干扰, 稳压等性能, 在外电网突然掉电时, 以保证微电脑系统的正常运行。



PH-1KL



PH-3KL



PH-600

## “宝合”UPS 电源获奖一览表

年 度	获 奖 名 称
1989 1991, 1992	北京市新技术产业开发试验区优秀拳头产品
1991	91 年度国家级新产品
1992	被国家科委成果管理办公室推荐为国内优秀产品 北京市科学技术进步三等奖
1993	被北京市技术监督局评为北京市企业标准成果奖 全国第七届运动会指定产品



北京希望电脑公司电源部

地址: 北京海淀路 82 号  
邮编: 100080  
信箱: 北京 8721 信箱  
电话: 2567819, 2561058  
传真: 2561057

上海希望电脑公司

地址: (200052) 上海新华路 416 号  
电话(传真): (021) 2521656, 2569929  
南京希望电脑技术公司  
地址: (210005) 南京市中山南路 105 号  
电话: (025) 4410794, 4410549  
传真: (025) 4410548

成都希望电脑公司

地址: (610015) 成都市新南路四维村街 6 号  
电话(传真): (028) 5589787, 5556333  
广州希望电脑技术公司  
地址: (510620) 广州天河体育西路育菁三街二  
电话: (020) 7505151, 7505152, 7505153  
传真: (020) 7500275





北京希望电脑公司出版、发行的技术资料，具有内容新、种类多、出版快的特点，并以高效率、高质量、高信誉的服务赢得广大用户的好评。

**北京希望电脑公司资料事业部**

地址：北京海淀路 82 号

信箱：(100080) 北京 8721 信箱

电话：2562329、2541992

传真：2561057



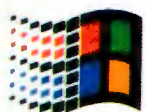




# Microsoft WIN32

SOFTWARE DEVELOPMENT KIT

*Tools for Writing 32-Bit Applications for Windows.*



MICROSOFT  
WINDOWS NT<sup>™</sup>  
COMPATIBLE  
32-Bit Application

微软公司北京代表处

北京新世纪饭店写字楼五层551室

中国北京首都体育馆南路六号

邮政编码: 100046

电话: (86-1)849-2148~50

传真: (86-1)849-2151



# Microsoft VISUAL C++

*Development System for Windows<sup>™</sup> and Windows NT<sup>™</sup>*



类型	存放返回值的地方
char	AX
unsigned char	AX
short	AX
unsigned short	AX
int	AX
unsigned int	AX
long	AX/DX
unsigned long	AX/DX
struct(<4Byte)	AX/DX
struct	静态存储区, 指针在 AX 或 AX/DX 中
float	静态存储区, 指针在 AX 或 AX/DX 中
double	静态存储区, 指针在 AX 或 AX/DX 中
near point	AX
far point	AX/DX

## 五、寄存器正确使用的规则

在汇编语言与 C++ 语言的混合编程中, 还应该注意约定寄存器的正确使用, 只有弄清这些以后, 才能写出正确的函数。8086/8088/80286 共有 14 个寄存器, 从调用函数进入被调用函数时, 以及从被调用函数返回到调用函数时, 这些寄存器可能分别包含有什么值, 哪些寄存器的值应该保持不变, 哪些寄存器的值可以改变, 这也是混合编程时必须弄清楚的问题。

在 C++ 编译器中, 大体上可以把这些寄存器分为三类:

第一类是: CS、IP、SS、SP、DS、ES。只要按照上面已讲述的各项原则正确地把两种语言模块连接起来了, 能够正确地进入被调用函数并正确地返回到调用函数, 就说明对 CS、IP、SS、SP 四个寄存器的处理是正确的。在被调用函数内, 当然会改变 IP 和 SP 的值(隐式地), 如果有必要, 也可以改变 CS 和 SS 的值, 只要保证能正确地返回就可以了。在某些 C++ 编译存储模式下(只有一个数据段), 如果 C++ 语言模块和汇编语言模块连接正确, 那么在从 C++ 语言进入汇编语言时, DS 寄存器就已经指向所要求的数据段, 用不着在汇编语言程序中再去初始化 DS。如果所使用的 C++ 编译存储模式会产生多个数据段, 并且相互连接的 C++ 模块和汇编模块的数据不在一个数据段, 则在汇编语言程序中需要修改 DS, 使之指向自己的数据段。但是, 如果修改了 DS, 就

应该在返回到 C++ 语言以前, 恢复入口时的 DS 寄存器的值。至于 ES 寄存器, 则没有任何约定, 既不寄希望于入口时 ES 会指向什么固定的位置, 也不要求在返回时保持不变。总之, 寄存器变量 CS、DS、SS 和 ES, 根据所使用的存储模式有特定的值, 它们的关系是:

Tiny 模式时, CS=DS=SS, ES=任意值;  
Small, Medium 模式时, CS!=DS, DS=SS, ES=任意值;

Compact, Large 模式时, CS!=DS!=SS, ES=任意值(每个模块有一个 CS);

Huge CS!=DS!=SS ES=任意值(每个模块有一个 CS 和 DS)。

第二类是 BP、SI、DI 寄存器。在汇编语言子程序中, 必须在一开始就把 BP 寄存器的值压入堆栈, 在返回到 C++ 语言前再把这个值弹回给 BP, 保证 BP 的值不发生变化; 还有 SI 和 DI 两个寄存器, 这两个寄存器一般被 C++ 用作寄存器变量, 在汇编语言子程序内也应该保持它们不被改变。如果在一个汇编语言过程中使用它们, 应该在进入时保存它们, 退出时再加以恢复, 但是如果在行编译时使用 -r- 选项(或者是在集成环境下 Code Generation 对话框中 Register Variable 为非检查状态)来编译 C++ 程序, 就不必保存 SI 和 DI 寄存器, 这时在汇编语言子程序内就可以任意使用 SI 和 DI, 而不用担心对 C++ 程序会产生什么影响。

第三类是 AX、BX、CX、DX 寄存器。这四个寄存器在汇编语言子程序内可以任意使用, 尤其是 BX 和 CX 寄存器。AX 和 DX 寄存器还担负着传递返回值的作用, 因此如果汇编语言子程序有返回值, 则应该保证 AX 和 DX 中存有正确的返回值。

虽然不同语言间编程的接口技术是一个比较复杂的问题, 但是只要掌握原则, 潜心练习, 必然能编制出具有强大功能、更高效的优秀软件。下面举一个简单的两种语言间交互调用的过程(或函数)的例子本例在 AST386 机上运行通过。

下面分别为汇编源程序和 C++ 源程序:

```
; 汇编程序名 AVERAGE. ASM, 其功能如下:
; C++ 可调用的子模式函数, 它返回一组
; 整数的平均数;
; 调用 C++ 函数 Int Divide() 来完成最后的除
; 法运算。函数原型如下:
extern float Average(int far * ValuePtr,
int NumberOfValues);
```

```
; 其中输入参数:
; int far * ValuePtr; ; 指向一组求平
; 均数的指针
; int NumberOfValues; ; 求平均数的个数
DOSSEG
.MODEL SMALL
; 定义存储模式为小模式
EXTRN -IntDivide; PROC
; 定义外部函数
.CODE
PUBLIC -Average
; 定义公有属性
-Average PROC
push bp
mov bp, sp
les
bx, [bp+4]
; 使 ES, BX 指向求平均数的数组
mov cx, [(bp+8)] ; 求平均数的数目
mov ax, 0 ; 总和先清零
```

```
AverageLoop:
add ax, es:[bx] ; 加入当前值
add bx, 2 ; 指向下一个数
loop AverageLoop
push WORD PTR [BP+8]
; 获得求平均数的数目并且传给函数
; IntDivide 作为最右边参数
push ax
; 传递总和作为最左边参数
call -IntDivide
; 调用计算浮点除法函数
add sp, 4
; 调整指针 SP 使之越过参数
pop bp
ret
; 平均数放在 8087's 的 TOS 寄存器并返回
```

```
-Average ENDP
END
/* C++ 程序名 CALCAVG. CPP, 其功能如下:
C++ 程序调用汇编模块, 而且定义一个
被汇编代码调用的、完成浮点计算的 C++ 程
序 */
extern float Average(int far * ValuePtr,
int NumberOfValues);
#define NUMBER_OF_TEST_VALUES 10
int TestValues[NUMBER_OF_TEST_VALUES] = {
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
};
main() /* 主程序开始 */
{
printf("The average value is: %f\n",
Average(TestValues, NUMBER_OF_TEST_VALUES));
/* 调用汇编子程序 */
}
float IntDivide(int Dividend, int Divisor) /*
* 定义被汇编程序调用的 C++ 函数 */
{ return ((float) Dividend / (float) Divisor);
/* 返回结果 */
}
```

# 汇编语言的递归调用 和过程自调用

□ 郑 祥

## 一、递归调用

程序员都知道绝大多数的高级语言都提供了递归调用的功能,其实汇编语言中也能实现递归调用,本文以实例方式介绍递归调用的方法:

实现递归调用的关键是要保证两点:

(1)递归过程有出口,即当满足一定条件时,过程返回,并且这个条件是可以满足的。

(2)保证递归过程的每一次返回能遵守相同的返回约定。在汇编语言中对返回值可作如下约定:

- ①如果返回值为单字节,则返回在 AL 中。
- ②如果返回值为字,在返回在 AX 中。
- ③如果返回双字,则在 DX:AX 中。

实际使用时,可以根据需要灵活选择,但必须保证每一次返回于相同的寄存器或单元中。

(3)必须保证堆栈有足够的深度。

下面我们以两个实例,看看它们如何满足条件。

### 求整数 N 的阶乘 N!

求整数的阶乘是一个典型的递归调用,这在 C 语言中可用如下的函数实现:

```
nmulti(n)
unsigned n;
{ if(n==1)return(1)
  else return(N * multi(n-1));
}
```

在汇编语言中可用类似的方法实现,下面为求整数阶乘的汇编语言程序,递归过程写成了供 C 语言函数调用的形式:

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
ORG 100H

BEGIN:
MOV AX,6
PUSH AX
CALL NMULTI
ADD SP,2

EXIT:
MOV AH,4CH
INT 21H
```

```
;
NMULTI PROC ;递归过程
PUSH BP
MOV BP,SP
PUSH BX
PUSH CX
MOV BX,[BP+4] ;取本次整数输入
BX
CMP BX,1 ;if(n==1)return(1)
JZ RETURN
DEC BX
PUSH BX
CALL NMULTI ;求 mutli(n-1)
ADD SP,2
MOV BX,[BP+4]
PUSH AX
MOV AX,DX
MUL BX
MOV CX,AX
POP AX
MUL BX
ADD DX,CX ;n * mutli(n-1)
POP CX ;返回在 DX:AX 中
POP BX
POP BP
RET
```

```
RETURN:
XOR DX,DX ;return(1)
MOV AX,1 ;DX=0,AX=1
POP CX
POP BX
POP BP
RET
NMULTI ENDP
```

```
CODE ENDS
END BEGIN
```

本例中,出口条件是  $N=1$ ,这个条件显而易见肯定能得到满足,过程返回值在 DX:AX 中,分析过程会发现,mutli(n-1)返回值在 DX:AX 中,而  $n * mutli(n-1)$  的返回值同样在 DX:AX 中。本例中堆栈有足够的深度,因此未作处理。

本例所举例与 C 语言相仿,因此较容易理解。下面看另外一个例子,该例子的递归过程不象这个例子那么明显,但它满足了前面所谈的三个条件,因此能正确执行。该例子将 AX 中的数值变成十进制 ASCII 码输出,从本例中可以看出使用递归的一个重大优点:程序简洁有效,短小精悍。

```
CODE SEGMENT
ASSUME DS:CODE,DS:CODE
ORG 100H
BEGIN: MOV AX,200H
CALL HEX2ASC
```

```
MOV AH,4CH
INT 21H

HEX2ASC PROC ;NOTICE
MOV CX,10 ;**
XOR DX,DX
DIV CX
OR AX,AX
JZ H2A__END
PUSH DX
CALL HEX2ASC
POP DX

H2A__END:
ADD DL,'0'
MOV AH,2
INT 21H
RETN
HEX2ASC ENDP

CODE ENDS
END BEGIN
```

本例子的出口为  $AX=0$ ,即商为 0,这个条件总能满足,该递归过程每次返回一位位于 DL 中,并输出之。

本过程是一个很有价值的过程,它可以将 AX 的值变成任何进制的 ASCII,要变 XX 进制的 ASCII,只要将带 \* 号的语句改作 MOV CX,XX 即可。当然输出时需要根据实际情况选择合适的 ASCII 码(比如:对十六进制,超过 10 时使用 A-F 表示)。

## 二、过程的自调用

高级语言函数不能有多多个入口,而汇编语言则不然。汇编语言的过程不但能够有多多个入口,而且还能够调用自己的一部分。下面举例说明这一点,该例子用于将 AX 的数值以 16 进制 ASCII 码输出。

```
CODE SEGMENT
ASSUME CS:CODE,DS:CODE
ORG 100H

BEGIN:
MOV AX,200H
CALL HEX2ASC1
MOV AH,4CH
INT 21H

HEX2ASC1 PROC
PUSH AX
MOV AL,AH
CALL BYTE2ASC
POP AX

BYTE2ASC:
PUSH AX
MOV CL,4
```

(下转第 66 页)

# 栏目编辑的话

数据库管理系统是开发人员进行 MIS 开发工作的重要工具,也是目前应用最为广泛的开发环境之一,dBASE、FoxBASE 已为广大计算机应用开发人员所熟知。因此,我们将数据库技术作为一个单独的栏目独立出来,以期能为广大的数据库用户及其开发人员提供一个交流的天地。本栏目将介绍 dBASE、FoxBASE、FoxPro 以及 ORACLE 等的新功能、新用法,以及利用这些数据库系统开发环境进行开发工作的经验、编程技术、技巧和编程方法。

## 巧用 FoxPro2.5 屏幕生成器

□ 仁 宇

**F**oxPro2.5 的屏幕生成器功能十分强大,它能交互式地定义数据录入、显示用户界面和内部操作,十分简单地生成大量 FoxPro2.5 的源程序。更重要的是 Screen Builder 使编程规范化、使用户的程序编制标准不出漏洞。另一点是所有 Screen Builder 的定义可以跨平台移植,这使得 FoxPro 应用系统在 DOS、Windows、Macintosh 几个平台上不再为移植而大伤脑筋。

若读者从未使用过 Screen Builder 可能需要一个简单的适应过程,让自己熟悉设计用户界面的方法:以交互手段把一定形状的文本、数据放置在屏幕的合适位置上。其基本操作与 FoxPro2.0、FoxPro2.5 For DOS、FoxPro2.5 For Windows 大体一致。FoxPro Windows 版更复杂些。

定义好用户界面后,用户靠系统菜单中的 Program 中的 Generate 产生源程序。Generate 会弹出一个很复杂的窗口,一般说来,你不必关心其中的每个选项,都取默认值便可生成源程序。生成的源程序后缀是 .SPR 而不是 .PRG。运行生成的程序可以在命令窗口中运行

DO \* \* \*.SPR

即可看到界面设计的最后效果。SPR 文件实际上就是 FoxPro 的源程序跟 .PRG 文件一样。如果您发现 FoxPro 生成的程序有些地方不符合自己的要求,你可以自己去修改。不过最后的 .SPR 文件最好改成 .PRG 文件,否则重新生成后的 .SPR 会把你自己的程序冲掉。

启动屏幕生成器也可以在命令窗口中以命令:

create screen \* \* \*

或 modi Screen \* \* \*

进行。

### 1. 基本概念

FoxPro2.5 的用户界面设计遵从于国际标准化的用户界面。基本上有下面几种组件:

Windows, Check Boxes (按钮) Invisible Buttons (不可见按钮)

Lists (菜单带滚动条)、Popups (菜单条不带滚动条)

Push Buttons (功能按钮) Radio Buttons

(几种选择按钮,用户从中选一)

Spinners (计数器), Text Editing Region (文本编辑区)

所有这些构件都可以在 Screen Builder 中定义,每一个构件都有许多属性设置,熟悉了这些设置您的程序设计便会上一个台阶。

所有这些构件在命令方式上都可以通过 @.....GET 命令创建(包括 @.....EDIT)

@.....get 命令中的 FUNCTION、PICTURE 子句决定了构件的种类和外观,许多组件选中时要启动一定的动作,这些动作便可以用 VALID 子句来实现、@.....get 和 @.....edit 以 READ cycle 和 READ 来启动。READ 和 READ cycle 当一个构件被选中时既可以被中止也可继续。所有构件的 Function picture 子句形式如下:

checkboxes

@.....get mchoices FUNCTION ' \* C

按钮名'

Lists

@.....get FUNCTION '&N' 选中不结

束 READ

popups

@.....get FUNCTION '选择1;选择2;选择3'.....

push buttons

@.....get FUNCTION ' \* OK;Cancel'

OK,Cancel 为动作按钮的文字。

Radio buttons

@.....get FUNCTION ' \* R

男;女;性别不定'

男、女性别不定为选项的名字

编辑不定长文本

@..... get EDIT 文本变量名

### 2. 窗口定义

限于篇幅,只介绍 Windows 窗口的一些选项的定义方法(FoxPro2.5 DOS 版为例)

启动屏幕生成器后,系统菜单中会增加一项 Screen. Screen 下面接着许多菜单项如: Screen Layout、Box、Field、Text、Push Button、Radio Button、Check Box 等,选中 Screen Layout 一项定义窗口的属性:

name, title, footer,

name 定义窗口名字,窗口名字最好起得有意

# 用 Browse 使编程量减掉一半

□ 齐 人

**数**据库中最常见的无非是四种操作：浏览查询、修改、录入、删除，而 FoxPro 中的 Browse 一条命令便实现了对数据库的所有这四种操作。可以说 Browse 是数据库中最常用也是功能最强的一条命令。用 Browse 可以打开一个浏览窗口从而显示一个库中的所有记录。你可以一条一条记录地浏览，也可以用 PgUp 和 PgDn 键快速浏览，更可以插入、删除、修改库中的数据，最后以 Ctrl+W 或 Ctrl+END 存储你的数据，以 Ctrl+Q 放弃你的数据。可以说会使用 Browse，FoxPro 的大部分应用系统都可相当简化。下文中我们将介绍 Browse 命令中的几个子句的用法，以及程序方式下如何利用 Browse 作数据库的插入、删除。

## 一、数据库字段子句的使用技巧

Browse 的字段子句语法为：

```
<field>[:R][:columnwidth]
[:V=<expr1>[:F][:E=<expcl>]]
[:P=<expc2>]
[:B=<expc2>,<expc3>[:F]]
[:H=<expc3>]
```

义而且跟其他窗口不重名

title 是窗口显示时，显示在窗口头部中间的一串字符。由于 FoxPro2.5 多运行在中西文兼容汉字系统下，所以 title 的输入时前后最好加一个空格，如：

‘学生管理窗口’

footer 是窗口显示时，显示在窗口末尾中间的一串字符，跟 title 一样最好在输汉字时前后各加一个空格。

其属性：□Setup……□clean&. Procs 一定要选择

Setup 是定义窗口启动前要做的许多工作，如给要录入的变量赋值、判断库存在不存在、网络上对要修改的记录加锁等等。

下面是一个系统中的 Setup 的语句：

```
set mouse on
nianji=1
if eof()
Wait Window '数据库为空' nowait
```

```
[:W=<expr1>]
[:,<field2>[:R]…]
```

其选项：V = <expr1>[:F][:E = <expcl>]]

对每个字段进行合法性检查。当光标移开一个字段域时，如果 expr1 取值为 (.T.)，则 FoxPro 认为刚才输入的数据是正确的，光标进入下一个字段。如果 expr1 取值为假 (.F.)，输入的数据被认为是不正确的，光标仍留在原来字段上，系统提示一个出错信息，如果 expr1 取值为 0，则没有出错信息。这里的 expr1 既可以是一个表达式也可以是一个函数。

如：

```
browse fields cno;V=cno>0
browse fields cno;V=TT()
```

TT() 是一个函数。在这个函数里可以对刚才录入的数进行更复杂的有效性检查。如判断录入的 cno 是否已重复等等。

:E 选项的含义是，当合法检查的表达式：V = expr1 取值为真 (.T.) 时，光标进入下一字段。如果表达式计算得值为假，系统会提示 expcl 而不提示 “Invalid

input”。这一点对国内的 FoxPro 的开发者相当有益，因为我们的应用系统给最终用户的提示信息不可能是英文 “Invalid Input”，需要改为更有意义的提示，诸如 “编号必须大于 0” 之类。下面的例子中用到了调用：V 选项。

```
close databases
use customer
browse fields cno;V=cno>0;
:E="编号必须大于0";
,Company;V=not empty(company);
:E="公司名字不能为空"
```

选项：H 对国内用户来说更有用处。因为 FoxPro 数据库的字段名一般都是英文字，而 Browse 命令弹出窗口中的每个字段都是英文显然不合适。还有一种情况，即使用了中文字段名，Browse 窗口中的字段名最好提示得详细一些。这些都需用：H 选项。Browse 命令在默认情况下，取数据库的字段名作为 Browse 窗口每列的台头。有了：H 选项，列的台头取作 expc3。下面的例子中，列的台头以 “公司名称” 取代了字段名 Company，“编号” 取代

return

endif

set filter to nianji>1

go top

clean up 是窗口关闭后要做的一些动作，如关闭数据库、释放掉不用的变量等。

下面是一个系统中的 clean up 的例子：

```
set mouse off
release nianji,banji
Set libr to
Set filter to
```

选项 type 一般也要定义，选中 type 后系统弹出一个窗口让你定义窗口式样，一般 type 选中 user (用户自定义类)。Attributes 选择 Shadow 这样窗口显示时带阴影，Border 有几种可选：

```
None 是无边界
Single 是单线框
```

Double 是双线框

Panel 是宽带框

System 是宽带框

由于有些中西文兼容汉字系统对表格线处理得不理想，一般选 Panel, System 较为理想。

选项：color schemes 最好不要选择，因为 FoxPro2.5 本身可以在 DOS、Windows 下跨平台运行 (可以在单显上、彩显上运行)，除非你已是 FoxPro2.5 的熟练使用者。这一点对 FoxPro2.5 的初学者甚为重要，望大家千万予以重视。

限于篇幅，FoxPro2.5 的 Screen Builder 中的其他选项的使用技巧不再介绍。最后提醒读者的是，如果 FoxPro2.5 的应用系统要跨平台运行时，系统中的所有界面建议使用 Screen Builder 生成，在系统编程上要习惯于使用 Project 和 Screen Builder。

了 no

```
close databases
use customers
browse fields no:H="编号",company:H="公司名称"
```

上面讲到的:V、:E 选项是对库记录的每个字段进行有效性检查,实际上在有些场合还需要对整条记录作有效性检查,此时我们可以使用 Browse 命令的 VALID 子句和 ERROR 子句。

其格式为:

```
Browse .....VALID expl2 ERROR expc5
```

VALID 子句当光标从本条记录移至另一条记录,且本条记录被修改时执行,如果 expl2 的计算结果为真,则光标可移至另一条命令;若结果为假,则光标仍停留在原来的记录位置上,并且系统会提示错误信息 expc5,下面是一个例子

```
Browse VALID no>0 and not empty
(company);
```

ERROR“编号必须大于0,公司名称不能为空”

## 二、改造 Browse 的插入、删除命令

在 Browse 命令的标准设置中,Fox-Pro 允许用 Ctrl+N, Ctrl+T 添加和删除记录。但当你建立一个复杂的应用系统时,可能希望不局限于这种标准的添加和删除记录机制,特别是为普通用户作应用系统时,如:添加一条记录时可能还需为记录的一些字段填上默认值,删除记录时还需从相关表中删除匹配的数据,这便需用到 FoxPro 的另一条命令。

```
ON KEY LABEL <keyname><command>
```

<keyname>定义功能键名,如 F1、F2、F4、Ctrl+F6、Alt+F1 等。<command>可以是任何 FoxPro 命令,不过常常是调用一个过程的 DO 命令。如下面的命令就以 F1、F2 作添加、删除键,用户按下这两个键就会调用名为 insrec 和 delrec 的两个过程

```
ON KEY LABEL F1 DO insrec
ON KEY LABEL F2 DO delrec
```

insrec 和 delrec 是两个过程。你可以把这些过程写在一个文件中,用 Set Procedure 命令来打开此过程。Browse 命令和 ON KEY LABEL 的命令应写在一起,而且在执行完 Browse 后应立即取消这些键的定义,否则用户每键入 F1 则系统自动进入 insrec 程序。取消键的定义很简单,以命令 ON KEY LABEL <keyname> 即可,如:

```
ON KEY LABEL F1
ON KEY LABEL F2
```

这几条命令看似简单,但在应用系统的编制过程中却非常有用,因为在脱离 Fox-Pro 环境而运行的系统中一般都用到

```
set sysmenu off
```

这样,系统中 Ctrl+N 和 Ctrl+T 便不能用来插入、删除数据库记录,简单的途径使 Browse 仍能作录入和删除便是用命令 ON KEY LABEL。ON KEY LABEL 命令非常有用,其编程模式上已不再是传统方式而是基于事件驱动方式,由 FoxBase,转至 FoxPro2.5 时编程模式上一定要有所转变,否则程序的编写质量很难提高。

## 三、示例程序

```
* MAILBROW.PRG
* PROGRAM TO UPDATE MAILING LIST
TABLE
* IN A BROWSE WINDOW
* WRITTEN:11/01/93
* Open and link tables
CLOSE DATABASES
USE Maillist IN 0 ORDER idcode
USE sales IN 0 ORDER idcode
* Establish add and delete routines
ON KEY LABEL Ctrl+A DO Addrec
ON KEY LABEL Ctrl+D DO Deleterec
ON KEY LABEL Ctrl+R DO Recallrec
BROWSE NOAPPEND NODELETE FIELDS;
: idcode :H="身份证号码";P="@!";
:W=EMPTY(idcode);
name:20;V=NOT EMPTY(name);
:E="姓名不能为空";
city:12;
state:P="@A!";
zip:V=Zipvalid(country,zip);F;
:E="无效的邮政编码"
arecode:H="A/C";
telephone:P="999-9999";
:W=NOT EMPTY(arecode);
firstdate:H="第一个日期"
lastdate:H="最新日期"
:B=,DATE();
years=(DATE()-firstdate)/365;
VALID NOT (EMPTY(name) AND EMPTY
(company));
```

```
AND lastdate>firstdate;
```

```
ERROR 姓名“公司名称必须写上,”+;
“最新日期必须在第一个日期后。”
```

```
* Cancel add and delete routines
```

```
on key label Ctrl+A
```

```
on key label Ctrl+D
```

```
on key label Ctrl+R
```

```
CLOSE DATABASES
```

```
RETURN
```

```
* Custom record add routine
```

```
PROCEDURE ADDREC
```

```
APPEND BLANK
```

```
REPLACE city WITH "San Francisco";
```

```
state WITH "CA",arecode WITH "415"
```

```
firstdate WITH DATE(),lastdate WITH
```

```
DATE()
```

```
RETURN
```

```
* Custom record delete routine
```

```
PROCEDURE Deleterec
```

```
SELECT sales
```

```
SEEK maillist.idcode
```

```
SELECT maillist
```

```
IF FOUND("sales")
```

```
WAIT WINDOW "因为块存在不能删除"
```

```
ELSE
```

```
DELETE
```

```
WAIT WINDOW "deleted"
```

```
ENDIF
```

```
RETURN
```

```
* Custom record recall routine
```

```
PROCEDURE Recallrec
```

```
RECALL
```

```
RETURN
```

```
* Validate ALP codes
```

```
FUNCTION zipvalid
```

```
PARAMETERS mcountry,mzip
```

```
mreturn=. T. && US Zip codes
```

```
DO CASE
```

```
CASE EMPTY(MCOUNTRY)
```

```
IF LEN(TRIM(mzip))<5 AND;
```

```
LEN(TRIM(mzip))<10
```

```
mreturn=. F. && Wrong length
```

```
ENDIF
```

```
CASE UPPER(mcountry)='CANADA'
```

```
IF LEN(TRIM(mzip))=10 AND;
```

```
SUBSTR(mzip,6,1)<"-
```

```
mreturn=. F. && Wrong length
```

```
ENDIF
```

```
ENDCASE
```

```
RETURN mreturn
```

```
* End of program MAILBROW.PRG
```

其中 ADDREC 过程很简单,它用 APPEND BLANK 命令添加一个新的空白记录,然后为一些字段填入缺省值。DELETREC 过程稍复杂些,它表示出了当你用两个相关表工作时的一个典型需求。当在 SALES 表中有匹配的记录时,该过程可防止从 MAILLIST 表中删除记录。ADDREC 过程选择了 SALES 表并用 SEEK 命令来查找与前邮寄表的记录相匹配的值。在返回到 MAILLIST 表后,则用 FOUND() 函数测试 MAILLIST 表。如果 FOUND() 函数返回 .T., 则此函数显示一个警告信息,否则就删除当前邮寄表的记录并显示一个确认信息。

WAIT WINDOW 命令很简单又很有用。WAIT 能延迟程序的执行并在屏幕上任意显示一个定制信息。如果添加了关键字 WINDOW, 信息会在屏幕的右上角显示,就象 FoxPro 自己的信息一样。

RECALLREC 过程很简单易懂。它只是简单地执行 RECALL 命令以恢复当前的记录。

# 用 FBASE 扩充 FoxPro 的图文声象功能

□肖 贻

Microsoft FoxPro 是建立在 XBASE 语言基础上的关系数据库系统,它特有的 Rushmore 技术、完全集成的 SQL、跨多种平台(如: DOS、Windows、Macintosh、UNIX)的特性、与 XBASE 的高度兼容性、多种强大的软件设计工具、真编译等特色使广大微机用户已纷纷转向 FoxPro。FoxPro 成为微机上公认的优秀的数据库系统,并已成为事实上的微机数据库标准。

Rushmore 使 FoxPro 成为微机上最快的桌上型 DBMS,它能处理百万条记录以上的大型数据库,速度可与大型机相媲美。FoxPro 支持多种平台,使开发者的应用系统的跨平台移植问题变得很容易。FoxPro 的 Screen Builder、Report Writer、Label Designer、Menu Builder、Project Manager 等工具,大大减少了开发者的工作量。FoxPro 的真编译特性,使应用系统可脱离 FoxPro 环境,近而又使程序的扩散和加密成为可能。

鉴于 FoxPro 的上述特点,我国的数据库用户正面临着从 XBASE 向 FoxPro 转变的紧迫局面。

据统计,我国 80% 的计算机运行着数据库管理系统,而应用最广泛的是 XBASE 系列数据库系统(如 dBASE、FoxBASE 等),它们早已深入人心, XBASE 系列数据库系统均在 DOS 上运行。今后几年仍有大量用户在 DOS 下开发各种数据库系统。

尽管 FoxPro 在性能上较 dBASE、FoxBASE 有了质的飞跃,然而令人遗憾的是 FoxPro DOS 版是一个文本模式下工作的数据库系统。这就决定了用户无法开发类似 Windows 的用户界面,更无法实现对图形、图象、声音汉字等多种媒体的管理,虽然 FoxPro Windows 版已初步解决了这些问题,而 DOS 版却实实在在地存在这方面的功能不足。鉴于国内 DOS 下的应用系统的开发需要,对 FoxPro DOS 版进行扩充使之能处理声音、图

形、图象、矢量字等多种媒体,而 FoxPro 原有的性能又保持不变,便很有必要且意义深远,现在市场上出售的北京新未来公司的 FBASE 就是这样一个 FoxPro 2.5 DOS 版的扩充产品。

FBASE 实际运行在 FoxPro 2.5 DOS 版下, FoxPro 2.5 的应用系统当然无需改动就可运行。利用这些扩充的命令和函数,可以管理二值、灰度、彩色和真彩色图象,并可对图象作压缩存储(JPEG 算法)解压缩显示,作图象的各种显示效果(如:无级缩放、旋转、游动、镜像等)也可管理声音、图形等其他多媒体信息。这些函数首次在 DOS 的数据库系统中引入矢量字,矢量字有多种字体,字显示时可无级缩放、旋转。操作界面清晰美观。其丰富的图形功能,足以生成各种风格的图形界面(类 Windows 风格),也能方便地实现各种圆饼图、直方图等统计图,尤其吸引人的地方是所有扩充功能可跟 FoxPro 的其他功能一样编译成 .EXE 文件在 DOS 下脱离 FoxPro 而运行。

FBASE 的这些性能特点,使之成为 FoxPro 2.5 下的一个强有力的开发工具, FoxPro 2.5 + FBASE 已成为 DOS 下的最佳数据库开发平台。本文着重介绍如何在 FoxPro 2.5 DOS 版中利用 FBASE 来管理各种图象信息。利用 FBASE 来扩充 FoxPro 2.5 DOS 版的界面设计、矢量汉字、声音功能另文介绍。

大家都知道,在数据库系统中处理图象(黑白、彩色照片、图片等)在许多行业中非常必要,如公安系统中的户籍管理、罪犯、通缉犯管理、交警大队的车辆驾驶管理,各部门的人事,房地产业的房产介绍等等。在 FoxBase、dBASE、clipper 中可以处理图象,但都非常困难,以致于系统无法真正使用。在 XBASE 系统升级到 FoxPro 2.5 以后,这个问题便有了较理想的解决方法,即是在 FoxPro 2.5 DOS 版下调用 FBASE 的函数。

## 1. 图象处理的概念

计算机处理的各种照片、图片有几种:一是二值图象,象黑白文稿、各种档案、设计图纸录入计算机后便是二值图象,图象中的每一点都只有黑白两色;一种是灰度图象,象人物的照片、标本的图片,这些图片黑白之间有一定的层次(可以有 16 级、256 级等几种);一种是彩色图片,彩色图片录入计算机时一般占的硬盘空间都很大,一幅 2 寸的彩照一般有 100k 左右,所以使用的图片管理系统对彩色图片一定要压缩。

向计算机录图片,典型的方式是通过扫描仪。目前市场上流行的扫描仪有手持、台式、滚筒(从扫描方式上分)几种,也可分为(二值、灰度、彩色)几种。一般手持扫描仪可扫二值、灰度、彩色图象,其价格比台式、滚筒便宜,而台式扫描仪可扫 A3、A4 幅面的图片,一般只扫灰度、彩色图片,滚筒式扫描仪最大可扫 A0 幅面的图片(如工程图纸)。

计算机要显示图象(静态),其显示卡必须是 VGA 以上显示卡如 VGA、TVGA、SVGA、ET4000 等。

如果对彩色图片要做旋转、游动、动画处理,内存最好是 4M 以上。

## 2. FoxPro 2.5 中存储图象的方法

FoxPro 2.5 DOS 版是一个字符方式下运行的软件,其本身并不具备图象处理能力,而 FoxPro 2.5 Windows 版是一个在图形方式下运行的软件,本身已初步具备了图象处理功能,但有些地方还需要用户自己做进一步加强。我们先介绍 DOS 版中处理图象的方法。

FoxPro 2.5 DOS 版中除了传统的数据类型如字符、数值、日期、逻辑型外还有一种 Memo 字段,其 Memo 字段可与其他类型一样进行增、删、改,在网络上也可进行并发控制,如加锁、解锁等。所以 FoxPro 2.5 的图象数据完全可以存放于 FoxPro 2.5 的 Memo 字段中,这样图象的存储问题无论是单机上、网络上都可以解决。如果自己解决图象的存储问题,便至



少会有以下弊病:

- ①数据一致性差,图象字段与其他相关字段不易保持一致。
- ②数据完整性差,尤其在网络上无法解决数据共享问题等等。

用 FoxPro 2.5 的 Memo 字段存储图象经常用到的命令是:

#### ①存贮

Append memo 图象字段 from \* \* \*  
\* \* \* 表示存贮图象的文件名字。

②另一条是 Append memo 图象字段 from \* \* \* OVERWRITE  
OVERWRITE 的含义是覆盖掉原来 Memo 字段的内容。

#### ③取图象数据:

copy memo 图象字段名 to 图象文件名。  
做此命令之前要把当前记录移到数据库的相应位置上。

#### ④整理数据

pack memo

所有的 Memo 字段的数据在 FoxPro 2.5 中都存在“库名.FPT”中。FoxPro 本身并不是每次操作都整理 Memo 字段占的硬盘空间,删除、修改操作都不整理,所以需自己做管理硬盘空间的工作。pack memo 删掉 FPT 不用的空间。

### 3. 图象的录入和显示

FoxPro 2.5 DOS 版中要录入图象有两种途径:一是通过扫描仪,二是通过文件读入。北京新未来公司开发的通用扫描仪驱动程序,可以支持目前市场上较流行的 Microtek、HP、TH、台湾 SPI 标准的扫描仪等多种扫描仪,驱动程序是纯中文界面,一目了然,它有下面几项功能:

- 设置扫描仪类型、扫描精度

扫描精度一般以每英寸多少点计算,每英寸点数越多图片越清晰,其占的硬盘空间越大。据我们的经验,照片的分辨率以 150DPI 较合适。

- 设置扫描照片的亮度、对比度
- 对获得的照片进行裁剪
- 存贮获得的照片
- 从文件中读取图片等

一般地,扫描仪驱动程序是一个 DOS 下的 .EXE 文件,在 FoxPro 2.5 下可以外部命令的方式进行调用。下面是一段程序,扫描驱动程序是 main

```
use student
append blank
dele file scano.img
!main
if (file('scano.img'))
append memo 照片 from scano.img;
overwrite
```

endif

运行 main 获得一幅图片的数据,图片数据存放在 scano.img 文件中,以数据库命令 append memo 将图片数据存于库中。

由于 FoxPro 2.5 是一套西文字符方式下的软件,如果在 FoxPro 2.5 下要显示图象必须切换屏幕模式,使屏幕模式处在图形方式。16色模式下可做灰度、二值图片处理,而 256 色模式下可做彩色图象处理。新未来公司的 FBASE 包括 FoxPro 2.5 下的一个函数库,它包括了对 FoxPro 扩充的近百个函数,调用这些函数便可方便地在 FoxPro 2.5 环境下显示图象文件或 memo 字段中的图象数据

由于 FBASE 函数库是 FoxPro 2.5 下的函数库,所以在启动 FoxPro 2.5 以后,加载 FBASE 函数库即可运行 FBASE 的所有函数,而 FoxPro 2.5 的所有命令和函数仍可照常使用。FBASE 用于图象显示的函数有下面几个:

\_DImage() 将二值、灰度、彩色图象的局部或整体显示在屏幕上。

\_CmpImage() 用国际标准 JPEG 算法压缩灰度及彩色图象

\_DmpImage() 用 JPEG 算法对压缩的灰度、彩色图象进行解压缩

\_ZoomImage() 对图象作无级缩放,并显示在屏幕上

\_GImgcols() 获取图象宽度

\_GimgRows() 获取图象类型

\_ImgFLIP-H() 设水平镜像标志

\_ImgFLIP-V() 设图象垂直镜像标志

\_SImgRotat() 置图象旋转角度

\_GImgRotat() 取当前图象旋转角度

\_SaveArea()

\_Restowin()

\_LoadImage()

\_Scrtofile()

\_FiletoScr()

\_Setvideomod()等

有了这些函数便可以做图象的各种显示效果如:

图象的整体、局部及游动显示

图象的局部、整体无级缩放

图象的旋转、镜像

图象在窗口内滚动显示等

下面是一段 FoxPro 2.5 的图象显示程序:

```
Procedure Image
=-Svideomod(Mode256)
* 设置模式
```

use demo

go 5

Width=-gimg cols(@image)

&& 取图象列数(宽度)

height=-gimgrows(@image)

&& 取图象高度

=-unimgflip()&& 取消镜像

=-Simgrotate(0)&& 不旋转

=-dimage (@image, 0, 0, 300, 300,0,0)&& 显示图象

=-uningflip()

=-Simgrotate(90)&& 旋转90度

=-dimage (@image, width, 0, 300,300,0,0)

=-inkey(2)

=-Svidemode(3)&& 恢复屏幕模式

当然为了使 FoxPro 2.5 DOS 版的图象处理效果更好,还需要用到一些图形、文字功能,这样可显示图文合一的用户界面,这些 FBASE 函数库中都有。其文字显示能处理二十几种字体,任意字号的矢量字,字体美观漂亮,加上图形功能中的画线、填充矩形、画圆、画点等操作, FoxPro 2.5 DOS 版的界面可以相当丰富,一改 FoxPro 2.5 DOS 版字符方式的界面风格。


由于 Foxpro 2.5 DOS 版对机器配置要求较低,加之国内用户对 DOS 环境较熟悉,所以它一段时间内会十分流行,而我们文中的让 FoxPro 具备图象处理能力的方法是一种较为理想的方案。此方案让 FoxPro 2.5 处理图象变得很简单,用户的所有精力可放在 FoxPro 的数据处理上,而图象的录入、存贮、显示只有几条命令和函数。更引人注目的是所有这些函数都可同 FoxPro 的其他命令函数一样可编译,用户的系统扩散、加密也不再有问题。

### 4. FoxBASE、dBASE 的系统如何增加图象处理能力

FoxBASE、dBASE 已在国内使用很久,用户已用它们开发了许多应用系统,这些系统凝聚了开发者的大量心血,它们的使用也已积累了大量的数据,如果为了使 FoxBASE、dBASE 能处理图象而废弃原有的系统显然是不现实的。方法是首先把原有系统移到 FoxPro 2.5 DOS 版下,这样原有的系统功能丝毫不变,而性能(查询速度等)大大提高,源程序又可真编译为 .EXE 文件。可以预计,很快 DOS 下的 FoxBASE、dBASE 将不再有人使用,大家都纷纷转向 FoxPro 2.5。

# FOXBASE+ 通用立体投影 下拉式菜单程序设计

□ 刘耀东

 多用户在开发系统时已愈来愈将用户界面的好坏作为评价系统优劣的重要因素之一,其中通常的选择之一是采用下拉式菜单。FOXBASE + V2.10 版虽然提供了下拉菜单设计的命令,但是由于系统汉化的问题,在移动子菜单时不能及时清除有关屏幕区域,因而几乎不能使用。此外,每设计一个系统,都要重新定义若干二维数组,很难进行维护。

基于上述考虑,本文提出了一种通用立体投影下拉式菜单程序设计方法,并在 FOXBASE+ V2.0 及以上版本实现,且成功地应用到多个系统开发之中,取得了高效、快速、可维护性强的效果。

## 一、系统设计思想

### 1. 下拉式菜单设计应考虑的问题

下拉式菜单设计应考虑的问题包括:

在屏幕指定的位置上按指定的颜色显示顶层菜单;定义所有欲显示的子菜单在屏幕上所开窗口的左上角、右下角的坐标,并按坐标值按指定颜色弹出窗口;在上述坐标定义的窗口区域内显示相应子菜单的栏目名称;用来在子菜单栏目中上下移动,按指定颜色分别设置当前光标、非当前光标所在栏目颜色。若欲显示子菜单的菜单项个数多于窗口所能显示的行数,则应滚动显示;回车键选中且执行相应操作;→、←键用来在子菜单之间进行移动,且按键后应清除前一子菜单所在屏幕区域,恢复正常显示;若欲显示的顶层菜单在一屏内显示不下,则可翻屏显示。

### 2. 通用菜单程序设计应考虑的问题

通用菜单程序设计应考虑的问题包括:

系统控制变量的读取、存储与程序无关;顶层菜单的显示坐标内容与程序无关;子菜单的窗口坐标、颜色控制显示内容与程序无关;系统功能发生变化或扩展时所做的修改或扩充与程序无关。

## 二、系统的设计实施

### 1. 数据库设计

为了做到所开发的系统有更大的独立性,一共设计了四个数据库,分别是系统变量及参数控制库 Var.dbf,顶层菜单每屏显示栏目首尾指针库 Pagewj.dbf,顶层菜单管理库 Cdk.dbf,子菜单管理库 Cdglwj.dbf,下面分别进行介绍。

系统变量及参数控制库 VAR.DBF 库结构如下:

```
Structure for database : VAR.DBF
Number of data records : 20
```

Last updated : 08/11/93 at 10:22

Field Field name Type Width Dec

1 VAR-NAME Character 10

2 VAR-TYPE Character 1

3 VAR-VAL Character 30

\*\*\* Total \*\*\* 42

VAR-NAME 用于存放 20 个系统调用变量及参数的名字。VAR-TYPE 用于存放 20 个系统调用变量及参数对应的数据类型。VAR-NAME、VAR-TYPE 字段不能修改,VAR-VAL 字段可以修改。顶层菜单每屏显示栏目首尾指针库 PAGEWJ.DBF 库结构如下:

Structure for database : PAGEWJ.DBF

Number of data records : 2

Last updated : 08/07/93 at 11:08

Field Field name Type Width Dec

1 PAGEUP Numeric 3

2 PAGEDN Numeric 3

\*\*\* Total \*\*\* 7

PAGEUP 字段用于存放顶层菜单每屏显示首项的菜单序号。PAGEDN 字段用于存放顶层菜单每屏显示尾项的菜单序号。PAGEWJ.DBF 库的记录个数为欲显示顶层菜单需显示的屏数。顶层菜单管理库 CDK.DBF 库结构如下:

Structure for database : CDK.DBF

Number of data records : 20

Last updated : 08/03/93 at 14:12

Field Field name Type Width Dec

1 CDMC Character 20

2 CDCX Character 20

3 CDJB Numeric 3

\*\*\* Total \*\*\* 44

CDMC 字段用于存放某一子菜单(CDJB 字段取同一值时)欲显示的菜单名称;CDCX 字段用于存放某一子菜单欲调用的 PRG 命令文件名称(不写扩展名);CDJB 字段用于存放各子菜单对应的顶层菜单序号。

子菜单管理库 CDGLWJ.DBF 库结构如下:

Structure for database : CDGLWJ.DBF

Number of data records : 7

Last updated : 08/10/93 at 9:35

Field Field name Type Width Dec

1 RR01 Numeric 2

2 RR02 Numeric 2

```

3 CLL01 Numeric 2
4 CLL02 Numeric 2
5 C01 Character 80
6 MS01 Character 80
7 RROW01 Numeric 2
* * Total * * 171

```

RR01.CLL01 字段用于存放顶层菜单对应的子菜单左上角坐标。RR02.CLL02 字段用于存放顶层菜单对应的子菜单右下角坐标。C01 字段用于存放顶层菜单欲显示的菜单名称。MS01 字段用于存放顶层菜单对应的提示帮助信息。RROW01 字段用于存放顶层菜单对应的显示行坐标。该库记录个数为顶层菜单个数。

CDK.DBF、CDGLWJ.DBF 库的内容可按照用户的不同要求进行修改。

## 2. 动态数组设计

存储在上述四个库中的数据,只能作为数据的存储仓库,系统调用时将上述数据转换为内存变量数组,以避免反复读取硬盘而影响显示速度。

全局变量说明

JBMAX:顶层菜单栏目个数,不限。

JB:顶层菜单栏目序号指针,最大值为 JBMAX。

PGMAX:欲显示的顶层菜单的最大屏数。

PTR:某一子菜单当前光标所在栏目指针。

动态变长数组说明

MENUUM(JBMAX):存放各子菜单的菜单项个数数组。

RR1(JBMAX)、RR2(JBMAX)、CLL1(JBMAX)、CLL2(JBMAX)用于存放各子菜单欲开窗口的左上角、右下角坐标数组。

MS(JBMAX):顶层菜单提示信息数组。

RROW(JBMAX):顶层菜单行坐标数组。

C1(JBMAX):顶层菜单显示菜单名称数组。

PGUP(PGMAX):顶层菜单每屏显示菜单首项指针数组。

PGUP(PGMAX):顶层菜单每屏显示菜单尾项指针数组。

&CDHZSZ(MENUUM(JB)):存放子菜单菜单名称动态数组,共有 JBMAX 个数组,每个数组维数各顶层菜单对应的子菜单菜单个数。

&CDPGSZ(MENUUM(JB)):存放子菜单程序名称动态数组,共有 JBMAX 个数组,每个数组维数各顶层菜单对应的子菜单菜单个数。

上述两个数组的命名规则如下:

CDHZSZ="CDHZ"+IIF(JB<10,1,IIF(JB<100,2,IIF(JB<1000,3,4)))

CDPGSZ="CDPG"+IIF(JB<10,1,IIF(JB<100,2,IIF(JB<1000,3,4)))

这里 JB 子菜单所对应的顶层菜单序号,IIF 为 FOXBASE 中的 IIF 函数。由命名规则可知:只要内存空间足够,最多可定义 9999 个顶层菜单。数组 &CDHZSZ、&CDPGSZ 的维数仅与相应子菜单的菜单实际个数有关,没有一点浪费,因而大大节约了内存空间。

## 3. 显示屏数不受限制的菜单设计

在进行菜单系统设计时,如欲显示的顶层菜单项目太多,以至于难以在一屏内显示,这时就要考虑分屏显示的问题。在此,我们通过设计四个指针变量:顶层菜单序号指针变量 JB,显示屏页号指针变量 PAGE,顶层菜单每屏显示的首项指针变

量数组 PGUP(PAGE),尾项指针变量数组 PGDN(PAGE),当按键为→、←时,上述指针变量分别随之变化,以达到控制分屏显示的目的。由于控制显示屏数的参数 PGMAX 放在 VAR.DBF 库中,不依赖于程序,故可任意分屏显示。这里仅以按键为"→"时为例,来说明实现该功能的原理。步骤为:读入"→"键值;若当前光标所在子菜单对应的顶层菜单序号指针满足:JB>PGUP(PAGE)且 JB<PGDN(PAGE),则按"→"键前顶层菜单位于第 PAGE 屏,JB←JB+1,PAGE 值不变,弹出 JB 对应的子菜单;如果 JB=PGDN(PAGE),则若 PAGE=PGMAX,PAGE←1,JB←1,清屏,显示第一屏顶层菜单,弹出 JB 对应的子菜单;否则,JB←JB+1,PAGE←PAGE+1,清屏,显示第 PAGE 屏顶层菜单,弹出 JB 对应的子菜单。

## 4. 子菜单的滚动设计

在子菜单数目多于欲显示菜单数时,当按键为"↑","↓"时,子菜单应具有屏幕滚动功能。这一功能通过设置显示栏目首"尾"指针 pt1、pt2,当前光标所在子菜单栏目指针 ptr 及若干控制变量来实现。设子菜单欲显示菜单行数为 xshs,子菜单数为 mnum,在此仅考虑按键为↓时的情形,分三种情况:

(1)  $1 \leq ptr \leq xshs$  (2)  $xshs < ptr < mnum$  (3)  $ptr = mnum$

对于第一种情况,当前光标所在子菜单栏目指针进行加一操作,即  $ptr \leftarrow ptr + 1$ ,pt1、pt2 值不变。此时整个子菜单不应滚动,按"↓"键前的 ptr 所指向的菜单栏目设置为非光标所在行的颜色,ptr+1 指向的菜单栏目应设为光标所在行的颜色,其余栏目颜色设置不变。

对于第二种情况, $pt1 \leftarrow pt1 + 1$ , $pt2 \leftarrow pt2 + 1$ , $ptr \leftarrow ptr + 1$ ,清除子菜单所在屏幕区域,显示序号从 pt1 至 pt2 的子菜单,光标定位在 pt2 所指的菜单栏上,此时 pt2 指向的菜单设置为光标所在行的颜色,其余栏目颜色设置为非光标所在行的颜色。

对于第三种情况 pt1、pt2、ptr 值不变,屏幕不滚动。

## 三、系统的使用

在完成系统的设计后,使用之前应完成下列操作:

1. 正确将有关系统控制变量及参数输入到 VAR.DBF 库中。

2. 将每屏显示的顶层菜单库首尾指针值输入到 PAGEWJ.DBF 库中。

3. 将顶层菜单库提示帮助信息对应的显示名称。各子菜单欲在屏幕上显示的窗口左上角、右下角坐标值输入到 CDGLWJ.DBF 库中。

4. 将子菜单对应的顶层菜单序号欲显示的菜单名称。调用的 PRG 命令文件名称输入到 CDK.DBF 库中。

5. 调整好 FOXBASE+ 系统配置文件 CONFIG.FX 有关内存配置的参数,以免使用系统发生内存不够的情形。

本通用程序在 FoxBASE+V2.0 及以上版本。UCDOS 2.0 汉字系统,HP386 上调试通过。

## 源程序清单

```

* : xledxt.prg
close all
close data
set alter off
set bell off
set talk off

```

```

set cons off
set safety off
set escape off
set status off
set scor off
public jymax,page,pt1,pt2,ptr,gbkzbl,jb,pgmax
&& 读入有关的系统变量及参数
use var
ii=1
do while ii<=reccount()
    store var—name to ty...
    if upper(ltrim(trim(var—type)))="N"
        store val(var—val) to &ty
    else
        store var—val to &ty
    endif
    ii=ii+1
skip
enddo
use
public dime menuum(jymax),rr1(jymax),rr2(jymax),
    ll1(jymax),cll2(jymax),
    ,c1(jymax),ms(jymax),rrow(jymax),
    pgup(pgmax),pgdn(pgmax)
sele a
use pagewj
reco=reccount()
&& 读入顶层菜单每屏显示栏目之首尾指针变量数组
go top
page=1
do while . not. eof()
    store pageup to pgup(page)
    store pagedn to pgdn(page)
    skip
    page=page+1
enddo
use
&& 读入各子菜单栏目名称. 调用 prg 文件名于动态数组
page=1
sele a
use cdk
sele b
use cdglwj
sele a
do while jb<=jymax
    cdhysz="cdhz"+iif(jb<10,1,iif(jb<100,2,iif(jb<1000,3,
    4)))
    cdpysz="cdpg"+iif(jb<10,1,iif(jb<100,2,iif(jb<1000,3,
    4)))
    set filt to cdjb=jb
    coun to menuum(jb)
    go top
    public dime &cdhysz(menuum(jb)),&cdpysz(menuum(jb))
    tt=1
    do while tt<=menuum(jb)
        &cdhysz(tt)=ltrim(trim(cdmc))
        &cdpysz(tt)=ltrim(trim(cdcx))
        tt=tt+1

```

```

skip
enddo
sele b
&& 读入各子菜单的窗口左上角右下角坐标
rr1(jb)=rr01
rr2(jb)=rr02
cll1(jb)=cll01
cll2(jb)=cll02
&& 读入顶层菜单名称. 提示信息. 顶层菜单之横坐标
c1(jb)=ltrim(trim(c01))
ms(jb)=ltrim(trim(ms01))
rrow(jb)=rrow01
skip
sele a
jb=jb+1
enddo
jb=1
close all
set colo to &xtcolo
clear
do while . t.
    set colo to &xtcolo    && 设置系统显示颜色
    if gbkzbl=0
        @ xtrow1,xtcol1 clea to xtrow2,xtcol2
    endif
    ii=pgup(page)
    do while ii<=pgdn(page)
        set colo to &menucolo
        @ coll0,rrow(ii) get c1(ii)
        ii=ii+1
    enddo
    && 显示顶层菜单标题
    ii=pgup(page)
    clea gets
    set colo to &choice    && 设置状态提示信息栏颜色
    @ mesrow,0
    @ mesrow,mescol say space((40-len(ms(jb))/2))+ms(jb)
    cdhysz="cdhz"+iif(jb<10,1,iif(jb<100,2,iif(jb<1000,3,4)))
    cdpysz="cdpg"+iif(jb<10,1,iif(jb<100,2,iif(jb<1000,3,4)))
    @ menurow,rrow(jb) say c1(jb)
&& 调用子菜单窗口处理函数
    do xlcd with rr1(jb),cll1(jb),rr2(jb),cll2(jb),menuum(jb),
        jb,jymax,backcolo,boxcolo,rowcol1,rowcol2
        set colo to &xtcolo
    enddo
* : eof: xlcdxt. prg
* : xlcd. prg

para brow1,bcol1,brow2,bcol2,mnum,jb,jymax,
    backcolo,boxcolo,rowcol1,rowcol2
&& brow1. bcol1=窗口左上角坐标
&& brow2. bcol2=窗口右下角坐标
&& mnum =子菜单的菜单项个数
&& JB =顶层菜单栏目序号指针
&& JBMAX =顶层菜单栏目个数
&& backcolo =窗口阴影部分颜色
&& boxcolo =窗口框线部分颜色

```

```

&&.rowcol1 =非光标所在行的颜色
&&.rowcol2 =光标所在行的颜色
&&.设置键盘控制变量.开窗口
if mnum>1
    do box with brow1,bcol1-1,brow2,bcol2,backcolo,boxcolo
        keychoice="pd # 4. and. pd # 24. and. pd # 5. and. pd # 13.
            and. pd # 19. and. pd # 27"
else
    keychoice="pd # 13. and. pd # 4. and. pd # 19"
endif
xshs=brow2-brow1-1 &&.确定应显示子菜单栏目个数
pt1=1 &&.为显示子菜单栏目首指针赋初值
ptr=1 &&.为当前光标所在子菜单栏目指针赋初值
pt2=xshs &&.为显示子菜单栏目尾指针赋初值
bzz=0 &&.子菜单滚动控制变量
bzkz=0 &&.显示状态控制变量
pd=0 &&.读入键值变量
do while .t.
    if mnum>1
        if bzz=0. and. (pd=24. or. pd=5. or. pd=0)
            ptr=pt1
            ip=1
            do while ip<=xshs
                if (ptr=pt1. and. bzkz=0). or. (pt1 # 1. and. ip=xshs.
                    and. pd=24)
                    &&.设置非光标所在行的颜色
                    set colo to &.rowcol1
                else
                    &&.设置光标所在行的颜色
                    set colo to &.rowcol2
                endif
                &&.显示子菜单栏目
                @ brow1+ip,bcol1+1 say &.cdhysz(ptr)+space
                    (bcol2-bcol1-len(&.cdhysz(ptr))-1) if (ptr=
                        pt1. and. bzkz=0). or. (pt1 # 1. and. ip=xshs. and.
                            pd=24)
                    row1=row()-1
                endif
                ptr=ptr+1
                ip=ip+1
            enddo
            ptr=iif(bzkz=0,ptr-xshs,iif(pd=24,pt2,iif(pd=5,
                pt1,ptr)))
        endif
        @ brow1+row1-1,bcol1+1 say "" &&.光标定位
    endif
    pd=0
    &&.读入控制键值
    do while &.keychoice
        clea typea
        pd=inkey(0)
    enddo
    do case
        case pd=4. or. pd=19
            gbkbzl=1
            &&.清除按→.←键前前一子菜单所在窗口区域
            set colo to &.xtcolo
            @ brow1,bcol1-3 clea to brow2+2,bcol2+2

```

```

do case
    case pd=4 &&.输入键为→
        do case
            case jb<pgdn(page) &&.顶层菜单序号加1
                jb=jb+1
            case jb=pgdn(page)
                clear
                gbkbzl=0
                if page=pgmax
                    page=1 &&.页码指针指向第一屏
                    jb=1 &&.顶层菜单指向首项
                return
            endif
            jb=jb+1 &&.顶层菜单序号加1
            page=page+1 &&.页码指针指向下一屏
        endcase
    case pd=19 &&.输入键为←
        do case
            case jb<=1
                page=pgmax &&.页码指针指向最后一屏
                clea
                jb=pgup(pgmax) &&.顶层菜单指向最后一屏项
            case jb>pgup(page)
                jb=jb-1 &&.顶层菜单序号减1
            case jb=pgup(page). and. page>1
                page=page-1 &&.页码指针指向前一屏
                jb=jb-1 &&.顶层菜单序号减1
            gbkbzl=0
            clea
        endcase
    endcase
    return
case pd=13 &&.输入键为回车键
    progame=&.cdpgsz(ptr)+'.prg'
    if .not. file(progame)
        @ mesrow,20 say "不存在欲调用的命令文件,按任一健
            返回!"
        clea typea
        pd=inkey(0)
        @ mesrow,0
        set colo to &.xtcolo
        return
    endif
    do &.progame &&.调用相应的 prg 命令文件
    clea
    return
case pd=24 &&.输入键为↓键
    ptr=ptr+1
    ji=ptr
    pt3=pt2+1
    bzkz=iif(ptr>pt2,1,bzkz)
    bzz=iif(ptr>pt2,iif(pt3>mnum,1,0),1)
    pt1=iif(ptr>pt2,iif(pt3>mnum,pt1,(pt1+1)),pt1)
    pt2=iif(ptr>(pt3-1),iif(pt3>mnum,mnum,(pt2+1)),
        pt2)

```

(下转第47页)

# 栏目编辑的话

随着计算机技术的不断发展和应用日益深入,计算机安全技术更显重要,它已经成为计算机用户维护自身利益和确保系统正常运行的必不可少的技术。本栏目向广大读者介绍计算机安全方面的最新实用技术和实现自身系统安全的具体编程技术与技巧,如加密技术、反跟踪技术、反病毒技术,以及具体病毒的检测和消除技术等。

## 预测未来病毒

□王振祥

**病**毒一般分为引导区引导的病毒、文件传染的病毒和复合型传染的病毒(既传染文件又传染引导区)三类,本文不讨论只感染主引导区的病毒,而只讨论感染文件的病毒(只感染运行文件或既感染运行文件又感染引导区)感染文件的方法,根据病毒传染运行文件的方法,提出了一种预测未来的传染运行文件病毒的方法,这就是许多病毒卡所声称的能预防未来病毒的方法。

病毒的危害在于它的传染性和破坏性,如果没有传染性,那么其危害也就极其有限了,因此病毒的危害主要在于它的传染性。病毒传染文件时,如果只传染某一固定文件,那么其危害也就是只限于该文件,这也不符合病毒的“精神”(当然 COMMAND.COM 作为特殊的系统文件,属于例外),因此病毒传染文件时,必先随机寻找一个运行文件(.COM 文件或 .EXE 文件)。一般病毒不会传染数据文件,因为数据文件没有特定的环境得不到执行,从而也就不能传播,当然病毒可用此方法破坏数据文件。病毒在随机寻找到一个运行文件时,将其打开然后将病毒程序写入该运行文件中,但实际工作中,需要对运行文件进行修改的情况极少,就目前所知,有以下两种情况需要修改运行文件:

(1) LINK 程序会修改运行文件。

(2) 某些对运行文件增加外壳的程序会修改运行文件,比如:为运行文件增加外壳以使之具有反跟踪功能等,程序使得编程者不必重复编写某些程序段,比如反跟踪程序段,而只需由该类程序为其增加一个外壳即可。

但这两类情况在修改运行文件时均是对指定的运行文件进行修改,而不是随机取一个运行文件进行修改,这是区分病毒修改运行文件和普通程序修改运行文件的重要标志。根据这一思想,笔者提出以下方法预测未来传染文件的病毒。

病毒程序随机取一个运行文件,这需要通过查找功能来实现,因此我们可拦截查找功能,当查找 \*.COM 或 \*.EXE 时,我们将查找得到的文件名字记录下来。病毒感染文件之前

必须先打开文件,因此我们再拦截文件的打开功能,当打开文件时我们比较要打开的文件名与随机查找得到的文件名是否相同(即是否打开查找得到的文件),如果是打开查找得到的文件,就将该文件的文件号记录下来,我们再将控制写文件的功能,当对查找得到的文件进行写操作时不予执行而给予报警。

DOS 为文件操作提供了两种不同的方法:(1)通过文件控制块 FCB 进行;(2)通过文件名进行。目前绝大多数的程序都是通过第二种方法进行文件操作,但由于下面的两种原因:有些特殊的应用只有通过 FCB 才能实现;第二种方法只有 2.0 版及更高版本才提供。所以为保证全面性,需要对这两种文件操作均进行控制。

上面讲述的所谓文件号是针对第二种方法而言的,对使用 FCB 的文件操作,在控制写操作时,可通过比较文件名来决定是否是查找得到的文件。

该方法能严格区分是病毒修改运行文件还是普通程序修改运行文件,当然不排除会有应用程序随机查找一运行文件然后对其进行修改,但就目前所知,尚未见到这种应用程序,因此该方法的误报情况极少,用该方法预测未来的传染文件的病毒是可靠的。

下面的程序 NEWVIRUS.ASM 实现了该思想。

NEWVIRUS.ASM 源程序如下:

```
ON      EQU    0FFH
OFF     EQU    0
CMPAHJP MACRO CMD _VAL, JUMP _LABEL

        CMP    AH, CMD _VAL
        JZ     $ + 5
        JMP    JUMP _LABEL
    ENDM
CALL _OLD21 MACRO
    PUSHF
    CALL      DWORD PTR CS, 0FF21
    ENDM
XCHG DSES MACRO
    PUSH     DS
    PUSH     ES
    POP      DS
    POP      ES
    ENDM
```



```

PUSHALL    MACRO
    PUSH    AX
    PUSH    BX
    PUSH    CX
    PUSH    DX
    PUSH    DS
    PUSH    ES
    PUSH    SI
    PUSH    DI
    PUSHF
    ENDM

POPALL     MACRO
    POPF
    POP     DI
    POP     SI
    POP     ES
    POP     DS
    POP     DX
    POP     CX
    POP     BX
    POP     AX
    ENDM

CODE       SEGMENT
    ASSUME  CS:CODE,DS:CODE
    ORG     100H

START:     JMP     INSTALL
OPEN _FCB  DB      OFF
OPEN _HDL  DB      OFF
FCB _SEARCH DB     OFF
HDL _SEARCH DB     OFF
FCB _FIRST DB     OFF
HDL _FIRST DB     OFF
FILE _NAME DB      64 DUP(0)

NAME _POS  DW      ?
HANDLE     DW      ?
COM        DB      'COM'
EXE        DB      'EXE'
SYS        DB      'SYS'
FCB _ADDR  DW      ?

CLEAN _BUFFER  PROC
    PUSHALL
    CLD
    PUSH     CS
    POP      ES
    LEA      DI,FILE _NAME
    MOV      CX,64
    XOR      AL,AL
    REP      STOSB
    POPALL
    RET
CLEAN _BUFFER  ENDP

CHECK _COMEXE  PROC
    MOV      CX,8
    CLD

CMP _LOOP:
    LODSB
    CMP      AL,'?'
    JNZ     NOT _STAR
    LOOP    CMP _LOOP
    PUSH     SI
    LEA      DI,COM
    MOV      CX,3
    REPZ    CMPSB
    POP      SI
    JZ      CERET
    LEA      DI,EXE
    MOV      CX,3
    REPZ    CMPSB

CERET:
    RET

NOT _STAR:
    MOV      AL,1
    OR       AL,AL
    RET

CHECK _COMEXE  ENDP

```

```

CHECK _FCB    PROC
    PUSH     CS
    POP      ES
    XCHGDSSES
    LEA      SI,FILE _NAME
    MOV      DI,DX
    CMP      ES:BYTE PTR[DI],0FFH
    JNZ     NOT _EXT _FCB1
    ADD      DI,6
    NOT _EXT _FCB1
    INC      DI
    MOV      CX,11
    CLD
    REPZ    CMPSB
    RET

CHECK _FCB    ENDP

CHECK _PATH    PROC
    PUSH     CS
    POP      ES
    XCHGDSSES
    MOV      DI,DX
    XOR      AL,AL
    CLD
    REPNZ   SCASB
    MOV      CX,DI
    SUB      CX,DX
    MOV      DI,DX
    LEA      SI,FILE _NAME
    REPZ    CMPSB
    RET

CHECK _PATH    ENDP

NEW21         PROC
    CMPAHJP 11H,IS12?
    MOV      CS,FCB _ADDR,DX
    CALL _OLD21
    PUSHALL
    MOV      SI,CS,FCB _ADDR
    LODSB
    CMP      AL,0FFH
    JNZ     NOT _EXT _FCB
    ADD      SI,6 ;extend fcb 0-5
    NOT _EXT _FCB:
    PUSH     CS
    POP      ES
    CALL     CHECK _COMEXE
    JZ      SET _SEARCH _FLAG
    MOV      CS,FCB _FIRST,0FF
    MOV      CS,FCB _SEARCH,0FF

COMM _IRET:
    POPALL
    RETF     2 ;OLD HERE IS:iret

SET _SEARCH _FLAG:
    PUSH     CS
    POP      DS
    MOV      FCB _SEARCH,ON
    MOV      FCB _FIRST,ON
    MOV      AH,2FH ;GET DTA ADDRESS IN ES:BX
    CALL _OLD21
    MOV      SI,BX
    XCHGDSSES
    INC      SI
    CALL     CLEAN _BUFFER
    LEA      DI,FILE _NAME
    MOV      CX,11
    CLD
    REP      MOVSB;SAVE SEARCH FILE NAME
    JMP      COMM _IRET

IS12?
    CMPAHJP 12H,IS4E?
    CMP      CS,FCB _FIRST,ON
    JZ      NEXT _OK
    JMP      OLD21

NEXT _OK:
    CALL _OLD21
    PUSHALL
    JMP      SET _SEARCH _FLAG

IS4E?:
    CMPAHJP 4EH,IS4F?

```

```

CALL-OLD21
PUSHALL
CALL CLEAN-BUFFER
PUSH DS
POP ES
MOV DI,DX
MOV CX,64
XOR AL,AL
CLD
REPNZ SCASB
MOV CX,DI
SUB CX,DX
STD
DEC DI
MOV AL,'\'
REPNZ SCASB
JZ TRANS-PATH
MOV SI,DX
CMP BYTE PTR [SI+1],','
JNZ CURRENT-DIR
LEA DI,FILE-NAME
MOV AX,[SI]
MOV CS:[DI],AX
ADD DI,2
SAVE-POS:
MOV CS:NAME-POS,DI
JMP JUDGE-NAME
TRANS-PATH:
MOV SI,DX
MOV CX,DI
SUB CX,SI
INC CX
INC CX
CLD
LEA DI,FILE-NAME
PUSH CS
POP ES
REP MOVSB
JMP SAVE-POS
CURRENT-DIR:
LEA DI,FILE-NAME
JMP SAVE-POS
JUDGE-NAME:
MOV AH,2FH
CALL-OLD21
PUSH CS
POP DS
XCHGDS
MOV SI,BX
INC SI ;DTA 0-20 ARE FCB,DOS USE INNER-AL
CALL CHECK-COMEXE
JZ COMEXEOK
MOV CS:HDL-FIRST,OFF
MOV CS:HDL-SSEARCH,OFF
JMP COMM-IRET
COMEXE-OK:
MOV CS:HDL-SEARCH,ON
MOV CS:HDL-FIRST,ON
MOV SI,BX
ADD SI,30 ;DTA+30 IS FILE NAME
MOV DI,CS:NAME-POS
MOV CX,42-30+1
CLD
REP MOVSB
JMP COMM IRET
IS4F?:
CMPAHJP 4FH,IS0F?
CMP CS:HDL-FIRST,ON
JZ HDL-SEAR-NEXT
JMP OLD21
HDL-SEAR-NEXT:
PUSHALL
JMP COMEXE-OK
IS0F?:
CMPAHJP 0FH,IS3D?
CMP CS:FCB-SEARCH,ON
JZ FCB-OPEN
JMP OLD21
FCB-OPEN:
CALL-OLD21
PUSHALL
CALL CHECK-FCB
JNZ FCB-OK1
MOV CS:OPEN-FCB,ON
FCB-OK1:
JMP COMM-IRET
IS3D?:
CMPAHJP 3DH,IS15?
CMP SL,1
JNB CMP-IT
JMP OLD21
CMP-IT:
CMP CS:HDL-SEARCH,ON
JZ F3D-ON
JMP OLD21
F3D-ON:
CALL-OLD21
MOV CS:HANDLE,AX
PUSHALL
CALL CHECK-PATH
JNZ HDL-OK
MOV CS:OPEN-HDL,ON,IS SEARCH FILE
;SET IT,OTHER
;NOT CHANGE
HDL-OK:
JMP COMM-IRET
IS15?:
CMPAHJP 15H,IS22?
FCB-WRITE1:
CMP CS:OPEN-FCB,ON
JZ FCB-WRITE
JMP OLD21
FCB-WRITE:
PUSHALL
CALL CHECK-FCB
JZ BEEP
POPALL
JMP OLD21
BEEP:
PUSH AX
PUSH BX
MOV AX,0E07H
INT 10H
XOR AH,AH
INT 16H
POP BX
POP AX
RETF 2
IS22?:
CMPAHJP 22H,IS40?
JMP FCB-WRITE1
IS40?:
CMPAHJP 40H,IS10?
CMP CS:OPEN-HDL,ON
JZ CHECK-HDL
JMP OLD21
CHECK-HDL:
CMP BX,CS:HANDLE
JZ BEEPI
JMP OLD21
BEEPI:
PUSH BX
PUSH CX
MOV AX,0E07H
INT 10H
MOV AX,0E07H
INT 10H
XOR AH,AH
POP AX
POP BX
CLC
RETF 2
IS10?:

```

```

CMPAHJP 10H,IS3E?
CALL-OLD21
CMP     CS,OPEN-FCB,ON
JZ      F10-CHECK
RETF    2
F10-CHECK:
PUSHALL
CALL    CHECK-FCB
JZ      INIT-FCB
JMP     COMM-IRET
INIT-FCB:
MOV     CS,OPEN-FCB,0FF
JMP     COMM-IRET
IS3E?:
CMPAHJP 3EH,ISFF?
CMP     CS,OPEN-HDL,ON
JZ      F3E-CHECK
JMP     LOD21
F3E-CHECK:
CMP     CS,HANDLE,BX
JZ      INTIT-HDL
JMP     OLD21
INTIT-HDL:
MOV     CS,OPEN-HDL,0FF
JMP     OLD21
ISFF?:
CMP     AX,-1
JNZ     OLD21
XOR     AX,AX
IRET
OLD21:
DB      0EAH
OFF21   DW      ?
SEG21    DW      ?
NEW21    ENDP
ALREADY  DB      10,13,' NEW VIRUS DETECTIVE
          ALREADY INSTALLED?',7,10,13,' $
INSTALLED DB      10,13,' NEW VIRUS DETECTIVE
          INSTALLED!',13,10,' $ '
INSTALL:
MOV     AX,-1
INT     21H
OR      AX,AX
JNZ     INSTALL-IT
LEA     DX,ALREADY
MOV     AH,9
INT     21H
MOV     AH,4CH
INT     21H
INSTALL-IT:
MOV     AX,3521H
INT     21H
MOV     0FF21,BX
MOV     SEG21,ES
LEA     DX,NEW21
MOV     AX,2521H
INT     21H
LEA     DX,INSTALLED
MOV     AH,9
INT     21H
LEA     DX,ALREADY
INC     DX
INT     27H
CODE
ENDS
END      START

```

(上接第43页)

```

ptr=iif(ptr>(pt3-1),iif(pt3>mnum,(pt3-1),ptr),ptr)
if ji<=(pt3-1)
  && 设置光标所在行的颜色
  set colo to &rowcol2
  @ brow1+row1-1,bcol1+1 say &cdhzzsz(ptr-1)+space
    (bcol2-bcol1-len(&cdhzzsz(ptr-1))-1)
  && 设置非光标所在行的颜色
  set colo to &rowcol1
  @ brow1+row1-2,bcol1+1 say &cdhzzsz(ptr)+space
    (bcol2-bcol1-len(&cdhzzsz(ptr))-1)
  row1=row1+1
endif
case pd=5 && 输入键为 ↑ 键
  bzkz=1
  bzz=1
  if pt1<ptr. and. ptr<=pt2
    ptr=ptr-1
    && 设置非光标所在行的颜色
    set colo to &rowcol2
    @ brow1+row1-1,bcol1+1 say &cdhzzsz(ptr+1)+space
      (bcol2-bcol1-len(&cdhzzsz(ptr+1))-1)
    && 设置光标所在行的颜色
    set colo to &rowcol1
    @ brow1+row1-2,bcol1+1 say &cdhzzsz(ptr)+space
      (bcol2-bcol1-len(&cdhzzsz(ptr))-1)
    row1=row1-1
  else
    if ptr=pt1. and. ptr>1
      bzkz=0
      pt1=pt1-1
      pt2=pt2-1
      bzz=0

```

```

ptr=pt1
endif
endif
endcase
gbkzbl=0
enddo
* : box. prg
parameters row01,col01,row02,col02,backcolo,boxcolo
&& row01.col01=窗口左上角坐标
&& row02.col02=窗口右下角坐标
&& 汇制显示子菜单窗口阴影部分
&& backcolo=窗口阴影部分颜色
&& boxcolo=窗口框线部分颜色
set colo to &backcolo
@ row01+1,col01-2 clea to row02+1,col01-1
@ row02,col01-2 clea to row02+1,col02-2
&& 汇制显示子菜单窗口
set colo to &boxcolo
@ row01,col01 say 'z'
@ row01,col01+2 say replicate(' ',(col02-col01)/2)
@ row01,col02 say 'C'
row=row01+1
do while row<row02
  @ row,col01 say 'n'
  @ row,col02 say 'n'
  row=row+1
enddo
@ row02,col01 say 'G'
@ row02,col01+2 say replicate(' ',(col02-col01)/2)
@ row02,col02 say 'K'
return

```

# CPAV 免疫功能剖析

□付小兵

**C**PAV 是一个集病毒的检测、清除、免疫于一体的实用软件,它采用窗口菜单式的用户界面,使用起来十分方便。它的免疫功能主要包括:①对引导扇区加免疫;②对 EXE 文件加免疫;③对 COM 文件加免疫。关于其实现操作,在该软件的帮助信息及许多书本上都有介绍,这里不再多述。本文仅从 CPAV 给 COM 文件增加的免疫功能来进行分析和评价。

## 一、免疫过程及功能分析

CPAV 给 COM 文件加免疫功能是通过替换 COM 文件的前 0EH 个字节,使控制首先转向免疫程序代码,增加的免疫代码共 998 个字节,附加在文件尾部。

在这里,我们首先区分以下几个提法:

(1)原文件的头 0EH 个字节代码:指文件加免疫前的头 0EH 个字节,它保存于免疫代码中。

(2)正常文件的头 20H 个字节代码:指 CPAV 给 COM 文件加免疫后的头 20H 个字节,它有一个副本保存于免疫代码中,主要用来同现文件的头 20H 个字节进行比较,以及在免疫的自重建功能中恢复正常文件头时使用。

(3)现文件的头 20H 个字节代码:指加免疫功能后的文件,在以后执行时的头 20H 个字节,它完全有可能已被病毒修改或替换掉了,因此免疫程序要检查它的真实性。

(4)原文件长度:COM 文件加免疫前的文件长度,它保存在正常文件头的 04H 处。

(5)正常文件长度:COM 文件加免疫后的文件长度,它等于原文件长度加上 998 个字节之和。它保存在免疫代码中。

(6)现文件长度:COM 文件执行时的文件长度,它可能被病毒改变了。免疫程序将现文件长度同正常文件长度进行比较,以判定该文件是否被修改过。

(7)正常执行流程:COM 文件加免疫功能前的执行流程。

(8)恢复正常执行流程:指免疫程序通过恢复内存中 CS:0100H 开始的 0EH 个字节,并使控制转向正常执行流程。

以下是免疫的主要过程及说明:

1. 在代码段的 0100H 处,使控制转向免疫代码段

4806:0100 E9FA08 JMP 09FD ;使第一条指令就转向免疫程序

2. 打开并读取现文件的头 20H 个字节及现文件长度

```
4806:0732 B000 MOV AL,00;打开文件读
4806:0734 B43D MOV AH,3D
4806:0736 CD21 INT 21
4806:0738 7303 JNB 073D
4806:073A E93C01 JMP 0879 ;失败时转恢复正常执行流程
4806:073D 8BD8 MOV BX,AX
4806:073F 0E PUSH CS
4806:0740 1F POP DS
4806:0741 B92000 MOV CX,0020;读 20H 个字节
4806:0744 BAA703 MOV CX,03A7
4806:0747 03D5 ADD DX,BP
4806:0749 B43F MOV AH,3F
4806:074B CD21 INT 21
4806:074D B80242 MOV AX,4202
4806:0750 33C9 XOR CX,CX
```

```
4806:0752 33D2 XOR DX,DX
4806:0754 CD21 INT 21 ;至文件尾部,并返回现文件长度
4806:0756 50 PUSH AX
4806:0757 B43E MOV AH,3E ;关闭文件
4806:0759 CD21 INT 21
4806:075B 58 POP AX
```

3. 比较文件长度:将执行文件长度同正常文件长度相比较

```
4806:075C 2E CS;
4806:075D 3B96D103 CMP DX,[BP+03D1] ;比较文件长度的高字
4806:0761 7711 JA 0774 ;文件增大转
4806:0763 7403 JZ 0768
4806:0765 E91101 JMP 0879 ;文件长度变小转正常流程
4806:0768 2E CS;
4806:0769 3B86CF03 CMP AX,[BP+03CF];比较文件长度的低字
4806:076D 740B JZ 077A ;文件长度没变化
4806:076E 7703 JA 0774 ;文件增大转
4806:0771 90501 JMP 0879;文件长度变小转恢复正常执行流程
```

若现执行文件长度减小,则系统不作任何处理,控制转向恢复正常执行流程,以下是恢复正常执行流程的程序代码:

```
4806:0879 8CC8 MOV AX,CS
4806:087B 8ED8 MOV DS,AX
4806:087D 8EC0 MOV ES,AX
4806:087F BED503 MOV SI,03D5
4806:0882 03F5 ADD SI,BP
4806:0884 BF0001 MOV DI,0100
4806:0887 B90E00 MOV CX,000E
4806:088A F3 REPZ
4806:088B A4 MOVSB ;在内存中恢复原文件的头 0EH 个字节代码
4806:088C 5D POP BP
4806:088D 5F POP DI
4806:088E 5E POP SI
4806:088F 5A POP DX
4806:0890 59 POP CX
4806:0891 58 POP AX
4806:0892 BB0001 MOV BX,0100
4806:0895 53 PUSH BX
4806:0896 33DB XOR BX,BX
4806:0898 C3 RET ;使程序控制转向 CS:0100H,即转正常执行流程
```

4. 将现文件的头 20H 个字节代码同正常文件的头 20H 个字节代码进行比较,若相同时,再考虑以下二种情况:若现文件长度没有变化,控制转向恢复正常执行流程;反之,若现文件长度增大了,系统提示用户,并在用户选择重建后,将现文件增大部分的代码从后面全部截去,以保证现文件长度恢复至正常文件长度状态。

以下是代码比较程序段:

```
4806:0774 2E CS;
4806:0775 C686A20301 MOV BYTE PTR [BP+03A2],01
4806:077A 2E CS;
4806:077B 8B8ED301 MOV CX,[BP+03D3]
4806:077F D1E9 SHR CX,1
4806:0781 2E CS;
4806:0782 8C86A503 MOV [BP+03A5],ES
4806:0786 1E PUSH DS
4806:0787 07 POP ES
4806:0788 BEA703 MOV SI,03A7
4806:078B 03F5 ADD SI,BP ;现文件的头 20H 个字节
4806:078D BF9F01 MOV DI,019F
4806:0790 03FD ADD DI,BP ;正常文件的头 20H 个字节
4806:0792 FE REPZ
```

```
4806:0793 A7      CMPSW
4806:0794 7523     JNZ      07B9 ;不相同时转
4806:0796 E99D00  JMP      0836 ;相同时转
```

免疫程序执行以下程序段可将文件增大部分的代码全部截去:

```
4806:0841 2E      CS;
4806:0842 8E9EA503 MOV     DX,[BP+03A5]
4806:0846 2E      CS;
4806:0847 8B96A303 MOV     DX,[BP+03A3]
4806:084B B92000 MOV     CX,0020 ;置文件属性
4806:084E B80143 MOV     AX,4301
4806:0851 CD21 INT     21
4806:0853 2E      CS;
4806:0854 8B96A303 MOV     DX,[BP+03A3]
4806:0858 B002 MOV     AL,02 ;打开文件进行读写
4806:085A B43D MOV     AH,3D
4806:085C CD21 INT     21
4806:085E 8BD8 MOV     BX,AX
4806:0860 B80042 MOV     AX,4200 ;恢复正常文件长度
4806:0863 2E      CS;
4806:0864 8B96CF03 MOV     DX,[BP+03CF]
4806:0868 2E      CS;
4806:0869 8D8ED103 MOV     CX,[BP+03D1]
4806:086D CD21 INT     21
4806:086F B440 MOV     AH,40
4806:0871 33C9 XOR     CX,CX
4806:0873 CD21 INT     21
4806:0875 B43E MOV     AH,3E ;关闭文件
4806:0877 CD21 INT     21
```

比较内容若不相同,说明现文件的头部字节遭到改变,很可能是因为病毒修改的缘故。因此免疫程序显示如下信息:

Central Point Anti-Virus(c)1991 CPS

Self Integrity Check warning-File was changed!

Choose an option:

[R]Self Reconstruction.

[C]Continue Execution.

[E]Exit to DOS.

Press R,C,or E:

用户可以按 R、C、E 键来执行相应的功能。

#### 1. 文件的自重建(Self Reconstruction)

选择该项时,免疫程序将正常文件内容的头 20H 个字节恢复至现文件的首部,以改变病毒感染后的程序转向,使其恢复至文件加免疫后的状态。

```
4806:07D8 1E      PUSH    DS
4806:07D9 2E      CS;
4806:07DA 8E9EA503 MOV     DS,[BP+03A5]
4806:07DE 2E      CS;
4806:07DF 8B96A303 MOV     DX,[BP+03A3]
4806:07E3 B92000 MOV     CX,0020
4806:07E6 B80143 MOV     AX,4301 ;置文件属性
4806:07E9 CD21 INT     21
4806:07EB 2E      CS;
4806:07EC 8E9EA503 MOV     DS,[BP+03A5]
4806:07F0 2E      CS;
4806:07F1 8B96A303 MOV     DX,[BP+03A3]
4806:07F5 B002 MOV     AL,02 ;打开文件进行读写
4806:07F7 B43D MOV     AH,3D
4806:07F9 CD21 INT     21
4806:07FB 8BD8 MOV     BX,AX
4806:07FD 1F      POP     DS
4806:07FE B92500 MOV     CX,0025
4806:0801 BFA203 MOV     DI,03A2
4806:0804 03FD ADD     DI,BP
4806:0806 C60500 MOV     BYTE PTR[DI],00
4806:0809 47      INC     DI
4806:080A E2FA LOOP    0806
4806:080C B440 MOV     AH,40 ;写文件头 CX 个字节
4806:080E 2E      CS;
```

```
4806:080F 8B8ED303 MOV     CX,[BP+03D3];CX 为恢复的字节数
4806:0813 BA9F01 MOV     DX,019F
4806:0816 03D5 ADD     DX,BP
4806:0818 CD21 INT     21
4806:081A 06      PUSH    ES
4806:081B 1F      POP     DS
4806:081C 2E      CS;
4806:081D 8B96D303 MOV     DX,[BP+03D3]
4806:0821 2E      CS;
4806:0822 8B8ECF03 MOV     CX,[BP+03CF]
4806:0826 2BCA SUB     CX,DX
4806:0828 B440 MOV     AH,40 ;恢复正常文件长度
4806:082A CD21 INT     21
4806:082C B440 MOV     AH,40
4806:082E 33C9 XOR     CX,CX
4806:0830 CD21 INT     21
4806:0832 B43E MOV     AH,3E ;关闭文件
4806:0834 CD21 INT     21
```

#### 2. 文件的继续执行(Continue Execution)

即用户可以忽略现文件头部字节所作过的修改,即使可能是病毒的缘故,也让其接着往下执行。

#### 3. 返回 DOS 操作系统(Exit to DOS)

即让程序中止运行,系统直接返回到 DOS 提示符下。

## 二、功能评价及其改进

CPAV 的上述免疫功能基本可以达到如下目的:

1. 及时发现文件被修改过,退出 DOS 状态,可将改变后的文件保存下来,作为下一步检测及分析病毒的标本。
2. 不影响程序的继续执行。
3. 即使文件确实染上了病毒,有了免疫功能就可自动将病毒消灭,而对文件本身却没有一点影响。

我们知道,文件型病毒一般都要增大被感染文件的长度,并修改被感染文件的头部字节,使其控制首先转向病毒代码。因此,CPAV 免疫程序以文件的长度及其头部的字节值为标准进行判断,是基本可以实现可执行文件对病毒的免疫目的的。

笔者通过对 CPAV 的免疫程序进行分析,发现该程序还存在如下缺陷:

1. 若文件长度小了,文件仍然照常执行,这很有可能让病毒程序继续执行。因此,我们最好使程序停止执行,返回 DOS,并进一步分析其原因。另外,我们还可以进一步检查文件的头字节值的真实性。

2. 文件自重建时,由于只考虑文件头部字节 06H 后的 4 个字节没有被病毒修改或覆盖的情况,因此,若病毒修改或覆盖的字节超过 06H 处时,免疫功能中的自重建将失去作用,大大降低了该程序的免疫作用。解决的办法是改变校验位置,如将校验位置 06H 改为 20H,校验的字节数可不改变。

具体修改步骤如下:

在 DOS 状态下,将 CPAV 软件拷贝到 C: 盘的 CPAV 子目录中,并进入该子目录,进入该子目录后,按如下顺序键入命令:

#### 1. 将 CPAV.EXE 换名

C:\CPAV>REN CPAV.EXE CPAV.DAT

#### 2. 用 DEBUG 工具修改相应参数

```
C:\CPAV>DEBUG CPAV.DAT
-R DS ;改变数据段地址
XXXX,YYYY ;其中 YYYY=XXXX+1F00H
-E 8BB
06:20 ;将校验位置 06H 改为 20H
-R DS 恢复数据段地址,
YYYY,XXXX
-W ;修改后存盘
-Q ;退出 DEBUG,返回 DOS
```

#### 3. 恢复 CPAV.EXE 文件名

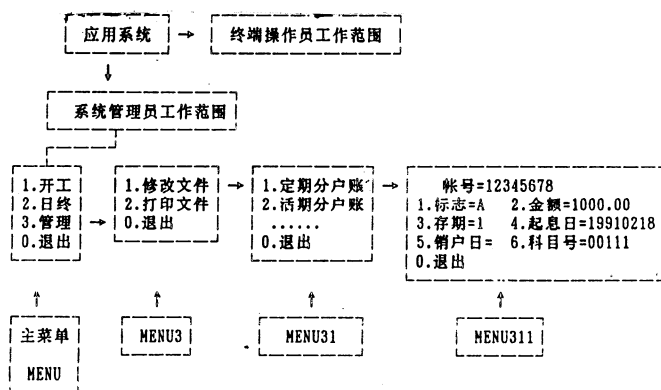
# 银行系统中全信息记录操作痕迹

□ 成 刚 杨明秋

前, 计算在银行得到了广泛的应用, 如何保证银行计算机的数据安全, 防止计算机犯罪, 是一个十分重要的问题。为此, 笔者结合多年的工作实践经验, 提出一个银行计算机应用系统中全信息记录用户作痕迹的现实方法。

全信息记录应用系统中操作痕迹的方法举例如下:

下图是一应用系统中系统管理员在主控台上的一系列操作(应用系统由 MS COBOL 编写)。



(1) 给应用系统中的每一个菜单画面起好名字, 如 MENU, MENU3 以及, MENU31, MENU311 等等。

(2) 设计好一个全信息跟踪文件, 其描述如下:

```

*****PCN-FILE *****
SELECT PCN-FILE ASSIGN TO RANDOM, "/usr/mdk/pcn"
ORGANIZATION IS LINE SEQUENTIAL
ACCESS MODE IS SEQUENTIAL
FILE STATUS IS FILE-S
***** PCN-REC *****
FD PCN-FILE
  DATA RECORD IS PCN-REC.
01 PCN-REC.
02 menu-name PIC x(10).      菜单名
02 acct-num PIC 9(02) comp.  菜单选中项
02 oper-num PIC 9(02) comp.  操作员号
02 op-date pic 9(08) comp.   进入菜单时的日期/修改数据时的日期
02 op-time pic 9(06) comp.   进入菜单时的时间/修改数据时的时间
02 modi-yn pic 9 comp.       标志 0--查询, 1--修改
02 modi-n1 pic x(8).         查询/修改的文件名字(如定期分户帐
  -----FJA-FILE)
02 modi-n2 pic x(8).         文件记录项名(如定期分户帐中的金额
  项-----FJA030).
```

```

02 modi-n3 pic 9(8) comp.  帐号
02 modi-n4 pic 9(10) comp. 修改前的数据内容
02 modi-n5 pic 9(10) comp. 修改后的数据内容
```

(3) 一般来讲, 系统管理员进入应用系统时, 首先要进入主菜单 MENU。一旦系统管理员(操作员号 01)从 MENU 中选中 3 进入服务菜单 MENU3, 即把“MENU”, 选中项 3, 操作员号 01, 选中 3 时的机器日期, 时间内容传给 PCN 的对应项, 由于没有进行修改帐户数据的操作, 故 modi-yn=0, 因此, PCN 的其他项为空。然后把结果写入 PCN 文件记录下来, 其程序实现见下面的 main. cbl:

```

***** main. cbl *****
IDENTIFICATION DIVISION.
PROGRAM-ID. MAIN.
AUTHOR. 成 刚.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. MS-COBOL.
OBJECT-COMPUTER. MS-COBOL.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  copy "/mnt/pcn-file".
.....
DATA DIVISION.
FILE SECTION.
  copy "/mnt/pcn-rec".
.....
WORKING-STORAGE SECTION.
.....
PROCEDURE DIVISION
BEGINING.
.....
  OPEN I-O PCN-FILE.
.....
w-pcn.
  move "MENU" to menu-name.
  move 3      to acct-num.
  move 01     to oper.
  move date1  to op-date.
  move time1  to op-time.
  move 0      to modi-yn.
  move space  to modi-n1 modi-n2.
  move 0      to modi-n3 modi-n4 modi-n5.
  write pcn-rec.
w-pcn-end.
.....
CLOSE PCN-FILE.
```

C:\CPAV>REN CPAV.DAT CPAV.EXE

4. 运行修改后的 CPAV.EXE

C:\CPAV>CPAV

通过使用修改过后的 CPAV 对可执行文件加免疫功能, 即可将校验位置改在 20H 处。

笔者参照 CPAV 免疫程序的主要思想用 C 语言设计了一个

实用程序, 主要实现以下两个功能:

(1) 文件的排它性。即防止其他用户使用自己的文件, 实现用户对可执行文件的独享。

(2) 文件的免疫功能。其功能基本同 CPAV 免疫程序。

通过使用该实用程序对执行文件进行处理后, 即可实现上述两个功能。但其增加的代码要比 CPAV 的 998 字节少得多。



.....

(4)当系统管理员从 MENU3 中选中 1,进入服务菜单 MENU31 时,PCN 文件又可将操作过程记录下来,其程序实现参见 main. cbl 中的 w-pcn----->w-pcn-end 段。

(5)当系统管理员从 MENU31 中选中 1(定期分户帐),并打开某一帐户的定期帐户文件,进入菜单 MENU311,PCN 文件又可将操作过程记录下来,其程序实现如 main. cbl 中的 w-pcn----->w-pcn-end 段。

(6)如果系统管理员在 MENU31 中选 2,将账号为 12345678 的帐户金额由 1000.00 改为 2000.00,则 PCN 会记下“MENU311,菜单选中项 1,操作员号 01,修改账户数据时的机器日期,时间,由于进行了修改账户数据的操作,故 modi-yn=1,查询/修改的文件名字---FJA,文件记录项名---定期分户帐中的金额项---FJA030,帐号 12345678,修改前的 FJA030 数据内容为 1000.00,修改后的 FJA030 数据内容为 2000.00 等全信息操作痕迹,其程序实现见下:

```
.....
77 m-before pic 9(10).
77 m-after pic 9(10)
.....
PROCEDURE DIVISION.
BEGINING.
.....
accept new-fja030 at 1020.
*** 接收新的 FJA030.
move fja030 to m-before.
*** 把修改前的 FJA030 传给 m-before. (m-before=1000.00)
move new-fja030 to fja030 m-after.
```

```
*** 把修改后的 FJA030 传给 m-after. (m-after=2000.00)
rewrite fja-rec.
*** 更新 FJA-file.
.....
OPEN I-O PCN-FILE.
.....
w-pcn.
```

```
move "MENU311" to menu-mame.
move 1 to acct-num.
move 01 to oper.
move datel to op-date.
move timel to op-time.
move 1 to modi-yn.
move "FJA" to modi-n1.
move "FJA030" to modi-n2.
move FJA020 to modi-n3.
move m-before to modi-n4.
move m-after to modi-n5.
write pcn-rec.
w-pcn-end.
.....
CLOSE PCN-FILE.
.....
```

从上面的分析可以看出,一旦系统管理员启动应用系统,全信息跟踪就开始发挥作用,新记录的系统管理员在系统中的全部操作痕迹存放在/usr/mdk/pcn(最好是建成隐含加密的文件名)中。可以在需要的时候将/usr/mdk/pcn的内容打印出来,查询、分析系统管理员的工作情况,为恢复机器正常运行,恢复数据提供了很大方便,同时也为保护银行计算机的数据安全提供了可靠保证。

## ○软件维护

# 修改 FORMAT 使得无系统启动时的提示更合理

□常 久

计算机用户都会遇到这样的情况:当我们开机启动系统时,系统提示:“Non-system disk or disk error, replace and strike any key when ready...”中断信息,由于习惯上我们总是感到开机启动是应该成功的,于是此时会首先想到:

- (1)计算机病毒破坏了系统?
- (2)硬盘出了故障?
- (3)A 驱故障?

面对这种情况,有的人多次启动计算机,甚至格式化硬盘尤其是初级用户,而当我们认真检查后,才发现是由于上次操作计算机时将数据盘或非系统盘遗留在 A 驱动器中所至,至此一场虚惊告终。实际上系统提示的上述信息源于 A 驱中软盘的引导区,用 DEBUG 的 L 命令装入软盘引导区可以在引导区的末尾发现这些信息,而这些信息是由系统程序 FORMAT.COM 建立的,因此我们可以修改 FORMAT.COM 命令使这一提示信息更合理,具体修改方法如下:

C)DEBUG FORMAT.COM ;用 DEBUG 装入 FORMAT.COM 文件

```
-E CS:0F0C "There is a disk in drive A, remove it, press any key when ready....."
```

由于 DOS 的版本不同,上述地址可能不同,我们可以用 DEBUG 的 S 命令搜索这一地址:

```
-S DS:100 XXXX "Non-system disk or disk error , replace and strike any key when ready..."
```

其中 XXXX 为 FORMAT.COM 文件的长度,可通过 DEBUG 的 R 命令查得 CX 的值即是。

至此写盘并退出:

```
-W 100
```

```
-Q
```

这样通过被修改的 FORMAT 命令格式化的磁盘在遇到上述情况时将提示:

```
There is a disk in drive A:, remove it, press any key when ready.....
```

用户可以根据提示判定出原因。

# 一种新的程序跟踪方法

□ 郑 祥

**现**在许多通用的加密软件(如 PROTECT 等)具有良好的反拷贝与反跟踪功能,使得传统的软件跟踪方法完全失效。读者可能觉得这有些言过其实,但在阅读过下面的摘自 PROTECT.EXE 的一段程序之后就不会有这种感觉了:

```
C>SYMDEB A;PROTECT.EXE
Microsoft(R)Symbolic Debug Utility Version 4.00
Copyright(C)Microsoft Corp 1984,1985. ALL rights reserved.
Processor is[80286]
-u 0 2;这两条指令将 DS,SS 设置为同样的值
1CA1:0000 16      PUSH    SS
1CA1:0001 1F      POP     DS
;下面执行这两条指令
-T
AX=0000 BX=0000 CX=46B0 DX=0000 SP=018C BP=0000 SI=
0000 DI=0000 DS=18D2 ES=18D2 SS=1C72 CS=1CA1 IP=0001
NV UP EI PL NZ NA PO NC
1CA1:0001 1F      POP     DS
-T
AX=0000 BX=0000 CX=46B0 DX=0000 SP=018E BP=0000 SI=
0000 DI=0000 DS=1C72 ES=18D2 SS=1C72 CS=1CA1 IP=0000
NV UP EI PL NZ NA PO NC
1CA1:0002 16      MOV     [020A],AX
;计算一下 CS,DS,SS 的差
-H CS DS
3913 002F
;从结果看出,CS 比 DS,SS 大 2FH,即 DS:[XXXX+2F0H]与;CS:[XXXX]为
同一单元。这一点很重要,因为程序中利用了这一特点。
```

下面为关键的程序段,该程序段用于恢复即将执行的程序段,即具有程序自生成;的特点。

```
-U 81 11E
1CA1:0081 FA      CLI
1CA1:0082 8B2EE02 MOV    SP,[02EE]
1CA1:0086 FD      STD     ;方向向前进行
1CA1:0087 33CC    XOR     CX,SP ;用 SP 值参加运算
1CA1:0089 33DC    XOR     BX,SP
1CA1:008B 33EC    XOR     BP,SP
1CA1:008D BE0AF4  MOV    SI,F40A
1CA1:0090 03F1    ADD     SI,CX
1CA1:0092 8132BA10 XOR    Word ptr[BP+SI],10BA
;DS:[398]=CS:[a8],及修改 A8 处指令,将 XOR SI,SI 改为 MOV SI,SP
1CA1:0096 8D97E2FE LEA     DX,[BX+FEE2]
1CA1:009A 018FF002 ADD     [BX+02F0],CX
1CA1:009E 03CA    ADD     CX,DX
1CA1:00A0 7904    JNS     00A6
1CA1:00A2 2BD1    SUB     DX,CX
1CA1:00A4 33C9    XOR     CX,CX
1CA1:00A6 87CA    XCHG    CX,DX
1CA1:00A8 33F6    XOR     SI,SI
```

该指令不断被修改(为 MOV SI,SP)、恢复

```
1CA1:00AA 8BFE      MOV    DI,SI
1CA1:00AC 2E8185BAFDA0F ADD    Word ptr CS:[DI+FD-
BA],0FAA
```

将 CS:[A8]处值加上 0FAA,实际用于将 A8 处指令从 MOV SI SP 再次恢复为

```
XOR SI,SI
NEG CX
NOT SI
LEA SI,[SI+06FD]
```

得到的 SI 为 40BH,即从 DS:40B 开始(即 CS:11B)向前进行

```
1CA1:00BB 83C3FE    ADD    BX,-02
1CA1:00BE 7903      JNS     00C3
1CA1:00C0 83EB85    SUB    BX,-7B
1CA1:00C3 AC        LODSB
1CA1:00C4 02E0      ADD    AH,AL
```

取指令码参与运算

```
1CA1:00C6 7B03      JPO     00CB
1CA1:00C8 42        INC     DX
1CA1:00C9 EB04      JMP     00CF
1CA1:00CB 304600    XOR     [BP+00],AL
```

将计算结果作为解密码

```
1CA1:00CE 45        INC     BP
1CA1:00CF E2F2      LOOP    00C3
1CA1:00D1 33CA      XOR     CX,DX
1CA1:00D3 75B8      JNZ     008D
1CA1:00D5 81CA2FF2  OR      DX,F22F
1CA1:00D9 81B36BF14EA0 XOR    Word ptr [BP+DI+
F16B],A04E
1CA1:00DF C4369002  LES     SI,[0290];INT 1C 的
中断向量地址
```

```
1CA1:00E3 FC        CLD
1CA1:00E4 26AD      LODSW ES;
1CA1:00E6 26C744FE6703 MOV    Word ptr ES:[SI-
02],0367
```

设置 INT 1C,但是新的 INT 1C 的处理程序等待恢复

```
1CA1:00EC A30C02    MOV     [020C],AX
1CA1:00EF 8CC8      MOV     AX,CS
1CA1:00F1 268704    XCHG    AX,ES:[SI]
1CA1:00F4 A30E02    MOV     [020E],AX
1CA1:00F7 C745BB8E01 MOV    Word ptr [DI-45],018E
1CA1:00FC C4368C02  LES     SI,[028C]
1CA1:0100 26AD      LODSW ES;
1CA1:0102 26C744FE4704 MOV    Word ptr ES:[SI-02],
0447
1CA1:0108 A31802    MOV     [0218],AX
1CA1:010B 8CC8      MOV     AX,CS
1CA1:010D 268704    XCHG    AX,ES:[SI]
1CA1:0110 A31A02    MOV     [021A],AX
1CA1:0113 C74600FB04 MOV    Word Ptr [BP+00],04FB
1CA1:0118 81360E04ED7E XOR    Word ptr[040E],7EED
1CA1:011E C3        RET
```

在解密程序段执行之前,设置寄存器 BP,BX,CX 的初始值:

```
-U 19 1D
1CA1:0019 8B2E6F03    MOV    BP,[036F]
1CA1:001D BA3202      MOV    DX,0232
-U 60 64
1CA1:0060 8B0E6B03    MOV    CX,[036B]
1CA1:0064 8B1E6D03    MOV    BX,[036D]
```

看一下[36B]、[36D]、[36F]的值  
 -DW 36B L3  
 1C72:036B 0997 0297 06FB  
 ;BP、BX、CX 的初始值为 6FBH、997H、297H  
 ;82 处的指令(MOV SP,[02EE]初始化 SP  
 -DW 2EE L1  
 1C72:02EE 02EE  
 ;将 SP 初始化为 2EEH

分析该解密程序段,会发现该程序段具有以下特点:

(1)程序自生成。

(2)程序段的代码段(11BH—79H)自身为解密码,并且采用循环叠加的计算方法,该代码段间的任何一个代码值被修改均会导致连锁反应,使得其后的解密码均不正确。这样就使得我们无法在该程序段间设置断点,因为断点的设置是通过将断点处的值修改为 0CCH(0CCH 为断点中断 INT 3 的机器码)或其他值(有的调试程序不用 INT 3 而改用其他中断作断点中断)实现的。也就是说,该程序段将自身完全保护起来。

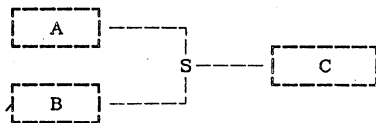
(3)程序段对自身的指令会进行修改与恢复。

(4)本程序的单步中断尚可使用,但是程序的计算量特别大,完全用单步方法根本不可能实现。

可以看出,使用传统的跟踪方法根本无法对该程序段进行跟踪。因为一旦进行跟踪就会导致生成的代码发生错误,也就使得程序无法继续执行。为此我们必须另寻出路。

在讲述新的跟踪方法之前,我们先提出一个假设,因为该假设是新方法的理论基础。该假设虽然未经严格证明,但至少对绝大多数的(甚至是所有的)情况是成立的。

假设数据块 A 与数据块 B 经过某种运算 S 成为数据块 C,那么运算 S 可以通过程序实现。图示为:



有了这一假设,我们分析前面的程序段:将被恢复的程序段看作数据块 B,而程序段自身作为解密码参加了运算,我们将其代码块看作数据块 A,恢复后的程序段看作数据块 C,因此我们可以编写程序 S 来实现这一恢复过程。程序 S 的执行过程与解密程序段既有区别又有联系。相同的是两者完成相同的功能,不同的则是程序段 S 自身不参与运算,而将解密程序段的代码作为处理数据参与运算,并且程序段 S 不再具有任何反跟踪功能。这样通过执行程序段 S 就可得到完整的恢复后的程序段。

当然要编写程序段 S,必须首先透彻地了解解密程序段的功能,因为这是编写程序段 S 的根据。

下面以编写恢复 PROTECT 程序段为例,说明编写程序段 S 的过程。

(1)用 SYMDEB 或 DEBUG 装载 PROTECT.EXE:

C:)>SYMDEB A:PROTECT.EXE

(2)按照程序原来的结构对程序段进行复制,将复制的程序段作为数据参与运算:

-M SS:0 L2F0 7000:0;将程序的数据段及堆栈段复制;到 7000H:  
 0(数据段及堆栈段共 2F0H 个字节)

-M CS:0 LCX 702F:0;因为 CS 比 SS 大 2FH,因此将代码段复制到 702FH:0,CX 为代码段的长度

(3)将 SS 设置为 7000H,使得 DS、SS 均指向参与运算的数据段:

-R SS 7000

(4)偏移量 00ACH 处的指令为:ADD Word Ptr CS:[DI+FDDB],0FAA,因为现在的 CS 相当于 702FH,因此需要修改此指令,通观整个程序段,会发现寄存器 ES 一致未使用,因此可将该指令修改为 ADD Word Ptr ES:[DI+FDDB],0FAA,并将 ES 设置为 0702FH

-E CS:AC 26;将用 CS 作段超越(代码值为 2EH)修改为用 ES 作段超越  
 ;(代码值为 26H)

-R ES 702F

(5)0092H 处的指令 XOR Word Ptr[BP+SI],10BA 用于将 0A8H 处指令由 XOR SI,SI 修改为 MOV SI,SP,而 00ACH 处的指令由不断将 00A8H 处的指令恢复为 XOR SI,SI,现在我们已经将 00ACH 处的指令修改为 ADD WORD PTR ES:[DI+DBA],0FAAH,因此该指令就不能在用于恢复 00A8H 处的指令。但是分析整个程序段发现 0092H 处将 00A8H 处指令修改后,只是将 SP 赋予 SI 之后立即由 AC 处指令恢复成原样,因此我们可以简单地将 00A8H 处指令修改为 MOV SI,SP,而将 0092H 处的指令修改为一系列的 NOP。

(6)现在即可执行 G D5,这样即可在 702FH 段中得到恢复的程序段。

-G D5

-R CS 702F

转入 702FH 段去阅读恢复后的程序段。

因为该程序的结构较特殊,因此我们没有必要重新编写新的程序,而可直接利用其原来的程序段,对其必要之处作简单修改即能满足要求。当然对另外的一些结构更复杂的程序则需要重新编写程序。假如在该程序段中,寄存器 ES 被另作它用,则直接修改 00ACH 处的指令就不能达到目的了,而必须开辟新的单元,将该单元的值设置为 702FH,在执行到 00ACH 处的指令时,将该单元的值赋予某寄存器(如 ES,当然必须首先保存原来的 ES,在该指令执行之后再恢复 ES),通过该寄存器来达到目的。

这种跟踪方法可以解决传统跟踪方法无法跟踪的程序,但是因为该方法在很多情况下需要另外编写程序 S,因此只有当用传统方法无法跟踪时才采用该方法。其实该方法对绝大多数程序均有效,这与前面所作的假设的正确程度一致。

FBASE——DPS 下的最佳多媒体数据库平台

新未来电子技术公司

Tel: 2573355—271, 272

地址:清华东门清华园宾馆一层

# Gene 病毒的检测与清除

□ 宁 波

**G**ene 病毒攻击 EXE 文件。因其病毒代码区中有如下字符: 'Gene'。病毒代码长 556H, 附在 EXE 文件的尾部, 并使之增大 556H(1360)+X 字节, 其中 X=16-(原文件长度 MOD 16)。其感染标志是文件尾部的两字节为 0AAAAH。

当运行带毒程序时, 病毒首先判断自身是否已驻留内存, 有则转去执行原程序, 否则即修改 MCB 的 (03) 单元及 PSP 段的 (02) 单元内容驻留内存, 而用 PC-T00L、CHKDSK 检查内存容量, 不会发现任何变化, 用 CPAV 的 VSAFE 无法检测到它的驻留。病毒进驻内存后修改 DOS 功能调用 INT21H 和时钟中断 INT1CH 的中断入口, 为己所用, 其中前者用于传染, 后者用于表现。病毒驻留并窃取中断后即转去执行原程序。

病毒修改了由 INT 21H 调用的 3 个子功能: 1AH, 11H, 12H 当系统通过调用 INT 21H 的 1A 号子功能设置 DTA 时, 病毒保存 DTA 的地址后执行原中断功能返回; 当系统每次调用 INT21H 中断的 11H 及 12H 子功能搜索匹配文件时, 病毒将获得控制权, 先执行原中断子功能后, 判断所搜索的匹配文件是否为 EXE 文件, 是则判断是否已染毒, 若为 EXE 文件且未染毒则进行传染。DOS 中最常用的 DIR 命令就要调用 INT21H 的 11H 及 12H 子功能, 因此每次使用 DIR 时所列文件中的所有 EXE 文件将被感染。

Gene 病毒修改了时钟中断 INT 1CH 用以表现。在 BIOS 的数据区 40: 6CH—6FH 的两个字存放着系统自午夜以来的时钟数, 当 40: 6E 一字的内容为 0004 时, 病毒开始从 0 计时, 当内存病毒中的计时器为 444H 时, 病毒在屏幕的左上角显示 'Gene!' 共 6 个字符, (彩色文本时为红色, 黑白文本状态时为反相显示)。

病毒采用了一些手段来防止被发现分析和解毒: 在传染文件前保存文件的日期、时间, 在传染后恢复; 在传染文件时动态地修改 INT24H 中断, 屏蔽可能出现的错误信息; 为防止静态分析, 病毒采用了密文技术, 对自身的多个子程序及数据进行加密, 运行时先进行解密, 运行完再加密, 在感染文件时, 先对内存中加密过的

两段代码进行解密, 而后将 BIOS 数据区 40: 6C 一个字的内容保存在病毒 CS: 581 处, 然后以 CS: 581 处一个字节的内容为密钥加密 CS: 11C 到 CS: 4F6 的代码 (包括原文件的文件头的控制信息), 接着以 CS: 582 字节的内容为密钥加密 CS: 540 到 CS: 581 的内容, 两次所用的加密方法是不同的, 且第二次加密把加密原文件的文件头的控制信息的密钥也进行了加密, 病毒把经过加密的自身传染给文件后, 再把代码解密, 最后再用固定的密钥把内存中的两段代码加密; 为防止动态跟踪, 病毒采用了一些反跟踪技术, 如在程序中动态地重新设置、恢复 INT3H 中断, 使 SS: SP 指向 CS: 589, 这样当病毒未被跟踪时, 无多余的入栈、出栈操作, 可以正常运行, 而一旦被跟踪时有多余的堆栈操作, 将会破坏 CS: 581、582 的内容, 从而使解密后的代码面目全非。下面是病毒其中的两段加(解)密程序:

```
-ucs:560 580
11AA:0560 B9B603    MOV    CX,03B6
11AA:0563 D1D9      RCR    CX,1
11AA:0565 BE1C01    MOV    SI,011C
11AA:0568 2E        CS;
11AA:0569 A08105    MOV    AL,[0581]
11AA:056C FA        CLI
11AA:056D 83F900    CMP    CX,+00
11AA:0570 740D      JZ     057F
11AA:0572 2E        CS;
11AA:0573 8A24      MOV    AH,[SI]
11AA:0575 32E0      XOR    AH,AL
11AA:0577 2E        CS;
11AA:0578 8824      MOV    [SI],AH
11AA:057A 46        INC    SI
11AA:057B 46        INC    SI
11AA:057C 49        DEC    CX
11AA:057D EBEE      JMP    056D
11AA:057F FB        STI
11AA:0580 C3        RET
```

```
-ucs:589 5a6
11AA:0589 B94200    MOV    CX,0042
11AA:058C BE4005    MOV    SI,0540
11AA:058F 2E        CS;
11AA:0590 A08205    MOV    AL,[0582]
11AA:0593 FA        CLI
11AA:0594 83F900    CMP    CX,+00
11AA:0597 740C      JZ     05A5
11AA:0599 2E        CS;
11AA:059A 8A24      MOV    AH,[SI]
11AA:059C 32E0      XOR    AH,AL
11AA:059E 2E        CS;
11AA:059F 8824      MOV    [SI],AH
11AA:05A1 46        INC    SI
11AA:05A2 49        DEC    CX
11AA:05A3 EBEF      JMP    0594
11AA:05A5 FB        STI
11AA:05A6 C3        RET
```

病毒的发现比较容易, 查看一下文件长

度, 若增加了 1366 字节, 则有可能已感染了病毒。病毒的确诊可以有多种方法:

1. 用 PCTOOL 或 FIND 在文件中查找是否有 'Gene — 1991 \_ in DUT (Dalian China)' 的字符串;

2. DEBUG 调入文件, 查看其前面几条指令是否为:

```
x:118 CLI
x:119 CALL 5AE
.
x:5AE CALL 589
x:5AA CALL 540
x:5AD RET
```

以及文件末尾是否为:

```
-dcs:5b5 lif
11AA:05B0          47 65 6E-65 5F 31 39 39
                      31 5F 69          Gene-
                      1991-i
11AA:05C0 6E 20 44 55 54 20 28 44-61 6C 69 61
                      6E 20 43 68          n DUT (Dalian
                      China)
11AA:05D0 69 6E 61 29
-dcs:648 le
11AA:0640          47 04 65 04 6E 04 65 04
                      G.e.n.e.
11AA:0650 20 04 21 04 AA AA          .!...
```

3. 利用 SCAN 的功能, 用编辑软件编辑如下一个文件: NEWVIR. DAT

```
# New, EXE Virus found in file GDS . EXE
# 1993-12-11
# Here is its words:
# "Gene-1991-in DUT (Dalian China)"
# And "Gene!"
# Here is its Code:
# x:118 CLI
# x:119 CALL 05AE
# x:5AE CALL 0589
# x:5B1 CALL 0540...
# And The Length of Virus is 556H(1366bytes)
# Its Sign is two Hex number "AA AA" in the
end of File
# Chen-Kai-Hui keyin
"B9 42 00 BE 40 05 2E A0 82 05 FA 83 F9 00 74 0C
2E"NEW-Virus-1[V-Gene]
```

而后用以下命令即可测出感染 Gene 病毒的文件了:

SCAN d: /EXT NEWVIR

内存中病毒的确诊可参照病毒本身的方法, 取得 INT21 中断的段址, 查看段址的偏移 103 处是否 AAAA(16 进制), 是则内存中有病毒。

内存中病毒的清除主要是恢复 INT21H 的中断向量, 原 INT21H 的中断向量保存在病毒 CS: 536 处; 由于病毒没有保存 INT1CH 的中断向量, 可将其指向 BIOS 中的任意一条 IRET 指令即可。

## 栏目编辑的话

本栏目对现行市场上出售的或用户中流行的维护及相应漏洞的完善为主要内容,为用户更得心用地使用这些常用软件系统提供实用的经验、技巧和方法。本栏目包括:为完善现行软件的某些命令或功能而对该命令或功能的扩充,以及现行软件存在的不足及其解决方法等。其是由于国内的一些软件系统在上市前缺乏测试过程,所以用户在使用过程中会发现一些不足,同时,国外厂商的软件在国内应用中也存在不尽适合国内情况的现象,通过本栏目,用户可以从中得到对这些问题的一些就急解决方法,同时也为厂商提供一个完善自己产品的依据,作为本栏目的编辑,我们真诚地希望广大读者为我们提供更多、更好的稿件。

# AutoCAD 12版软件的问题及其解决方法

□林己平

**A**utoCAD R12已推出近两年了,它是目前应用最广泛的 CAD 支撑软件,目前已在我国设计界开始普及。和任何大型软件系统一样,Autodesk 公司在 1992年6月推出 R12的第一个版本以来,亦进行过2次升级,分别称 R12\_\_C1和 R12\_\_C2,对已发现的软件错误及时作出了更正。由于软件规模庞大,存在问题在所难免,而且国内 AutoCAD 应用主要建立在早期的版本上,用户了解 R12的软件问题及其解决方法是非常必要的。本文对 AutoCAD 的 R12 DOS 平台版本已知的软件问题及其解决方法进行了探讨,希望对 AutoCAD 的开发与应用能起到抛砖引玉的作用。

## 一、网络支持问题

### 1. R12的网络兼容问题

R12与一些网络无法兼容,在运行 DOS5.0 或 DOS4.0 时,LANtastic、3COM、DECPathworks,IBM LAN Manager 等用户无法在网上加载图形文件,或者从网络服务器上执行 acad.exe。这是由于网络系统不能正确支持 DOS 中断所致。

解决方法:改用对 AutoCAD 支持较好的 Novell 网络系统,或者用新的 R12\_\_C2版本。

### 2. AutoCAD 不支持网络打印

AutoCAD 对网络打印机重定向不加理会,而试图直接通过硬件接口出图。例如,若网络的配置将 LPT1定向,将输出送到连接在另一台计算机上的绘图机,AutoCAD 这时依然只将输出送到本机的第一个并行口。

解决方法:改用指定文件出图方式,文件名取为用户所定的接口名称(如

LPT1),或者,取名为 AUTOSPOOL,然后使用环境变量 ACADPLCMD 指定 AutoCAD 在网络上发送文件的方式。例如,在 LANtastic 网络上,使用 SET ACADPLCMD = NET PRINT/B%%s\\PLOTS ERV\\@PLOTTER。

### 3. ACAD.PWD 和 Novell3.11

在 Novell3.11网络上,AutoCAD 在服务器加载的情况下,如遇两个以上用户试图同时启动 AutoCAD,此时用户会得到“Waiting for ACAD.PWD File missing or corrupt.”的信息,如果用户让 AutoCAD 继续等待 PWD 文件打开,有时会得到 Unable to open PWD file”的信息,接着就退回 DOS。这时会留下未删去的 ACAD.PWK 锁定文件,影响其余用户。

解决方法:在出现“Waiting for ACAD.PWD...”提示时,按下<Ctrl>-C,使 AutoCAD 不再继续读 PWD 文件,过一会,再试图启动 AutoCAD。

## 二、外设支持问题

由于外设驱动与硬件关系较多,是软件错误出现最多的部分。

### 1. Laser Jet 驱动程序成为懒汉

使用 R12的 LaserJet (PCL)驱动程序出图,比 R11的 LaserJet 驱动程序多用数倍的时间。

解决方法:如果用户使用 1MB 以上内存的 LaserJet II,可改用 HPGL/2驱动程序,它的速度会快得多,还提供可变线宽功能。或者使用随外设提供的 R11(或新的 R12)驱动程序。

### 2. 不良的绘图机驱动程序

HP 笔式绘图机的 NPGL/2和 Cal-Comp 驱动程序在绘制宽复线(Pline)和填充区(solid)时,常发生填充不正确的出





发现其值从菜单名改为 TSSn,其中 n 为 ssget 返回的选择集编号。若 ssget 返回 nil,该值改为\_\_X。此错误主要影响需要检测当前菜单调用状态的应用程序,对人工操作不构成妨碍。

解决方法:当前未找到有效方法,只有在编程时加以注意。

## 四、图块图层管理的问题

### 1. 可能会导致死锁的图块插入

在使用相对距离很大的坐标(如大地测量坐标)系统时,用0,0作为插入点会导致系统死锁。产生死锁的条件为:

- (1)图块插入点为0,0,其中含有非连续线型图元;
- (2)启动过 Dview 命令;
- (3)拖拉模式 DRAGMODE 打开;
- (4)LTSCALE 不等于1。

下列命令列在 R12各 DOS 版本均可能导致死锁:

```
DRAGMODE Auto
ZOOM 1999995,499995 2000105,500100
LINE 2000000,500000 @100,0<回车>
CHPROP Last <回车> LType HID-
DEN2<回车>
BLOCK LOCK UP 0,0 Last<回车>
DVIEW<回车><回车>
LTSCALE 50
INSERT LOCKUP 0,0
```

解决方法:在程序中插入任何图块之前,先将 LTSCALE 设为1,或者将拖拉模式先关闭,插入后再行恢复,或在交互操作时使用 DDINSERT 对话框,而不使用 INSERT 命令,也可避免死锁。

### 2. 无法选择的可见属性

当你在图层0上定义块属性,然后将其他到其他图层,如果图层0被冻结,则该属性无法选择。尽管该属性与图块均可以被正常显示出来,这是由于定义在0层上的图块具有继承所插入的图层的性质,但当对其使用 DDate 或其他编辑工具时,AutoCAD 却报告“No object found”。

解决方法:选择图块中其他非属性部分。如果图块中没有非属性部分,则要将0层解冻,或者在前面的操用中将0层关闭,而不应冻结。

### 3. Wblock \* 失去 Xref 图层的设定

如果使用 Wblock<图名>来清理含有外部引用(xref)的图形,AutoCAD 将所有 xref 图层的颜色和成型重设为外部图形原来的状态,而不理会在后来所作的任何新设定,也不理会 VISRETAIN 变量设为1以保持设定不变的约定。

解决方法:命名用 Purge 命令而不要使用 Wblock \*,有时由于块的多层嵌套,也许需要多次使用 Open 与 Purge 才能清理所有不必要的内容。

### 4. Ddate 命令会使属性无法对齐

某些情况下,使用 Ddate 命令编辑属性会使属性位置发生无法预知的改变。这种问题往往发生在刚刚使用过 Style 命令定义过字型的垂直、反向、转置或位置属性之后。

解决方法:首次要编辑属性时,先清除文字域的内容,然后按下 OK 按钮,再次键入 Ddate 键入新属性值,使用 Ddate 两次,可使 AutoCAD 使用正确的对齐方式,或者先写一次不使用上述位置属性的文字,再进行属性编辑。

## 五、字体、线型的问题

### 1. PostScript 驱动程序花容失色

当使用 AutoCAD 的 PostScript 驱动程序 PLPPOST.EXP 向彩色 PostScript 设备出图时,图形中如有填充图元(宽 Pline、填充区 solid 等),AutoCAD 无法正确处理颜色的描述,它虽然把填充图元的颜色画出了,但它却接下去以所填充的颜色画出后面所碰到的其他颜色定为随图层的非填充图元。

解决方法:可改用 Psout 命令试试,或者用 Chprop 命令将所有颜色随图层的图元改为某一指明颜色编号,或者将要绘制的图形定义为一个图块,在原位插入再行出图,最后将它炸碎(Explode)还原。

### 2. 缩水的 Postscript 字体

每当使用 Postscript Type1 字体(.PFB 文件),AutoCAD 产生的字高会比所指定的字高大约小30%。例如指定字高为10,实际写出的高度仅为6.9左右。

解决方法:你只能预先定义高30%~40%的字体尺寸,不同的.PFB 字体这个系数有所变化,只能通过试验决定。

### 3. R11生成的图形会导致全部线型的改变

打开一个由 R11生成的图形,如果其中含有非连续线型的外部引用,并且变量 VISRETAIN 改为1,R12会将在外部引用中的全部线型均改为连续线。

解决方法:在进入 R12之前,先在 R11中将变量 VISRETAIN 设为0,存图。在 R12中载入该图型后,再重新设置外部引用图层的可见性,将 VISRETAIN 变量重新置为打开状态。

## 六、与编程和系统有关的问题

### 1. 退出 R12出错

一些计算机在退出 R12时,显示“Pharlap Error:10025”或“PharLap Error:10026”表明 DOS IRQ7处理出错,需要重新启动。

解决方法:升级到 R12\_\_C2版本,或者从 Autodesk 获得 IRQ7X 修改程序。

### 2. R11的 ADS 应用程序在 R12上的问题

一些在 R11编译的 ADS 程序在 R12上执行时会出现过滤表无效的错误。问题出自 ads \_\_ssget()函数,它让用户在 R12中指定“W”或“C”参数,而 R11的 ADS.LIB 却不承认“W”或“C”的参数表传递,而这却是 R12必不可少的,即使它为 NULL。

解决方法:对 R11的 ADS 源程序应在 R12下重新编译,如果是商品软件,须由开发商给予升级。

### 3. ADS 不执行 RSVP

ADS 并非总是能发送 RQEND 查询码,该代码用于通知 ADS 应用程序,AutoCAD 打算要结束当前图形,使应用程序可以执行清理的作业。

解决方法:目前没有解决方法。

### 4. (SSGET)函数的“F”与“WP”,“CP”选择项不宜滥用

R12为 SSGET 函数提供了丰富的选择手段,但新增加的几个选择如(SSGET“F”点表),(SSGET“CP”点表),(SSGET“WP”点表)都存在误差,点表所在的图元需要选择时,无法保证它们全部进入选择集中。

解决方法:在使用诸如“end”或“mid”等物体捕捉方式获得点表时应多加小心。设法使点表包含的范围加以扩大。或者使用用户升级到 R12\_\_C2版本,在新版本中已对本问题作了修正。

### 5. 栏选(Fence)功能失常

如果栏选线的端点不在屏幕显示区内,该线亦非水平垂直方向,则所选的物体无法预知。如使用(ssget“F”)函数,符合上述条件时,返回不正确的选择集。

解决方法:如需使用栏选线,程序中则事先应测量屏幕中选线的端点,交互时则应放大屏幕范围,保证上它们位于屏幕之中,或者升级到 R12\_\_C2版本。

### 6. 对话框 DCL 语言中“alignmen=right”形同虚设。

在 DCL 文件中,对齐属性值“right”对文字框不起作用。例如下列 DCL 代码把 Hello,world 向左对齐,尽管由于“alignment=right”的作用,它是理应向右对齐的。

```
hello:dialog{
```

```
label="Sample Dialogue Box";
:text{label="Hello,world";
aligment=right
okonly;
```

解决方法:将文字框砖置于行(row)内,使用空档(spacer)框砖迫使文字向右排列。

### 7. 栏选(Fence)与修剪(Trim)和延伸(Extend)的联用

R12的栏选功能在修剪和延伸命令的联用中发挥作用最大,你可以在 AutoCAD 为 trim 或 extend 操作要求图元时键入 F,接着经过多个图元画一条栏选线。这种“多重修剪/延伸”功能有一个局限,当栏选线切割一条复线(Pline)多于一次,AutoCAD 只对此条复线的一段起作用,如果你画一个马蹄形的 Pline 线,并使一条栏选线通过马蹄的两肢,这对 AutoCAD 仅修剪或延伸其中一肢。

## 七、其他有关问题

### 1. Ddmodify 干扰 Undo 的执行

调出 Ddmodify 对话框之后,选择 Cancel 按钮,然后键入 U,退回以前的操作。AutoCAD 提示 Start of group encountered Must enter UNDO END to go back further 而不执行应有的动作。

解决方法:键入 Undo end 返回,为彻底解决该问题,应修改程序 ddmodify.lsp,找到(defun reset())在它之前加入(ai \_undo \_POP)一行即可。

### 2. 旧的 acad.mnx 问题

如果使用 R12以前的菜单 ACAD.NMX,在当前搜索路径中若有 ACAD.MNL 文件,则下拉菜单中有些区域会异常暗显,由于这些区域暗显,使之无法被

选择,甚至有时还会发生死锁。

解决方法:将所用的旧菜单改名,不要再用 ACAD.MNX 的名称即可。

### 3. 图纸空间的 ZOOM 命令超界

在图纸空间里执行 ZOOM 命令时,R12在计算范围时会错误地将画在由 VPFrozen 命令冻结的图元计算在内。例如,在图层 LL 上 0,0 处绘一个小图,而在图层 UR 的 1000,1000 处绘另一个图。这时将 TILEMODE 改为 0,用 Mview 建立一个视窗,在该视窗中用 VPLayer Freeze 将视窗中 UR 图层冻结,这时用 Mspace 命令进行该视窗执行 ZOOM 命令,R12本应只将左下方 0,0 处的圆包含在视窗之中,而实际上显示范围却包括被冻结的右上方的圆在内,与 UR 图层被 VPThaw 命令解冻一样,只是 UR 层内容不可见而已。

解决方法:将有关的图元改用另外的图层存放,使用 Layer Freeze 命令冻结;或者按所要求的尺寸定义视区,使用 View Restore 命令恢复视窗内容,而不使用 ZOOM 命令。

### 4. Polyline 使用夹点的难题

如果使用夹点(grip)对 2D Pline 进行编辑,这时若使用捕捉(Snap)将该 Pline 某点有不同 Z 方向高度的其它图元连接,R12会将 Pline 该顶点 Z 方向高度改变为新值,由于该 Pline 与内部图形数据库不一致,在审核(Audit)时出现“Nonplanar 2d Pline vertex”错误报告,并将各顶点全部设为新高度值。

解决方法:在与其它图元捕捉时,使用 XY 点过滤符,在 R12\_C2 版中,改为 Z 高度保持不变,因此可选择升级到 R12\_C2 版本。

### 5. 半圆弧与直线相切时,OFFSET 命令异常

如果复线(Pline)的直线段半圆弧段(包角 180 度)连接成交锐角,则若对其执行偏移(offset)命令,在向内偏移时 AutoCAD 的 offset 命令执行结果不正确。

解决方法:未有完善解决方法,只有在与圆弧连接时,避免使圆弧正好形成 180 度包角。

### 6. 界限检测干扰数字化仪的配置

在界限检测打开时,AutoCAD 拒绝重新配置数字化仪菜单区,R12总是指出 \*\*Outside Limits 的超界信息。

解决方法:在重新配置数字化仪菜单区之前,先关闭界限检测,命令 LIMITS OFF 或改变量 LIMCHECK 为 0。R11 的图形中如有 VISRETAIN=1 则线型丢失。

### 7. 命令文件 Script 无法退出

AutoCAD 是允许命令文件中加注释行的,只要以分号起始即可,但在 Quit 语句后面的注释行却会对执行有影响,如下命令文件原意是使 AutoCAD 出图完毕即返回 DOS,但实际上只能回到 AutoCAD 命令提示符下,无法如愿完成。

```
;NOQUIT. SCR
```

```
PLOT Limits NOQUIT yes
```

```
;end of script
```

注:此命令文件要求事先配置好绘图机或打印机。

解决方法:将最后的注释行删除,但 QUIT Yes 后应保留回车符。

以上仅提供了国内外不少 AutoCAD 使用者反映的问题,以上提及的问题大多经笔者使用多个 AutoCAD R12 版本加以验证。

## ○新产品

### 与国际工业接轨的成功典范

## 华光 VI 型电子出版系统推出

我国独具特色的计算机产品——电子出版系统喜讯频传。潍坊华光电子集团公司今年又隆重推出华光 VI 型电子出版系统。

据了解,华光 VI 型的核心是采用目前国际上先进的多处理器并行设计制成 PIP(Postscript 光栅图文处理器),其曲线轮廓字形的生成速度及图形、图象处理速

度比采用协处理器的 PIP 提高 5~10 倍;系统同时支持中西文 Postscript Level 2 页面描述语言和华光页面描述语言,具有兼容华光原系列的功能。系统采用了国际先进的“随机挂网技术”来处理图象和彩色图象,使彩色桌面系统对图象处理的质量可与传统电分机处理结果媲美。该系统还利用新的软平台技术,设计了一系列具

有多功能的新软件,并采用 APPLE 环境技术开发彩色图片和照片处理系统软件。国际优秀的 PS 排版软件与华光新软件互相扩充和发展,形成华光成果的独特优势。

世界著名的 APPLE 和 AST 公司分别和华光集团达成双方“技术合作”及建立“销售服务中心”的协议。

# 给 DOS 增加一个分页命令

□ 成 刚 杨明秋

**在** IBM PC 及其兼容机上的 DOS 操作系统中,为了加标题分页输出 myfile.prg,我们往往要借助于 wordstar、xe 等编辑软件,但是利用众多编辑软件进行分页打印输出时,用户除了要输入文件名外,还要回答一些问题(如:“分页打印吗?”)这给用户操作、装订文件带来不便。为此,笔者用 Turbo Pascal 6.0 编制了一个加标题分页输出程序——PR.pas,编译链接后得到 PR.EXE,执行此程序,用户即可得到满意的输出结果。

利用 PR.EXE,可以方便地将一个文本文件加标题分页输出到显示器上或打印机上,也可以输出到另一个文本文件中保存起来备用。PR.EXE 还给用户提供一个包含源文件的文件名、进行分页的 DOS 系统日期、页数等内容的完整标题。同时,设定每页为 66 行,其中标题占 2 行,正文文件占 56 行,其余为空行,输出到打印机时,完全不用人工干预,即可获得良好的打印效果,而且正常分页走纸,这给用户查阅及装订文件带来了方便。

命令使用说明:

1. 不带参数运行,便将使用说明显示出来:

```
C>PR
显示:Usage:[d:][path]PR[d:][path]filename
Example:PR filename
PR filename>filename2
PR filename>LPT1
PR A:\path1\path2\filename>LPT1
```

2. 把当前目录的源程序 myfile.prg 分页输出到 namel 中去

```
C>PR myfile.prg>namel
```

3. 把 A 盘中\path1\path2\myfile.prg 分页输出到打印机中打印

```
C>PR A:\path1\path2\myfile.prg>LPT1
```

4. 在 A 盘上执行 C 盘中的 PR 命令

```
A>C:PR myfile.prg
```

5. 当要分页输出的文件名没有找到时,显示出错

```
C>PR ABC
```

```
ABC—文件不存在! [filename not found!]
```

6. 该程序在 IBMPC 及其兼容机、DOS 3.30、Turbo Pascal 6.0 下编译运行通过。

7. 利用 PR.EXE 加标题分页打印输出的 PR.PAS 源程序清单附后:

```
*****filename =Pr. pas page=1 *****
*****print date=1994-3-15 *****
PROGRAM pr; {分页打印或显示文件程序 PR.PAS}
USES dos;
```

```

LABEL
1000;
TYPE
string136=string[136];
var
infile,dcrtf:text;
str,filename:string136;
kj,page,pagel:integer;
year,month,day,week:word;
BEGIN
filename=paramstr(1); {读入要打印或显示的文件名}
if filename='' then
begin {如输入 PR,则显示命令用法}
writeln('Usage: [d:][path] PR [d:][path] filename');
writeln('Example:PR filename');
writeln('PR filename>filename2');
writeln('PR filename>LPT1');
writeln('PR A:\path1\path2\filename>LPT1');
writeln;
exit;
end;
assign(infile,filename);
{$I-}reset(infile);{$I+};
if ioresult<>0 then {检查文件存在吗?}
begin
write(filename;4,'-----文件不存在![filename not found! ]');
{显示 filename 文件不存在}
exit;
end;
{$u+};
BEGIN
getdate(year,month,day,week); {取出 dos system 日期}
pagel:=1
assign(infile,filename);
reset(infile); {打开文件 filename}
assign(dcrtf,'');
rewrite(dcrtf);
repeat
writeln('***** filename =',4,filename,'page =',20,
pagel,'*****';44);
writeln('***** print date=',year,'-',month,'-',day,'*****');
{打印或显示标题}
writeln;
writeln; {空二行}
for kj:=1 to 56 do
if not eof(infile) then
begin
readln(infile,str);
writeln(dcrtf,str); {打印或显示五十六行正文}
end
else goto 1000;
writeln;writeln;writeln;writeln;writeln;writeln; {空六行}
pagel:=pagel+1 {页数加一}
until eof(infile);
*****filename =pr. pas page=2 *****
*****print date=1994-3-15 *****
end;
1000:
close(infile);close(dcrtf); {打印或显示完毕,关文件}
exit;
end.
```

# 数据备份盘的随机恢复技术

□ 马文骞

目前,计算机在各行中日益普及,应用越来越广泛,各单位对计算机的依赖性也日益增强。为了保证机内大量数据、文件、各种数据库以及各种信息的安全,定期地对系统进行备份是很重要的。

Backup 是 DOS 提供的将硬盘数据备份到软盘的工具。它功能强,使用也方便,能充分利用目标盘几乎所有字节的空间,能备份具有任何属性的文件,加上参数“/S”还能备份指定目录及其各层子目录中的所有文件,加上参数“/F”能在备份前先对新盘进行格式化。在复制一个几张盘才能装下的大文件(如汉字点阵库)时,使用 Backup 也是较理想的选择。

Restore 是 DOS 提供的与 Backup 相匹配的备份文件恢复工具。用 Backup 备份的文件,要用与之配套的 Restore 来恢复。Backup 产生的备份软盘可能有许多张,这些盘被 Backup 从1号开始依次按顺序编号,Restore 时严格按此顺序执行恢复。

这种每次都要求从第一号盘开始执行恢复的“顺序”恢复技术至少有以下两点不足:

1. 有时我们只想从备份盘中恢复一个文件或少数几个文件,并知道这些文件在第几号备份盘上(在后面的附录一中,将给出观察备份盘上所有文件的名字及其路径的方法)。换言之,当我们希望从若干张备份盘中仅恢复一张(尤其是最后一张)时,DOS 的“顺序”恢复法既浪费时间又磨损磁盘。

2. 如果备份盘中有一张遗失或严重受损,则其后的所有盘都无法再用 Restore 恢复,因为“顺序”恢复法不允许跳过任何一张盘。若遗失的是第一号盘,损失将最大。

本文给出的备份盘,“随机”恢复技术有效地解决了上述两个问题。它允许用户按任一顺序执行恢复,一次可仅恢复一张

也可恢复多张备份盘。其原理是:在分析了备份盘数据组织结构的基础上(下面给出了 MS-DOS 备份盘的详细结构),编制随机恢复工具 MRestore 直接访问备份盘上的信息执行恢复。

下面是使用 MRestore 的一些具体细节:

在运行 MRestore 时,要回答以下两个问题:

- ① 备份盘在[A:]驱动器中还是在[B:]驱动器中。缺省为[A:];

- ② 恢复出来的文件是放到当它们被 backup 时所在的原有目录中,还是放到一个名为“C:\Restore\”的公共子目录中。缺省为“C:\Restore\”。(这也是比 DOS 的 Restore 更灵活的一个方面)。

在 MRestore 执行恢复时,会显示出下列信息:

- ① 备份盘的备份日期和时间;
- ② 备份盘的盘号,即该盘是第几号备份盘,并指出是否为最后一张盘;
- ③ 子目录名,以及该子目录在当前盘上的文件数;对于不存在的子目录,会给出相应目录被创建的信息。
- ④ 文件名。

随机恢复技术的要点之一是如何处理“部分文件”。DOS 的 Backup 在备份文件时为了节约磁盘空间,很有可能将一个文件的前一部分装在一张盘的末尾,而后一部分装在下一张盘的开始。在恢复时能很方便地对这类“部分文件”进行拼接是 DOS 的“顺序”恢复法之所以要坚持“顺序”的主要原因。而在随机法中,我们的作法只能是:“牺牲”“部分文件”的原有扩展名,用“部分号”作为该文件该部分的扩展名。例如:黑体字库 HZK24H.DOT 经 DOS 的 Backup 被装在两张软盘上,执行 MRestore 后(不管先插入哪张盘),将生成两个文件:HZK24H.1 和 HZK24H.2。随机法中“部分文件”的拼接一般要靠人工参与,对于上面的例子即为:

```
C>COPY/B HZK24H.1+HZK24H.2 HZK24H.DOT
```

```
C>DEL HZK24H.1
```

```
C>DEL HZK24H.2
```

当然,在随机法中不靠人工参与的“部分文件”自动拼接方法也是不难实现的,这在后面的附录一中将谈到。限于篇幅,MRestore 不支持自动拼接,感兴趣的读者可自行扩充。

最后说明两点:第一,附录一给出的备份盘结构适用于 MS-DOS 3.3 以上版本,低版本 DOS 的 Backup 产生的备份盘的结构与此完全类似,只是控制文件的名字不是 Backup.XXX 和 CONTROL.XXX;第二,与 DOS 的 Restore 相比,MRestore 除了能打乱盘次序这一点外并没有其它优于前者的地方,不过,一旦你的重要数据备份盘中有一张丢失或是其它各种原因致使 Restore 拒绝工作并使你束手无策时,希望本文能帮你减少损失、摆脱困境。

## DOS 备份盘的数据组织结构

为了便于读者理解和扩充 MRestore,现将经笔者分析并反复测试而得到的 MS-DOS 备份盘结构介绍如下:

在备份软盘根目录下,只有两个文件,一个是 BACKUP.XXX,另一个是 CONTROL.XXX,这里 XXX 即为盘顺序号,如 001,002。

- ① CONTROL.XXX 记录了备份文件的各种控制信息,如目录名、文件名、文件属性、日期、文件总长度、文件在当前盘上存储的长度等。用 PCTools 等工具观察此文件,即可了解备份盘中所有文件的名字及其路径名。

- ② BACKUP.XXX 是各文件自身的数据信息,它是各文件的严格拼接,中间没有任何控制信息或标志信息。MRestore 的功能即为读 CONTROL.XXX 中的控制信息,把 BACKUP.XXX 中的各文件“拆开”。下面是 CONTROL.XXX 的

结构(该结构与软盘是高密还是低密无关)。

CONTROL. XXX 文件中共有三种结构块:

文件头,长度为139字节;

目录块,长度为70字节;

文件块,长度是34字节。

排列的次序是:文件头,第一个目录的目录块,第一个目录下所有备份文件的各个文件块,第二个目录的目录块,第二个目录下所有备份文件的各个文件块,....。因此,CONTROL. XXX 的总长度为:139+70 \* n+34 \* m 个字节,其中 n 是当前盘上所有目录的总个数;m 是当前盘上所有目录下的所有备份文件的总个数。

三种结构块的结构如下:

〈1〉文件头:

在139字节文件头中的偏移(单位为字节,从1开始计,下同)	长 度 (单位为字节,下同)	含 义
1	1	十六进制8B,文件类型标志
2	8	字符串“BACKUP”
10	2	盘顺序号
12	12	均为“00”,为保留区
139	1	标志本张盘是否为备份盘中的最后一张。“00”非最后一张,“FF”是最后一张。

总长139

〈2〉目录块:

在70字节目录块中的偏移	长 度	含 义
1	1	十六进制46,目录块开始标志
2	63	目录的全路径名,多余的字节为0(DOS)支持的最长路径为63字节,因此这里不会溢出)
65	2	当前盘上本目录下备份文件的总个数
67	4	当前盘上的下一个目录在文件 CONTROL. XXX 中的总偏移。若没有下一个目录了,则为:“FF,FF,FF,FF”。

总长70

〈3〉文件块

在34字节文件块中的偏移	长 度	含 义
1	1	十六进制22,文件块开始标志
2	12	文件的名字及其扩展名
14	1	文件的最后一个“部分”存于当前盘上时为03,否则为02。
15	4	文件的总长度,即原始长度。(这是在随机法中进行自动拼接的关键数据。用它即可判断出一个文件的各“部分”是否都被恢复,若是,则进行自动拼接、恢复文件的原有扩展名并删除中间文件)
19	2	文件的“部分号”
21	4	文件数据在 BACKUP. XXX 中的偏移
25	4	文件数据在 BACKUP. XXX 中的长度(单位为字节)
29	1	文件备份前的属性
30	1	保留
31	4	文件的日期和时间信息

总长34



## 附二: MRestore 的 Turbo Pascal 源代码

```

Program MRestore;
uses Dos, Crt;
var
  f1          : file; {for control. xxx}
  f2          : file; {for backup. xxx}
  f3          : file; {for output-file}
  Drive, Ch   : char;
  FTime, I, J, K, M, Start, Size : longint;
  DT          : datetime;
  Buf1        : array[1..1024] of char;
  Buf2        : array[1..60000] of char;
  NumRead, NumWrite : word;
  Dirpath, Dir, Dir2, Numstr,   : string;
  Point, Flag, Pos1            : integer;
  Control, Backup, ResFile     : searchrec;
  Procedure IfDosError(sl:string);
    begin if DosError(>0) then begin writeln(sl); Halt(1); end; end;
  Procedure IfIOError(sl:string);
    begin if IOResult(>0) then begin writeln(sl); Halt(1); end; end;
  Function Asc(c9:char):longint;
    begin Asc:=ord(c9);      end;
begin {main}
  ClrScr;writeln('MRestore. Restore (Backup File).');
  writeln('[DOS Commands Extend Package]. Ma Wenqian, Beijing, 100872');
  writeln; write('Your Source Disk In (1)Drive A; or (2)Drive B; ?');
  Ch:=Readkey;if Ch='2' then Drive:='B';   else Drive:='A';
  writeln(' ', Drive, ' ');
  writeln('Restore To (1) c:\Restore\ or ');
  write('      (2) Directory from where the file was backedup. ?');
  Ch:=Readkey; if Ch='2' then Flag:=2 else Flag:=1;writeln(' ', Flag, ' ');
  Repeat
    writeln('Please Insert Your Source Disk In Drive ', Drive, ' ');
    writeln('Press Any Key When Ready...');    Ch:=readkey;
    writeln('-----');
  FindFirst(Drive+' \control. *', ReadOnly, Control);
    IfDosError('Warning!No File Can Be Restored. ');
  Assign(F1, Drive+' \'+Control. name);
  FileMode:=0; {Read-Only Mode. }
  {$I-} Reset (F1, 1); {$I+}
    IfIOError('Error!File Not Found or Drive Not Ready. ');
  GetFTime(F1, FTime); IfDosError('Error!File Error or Drive Not Ready. ');
  UnPack Time(FTime, DT);
  write('File was Backedup on: ', DT. year, '-', DT. month, '-', DT. day, ' ');
  writeln(DT. hour, ': ', DT. min, ': ', Dt. sec);
  Point:=0; Seek(F1, Point);BlockRead(F1, Buf1, 8, NumRead);
  if (Buf1[1]<>Chr(139)) or (Buf1[7]<>Chr(80))then
    begin writeln('Incorrect File (CONTROL. XXX)');Halt(1); end;
  Point:=9; Seek(F1, Point); BlockRead(F1, Buf1, 2, NumRead);
  I:=Asc(Buf1[1])+Asc(Buf1[2]) * 256; write('This is the No. 00', I, 'Disk');
  Point:=138; Seek(F1, Point); BlockRead(F1, Buf1, 1, NumRead);
  J:=Asc(Buf1[1]);if J=0 then writeln('And this is NOT the LAST Disk. ')
    else writeln('And this is just the LAST Disk. ');
  FindFirst(Drive+' \backup. ' +Control. name[9]+
    Control. name[10]+Control. name[11], ReadOnly, Backup);
    IfDosError('Warning!File Backup. XXX NOT FOUND. ');
  Assign(F2, Drive+' \'+Backup. name);
  {$I-} Reset (F2, 1); {$I+}
    IfIOError('File Not Found or Drive Not Ready. ');
  "GetDir(0, Dir) GetDir(3, Dir2);
  Repeat
    Point:=Point+67; Seek(F1, Point); BlockRead(F1, Buf1, 4, NumRead);
    J:=Asc(Buf1[1])+Asc(Buf1[2]) * 256 +Asc(Buf1[3]) * 256 * 256;
    Seek(F1, Point-65); BlockRead(F1, Buf1, 63, NumRead);
    write('Directory: \'); for k:=1 to 63 do write (Buf1[k]);writeln;
    if Flag=1 then Dirpath:='C:\RESTORE \ else Dirpath:='C:\';
    k:=1;while (Asc(Buf1[k])<>0)and(k<=63)do

```

```

begin Dirpath:=Dirpath+Buf1[k];
k:=k+1;end;
Dirpath:=Dirpath+'\' ; OneDir:='' ; chDir('C:\');
for k:=4 to length(Dirpath)do
if Dirpath[k]<>'\' then OneDir:=OneDir+Dirpath[k]
else begin
{ $ I- } ChDir(OneDir); { $ I+ }
if IOResult<>0 then
begin
MkDir(OneDir);
ChDir(OneDir);
writeln(' * * * * ',OneDir,'Has been Created. ');
end;
OneDir:='';
end;
Seek(F1,Point-2);
BlockRead(F1,Buf1,2,NumRead);
K:=Asc(Buf1[1])+Asc(Buf1[2])*256;
writeln('To',Dirpath,' ',K,'Total File(s). ');
Point:=Point+5;
for I:=1 to k do
begin
Seek(F1,Point);
BlockRead(F1,Buf1,33,NumRead); ResFile.name:='';
m:=1;while (Asc(Buf1[m])<>0) and (m<=12)do
begin ResFile.name:=ResFile.name+Buf1[m]; m:=m+1; end;
write(' ',ResFile.name);
Str(Asc(Buf1[18])+Asc(Buf1[19])*256,NumStr);
start:=Asc(Buf1[20])+Asc(Buf1[21])*256+Asc(Buf1[22])*256*256+Asc(Buf1[23])*256*256*256;
ResFile.size:=Asc(Buf1[24])+Asc(Buf1[25])*256+Asc(Buf1[26])*256*256+Asc(Buf1[27])*256*256*256;
ResFile.attr:=Asc(Buf1[28]);
ResFile.time:=Asc(Buf1[30])+Asc(Buf1[31])*256+Asc(Buf1[32])*256*256+Asc(Buf1[33])*256*256*256;
if (Asc(buf1[13])<>3) or (Asc(Buf1[18])<>1) then
begin Pos1:=Pos('.',ResFile.name);
if Pos1>0 then
begin Delete(ResFile.name,Pos1+1,Length(ResFile.name)-Pos1);
Insert(NumStr,ResFile.name,Pos1+1);
end
else ResFile.name:=ResFile.name+'.'+NumStr;
write('----',Resfile.name);
end;
end;
Assign(F3,ResFile.name);
{ $ I- } ReWrite(F3,1); { $ I+ }
IF IOError('Error ! File Write Error of Drive Not Ready. ');
Seek(F2,Start);
Size1:=ResFile.size;
repeat
if Size1>sizeof(Buf2) then Size1:=sizeof(Buf2);
BlockRead (F2,Buf2,Size1,NumRead);
BlockWrite (F3,Buf2,NumRead,NumWrite);
ResFile.size:=Resfile.size-Size1;
Size1:=ResFile.size;
until (ResFile.size=0) or (NumRead=0);
SetFTime(F3,ResFile.time);
Close(F3);
Assign(F3,ResFile.name);
SetFAttr(F3,ResFile.attr);
Point:=Point +34;writeln;
end; {all files in one directory, LOOP};
Point:=J-1;
until J>16000000; {all directories in one disk,LOOP}
Close(F1); Close(F2); ChDir(Dir2);ChDir(Dir);
writeln;writeln('Restore another disk(Y/N)?');
Ch:=Readkey;
until (Ch<>'y')and (Ch<>'Y');
end.

```

# 栏目编辑的话

实用软件是计算机用户维护系统和充分利用计算机系统资源的工具,目前各种实用工具应用面非常广泛,同时各种实用工具也在不断推出和发展。本栏目将重点介绍如系统维护工具、压缩工具、调试工具(Word、PCTOOLS、NORTON、DUP、PKZIP、ARJ)等实用软件的功能和特点,为用户合理、高效地利用计算机系统资源,提高自身工作效率和利用工具进行计算机系统的维护奠定基础。我们真诚地希望在实用软件的使用中有经验所得的广大专家、学者给我们以支持和帮助。

## Microsoft MS-DOS 6.2 简体中文版

**M**S-DOS 6.2 US 简体中文版是一个地区性的本地化产品。它是在 MS-DOS 6.2 US 版基础上开发的,并且置上了 DBCS 标志。新增的功能是:为 command.com 以及 MS-DOS 本身的 help.com 增加了双语言消息,此外还增加了汉字驱动程序,以支持简体中文的输入/输出功能。

该版本为支持更好的网络连接功能还包含有 WC 功能(Workgroup Connection——网络连接)。

### 1. 系统需求

MS-DOS 6.2 简体中文版可运行于至少 1 MB 内存的 286 PC 或更高档的计算机系统;VGA 或 Hercules 显示卡及监视器以及 1.2 MB、1.44 MB 的软盘驱动器,同时可以选择适用 LQ 1600K、CR3240 或 HP III 打印机,能运行于任何版本的 MS-DOS 之上。

### 2. MS-DOS 6.2 内核所支持的 DBCS

所有的 MS-DOS 命令和实用程序都识别 DBCS 字符,因此当一个 DBCS 系列的第二个字符满足 DOS 分隔符的条件时,是会产生错误分解的。分解的项(如文件说明)可以包含 DBCS 字符。

只有 MS-DOS 的 SBCS 版本所用的分隔符才会被当作分隔符来识别。可识别 DBCS 空白。在 DBCS 字符中,不会产生折叠。使用 SORT 时,每个 DBCS 字符被当作两个 SBCS 字符来对待。文件说明和卷标中可以包含 DBCS 字符。在文件名的前八个字符中和文件扩展名中,不能使用 DBCS 字符打头。

### 3. Command.com 和 Help.com 的双音消息

MS-DOS 6.2 简体中文版将 command.com 和 help.com 的信息翻译成了简体中文字符。这种消息将支持两种语言——英语和简体中文,用户可以在英文系统和汉字系统之间来回切换。在默认的情况下,当用户切换到汉字系统时,系统将显示简体中文信息,而当用户切换到英文系统时,就会显示英文信息。

### 4. 代码系统

MS-DOS 6.2 简体中文版的内码是 GB

2312-80, GB 2312-80 代码系统定义双字节的汉字编码,分别由区(行,每一字节的值-0a0h)和值(列,第二字节的值-0a0h)。编码范围是第一字节和第二字节都为 0a0h 到 0feh。每个区内有 94 个汉字。

01~10 区(第一字节为 0a1h~0aah):全角字符,包括全角 ASCII 符号。

11~15 区(第一字节为 0abh~0afh):空。

16~55 区(第一字节为 0b0h~0d8h):一级汉字,以拼音顺序排列。

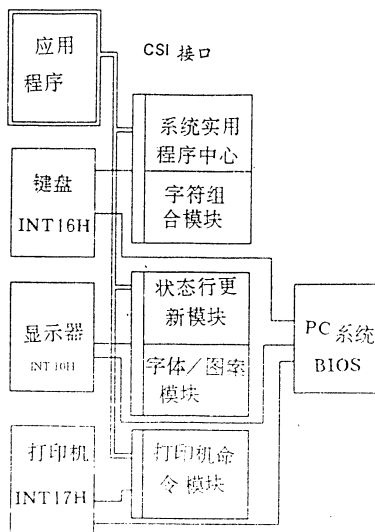
56~87 区(第一字节为 0d9h~0f8h):二级汉字,按笔划顺序排列。一级汉字加二级汉字,总共有 6763 个汉字。

88~94 区(第一字节为 0f8h~0feh):空。

简体中文代码的代码页为 936。

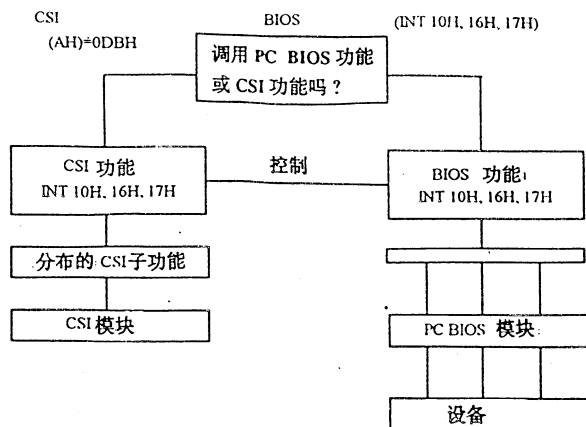
### 5. 汉字系统接口——CSI

CSI 是汉字系统输入输出接口规范(Chinese System I/O Interface Specification)写。CSI 定义了应用程序与简体中文系统之间的应用程序接口(Application Interface)。CSI 是在台湾的 CMES CSI 基础上根据 PRC(中华人民共和国)市场使用情况进行修改而形成的。它由五个模块组成:系统实用程序模块、字符组合模块、状态行更新模块、字体/图案模块、打印机命令模块。下图给出了应用程序、CSI 接口



和汉字系统之间的关系。

从系统组成的图示中可以看出,CSI 是系统 BIOS 的一个扩展。一旦用户从键盘调用汉字系统时,它就会要求系统 BIOS 来读取一个简体汉字字符,在 CSI 中的输入方法功能开始工作。当合成了一个汉字时,系统 BIOS 就将其返回给用户。在系统 BIOS 提供基本功能来控制外部 I/O 设备时,CSI 将组织这些功能来满足汉字系统的需要。下图描述了 CSI 和 PC BIOS 之间的关系。



如图所示,当用户使用 INT 指令来调用某个中断向量时,系统将使用 AH 寄存器中的值来决定这一功能是否应由 PC BIOS 功能处理,还是由 CSI 功能处理。对于外部的 I/O 设备,CSI 将把控制交给系统 BIOS。例如,你可以调用 CSI 功能来设定“下划线”和“双线打印”,但 CSI 仍需要通过 INT 17h 来控制打印机的操作。

#### 6. 简体中文字符输入输出系统→汉字系统

在此版本中,有一项很重要的加载功能,就是汉字输入输出系统,命名为汉字系统(han-system),它为用户提供汉字输入输出功能。从根本上讲,汉字系统是作为 MS-DOS 环境下的系统驱动程序来实现的。

汉字系统有一些功能,能帮助用户运行英文应用程序,比如在 MS-DOS 6.2 简体中文版下运行时要更舒服一些。用户使用一个切换开关(打开/关闭功能)来回地切换,便可得到最佳的汉字输入输出,而无论是在英文应用程序中还是在简体中文应用程序中。

- 半个汉字显示:半个汉字不能显示在一行的最后,应将整个字移到下一行。

- 光标调整:在汉字字符位置上的任何光标的移动、退格或删除,都按双字节处理。

- 画线代码控制:自动地判断从 0b1h 至 0dfh 之间的画线代码。这些代码位于 GB 码中。这么判断的目的是为了用真正的画线码字符代替简体字。

#### (1) 汉字系统的内核

汉字系统的内核初始化汉字系统所用的各个参数,管理每个已安装或未安装的驱动程序,解释应用程序和汉字系统之间的 CSI。在系统引导时,如果在 CONFIG.SYS 中指定驱动程序名(默认为 PBIOS.SYS),汉字系统就会自动加载。

由于 CSI 是一个开放的结构,因此很容易再往系统中加挂 IME/字体/打印机驱动程序。用户或开发厂家,也很容易使用自

己的显示器/打印机监控程序。汉字系统的内核能动态地管理驱动程序/监视程序的安装或撤消。

#### (2) 显示器驱动程序

在其零售产品中,本产品带有两个显示器驱动程序,它们是 VGA 和 Hercules,并支持一行或三行的状态行显示,下表描述了它们的特点:

	VGA (640×480)	Hercules (640×400)
显示一个状态行	每行中的字符为 16×18 (带两条空线) 文字区为 0 线至 449 线 状态区为 463 线至 479 线	每行中的字符为 16×15 (无空线) 文字区为 0 线至 374 线 状态区为 382 至 397 线
显示三个状态行	每行中的字符为 16×17 (带一条空线) 文字区为 0 至 424 线 状态区为 428 至 479 线	每行中的字符为 16×14 (无空线) 文字区为 0 至 349 线 状态区为 355 至 397 线

因为有些应用程序可能含有 DAS (Direct Access Screen——直接访问屏幕)的代码,因而要直接访问显示内存,并且因为汉字系统的显示器驱动程序用图形模式模拟 PC BIOS 里的文本模式,所以显示器驱动程序应该使用定时中断来解释和维护 DAS 里的文本缓冲区。每个显示器驱动程序的 DAS 显示内存布局如下:

- 至少带有 512 K RAM 的 VGA:

位平面 3  
位平面 2  
位平面 1  
位平面 0  
A0000H \_\_\_\_\_  
VARM  
(640×480)/8=38400 字节  
A9600H \_\_\_\_\_  
B0000H \_\_\_\_\_  
B8000H \_\_\_\_\_  
本文缓冲区 8 页  
BF000H \_\_\_\_\_

- 至少带有 256 K RAM 的 VGA:

位平面 3  
位平面 2  
位平面 1  
位平面 0  
A0000H/B0000H \_\_\_\_\_  
(640×480)/8=38400 字节 VARM  
A7F80H/B7F80H \_\_\_\_\_  
A8000H/B8000H \_\_\_\_\_  
文本缓冲区 6 页  
AFFFFH/BFFFFH \_\_\_\_\_

- Hercules, 640×640 模式, 图形和文本并存:

B0000H \_\_\_\_\_  
文本缓冲区  
B8000H \_\_\_\_\_  
图形缓冲区  
BFFFFH \_\_\_\_\_

#### (3) 键盘驱动程序

键盘驱动程序是在 CSI 规范基础上开发的,以提供绝大多数流行的输入方法,如拼音、双拼、区位码和国标码(GB 代码)等输入法。由于 CSI 是一个开放的结构规范,因此 OEM 供应者可以很容易地在系统中加挂他们自己的输入方法驱动程序。但是由于在中国,目前有多达 500 种输入方法,而其中的绝大多数是数据驱动的(象拼音库输入方法或笔划库输入方法)。键盘

驱动程序提供了一个通用输入方法,或称作表输入方法,最终用户可以很容易地用它来往系统中增加一种新的输入方法。

用户只需提供一个表文件,其中表明了 GB 码与输入键码之间的对应关系。此表可由任何文本编辑器生成。然后使用 DICTMANT 实用程序将此表转换成系统能读的格式。于是,用户便可往系统中添加一种输入方法。

当然,所有的键盘驱动程序都有某些智能功能,以帮助用户更容易地输入一个汉字。这些功能是:

- 词或短语:用户一次可输入一个词或短语,而不仅仅是一个字。
- 联想:用户可以自动地得到关联的词或短语。
- 自动判定:用户可以不用按键,就能在已按的键基础上,得到一个完整的词或短语。
- 可调整性:用户可根据使用的频度来调整词/短语的顺序。

#### (4) 字体驱动程序

本产品的零售软件包中,带有两种类型的字体驱动程序:位图驱动程序和 Suman 变倍字体驱动程序:

- 位图字体驱动程序:此驱动程序处理用于显示的 16×16 点阵汉字位图型字体,以及用于打印的四种 24×24 点阵的位图型字体(宋、仿宋、黑和楷)。各种 ASCII 码的 8×16 点阵的位图型字体取自原始的 PC BIOS。
- 可变倍字体驱动程序:此驱动程序处理四种汉字可变倍字体(宋、仿宋、黑和楷)以及相对应的四种英文变倍字体。可变倍大小的尺寸范围是从 24×24 到 2047×2047。此驱动程序同时还提供在 CSI 里定义的属性。

#### (5) 打印机驱动程序

MS-DOS 6.2 简体中文版里的打印机驱动程序包括:一个支持点阵打印机 LQ 1600K 的驱动程序,一个支持彩色打印机 CR 3240 的驱动程序,一个支持激光打印机 HP Laserjet III 的驱动程序。这三种打印机是市场中各类打印机中最流行的三种。

#### (6) 打印监视程序

应用程序可通过调用打印机驱动程序来打印出带有 CSI 中所定义属性的汉字。通常情况下,他们往打印机驱动程序中所写的文本序列都能打印出他们所希望的汉字效果。在中国所

使用的大多数打印机控制序列是晓军 2.13H 打印控制序列。当然,我们提供了一个打印监视程序来解释晓军 2.13H 打印机控制序列。

#### 7. 新的实用程序

新增加了几个实用程序来维护汉字系统:

- INSTDICT:往系统中安装 IME 字典。
- DICTMANT:为表输入方法维护 IME 字典。
- FONTMAKE:往图型字体的造字程序。
- CCSETUP:用于设置系统参数。包括键盘状态参数,显示风格参数和打印机控制参数。新设置系统参数在重新引导后才起作用。
- OLFNTMAK:一个可变倍的字体造字程序。

#### 8. 安装程序

MS-DOS 6.2 简体中文版的安装程序,用于安装 MS-DOS 的 3 个部分:MS-DOS 6.2、WC 和汉字系统。此程序的显示消息只有英文信息,因为这一版本是升级版本,而我们不能保证用户的老操作系统版本支持简体中文。

#### 9. Windows DOSBOX 支持

因为在任何版本的 Windows 运行之前,先要运行此版本的 MS-DOS,因此在 Windows 环境下的 DOS 块会自动地支持汉字。所有的 DOS 应用程序在 DOS 块下都会很正常地运行。但是也有一些限制,因为汉字系统并不是真正的文本模式:

- 拷贝和粘贴:对 MS-DOS 的 US 版本,所使用的是真正的文本模式,因而用户可以将窗口尺寸的 DOS 块中的文本拷贝和粘贴进 Windows 应用程序中。但是在 MS-DOS 6.2 简体中文版中,只能拷贝图象。为了解决这个问题,我们提供了一个新的 TSR 实用程序,名为 prntscrn,由它来完成全屏 DOS 块里的全部工作。
- 在 DOS 块的滚动速度较慢。因为系统需要处理位图图象。而不是字符代码,这就是为什么 DOS 块的滚动速度较慢的原因。
- 在 Windows 里不能得到 DOS 块中的字型。

#### 10. 网络支持

MS-DOS 6.2 简体中文版将支持 Microsoft Workgroup Connection、Microsoft LanManager 和 Novell。

(上接第 34 页)

```
SHR    AL,CL
CALL   DISP__ASC
POP     AX
AND     AL,0FH

DISP__ASC:
        CMP    AL,9
        JBE    DA1
        ADD    AL,7

DA1:
        ADD    AL,'0'
        MOV    DL,AL
        MOV    AH,2
        INT    21H
        RETN

HEX2AXC1    ENDP
```

```
;
CODE      ENDS
                END    EBGIN
```


可以看出:在该过程中,BYTE2ASC和 DISP-ASC 不但可作为过程 HEX2ASC 的新的入口,而且被本过程自己调用。自调用的原则是:过程的前面部分调用后面的部分,产生这个原则的根据是:必须保证过程能正确结束,这与递归调用的第一条原则基本相同。

从汇编语言程序的这两个使用技巧,可以得出如下结论:

程序只是一种机械的指令序列,程序怎么写,机器就怎么执行,因此不必拘泥于人类的思考问题的原则,考虑这种作法的道理何在,那种作法有何根据,只要保证机器代码正确就可保证程序的正确执行。希望本文能给您一些启发,使您能开发出更多的编程技巧来。

# FoxPro 的过去、现在和未来

□ 管 蕾

 FoxPro 2.5 的出现有其很长的一段历史渊源。最初当 IBM-PC 和兼容机在美国普及时, Ashton-Tate 公司推出 dBASE III 和 dBASE III+。由于它的功能, dBASE 成为微机上最普及的数据库管理系统。在其最高峰, 估计大约 80% 至 85% 的 PC 兼容机使用 dBASE, dBASE 成为一种事实上的工业标准。

为了解决 dBASE III 和 dBASE plus 的编译、伪编译等问题, 几个与 Ashton-Tate 无关的公司推出了与 dBASE III 兼容的产品, 其中著名的有 clipper, FoxBASE, FoxBASE+, FoxBASE+ 与 dBASE 兼容, 能伪编译 FoxBASE 的源程序, 被公认为是 dBASE 最好的编译系统。这是 dBASE 发展的第二阶段。

当 FoxPro 1.0 和 2.0 版出现时, dBASE 的发展进入第三阶段。FoxPro 1.0 比 dBASE III plus 快 8 倍, 比 dBASE III 快 16 倍, 比 FoxBASE + 2.10 快 2 倍, 并与 dBASE IV 兼容(与 dBASE III plus 和 FoxBASE+ 全兼容), 并且许多功能有提高, 比 dBASE IV 多 140 多条语句, 比 FoxBASE + 2.10 多 200 多条语句, 有 90 多条两者都没有的命令和函数。1991 年 7 月推出的 FoxPro 2.0 比 1.0 版又上一个台阶, 除了支持 FoxPro 1.0 的全部功能外, 又增加了 100 条新的函数和命令。更重要的, FoxPro 提供了一整套基于 GUI 的程序开发工具, 如:

Project Manager 供程序开发、编译、版本控制用。

Screen Builder 屏幕生成器, 以交互手段定义程序的录入, 显示界面并能生成 FoxPro 源程序。

Menu Builder 交互定义下拉菜单, 同样可生成 FoxPro 的源程序。

Report Builder 报表生成器。

Label Builder 标签生成器等。

FoxPro 在编译上实现真编译, 能使 FoxPro 的源程序编译成 EXE 文件, 在 DOS 下脱离 FoxPro 环境而运行。

FoxPro 引入关系数据库标准的查询语言 SQL(美国国家标准局标准)使 FoxPro 有了大多数数据库系统(ORACLE、SYBASE、INFORMIX)引以自豪的标准查询语言, 数据库查询能力大大提高而操作大大简化。

FoxPro 2.0 性能的核心是引入了 Rushmore 技术。Rushmore 使 FoxPro 2.0 能在个人计算机上处理 Very large Data Base(真正的大数据库, 记录数可达百万), 速度可与大型机媲美。

Microsoft 公司推出 FoxPro 2.5(DOS 版、Windows 版、Macintosh、Unix 版)后使 XBASE 进入第四阶段。Microsoft

FoxPro 2.5 是目前 PC 机上速度最快、数据类型最丰富的关系数据库系统, 是所有 FoxBASE、Clipper 用户最有效的数据库升级方案。FoxPro 2.5 以 XBASE 语言为基础向下兼容(FoxBASE、FoxPro 2.0、dBASE IV), 内嵌 SQL 语言, 全 GUI 界面。其速度是 FoxPro 2.0 的三倍, 是 dBASE 的三十倍。最引人注目是 FoxPro 2.5 由 Microsoft 支持, 所有 FoxPro 2.5 程序可以在 DOS、Macintosh、Unix 多个平台上运行, 使 XBASE 的跨平台移植问题变得十分容易。FoxPro 2.5 Windows 版本扩充了许多数据类型, 全图形界面, 借助于 DDE, 可与 Windows 平台上其他工具进行数据交换, 使数据库可以处理多种媒体, 诸如: 图象、声音、超长文本、图形等, 其网络上的事务处理功能首次在 XBASE 产品中引入事务概念, 使得 PC 机上的数据库系统可以拥有小型机、大型机上的高可靠性、安全性。其提供的 ODBC 联接机制, 使 FoxPro 能访问其他小型机、大型机上的数据库的数据, 使数据库的应用领域跨入分布处理的时代。FoxPro 2.5 可以说是目前微机上最简单、最有用户基础、最有前途的数据库系统。

现在国内 PC 机上的数据库用户已纷纷转向 FoxPro 2.5。FoxPro 2.5 DOS 中文版、Windows 中文版已推出。许多与 FoxPro 2.0、2.5 配套的产品如增加其汉字功能、图形功能、图象功能等的产品正陆续上市, FoxPro 2.5 配套产品已成为微机数据库市场的热点。

将于 1994 年末推出的 FoxPro 3 将提供一个 Windows 开发的新结构功能如下:

- 增强了事件模型, 包括基于 Windows 事件
- 对象编程扩展
- 屏幕编辑器加强
- 更多数据类型
- 丰富的 SQL 命令
- 空支持
- 全透明远程数据访问
- 数据库主机分布查询处理机
- 后向兼容性等。

FoxPro 3 之后, Microsoft 打算象它的其他产品的结构方式一样, 将 FoxPro 组件化, 并不断增强它的功能。这将意味着使产品成为功能组件, 可以与其他产品共用。这样, 在 FoxPro、Access 或 Excel 中可以使用同一查询对话框类抽取数据。

问题的关键在于 OLE2, 它使对象连接和嵌入进化为一种新的方式, 使产品可以在 ID 级别上协作。一旦认识到 OLE 2.0 怎样改变了计算环境的本质, 就能够预知将来的 FoxPro 会更适应 Microsoft 应用程序开发系统。



# 栏目编辑的话

本栏目将用户开发与应用过程中时常遇到的典型问题一一进行解答。它以简练的语言提供实用问题的解决方法。实际上本栏目的每一问一答都是读者在用机过程中时常遇到的一些有待解决的问题,同时也是实用性的小文章,我们将它们高度浓缩出来贡献给读者,以期能贯彻本刊让读者用最少的的时间汲取更多知识和经验的办刊方针。同时我真诚地希望通过读者反馈卡,给我们提出意见和见议,本栏目内容是否丰富,内容是否需要加强,栏目怎样设置更醒目等,以使本栏目办得更好。

## DOS 与 Windows

**?** 当我用带/S选项格式化盘时DOS提示“Invalidet DOS disk in drive A”当我将DOS盘插入驱动器后,DOS重又提示该信息然后什么也不做,到底怎么回事了?

**!** DOS生成一张可启动盘,需IO.SYS、MSDOS.SYS、COMMAND.COM这三个文件支持,其中前两个文件为隐含文件,而COMMAND.COM显示DOS提示符并对输入指令进行处理。要生成一张可启动盘,DOS必须将这三个文件拷到你已格式化的盘上。

DOS会在当前驱动器上寻找两个隐含文件。对COMMAND.COM还会在COMSPEC环境设置的目录中寻找。如果硬盘有两个或多个逻辑分区,你一定要把可启动分区(通常为C驱)当成启动盘。如果你有两个软驱并指定其中一个为当前驱动器并对另一个执行FORMAT/S时,一定要确定该驱动器内有与硬盘DOS版本一致的启动盘。

**?** 我有两台计算机:一台是运行DOS 5.0的便携机,另一台是运行DOS 3.3的286PC机。我在便携机出问题时用286作备份系统,尽管我可以用DOS的BACKUP命令备份便携机的数据,但当我在286上恢复数据时总遇到“Incorrect DOS version”错误提示,这该怎么办?

**!** 原因是不同DOS版本的BACKUP和RESTORE命令会有变化。用DOS 5.0版本下的BACKUP备份的文件在DOS 3.3版本下的RESTORE的通常不能恢复。你可以为80286做一张DOS 5.0启动盘并用其中的RESTORE命令将文件恢复到80286硬盘上。如果能将你的计算机升到DOSS. 5.0该过程就很简单方便了。

**?** 我用RESTORE命令恢复一BACKUP备份的磁盘,在C>状态下键入:RESTORE A: C:但在读第一张盘后马上就提示插入第二张盘,等到全部恢复完后,硬盘中并没有被恢复的文件,是什么原因?

**!** RESTORE命令是和BACKUP命令配合使用的恢复和备份程序。一般而言,BACKUP在备份硬盘中一个子目录下的文件时,由于该子目录下存在同时要备份的子目录,于是采用如下命令:BACKUP C:\SUBDIR/S,将SUBDIR及其下级子目录的文件一并备份出来。此种BACKUP盘在RESTORE时会出现上述现象。对于此问题的解决可采用如下命令:RESTORE A: C:/S。如果你想查看BACKUP盘中的内容所在的子目录可用DEBUG调入Control. 00X文件来查看。

**?** 我作XCOPY的/E选项时总有出错信息“Invalid Switch”你能告诉我怎么回事?

**!** XCOPY的/E选项告之XCOPY复制包括空目录的所有子目录,此选项不能单独使用,应首先用/S选项告之XCOPY复制非空的目录和子目录。若无/S选项,/E项会返回出错信息的。

**?** 由于我购买的磁盘质量较差,一段时间后,0磁道坏了,用FORMAT命令不能格式化,如何用最简单的方法再利用这张磁盘?

**!** 最简单,而又不需要什么技术的方法是打开磁盘的上部将盘片取出,翻个面再放入,然后重新格式化即可。这样原盘的1面变成现在的0面就避开了原来毁坏的0磁道。

**?** 为什么DOS应用程序不能在XMS内存中运行,而WINDOWS应用程序确可以?

**!** 所有的DOS应用程序以及软设备的驱动程序都运行于所谓的实方式下,在这种方式下,CPU只能用1MB以内的内存空间,其中包括为系统硬件保留的640KB到1MB之间的384KB空间,这就是DOS640KB的界限。事实上,DOS认为1MB以上的内存空间是不存在的,80386以上的芯片,在不同的方式下可

以寻址更高的内存,WINDOWS 可以在保护方式下运行,因此突破 1MB 的界限。

**?** 在 DOS5.0 下,如果 CONFIG.SYS 文件中有如下语句:

```
DEVICE = C:\DOS\EMM386.EXE
RAM 1024
```

系统运行时挂起;当用上述命令装入 EMM386.EXE 后,有时硬盘的响应速度很慢,或出现“Driver can not ready”的错误信息,原因何在?

**!** 这个问题可能是由于内存地址冲突引起的。EMM386.EXE 在内存的高端要建立一个内存映像页,该内存映像页占用 64K 字节,用以实现传统内存和 EMS 内存的数据交换,EMM386.EXE 占用的地址可能和硬盘 ROMBIOS 的控制程序占用的地址冲突,因而造成了这一现象。内存块值如下:

为了避免这一问题,我们可以采用下列语句改变 EMS 映像页的地址:

```
EMM386 Mx 1024 RAM
```

x 为下表中的内存块值之一,通过设置 x

内存块值	占用地址	内存块值	占用地址
1	C000H	8	DC00H
2	C400H	9	E000H
3	C800H	10	8000H
4	CC00H	11	8400H
5	D000H	12	8800H
6	D400H	13	8C00H
7	D800H	14	9000H

的值可以避免这一问题。

除此之外,在应用 EMM386.EXE 时还可能出现一些其他的错误信息,如“EMM386 Exception...”等,这些问题基本上是由于内存冲突造成的。EMM386.EXE 还提供了一个开关 X=xxxx-xxxx,它可以避免 EMM386 和系统硬件抢占 UMA,具体用法是在 CONFIG.SYS 文件中加入语句:

```
DEVICE=C:\DOS\EMM386.EXE
X=xxxx-xxxx(如 X=A000-EFFF)
```

EMM386.EXE 可能出现的另一个问题是由于堆栈引起的,如“EMM386 Exception Error #12,这一问题可能是由于处理由鼠标和键盘产生的中断时 DOS 产生堆栈溢出而造成的,解决这一问题可在 CONFIG.SYS 文件中加入如下语句:

```
STACKS=X,Y
```

X 为堆栈数,其值为 0 到 64

Y 为堆栈的大小,其值为 32 到 512。

**?** 在 80386 微机上使用 DOS5.0 操作系统,系统配置文件 CONFIG.SYS

中有 HIMEM.SYS、EMM386.SYS,同时在 AUTOEXEC.BAT 中装有 SMARTDRV.EXE 驱动程序和鼠标驱动程序,一般而言,MOUSE 和 SMARTDRV 都将在系统基本内存中占 30K 的空间,那么我们能否将这两个程序的驻留部分装入高端内存?如果能,怎么办?

**!** 可以将这两个程序装入高端内存,一般而言,采用 DOS5.0 的 80386 以上微机,可以利用两个命令 DEVICE-HIGH 和 LOADHIGH 来完成这一功能,将 MOUSE 和 SMARTDRV 的驻留部分装入扩展内存(EXTENDED MEMORY),即 640KB 的基本内存到 1MB 以内的 384KB 空间中。具体作法是在 CONFIG.SYS 的首行加入:DOS=HIGH,UMB,此语句通知 DOS 将自身装入扩展内存的前 64K 中,此区域称之为 HMA(High Memory Area)。而 UMB 开关用以通知 DOS 用户将通过 LOADHIGH 或 DEVICEHIGH 命令将程序装入高端内存,并为 DOS 和装入高端内存和程序建立一种联系。不使用 UMB 则 LONDRHIGH 或 DEVGOEHIGH 装入的程序可能不能正常运行。DOS 如果使用扩展内存则必须通过扩展管理程序,因此 CONFIG.SYS 文件中的第二行应该是:DEVICE=C:\DOS\HIMEM.SYS,至此整个 CONFIG.SYS 文件可以构造如下:

```
C>copy con Config.sys
DOS=HIGH,UMB
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=EMM386.EXE XXXX RAM
DEVICEHIGH=C:\DOS\MOUSE.SYS
```

在 AUTOEXEC.BAT 中加入:

```
LOADHIGH C:\DOS\SMART-
DRV.EXE YYYY,其中 XXXX,YYYY
为内存空间容量,XXXX 应大于 YYYY。
```

**?** 近来我安装了 Windows3.1 并经常频繁运行 DOS 程序。在 Windows3.1 的窗口内运行 DOS 程序没什么问题,只是在 DOS 程序内鼠标死掉,这是为什么?

**!** 首先,确认你已在 Windows 启动前正确地装入鼠标驱动程序。做法是在 CONFIG.SYS 中加入:

```
DEVICE=MOUSE.SYS
```

或者在 AUTOEXEC.BAT 文件加入 MOUSE.COM。

注意,如果在 DOS 下程序不能驱动鼠标,则在 Windows 下该程序仍不能驱动鼠标。

如果鼠标驱动程序正确安装,在

Windows 调入程序后仍不能驱动鼠标,可能是鼠标驱动程序的版本太老,需要换个新版本,如果是 Microsoft 鼠标,版本应为 Version8.2。

**?** 如何使用 DOS 的 PRINT 命令交换 LPT2 和 LPT1,由于 DOS 不允许在打印进程中使用一次以上的/D 参数,不得不在每条指令后都加/D。

**!** DOS 的 PRINT 命令不允许在打印文件队列中切换并行口。解决办法是用批文件执行打印任务。由于批文件允许每次执行“新的”PRINT 命令,你可以在打印两个文件的中间切换并行口,这里有一个叫 PRINTAC1.BAT 的例子:

```
@ECHO OFF
PRINT File1.TXT /D:LPT1
PRINT File2.TXT /D:LPT2
PRINT File3.TXT /D:LPT1
PRINT File4.TXT /D:LPT2
```

不过 PRINAC1.BAT 虽可以完成任务,但 PRINT 就不能在 DOS 命令做其它工作时成为后台进程。

**?** 是否可以通过软件方法将 LPT1 打印机端口改向到 LPT2 端口?

**!** 可以,LPT1 和 LPT2 的端口地址存储在 BIOS 的数据区中,将这两个端口地址互换,即可实现打印机的改向输出。下列程序可以将打印机的输出从一个并行口改向到另一个并行口。

在适当的编辑器下以文本格式编辑

```
PORT.SCR:
```

```
N PORT.SCR
A 100
XOR AX,AX ;AX 寄存器清零
MOV DS,AX ;将 DS 段寄存器清零
MOV SI,408 ;BIOS 数据区偏移地址
MOV AX,[SI] ;取 LPT1 端口地址
MOV BX,[SI+2] ;取 LPT2 端口地址
XCHG AX,BX ;交换寄存器的值
MOV [SI],AX ;——交换 LPT1 和 LPT2
MOV [SI+2],BX ;——两打印端口地址
INT 20
RCX
15
W 100
Q
```

此程序编辑完成之后,在 DOS 提示符下键入:

```
DEBUG<PORT.SCR:
```

DEBUG 则自动创建 PORT.COM 程序,在以后运行中运行 PORT 就可以改向打印机的输出,再次运行 PORT 则打印改向又被恢复。值得一提的是这里给出了一种用 DEBUG 建立较长的可执行文件的方法,由于 DEBUG 的 A 命令不能进行编辑,输入较长的文件时很不方便,通过这种方法可弥补这一缺陷。

# 栏目编辑的话

本栏目以国内外出现的有代表性的新产品为对象,介绍这些产品的特性、特点及功能,为用户选择适合自身的产品提供依据。

从创刊开始我们将紧跟市场发展,发现新的、适合用户的产品并跟踪这些产品的完善与发展,同时我们也真诚地希望广大读者为我们更好地追踪新产品提供更多的稿件或线索。

## 一种新兴的网络系统 ——向宇对等网系统

□ 易 言

有关资料介绍,目前国内拥有的微机数量已达一百四十余万台,数量巨大,但是联网的机器很少,资源被白白地浪费。众所周知,大型局域网需要价格昂贵的专用服务器,而且需要专业人员进行维护;同时,加之国内还不具有主干网路,没有雄厚的基础设施,形成大规模的网络群比较困难,可以说,目前大型局域网还不适合中国的实际情况,鉴于此种情况,向宇计算机公司从实际出发投资研制成功了符合中国国情、具有中国特色的对等网系统——X&YNET,填补了国内在网络系统方面的空白。这一系统是我国第一套具有独立版权的对等网系统,向上可以和大网相联,同时自身又是一种独立的网络环境。用户可以在此环境下进行开发、应用。它的研制成功标志着我国对等网络技术跃上了一个新的台阶,必将对我国计算机行业产生深远影响。

对等网(Peer-to-Peer Network)是局域网的一种,在网络中各节点所处的地位是对等的,即没有主、从之分,各节点之间可以共享资源,即共享数据、共享文件、共享外设等等,网络各节点之间的数据传送不通过专门的服务器,直接在各节点间形成传输链路,具有操作简单(熟悉DOS的用户不需要专门的培训和额外的学习过程即可象在DOS下操作微机一样),使用方便,无需专门技术人员进行网络维护等特点。与此同时,X&YNET具有与大型局域网相兼容的功能,主要适用于机关、企事业单位、公司内部管理,X&YNET所管理的节点由几个可以增至数百个不等。

目前在西方发达国家,对等网络发展十分迅速,销售量成倍增长,已经成为网络系统中不可忽视的重要一员。据外电报道,1993年全美国,对等网的销售额已突破亿元大关,网络专家估计1994年的销售总额将达到十亿美元,市场潜力巨大。对等网已成为微型计算机网络系统的一匹耀眼的“黑马”,在我们计算机网络应用中将起到积极的促进作用。

向宇计算机公司紧跟世界网络技术发展的潮流,瞄准国内市场,抓住中国特色,率先研制成功了向宇对等网系统,并即将投放市场。向宇对等网除具备一般对等网的文件共享、打印共享功能外,还紧扣中国特色,发展了一大批新的、适合汉字操作环境的功能。首先向宇对等网能与汉字软件系统兼容,能使网上的节

点共享汉字系统如WPS等,解决了汉字文件的传输、汉字软件的共享使用问题;另外针对计算机外设价格昂贵,而大部分工作单位没有足够的财力为每一台计算机都配齐一整套外设的情况,开发了外部设备共享功能,使诸多种类的外设可以在网络上互相使用。这些设备包括绘图仪、激光打印机、FAX/Modem卡、数字化仪等,可以为用户节省大量金钱,提高了工作效率;为了适合学校的教学任务,X&YNET开发了各节点微机间的操作透明功能。向宇计算机公司从用户的观点出发,苦心研究,一反网络系统映射驱动器的作法,采用不改变DOS操作习惯的映射子目录的方法,使得用户共享X&YNET网络上其它用户机器的文件时,如同使用本机上子目录中的文件一样,大大提高了用户界面的友好性,使用户能克服烦琐学习过程的阻碍,方便迅速地进行使用。在向宇对等网的设计过程中,考虑到我国具体国情,吸取大型网络文件管理和用户权限设定的经验,加强了网络文件和用户使用权限的管理,形成了一套独特的网络安全方法,这些方法在网络安全管理方面具有鲜明的特色。X&YNET在系统安装上更是精益求精,基本上做到了智能化的安装,用户不需过多的人工干预,即可安装成功。

尤其值得一提的是向宇计算机公司顺应世界潮流,走开放式道路,不久将公开对等网接口细节,欢迎第三方开发商在此平台上开发应用软件;同时公司亦筹集了大量的资金和众多的开发技术人员,准备下大力气、花大力量进行应用软件的开发。希望此系统能更好地满足社会各界的需要。一手抓通用软件工具、通用应用系统的开发,一手抓专业应用软件的开发,并将两者结合起来,形成一套丰富多彩、功能强大的对等网络系统,是向宇公司对等网络系统的目标。

**向宇对等网(X&YNET)**

北京向宇计算机公司

地址:北京复兴路甲20号27分号

312, 314 邮编:100036

电话:8263441 2063366 呼1511

## 读者意见征询卡(复印有效)

**为**使本刊越办越好,请读者填写此卡。我们将根据寄回的征询卡,挑选 100 名本刊热心读者,免费赠送本刊 1995 年全年杂志。

读者姓名:\_\_\_\_\_年龄:\_\_\_\_\_职务或职称:\_\_\_\_\_

工作单位:\_\_\_\_\_电话:\_\_\_\_\_

通信地址:\_\_\_\_\_邮编:\_\_\_\_\_

▲您对本刊所设栏目的看法:(好:✓,一般:○,不好:✕)

- |                                |                               |                                  |                                 |
|--------------------------------|-------------------------------|----------------------------------|---------------------------------|
| <input type="checkbox"/> 新技术追踪 | <input type="checkbox"/> 软平台  | <input type="checkbox"/> 图形图象处理  | <input type="checkbox"/> 汉字处理   |
| <input type="checkbox"/> 编程语言  | <input type="checkbox"/> 数据库  | <input type="checkbox"/> 计算机安全   | <input type="checkbox"/> 网络与通讯  |
| <input type="checkbox"/> 软件维护  | <input type="checkbox"/> 实用软件 | <input type="checkbox"/> 硬件维护    | <input type="checkbox"/> 电脑博士信箱 |
| <input type="checkbox"/> 新产品   | <input type="checkbox"/> 明星企业 | <input type="checkbox"/> 反病毒技术公告 |                                 |

您认为本期各栏目的质量如何(好:✓,一般:○,不好:✕)

- |                                 |                                |                                |
|---------------------------------|--------------------------------|--------------------------------|
| <input type="checkbox"/> 软平台    | <input type="checkbox"/> 图形与图象 | <input type="checkbox"/> 汉字处理  |
| <input type="checkbox"/> 编程语言   | <input type="checkbox"/> 数据库   | <input type="checkbox"/> 计算机安全 |
| <input type="checkbox"/> 软件维护   | <input type="checkbox"/> 实用软件  | <input type="checkbox"/> 硬件维护  |
| <input type="checkbox"/> 电脑博士信箱 | <input type="checkbox"/> 新产品   |                                |

▲您对本刊的总体看法:(好:✓,一般:○,不好:✕)

- |                               |                               |                               |                               |
|-------------------------------|-------------------------------|-------------------------------|-------------------------------|
| <input type="checkbox"/> 文章内容 | <input type="checkbox"/> 栏目设置 | <input type="checkbox"/> 版面安排 | <input type="checkbox"/> 编辑质量 |
| <input type="checkbox"/> 服务水平 | <input type="checkbox"/> 广告质量 | <input type="checkbox"/> 印刷质量 |                               |

▲您认为本刊是否适合你的需要? ☐ 很适合 ☐ 基本适合 ☐ 不适合

▲您最喜欢的本期栏目及文章是哪个(篇)?

▲您还希望本刊增加哪些栏目或哪些方面的内容?

▲您是否还有其他建议?

### 填卡须知:

请在每个选项前的方框内按要求作标记。需填写的项目,请用工整的字迹填写,写不下可另附纸。填好后沿虚线剪下(或复印),寄往:(100006)北京市劳动人民文化宫内《电脑编程技巧与维护》杂志社。

## 读者服务卡(复印有效)

**凡**需要本刊代为联系本刊中所出现的有关公司的读者,请填此卡。

读者姓名:\_\_\_\_\_职务或职称:\_\_\_\_\_传真:\_\_\_\_\_

工作单位:\_\_\_\_\_电话:\_\_\_\_\_

通信地址:\_\_\_\_\_邮编:\_\_\_\_\_

对本刊\_\_\_\_年第\_\_\_\_期第\_\_\_\_页 ☐ 彩色广告 ☐ 黑白广告 ☐ 正文中出现的  
公司的\_\_\_\_\_产品感兴趣

希望:☐ 寄取公司资料 ☐ 寄取产品目录 ☐ 寄取产品资料  
☐ 询问价格 ☐ 建立业务联系 ☐ 其他\_\_\_\_\_

## 《电脑编程技巧与维护》订阅单(复印有效)

地 址				邮 编	
单 位				收 件 人	
起订日期	年 月 日			共 计	期
订阅份数		款 数		汇款方式	

## 《电脑编程技巧与维护》订阅单(复印有效)

地 址				邮 编	
单 位				收 件 人	
起订日期	年 月 日			共 计	期
订阅份数		款 数		汇款方式	

## 诚征全国直销代理

自《电脑编程技巧与维护》月刊征订启事和各报刊杂志的消息及广告刊出之后,不少单位的图书馆、资料室、培训部门、书刊经销商以及广大热心读者来信、来电询问本刊的出版发行情况,并要求作为本刊的直销代理,有些读者、同仁甚至不辞劳苦来本刊编辑部谈直销事宜。这一方面反应出本刊“实用第一、智慧密集的办刊原则”的正确性,另一方面又反应了读者想尽早、方便地得到本刊的愿望,为此,我社决定,在全国诚征直销代理(单位或个人),其原则是:方便读者互惠互利、大家共享、共同发展。

### 具体办法如下:

- ① 包销:杂志社给予包销客户以较大优惠。
  - ② 代销:杂志社给予较好的优惠,在结清第一次代销杂志的款项后,按代销客户的要求发放下批杂志。
- 为保证包销客户的利益,本杂志社对首次代销的客户以 50% 的优惠价提供 10 份样刊试销,代销和包销的优惠率及合同如下:

### 包销与代销优惠率:

包销	优惠率	代销	优惠率
10~50 份	20%	10~50 份	18%
50~100 份	25%	50~100 份	22%
200 份以上	30%	200 份以上	25%

### 包销与代销合同(复印有效)

甲方名称					
通信地址					
邮政编码		电 话		传 真	
甲方代表签字			经办人		
乙方名称	《电脑编程技巧与维护》杂志社				
通信地址	北京市劳动人民文化宫内				
邮政编码	100006	电 话	5232502	传 真	
乙方代表签字	孙茹萍		经办人	刘伟	

经双方协商同意×××××××(甲方)愿作乙方(《电脑编程技巧与维护》杂志社)的包销或代销

者,按×××份的优惠率进行结算。本合同自双方代表签字之日起生效。

### 订 阅 须 知:

△本刊每月 8 日出版,每期订价:3.80 元,全年订价:45.6 元。1994 年出版发行 6 期。1994 年下半年交邮局发行,邮发刊号 24—106。  
 △请将本卡寄至:(100006)北京劳动人民文化宫内《电脑编程技巧与维护》杂志社发行部。联系电话:(01)5123823。  
 △如从邮局汇款,请按上述地址汇寄;如从银行汇款,请汇至:开户银行:中国农业银行北京分行东四北分理处 142 服。银行帐号:8014054。  
 帐户:电脑编程技巧与维护编辑部。款到后即按月寄刊。