386保护模式程序设计

技巧与实例

亦　欧
潘旭燕　编
徐　曼

北京希望电脑公司

北京希望电脑公司高级程序设计丛书

386保护模式程序设计
技巧与实例

亦 欧 潘旭燕 徐 曼 编
希 望 审

北京希望电脑公司

内容简介

本书对在80386微机上进行保护模式程序设计的方法和技术从理论和实践两个方面进行了说明，深入浅出，通俗易懂。除了介绍80386保护模式程序设计的理论和方法、技术之外，还介绍了将实模式下的程序转换为保护模式程序的方法，并且对屏幕、窗口、键盘、鼠标等程序设计方面给出了实用工具函数实例。书中还讨论了DOS扩展管理程序的功能和用法。

本书不仅可以作为关于80386保护模式程序设计的理论参考书，更可以使用实用技术手册来使用。

需要本书的用户，请直接与北京8721信箱联系。电话2562329，邮码100080。

京准印字：3584－91584

内部成本：17.00元

前　言

保护模式程序设计是在80386微处理器上进行的一种高效的程序设计方法和技术。80386微处理器保护模式提供了一种非常巧妙的基于硬件的设计方法，它能阻止一个程序去访问另一程序码和数据段。若在计算机上安装了这种保护模式的程序，那么在它运行时，任何程序只能访问它自己的程序码和数据。这种数据隔离方式对多任务环境非常有用。比如，在多任务操作系统下，一个正在运行的程序无法去破坏另一个正在运行程序数据。

本书对在80386微机上进行保护模式程序设计的方法和技术从理论和实践两个方面进行了说明，深入浅出，通俗易懂。对已经采用80386保护模式进行程序设计的人员来说，这是一本很好的参考书，书中给出的实例已经成熟，可以直接应用到程序设计实践中；对于初次接触80386保护模式的程序设计来说，本书也是一本比较好的入门读物，书中给出的万行程序能够帮助初学者迅速地入门。

本书除了介绍80386保护模式程序设计的理论和方法、技术之外，还介绍了将实模式下的程序转换为保护模式程序的方法，并且对屏幕、窗口、键盘、鼠标等程序设计方面给出了实用工具函数实例，这些例程可以直接应用。在第一章中还讨论了DOS扩展管理程序的功能和用法。因此，本书不仅可以作为理论参考书，更可以使用实用技术手册来使用。

编　者

目 录

第一章　保护模式程序概述

保护模式程序工具涉及到 8086 汇编和 C 编译器，但 16 位编译器和 32 位的保护模式编译器之间存在着一定的联系和差别，并且影响 C 程序。在处理同一件事时，使用 32 位 80386 保护模式程序要比使用 16 位 8086 程序工具容易些。

80386 微处理器保护模式提供了一种非常巧妙的基于硬件的设计方法，它能阻止一个程序去访问另一程序码和数据段。若在计算机上安装了这种保护模式的程序，那么在它运行时，任何程序只能访问它自己的程序码和数据。这种数据隔离方式对多任务环境非常有用。比如，在多任务操作系统下，一个正在运行的程序无法去破坏另一个正在运行程序数据。

8086 实模式的程序在硬件上进行直接内存访问，这样一个程序就能读写另一个程序的代码码和数据段。

举个例子，有一组对屏幕内存直接进行写操作的程序。在彩色文本方式下对段 0xB800 进行写操作。但是，如果在保护方式下运行这段子例程，80386 将标识这是对程序代码和数据段进行写操作。程序运行时由于对保护模式的内存进行了扰乱，终端就表现为胡乱滚屏。另一方面，如果是在 BIOS 屏幕管理方式下，将会出现死机。

幸运的是，在本章后面要介绍的 DOS 扩展使用了内存区域“硬线”，在此称之为局部描述符表（LDT）。Phar Lap 的 DOS 扩展管理程序为屏幕内存建立了 LDT 表。在第五章中，我们还将讲到如何去访问屏幕 LDT，以实现直接视频内存访问。直接视频内存访问就是对屏幕处理程序进行快速的、专业化的处理。

本章将介绍 C 保护模式程序设计工具的三种基本概念。包括：

- 32 位 C 编译器；
- 80386 宏汇编器；
- DOS 扩展管理程序

同时，我们还要介绍

- 如何设计保护模式的字典

本书对那些只想使用 C 的库函数来编程，并用标准保护模式 C 编译器来编译程序的读者来说没有多大的实用价值。但在 C 编程中，常常会涉及到屏幕、窗口、键盘、鼠标、图形等专业化库文件。所以，这些人员读一读本书就会大有好处了。

8086 汇编程序可在 80386 下运行。因为 80386 芯片能把 8086 汇编源代码转换成 80386 汇编源代码。当然，这并不是说两者之间没有差别，8086 汇编能替代 80386，它们之间还是有差别的。

下面，我们先讲讲 32 位保护模式下的 C 编译器。

1.1 32 位的 80386 C 编译器

32 位 80386 编译器优越于其他编译器，其原因在于它的内存处理模式采用了 32 位保护模式编译器。

在 16 位编译器中，只存在代码和数据模式之分。而在 32 位 C 程序编译中，则要区分以下不同的概念：

- 小型和大型的代码模式
- 小型和大型的数据模式

小型代码模式是用 32 位近程调用相应的 CS(程序代码段的起始地址)，也就是说存放代码的容量为 4Gb（千 Mb)。

大型代码模式采用的是 48 位远程调用，这样容纳代码量超过 4Gb。

数据模式类似于代码模式。小型数据模式的数据量不超过 4Gb，而大型数据模式的数据量超过 4Gb。

由以上两种代码模式和数据模式自由组合成以下五种内存模式。

内存模式	代码模式	数据模式
微型	小	小
小型	小	小
中等	大	小
紧凑型	小	大
巨型	大	大

微型内存模式和小型内存模式的区别在于，微型内存模式中代码段和数据段之和不超过 4Gb，而在小型内存模式中，代码段和数据段各小于等于 4Gb。

由于我们实验用的 80386 机内存只有 2Mb 扩展内存，因而下面以微型模式为例。在叙述之前，提醒大家记住：4Gb = 4 000Mb = 4 000 000 000 字节。

比较 16 位和 32 位的几组数据：

数据类型	8086	80386
char	8 位	8 位
short int	16 位	16 位
int	16 位	32 位
long int	32 位	32 位

16 位编译器定义的 int 型数是 16 位，而 32 位编译器定义的 int 型数是 32 位。这样要把 16 位编译的 C 程序转移到 32 位编译器中，将出现好些意想不到的怪现象。例如，

在程序中定义 RECT 结构:

```
typedef  struct {
    int x1;
    int y1;
    int x2;
    int y2;
} RECT;
```

在 C 源代码中，定义变量 R2 为 RECT 结构

```
RECT R1;
```

然后在 C 库中编制一个汇编程序，用来定义一个指针，并让指针指向 RECT 结构。程序中寄存器 si 存储 RECT 结构的地址偏移量。先读取 RECT 结构中的 X1 值，然后 si 加 2，可获得 y1 值，以此类推，读取 RECT 结构中的其它值。汇编程序码如下:

```
holdx1    DW ?
holdy1    DW ?
    ...
    mov   AX, [si]                          ; 读取R1→x1值
    mov   [holdx1], AX                      ; 给内存变量holdx1赋值
    inc   si                                ; si加2
    inc   si
    mov   AX, [si]                          ; 读取R1→y1值
    mov   [holdy1], AX                      ; 给内存变量holdy1赋值
    ...
```

这一个汇编小程序能在 16 位整数的 16 位编译器上正常运行。可您知道为什么不能在 32 位的 C 编译器上运行吗？这是由于在程序中设定的整数是 16 位整数。若用 32 位整数（4 字节）来替换 16 位整数（2 字节），程序能在 32 位编译器下正常运行。

下面介绍把上述小程序转换为能在 32 位 C 编译上运行的方法。

方法一，不改变汇编小程序，重新定义 RECT 结构，如下:

```
typedef struct {
    short int x1;
    short int y1;
    short int x2;
    short int y2;
} RECT;
```

定义每个变量为短整数变量（2 字节），这样汇编程序就能正确地运行。

另一种方法是，保持 RECT 结构不变，改变汇编小程序。如下所示:

```
holdx1    DW ?
holdy1    DW ?
    ...
    mov   AX, [Esi]                    ; 获取R1→x1
    mov   [holdx1], AX                 ; 赋值给内存变量
    inc   Esi                          ; Esi加4
    inc   Esi
    inc   Esi
    mov   AX, [Esi]                    ; 读取R1→y1值
    mov   [holdy1], AX                 ; 赋值给内存变量
```

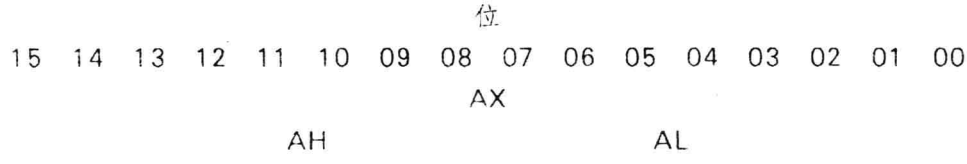
在汇编程序中，用 32 位 Esi 寄存器替代 16 位 si 寄存器。Esi 寄存器加 4 对应于 si 寄存器加 2。这样，Esi 寄存器加 4 刚好为一个整数，即 32 位（4 字节）整数。

一般来说，用 32 位 386C 编译器写的程序可获得 4Gb 字节以下的变量。在 8086 16 位实模式下运行的 C 程序成功地转换成在 80386 保护模式下运行的 32 位源程序。只是在转换过程中，要注意一下细节。

1.2 32 位 80386 汇编

本节不想对 80386 汇编程序作更深的探讨，主要是介绍 80386 中几个主要的寄存器以及如何利用寄存器来更改 8086 汇编子例程函数。

在 8086 芯片中，AX 为 16 位寄存器，两个 8 位寄存器 AH 和 AL。如下所示，为 16 位的 AX 寄存器：



80386 的 32 位 EAX 寄存器是 8086 的 16 位 AX 寄存器的扩展。对 32 位 EAX 寄存器进行操作，也就是对 AX、AH 及 AL 进行操作。如：

```
inc   EAX        ; 合法的32位累加器
inc   AX         ; 合法的16位累加器
inc   AH         ; 合法的8位累加器
inc   AL         ; 合法的8位累加器
```

4 个常用的 32 位寄存器有：

EAX（EAX 32 位，AX 16 位，AH 8 位，AL 8 位）

EBX（EBX 32 位，BX 16 位，AH 8 位，AL 8 位）

ECX（ECX 32位，CX 16位，CH 8位，CL 8位）

EDX（EDX 32位，DX 16位，DH 8位，DL 8位）

在386汇编中，地址偏移量为32位，而在8086中，是16位。因而，在386中DI，SI，BP和SP寄存器相应地扩展为32位。从而出现以下四个32位的新寄存器：

EDI（EDI 32位，DI 16位）

ESI（ESI 32位，SI 16位）

EBP（EBP 32位，BP 16位）

ESP（ESP 32位，SP 16位）

在80386中，保留了8086芯片的16位段寄存器，同时新增了两个段寄存器。下面列出80386的所有段寄存器：

CS （旧）

DS （旧）

ES （旧）

FS （新）

GS （新）

SS （旧）

两个新增段寄存器FS和GS，具有与旧寄存器ES一样的功能，没有其他特别功能。

最后，讲讲IP和FLAGS的扩展寄存器EIP和EFLAGS：

EIP（EIP 32位，IP 16位）

EFLAGS（EFLAGS 32位，FLAGS 16位）

80386微处理器支持8086微处理器中的所有旧的寻址方式，除此之外，同时还支持新增寻址方式；本书中有关80386汇编部分的着重点在于重组8086汇编子程序，以便于能被80386 32位编译C函数调用，同时在80386汇编源程序中也不使用新的寻址方式。

后面我们要专门讲解把8086汇编转换成80386汇编程序的方法。在讲方法之前，我们首先探讨一下两个问题。

第一，80386中如何处理循环。为更好地解决这一问题，我们先看看8086是如何解决这一问题的：

```
    ...
    mov  cx, 10
rep  movsb
    ...
```

程序先设寄存器CX循环次数，然后重复运行10次movsb。现在请读者回答：为什么这一段程序不能在80386上运行呢?。在回答这一问题之前，我们还是先来看看80386程序码：

```
    ...
    mov Ecx, 10
```

```
    rep  movsb
    ...
```

80386 中把 ECX 作为循环次数寄存器。若只给 CX 寄存器赋值，则 ECX 寄存器的高位没有被赋值，循环程序无法正确执行。只有把整个 ECX 寄存器作为计数器，才不致于出错。

其二，与使用 BIOS 或 DOS 有关。例如，对于 16 位汇编芯片：

```
    ...
hello    DB                          'Hello chunk! ', 0Ah, 0Dh, '$'
    ...
    lea  DX, hello                   ; 把变量hello的地址偏移量装入DX寄存器
    mov  AH, 9                       ; 用DOS打印字符串
    int  21h
```

而相对应的 80386 汇编程序为：

```
    ...
hello    DB                          'Hello chunk! ', 0Ah, 0Dh, '$'
    ...
    lea  EDX, hello                  ; 把变量hello的地址偏移量装入EDX寄存器
    mov  AH, 9                       ; 用DOS打印字符串
    int  21h
    ...
```

总而言之，要记住在编制循环程序时，使用 ECX 寄存器作计数器，而不是 CX 寄存器。同样，在 8036 汇编中，所有标记地址偏移量的寄存器必须是 32 位的，(如 EDX)，而不能用 16 位 DX 寄存器。

1.3 DOS 扩展管理程序

DOS 不是保护模式的操作系统，DOS 扩展管理程序是为了适应 80386 保护模式操作环境而扩充了 DOS 功能。也就是说，借助 DOS 扩展管理程序，很容易完成 C 库函数从普通环境到保护模式操作环境的转换，并生成新的操作系统，如 OS／2。

程序员可能会担心"DOS 扩展管理程序是否不支持新生成的操作系统"。事实上，所有 80386 机器都能访问 DOS，目前还没有机器不能使用新生成的保护模式操作系统。这儿要强调的是，保护模式下的程序不仅要考虑到怎样写程序，而且要考虑到用户的使用环境。DOS 扩展管理程序在标准 80386 上运行保护模式程序，至少需要 1M 的扩展内存。

实质上，DOS 扩展不停地调用 BIOS 或 DOS 系统中断，并通过 80386 实模式运行它。由于不停地把保护模式转换成实模式，因而要减慢程序的执行速度。通过实验，我们

可以得出这样的结论：386 保护模式的 Paradox 版本比 8086 实模式版本速度慢。若以这一结论为前提，那么，32 位指令程序在保护模式下运行会慢于在 8086 实模式下运行的同一版本的 16 位指令程序。

在理论上，通过 DOS 扩展管理程序来调用 BIOS 和 DOS 过程，应考虑到会减慢程序执行速度。

利用 DOS 扩展管理程序的功能，有两种方法可以加快程序运行速度。方法一，从命令行调用 DOS 扩展管理程序。本书中的所有程序例子，都是在 Phar Lap 的命令行版本 DOS 扩展管理程序下运行的。在第二章中，我们将告诉大家怎样编写两个版本的"Hello chuck!"程序，汇编版本和 C 版本都需要 DOS 扩展管理程序才能运行。

编译并连接 C 版本"Hello chuck!"程序后，在硬盘上生成名为 HELL0.EXP 的文件。保护模式的可执行文件带有.EXP 扩展。用 Phar Lap 的 Dos 扩展管理程序，只需键入如下内容并按回车键即可执行 HELL0.EXP 文件

```
RUN386 HELLO
```

这种方法要求用户先执行 DOS 扩展管理程序。

下面介绍第二种方法，这种方法比较完善。先生成.EXP 文件，再与 DOS 扩展管理程序连接起来，生成 DOS 环境下的 EXE 文件。当执行.EXE 文件时，80386 自动转换到保护模式下。例如，若使用 Phar Lap 的连接工具，在编制好软件后，就不用担心用户是否知道 DOS 扩展管理程序和 80386 保护模式的内容了，而只需执行.EXE 文件即可。否则，在编制软件出售时，还需要考虑附上保护模式扩展管理程序。

DOS 扩展管理程序，是连接应用程序和 DOS 的交换界面，它支持并访问大多数 BIOS 和 DOS 系统调用过程。注意这儿我们说的是"大多数"，但因种类繁多，无法一一告诉读者，哪些 BIOS 和 DOS 系统调用一定能支持，哪些可能支持。

在本章的前面部分，我们曾提到过，DOS 扩展能利用 LDT（局部描述符表）来访问计算机内存。但是，利用直接内存视频访问 Phar Lap LDT 的保护模式程序不一定在 DOS 扩展管理程序上都能运行，所以只能使用 BIOS 屏幕读、写操作，这样就导致了程序功能的局限性。

因此，我们建议读者，先确定使用的是一种 DOS 扩展管理程序，还是多种扩展管理程序，然后再开始编制程序。本书介绍的例子，均选择 Phar Lap DOS 扩展管理程序，程序在 Phar Lap DOS 扩展支持下运行。

总而言之，DOS 扩展管理程序丰富并完善了 DOS 操作系统的功能。利用 DOS 扩展管理程序，可以实现大多数 BIOS 和 DOS 系统调用。遗憾的是，并非所有的 DOS 扩展管理程序都能运行保护模式程序。

1.4 设计保护模式程序库

由于应用系统的开发需要各种 C 语言工具，因此，需要建立保护模式的 C 库，并不断开发、完善这些 C 语言的库函数。

首先，考虑编制库源代码函数，使之与实模式 C 兼容。这便于实现把 8086 的 C 语言

程序移植到 32 位的 80386 中。

函数库的范畴一旦确定下来，接下来便开始选择工具。

由于 Phar Lap 工具的速度快、使用方便，因而我们选择使用 Phar Lap 的 DOS 扩展管理程序。

本章讲述的内容可以概括如下:

1. 确定库内容;
2. 选择 C 编译器;
3. 选择宏汇编器;
4. 选择 DOS 扩展管理程序。

下一章，将向大家介绍如何编写保护模式的程序，以及如何运行它。

第二章　编写和运行保护模式程序

本章我们将向读者介绍如何编写和运行保护模式程序，以 Watcom 的 386C 8.0 版本为例。由于 Watcom 编译器功能强，灵活性好，所以我们使用这种编译器来构建保护模式的 C 语言函数库，同时我们还要阐述一些相关的编译器特征。

同所有的编译器一样，Watcom 也提供了编译和连接程序。这一公用程序名为 WCL386，它编译由合法命令集组成的程序，生成目标文件，然后连接它们的目标文件。读者能读到的大部分资料都是先给出编译-连接命令，然后再进行讲解。本章我们将说明如何进行编译、如何实现函数调用。

本章的第二节将讲述有关目标程序如何反汇编成各种参数调用汇编子过程的内容。

2.1　用 C 语言编写保护模式程序

在安装完编译程序后，就可以编译并连接程序了。程序 2-1 列出了程序 HELL01.C 的完整清单，下面我们先说明一下它的使用方法。

运行该程序时，屏幕显示

Hello chick!

然后回到 DOS 状态。我们用 TYPE 命令可以看到如程序 2-1 所示的程序清单。

程序 2-1　HELL01.C 源程序

```
//////////////////////////////////
//
// HELLD1.C
//
// Description:
// Prints "Wello Chuck!" to the screen
//
//////////////////////////////////
// include files here

#include <stdio.h>

// global date

char hello[]={"Hello Chuck!"};                    // String printed to the screen
```

```
// function declarations

void main(void);
void show_mess(char *);

void
show_mess(char * mess)
{
puts(mess);
}

void
main()
{
show_mess(hello);
}

//////////////////////////////////////
```

HELL01.C 程序非常简单。程序中有两个函数，main（）和 show_mess(…)，main（）函数为主程序，show_mess(…) 被主程序调用，用来打印字符串指针所指的字符串。

编译和连接保护模式程序，键入

WCL386 HELL01.C / 3s

并且按回车键。/ 3s 开关告诉编译器使用的栈参数。参数的传递是 Microsoft C，Turbo C 及目前市场使用的其他编译程序所常用的。用 WCL386 编译连接 HELL01.C 文件就在硬盘上产生 HELL0.EXP 文件。用 Phar Lop Dos 扩展运行 HELL01.EXP，键入

RUN386 HELL01

并按回车键。正常情况下，屏幕将显示：

Hello Chuck!

文件 HELL01.EXP 有 6 748 字节。

Watcom 提供了一个实用工具程序 WDISASM，它用于反汇编目标文件。

举例反汇编 HELL01.OBJ 文件，键入

WDISASM HELL01.OBJ > HELL01.LST

并按回车键，WDISASM 程序工具反汇编 HELL01.OBJ 文件，同时把反汇编结果列在 HELL01.OBJ 文件中。程序 2-2 列出在反汇编 HELLO1.OBJ 的过程中使用参数堆栈的方法。

分析程序 2-2，可以发现好些有趣东西。现在来看看调用 show_mess 函数一节的汇编码。