# Windows 环境下对内存进行跨段访问

金　一　(北京工业大学)

GXT 医学图象处理系统是一个通用的计算机图象处理系统.它是以 Borland C++3.1 作为开发工具,在 Windows 环境下进行开发的。由于图象在内存中所占的存储空间很大,我们不可能在一个段内完全访问到所有的数据,因此,当我们对其进行处理时,必须在内存中对图象进行跨段访问,以此得到正确的数据。这个问题在 DOS 环境下不难解决。但在 Windows 环境下有其特别之处。本文就是对此问题的一些粗浅认识。

图象的数据量一般都很大,如何在内存中开辟一个大的缓冲区,并且能对数据进行跨段访问是一个很重要的问题。

在 DOS 环境下 C 语言中的 far 和 Huge 指针可存取多于 64K 的数据,far 指针中计算机地址和操作上有几个潜在问题,这里不想多谈,而 Huge 指针是规格化的,可解决 far 指针存在的问题,但 Huge 指针处理时需调用专用子程序,所以 huge 指针处理时比 far 指针要慢.C++ 中也提供了 far／huge 这两种指针。由于 huge 指针处理时比较慢,所以在 Windows 开发 GXT 医学图象处理过程中,采用了 far 指针而没用 huge 指针存取数据,我们知道 far 指针不能自动跨段访问,所以在对数据进行跨段访问时,在程序中将段地址加 1,但却不能访问到正确数据。原因何在?为了说明这个问题,我们首先谈谈 Windows 3.1 对系统内存的管理.Windows 对内存管理使编程者不必过多涉及对硬件的直接操作,为 Windows 的多任务管理提供了方便,它使驻留在内存中的程序互不干扰,并且提供虚拟内存。但同时由于编程者不知道 Windows 如何管理系统内存,给编程者带来了很多的麻烦。Intel×86 系列芯片支持 Windows 采用的操作模式有实模式、标准模式以及增强模式三种。而在开发 GXT 图象处理应用程序中,应用程序是在 386 增强模式下运行。386 增强模式具有保护模式的所有优点,并提供 4 倍于物理内存的虚拟内存。保护模式是指处理器的一种状态,当一个程序在保护模式下存储地址时,同样以逻辑地址表示,由段选择符和偏移量两部分组成,而段选择符与基地址并无直接关系,只是一个逻辑上识别段的索引而已,它指向段描述符表中的一个段描述符。段描述符才真正定义了逻辑段所对应的物理存储器地址。因此在保护模式下,从逻辑地址到物理地址的转换不是简单的移位和加操作,而是根据操作系统创建并维护的全局描述符表(GDT)和局部描述符表(LDT)对内存进行文章的。因此,编程者不能直接与内存打交道,而只能访问描述符表。一个描述符表包含了 8192 个(64K ÷ 8)8 个字节的描述符,称其为段描述符。组成描述符表的段描述符的索引称为段选择符。它标志段描述符,而真正的段地址就是段描述符的内容,但这个段地址对程序来说是不可见的,而且是经常变化的,只有 CPU 可以访问它。通过段描述符的索引(段选择符)可得到真正的段地址。当把一个段选择符的值装入段寄存器,段选择符指向的段描述符自动被装入段高速缓存器。此后,当一个程序引用一个内存位置时,CPU 将访问这个段内单元,并使用高速缓存器中相应的字段值,执行从逻辑地址到物理地址的高速变换。但实际上,段选择符的 16 位中只有 13 位用来做为描述符表的索引。另外三位是附加区域,它们寻址系统和保护模式的保护机构有重要作用。位 2 是一个指明哪个描述符表被使用的标志。即 GDT 和 LDT,设定程序使用的地址空间,其意思是 GDT 包含了分给系统的所有内存,而 LDT 则代表程序私有地址空间。位 0 和位 1 描述了段的请求特权级(RPL),由操作系统软件设置这些值,以四个特权级别(0～3)建立和实行内存保护体系。如图 1 所示:

由此知道,长指针的高字位中并非每位都是描述符表下标,而是从第四位起才是下标值。因此,不能直接将 1 加在高字位上,而应将 1 加在高字的第四位上。对此我们只需用 Windows 3.1 中提供的宏 MAKELONG

#define MAKELONG(low,high)\
((LONG)(((WORD)(LOW)) (((DWORD)((WORD (high)))<<16)))

使长指针的高位字乘以 8(相当于左移 3 位)加在高字位上,这样就解决长指针的跨段问题,访问到正确数据。

下面给出一个在 Windows 下进行跨段访问数据的实例 CMHyperbolic 是类 GXTChild 中的一个成员函数,它的功能是进行直方图立方根变化,以达到图象增强的效果。

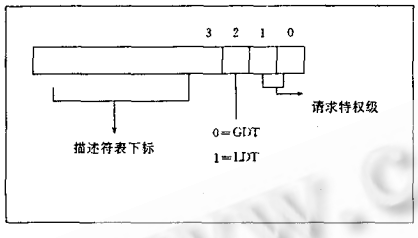


图 1

源程序清单如下:

```
/*
* * * * * * * * * * * * * * * * * * * * * * * * * * * *
                    直方图对数变化
* * * * * * * * * * * * * * * * * * * * * * * * * * * *

void GxtChild::CMHyperbolic(TMessage&)
{
  HBITMAP hBitl,hBit2;
  long HelpTooll,HelpTool2;
  LPSTR lpBit1,lpBit2;
  long int g2[256]; / / c[256],gl[256];
  BYTE b2[256];
  DWORD fFreq;
  DWORD xreso,yreso,wI,wJ;
  DWORD BitCount,item;
  int e,m,gmin,gmax;
  double em,dm,wm,am,bm;
  BITMAP Bitmap;
  HCURSOR hSaveCursor;
  HCURSOR hHourGlass;

  GetObject(hBitmap,sizeof(BITMAP),&Bitmap);
  yreso = Bitmap,bmHeight; / / 图象的高度;
  xreso = Bitmap,bmWidth; / / 图象的宽度;
  BitCount = (DWORD)xreso * (DWORD)yreso;
    hBit1 = (HBITMAP)GlobalAlloc(GMEM MOVEABLE |
GMEM_ZEROINIT,
    BitCount); / / 在内存中为原图象开一个空间;
     if(!hBit1)
      MessageBox(NULL,"cann,t hBitl","error",MB_OK);
    hBit2 = (HBITMAP)GlobalAlloc(GMEM_MOVEABLE
GMEM_ZEROINIT,BitCount);   / / 为处理完的图象开一个空间;
     if(!hBit2)
      MessageBox(NULL,"cann't hBit2","error",MB_OK);
    lpBit1 = (LPSTR)Globailock(hBit1);
    if(!lpBit1)
      MessageBox(NULL,"cann't lock lpBit1","error",
          MB_OK);
    GetBitmapBits(hBitmap,BitCount,lpBit1);
          / / 将图象信息读入内存;

    lpBit2 = (LPSTR)Globailock((HGLOBAL)hBit2);
     if(!lpBit2)
      MessageBox(NULL,"can't lock lpBit","error",MB_OK);
    HelpTool1 = MAKELONG(LOWORD(ipBit1),HIWORD
          (lpBit1));
          / / 将图象句柄的长指针型转为长整型;
    HelpToo12 = MAKELONG(LOWORD(ipBit2),HIWORD
          (lpBit2));
    hHourGlass = loadCursor(NULL,IDC WAIT);
    SetCapture(HWindow);
    hSaveCursor = SetCursor(hHourGlass);

    for (wI = 0; wI < 256; wI++) {
      g2[wI] = 0.0;
     }
    e = 0;
     gmin = 255;gmax = 0;
     for (wI = 0; wI < yreso; wI++) {;
      for (wJ = 0;wJ < xreso;wJ++) {
        item = MAKELONG(LOWORD(wI * xreso+wJ),
       HIWORD(wI * xreso+wJ) * 81); / / 确定图象数据位置;
       e = (BYTE) * (LPSTR) (HelpTool1+item); / / 获取图
          象数据;
        g2[e] = g2[e]+1;
        if (e < gmin) gmin = e;
        if (e > gmax) gmax = e;

      }
     }
      fFreq = BitCount;  / * sum of input image's elements
* /
/ *
                进行图象数据处理
   for(e = gmin;e < gmax;e++) g2[e+1]+ = g2[e];
   wm = (double)1 / (double)3;
```

```
dm = pow((double)gmax,wm);
am = pow((double)gmin,wm);
for(m = gmin;m < = gmax;m++) {
    em = (double)(g2[m] * (dm-am)) / (double)fFreq+am;
    bm = pow(em,(double)3);
    b2[m] = (int)bm;
    }
for (wI = 0;wI < yreso;wI++) {
    for (wJ = 0;wJ < xreso;wJ++) {
        item = (DWORD)(MAKELONG(LOWORD(wI
            * xreso+wJ),
            HIWORD(wI * xreso+wJ) * 81));
        e = (BYTE) * (LPSTR)(HelpTool1+item);
        if(b2[e]> 220) b2[e]= 220;
        if(b2[e]< 10);
        * (LPSTR)(HelpToo12+item) = b2[e];
    }
  }
SetBitmapBits(hBitmap,BitCount,lpBit2);//存入处理后的图
        象信息;
GlobalUnlock(hBit1);
GlobalUnlock(hBit2);
GlobalFree(hBit1);
GlobalFree(hBit2);
SetCursor(hSaveCursor);
ReleaseCaptureO;
AdjustScrollerO;
```