DOS 及 Windows 下大块内存管理

李涛　（北京理工大学　100081）

在编程过程中，特别是在图象处理中，我们常常遇到需要显示和处理大幅图象（大于 64K）的情况下，这时简单的应用指针加减并不能正确完成此工作。下面分别在 DOS 和 Windows 下讨论几种处理方法。

1. DOS 下的几种处理方法

我们知道，所谓逻辑地址就是程序员在编程中所用的地址，而物理地址则是处理器向物理内存读写所用的地址。逻辑地址包括两部分，即段标志符和偏移量。在 DOS 下，逻辑地址和物理地址直接对应，所以可以通过直接修改段地址来进行大块内存管理。

(1)调整远指针。远指针包括一个段址和一个偏移值。大模式下的指针运算只影响偏移值，不影响段值。这意味着只简单的进行指针的加减并不能正确处理超过 64K 的连续内存区。可用下面的函数 farptr()来解决。

```
char far * farptr(char far * p, long l)
{
unsigned int seg, off;
set = FP-SEG(p);
off = FP-OFF(p);
seg + = off/16;
off + = (unsigned int) (l&OxooofL);
seg + = (1/16L);
p = MK-FP(seg. off);
return(p);
}
```

在函数 farptr()中，通过调整段地址和偏移量来完成指针的加减。下面给出一个从文件中读数据的例子来说明怎样用此函数。

```
:
:
fr = fopen(filename, "rb");
p = (unsigned char huge * ) farmalloc ((long) Image-Size);//ImageSize 为要读取的图象大小
lpl = (unsigned char huge * )farmalloc((long)32768);//工作内存块
num = ImageSize/32768;//要读写的次数
for(k = 0;k< num;k + + )
  {
  fseek(fr, (long)32768 * k, SEEK-SET;
  fread(lpl, 1, (long)32768, fr);
  memcpy( farptr ( lp, ( long ) k * ( long ) 32768 ), lpl, 32768);
  }
left = ImageSize - (long)32768 * num; //left 是剩余的数据
fseek(fr, (long)32768 * num, SEEK-SET);
fread(lpl, 1, left, fr);
memcpy(farptr(lp, (long)32768 * num), lpl, l, left);
farfree(lp);
farfree(lpl);
:
:
```

得到 lps 以后，就可以直接对它的数据进行处理了。

(2)强制类型转换。因为巨型指针在增减时能同时改变段地址和偏移量，这意味着在内存段的边缘具有连续性，可直接进行大块内存的处理。利用此特性，可以将指针直接转换为巨型指针进行处理，如下所示：

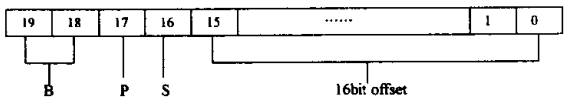
```
# define FARPTR( p, L) (unsigned char huge * ) (unsigned char huge * )p + (long)L)
```

定义了 FARPTR()以后，对于(1)中的例子，只需将 memcpy()语句作如下修改即可。

```
memcpy(FARPTR(lp, (long)k * 32768), lpl, 32768);
```

(3)修改 BPS 值。这种方法是针对特定的显示卡，通过修改视频缓冲区的段地址来完成对大于 64K 图象的显示。下面以 TVGA8900 卡为例来说明。对于其他的 Super VGA，改变相应的寄存器即可。

TVGA 有两种页模式，128K 模式和 64K 模式。由于 64K 模式编程简单且兼容性好，所以 TVGA8900 的缺省页模式为 64K 页模式。在此模式下，一般寻址如下图所示，其地址中的所有高位值由 I/O 端口 3C5.0EH 控制。

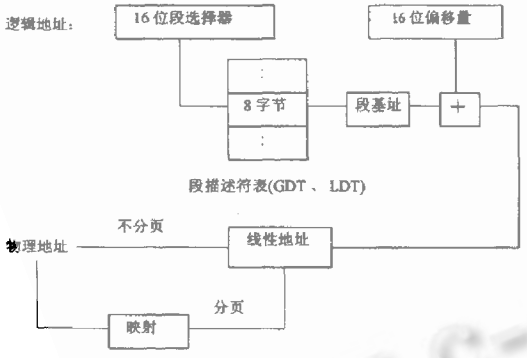


在显示大于 64K 的图象时，只需将 BPS 值进行相应变换即可。下面以 64K 页模式、800X600、256 色的视频模式(0X5E)为例说明。

```
        :
        :
for(j=0;j<depth;j++) //depth 图象的高度
  {
   fread(p,1,width,fp);//width 是图象的宽度
   offset=800L *j>>16; //offset 是高位值
   outp(0x3c4,0x0e);//设置 I/O 端口 0x3c5 的状态
   outp(0x3c5,0x02^offset);//改变高位地址
   -fmemcpt(MK-FP(0xa000,(long)j * 800L),p,
width);//0xa000 是视频缓冲区的段地址
  }
```

2. Windows 下处理方法

Windows 的内存管理与 DOS 的不同。在 Windows 下，与 DOS 下的段地址相对应的是段选择器(Segment Selector)。段选择器定义一个段信息描述表，称为段描述符表，其寻址过程如下所示：



16 位段选择器的高 13 位是段描述符索引，第 2 位决定是全局描述符(GDT)还是局部描述符(LDT)，最低两位是优先权级。在 16 位偏移量的情况下，每一段内存仍限制在 64K，此时 Windows 通过选择器堆砌来存取大于 64K 的内存块。当某个程序到达 64K 的内存块尾时，就切换到另一选择器中。由于选择器的底端三位不能够用来引用描述符，简单的向选择器加一，并不能解决问题，应该至少加 8 才行。例如，使第一个选择器的值为 097FH，来分配一个 200K 大小的内存，会产生如下的结果：

选择器	内存区域
097FH	0—(64K－1)
0987H	64K—(128K－1)
098FH	128K—(192K－1)
0997H	192K—(200K－1)

大多数的 Windows 编译程序已经知道这种选择器堆砌策略，所以能自动加载正确的选择器值。在 C 和 C++中，使用 huge 声明数组，就会自动引用选择器堆砌策略。需要注意的是：数值 8 是从-AHINCR 中获得的，而不是通过硬编码获得的，这一点很重要。-AHINCR 是预定义的一个 KERNEL 常量。下面给出一个从文件中读取大幅图象到—huge 指针的例子。

```
GLOBALHANDLE BitsHandle;
long Count;
long Start, ToAddr, Bits;
HFILE TheFile;
TheFile=-lopen(FileName,OF-READ);
BitsHandle=GlobalAlloc(GHND,(long)BitsByteSize); //
BitsByteSize 为图象大小
Start=0L;
Bits=(long) GlobalLock(BitsHandle);
Count=BitsByteSize-Start;
while(Count>0)
{
ToAddr=MAKELONG(LOWORD(Start),
    HIWORD(Bits)+(HIWORD(Start) * FP--OFF(-
ahIncr)));
      //FP-OFF(-ahIncr)得到选择器堆彻所需的常数
      //HIWORD(Start)得到堆彻选择器的个数
if(Count>0x4000)
  Count=0x4000;
-lread(TheFile,(LPSTR)ToAddr,(WORD)Count);
Start=Start+Count;
Count=BitsByteSize-Start;
}
GlobalUnlock(BitsHandle);
GlobalFree(BitsHandle);
-lclose(TheFile);
:
    :
```

在以后的程序中，可以直接锁住 BitsHandle，然后对其中的数据进行处理即可。 (来稿时间:1996 年 11 月)