浙江科技学院学报,第 17 卷第 3 期,2005 年 9 月
Journal of Zhejiang University of Science and Technology
Vol. 17 No. 3, Sep. 2005

基于内存映射文件的特大机械图纸快速处理技术

王　华
(浙江科技学院 自动化与电气工程学院,浙江 杭州 310023)

摘　要:在对位图图像进行处理时,为了追求效率,往往是将位图全部读进物理内存,然后在内存中进行处理,这种方法对于尺寸较小的位图图像比较理想,但是对于特大型的位图图像就显得无能为力了。针对一些位图格式的特大机械图纸的实时处理要求,本文提出了用面相对象编程工具 Delphi 实现基于内存映射文件的特大机械图纸快速处理,该方法无需将位图全部读进物理内存中,从而解决了物理内存不足而带来的缺憾,取得了令人满意的效果。

关键词:位图;Delphi;内存映射
中图分类号: TP311　　**文献标识码**: A　　**文章编号**: 1671-8798(2005)03-0179-05

Fast processing techniques of large mechanical bitmap based on memory mapping file

WANG Hua
(School of Automation and Electrical Engineering, Zhejiang University of Science and Technology, Hangzhou 310023, China)

Abstract: In order to achieve high efficiency when processing bitmap file, the common method is loading the whole bitmap file into physical memory, this method is suitable for processing small bitmap file, while is not always the case in processing large bitmap file. In view of the real time requirement of large bitmap processing, this paper brought forward this techniques of "memory mapping" together with Delphi to processing large mechanical bitmap, the method has solved the defection of limited physical memory and the speed is perfect.

Key words: bitmap; Delphi; memory mapping

目前,对于位图文件的处理普遍采用先将位图文件全部读进物理内存,然后在内存中再对点阵位图进行处理的方法。由于计算机物理内存的限制,对一些特大位图如机械图纸处理的实时性很不理想,甚至出现无法处理的情况,大型位图文件的输入、输出需要花费大量时间,因而,依靠有限的物理内存来处理特大位图是不现实的。由于 Win32 实现的是页式虚拟存储管理,内存映射文件是该内部已有的内存管理组件的一个扩充。在 Win32 中,内存映射文件可以使我们访问磁盘文件就如同正在访

收稿日期: 2005-04-20
作者简介: 王　华(1978—　),男,浙江淳安人,硕士,助教,主要从事数字图像测控系统、网络监控系统和 CAD 技术研究。

问内存文件一样，并且省去了对文件的输入、输出操作，它先是保留一段虚拟的地址空间，然后将磁盘文件提交给这段内存空间。内存映射文件实现调入和执行 EXE 文件以及 DLL 文件可以节省页交换文件的空间，通过操纵指向该内存区域的指针，就可以实现对映射文件快速的数据访问，这样不仅简化了对文件的访问，也不需要编写各种缓冲方案。本文重点探讨如何通过内存映射文件方式实现对信息量较大的位图文件的快速处理技术。

1 内存映射文件的使用

1.1 文件创建与打开

对磁盘文件进行操作必须要先获得该文件的句柄，在 Delphi 的 SysUtils. pas 单元中声明了两个函数如下：

Function Filecreate(const Filename: string): Integer;

Function FileOpen(const Filename: string; Mode: Word): Integer

Filecreate 函数创建一个文件名为 Filename 的新文件，如果函数调用成功则返回一个有效的文件句柄，否则返回 INVALID_HANDLE_VALUE 常量，FileOpen 函数用于以某种模式打开一个已经存在的文件，参数一是映射文件的名称和完全路径，参数二包括以下一些可取值：只读 fmopenread，只写 fmopenwrite，可读写 fmopenreadwrite，模式值还可以为“0”，此时为不能读也不能写。只有获得文件句柄后，才能获得文件映射对象。

1.2 创建文件映射对象

函数 CreateFileMapping()可以用来创建命名的或者是没有命名的内存映射文件对象[1]，该函数的声明如下：

Function CreateFileMapping(hFile: HANDLE,
lpFileMappingAttributes: LPSECURITY_ATTRIBUTES, flProtect: DWORD,
dwMaximumSizeHigh: DWORD,
dwMaximumSizeLow: DWORD,
lpName: LPCTSTR);

传递给 CreateFileMapping 函数的参数为系统创建文件映射对象提供必要的信息。参数 hFile 是先前调用的 Filecreate()或者 FileOpen()函数返回的文件句柄，参数 lpFileMappingAttributes 是一个 LPSECURITY_ATTRIBUTES 类型的指针，指向文件映射的安全属性，一般情况下，此参数为 Null，参数 flProtect 用来指定文件视图的保护类型，该值必须与打开文件以获取文件句柄时用的参数一样。参数 dwMaximumSizeHigh 和 dwMaximumSizeLow 将指定文件映射的大小。如果指定大小的内存映射文件不是使用系统页面文件，则磁盘上的物理文件将根据这一新的大小进行相应的调整以使两个值相等。另外一种处理方法是在映射磁盘上的文件时，将映射大小第四和第五个参数置为“0”，这样得到的内存映射文件与原始磁盘上的文件大小是一致的。当映射页面文件的某个区域时，必须要指定内存映射文件的大小参数 lpszMapName，可以给内存映射文件指定一个名字。如果想打开一个已存在的文件映射对象，该对象必须要命名。对该名字字符串的要求仅限于未被其他对象使用过的名字即可。如果函数 CreateFileMapping()调用成功，则返回文件映射对象的句柄；如果调用失败则返回“0”，此时，必须调用 GetLastError()函数得到失败原因。

1.3 映射文件的视图到进程的地址空间

获得内存映射文件对象的有效句柄后，该句柄即可用来在进程的地址空间内映射该文件的一个视图。在内存映射文件对象已存在的情况下，视图可以任意映射或取消映射。当一个文件的视图被映射时，系统将为此分配系统资源，在进程地址空间内，一个足够以覆盖文件视图的连续地址空间将被指定给此文件视图，尽管如此，内存的物理页面还是基于实际使用中的需求来分配。真正分配一个对应于内存映射文件视图页面的物理内存页面是在发生该页的缺页中断时进行的，在第一次读写内存页面中任一地址时自动完成的。使用 MapViewOfFile 函数来映射内存映射文件的一个视图，这个函数要求的第一个参数是内存映射文件对象的句柄。函数声明如下：

Function MapViewOfFile(HFileMappingObject: HANDLE,
DwDesiredAccess: DWORD,
DwFileOffsetHigh: DWORD,
DwFileOffsetLow: DWORD,
DwNumberOfBytesToMap DWORD);

参数 HfileMappingObject 是由 CreateFileMapping()函数返回的文件映射对象的句柄，参数 DwDesiredAccess 用于指定文件数据访问模式。参数 DwFileOffsetHigh 和 DwFileOffsetLow 分别用来指定内存映射

文件的 64 位偏移地址低 32 地址和高 32 位地址,这个偏移地址是从内存映射文件头位置到视图开始位置的距离。参数 dwNumberOfBytesToMap 指定视图的大小,若该参数置为"0",则偏移地址将被忽略,整个文件被映射成一个视图,该函数返还指向文件视图在进程地址空间中的起始地址的指针。

内存映射文件函数可以看作是虚拟内存管理函数的姐妹组函数。它与虚存管理函数一样,都将直接影响到进程的地址空间和物理内存页面,而且在管理文件视图上,除了存在于所有进程中的基本虚存管理之外没有其他的开销。这些函数只处理内存中的保留页面和进程中的确定地址区域。

1.4　撤销内存映射文件的视图映射

一个内存映射文件的视图被映射后,该视图可在完成对文件的处理后通过调用函数 UnmapViewOfFile()撤销其映射。函数声明如下:

Function unmapViewOfFile (lpBaseAddress: pointer):bool;

该函数的唯一参数 lpBaseAddress 必须指向映射区域的起始地址,也就是 MapViewOfFile()han 函数的返回值,如果在文件处理后没有撤销视图映射,则只能等到进程终止,系统才会释放映射区所占资源。

1.5　关闭文件映射和文件内核对象

由于函数 Filecreate 和 Fileopen 以及 CreateFileMapping 都将打开内核对象,必须通过 CloseHandle()函数来释放创建的映射文件的句柄。

在 Win32 中,内存映射文件函数可以看作是虚拟内存管理函数的姐妹组函数。它与虚存管理函数一样,都将直接影响到进程的地址空间和物理内存页面,而且在管理文件视图上,除了存在于所有进程中的基本虚存管理之外没有其他的开销,这些函数只处理内存中的保留页面和进程中的确定地址区域。

2　位图文件的结构与处理

2.1　位图文件的结构

位图的结构包括四个部分[2]:位图文件头(bitmap-file header)、位图信息头(bitmap-information header)、彩色表(color table)和定义位图的字节阵列,位图文件头包含有关于文件类型、文件大小、存放位置等信息,结构如下:

```
typedef struct tagBITMAPFILEHEADER{
WORD        bfType;
DWORD         bfSize;
WORD        bfReserved1;
WORD        bfReserved2;
DWORD         bfOffBits;
                    BITMAPFILEHEADER;
```

这个结构的长度是固定的,为 14 个字节。第二部分为位图信息头 BITMAPINFOHEADER,也是一个结构,其定义如下:

```
typedef struct tagBITMAPINFOHEADER{
DWORD              biSize;
LONG          biWidth;
LONG          biHeight;
WORD          biPlanes;
WORD          biBitCount;
DWORD         biCompression;
DWORD         biSizeImage;
LONG          biXPelsPerMeter;
LONG          biYPelsPerMeter;
DWORD             biClrUsed;
DWORD             biClrImportant;
}       BITMAPINFOHEADER;
```

这个结构的长度为 40 个字节,对于 256 色位图。它的结构中含有颜色索引表,即调色版。而对于真彩色位图则没有调色板信息,由三个字节构成一个像素点的实际位图数据区就可以代表所有的颜色信息,因而对于一幅真彩色位图从头文件到实际数据区域的地址偏移是 54 个字节,对于真彩色位图只需处理实际位图数据区的各个像素点,便达到了图像处理的目的。

2.2　特大位图的处理

对于一幅 12 000×12 000 的真彩色位图,大小约 432 M,即使物理内存有 512 M,处理该位图也是比较困难的,而用内存映射文件方式处理起来那就简单了,因为每个进程的虚拟内存范围是 4 G,故而有足够的空间,结合位图的结构进行实际的特大位图处理。

2.3　位图的取反

对于一幅真彩色位图,要对其进行取反操作只需对每个像素点所对应的 R,G 和 B 三个分量分别取反,核心代码如下:

```
procedure TForm1. BitmapInvert(var:Btmap-
FileName:String);
```

```
    var
      FFileHandle: THandle;
      FMapHandle: THandle;
      Pdata: pbyte;//内存映射文件头指针
    FFileSize: Integer;//文件大小
    begin
        FFileHandle: = FileOpen (FileName,
fmOpenReadWrite); //获得文件句柄
        FFileSize: =GetFileSize(FFileHandle,
nil); //获取物理文件大小
        FMapHandle: = CreateFileMapping
(FFileHandle, nil, PAGE _ READWRITE, 0,
FFileSize, nil); //获取映射文件句柄
        CloseHandle(FFileHandle);
        Pdata: = MapViewOfFile (FMapHan-
dle, FILE_MAP_ALL_ACCESS, 0, 0, FFile-
Size); //返回映射文件头指针
        CloseHandle(FMapHandle);
        try
          i:=0;
          inc(integer(FData), $ 35 ); //从位
图文件头到实际数据区偏移 54 个字节
          while i<FFileSize do //像素取反
操作
           begin
            FData^:=not FData^;
            inc(integer(FData) ,1);
            inc(i,1);
           end;
        finally
          UnmapViewOfFile(FData); //撤
消映射
    end;
```

2.4 位图的分块显示

大型位图的另外一个问题就是比较难于整体显示，即使能够整体显示，速度也不理想，而基于内存映射的特大位图的分块显示则解决了这个问题。在 Delphi 的 windows 单元中封装了一个快速显示位图的 API 函数 StretchDIBits()[3]，声明如下：

```
StretchDIBits(
HDC hdc,// 设备上下文句柄
int XDest,// 目标矩形的左上角 x 坐标
int YDest,//目标矩形的左上角 y 坐标
int nDestWidth,//目标矩形的宽
int nDestHeight,//目标矩形的高
int XSrc,//源矩形的左上角 x 坐标
int YSrc,//源矩形的左上 y 坐标
int nSrcWidth,//源矩形的宽
int nSrcHeight,//源矩形的高
CONST VOID * lpBits,//位图的首地址
CONST BITMAPINFO * lpBitsInfo,//位图
数据首地址
UINT iUsage,
DWORD dwRop//光栅操作
);
```

只需将位图的首地址和数据地址赋给函数，并且可以通过定义源矩形的位置以及宽和高来决定显示的位图的大小，由于内存映射文件方式返回的正是位图文件的首地址，然后通过首地址以及偏移地址计算出位图实际数据区地址，光栅值设置为 SRCCOPY 模式，自定义大小的位图就显示在指定的设备上了。核心代码如下：

```
procedure TForm1. Dispay ( var FName:
string);
    var
      FFileHeader: PBitmapFileHeader;// 指向
位图文件头结构体
      FInfoHeader: PBitmapInfoHeader; // 指
向位图信息头
      FInfo: PBitmapInfo;
      FPixelStart: pointer; // 指向位图实际数
据区
      FBitmapWidth: integer;
      FBitmapHeight: integer;
      FFileHandle: THandle; // 位图的文件
句柄
      FMapHandle: THandle; // 位图映射文件
句柄
      FData: pointer;//位图的首地址
    Begin
      FData: = MapViewOfFile (FMapHandle,
FILE_MAP_ALL_ACCESS, 0, 0, FFileSize);
      CloseHandle(FMapHandle);
      FFileHeader:= FData;//获取映射文件的
头指针
```

```
FInfoHeader:= pointer(integer(FData) + sizeof(TBitmapFileHeader));
FInfo:= pointer(FInfoHeader); //位图信息头指针
FPixelStart:= pointer(integer(FData) + FFileHeader^.bfOffBits); //实际数据区
FBitmapHeight:= FInfoHeader^.biHeight; //获取位图的实际大小值
FBitmapWidth:= FInfoHeader^.biWidth;
image1.Width:= FbitmapWidth div 2; //显示一半宽度
image1.Height:= FBitmapHeight;
StretchDIBits(image1.Canvas.Handle,0,0,FBitmapWidth,FBitmapHeight, image1.Width,0, itmapWidth, FBitmapHeight, FPixelStart, FInfo^, DIB_RGB_COLORS, SRCCOPY);
UnmapViewOfFile(FData);
end;
```

通过调节源矩形的位置和大小,可以快速地分块显示大型的位图文件。

3　快速处理的实验结果

对于硬件配置为 CPU P4 1.6 G,内存 256 M,32 M 显存的 PC 机,在处理大小为 10 000×10 000 的 24 bit 的真彩色位图,此时位图大小约为 300 M 结果对照如表 1 所示。

表 1　内存为 256 M 时全部加载到物理内存法与内存映射方法效果比较

操作类型	全部加载到物理内存法	内存映射文件法
加载位图	加载失败	只需获得位图在虚拟内存中的指针信息
位图取反	无法加载,故而无法处理	处理时间:0.5 s
显示位图	无法加载,故而无法显示	只需动态加载定制窗口大小位图,速度很快

原因分析:对于物理内存只有 256 M 的 PC 机,处理一幅 300 M 大小的真彩色位图,如果将位图全部加载到物理内存是不可能的,故而传统的物理内存加载法会失败,也就谈不上处理。而内存映射文件则恰恰可以避免这个问题,加载的只是位图文件在虚拟内存中的指针信息,无需将所有的位图文件内容加载到内存中,由于是指针操作,所以速度很快,对于位图的显示,传统方式是将位图全部加载到内存,如果位图范围超过窗口的范围,则用浮动条来控制位图的显示,这样同样受限于有限的物理内存。而内存映射文件方式只是动态的加载用户指定的显示区域,每次读取的位图数据仅仅是窗口大小范围的数据,通过滚动条的控制来动态加载所需范围的数据,这种动态局部加载方式可以大大提高速度。

对于配置为 CPU P4 1.6 G,内存 512 M,32 M 显存的 PC 机,处理相同的位图时,结果对照如下(表 2)。

表 2　内存为 512 M 时全部加载到物理内存法与内存映射方法的效果比较

操作类型	全部加载到物理内存法	内存映射文件法
加载位图	加载成功,需要时间 1.5 s	只需获得位图在虚拟内存中的指针信息
位图取反	需要时间在 1.2～1.5 s 之间	处理时间 0.3 s
显示位图	拖动滚动条,图像显示不流畅	拖动滚动条,图像是流畅

原因分析:加大物理内存到 512 M 以后,对于 300 M 的位图虽然可以全部加载到物理内存中,但是由于系统本身需要的内存开销,所以加载的速度并不是很快,对位图进行处理时,除去操作系统占用的内存,可供使用的物理内存也不充裕,位图取反的处理速度也不理想。在显示不同区域的位图时,由于整幅图都在物理内存中,位置的变化使得数据交换不是很流畅,图像显示也就不很流畅。然而,内存映射方式克服了以上一些缺憾,巧妙地获取虚拟内存中位图文件指针信息,对指针进行操作,只读取需要显示部分的位图信息,大大提高了处理速度。

4　结束语

基于内存映射文件对特大位图文件的处理,可以返回位图文件在虚拟内存中的首地址,通过指针访问模式极大地加快了位图文件的处理速度,克服了有限的物理内存处理大型位图文件的速度缺憾,由于可以很方便的返回位图的文件头指针和数据区首地址,因而位图的其他处理如亮度、对比度、滤波的操作都很简单,效果令人满意。

参考文献:

[1] Steve Teixeira, Xavier Pacheco. Delphi5 开发人员指南 [M]. 任旭钧译. 北京:机械工业出版社,2000.

[2] 吕凤军. 数字图像处理编程基础[M]. 北京:机械工业出版社,1998.

[3] 何清刚. 扫描工程图象的浏览圈阅与编辑技术研究[J]. 组合机床与自动化加工技术,2000,(1):25－28.