实验 6-1 Windows 系统存储管理

一、Windows 内存结构

背景知识

Windows XP 是 32 位的操作系统，它使计算机 CPU 可以用 32 位地址对 32 位内存块进行操作。内存中的每一个字节都可以用一个 32 位的指针来寻址。这样，最大的存储空间就是 232 字节或 4000 兆字节 (4GB) 。这样，在 Windows 下运行的每一个应用程序都认为能独占可能的 4GB 大小的空间

而另一方面，实际上没有几台机器的 RAM 能达到 4GB，更不必说让每个进程都独享 4GB 内存了。Windows 在幕后将虚拟内存 (virtual memory，VM) 地址映射到了各进程的物理内存地址上。而所谓物理内存是指计算机的 RAM 和由 Windows 分配到用户驱动器根目录上的换页文件。物理内存完全由系统管理。

实验目的

1） 通过实验了解 windowsXP 内存的使用，学习如何在应用程序中管理内存、体会 Windows 应用程序内存的简单性和自我防护能力。
2） 了解 windowsXP 的内存结构和虚拟内存的管理，进而了解进程堆和 windows 为使用内存而提供的一些扩展功能。

工具/准备工作

您需要做以下准备：
一台运行 Windows XP Professional 操作系统的计算机
计算机中需安装 Visual C++ 6.0 专业版或企业版

实验内容与步骤

Windows 提供了一个 API 即 GetSystemInfo() ，以便用户能检查系统中虚拟内存的一些特性。程序 5-1 显示了如何调用该函数以及显示系统中当前内存的参数。

步骤 1：登录进入 Windows XP Professional。

步骤 2：在“开始”菜单中单击“程序”-“Microsoft Visual Studio 6.0”–“Microsoft Visual C++ 6.0”命令，进入 Visual C++窗口。

步骤 3：在工具栏单击“打开”按钮，在“打开”对话框中找到并打开实验源程序 5-1.cpp。

程序 5-1：获取有关系统的内存设置的信息

```
// 工程 vmeminfo
#include <windows.h>
#include <iostream>
#include <shlwapi.h>
#include <iomanip>
#pragma comment(lib, "shlwapi.lib")

void main()
{
    // 首先，让我们获得系统信息
    SYSTEM_INFO si;
    :: ZeroMemory(&si, sizeof(si) ) ;
    :: GetSystemInfo(&si) ;
```

```
    // 使用外壳辅助程序对一些尺寸进行格式化
    TCHAR szPageSize[MAX_PATH];
    ::StrFormatByteSize(si.dwPageSize, szPageSize, MAX_PATH) ;

    DWORD dwMemSize = (DWORD)si.lpMaximumApplicationAddress -
        (DWORD) si.lpMinimumApplicationAddress;
    TCHAR szMemSize [MAX_PATH] ;
    :: StrFormatByteSize(dwMemSize, szMemSize, MAX_PATH) ;

    // 将内存信息显示出来
    std :: cout << "Virtual memory page size: " << szPageSize << std :: endl;

    std :: cout.fill ('0') ;
    std :: cout << "Minimum application address: 0x"
            << std :: hex << std :: setw(8)
            << (DWORD) si.lpMinimumApplicationAddress
            << std :: endl;
    std :: cout << "Maximum application address: 0x"
            << std :: hex << std :: setw(8)
            << (DWORD) si.lpMaximumApplicationAddress
            << std :: endl;

    std :: cout << "Total available virtual memory: "
            << szMemSize << std :: endl ;
}
```

步骤 4: 单击“Build”菜单中的“Compile 5-1.cpp”命令，并单击“是”按钮确认。系统对 4-1.cpp 进行编译。

步骤 5: 编译完成后，单击“Build”菜单中的“Build 5-1.exe”命令，建立 5-1.exe 可执行文件。

操作能否正常进行？如果不行，则可能的原因是什么？

_____

_____

步骤 6: 在工具栏单击“Execute Program” (执行程序) 按钮，执行 5-1.exe 程序。运行结果 (分行书写。如果运行不成功，则可能的原因是什么？) ：

1) 虚拟内存每页容量为：_____
2) 最小应用地址：_____
3) 最大应用地址为：_____
4) 当前可供应用程序使用的内存空间为：_____
5) 当前计算机的实际内存大小为：_____

阅读和分析程序 5-1，请回答问题：

1) 理论上每个 windows 应用程序可以独占的最大存储空间是：_____
2) 在程序 5-1 中，用于检索系统中虚拟内存特性的 API 函数是：_____

提示：可供应用程序使用的内存空间实际上已经减去了开头与结尾两个 64KB 的保护区。虚拟内存空间中的 64KB 保护区是防止编程错误的一种 Windows 方式。任何对内存中这一区域的访问（读、写、执行）都将引发一个错误陷井，从而导致错误并终止程序的执行。也就是说，假如用户有一个 NULL 指针（地址为 0），但仍试图在此之前很近的地址处使用另一个指针，这将因为试图从更低的保留区域读写数据，从而产生意外错误并终止程序的执行。

二、Windows XP 虚拟内存

背景知识

在 Windows XP 环境下，4GB 的虚拟地址空间被划分成两个部分：低端 2GB 提供给进程使用，高端 2GB 提供给系统使用。这意味着用户的应用程序代码，包括 DLL 以及进程使用的各种数据等，都装在用户进程地址空间内 (低端 2GB) 。用户过程的虚拟地址空间也被分成三部分：

1）虚拟内存的已调配区 (committed) ：具有备用的物理内存，根据该区域设定的访问权限，用户可以进行写、读或在其中执行程序等操作。

2）虚拟内存的保留区 (reserved) ：没有备用的物理内存，但有一定的访问权限。

3）虚拟内存的自由区 (free) ：不限定其用途，有相应的 PAGE_NOACCESS 权限。

与虚拟内存区相关的访问权限告知系统进程可在内存中进行何种类型的操作。例如，用户不能在只有 PAGE_READONLY 权限的区域上进行写操作或执行程序；也不能在只有 PAGE_EXECUTE 权限的区域里进行读、写操作。而具有 PAGE_ NOACCESS 权限的特殊区域，则意味着不允许进程对其地址进行任何操作。

在进程装入之前，整个虚拟内存的地址空间都被设置为只有 PAGE_NOACCESS 权限的自由区域。当系统装入进程代码和数据后，才将内存地址的空间标记为已调配区或保留区，并将诸如 EXECUTE、READWRITE 和 READONLY 的权限与这些区域相关联。

如表 3-2 所示，给出了 MEMORY_BASIC_INFORMAITON 的结构，此数据描述了进程虚拟内存空间中的一组虚拟内存页面的当前状态，期中 State 项表明这些区域是否为自由区、已调配区或保留区；Protect 项则包含了 windows 系统为这些区域添加了何种访问保护；type 项则表明这些区域是课执行图像、内存映射文件还是简单的私有内存。VirsualQueryEX() API 能让用户在指定的进程中，对虚拟内存地址的大小和属性进行检测。

Windows 还提供了一整套能使用户精确控制应用程序的虚拟地址空间的虚拟内存 API。一些用于虚拟内存操作及检测的 API 如表 3-2 所示。

表 3-1 MEMORY_BASIC_INFORMAITON 结构的成员

成员名称	目的
PVOID BaseAddress	虚拟内存区域开始处的指针
PVOID AllocationBase	如果这个特定的区域为子分配区的话，则为虚拟内存外面区域的指针；否则此值与 BaseAddress 相同
DWORD AllocationProtect	虚拟内存最初分配区域的保护属性。其可能值包括：PAGE_NOACCESS,PAGE_READONLY,PAGE_READWRITE 和 PAGE_EXECUTE_READ
DWORD RegionSize	虚拟内存区域的字节数
DWORD State	区域的当前分配状态。其可能值为 MEM_COMMIT,MEM_PREE 和 MEM_RESERVE
DWORD Protect	虚拟内存当前的保护属性。可能值与 AllocationProtect 成员的相同
DWORD Type	虚拟内存区域中出现的页面类型。可能值为 MEM_IMAGE,MEM_MAPPED 和 MEM_PRIVATE

表 3-2 虚拟内存的 API

API 名称	描述
VisualQueryEX()	通过填充 MEMORY_BASIC_INFORMATION 结构检测进程内虚拟内存的区域
VisualAlloc()	保留或调配进程的部分虚拟内存，设置分配和保护标志
VirsualFree()	释放或收回应用程序使用的部分虚拟地址

VirsualProtect()	改变虚拟内存区域保护规范
VirsualLock()	防止系统将虚拟内存区域通过系统交换到页面文件中
VirsualUnlock()	释放虚拟内存的锁定区域，必要时，允许系统将其交换到页面文件中

提供虚拟内存分配功能的是 VirtualAlloc() API。该 API 支持用户向系统要求新的虚拟内存或改变已分配内存的当前状态。用户若想通过 VirtualAlloc() 函数使用虚拟内存，可以采用两种方式通知系统：

1)简单地将内存内容保存在地址空间内

2)请求系统返回带有物理存储区 (RAM 的空间或换页文件) 的部分地址空间

用户可以用 flAllocation Type 参数 (commit 和 reserve) 来定义这些方式，用户可以通知 Windows 按只读、读写、不可读写、执行或特殊方式来处理新的虚拟内存。

与 VirtualAlloc() 函数对应的是 VirtualFree() 函数，其作用是释放虚拟内存中的已调配页或保留页。用户可利用 dwFree Type 参数将已调配页修改成保留页属性。

VirtualProtect() 是 VirtualAlloc() 的一个辅助函数，利用它可以改变虚拟内存区的保护规范。

实验目的

1）通过实验了解 Windows 2000 内存的使用，学习如何在应用程序中管理内存，体会 Windows 应用程序内存的简单性和自我防护能力。

2）学习检查虚拟内存空间或对其进行操作。

3）了解 Windows 2000 的内存结构和虚拟内存的管理，进而了解进程堆和 Windows 为使用内存而提供的一些扩展功能。

工具/准备工作

在开始本实验之前，请回顾教科书的相关内容。

您需要做以下准备：

1) 一台运行 Windows 2000 Professional 操作系统的计算机。

2) 计算机中需安装 Visual C++ 6.0 专业版或企业版。

实验内容与步骤

1．虚拟内存的检测

清单 5-2 所示的程序使用 VirtualQueryEX()函数来检查虚拟内存空间。

步骤 1：登录进入 Windows 2000 Professional。

步骤 2: 在“开始”菜单中单击“程序”-“Microsoft Visual Studio 6.0”–“Microsoft Visual C++ 6.0”命令，进入 Visual C++窗口。

步骤 3: 在工具栏单击“打开”按钮，在“打开”对话框中找到并打开实验源程序 5-2.cpp。

清单 5-2　检测进程的虚拟地址空间

```
// 工程 vmwalker
#include <windows.h>
#include <iostream>
#include <shlwapi.h>
#include <iomanip>
#pragma comment(lib, "Shlwapi.lib")

// 以可读方式对用户显示保护的辅助方法。
```

```
// 保护标记表示允许应用程序对内存进行访问的类型
// 以及操作系统强制访问的类型
inline bool TestSet(DWORD dwTarget, DWORD dwMask)
{
    return ((dwTarget &dwMask) == dwMask) ;
}
# define SHOWMASK(dwTarget, type) \
if (TestSet(dwTarget, PAGE_##type) ) \
    {std :: cout << ", " << #type; }
void ShowProtection(DWORD dwTarget)
{
    SHOWMASK(dwTarget, READONLY) ;
    SHOWMASK(dwTarget, GUARD) ;
    SHOWMASK(dwTarget, NOCACHE) ;
    SHOWMASK(dwTarget, READWRITE) ;
    SHOWMASK(dwTarget, WRITECOPY) ;
    SHOWMASK(dwTarget, EXECUTE) ;
    SHOWMASK(dwTarget, EXECUTE_READ) ;
    SHOWMASK(dwTarget, EXECUTE_READWRITE) ;
    SHOWMASK(dwTarget, EXECUTE_WRITECOPY) ;
    SHOWMASK(dwTarget, NOACCESS) ;
}

// 遍历整个虚拟内存并对用户显示其属性的工作程序的方法
void WalkVM(HANDLE hProcess)
{
    // 首先，获得系统信息
    SYSTEM_INFO si;
    :: ZeroMemory(&si, sizeof(si) ) ;
    :: GetSystemInfo(&si) ;

    // 分配要存放信息的缓冲区
    MEMORY_BASIC_INFORMATION mbi;
    :: ZeroMemory(&mbi, sizeof(mbi) ) ;

    // 循环整个应用程序地址空间
    LPCVOID pBlock = (LPVOID) si.lpMinimumApplicationAddress;
    while (pBlock < si.lpMaximumApplicationAddress)
    {
        // 获得下一个虚拟内存块的信息
        if (:: VirtualQueryEx(
            hProcess,                    // 相关的进程
            pBlock,                      // 开始位置
            &mbi,                        // 缓冲区
            sizeof(mbi))==sizeof(mbi) )  // 大小的确认
        {
            // 计算块的结尾及其大小
            LPCVOID pEnd = (PBYTE) pBlock + mbi.RegionSize;
            TCHAR szSize[MAX_PATH];
            :: StrFormatByteSize(mbi.RegionSize, szSize, MAX_PATH) ;

            // 显示块地址和大小
            std :: cout.fill ('0') ;
            std :: cout
                << std :: hex << std :: setw(8) << (DWORD) pBlock
                << "-"
```

```
                << std :: hex << std :: setw(8) << (DWORD) pEnd
                << (:: strlen(szSize)==7? " (" : " (") << szSize
                << ") " ;

            // 显示块的状态
            switch(mbi.State)
            {
                case MEM_COMMIT :
                    std :: cout << "Committed" ;
                    break;
                case MEM_FREE :
                    std :: cout << "Free" ;
                    break;
                case MEM_RESERVE :
                    std :: cout << "Reserved" ;
                    break;
            }

            // 显示保护
            if(mbi.Protect==0 && mbi.State!=MEM_FREE)
            {
                mbi.Protect=PAGE_READONLY;
            }
            ShowProtection(mbi.Protect);

            // 显示类型
            switch(mbi.Type){
                case MEM_IMAGE :
                    std :: cout << ", Image" ;
                    break;
                case MEM_MAPPED:
                    std :: cout << ", Mapped";
                    break;
                case MEM_PRIVATE :
                    std :: cout << ", Private" ;
                    break;
            }

            // 检验可执行的影像
            TCHAR szFilename [MAX_PATH] ;
            if (:: GetModuleFileName (
                (HMODULE) pBlock,          // 实际虚拟内存的模块句柄
                szFilename,                //完全指定的文件名称
                MAX_PATH)>0)               //实际使用的缓冲区大小
            {
                // 除去路径并显示
                :: PathStripPath(szFilename) ;
                std :: cout << ", Module: " << szFilename;
            }

            std :: cout << std :: endl;
            // 移动块指针以获得下一下个块
            pBlock = pEnd;
        }
    }
}
```

```
void main()
{
    // 遍历当前进程的虚拟内存
    ::WalkVM(::GetCurrentProcess());
}
```

清单 5-2 中显示一个 walkVM()函数开始于某个进程可访问的最低端虚拟地址处，并在其中显示各块虚拟内存的特性。虚拟内存中的块由 VirsualQueryEX()API 定义成连续快或具有相同状态（自由区，已调配区等）的内存，并分配以一组统一的保护标志（只读、可执行等）。

步骤 4: 单击“Build”菜单中的“Compile 5-2.cpp”命令，并单击“是”按钮确认。系统对 5-2.cpp 进行编译。

步骤 5: 编译完成后，单击“Build”菜单中的“Build 5-2.exe”命令，建立 5-2.exe 可执行文件。

操作能否正常进行？如果不行，则可能的原因是什么？

_____

_____

步骤 6: 在工具栏单击“Execute Program” (执行程序) 按钮，执行 5-2.exe 程序。

1）分析运行结果（如果运行不成功，则可能的原因是什么）

按 committed,reserved,free 等三种虚拟地址空间分别记录实验数据，其中“描述”是对该组数据的简单描述，例如，对下列一组数据：

00010000-00012000<8.00KB>Committed,READWRITE,Private 可描述为：具有 READWRITE 权限的已调配私有内存区。

将系统当前的自由区（Free）虚拟地址空间填入表 3-3 中。

表 3-3 实验记录

地址	大小	虚拟空间类型	访问权限	描述
		free		
		free		
		free		
		free		
		free		
		free		
		free		

将系统当前的已调配区（Committed）虚拟地址空间填入表 3-4 中。

表 3-4 实验记录

地址	大小	虚拟空间类型	访问权限	描述
		Committed		
		Committed		
		Committed		
		Committed		
		Committed		
		Committed		
		Committed		

将系统当前的保留区（Reserved）虚拟地址空间填入表 3-5 中。

表 3-5 实验记录

地址	大小	虚拟空间类型	访问权限	描述

		Reserved		
		Reserved		
		Reserved		
		Reserved		
		Reserved		
		Reserved		
		Reserved		

2)从上述输出结果，对照分析清单 5-2 的程序，请简单描述程序运行的流程：

_____
_____
_____
_____
_____
_____

2. 虚拟内存的分配与释放

能正确使用系统函数 GetMeoryStatus()和数据结构 MEMORY_STATUS 了解系统内存和虚拟存储空间使用情况，会使用 VirsualAlloc（）函数和 VirsualFree（）函数分配和释放虚拟内存空间。

```
// GetMemoryStatus.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "GetMemoryStatus.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

   void GetMemSta(void);
  //The one and only application object
  CWinApp theApp;
  using namespace std;

  int _tmain(int argc,TCHAR * argv[],TCHAR * envp[])
  {
     int nRetCode=0;
     LPVOID BaseAddr;
     char *str;

     GetMemSta();
     printf("Now Allocate 32M Virsual Memory and 2M Physical Memory\n\n");

    BaseAddr=::VirtualAlloc(NULL,1024*1024*32,MEM_RESERVE|MEM_COMMIT,PA
    GE_READWRITE);//分配虚拟内存
    if (BaseAddr==NULL) printf("Virsual Allocate Fail.\n");
    str=(char *)malloc(1024*1024*2);                    //分配内存
    GetMemSta();
```

```
    printf("Now Release 32M Virsual Memory and 2M Physical Memory\n\n");
    if (::VirtualFree(BaseAddr,0,MEM_RELEASE)==0)                    //释放虚拟内存
        printf("Release Allocate Fail.\n");
    free(str);                                                       //释放内存
    GetMemSta();
    return nRetCode;
}

void GetMemSta(void)
{
    MEMORYSTATUS MemInfo;
    GlobalMemoryStatus(&MemInfo);

    printf("Current Memory Status is :\n");
    printf("\t Total Physical Memory is %d MB\n",MemInfo.dwTotalPhys/(1024*1024));
    printf("\t Available Physical Memory is %d MB\n",MemInfo.dwAvailPhys/(1024*1024));
    printf("\t Total Page File is %d MB\n",MemInfo.dwTotalPageFile/(1024*1024));
    printf("\t Available Page File is %d MB\n",MemInfo.dwAvailPageFile/(1024*1024));
    printf("\t Total Virtual Memory is %d MB\n",MemInfo.dwTotalVirtual/(1024*1024));
    printf("\t Available Virsual memory is %d MB\n",MemInfo.dwAvailVirtual/(1024*1024));
    printf("\t Memory Load is %d %%\n\n",MemInfo.dwMemoryLoad);

}
```

步骤 1: 在 VC 6.0 环境下选择 Win32 Console Application 建立一个控制台工程文件，选择 An application that Supports MFC。

步骤 2: 编辑并编译完成后，单击“Build”菜单中的“Build GetMemoryStatus.exe”命令，建立 GetMemoryStatus.exe 可执行文件。

操作能否正常进行？如果不行，则可能的原因是什么？

_____
_____

步骤 3: 在工具栏单击“Execute Program”按钮，执行 GetMemoryStatus.cpp.exe 程序。

分析程序 GetMemoryStatus.cpp 的运行结果

1) 请描述运行结果 (如果运行不成功，则可能的原因是什么？) ：

_____
_____
_____
_____
_____

2) 根据运行输出结果，若要改变分配和回收的虚拟内存和物理内存的大小，要改变程序代码的语句，分别为：

_____
_____

3) 根据运行输出结果，对照分析 4-2 程序，可以看出程序运行的流程吗？请简单描述：

_____
_____
_____
_____
_____