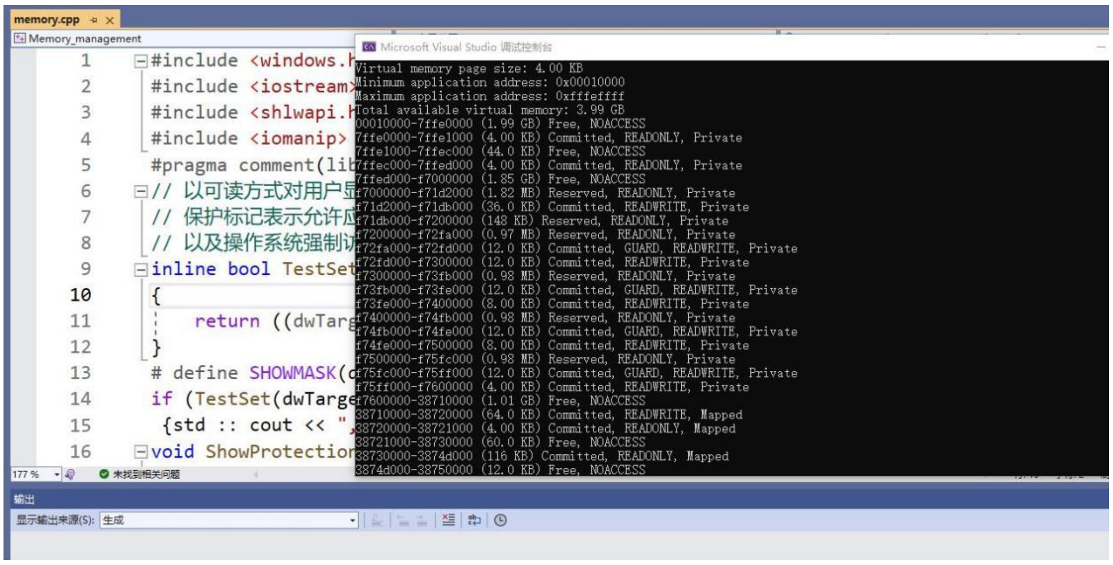
实验代码：

```
#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 ShowVirtualMemory()
{
    // 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;
}
void main()
{
    //显示虚拟内存的基本信息
    ShowVirtualMemory();
    // 遍历当前进程的虚拟内存
    ::WalkVM(::GetCurrentProcess());
}
```

实验结果：



memory.cpp

Memory_management

```
#include <windows.h
#include <iostream>
#include <shlwapi.h
#include <iomanip>
#pragma comment(lib
// 以可读方式对用户显
// 保护标记表示允许应
// 以及操作系统强制访
inline bool TestSet
{
    return ((dwTarg
}
# define SHOWMASK(d
if (TestSet(dwTarge
 {std :: cout << ",
void ShowProtection
```

177 %　未找到相关问题

输出

显示输出来源(S): 生成

Microsoft Visual Studio 调试控制台

```
Virtual memory page size: 4.00 KB
Minimum application address: 0x00010000
Maximum application address: 0xfffeffff
Total available virtual memory: 3.99 GB
00010000-7ffe0000 (1.99 GB) Free, NOACCESS
7ffe0000-7ffe1000 (4.00 KB) Committed, READONLY, Private
7ffe1000-7ffec000 (44.0 KB) Free, NOACCESS
7ffec000-7ffed000 (4.00 KB) Committed, READONLY, Private
7ffed000-f7000000 (1.85 GB) Free, NOACCESS
f7000000-f71d2000 (1.82 MB) Reserved, READONLY, Private
f71d2000-f71db000 (36.0 KB) Committed, READWRITE, Private
f71db000-f7200000 (148 KB) Reserved, READONLY, Private
f7200000-f72fa000 (0.97 MB) Reserved, READONLY, Private
f72fa000-f72fd000 (12.0 KB) Committed, GUARD, READWRITE, Private
f72fd000-f7300000 (12.0 KB) Committed, READWRITE, Private
f7300000-f73fb000 (0.98 MB) Reserved, READONLY, Private
f73fb000-f73fe000 (12.0 KB) Committed, GUARD, READWRITE, Private
f73fe000-f7400000 (8.00 KB) Committed, READWRITE, Private
f7400000-f74fb000 (0.98 MB) Reserved, READONLY, Private
f74fb000-f74fe000 (12.0 KB) Committed, GUARD, READWRITE, Private
f74fe000-f7500000 (8.00 KB) Committed, READWRITE, Private
f7500000-f75fc000 (0.98 MB) Reserved, READONLY, Private
f75fc000-f75ff000 (12.0 KB) Committed, GUARD, READWRITE, Private
f75ff000-f7600000 (4.00 KB) Committed, READWRITE, Private
f7600000-38710000 (1.01 GB) Free, NOACCESS
38710000-38720000 (64.0 KB) Committed, READWRITE, Mapped
38720000-38721000 (4.00 KB) Committed, READONLY, Mapped
38721000-38730000 (60.0 KB) Free, NOACCESS
38730000-3874d000 (116 KB) Committed, READONLY, Mapped
3874d000-38750000 (12.0 KB) Free, NOACCESS
```