### 实验3 内存监视

班级： 07152001 学号： 1120201198 姓名： 史桠彬

一、实验目的

独立设计并实现一个内存监视器，以加深对内存管理的理解。

二、实验内容

在Windows系统下设计实现一个内存监视器，使用该内存监视器：

1. 能够实时显示当前系统中内存的使用情况，包括系统地址空间的布局，物理内存的使用情况；
2. 能够实时显示某个进程的虚拟地址空间布局和工作集信息等。

三、程序设计与实现

**1. 实验环境**  
操作系统：Windows10  
开发环境：Visual Studio 2022

1. **设计思路**

① 获取系统信息[#](https://www.cnblogs.com/harrypotterjackson/p/12113387.html" \l "3238262322)

* SYSTEM\_INFO

typedef struct \_SYSTEM\_INFO {

union {

DWORD dwOemId;

struct {

WORD wProcessorArchitecture;

WORD wReserved;

} DUMMYSTRUCTNAME;

} DUMMYUNIONNAME;

DWORD dwPageSize;

LPVOID lpMinimumApplicationAddress;

LPVOID lpMaximumApplicationAddress;

DWORD\_PTR dwActiveProcessorMask;

DWORD dwNumberOfProcessors;

DWORD dwProcessorType;

DWORD dwAllocationGranularity;

WORD wProcessorLevel;

WORD wProcessorRevision;

} SYSTEM\_INFO, \*LPSYSTEM\_INFO;

* GetNativeSystemInfo

void GetNativeSystemInfo(

LPSYSTEM\_INFO lpSystemInfo

);

其中，LPSYSTEM\_INFO是指向SYSTEM\_INFO的指针。

* 信息输出

DWORD mem\_size = (DWORD\*)si.lpMaximumApplicationAddress - (DWORD\*)si.lpMinimumApplicationAddress;

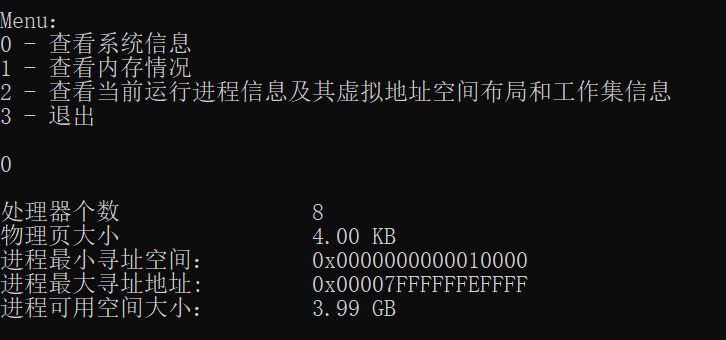
printDword(L"处理器个数", si.dwNumberOfProcessors);

printStrFormatByte(L"物理页大小", si.dwPageSize);

printAddress(L"进程最小寻址空间：0x", si.lpMinimumApplicationAddress);

printAddress(L"进程最大寻址地址: 0x", si.lpMaximumApplicationAddress);

printStrFormatByte(L"进程可用空间大小：", mem\_size);



② 获取物理内存信息[#](https://www.cnblogs.com/harrypotterjackson/p/12113387.html" \l "3003221194)

主要使用到的数据结构和函数有MEMORYSTATUSEX与GlobalMemoryStatusEx

及PERFORMANCE\_INFORMATION与GetPerformanceInfo。

* MEMORYSTATUSEX

typedef struct \_MEMORYSTATUSEX {

DWORD dwLength;

DWORD dwMemoryLoad;

DWORDLONG ullTotalPhys;

DWORDLONG ullAvailPhys;

DWORDLONG ullTotalPageFile;

DWORDLONG ullAvailPageFile;

DWORDLONG ullTotalVirtual;

DWORDLONG ullAvailVirtual;

DWORDLONG ullAvailExtendedVirtual;

} MEMORYSTATUSEX, \*LPMEMORYSTATUSEX;

在使用该数据结构之前，dwLength必须进行指定为 sizeof(MEMORYSTATUSEX)。

* GlobalMemoryStatusEx

BOOL GlobalMemoryStatusEx(

LPMEMORYSTATUSEX lpBuffer

);

其中，lpBuffer是指向MEMORYSTAUSEX的指针，用于保存信息。

* PERFORMANCE\_INFORMATION

typedef struct \_PERFORMANCE\_INFORMATION {

DWORD cb;

SIZE\_T CommitTotal;

SIZE\_T CommitLimit;

SIZE\_T CommitPeak;

SIZE\_T PhysicalTotal;

SIZE\_T PhysicalAvailable;

SIZE\_T SystemCache;

SIZE\_T KernelTotal;

SIZE\_T KernelPaged;

SIZE\_T KernelNonpaged;

SIZE\_T PageSize;

DWORD HandleCount;

DWORD ProcessCount;

DWORD ThreadCount;

} PERFORMANCE\_INFORMATION, \*PPERFORMANCE\_INFORMATION, PERFORMACE\_INFORMATION, \*PPERFORMACE\_INFORMATION;

* GetPerformanceInfo

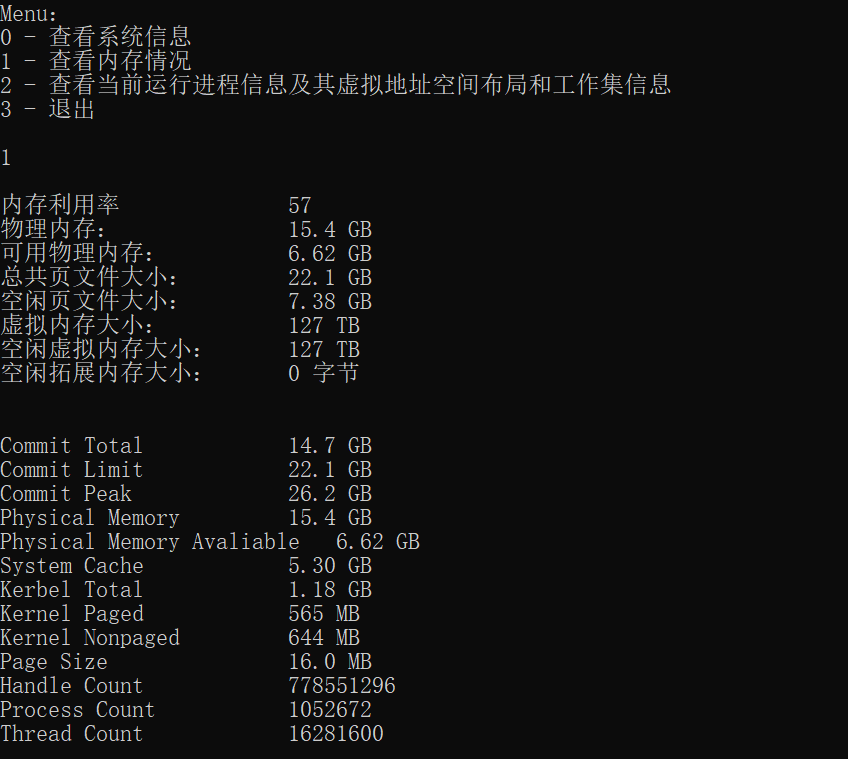
BOOL GetPerformanceInfo(

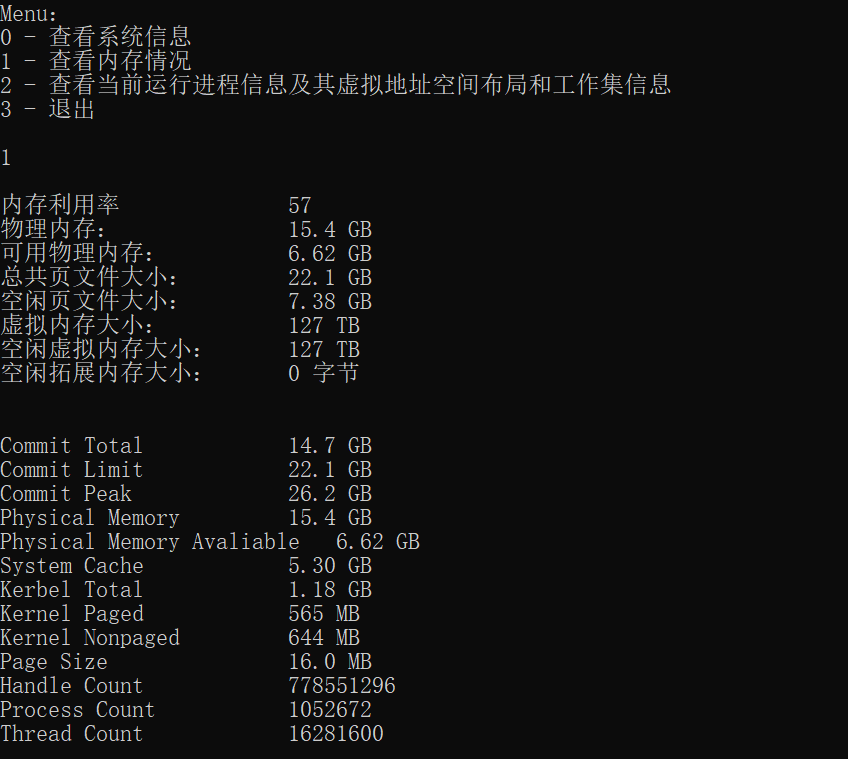
PPERFORMANCE\_INFORMATION pPerformanceInformation,

DWORD cb

);

其中，pPerformanceInformation是指向用于保存返回信息的指针，cb需要指明PERFORMANCE\_INFORMATION结构体的大小。





获取所有进程的主要过程是先获取所有进程的一个snapshot，由于进程信息和数量是动态变化的，所以需要先获取一个静态的信息集；其次，类似于目录检索对snapshot进行顺序检索，获取进程信息。

③ 创建进程snapshot[#](https://www.cnblogs.com/harrypotterjackson/p/12113387.html" \l "132094462)

* CreateToolhelp32Snapshot

HANDLE CreateToolhelp32Snapshot(

DWORD dwFlags,

DWORD th32ProcessID

);

其中，DWORD dwFlags表明该函数获取多少有关属性到snapshot中，DWORD th32ProcessID指需要获取的进程的pid。0表示是最近的进程。

④ 遍历进程[#](https://www.cnblogs.com/harrypotterjackson/p/12113387.html" \l "909590826)

我们用到的数据结构为PROCESSENTRY32，用到的API为Process32First和Process32Next。

* PROCESSENTRY32

typedef struct tagPROCESSENTRY32 {

DWORD dwSize;

DWORD cntUsage;

DWORD th32ProcessID;

ULONG\_PTR th32DefaultHeapID;

DWORD th32ModuleID;

DWORD cntThreads;

DWORD th32ParentProcessID;

LONG pcPriClassBase;

DWORD dwFlags;

TCHAR szExeFile[MAX\_PATH];

} PROCESSENTRY32, \*PPROCESSENTRY32;

在使用前必须指定dwSize = sizeof(PROCESSENTRY32)。

* Process32First

BOOL Process32First(

HANDLE hSnapshot,

LPPROCESSENTRY32 lppe

);

HANDEL hSnapshot是从上述CreateToolhelp32Snapshot获得的，LPPROCESSENTRY32 lppe是PROCESSENTRY32的指针。

* Process32Next

BOOL Process32Next(

HANDLE hSnapshot,

LPPROCESSENTRY32 lppe

);

其中，hSnapshot是同一个，不同的是lppe此时有了值，用于保存当前项的下一项的进程的状态信息。



⑤ 获取单个进程的详细信息[#](https://www.cnblogs.com/harrypotterjackson/p/12113387.html" \l "2062963190)

使用到的主要数据结构有：SYSTEM\_INFO，MEMORY\_BASIC\_INFORMATION,使用到的主要API有：GetNativeSystemInfo，VirtualQueryEx和OpenProcess。

* MEMORY\_BASIC\_INFORMATION

typedef struct \_MEMORY\_BASIC\_INFORMATION {

PVOID BaseAddress;

PVOID AllocationBase;

DWORD AllocationProtect;

SIZE\_T RegionSize;

DWORD State;

DWORD Protect;

DWORD Type;

} MEMORY\_BASIC\_INFORMATION, \*PMEMORY\_BASIC\_INFORMATION;

其中，PVOID BaseAddress是页面区域的基地址，DWORD AllocationProtect是判定如果这个页面区域是初始获取的，则为其页面区域的保护方式，SIZE\_T RegionSize是当前块的大小，DWORD State是当前页面块的状态，MEM\_COMMIT、MEME\_FREE、MEM\_RESERVE是其三种状态，DWORD Protect是当前块中的页面访问方式，DWORD Type是块类型，MEM\_IMAGE、MEM\_MAPPED、MEM\_PRIVATE是其三种类型。

* OpenProcess

HANDLE OpenProcess(

DWORD dwDesiredAccess,

BOOL bInheritHandle,

DWORD dwProcessId

);

其中，DWORD dwDesiredAccess是访问该进程的方式，BOOL bInheritHandle为真时，该进程的子进程也将继承该函数的返回句柄，DWORD dwProcessId是要打开的进程的PID。

* VirtualQueryEx

SIZE\_T VirtualQueryEx(

HANDLE hProcess,

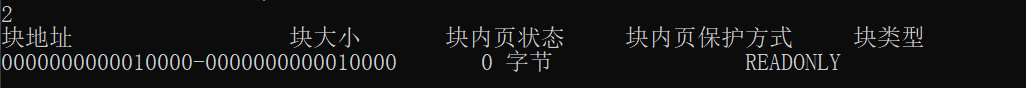
LPCVOID lpAddress,

PMEMORY\_BASIC\_INFORMATION lpBuffer,

SIZE\_T dwLength

);

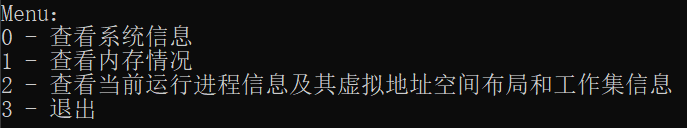
其中，hProcess是要查询的进程的句柄，lpAddress是要查询的进程的虚存的块的基地址，lpBuffer是指向要保存相关信息的数据的指针就是上文提到的MEMORY\_BASIC\_INFORMATION，dwLength = sizeof(MEMORY\_BASIC\_INFORMATION）。



1. **编写、编译代码**

在Visual Studio中编写C++代码，使用动态链接库，实现相应功能。

1. **运行得到结果**

****

四、实验结果及分析

操作过程中，我们通过对问题描述的分析，完成获取系统信息、获取物理内存信息、获取所有进程的基本信息、获取单个进程的详细信息，完成了内存监视器的设计，能实时地显示当前系统中内存的使用情况，包括系统地址空间的布局，物理内存的使用情况，也能实时显示某个进程的虚拟地址空间布局和工作集信息等，实验结果见附录。

五、实验收获与体会

“内存监视”是操作系统知识里非常重要的部分，通过本次实验，我进一步加深了对内存和进程概念的理解，深刻了解到内存监视的过程。

通过本实验，我掌握了许多在Windows下关于内存信息的系统调用函数，了解系统内部内存的工作方式和工作情况，对获取系统信息的API有了更加深入的了解，同时对操作系统内存分配方式有了进一步的印象。

Windows把每个进程的虚拟内存地址映射到物理内存地址，操作系统通过页式管理的方式对内存进行管理。

在亲自动手实验的过程中，我对本章知识点有了更深刻的理解，同时也复习了程序设计方面的知识，这次实验让我收获颇多！

附录：程序清单及说明

