操作系统课程设计实验报告

实验名称： 内存和进程地址空间实时显示

- 班级：`07112005`
- 姓名：`董若扬`
- 学号：`1120202944`

一、 实验目的

学习掌握Windows操作系统下存储器管理相关的API。熟悉Windows内存管理中提供的各种机制和实现的请求分页技术。通过实验，了解Windows内存结构和虚拟内存管理，并学习如何在应用程序中管理内存。了解当前系统中内存的使用情况，包括系统地址空间的布局和物理内存的使用；实时显示进程的虚拟地址空间布局和工作集信息。

二、 实验内容

设计一个内存监视器，能实时地显示当前系统中内存的使用情况，包括物理内存的使用情况；能实时显示某个进程的虚拟地址空间布局信息等等。相关的系统调用： `GetSystemInfo`，`VirtualQueryEx`，`GetPerformanceInfo`，`GlobalMemoryStatusEx` ...

三、 实验环境

> Windows版本： Windows 11 家庭中文版；
>
> 版本 22H2； 安装日期 2022/5/14；
>
> 操作系统版本 22623.746；
>
> 体验 Windows Feature Experience Pack 1000.22636.1000.0.

四、 程序设计与实现

实验原理

内存指的是计算机配置的RAM，系统可以管理所有的物理内存。操作系统（本实验特指Windows）通过分配RAM、页面文件或者二者中的空间，可以准确获取应用程序需要的内存。

在Windows下运行的每个应用程序都认为能够独占4GB的虚拟地址空间，其中，低2GB为进程私有地址空间，用于存放程序和动态链接库，高2GB为所有进程共享区，也就是操作系统占用区。实际上，少有进程可以占有2GB存储空间。Windows把每个进程的虚拟内存地址映射为物理内存地址。

设计实现思路：

获取系统信息

1. `SYSTEM_INFO`

Windows使用`SYSTEM_INFO`结构体保存当前计算机系统的信息，包括处理器的体系结构和类型，系统中处理器的数量，页面大小和其他信息。

- `dwPageSize`：页面大小和页面保护和提交的粒度，是`VirtualAlloc`函数使用的页面大小
- `lpMinimumApplictionAddress`：指向应用程序和动态链接库（DLL）可访问的最低内存指
- `lpMaximumApplicationAddress`：指向应用程序和动态链接库（DLL）可访问的最高内存指针
- `dwActiveProcessorMask`：配置到系统中的处理器集掩码
- `dwNumberOfProcessors`：当前组中的逻辑处理器数，若要检索，则需要调
- `GetLogicalProcessorInformation`函数
- `dwAllocationGranularity`：可以分配虚拟内存的其实地址的粒度
- 头文件：`sysinfoapi.h`（包含于`Windows.h`中）

```
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;
```

2. `GetSystemInfo`

Windows使用`GetSystemInfo`函数获取当前系统的信息。`LPSYSTEM_INFO`指向`SYSTEM_INFO`的指针，

- `lpSystemInfo`：指向接受信息的`SYSTEM_INFO`结构体指针
- 头文件：`sysinfoapi.h`（包括在`Windows.h`中）
- DLL：`KERNEL32.DLL`

```
void GetSystemInfo(
  LPSYSTEM_INFO lpSystemInfo
);
```

获取物理内存信息：

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

1. `MEMORYSTATUSEX`

Windows中使用`MEMORYSTATUSEX`结构体表示内存状态。该结构体包含了物理内存和虚拟内存的现有信息。

- `dwLength`：结构体大小，需要在调用`GlobalMemoryStatusEx`之前设定
- `dwMemoryLoad`：物理内存使用的百分比（从0到100）
- `ullTotalPhys`：实际物理内存的大小（单位：字节）
- `ullAvailPhys`：现在可用的物理内存大小（单位：字节）这部分物理内存是可以不需要写入磁盘立即被回收的内存。它是备用，空闲和零列表大小的综合
- `ullTotalPageFile`：系统或当前进程已经提交的内存限制中的较小者（单位：字节）。要获取系统范围的已提交内存限制需要调用`GetPerformanceInfo`
- `ullAvailPageFile`：当前进程可以提交的最大内存量（单位：字节）
- `ullTotalVirtual`：调用进程的虚拟地址空间的用户模式部分的大小（单位：字节）。此值取决于进程类型，处理机类型和操作系统配置，在x86处理器上多为2GB
- `ullAvailVirtual`：当前调用进程的虚拟地址空间的用户模式中未保留和未提交的内存量（单位：字节）
- `ullAvailExtendevVirtual`：保留值，始终为0
- 头文件：`sysinfoapi.h`（包括在`Windows.h`中）

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

```
typedef struct _MEMORYSTATUSEX {
  DWORD     dwLength;
  DWORD     dwMemoryLoad;
  DWORDLONG ullTotalPhys;
  DWORDLONG ullAvailPhys;
  DWORDLONG ullTotalPageFile;
  DWORDLONG ullAvailPageFile;
  DWORDLONG ullTotalVirtual;
  DWORDLONG ullAvailVirtual;
  DWORDLONG ullAvailExtendedVirtual;
} MEMORYSTATUSEX, *LPMEMORYSTATUSEX;
```

2. `GlobalMemoryStatusEx`

Windows使用`GlobalMemoryStatusEx`函数检索有关系统物理和虚拟内存当前使用情况的信息。`lpBuffer`是指向`MEMORYSTAUSEX`的指针，用于保存信息。

- `lpBuffer`：指向`MEMORYSTATUSEX`结构的指针，该结构接受有关当前内存可用性的信息
- 返回值
    - 若函数成功，则返回值非零
    - 若函数失败，则返回值为0
- 头文件：`sysinfoapi.h`（包括在`Windows.h`中）
- DLL：`KERNEL32.DLL`

```
BOOL GlobalMemoryStatusEx(
  LPMEMORYSTATUSEX lpBuffer
);
```

3. `PERFORMANCE_INFORMATION`

Windows使用`PERFORMANCE_INFORMATION`结构体保存性能信息

- `cb`：结构体大小（单位：字节）
- `CommitTotal`：系统当前提交的页数，提交页面（使用`VirtualAlloc`和`MEM_COMMIT`）会立即更新该值；但是在访问页面之前，物理内存不会被填充
- `CommitPeak`：自上次系统重新引导以来同时处于已提交状态的最大页数
- `PhysicalTotal`：实际物理内存，以页为单位
- `PhysicalAvailable`：当前可用的物理内存量，以页为单位。这部分内存是可以立即重用无需写入磁盘的内存，是备用，空闲和零列表大小的总和
- `SystemCache`：系统缓存内存量，以页为单位。该值为备用列表大小加上系统工作集
- `KernelTotal`：分页和非分页内核池中当前内存的总和，以页为单位
- `KernelPaged`：当前在分页内核池的内存，以页为单位
- `KernelNonpaged`：当前在非分页内核池中的内存，以页为单位
- `PageSize`：页面大小，以字节为单位
- `HandleCount`：当前打开手柄的数量
- `ProcessCount`：当前进程数
- `ThreadCount`：当前线程数
- 头文件：`psapi.h`

```
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;
```

4. `GetPerformanceInfo`

Windows使用`GetPerformanceInfo`函数获取性能信息，填充`PERFORMANCE_INFORMATION`结构中的性能值。

- `pPerformanceInformation`：指向`PERFORMANCE_INFORMATION`结构的指针
- `cb`：`PERFORMANCE_INFORMATION`结构的大小，（单位：字节）
- 返回值：
    - 函数成功，则返回`TRUE`
    - 函数失败，则返回`FALSE`
- 头文件：`psapi.h`
- DLL：PSAPI_VERSION=1时，使用`Psapi.dll`，否则使用`Kernel32.dll`

```
BOOL GetPerformanceInfo(
  PPERFORMANCE_INFORMATION pPerformanceInformation,
  DWORD                    cb
);
```

获取所有进程的基本信息

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。

1. `CreateToolhelp32Snapshot`

创建进程snapshot。`DWORD dwFlags` 用于表明该函数获取多少有关属性到snapshot中。具体说明可参考MSDN, 这里选用的参数是`TH32CS_SNAPALL`。`DWORD th32ProcessID` 需要获取的进程的`pid`。0表示是最近的进程。Windows使用`CreateToolhelp32Snapshot`函数获取指定进程的快照，以及这些进程使用的堆、模块和线程。

- `dwFlags`：包含在快照中的系统部分，本实验选用的是`TH32CS_SNAPPROCESS`，表示包括快照中的所有进程，可以使用`Process32First`枚举这些进程
- `th32ProcessID`：要包含在快照中的进程的标识符，本实验该参数为0，指示当前进程
- 返回值：
    - 若函数成功，则返回快照句柄
    - 若函数失败，则返回`INVALID_HANDLE_VALUE`
- 头文件：`tlhelp32.h`
- DLL：`KERNEL32.DLL`

```
HANDLE CreateToolhelp32Snapshot(
  DWORD dwFlags,
  DWORD th32ProcessID
);
```

2. `PROCESSENTRY32`

- `dwSize`：结构体大小，以字节为单位。在调用`Process32First`函数之前要设置为`sizeof(PROCESSENTRY32)`，若没有设置，则函数调用会是失败
- `th32ProcessID`：进程标识符`PID`
- `cntThreads`：进程启动的线程数
- `th32ParentProcessID`：创建此进程的进程标识符，即父进程的PID
- `pcPriClassBase`：此进程创建的线程的基本优先级
- `szExeFile`：进程的可执行文件
- 其他成员：不再使用，始终设置为0
- 头文件：`tlhelp32.h`

在使用前必须指定`dwSize = sizeof(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;

```

3. `Process32First`

Windows使用`Process32First`函数获取有关系统快照中遇到的第一额进程的信息。

- `hSnapshot`：从先前调用的`CreateToolhelp32Snapshot`函数返回的快照句柄
- `ppe`：指向`PROCESSENTRY32`结构的指针
- 返回值：若进程列表第一个条目已经复制到缓冲区，则返回`TRUE`，否则返回`FALSE`
- 注意：调用函数之前，必须将`PROCESSENTRY32`的`dwSize`设置为结构大小，否则会失败
- 头文件：`tlhelp32.h`
- DLL：`KERNEL32.DLL`

```
BOOL Process32First(
  HANDLE           hSnapshot,
  LPPROCESSENTRY32 lppe
);
```

4. `Process32Next`

Windows使用`Process32Next`函数获取有关系统快照中记录的下一个进程的信息。

- `hSnapshot`：从先前调用的`CreateToolhelp32Snapshot`函数返回的快照句柄
- `lppe`：指向`PROCESSENTRY32`结构的指针
- 头文件：`tlhelp32.h`
- DLL：`KERNEL32.DLL`

```
BOOL Process32Next(
  HANDLE           hSnapshot,
  LPPROCESSENTRY32 lppe
);
```

进程内存统计信息结构

1. `PROCESS_MEMORY_COUNTERS`

- `cb`：结构体大小
- `PageFaultCount`：页面错误的数量
- `WorkingSetSize`：当前工作集大小（单位：字节）
- `PeakWorkingSetSize`：峰值工作集大小（单位：字节）
- `QuotaPeakPagedPoolUsage`：峰值分页池使用情况（单位：字节）
- `QuotaPagedPoolUsage`：当前页面缓冲池使用情况（单位：字节）

- `QuotaPeakNonPagedPoolUsage`：非页面缓冲池使用率的峰值（单位：字节）
- `QuotaNonPagedPoolUsage`：当前非分页池使用情况（单位：字节）
- `PagefileUsage`：此进程的Commit Charge值（单位：字节），Commit Charge是内存管理器为正在运行的进程提交的内存总量
- `PeakPagefileUsage`：此进程的生命周期内提交的峰值（单位：字节）
- 头文件：`psapi.h`

```
typedef struct _PROCESS_MEMORY_COUNTERS {
  DWORD  cb;
  DWORD  PageFaultCount;
  SIZE_T PeakWorkingSetSize;
  SIZE_T WorkingSetSize;
  SIZE_T QuotaPeakPagedPoolUsage;
  SIZE_T QuotaPagedPoolUsage;
  SIZE_T QuotaPeakNonPagedPoolUsage;
  SIZE_T QuotaNonPagedPoolUsage;
  SIZE_T PagefileUsage;
  SIZE_T PeakPagefileUsage;
} PROCESS_MEMORY_COUNTERS;

```

获取进程内存使用情况

1. `GetProcessMemoryInfo`

Windows使用`GetProcessMemoryInfo`函数获取指定进程的内存使用情况的信息。

- `Process`：进程的句柄
- `ppsemCounters`：指向`PROCESS_MEMORY_COUNTERS`或`PROCESS_MEMORY_COUNTERS_EX`结构的指针
- `cb`：`ppsemCounters`结构大小
- 返回值：
    - 若函数成功，则返回值非零
    - 若函数失败，则返回值为零
- 头文件：`psapi.h`
- DLL：若PSAPI_VERSION=1，则使用`Psapi.dll`，否则使用`Kernel32.dll`

```
BOOL GetProcessMemoryInfo(
  HANDLE                   Process,
  PPROCESS_MEMORY_COUNTERS ppsmemCounters,
  DWORD                    cb
);
```

获取单个进程的详细信息

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`，`OpenProcess`

1. `MEMORY_BASIC_INFORMATION`

Windows使用`MEMORY_BASIC_INFORMATION`保存有关进程虚拟地址空间的一系列页面的信息，提供给`VirtualQuery`和`VirtualQueryEx`使用

- `BaseAddress`：指向页面区域的基址指针
- `AllocationBase`：指向`VirtualAlloc`函数分配的一系列页面的基址指针
- `AllocationProtect`：最初分配区域时的内存保护选项
- `RegionSize`：从基址开始的区域大小，其中所有页面都具有相同属性（单位：字节）
- `State`：页面的状态，可以是以下值：
  - `MEM_COMMIT`：表示已在内存中或磁盘页面文件中为其分配物理存储的已提交页面
  - `MEM_FREE`：表示调用进程无法访问且可以分配的空闲页面
  - `MEM_RESERVE`：表示保留进程的虚拟地址空间范围而不分配任何物理存储的保留页面
- `Protect`：该区域中页面的访问保护
- `Type`：页面的类型，可以是以下值：
  - `MEM_IMAGE`：指示区域内的内存页面映射到映射文件到视图中
  - `MEM_MAPPED`：指示区域内的内存页面映射到节的仕途
  - `MEM_PRIVATE`：指示区域内页面是私有的
- 头文件：`winnt.h`（包含于`Windows.h`中）

```
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;
```

2. `OpenProcess`

Windows使用`OpenProcess`函数打开现有的本地进程对象。

- `dwDesiredAccess`：对进程对象的访问权限
- `binheritHandle`：若此值为TRUE，则该进程创建的进程将继承该句柄。否则，子进程不会继承该句柄
- `dwProcessId`：要打开的本地进程的标识符，在实验中取`PROCESSENTRY32`结构中的`th32ProcessID`
- 返回值：
  - 若函数成功，则返回指定进程的打开句柄
  - 若函数失败，则返回NULL
- 头文件：`processthreadspai.h`（包含于`Windows.h`中）
- DLL：`KERNEL32.DLL`

```
HANDLE OpenProcess(
  DWORD dwDesiredAccess,
  BOOL  bInheritHandle,
  DWORD dwProcessId
);
```

3. `VirtualQueryEx`

`hProcess` 要查询的进程的句柄

`lpAddress` 要查询的进程的虚存的块的基地址

`lpBuffer` 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)`

```
SIZE_T VirtualQueryEx(
  HANDLE                    hProcess,
  LPCVOID                   lpAddress,
  PMEMORY_BASIC_INFORMATION lpBuffer,
  SIZE_T                    dwLength
);
```

显示页面保护

1.自定义函数

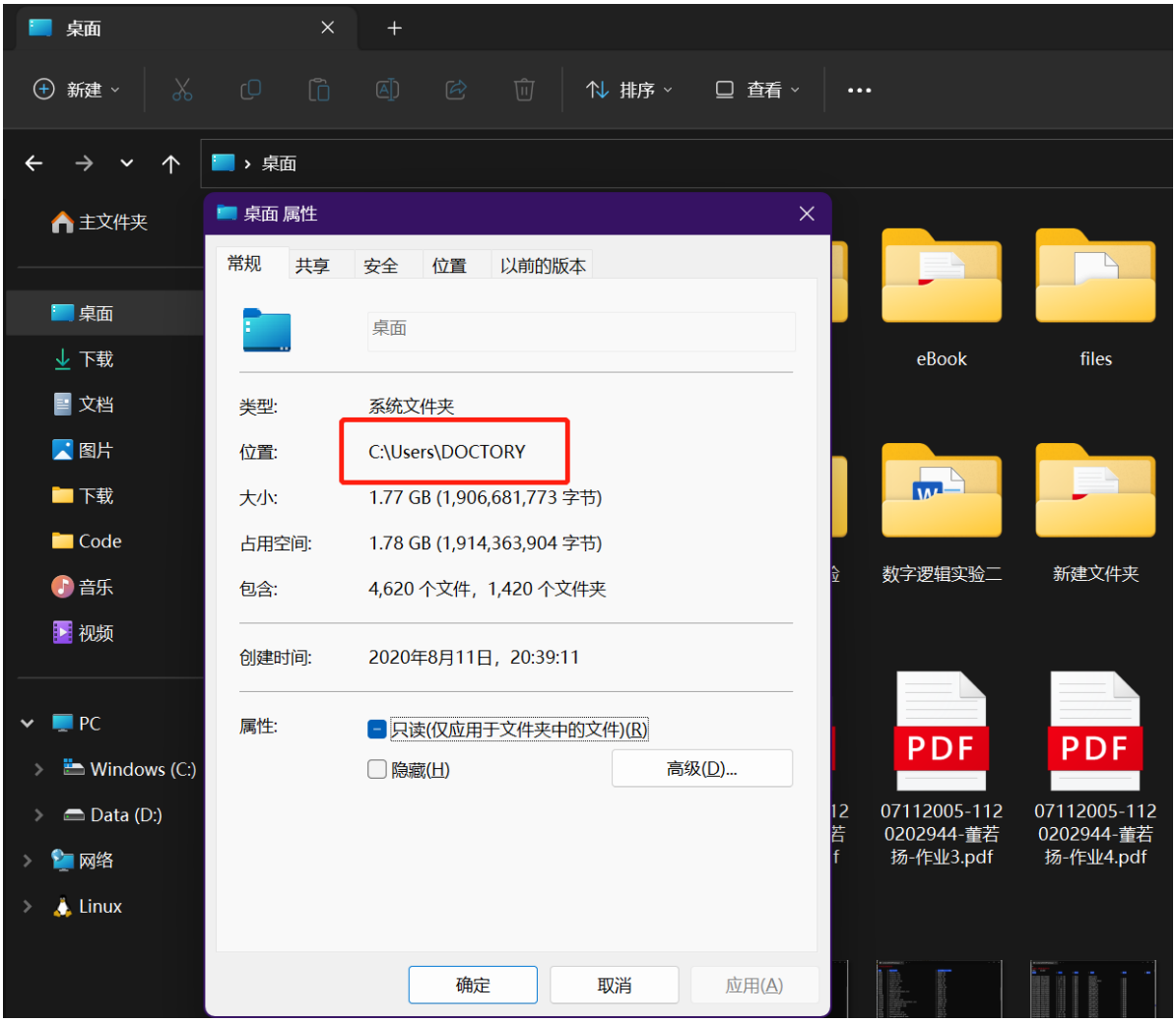
```
inline bool TestSet(DWORD dwTarget,DWORD dwMask){
    return ((dwTarget &dwMask)==dwMask);
}

#define SHOWMASK(dwTarget,type) if(TestSet(dwTarget,PAGE_##type)) {cout << "|" << #type;}

void ShowProtection(DWORD dwTarget){
    SHOWMASK(dwTarget,READONLY);
    SHOWMASK(dwTarget,GUARD);
    SHOWMASK(dwTarget,NOCACHE);
    SHOWMASK(dwTarget,READWRITE);
    SHOWMASK(dwTarget,WRITECOPY);
    SHOWMASK(dwTarget,EXECUTE_READ);
    SHOWMASK(dwTarget,EXECUTE);
    SHOWMASK(dwTarget,EXECUTE_READWRITE);
    SHOWMASK(dwTarget,EXECUTE_WRITECOPY);
    SHOWMASK(dwTarget,NOACCESS);
}

```

运行参数

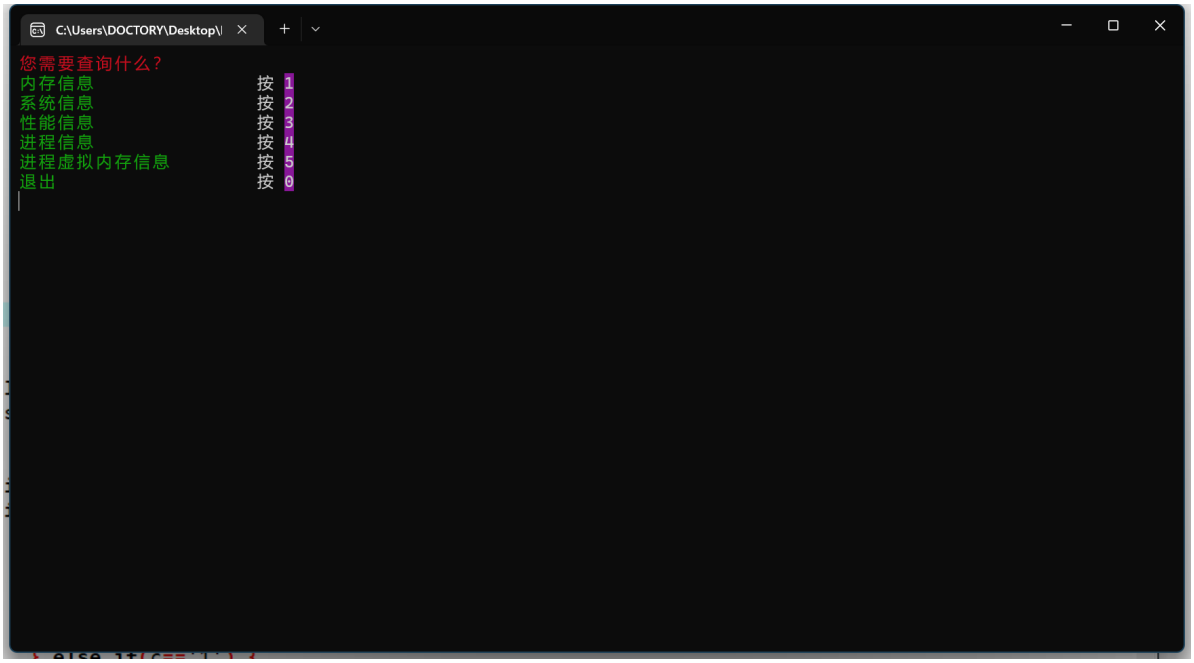
```
g++ MemoryWatch.cpp -lPsapi -lShlwapi -o MemoryWatch.exe
```

运行结果

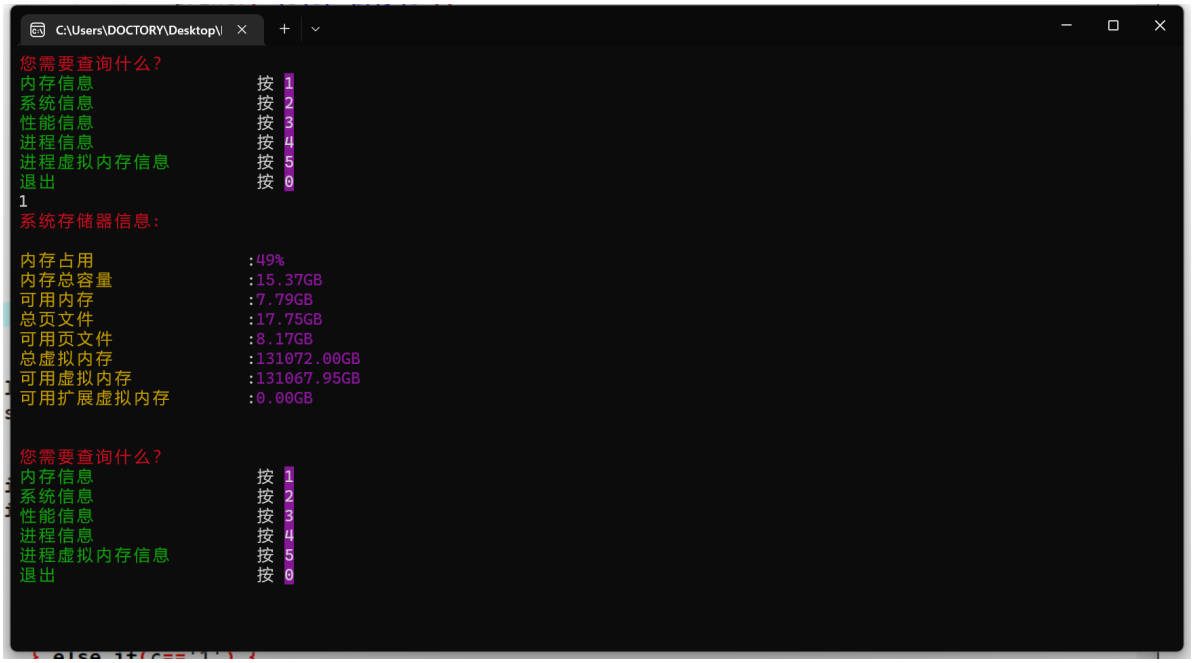
C:\Users\DOCTORY\Desktop 是我的桌面路径，即下文运行结果截图中出现的路径名


桌面
新建
排序
查看
桌面
主文件夹
桌面
下载
文档
图片
下载
Code
音乐
视频
PC
Windows (C:)
Data (D:)
网络
Linux
桌面 属性
常规
共享
安全
位置
以前的版本
桌面
类型: 系统文件夹
位置: C:\Users\DOCTORY
大小: 1.77 GB (1,906,681,773 字节)
占用空间: 1.78 GB (1,914,363,904 字节)
包含: 4,620 个文件，1,420 个文件夹
创建时间: 2020年8月11日，20:39:11
属性: 只读(仅应用于文件夹中的文件)(R)
隐藏(H)
高级(D)...
确定
取消
应用(A)
eBook
files
数字逻辑实验二
新建文件夹
PDF
07112005-112 0202944-董若扬-作业3.pdf
PDF
07112005-112 0202944-董若扬-作业4.pdf

目录


C:\Users\DOCTORY\Desktop\
您需要查询什么？
内存信息 按 1
系统信息 按 2
性能信息 按 3
进程信息 按 4
进程虚拟内存信息 按 5
退出 按 0

内存信息

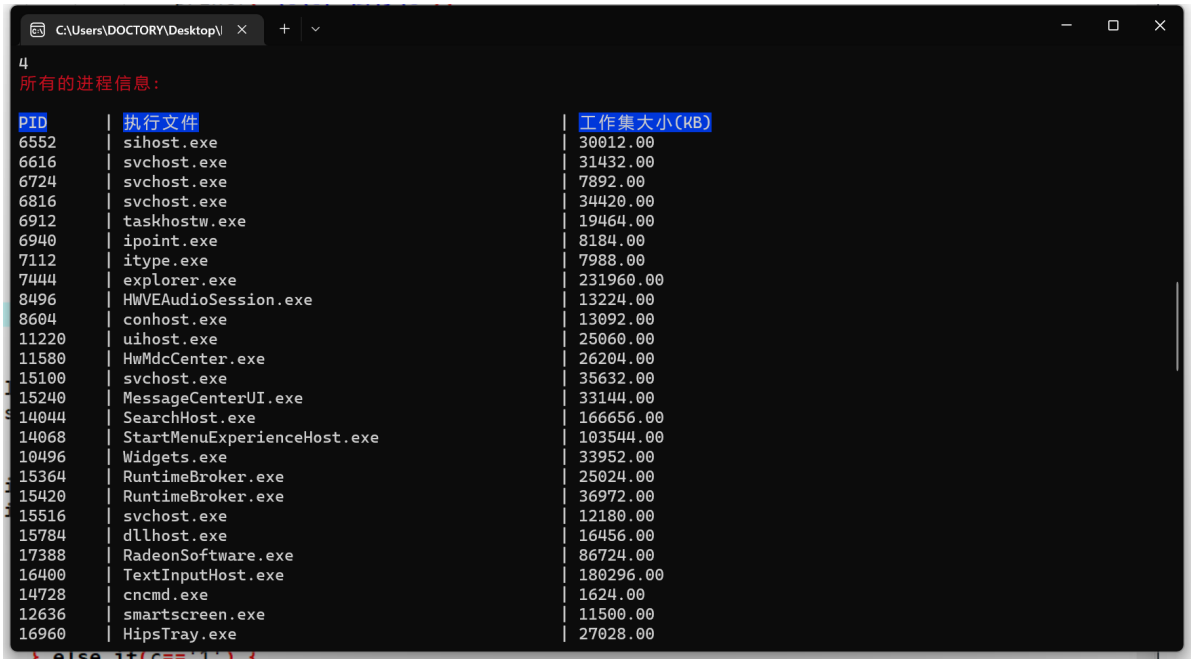

C:\Users\DOCTORY\Desktop\
您需要查询什么？
内存信息 按 1
系统信息 按 2
性能信息 按 3
进程信息 按 4
进程虚拟内存信息 按 5
退出 按 0
1
系统存储器信息：
内存占用 :49%
内存总容量 :15.37GB
可用内存 :7.79GB
总页文件 :17.75GB
可用页文件 :8.17GB
总虚拟内存 :131072.00GB
可用虚拟内存 :131067.95GB
可用扩展虚拟内存 :0.00GB
您需要查询什么？
内存信息 按 1
系统信息 按 2
性能信息 按 3
进程信息 按 4
进程虚拟内存信息 按 5
退出 按 0

系统信息

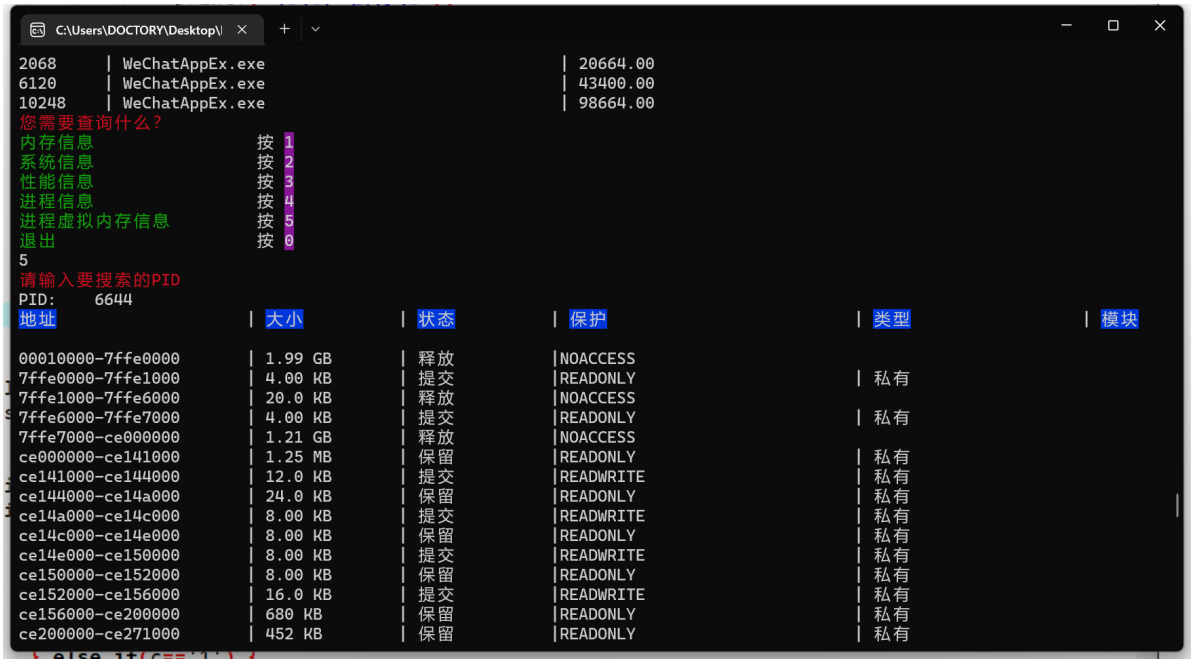

C:\Users\DOCTORY\Desktop\
总虚拟内存 :131072.00GB
可用虚拟内存 :131067.95GB
可用扩展虚拟内存 :0.00GB
您需要查询什么？
内存信息 按 1
系统信息 按 2
性能信息 按 3
进程信息 按 4
进程虚拟内存信息 按 5
退出 按 0
2
系统信息：
页面大小和页面保护和提交的粒度 :4KB
指向应用程序和动态链接库(dll)可访问的最低内存地址的指针 :0x00010000
指向应用程序和dll可访问的最高内存地址的指针 :0xfffeffff
当前组中的逻辑处理器数量 :16
可分配虚拟内存的起始地址的粒度 :64KB
您需要查询什么？
内存信息 按 1
系统信息 按 2
性能信息 按 3
进程信息 按 4
进程虚拟内存信息 按 5
退出 按 0

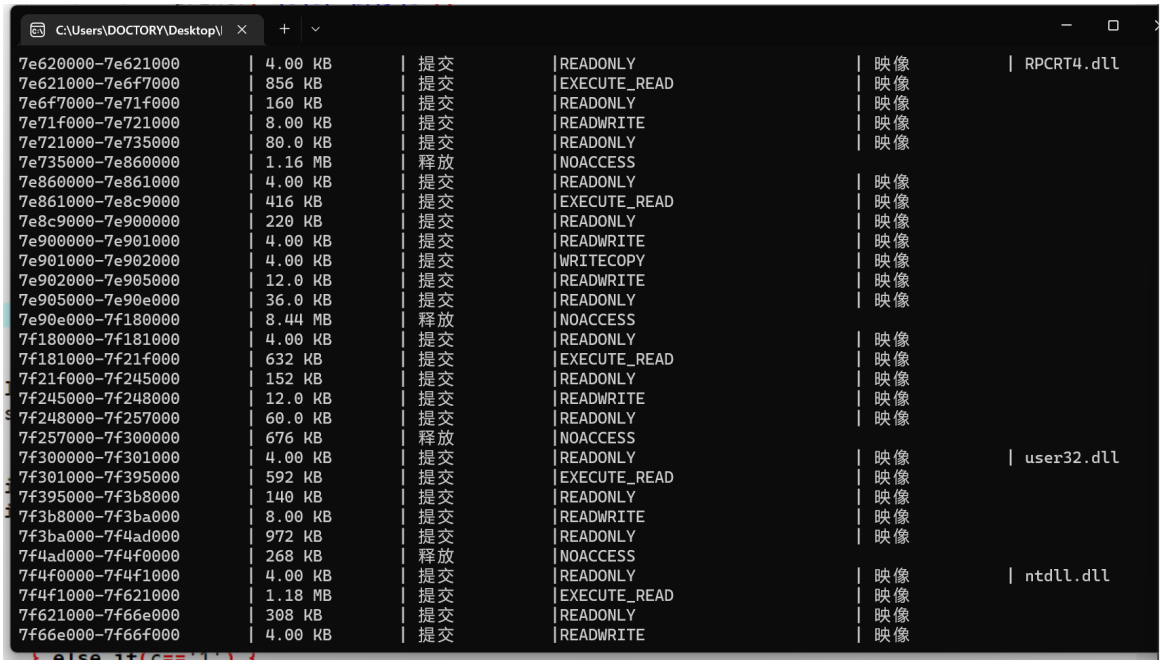
性能信息

进程信息



进程虚拟内存信息





五、 实验收获与体会

困难：编译报错`undefined reference to '__imp_PathStripPathA'`和`undefined reference to '__imp_StrFormatByteSizeA'`

解决：编译时加上编译参数`-lPsapi` `-lShlwapi`

困难：无法识别一些头文件，函数。

解决：工具链由MinGW切换成Visual Studio的MSVC

困难：输出内容太多，长长短短，难寻找有效信息。

解决：使用\t制表符控制格式。使用格式化输出，`\033[35m\033[m`其中`35m`可替换成其他值用于控制输出文本的字体颜色，背景颜色等属性以做辨别。例如标题使用红色，菜单栏使用绿色。特殊元素使用特殊样式以示突出。

困难：IDE中内置的终端似乎不是Windows中的终端，无法显示彩色格式化输出。转义乱码。

解决：不使用CMake构建，转用最基础的Devcpp开发。

通过本实验的操作实践，掌握了许多Windows系统下关于内存信息的系统调用函数，进一步认识了系统内部内存的工作方式与工作情况。对于获取系统信息的API有了更深入的了解。对操作系统内存分配方式有更深入的印象，对Windows系统如何把进程的虚拟内存地址映射到物理内存地址，如何对内存进行页式管理有了更深刻的认识。

由于Windows下关于内存信息的系统调用函数比较冷门（比较热门的是Linux），只能参照微软官方文档学习相关信息，进一步训练了阅读文档，理解函数，理解数据结构的能力。