实验三　内存监视

班级：　07112102　　学号：　1120210964　　姓名：　王英泰

一、实验目的

在 Windows 系统下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

1. 在 Windows 系统下设计实现一个内存监视器。
2. 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，系统正在运行的进程信息；
   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。
3. 可使用的相关系统调用包括：GetSystemInfo, VirtualQueryEx, VirtualAlloc, GetPerformanceInfo, GlobalMemoryStatusEx 等。

三、程序设计与实现

1. 查询当前系统内存使用情况
   1) 通过调用 GetSystemInfo 函数，可以查询内存页大小、分配粒度、程序最小地址、程序最大地址。

```
SYSTEM_INFO siSysInfo;
GetSystemInfo(&siSysInfo);
pageSize = siSysInfo.dwPageSize;
allocationGranularity = siSysInfo.dwAllocationGranularity;
minimumApplicationAddress = siSysInfo.lpMinimumApplicationAddress;
maximumApplicationAddress = siSysInfo.lpMaximumApplicationAddress;
```

   2) 通过调用 GlobalMemoryStatusEx 函数，可以查询当前系统内存使用百分比、物理内存总体容量、物理内存可用容量、虚拟内存总体容量、虚拟内存可用容量、页文件总大小，可用页文件大小。

```
MEMORYSTATUSEX statex;
statex.dwLength = sizeof(statex); // 结构体的长度，在使用函数前必须初始化此值
GlobalMemoryStatusEx(&statex);
std::cout << "Memory Percentage:\t" << statex.dwMemoryLoad << "%" << std::endl;
```

```
std::cout << "Physical Total Memory Size:\t" <<
convertUnit(statex.ullTotalPhys, unit::B, unit::GB) << " GB" << std::endl;
std::cout << "Physical Usable Memory Size:\t" <<
convertUnit(statex.ullAvailPhys, unit::B, unit::GB) << " GB" << std::endl;
std::cout << "Virtual Total Memory Size:\t" <<
convertUnit(statex.ullTotalVirtual, unit::B, unit::TB) << " TB" << std::endl;
std::cout << "Virtual Usable Memory Size:\t" <<
convertUnit(statex.ullAvailVirtual, unit::B, unit::TB) << " TB" << std::endl;
std::cout << "Total PageFile Size:\t" << convertUnit(statex.ullTotalPageFile,
unit::B, unit::GB) << " GB" << std::endl;
std::cout << "Usable PageFile Size:\t" << convertUnit(statex.ullAvailPageFile,
unit::B, unit::GB) << " GB" << std::endl;
```

3) 通过调用 GetPerformanceInfo 函数，可以查询有关页的信息，如当前提交页的总量、最大提交页数量、物理总页数、物理可用页数、句柄数、进程数、线程数等信息。

```
PERFORMANCE_INFORMATION pi;
pi.cb = sizeof(pi);
GetPerformanceInfo(&pi, sizeof(pi));
std::cout << "Commit Total:\t" << pi.CommitTotal << std::endl;
std::cout << "Commit Limit:\t" << pi.CommitLimit << std::endl;
std::cout << "Commit Peak:\t" << pi.CommitPeak << std::endl;
std::cout << "Physical Total:\t" << pi.PhysicalTotal << std::endl;
std::cout << "Physical Available:\t" << pi.PhysicalAvailable << std::endl;
std::cout << "System Cache:\t" << pi.SystemCache << std::endl;
std::cout << "Kernel Total:\t" << pi.KernelTotal << std::endl;
std::cout << "Kernel Paged:\t" << pi.KernelPaged << std::endl;
std::cout << "Kernel Non Paged:\t" << pi.KernelNonpaged << std::endl;
std::cout << "Handle Count:\t" << pi.HandleCount << std::endl;
std::cout << "Process Count:\t" << pi.ProcessCount << std::endl;
std::cout << "Thread Count:\t" << pi.ThreadCount << std::endl;
```

2. 查询当前系统正在运行的进程信息

1) CreateToolhelp32Snapshot 函数可以获取系统中正在运行的进程信息、线程信息等，函数的第一个参数应该为 TH32CS_SNAPPROCESS，表示包括系统中快照中的所有进程，此时第二个参数将被忽略，并且所有进程都包含在快照中。

```
HANDLE hSnapShot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
if (hSnapShot == INVALID_HANDLE_VALUE)
```

```
{
    std::cout << "Failed to create snapshot of running processes." << std::endl;
}
```

2）调用 Process32First 和 Process32Next 函数来获取当迭代进程信息。

```
PROCESSENTRY32 processEntry;
processEntry.dwSize = sizeof(PROCESSENTRY32);

if (!Process32First(hSnapShot, &processEntry))
{
    std::cout << "Failed to read first process entry." << std::endl;
    CloseHandle(hSnapShot);
}
std::cout << "PID\t" << "Thread Count\t" << "Parent PID\t"
    << "Priority\t" << "Name\t"<< std::endl;
do
{
    std::cout << processEntry.th32ProcessID << "\t";
    std::cout << processEntry.cntThreads << "\t\t";
    std::cout << processEntry.th32ParentProcessID << "\t\t";
    std::cout << processEntry.pcPriClassBase << "\t\t";
    std::cout << processEntry.szExeFile << std::endl;
} while (Process32Next(hSnapShot, &processEntry));
```

3）最后要关闭句柄，防止资源泄露。

```
CloseHandle(hSnapShot);
```

3. 查询某一进程的虚拟地址空间布局与工作集信息

1）通过调用 OpenProcess 函数来打开要查询的进程，获得进程句柄。该函数的第一个参数为进程访问权限，第二个参数表示是否继承当前进程，第三个参数表示 PID。

```
HANDLE handle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
if (handle == nullptr)
{
    std::cout << "Open process failed!\t" << GetLastError() << std::endl;
    return;
}
```

2）循环整个应用程序地址空间，调用 VirtualQueryEx 函数来检索指定进程的虚拟地址空间中的页范围的信息，MEMORY_BASIC_INFORMATION 结构体的 State 字段表示块的状态：MEM_COMMIT、MEM_FREE、MEM_RESERVE，MEMORY_BASIC_INFORMATION 结构体的 Protect 字段表示块的访问保护：PAGE_EXECUTE、PAGE_EXECUTE_READ、PAGE_EXECUTE_READWRITE、PAGE_NOACCESS、PAGE_READONLY、PAGE_READWRITE，MEMORY_BASIC_INFORMATION 结构体的 Type 字段表示块的类型：MEM_IMAGE、MEM_MAPPED、MEM_PRIVATE。

```
MEMORY_BASIC_INFORMATION mbi;  // 进程虚拟内存空间的基本信息结构
ZeroMemory(&mbi, sizeof(mbi)); // 分配缓冲区，用于保存信息

// 循环整个应用程序地址空间
for (LPCVOID pBlock = minimumApplicationAddress; pBlock <
maximumApplicationAddress; pBlock += mbi.RegionSize)
{
   // VirtualQueryEx 检索有关指定进程的虚拟地址空间中的页范围的信息
   if (VirtualQueryEx(handle, pBlock, &mbi, sizeof(mbi)))
   {
      std::cout << mbi.BaseAddress << "\t";
      std::cout << convertUnit(mbi.RegionSize, unit::B, unit::KB) << " KB\t";

      // 显示块的状态 switch (mbi.State)
      // 显示访问保护 mbi.State == MEM_COMMIT switch (mbi.Protect)
      // 显示页面的类型 mbi.State != MEM_FREE switch (mbi.Type)
}
```

3）调用 GetProcessMemoryInfo 函数来查询进程工作集信息。

```
PROCESS_MEMORY_COUNTERS pmc;
ZeroMemory(&pmc, sizeof(pmc));
GetProcessMemoryInfo(handle, &pmc, sizeof(pmc));
std::cout << "WorkingSet Size:\t" << convertUnit(pmc.WorkingSetSize, unit::B,
unit::KB) << " KB" << std::endl;
std::cout << "Peak WorkingSet Size:\t" << convertUnit(pmc.PeakWorkingSetSize,
unit::B, unit::KB) << " KB" << std::endl;
```

4）关闭进程句柄，防止资源泄露。

```
CloseHandle(handle);
```

四、实验结果及分析

1. 查询当前内存使用情况，程序运行结果如下：

```
--------------------MemoryManger--------------------
1       Memory Status
2       Running Processes Info
3       Virtual Address Space Layout and WorkingSet Information of a Process
4       Exit
----------------------------------------------------

Please Input the Index of Instruction:  1

Page Size:      4 KB
Min App Address:        0x10000
Max App Address:        0x7ffffffeffff
Allocation Granularity: 65536
Memory Percentage:      53%
Physical Total Memory Size:     13.8555 GB
Physical Usable Memory Size:    6.4977 GB
Virtual Total Memory Size:      128 TB
Virtual Usable Memory Size:     127.996 TB
Total PageFile Size:    20.3555 GB
Usable PageFile Size:   6.9315 GB
Commit Total:   3519029
Commit Limit:   5336079
Commit Peak:    5344859
Physical Total: 3632143
Physical Available:     1703334
System Cache:   1577354
Kernel Total:   329717
Kernel Paged:   164610
Kernel Non Paged:       165107
Handle Count:   115629
Process Count:  242
Thread Count:   3584
```

可以看到，在我的计算机上，页大小为 4KB，内存使用百分比为 53%，物理内存大小为 13.8555GB,物理可用内存大小为 6.4977GB;物理页个数为 3632143，换算成为 13.8555GB，与实际物理内存大小相等；当前运行的进程数为 242，平均每个进程使用 31MB 内存。

2. 查询当前系统正在运行的进程信息

```
--------------------MemoryManger--------------------
1       Memory Status
2       Running Processes Info
3       Virtual Address Space Layout and WorkingSet Information of a Process
4       Exit
----------------------------------------------------

Please Input the Index of Instruction:  2

PID     Thread Count    Parent PID      Priority        Name
0       32              0               0               [System Process]
4       337             0               8               System
204     4               4               8               Registry
732     2               4               11              smss.exe
1148    19              976             13              csrss.exe
1304    2               976             13              wininit.exe
1376    5               1304            9               services.exe
1388    8               1304            9               lsass.exe
1508    12              1376            8               svchost.exe
1536    5               1304            8               fontdrvhost.exe
1608    12              1376            8               svchost.exe
1656    3               1376            8               svchost.exe
1852    3               1376            8               svchost.exe
1860    8               1376            8               svchost.exe
1868    12              1376            8               svchost.exe
2024    2               1376            8               svchost.exe
2032    4               1376            8               svchost.exe
2040    2               1376            8               svchost.exe
1808    5               1376            8               svchost.exe
1964    10              1376            8               svchost.exe
2072    7               1376            8               svchost.exe
2116    13              1376            8               svchost.exe
2228    17              1376            8               svchost.exe
2428    2               1376            8               svchost.exe
2436    1               1376            8               svchost.exe
2448    10              1376            8               NVDisplay.Container.exe
2616    4               1376            8               amdfendrsr.exe
2628    4               1376            8               atiesrxx.exe
2640    6               1376            8               svchost.exe
2664    16              1376            8               svchost.exe
```

可以看到，0 号进程为系统进程，优先级为 0，不过它是一个伪进程，用来统计空闲 CPU 时间。

3. 查询某一进程的虚拟地址空间布局与工作集信息

```
--------------------MemoryManger--------------------
1       Memory Status
2       Running Processes Info
3       Virtual Address Space Layout and WorkingSet Information of a Process
4       Exit
----------------------------------------------------

Please Input the Index of Instruction:  3

Please Input PID:       14860
BaseAddress     RegionSize      State           Protect         Type
0x10000 2.09696e+06 KB  FREE
0x7ffe0000      4 KB    COMMIT          READONLY        PRIVATE
0x7ffe1000      4 KB    COMMIT          READONLY        PRIVATE
0x7ffe2000      8.96125e+08 KB  FREE
0xd627200000    872 KB  RESERVE         PRIVATE
0xd6272da000    12 KB   COMMIT          READWRITE       PRIVATE
0xd6272dd000    1164 KB RESERVE         PRIVATE
0xd627400000    2024 KB RESERVE         PRIVATE
0xd6275fa000    12 KB   COMMIT          PRIVATE
0xd6275fd000    12 KB   COMMIT          READWRITE       PRIVATE
0xd627600000    1.99693e+09 KB  FREE
0x2b242890000   64 KB   COMMIT          READWRITE       MAPPED
0x2b2428a0000   12 KB   COMMIT          READONLY        MAPPED
0x2b2428a3000   52 KB   FREE
0x2b2428b0000   124 KB  COMMIT          READONLY        MAPPED
0x2b2428cf000   4 KB    FREE
0x2b2428d0000   16 KB   COMMIT          READONLY        MAPPED
0x2b2428d4000   48 KB   FREE
0x2b2428e0000   4 KB    COMMIT          READONLY        MAPPED
0x2b2428e1000   60 KB   FREE
0x2b2428f0000   8 KB    COMMIT          READWRITE       PRIVATE
0x2b2428f2000   56 KB   FREE
0x2b242900000   196 KB  COMMIT          READONLY        MAPPED
0x2b242931000   60 KB   FREE
0x2b242940000   12 KB   COMMIT          READONLY        MAPPED
0x2b242943000   52 KB   FREE
0x2b242950000   8 KB    COMMIT          READWRITE       PRIVATE
0x2b242952000   384 KB  RESERVE         PRIVATE
0x2b2429b2000   56 KB   FREE
0x7ff930f38000  12 KB   COMMIT          READWRITE       IMAGE
0x7ff930f3b000  8 KB    COMMIT          IMAGE
0x7ff930f3d000  12 KB   COMMIT          READWRITE       IMAGE
0x7ff930f40000  28 KB   COMMIT          READONLY        IMAGE
0x7ff930f47000  5284 KB FREE
0x7ff931470000  4 KB    COMMIT          READONLY        IMAGE
0x7ff931471000  440 KB  COMMIT          EXECUTE_READ    IMAGE
0x7ff9314df000  164 KB  COMMIT          READONLY        IMAGE
0x7ff931508000  4 KB    COMMIT          READWRITE       IMAGE
0x7ff931509000  8 KB    COMMIT          IMAGE
0x7ff93150b000  4 KB    COMMIT          READWRITE       IMAGE
0x7ff93150c000  40 KB   COMMIT          READONLY        IMAGE
0x7ff931516000  40 KB   FREE
0x7ff931520000  4 KB    COMMIT          READONLY        IMAGE
0x7ff931521000  516 KB  COMMIT          EXECUTE_READ    IMAGE
0x7ff9315a2000  220 KB  COMMIT          READONLY        IMAGE
0x7ff9315d9000  4 KB    COMMIT          READWRITE       IMAGE
0x7ff9315da000  4 KB    COMMIT          IMAGE
0x7ff9315db000  36 KB   COMMIT          READONLY        IMAGE
0x7ff9315e4000  48 KB   FREE
0x7ff9315f0000  4 KB    COMMIT          READONLY        IMAGE
0x7ff9315f1000  424 KB  COMMIT          EXECUTE_READ    IMAGE
0x7ff93165b000  224 KB  COMMIT          READONLY        IMAGE
0x7ff931693000  4 KB    COMMIT          READWRITE       IMAGE
0x7ff931694000  4 KB    COMMIT          IMAGE
0x7ff931695000  12 KB   COMMIT          READWRITE       IMAGE
0x7ff931698000  36 KB   COMMIT          READONLY        IMAGE
0x7ff9316a1000  19516 KB         FREE
0x7ff9329b0000  4 KB    COMMIT          READONLY        IMAGE
0x7ff9329b1000  1220 KB COMMIT          EXECUTE_READ    IMAGE
0x7ff932ae2000  312 KB  COMMIT          READONLY        IMAGE
0x7ff932b30000  4 KB    COMMIT          READWRITE       IMAGE
0x7ff932b31000  8 KB    COMMIT          IMAGE
0x7ff932b33000  36 KB   COMMIT          READWRITE       IMAGE
0x7ff932b3c000  556 KB  COMMIT          READONLY        IMAGE
0x7ff932bc7000  2.85288e+07 KB  FREE

WorkingSet Size:        4784 KB
Peak WorkingSet Size:   5276 KB
```

可以看到每个进程都有自己的虚拟地址空间布局，包括代码段、数据段、堆栈，这些区域的大小和布局可以通过 VirtualQueryEx 等函数进行查询；进程的

工作集是进程的虚拟地址空间中当前驻留在物理内存中的一组页面，它代表了进程在物理内存中的活动区域，通过查询工作集信息，我们可以了解进程实际使用的物理内存情况。

五、实验收获与体会

通过本次实验，我深入了解了 Windows 系统的内存管理机制，包括物理内存、虚拟内存、地址空间布局和工作集等概念。在实验过程中，我使用了许多系统调用和 API 函数来获取内存信息，这些函数让我对 Windows 系统的内部实现有了更深入的了解。

通过这次实验，我意识到内存优化在程序设计和开发中的重要性。对于大型程序或需要处理大量数据的程序，有效地管理和优化内存使用可以显著提高程序的性能。通过合理的内存分配和释放策略，可以减少内存碎片，提高程序的响应速度和运行效率。

此外，每个进程都有自己的虚拟地址空间和工作集，这使得 Windows 系统能够有效地支持多任务处理和内存共享。这种设计使得每个进程都能独立地运行，同时又能够在需要时共享系统资源。

总的来说，本次实验加深了我对内存管理的理解，也提高了我的编程技能和对操作系统原理的理解。

附录：程序清单及说明

MemoryMonitor.h：定义了 MemoryMonitor 类，该类实现了查询当前系统内存使用情况、查询当前系统正在运行的进程信息和查询某一进程的虚拟地址空间布局与工作集信息等功能，并提供接口供外界调用。

Main.cpp：主程序，其主要功能是显示程序界面，接收用户命令，并显示相应的结果。