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：
3. 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；
4. 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。
5. 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
2. 通过调用[GetSystemInfo](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-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;

1. 通过调用[GlobalMemoryStatusEx](https://learn.microsoft.com/zh-cn/windows/win32/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex?redirectedfrom=MSDN)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;

1. 通过调用[GetPerformanceInfo](https://learn.microsoft.com/zh-cn/windows/win32/api/psapi/nf-psapi-getperformanceinfo?source=recommendations)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;

1. 查询当前系统正在运行的进程信息
2. [CreateToolhelp32Snapshot](https://learn.microsoft.com/zh-cn/windows/win32/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot?redirectedfrom=MSDN)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;

}

1. 调用[Process32First](https://learn.microsoft.com/zh-cn/windows/win32/api/tlhelp32/nf-tlhelp32-process32first)和[Process32Next](https://learn.microsoft.com/zh-cn/windows/win32/api/tlhelp32/nf-tlhelp32-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));

1. 最后要关闭句柄，防止资源泄露。

CloseHandle(hSnapShot);

1. 查询某一进程的虚拟地址空间布局与工作集信息
2. 通过调用[OpenProcess](https://learn.microsoft.com/zh-cn/windows/win32/api/processthreadsapi/nf-processthreadsapi-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;

}

1. 循环整个应用程序地址空间，调用[VirtualQueryEx](https://learn.microsoft.com/zh-cn/windows/win32/api/memoryapi/nf-memoryapi-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)

}

1. 调用[GetProcessMemoryInfo](https://learn.microsoft.com/zh-cn/windows/win32/api/psapi/nf-psapi-getprocessmemoryinfo?redirectedfrom=MSDN)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;

1. 关闭进程句柄，防止资源泄露。

CloseHandle(handle);

四、实验结果及分析

1. 文本

   描述已自动生成查询当前内存使用情况，程序运行结果如下：

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。

1. 查询当前系统正在运行的进程信息

可以看到，图片包含 文本

描述已自动生成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。

1. 文本

   描述已自动生成图片包含 图形用户界面

   描述已自动生成查询某一进程的虚拟地址空间布局与工作集信息

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。