
操作系统课程设计实验报告

实验名称： 内存监视

姓名/学号： 张惟振/1120170117

一、 实验目的

通过实验熟悉 Windows 存储器管理中提供的各类机制和实现的请求调页技术。

了解 Windows 内存结构和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。

熟悉使用 Windows 存储器相关的 API。

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，包括系统地址空间的布局，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和工作集信息等。

三、 实验环境

1、软件环境

Windows10 操作系统，版本号：1909

2、硬件环境

Intel® Core™ i5-7200U CPU @ 2.50GHz×4

四、 程序设计与实现

1、实验思路

（1）对于查看性能信息的功能，可以使用 `GetPerformanceInfo()` 来实现，该函数得到一个 `PERFORMANCE_INFORMATION` 结构体，保存性能信息。

（2）对于查看内存信息的功能，可以使用 `GlobalMemoryStatusEx()` 来实现，该函数得到一个 `MEMORYSTATUSEX` 结构体，包含了物理内存和虚拟内存的现有信息。

（3）对于查看系统信息的功能，可以使用 `GetSystemInfo()` 来实现，该函数得到一个 `SYSTEM_INFO` 结构体，保存有当前计算机系统的信息。

(4) 对于查看进程信息的功能，可以先使用 `CreateToolhelp32Snapshot` (`()`) 获取当前运行进程的快照，然后使用 `Process32First` (`()`) 获取第一个进程的句柄，进而使用 `Process32Next` (`()`) 遍历进程快照中所有的进程，即可查看所有进程的信息。

(5) 对于查看进程的虚拟地址空间布局功能，可以在对程序的地址空间进行遍历的过程中使用 `VirtualQueryEx` (`()`) 检查虚拟内存的当前信息，该函数将查询到的信息存入 `MEMORY_BASIC_INFORMATION` 结构体中，借此可得到块的地址和大小以及状态、类型等信息。

2、Windows 系统实现中的 API 和数据结构

(1) `GetPerformanceInfo` (`()`) 获取性能信息

`PERFORMANCE_INFORMATION` 结构体部分信息如下：

```
typedef struct _PERFORMANCE_INFORMATION {
    DWORD    cb; //大小
    SIZE_T   CommitTotal; //系统当前提交的页面总数
    SIZE_T   CommitLimit; //系统当前可提交的最大页面数
    SIZE_T   CommitPeak; //系统历史提交页面峰值
    SIZE_T   PhysicalTotal; //按页分配的总物理内存
    SIZE_T   PhysicalAvailable; //当前可用的物理内存
    SIZE_T   SystemCache; //系统缓存的容量
    SIZE_T   KernelTotal; //内存总量
    SIZE_T   KernelPaged; //分页池大小
    SIZE_T   KernelNonpaged; //非分页池大小
    SIZE_T   PageSize; //页的大小
    DWORD    HandleCount; //打开的句柄个数
    DWORD    ProcessCount; //进程个数
    DWORD    ThreadCount; //线程个数
} PERFORMANCE_INFORMATION, *PPERFORMANCE_INFORMATION;
```

实验使用如下：

```

void ShowPerformanceInfo() //获取性能信息
{
    PERFORMANCE_INFORMATION pi;
    pi.cb = sizeof(pi);
    GetPerformanceInfo(&pi, sizeof(pi));
    cout << "性能信息: " << endl;
    cout << left << setw(30) << "系统当前提交的页面总数为: " << pi.CommitTotal << endl;
    cout << left << setw(30) << "系统当前可提交的最大页面数为: " << pi.CommitLimit << endl;
    cout << left << setw(30) << "系统历史提交页面峰值为: " << pi.CommitPeak << endl;
    cout << left << setw(30) << "按页分配的总物理内存为: " << pi.PhysicalTotal << endl;
    cout << left << setw(30) << "当前可用的物理内存为: " << pi.PhysicalAvailable << endl;
    cout << left << setw(30) << "系统缓存的容量为: " << pi.SystemCache << endl;
    cout << left << setw(30) << "内存总量为: " << pi.KernelTotal << endl;
    cout << left << setw(30) << "分页池的大小为: " << pi.KernelPaged << endl;
    cout << left << setw(30) << "非分页池的大小为: " << pi.KernelNonpaged << endl;
    cout << left << setw(30) << "页的大小为: " << pi.PageSize / 1024 << "KB" << endl;
    cout << left << setw(30) << "打开的句柄个数为: " << pi.HandleCount << endl;
    cout << left << setw(30) << "进程个数为: " << pi.ProcessCount << endl;
    cout << left << setw(30) << "线程个数为: " << pi.ThreadCount << endl;
    cout << endl;
    << endl;
    return;
}

```

(2) GlobalMemoryStatusEx () 获取内存状态

MEMORYSTATUSEX 结构体的部分信息如下:

```

typedef struct _MEMORYSTATUSEX {
    DWORD        dwLength;//MEMORYSTATUSEX 结构的大小
    DWORD        dwMemoryLoad;//内存的使用率
    DWORDLONG    ullTotalPhys;//内存的总容量
    DWORDLONG    ullAvailPhys;//可用内存大小
    DWORDLONG    ullTotalPageFile;//页文件的总容量
    DWORDLONG    ullAvailPageFile;//可用的页文件大小
    DWORDLONG    ullTotalVirtual;//虚拟内存的总容量
    DWORDLONG    ullAvailVirtual;//可用的虚拟内存大小
    DWORDLONG    ullAvailExtendedVirtual;//可用的扩展虚拟内存大小
} MEMORYSTATUSEX, *LPMEMORYSTATUSEX;

```

实验使用如下:

```
void ShowMemoryInfo() //获取内存信息
```

```
{  
    MEMORYSTATUSEX ms;  
    ms.dwLength = sizeof(ms);  
    GlobalMemoryStatusEx(&ms);  
    cout << "内存信息: " << endl;  
    cout << left << setw(25) << "内存的使用率为: " << ms.dwMemoryLoad << "%" << endl;  
    cout << left << setw(25) << "内存的总容量为: " << fixed << setprecision(2) << (float)ms.ullTotalPhys / 1024 / 1024 / 1024 << "GB" << endl;  
    cout << left << setw(25) << "可用的内存为: " << fixed << setprecision(2) << (float)ms.ullAvailPhys / 1024 / 1024 / 1024 << "GB" << endl;  
    cout << left << setw(25) << "页文件的总容量为: " << fixed << setprecision(2) << (float)ms.ullTotalPageFile / 1024 / 1024 / 1024 << "GB" << endl;  
    cout << left << setw(25) << "可用的页文件为: " << fixed << setprecision(2) << (float)ms.ullAvailPageFile / 1024 / 1024 / 1024 << "GB" << endl;  
    cout << left << setw(25) << "虚拟内存的总容量为: " << fixed << setprecision(2) << (float)ms.ullTotalVirtual / 1024 / 1024 / 1024 << "GB" << endl;  
    cout << left << setw(25) << "可用的虚拟内存为: " << fixed << setprecision(2) << (float)ms.ullAvailVirtual / 1024 / 1024 / 1024 << "GB" << endl;  
    cout << left << setw(25) << "可用的扩展虚拟内存为: " << fixed << setprecision(2) << (float)ms.ullAvailExtendedVirtual / 1024 << "KB" << endl;  
    cout << endl  
        << endl;  
    return;  
}
```

(3) GetSystemInfo () 获取系统信息

SYSTEM_INFO 结构体的部分信息如下:

```
typedef struct _SYSTEM_INFO {  
    union {  
        DWORD dwOemId;  
        struct {  
            WORD wProcessorArchitecture;//处理器的体系结构  
            WORD wReserved;  
        } DUMMYSTRUCTNAME;  
    } DUMMYUNIONNAME;  
    DWORD dwPageSize;//内存页大小  
    LPVOID lpMinimumApplicationAddress;//每个进程可用地址空间的最小  
内存地址  
    LPVOID lpMaximumApplicationAddress;//每个进程可用地址空间的最大  
内存地址  
    DWORD_PTR dwActiveProcessorMask;//处理器掩码  
    DWORD dwNumberOfProcessors;//处理器数量  
    DWORD dwProcessorType;//处理器类型  
    DWORD dwAllocationGranularity;//能够保留地址空间区域的最小单  
位  
    WORD wProcessorLevel;//处理器等级  
    WORD wProcessorRevision;//处理器版本  
} SYSTEM_INFO, *LPSYSTEM_INFO;
```

实验使用如下:

```

void ShowSystemInfo() //获得系统信息
{
    SYSTEM_INFO si;
    ZeroMemory(&si, sizeof(si));
    GetSystemInfo(&si);
    cout << "系统信息: " << endl;
    cout << left << setw(41) << "内存页的大小为: " << (int)si.dwPageSize / 1024 << "KB" << endl;
    cout << left << setw(41) << "每个进程可用地址空间的最小内存地址为: 0x" << setw(8) << si.lpMinimumApplicationAddress << endl;
    cout << left << setw(41) << "每个进程可用地址空间的最大内存地址为: 0x" << setw(8) << si.lpMaximumApplicationAddress << endl;
    cout << left << setw(41) << "系统配备的CPU的数量为: " << si.dwNumberOfProcessors << endl;
    cout << left << setw(41) << "能够保留地址空间区域的最小单位为: " << si.dwAllocationGranularity / 1024 << "KB" << endl;
    cout << endl;
    << endl;
    return;
}

```

(4) CreateToolhelp32Snapshot() 获取进程快照，当获取快照之后可以利用 Process32First() 获取其中第一个进程的句柄，再结合 Process32Next() 遍历所有进程。

实验使用如下：

```

void ShowProcessInfo() //获得进程信息
{
    PROCESSENTRY32 pe;
    pe.dwSize = sizeof(pe);
    //获取当前运行进程的快照
    HANDLE hProcess = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    //获取第一个进程的句柄
    bool b = Process32First(hProcess, &pe);
    if (b == false)
        cout << "false" << endl;
    string pid = "PID";
    string pname = "进程名称";
    string psize = "进程工作集大小";
    cout << "进程信息: " << endl;
    printf("\n%5s%70s%25s\n", pid.c_str(), pname.c_str(), psize.c_str()); //进程ID, 进程名称, 进程工作集大小
    while (b) //遍历所有进程
    {
        HANDLE h = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pe.th32ProcessID);
        PROCESS_MEMORY_COUNTERS pmc;
        ZeroMemory(&pmc, sizeof(pmc));
        if (GetProcessMemoryInfo(h, &pmc, sizeof(pmc)))
        { //获取进程的使用情况
            printf("%5d%70s%25.2fKB\n", pe.th32ProcessID, pe.szExeFile, ((float)pmc.WorkingSetSize) / 1024);
        }
        b = Process32Next(hProcess, &pe);
    }
    CloseHandle(hProcess);
    cout << endl;
    << endl;
    return;
}

```

(5) VirtualQueryEx() 检查进程虚拟内存的当前信息，会将查询到的信息存入 MEMORY_BASIC_INFORMATION 结构体中，该结构体的部分信息如下：

```

typedef struct _MEMORY_BASIC_INFORMATION {
    PVOID BaseAddress;

    PVOID AllocationBase;

    DWORD AllocationProtect;

    SIZE_T RegionSize; //从基址开始的虚存区大小
}

```

```
DWORD State;//页面状态

DWORD Protect;//页面的访问保护

DWORD Type;//页面的类型

} MEMORY_BASIC_INFORMATION, *PMEMORY_BASIC_INFORMATION;
```

实验使用如下：

```
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

LPCVOID pBlock = (LPCVOID)si.lpMinimumApplicationAddress;
for (; pBlock < si.lpMaximumApplicationAddress;) //遍历整个程序的地址空间
{
    if (VirtualQueryEx(hProcess, //相关的进程
        pBlock, //开始位置
        &mbi, //缓冲区
        sizeof(mbi)) == sizeof(mbi)) //长度的确认
    {
        //查询地址空间中内存地址的信息
        //计算块的结尾及其长度
        LPCVOID pEnd = (PBYTE)pBlock + mbi.RegionSize; //页基地址+虚存区大小
        TCHAR szSize[MAX_PATH];
        StrFormatByteSize(mbi.RegionSize, szSize, MAX_PATH); //格式化文件的大小
        //显示块地址和长度
        printf("块地址: %8X-%8X(%ls) ", (DWORD)pBlock, (DWORD)pEnd, szSize);
        int l = lstrlen(szSize);
        int x = 10 - l;
        for (int i = 0; i < x; i++) //对齐格式
            printf(" ");
        //显示块的状态
        if (mbi.State == MEM_COMMIT)
            cout << "被提交";
        else if (mbi.State == MEM_FREE)
            cout << "空闲的";
        else if (mbi.State == MEM_RESERVE)
            cout << "被保留";
        //显示保护
        if (mbi.Protect == 0 && mbi.State != MEM_FREE)
            mbi.Protect = PAGE_READONLY;
        ShowProtection(mbi.Protect);
        //显示类型
        if (mbi.Type == MEM_IMAGE)
            cout << ",Image";
        else if (mbi.Type == MEM_PRIVATE)
            cout << ",Private";
        else if (mbi.Type == MEM_MAPPED)
```

3、实验结果

查看性能信息

```
C:\Users\11039\Documents\Visual Studio 2019\OSExperiment4\x64\Debug\OSExperiment4.exe
菜单
输入数字以使用对应的功能：
1: 查看性能信息
2: 查看内存状态
3: 查看系统信息
4: 查看进程信息
5: 查看某一进程的虚拟地址空间布局和工作集信息
-1: 返回

1
性能信息：
系统当前提交的页面总数为：    2005238
系统当前可提交的最大页面数为： 2662125
系统历史提交页面峰值为：      2151379
按页分配的总物理内存为：      2072301
当前可用的物理内存为：        695256
系统缓存的容量为：            688805
内存总量为：                  212362
分页池的大小为：              116542
非分页池的大小为：            95820
页的大小为：                  4KB
打开的句柄个数为：            126452
进程个数为：                  259
线程个数为：                  3433

菜单
输入数字以使用对应的功能：
1: 查看性能信息
2: 查看内存状态
```

查看内存状态

```
C:\Users\11039\Documents\Visual Studio 2019\OSExperiment4\x64\Debug\OSExperiment4.exe
菜单
输入数字以使用对应的功能：
1: 查看性能信息
2: 查看内存状态
3: 查看系统信息
4: 查看进程信息
5: 查看某一进程的虚拟地址空间布局和工作集信息
-1: 返回

2
内存信息：
内存的使用率为：    66%
内存的总容量为：    7.91GB
可用的内存为：      2.61GB
页文件的总容量为：  10.16GB
可用的页文件为：    2.49GB
虚拟内存的总容量为： 131072.00GB
可用的虚拟内存为：  131067.91GB
可用的扩展虚拟内存为： 0.00KB

菜单
输入数字以使用对应的功能：
1: 查看性能信息
2: 查看内存状态
3: 查看系统信息
4: 查看进程信息
5: 查看某一进程的虚拟地址空间布局和工作集信息
```

查看系统信息

```
C:\Users\11039\Documents\Visual Studio 2019\OSExperiment4\x64\Debug\OSExperiment4.exe

菜单
输入数字以使用对应的功能:
1: 查看性能信息
2: 查看内存状态
3: 查看系统信息
4: 查看进程信息
5: 查看某一进程的虚拟地址空间布局和工作集信息
-1: 返回

3
系统信息:
内存页的大小为: 4KB
每个进程可用地址空间的最小内存地址为: 0x00000000000010000
每个进程可用地址空间的最大内存地址为: 0x00007FFFFFFF
系统配备的CPU的数量为: 4
能够保留地址空间区域的最小单位为: 64KB

菜单
输入数字以使用对应的功能:
1: 查看性能信息
2: 查看内存状态
3: 查看系统信息
4: 查看进程信息
5: 查看某一进程的虚拟地址空间布局和工作集信息
-1: 返回
```

查看进程信息

```
4: 查看进程信息
5: 查看某一进程的虚拟地址空间布局和工作集信息
-1: 返回

4
进程信息:

PID                                进程名称                                进程工作集大小
5464                               SynTPEnh.exe                            18096.00KB
6848                               tpmumkld.exe                            8008.00KB
6864                               tpsod.exe                               12200.00KB
3288                               rundll32.exe                            20400.00KB
8044                               nvcontainer.exe                         20968.00KB
8132                               nvcontainer.exe                         36368.00KB
2380                               sihost.exe                              35800.00KB
4028                               svchost.exe                             26168.00KB
8212                               svchost.exe                             34740.00KB
8316                               taskhostw.exe                           21268.00KB
8788                               igfxEM.exe                              18036.00KB
8828                               explorer.exe                            192760.00KB
8396                               svchost.exe                             32948.00KB
9616                               dllhost.exe                             21940.00KB
9632                               StartMenuExperienceHost.exe             68044.00KB
9748                               PowerMgr.exe                            3656.00KB
9772                               RuntimeBroker.exe                       29816.00KB
10160                              RuntimeBroker.exe                       26908.00KB
10312                              RemindersServer.exe                     18328.00KB
10408                              ApplicationFrameHost.exe                33488.00KB
10804                              NVIDIA Share.exe                        41332.00KB
10824                              dllhost.exe                             10140.00KB
11160                              RuntimeBroker.exe                       36024.00KB
11464                              SettingSyncHost.exe                     5600.00KB
11648                              LockApp.exe                             34620.00KB
11808                              RuntimeBroker.exe                       25356.00KB
11604                              CastSrv.exe                             6468.00KB
8492                               NVIDIA Share.exe                        22548.00KB
11744                              SecurityHealthSystray.exe                7728.00KB
10480                              NVIDIA Web Helper.exe                   39584.00KB
4724                               conhost.exe                             5316.00KB
11776                              smartscreen.exe                         25328.00KB
12384                              yundetectservice.exe                    8188.00KB
12780                              SearchUI.exe                            75444.00KB
12980                              AppleMobileDeviceProcess.exe            10416.00KB
13288                              WindowsInternal.ComposableShell.Experiences.TextInput.InputApp.exe 35776.00KB
3256                               NVIDIA Share.exe                        62088.00KB
13432                              igfxext.exe                             8424.00KB
14024                              Lenovo.Modern.ImController.PluginHost.CompanionApp.exe 50108.00KB
3252                               ShellExperienceHost.exe                  94788.00KB
```


查看某一进程的虚拟地址空间布局和工作集信息

```
C:\Users\11039\Documents\Visual Studio 2019\OSExperiment4\Debug\OSExperim...
4: 查看进程信息
5: 查看某一进程的虚拟地址空间布局和工作集信息
-1: 返回

5
请输入进程PID:
17316

查看17316号进程的虚拟空间布局和工作集信息:

块地址:      10000-   370000 (3.37 MB)      空闲的, NOACCESS
块地址:      370000-  371000 (4.00 KB)      被提交, READONLY, Image
块地址:      371000-  372000 (4.00 KB)      被保留, READONLY, Image
块地址:      372000-  387000 (84.0 KB)      被提交, READONLY, Image
块地址:      387000-  388000 (4.00 KB)      被保留, READONLY, Image
块地址:      388000-  38F000 (28.0 KB)      被提交, READONLY, Image
块地址:      38F000-  390000 (4.00 KB)      被保留, READONLY, Image
块地址:      390000-  3A0000 (64.0 KB)      被提交, READWRITE, Mapped
块地址:      3A0000-  3A2000 (8.00 KB)      被提交, READWRITE, Private
块地址:      3A2000-  3AE000 (48.0 KB)      被保留, READONLY, Private
块地址:      3AE000-  3B0000 (8.00 KB)      空闲的, NOACCESS
块地址:      3B0000-  3B4000 (16.0 KB)      被提交, READONLY, Mapped
块地址:      3B4000-  3B8000 (16.0 KB)      被保留, READONLY, Mapped
块地址:      3B8000-  3C0000 (32.0 KB)      空闲的, NOACCESS
块地址:      3C0000-  3DB000 (108 KB)      被提交, READONLY, Mapped
块地址:      3DB000-  3E0000 (20.0 KB)      空闲的, NOACCESS
块地址:      3E0000-  3E4000 (16.0 KB)      被提交, READONLY, Mapped
块地址:      3E4000-  3F0000 (48.0 KB)      空闲的, NOACCESS
块地址:      3F0000-  3F1000 (4.00 KB)      被提交, READONLY, Mapped
块地址:      3F1000-  400000 (60.0 KB)      空闲的, NOACCESS
块地址:      400000-  541000 (1.25 MB)      被保留, READONLY, Private
块地址:      541000-  546000 (20.0 KB)      被提交, READWRITE, Private
块地址:      546000-  54C000 (24.0 KB)      被保留, READONLY, Private
块地址:      54C000-  552000 (24.0 KB)      被提交, READWRITE, Private
块地址:      552000-  56D000 (108 KB)      被保留, READONLY, Private
块地址:      56D000-  573000 (24.0 KB)      被提交, READWRITE, Private
块地址:      573000-  579000 (24.0 KB)      被保留, READONLY, Private
块地址:      579000-  57C000 (12.0 KB)      被提交, READWRITE, Private
块地址:      57C000-  600000 (528 KB)      被保留, READONLY, Private
块地址:      600000-  635000 (212 KB)      被保留, READONLY, Private
块地址:      635000-  638000 (12.0 KB)      被提交, GUARD, READWRITE, Private
块地址:      638000-  640000 (32.0 KB)      被提交, READWRITE, Private
块地址:      640000-  735000 (980 KB)      被保留, READONLY, Private
块地址:      735000-  737000 (8.00 KB)      被提交, GUARD, READWRITE, Private
块地址:      737000-  740000 (36.0 KB)      被提交, READWRITE, Private
块地址:      740000-  742000 (8.00 KB)      被提交, READWRITE, Private
块地址:      742000-  750000 (56.0 KB)      空闲的, NOACCESS
块地址:      750000-  751000 (4.00 KB)      被提交, READWRITE, Mapped
块地址:      751000-  760000 (60.0 KB)      空闲的, NOACCESS
块地址:      760000-  76A000 (40.0 KB)      被提交, READWRITE, Private
```

五、实验收获与体会

这次实验其实说难也不难，说简单也不简单。其中重点内容就是调用 Windows 系统中的 API。在对 API 进行调用的时候就，需要了解各种参数的意义和使用，需要分别对其进行准确的对应起来，才可以保证 API 的正确调用,通过这次实验我对获取系统信息的 API 有了更多的了解。

此外，在这个实验中除了调用 API，另外一个重点的内容是需要了解 windows

中的几个结构体，比如 `SYSTEM_INFO`、`MEMORYSTATUS` 和 `PERFORMANCE_INFORMATION` 等结构体。在这些结构体中，有很多我们需要的信息，这些信息就是在本实验中需要用到的信息，了解这些系统预定义的结构体时完成这次试验的重要部分。

最后，我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。