



操作系统实验二

内存管理



目录

一 . 实验内容与要求	1
二 . 程序设计 with 实现	2
1 . 设计思路	2
2 . 程序运行结果	3
3 . 实验结论	13
4 . 源码附件	13

2017-12-10

[裴子祥 计科七班 学号 2015211921]

[指导老师：孟祥武]

一．实验内容与要求

题目：虚拟内存分配方法的实现。

实验目的：在本次实验中，需要从不同的侧面了解 Windows 2000/XP 的虚拟内存机制。在 Windows 2000/XP 操作系统中，可以通过一些 API 操纵虚拟内存。主要需要了解以下几方面：

1. Windows 2000/XP 虚拟存储系统的组织
2. 如何控制虚拟内存空间
3. 如何编写内存追踪和显示工具
4. 详细了解与内存相关的 API 函数的使用

实验内容：Windows 2000/XP 在 x86 体系结构上利用二级页表结构来实现虚拟地址向物理地址的变换。一个 32 位虚拟地址被解释为三个独立的分量——页目录索引、页表索引和字节索引——它们用于找出描述页面映射结构的索引。页面大小及页表项的宽度决定了页目录和页表索引的宽度。

使用 Win32 API 函数，编写一个包含两个线程的进程，一个线程用于模拟内存分配活动，一个线程用于跟踪第一个线程的内存行为。模拟内存活动的线程可以从一个文件中读出要进行的内存操作，每个内存操作包含如下内容：

1. 时间：开始执行的时间。
2. 块数：分配内存的粒度
3. 操作：包括保留一个区域、提交一个区域、释放一个区域、回收一个区域、锁与解锁一个区域，将这些操作编号存放于文件中。
4. 大小：块的大小。
5. 五种访问权限：PAGE_READONLY、PAGE_READWRITE、PAGE_EXECUTE、PAGE_EXECUTE_READ 和 PAGE_EXECUTE_READWRITE，将这些权限编号，存放于文件中。
6. 跟踪线程，将页面大小、已使用的地址范围、物理内存总量以及虚拟内存总量等信息显示出来。

实验环境：

MICROSOFT WINDOW 10

Visual Studio 2015

二 . 程序设计与实现

1 . 设计思路

设计思路

Windows 进程的虚拟地址空间中也有三种状态的页面:空闲页面、保留页面和提交页面。空闲(Free)页面:空闲页面是指那些可以保留或提交的可用页面。保留(Reserved)页面:保留页面是逻辑页面已分配但没有分配物理存储的页面。设置这种状态的效果是可以保留一部分虚拟地址,这样,如果不预先释放这些地址,就不能被其他应用程序(如 Malloc, LocalAlloc 等)的操作所使用。试图读或写空闲页面或保留页面将导致页面出错异常。保留页面可被释放或提交。提交(Committed)页面:提交页面是物理存储(在内存中或磁盘上)已被分配的页面。可对它加以保护,不许访问或允许只读访问,或允许读写访问。提交也可以被回收以释放存储空间,从而变成保留页面。

主要 API 函数接口

实现内存存在管理的主要 API 函数有: GetSystemInfo

函数功能:返回当前系统信息,存放入 lpSystemInfo 中。

GlobalMemoryStatus

函数功能:获得计算机系统中当前使用的物理内存和虚拟内存的信息。使用 GlobalMemoryStatus 函数 可以判断应用程序能够分配多少与其它应用程序不冲突的内存空间。但 GlobalMemoryStatus 函数返回的信息是不稳定的,我们不能保证两次调用该函数都能返回到相同的信息。

VirtualQuery

函数功能:提供有关调用进程虚拟空间中的页面信息。

VirtualAlloc

函数功能:在调用进程的虚拟地址中保留或提交页面。除非设置了 MEM_RESET 标志,否则被这个函数分配的内存单元被自动初始化为 0。

VirtualAllocEX

函数功能:使用该函数可以保留、提交或者保留和提交进程虚拟空间的页面的基址,否则返回 NULL。

VirtualFree

函数功能:可以释放或注销调用进程虚拟空间中的页面。成功则返回一个非零值,否则返回零值。

VirtualFreeEx

函数功能:该函数可以释放或注销指定进程虚拟空间中的页面。VirtualFreeEx 函数和 VirtualFree 函数区别是:VirtualFree 函数释放调用进程的地址空间,而 VirtualFreeEx 函数可以释放任意指定的进程的地址空间。如果函数调用成功则返回非零值,否则返回零值。

VirtualLock

功能:该函数可以将进程虚拟空间中的内存加锁。以确保后面的对该区域的存取操作不会失败。成功则返回一个非零值,否则返回一个零值。

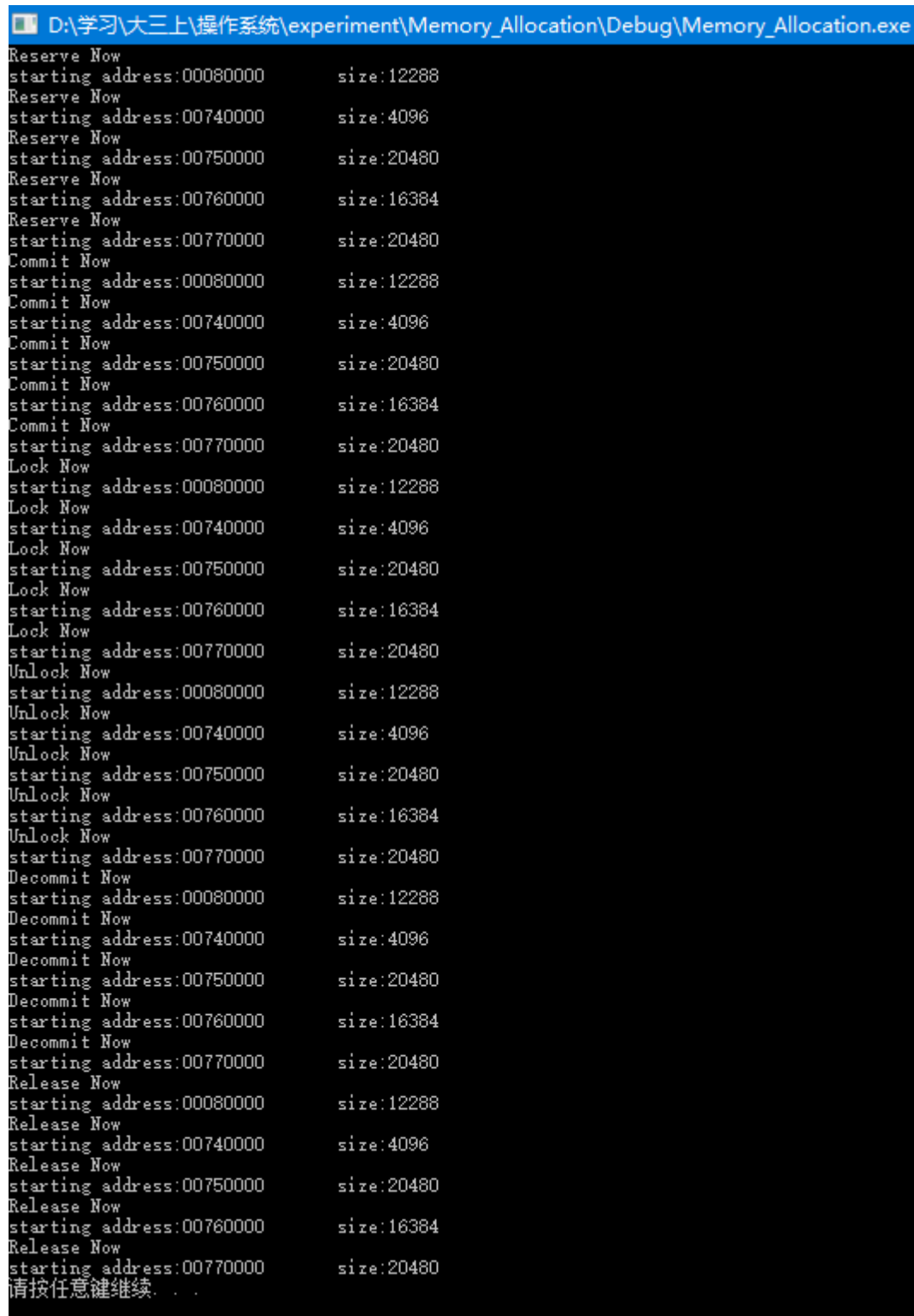
VirtualUnlock

功能:该函数可以将进程虚拟空间指定范围内的页面解锁,从以系统在必要时可以将这些页面换出 到页面文件中。函数调用成功则返回一个非零值,否则返回零值。

实例程序中“memeory_op.cpp”运行后得不到内存输出结果,故重写为“MemAllocation.cpp”,详见“源码附件”。“MemAllocation.cpp”,其中直接使用“makefile.cpp”生成的“opfile”文件。

2. 程序运行结果

程序运行截图



```
D:\学习\大三上\操作系统\experiment\Memory_Allocation\Debug\Memory_Allocation.exe
Reserve Now
starting address:00080000      size:12288
Reserve Now
starting address:00740000      size:4096
Reserve Now
starting address:00750000      size:20480
Reserve Now
starting address:00760000      size:16384
Reserve Now
starting address:00770000      size:20480
Commit Now
starting address:00080000      size:12288
Commit Now
starting address:00740000      size:4096
Commit Now
starting address:00750000      size:20480
Commit Now
starting address:00760000      size:16384
Commit Now
starting address:00770000      size:20480
Lock Now
starting address:00080000      size:12288
Lock Now
starting address:00740000      size:4096
Lock Now
starting address:00750000      size:20480
Lock Now
starting address:00760000      size:16384
Lock Now
starting address:00770000      size:20480
Unlock Now
starting address:00080000      size:12288
Unlock Now
starting address:00740000      size:4096
Unlock Now
starting address:00750000      size:20480
Unlock Now
starting address:00760000      size:16384
Unlock Now
starting address:00770000      size:20480
Decommit Now
starting address:00080000      size:12288
Decommit Now
starting address:00740000      size:4096
Decommit Now
starting address:00750000      size:20480
Decommit Now
starting address:00760000      size:16384
Decommit Now
starting address:00770000      size:20480
Release Now
starting address:00080000      size:12288
Release Now
starting address:00740000      size:4096
Release Now
starting address:00750000      size:20480
Release Now
starting address:00760000      size:16384
Release Now
starting address:00770000      size:20480
请按任意键继续...
```

控制台输出：

```
Reserve Now
starting address:00080000      size:12288
```

Reserve Now	
starting address:00740000	size:4096
Reserve Now	
starting address:00750000	size:20480
Reserve Now	
starting address:00760000	size:16384
Reserve Now	
starting address:00770000	size:20480
Commit Now	
starting address:00080000	size:12288
Commit Now	
starting address:00740000	size:4096
Commit Now	
starting address:00750000	size:20480
Commit Now	
starting address:00760000	size:16384
Commit Now	
starting address:00770000	size:20480
Lock Now	
starting address:00080000	size:12288
Lock Now	
starting address:00740000	size:4096
Lock Now	
starting address:00750000	size:20480
Lock Now	
starting address:00760000	size:16384
Lock Now	
starting address:00770000	size:20480
Unlock Now	
starting address:00080000	size:12288
Unlock Now	
starting address:00740000	size:4096
Unlock Now	
starting address:00750000	size:20480
Unlock Now	
starting address:00760000	size:16384
Unlock Now	
starting address:00770000	size:20480
Decommit Now	
starting address:00080000	size:12288
Decommit Now	
starting address:00740000	size:4096
Decommit Now	
starting address:00750000	size:20480
Decommit Now	

```
starting address:00760000      size:16384
Decommit Now
starting address:00770000      size:20480
Release Now
starting address:00080000      size:12288
Release Now
starting address:00740000      size:4096
Release Now
starting address:00750000      size:20480
Release Now
starting address:00760000      size:16384
Release Now
starting address:00770000      size:20480
```

“outMemState.txt”文件内容：内存与系统信息如下

系统信息，每次操作不变：

```
dwActiveProcessorMask  15
dwAllocationGranularity 65536
dwNumberOfProcessors  4
dwOemId  0
dwPageSize 4096
dwProcessorType 586
IpMaximumApplicationAddress 7FFEFFFF
IpMinimumApplicationAddress 00010000
wProcessorArchitecture  0
wProcessorLevel  6
wProcessorRevision  15363
wReserved  0
```

显示内存基本信息，每次操作后内存基本信息不变：

```
AllocationBase  00000000
AllocationProtect 0
BaseAddress 00010000
Protect 1
RegionSize 327680
State 65536
Type 0
```

~~~~~  
~~~~~

内存状况，动态变化：

```
dwAvailPageFile 3663351808
```

6

7

8

9

10

11

12

3. 实验结论

以页为单位的虚拟内存分配方法，适用于大型对象或结构数组。

dwAvailPhys、dwLength、dwMemoryLoad、dwTotalPageFile、dwTotalPhys、dwTotalVirtual，它们都是内存状态信息中的不变量。

dwAvailPageFile、dwAvailVirtual 是内存状态信息中的动态变量，他会随线程的开始与结束而相应的调整可用页文件大小与可以虚拟地址空间大小。

现在多数的计算机页大小为 4 KB (4096 Byte)；

‘VirtualQuery’ 可以查看虚拟内存分配情况；

‘BaseAddress’ 表示查询的内存基址（64 位）；

‘AllocationAddress’ 表示系统为内存分配的基址（64 位）；

‘AllocationProtect’ 表示申请内存时的访问保护；

‘RegionSize’ 表示申请区域的大小（字节为单位）；

‘Protect’ 表示当前的内存访问保护；

‘State’ 表示当前内存分配状态；

‘Type’ 表示当前内存分配的类型；

‘VirtualAlloc’ 可以申请系统分配虚拟内存；

使用 ‘MEM_RESERVE’ 可以申请**保留**一段空间；

使用 ‘MEM_COMMIT’ 表示**提交**申请，获取保留的空间；

‘VirtualLock’ 用于**锁定**虚拟内存于物理内存中，保证之后对其访问 *不引起缺页中断*；

‘VirtualUnlock’ **取消**刚刚的**锁定**；

‘VirtualFree’ 释放申请的虚拟内存；

使用 ‘MEM_DECOMMIT’ 可以**撤销提交**，返回保留状态；

使用 ‘MEM_RELEASE’ 用于**撤销保留**，其他进程可以申请使用这块内存。

4. 源码附件

“MemAllocation.cpp”

```
#include <iostream>
#include <fstream>
#include <thread>
#include <functional>
#include <vector>
#include <exception>
#include <windows.h>
using namespace std;

typedef struct operation{                //一次内存操作
    int time;//起始时间
    int block;//内存页数
    int oper;//操作
    int protection;//权限
```

```
}Operation;

typedef struct trace {                                //跟踪每次分配活动的数据结构
    LPVOID start; //起始地址
    long size;    //分配的大小
}Trace;

int main()
{
    auto semaphoreTrack = CreateSemaphore(NULL, 1, 1, NULL);
    auto semaphoreAlloc = CreateSemaphore(NULL, 1, 1, NULL);
    auto isEnd = false;

    //追踪线程
    auto threadTrack = thread([&]() {
        ofstream outfile("outMemState.txt");          //输出重定向

        auto outputInfo = [&]() {

            /*//以下一段显示系统信息，每次操作后系统信息不变
            //如要查看系统信息，可以取消注释

            SYSTEM_INFO systemInfo;                  //获取系统信息
            GetSystemInfo(&systemInfo);
            //输出系统信息
            outfile << "dwActiveProcessorMask" << '\t' <<
systemInfo.dwActiveProcessorMask << endl;
            outfile << "dwAllocationGranularity" << '\t' <<
systemInfo.dwAllocationGranularity << endl;
            outfile << "dwNumberOfProcessors" << '\t' << systemInfo.dwNumberOfProcessors
<< endl;
            outfile << "dwOemId" << '\t' << systemInfo.dwOemId << endl;
            outfile << "dwPageSize" << '\t' << systemInfo.dwPageSize << endl;
            outfile << "dwProcessorType" << '\t' << systemInfo.dwProcessorType << endl;
            outfile << "lpMaximumApplicationAddress" << '\t' <<
systemInfo.lpMaximumApplicationAddress << endl;
            outfile << "lpMinimumApplicationAddress" << '\t' <<
systemInfo.lpMinimumApplicationAddress << endl;
            outfile << "wProcessorArchitecture" << '\t' <<
systemInfo.wProcessorArchitecture << endl;
            outfile << "wProcessorLevel" << '\t' << systemInfo.wProcessorLevel << endl;
            outfile << "wProcessorRevision" << '\t' << systemInfo.wProcessorRevision <<
endl;
            outfile << "wReserved" << '\t' << systemInfo.wReserved << endl;
```

```

        outfile <<
"*****"
<< endl;

    */

    MEMORYSTATUS status;                //获取内存状态数据
    GlobalMemoryStatus(&status);
    //输出内存信息
    outfile << "dwAvailPageFile" << '\t' << status.dwAvailPageFile << endl;
    outfile << "dwAvailPhys" << '\t' << status.dwAvailPhys << endl;
    outfile << "dwAvailVirtual" << '\t' << status.dwAvailVirtual << endl;
    outfile << "dwLength" << '\t' << status.dwLength << endl;
    outfile << "dwMemoryLoad" << '\t' << status.dwMemoryLoad << endl;
    outfile << "dwTotalPageFile" << '\t' << status.dwTotalPageFile << endl;
    outfile << "dwTotalPhys" << '\t' << status.dwTotalPhys << endl;
    outfile << "dwTotalVirtual" << '\t' << status.dwTotalVirtual << endl;
    outfile <<
"%%%%%%%%%"
<< endl;

    /* //以下一段显示内存基本信息，每次操作后内存基本信息不变
    //如要查看内存基本信息，可以取消注释
    MEMORY_BASIC_INFORMATION memBasicInfo; //内存基本信息
    VirtualQuery(systemInfo.lpMinimumApplicationAddress, &memBasicInfo,
        sizeof(MEMORY_BASIC_INFORMATION));
    outfile << "AllocationBase" << '\t' << memBasicInfo.AllocationBase << endl;
    outfile << "AllocationProtect" << '\t' << memBasicInfo.AllocationProtect <<
endl;

    outfile << "BaseAddress" << '\t' << memBasicInfo.BaseAddress << endl;
    outfile << "Protect" << '\t' << memBasicInfo.Protect << endl;
    outfile << "RegionSize" << '\t' << memBasicInfo.RegionSize << endl;
    outfile << "State" << '\t' << memBasicInfo.State << endl;
    outfile << "Type" << '\t' << memBasicInfo.Type << endl;
    outfile <<
"~~~~~"
~~~~~" << endl;

    */

};

while (!isEnd)

```



```
{
    // 等待 allocator 一次内存分配活动结束
    WaitForSingleObject(semaphoreTrack, INFINITE);
    outputInfo();
    // 释放信号量通知 allocator 可以执行下一次内存分配活动
    ReleaseSemaphore(semaphoreAlloc, 1, NULL);
}

outfile.close();
});

//模拟分配线程
auto threadAlloc = thread([&]() {
    ifstream in("opfile", ifstream::binary);

    Operation op;
    SYSTEM_INFO systemInfo;
    GetSystemInfo(&systemInfo);

    Trace traceArray[5];    //5 种对应权限

    const vector<DWORD> protection = {
        PAGE_READONLY,
        PAGE_READWRITE,
        PAGE_EXECUTE,
        PAGE_EXECUTE_READ,
        PAGE_EXECUTE_READWRITE,
        PAGE_READONLY
    };

    size_t index = 0;        //对应操作
    const vector<function<void(void)>> action = {
        //保留一个区域
        [&]()
        {
            cout << "Reserve Now" << endl;

            traceArray[index].start = VirtualAlloc(NULL, op.block *
systemInfo.dwPageSize,
                MEM_RESERVE, PAGE_NOACCESS);
            traceArray[index].size = op.block * systemInfo.dwPageSize;
            cout << "starting address:"
                << traceArray[index].start << '\t' << "size:" <<
traceArray[index].size << endl;
```

```
    },

    //提交一个区域
    [&]()
    {
        cout << "Commit Now" << endl;

        traceArray[index].start = VirtualAlloc(traceArray[index].start,
            traceArray[index].size, MEM_COMMIT, protection[op.protection]);
        cout << "starting address:"
            << traceArray[index].start << '\t' << "size:" <<
traceArray[index].size << endl;
    },

    //锁住一个区域
    [&]()
    {
        cout << "Lock Now" << endl;
        cout << "starting address:" << traceArray[index].start << '\t' <<
            "size:" << traceArray[index].size << endl;
        if (!VirtualLock(traceArray[index].start, traceArray[index].size))
        {
            cout << GetLastError() << endl; //GetLastError()函数返回错误号
        }
    },

    // 解锁一个区域
    [&]()
    {
        cout << "Unlock Now" << endl;
        cout << "starting address:" << traceArray[index].start <<
            '\t' << "size:" << traceArray[index].size << endl;
        if (!VirtualUnlock(traceArray[index].start, traceArray[index].size))
        {
            cout << GetLastError() << endl; //返回错误号
        }
    },

    // 回收一个区域
    [&]()
    {
        cout << "Decommit Now" << endl;
        cout << "starting address:" << traceArray[index].start << '\t' <<
            "size:" << traceArray[index].size << endl;
```

```
        if (!VirtualFree(traceArray[index].start, traceArray[index].size,
MEM_DECOMMIT))
        {
            cout << GetLastError() << endl;           //返回错误号
        }
    },

    // 释放一个区域
    [&]()
    {
        cout << "Release Now" << endl;
        cout << "starting address:" << traceArray[index].start << '\t' <<
            "size:" << traceArray[index].size << endl;
        if (!VirtualFree(traceArray[index].start, 0, MEM_RELEASE))
        {
            cout << GetLastError() << endl;           //返回错误号
        }
    }
};

try
{
    while (1)
    {
        //等待 tracker 打印结束的信号量
        WaitForSingleObject(semaphoreAlloc, INFINITE);

        in.read(reinterpret_cast<char *>(&op), sizeof(Operation));
        if (in.eof())
        {
            isEnd = true;
            break;
        }
        if (op.protection == 0)
            index = 0;
        this_thread::sleep_for(chrono::milliseconds(op.time));
        //0-保留; 1-提交; 2-锁; 3-解锁; 4-回收; 5-释放
        action[op.oper]();
        index++;

        //释放信号量通知 tracker 可以打印信息
        ReleaseSemaphore(semaphoreTrack, 1, NULL);
    }
}
catch (exception e)
```

```
{  
    cout << e.what() << endl;  
}  
  
in.close();    //关闭写入文件  
  
});  
  
if (threadAlloc.joinable())  
    threadAlloc.join();  
if (threadTrack.joinable())  
    threadTrack.join();  
  
CloseHandle(semaphoreAlloc);  
CloseHandle(semaphoreTrack);  
  
system("pause");  
return 0;  
}
```