操作系统实验三

Windows 虚拟内存分配问题

董凯杰

无 34

2013010572

2015 年 12 月 4 日

目录

一、 实验目的

通过对内存分配活动的模拟和跟踪的编程实现，从不同侧面对 Windows 对用户进程的虚拟内存空间的管理、分配方法，对 Windows 分配虚拟内存、改变内存状态，以及对物理内存和页面文件状态查询的 API 函数的功能、参数限制、使用规则进一步有所了解。同时了解跟踪程序的编写方法。

二、 实验题目

Windows 虚拟内存分配问题

三、 问题描述

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。

要求模拟的操作包括保留一个区域、提交一个区域、回收一个区域、释放一个区域，以及锁定与解锁一个区域。跟踪线程要输出每次内存分配操作后的内存状态。

四、 相关 API 函数功能

1. GetSystemInfo
   函数功能：返回当前系统信息, 存放入 lpSystemInfo 中。

2. 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。

3. VirtualQuery
   函数功能：查询进程地址空间中内存地址的信息.

4. 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。

5. 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。

6. VirtualLock
   函数功能：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。

7. VirtualUnlock
   函数功能：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。

五、 实验设计

设计思路

创建内存分配线程和跟踪线程，分配线程中依次执行保留、提交、锁定、解锁、回收、释放内存操作，同时在提交内存时改变提交页面的保护方式，观察不同之处。每执行依次操作，通过跟踪线程输出此时内存基本信息以及内存状态。

程序框架

```
//跟踪线程
DWORD WINAPI Tracker(LPVOID param)
{
        for (int i = 0;i < 30;i++)
        {
                WaitForSingleObject(TrackSemaphore, INFINITE); //请求trac信号量
                printMemoryStatus();   //打印内存状态信息
                printMemoryInfo(start); //打印内存基本信息
                ReleaseSemaphore(AllocatorSemaphore, 1, NULL); //释放allo信号量
        }
        return 0;
}
//内存分配线程
DWORD WINAPI Allocator(LPVOID param)
{
```

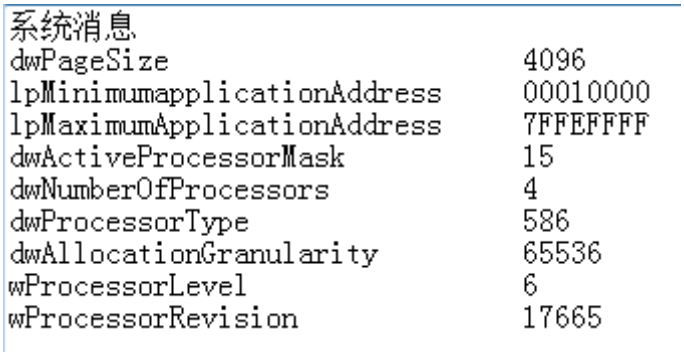
```
SYSTEM_INFO info;
GetSystemInfo (&info);
//5种保护方式
DWORD Protect[5] = {PAGE_READONLY, PAGE_READWRITE,
    PAGE_EXECUTE, PAGE_EXECUTE_READ,
    PAGE_EXECUTE_READWRITE};
string protect[5] = {"PAGE_READONLY", "PAGE_READWRITE", "
    PAGE_EXECUTE", "PAGE_EXECUTE_READ", "
    PAGE_EXECUTE_READWRITE"};
long size;
out<<"模拟内存分配开始: "<<endl;
printMemoryStatus();
for (int j = 0; j<30; j++)
{
        //请求allo信号量
        WaitForSingleObject(AllocatorSemaphore, INFINITE);
        int i = j % 6;
        int Pages = j / 6 + 1;
        if (i == 0)
        {
                out<<"保留内存"<<endl;
                start = VirtualAlloc(NULL, Pages * info.dwPageSize,
                    MEM_RESERVE, PAGE_NOACCESS);
                size = Pages * info.dwPageSize;
                out<<"起始地址: "<<start<<"\t"<<"内存大小: "<<size /
                    1024<<"KB"<<endl;
        }
        if (i == 1)
        {
                out<<"提交内存"<<endl;
                start = VirtualAlloc(start, size, MEM_COMMIT, Protect[
                    int(j / 6)]);
                out<<"起始地址: "<<start<<"\t"<<"内存大小: "<<size /
                    1024<<"KB"<<endl;
                out<<"保护级别: "<<protect[j / 6]<<endl;
        }
```

```
            if (i == 2)
            {
                out<<"锁定内存"<<endl;
                out<<"起始地址: "<<start<<"\t"<<"内存大小: "<<size /
                    1024<<"KB"<<endl;
                if (!VirtualLock(start, size))
                    out<<GetLastError()<<endl;
            }
            if (i == 3)
            {
                out<<"解锁内存"<<endl;
                out<<"起始地址: "<<start<<"\t"<<"内存大小: "<<size /
                    1024<<"KB"<<endl;
                if (!VirtualUnlock(start, size))
                    out<<GetLastError()<<endl;
            }
            if (i == 4)
            {
                out<<"回收内存"<<endl;
                out<<"起始地址: "<<start<<"\t"<<"内存大小: "<<size /
                    1024<<"KB"<<endl;
                if (!VirtualFree(start, size, MEM_DECOMMIT))
                    out<<GetLastError()<<endl;
            }
            if (i == 5)
            {
                out<<"释放内存"<<endl;
                out<<"起始地址: "<<start<<"\t"<<"内存大小: "<<size /
                    1024<<"KB"<<endl;
                if (!VirtualFree(start, 0, MEM_RELEASE))
                    out<<GetLastError()<<endl;
            }
            //释放trac信号量
            ReleaseSemaphore(TrackSemaphore, 1, NULL);
        }
        Sleep(1000);
```

```
        return 0;
}
```

六、 实验结果

使用 GetSystemInfo 函数，返回当前系统信息，如下：



系统消息	
dwPageSize	4096
lpMinimumapplicationAddress	00010000
lpMaximumApplicationAddress	7FFEFFFF
dwActiveProcessorMask	15
dwNumberOfProcessors	4
dwProcessorType	586
dwAllocationGranularity	65536
wProcessorLevel	6
wProcessorRevision	17665

可以看到页面大小为 4096Bytes（4KB），分配粒度为 64436Bytes（64KB），用户可访问的地址空间为 00010000——7FFEFFFF。

对内存进行模拟分配，利用 GlobalMemoryStatus 和 VirtualQuery 函数查看内存相关信息，如下：

```
程序开始：
模拟内存分配开始：
内存状态
dwAvailPhys     948744KB
dwAvailPageFile 3246268KB
dwAvailVirtual  2080832KB
*****************************************
保留内存
起始地址：00030000      内存大小：4KB
内存状态
dwAvailPhys     948632KB
dwAvailPageFile 3246220KB
dwAvailVirtual  2080828KB
内存基本信息
AllocationProtect       PAGE_NOACCESS
RegionSize      4KB
State   MEM_RESERVE
Protect does not have access
*****************************************
提交内存
起始地址：00030000      内存大小：4KB
保护级别：PAGE_READONLY
内存状态
dwAvailPhys     948320KB
dwAvailPageFile 3246220KB
dwAvailVirtual  2080828KB
内存基本信息
AllocationProtect       PAGE_NOACCESS
RegionSize      4KB
State   MEM_COMMIT
Protect PAGE_READONLY
*****************************************
```

```
锁定内存
起始地址：00030000      内存大小：4KB
内存状态
dwAvailPhys     948320KB
dwAvailPageFile 3246220KB
dwAvailVirtual  2080828KB
内存基本信息
AllocationProtect       PAGE_NOACCESS
RegionSize      4KB
State   MEM_COMMIT
Protect PAGE_READONLY
*********************************************
解锁内存
起始地址：00030000      内存大小：4KB
内存状态
dwAvailPhys     948320KB
dwAvailPageFile 3246220KB
dwAvailVirtual  2080828KB
内存基本信息
AllocationProtect       PAGE_NOACCESS
RegionSize      4KB
State   MEM_COMMIT
Protect PAGE_READONLY
*********************************************
回收内存
起始地址：00030000      内存大小：4KB
内存状态
dwAvailPhys     948324KB
dwAvailPageFile 3246220KB
dwAvailVirtual  2080828KB
内存基本信息
AllocationProtect       PAGE_NOACCESS
RegionSize      4KB
State   MEM_RESERVE
Protect does not have access
*********************************************
释放内存
起始地址：00030000      内存大小：4KB
内存状态
dwAvailPhys     948316KB
dwAvailPageFile 3246220KB
dwAvailVirtual  2080832KB
内存基本信息
AllocationProtect       does not have access
RegionSize      64KB
State   MEM_FREE
Protect PAGE_NOACCESS
*********************************************
```

可以看到经过保留内存操作后，可用的虚拟内存大小减少了 4KB，而在之后的四种操作方式都没有改变，直到进行释放操作，虚拟内存大小恢复到初始大小。由于 GlobalMemoryStatus 函数查看的是全局内存信息，因此物理内存的测量会有不准确。但理论情况应该是在保留内存后不改变物理内存大小，在提交内存后物理内存减小相应的

大小，在回收内存后物理内存恢复，释放时不改变。

其次我们也可以看到，内存页的状态也会随着不同操作而改变，在保留操作后显示为 MEM_RESERVE，在释放操作后显示为 MEM_FREE，其余均为 MEM_COMMIT。

保护方式也是在提交内存后发生了预想中的变化。

七、 思考题

1. 分析不同参数的设置对内存分配的结果影响。
   这 6 种内存分配方式均有起始地址和内存大小这两个参数，也就是对哪一块内存进行操作。不同之处，保留操作需要设置（MEM_RESERVE，PAGE_NOACCESS）参数，提交操作需要设置（MEM_COMMIT）以及相应的保护方式，回收操作需要设置（MEM_DECOMMIT）参数，释放操作需要将大小设为 0，否则都将会报错。

2. 如果调换分配、回收、内存复位、加锁、解锁、提交、回收的次序，会有什么变化，并分析原因。
   会造成用户程序无法正常存储于存储器中，从而无法运行。程序在计算机内必须进入主存中才能正常运行。而装入主存前，首先要为其先分配一个存储空间。因此调换为上述顺序后将不能正常运行。

3. 在 Linux 操作系统上没有提供系统调用来实现“以页为单位的虚拟内存分配方式”，如果用户希望实现这样的分配方式，可以怎样做？解释你的思路。
   利用地址空间映射的方式手工模拟以页为单位的虚拟内存分配方式。

八、 实验总结

本次实验也用到了线程的相关 api，不过由于前两次实验的练习，这块内容变得比较简单。而这次实验的难点在于与虚拟内存分配的相关 api 比较难理解，同时也很难理解所显示的内存状态信息，每一个数据的具体含义。但经过不断的查阅资料，我对于这些函数以及内存状态的相关知识有了更深入的了解。最后也感谢马老师在程序上的一些指导。