```
#include <windows.h>
#include <stdio.h>
#include <process.h>
#include <time.h>

// 声明两个线程函数
unsigned _stdcall simulator(void *);
unsigned _stdcall inspector(void *);
LPVOID BASE_PTR;        // 无类型指针
int Actnum=0;           // 指示器

// 主要是用于 启动模拟虚存活动和进行监控的两个线程
int main(int argc,  char* argv[])
{
    unsigned ThreadID[2];
    int i=0;
    // 创建线程 simulator inspector
    _beginthreadex(NULL, 0, simulator, NULL, 0, &ThreadID[0]);
    _beginthreadex(NULL, 0, inspector, NULL, 0, &ThreadID[1]);
    getchar();// 读取字符，输入回车即可终止程序
    return 0;
}


// 模拟一系列的虚存活动，作为一个独立的线程运行
unsigned _stdcall simulator(void *)
{
    DWORD OldProtect;
    int randnum;
    printf("Now the simulator procedure has been started.\n");
    // 产生一个随机数种子
    srand((unsigned)time(NULL));

    randnum=-1;

    // 在一个死循环中，用随机数控制，不断进行虚存操作活动
    while(1)
    {
        Sleep(500);//控制整个模拟和监控的速度
        while(Actnum!=0)
        {
            Sleep(500);// 等待，直到监控线程捕捉到上一个模拟动作后跳出循环，再继续下一个动作
        }

        randnum=7&rand();
        switch(randnum) // 各个动作的虚存指针均使用BASE_PTR:
                        // 它在过程中由虚存分配函数VirtualAlloc动态调整
                        // 如果某动作分配不成功，则不会被监控线程监控到
        {
```

```
            // 分别给指示器赋不同的值，表示执行不同的动作
        case 0:
            // 保留或提交某一范围的虚拟地址
            if (BASE_PTR=VirtualAlloc(NULL, 1024*32, MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE))
            {
                Actnum=1;   // 虚存的保留与提交
            }
            break;
        case 1:
            // 释放提交的虚存或者释放被保留或者提交的进程虚拟地址空间
            if (VirtualFree(BASE_PTR, 1024*32, MEM_DECOMMIT))
            {
                Actnum=2;   // 虚存的注销
            }
            break;
        case 2:
            if (VirtualFree(BASE_PTR, 0, MEM_RELEASE))
            {
                Actnum=3;   // 虚存的注销并释放虚存空间
            }
            break;
        case 3:
            // 改变虚拟内存页的保护方式
            if (VirtualProtect(BASE_PTR, 1024*32, PAGE_READONLY, &OldProtect))
            {
                Actnum=4;
            }
            break;
        case 4:
            // 锁定虚拟内存页
            if (VirtualLock(BASE_PTR, 1024*12))
            {
                Actnum=5;
            }
            break;
        case 5:
            // 虚存的保留
            if (BASE_PTR=VirtualAlloc(NULL, 1024*32, MEM_RESERVE, PAGE_READWRITE))
            {
                Actnum=6;
            }
            break;
        default:
            break;
        }
    }
    return 0;
}
```

```
// 通过一个全局变量来监视另一模拟线程的模拟活动，并通过适当的信息查询函数，将存储的使用和活动情况打出报告
unsigned _stdcall inspector(void *)
{
    char para1[3000];
    char tempstr[100];

    // 存储系统的统计指标
    MEMORYSTATUS Vmeminfo;
    MEMORY_BASIC_INFORMATION inspectorinfo1;

    int QuOut=0;
    int structsize = sizeof(MEMORY_BASIC_INFORMATION);

    printf("Hi,  now inspector begin to work\n");

    // 在一个死循环中不断通过一个全局变量（监视器），来监控模拟线程是否有新的动作
    // 如果有，通过的API函数将相应虚存处（通过共用BASE_PTR实现）的信息进行检查
    // 从而验证该动作对存储使用的影响
    while (1)
    {
        Sleep(1000);
        if (Actnum != 0)
        {
            // 通过全局变量（监视器/指示器）Actnum来获取上一个虚存动作的类型
            // 及相应构造监控信息的头部
            switch (Actnum)
            {
            // 根据Actnum值得出相应动作后的存储使用信息
            case 1:
                memset(&inspectorinfo1, 0, structsize);
                // 查询一个进程的虚拟内存，inspectorinfo1用于接收查询到的信息
                VirtualQuery((LPVOID)BASE_PTR, &inspectorinfo1, structsize);
                // 连接字符串函数，将输出都保存在para1数组中
                strcpy(para1, "目前执行动作：虚存的保留与提交\n");
                break;
            case 2:
                memset(&inspectorinfo1, 0, structsize);
                VirtualQuery((LPVOID)BASE_PTR, &inspectorinfo1, structsize);
                strcpy(para1, "目前执行动作：虚存的除配\n");
                break;
            case 3:
                memset(&inspectorinfo1, 0, structsize);
                VirtualQuery((LPVOID)BASE_PTR, &inspectorinfo1, structsize);
                strcpy(para1, "目前执行动作：虚存的除配并释放虚存空间\n");
                break;
            case 4:
                memset(&inspectorinfo1, 0, structsize);
                VirtualQuery((LPVOID)BASE_PTR, &inspectorinfo1, structsize);
                strcpy(para1, "目前执行动作：改变虚存内存页的保护\n");
```

```
        break;
    case 5:
        memset(&inspectorinfo1, 0, structsize);
        VirtualQuery((LPVOID)BASE_PTR, &inspectorinfo1, structsize);
        strcpy(para1, "目前执行动作：锁定虚存内存页\n");
        break;
    case 6:
        memset(&inspectorinfo1, 0, structsize);
        VirtualQuery((LPVOID)BASE_PTR, &inspectorinfo1, structsize);
        strcpy(para1, "目前执行动作：虚存的保留\n");
        break;
    default:
        break;
    }

    // 实时显示固定格式的相关材料；通过目前监控到的动作所发生的虚存地址
    // 监控该活动对相应存储空间的影响
    // 输出完表头之后，输出内存的使用情况，（不同的操作对存储使用的影响）如下：
    // sprintf 字符串格式化命令
    sprintf(tempstr, "开始地址:0X%x\n", inspectorinfo1.BaseAddress);
    strcat(para1, tempstr);

    sprintf(tempstr, "区块大小:0X%x\n", inspectorinfo1.RegionSize);
    strcat(para1, tempstr);

    sprintf(tempstr, "目前状态:0X%x\n", inspectorinfo1.State);
    strcat(para1, tempstr);

    sprintf(tempstr, "分配时访问保护:0X%x\n", inspectorinfo1.AllocationProtect);
    strcat(para1, tempstr);

    sprintf(tempstr, "当前访问保护:0X%x\n", inspectorinfo1.Protect);
    strcat(para1, tempstr);

    strcat(para1, "(状态:10000代表未分配；1000代表提交；2000代表保留；)\n");
    strcat(para1, "(保护方式：0代表其它；1代表禁止访问；2代表只读；4代表读写;\n10代表可执");
    strcat(para1, "行;20代表可读和执行;40代表可读写和执行);\n**************************\n");

    // 全局信息，报告目前系统和当前进程的存储使用总体情况
    // 输出整个系统的信息
    GlobalMemoryStatus(&Vmeminfo);
    strcat(para1, "当前整体存储统计如下\n");
    sprintf(tempstr, "物理内存总数：%d(BYTES)\n", Vmeminfo.dwTotalPhys);
    strcat(para1, tempstr);
    sprintf(tempstr, "可用物理内存：%d(BYTES)\n", Vmeminfo.dwAvailPhys);
    strcat(para1, tempstr);
    sprintf(tempstr, "页面文件总数：%d(BYTES)\n", Vmeminfo.dwTotalPageFile);
    strcat(para1, tempstr);
    sprintf(tempstr, "可用页面文件数：%d(BYTES)\n", Vmeminfo.dwAvailPageFile);
```

```
            strcat(para1, tempstr);
            sprintf(tempstr, "虚存空间总数：%d(BYTES)\n", Vmeminfo.dwTotalVirtual);
            strcat(para1, tempstr);
            sprintf(tempstr, "可用虚存空间数：%d(BYTES)\n", Vmeminfo.dwAvailVirtual);
            strcat(para1, tempstr);
            sprintf(tempstr, "物理存储使用负荷：%%%d\n\n\n\n", Vmeminfo.dwMemoryLoad);
            strcat(para1, tempstr);
            printf("%s", para1);// 输出para1数组所有报告内容

            Actnum = 0;// 通知模拟线程可以进行下一个模拟动作
            Sleep(500);// 调节模拟和监控的总体速度
        }
    }
    return 0;
}
```

---

```
// 小程序：
#include <windows.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
    MEMORYSTATUS Vmeminfo;
    GlobalMemoryStatus(&Vmeminfo);
    printf("当前系统存储空间使用概况\n");
    printf("物理内存总数:%d(BYTES)\n", Vmeminfo.dwTotalPhys);
    printf("可用物理内存: %d(BYTES)\n", Vmeminfo.dwAvailPhys);
    printf("页面文件总数:%d(BYTES)\n", Vmeminfo.dwTotalPageFile);
    printf("可用页面文件数:%d(BYTES)\n", Vmeminfo.dwAvailPageFile);
    printf("虚存空间总数:%d(BYTES)\n", Vmeminfo.dwTotalVirtual);
    printf("可用虚存空间数:%d(BYTES)\n", Vmeminfo.dwAvailVirtual);
    printf("物理存储使用负荷:%%%d\n\n\n\n", Vmeminfo.dwMemoryLoad);
    getchar();
    return 0;
}
```