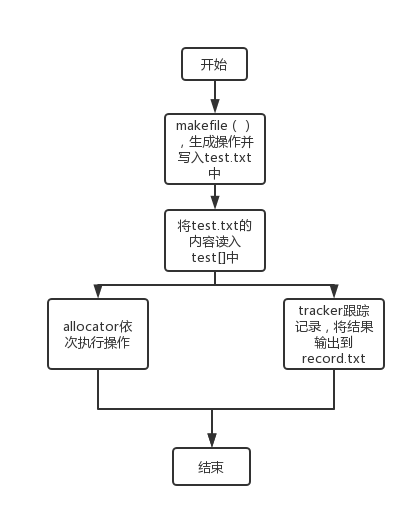
**操作系统课程设计报告** 日期： 5.29

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实习题目：内存管理 | | | | |
| 完成人 | 姓名： | 课序号： | 班级： | 学号： |
| 实习内容及要求简要描述 | 使用Windows 2000／XP的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，而且要求两个线程之间通过信号量实现同步。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。内存操作包括保留(reserve)一个区域、提交(commit)一个区域、释放(release)一个区域、回收(decommit)一个区域和加锁(10ck)与解锁(unlock)一个区域，可以将这些操作编号存放于文件。 | | | |
| 主要代码结构  （附注释） | （1）头文件  #include<windows.h>  #include<stdio.h>  #include<process.h>  #include<time.h>  #include<thread>  #include<iostream>  #include<fstream>  #include<string>  #include<mutex>   1. 宏定义   无  （3）全局变量  mutex allo,trac;//互斥量  int number=0;//样例数量  int return\_num=0;//操作执行结果  LPVOID BASE\_PTR;//基址指针  operate test[10];//存测试样例   1. 函数声明 2. makefile();   随机生成操作以及所需参数，将按结构体内容输出到test.txt中，以便之后读取。   1. allo\_func();   线程allocator的功能函数，主要执行以下内容：  a)等待Tracker输出的结束(即等待信号量allo的释放)。  b)读文件(makefile.cpp的输出结果)。  c)根据文件内容(protectlon)确定对内存操作时的权限。  d)根据文件内容(oper)确定对内存的具体操作。  e)释放信号量(trac)通知Tracker线程可以进行一次输出。  f)如果文件中所有的分配信息已经完成，线程退出，否则转到a)。  3）trac\_func();  线程tracker的功能函数，主要执行以下内容：  a)打开文件，准备输出。  b)等待线程Allocator的一次内存操作完毕(即等待信号量trac的释放)。  c)用函数GetSystemlnfo得到系统信息(该信息不随内存分配的变化而变化)。  d)用函数GlobalMemoryStatus得到内存信息(随内存的分配各项信息会有所改变)。用函数VirtualQuery得到虚拟内存基本信息(该信息不随内存分配的变化而变化)。  e)释放信号量，通知Allocator线程可以进行下一次内存分配活动。  f)如果已经记录了所有的内存分配信息，线程退出，否则转到b)。 | | | |
| 结果分析（或错误原因分析） | 首先调用makefile函数，可以看到操作内容成功写到test.txt中，然后创建线程allocator,tracker并启动，顺利结束后可以看到record.txt中正确记录了每次内存操作的相关信息。综合来看，程序执行正确无误。 | | | |

附录一、流程图



附录二、源程序

#include <windows.h>

#include <stdio.h>

#include <process.h>

#include <time.h>

#include<thread>

#include<iostream>

#include<fstream>

#include<string>

#include<mutex>

using namespace std;

mutex allo,trac;//互斥量

int number=0;//样例数量

int return\_num=0;//操作执行结果

LPVOID BASE\_PTR;//基址指针

int finish=1;//tractor是否结束

struct operate{//一个操作所需的参数

        int delay;//等待时间

        int size;//空间大小

        string op;//操作

        DWORD right;//权限

    };

operate test[10];//存测试样例

void allo\_func();

void trac\_func();

void makefile();

int main(){

    makefile();

    //读入文件内容

    fstream f;

    f.open("test.txt",ios::in);

    int delay=0,size=0;

    string op=" ";

    DWORD right=0;

    while(f>>delay>>size>>op>>right){

        test[number]={delay,size,op,right};

        number++;

    }

    f.close();

    trac.lock();//提前锁住，确保allocator先开始工作

    thread allocator(allo\_func);

    thread tracker(trac\_func);

    allocator.join();

    tracker.join();

   // system("pause");

    return 0;

}

void makefile(){

    fstream f;

    f.open("test.txt",ios::out);

    f.clear();

    operate te;

    int number=5;//操作次数

    int delay=0,size=0,m=0;

    string op=" ";

    DWORD right=0;

    srand(time(0));//随机数种子

    for(int i=0;i<number;i++){

        delay=(rand()%5)+1;

        size=(rand()%3)+1;

        m=rand()%6;

        switch(m){

            case 0:op="reserve";break;

            case 1:op="commit";break;

            case 2:op="release";break;

            case 3:op="decommit";break;

            case 4:op="lock";break;

            case 5:op="unlock";break;

            default:break;

        }

        m=rand()%5;

        switch(m){

            case 0:right=PAGE\_READONLY;break;

            case 1:right=PAGE\_READWRITE;break;

            case 2:right=PAGE\_EXECUTE;break;

            case 3:right=PAGE\_EXECUTE\_READ;break;

            case 4:right=PAGE\_EXECUTE\_READWRITE;break;

            default:break;

        }

        te={delay,size,op,right};

        f<<te.delay<<" "<<te.size<<" "<<te.op<<" "<<te.right<<'\n';

    }

    f.close();

}

void allo\_func(){

    cout<<"allocator begin to work\n";

    for(int i=0;i<number;i++){

        Sleep(1000\*test[i].delay);

        allo.lock();

        if(test[i].op=="unlock"){//解锁

            if(VirtualUnlock(BASE\_PTR,1024\*test[i].size))

                return\_num=6;

        }

        else if(test[i].op=="reserve"){//保留

            if (BASE\_PTR=VirtualAlloc(NULL, 1024\*32, MEM\_RESERVE, test[i].right))

            {

                return\_num=1;

            }

        }

        else if(test[i].op=="commit"){//提交

            if (BASE\_PTR=VirtualAlloc(NULL, 1024\*test[i].size, MEM\_RESERVE|MEM\_COMMIT, test[i].right))

            {

                return\_num=2;

            }

        }

        else if(test[i].op=="release"){//释放

            if (VirtualFree(BASE\_PTR, 0, MEM\_RELEASE))

            {

                return\_num=3;

            }

        }

        else if(test[i].op=="decommit"){//注销

            if (VirtualFree(BASE\_PTR, 1024\*test[i].size, MEM\_DECOMMIT))

            {

                return\_num=4;

            }

        }

        else if(test[i].op=="lock"){//上锁

            if (VirtualLock(BASE\_PTR, 1024\*test[i].size))

            {

                return\_num=5;

            }

        }

        trac.unlock();

    }

    Sleep(1000);

    cout<<"allocator finish\n";

}

void trac\_func(){

    char para1[3000];

    char tempstr[100];

    MEMORYSTATUS Vmeminfo;

    MEMORY\_BASIC\_INFORMATION inspectorinfo1;

    int QuOut=0;

    int structsize = sizeof(MEMORY\_BASIC\_INFORMATION);

    fstream fi;

    fi.open("record.txt",ios::out);

    fi.clear();

    cout<<"tracker begin to work\n";

    int i=0;

    while(i<number)

    {

        trac.lock();

        if(return\_num!=0)

        {

            switch(return\_num)

            {

            case 1:

                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;

            case 2:

                memset(&inspectorinfo1, 0, structsize);

                VirtualQuery((LPVOID)BASE\_PTR, &inspectorinfo1, structsize);

                strcpy(para1, "目前执行动作：虚存的提交\n");

                break;

            case 3:

                memset(&inspectorinfo1, 0, structsize);

                VirtualQuery((LPVOID)BASE\_PTR, &inspectorinfo1, structsize);

                strcpy(para1, "目前执行动作：虚存的释放\n");

                break;

            case 4:

                memset(&inspectorinfo1, 0, structsize);

                VirtualQuery((LPVOID)BASE\_PTR, &inspectorinfo1, structsize);

                strcpy(para1, "目前执行动作：虚存的注销\n");

                break;

            case 5:

                memset(&inspectorinfo1, 0, structsize);

                VirtualQuery((LPVOID)BASE\_PTR, &inspectorinfo1, structsize);

                strcpy(para1, "目前执行动作：虚存的锁定\n");

                break;

            case 6:

                memset(&inspectorinfo1, 0, structsize);

                VirtualQuery((LPVOID)BASE\_PTR, &inspectorinfo1, structsize);

                strcpy(para1, "目前执行动作：虚存的解锁\n");

                break;

            default:

                break;

            }

            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");

            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);

            fi<<para1;

            return\_num=0;

        }

        allo.unlock();

        i++;

    }

    fi.close();

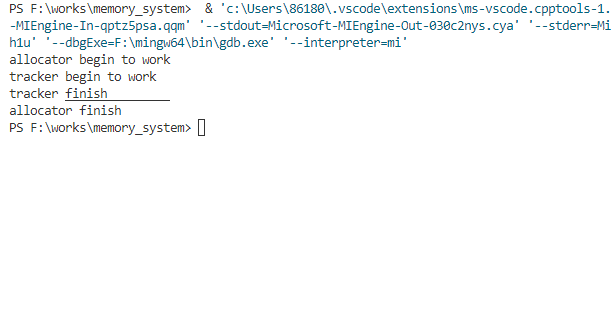
    Sleep(1000);

    cout<<"tracker finish\n";

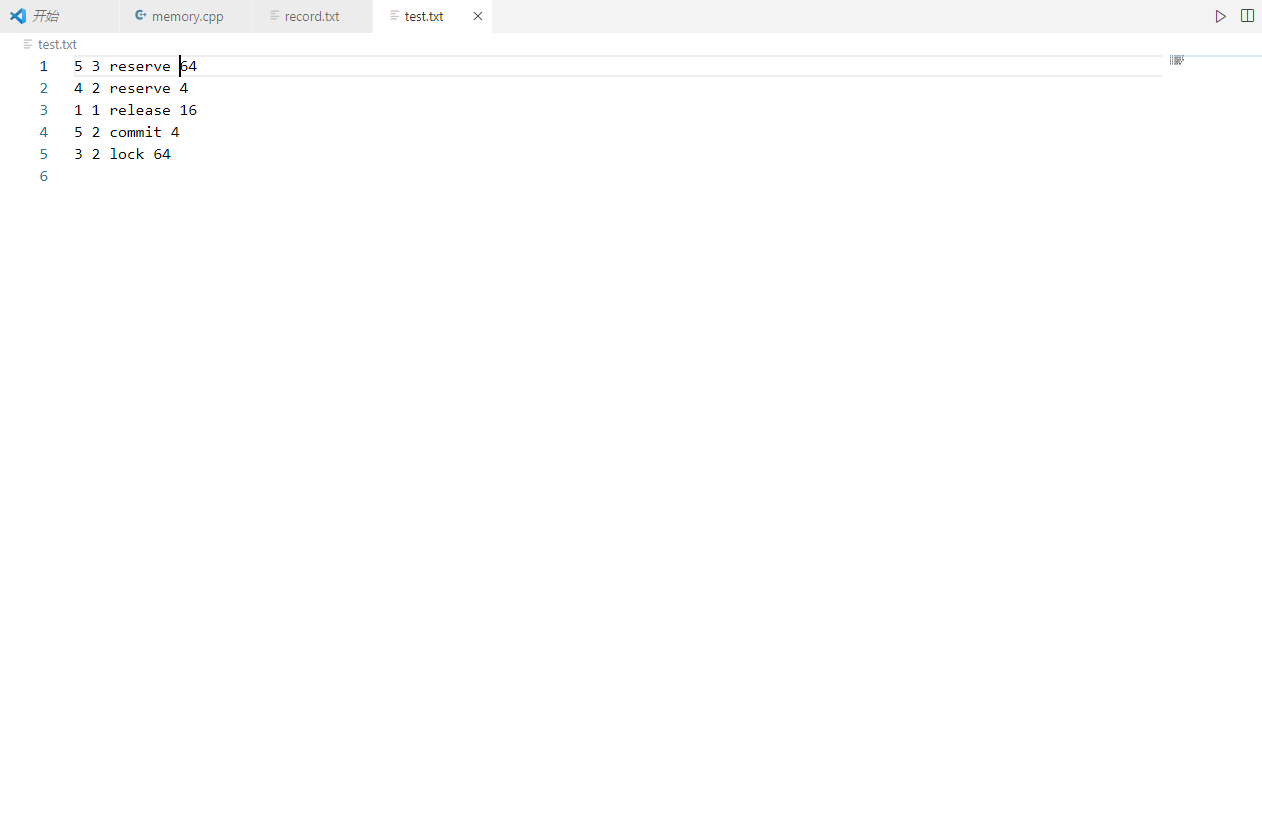
}

附录三、运行结果截屏

运行程序，allocator线程、tracker线程依次开始执行，然后结束。



Makefile函数，随机生成操作并写入test.txt文件中



tracker将跟踪记录的信息输出到record.txt文件中

