实验三 内存监视

班级： 07112201 学号： 1120220715 姓名： 刘秉致

一、实验目的

独立设计并实现一个内存监视器，以加深对内存管理的理解。

二、实验内容

在Windows系统下设计实现一个内存监视器，使用该内存监视器:

1. 能够实时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

2. 能够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、实验步骤

本次实验采用C++基于Windows API实现内存监视器的基本功能，并显示在终端。

考虑到该内存监视器具有两个主要功能，因此将内存监视器划分为两个工作模式：

1. 系统内存信息监视模式

在该模式下，内存监视器将获取当前时刻系统全局的内存使用情况并显示，之后系统休眠并清除终端上的显示信息，再次刷新信息。

///@brief:调用Windows API实现系统内存信息获取的函数

void DisplayGlobalMemoryInfo(){

    MEMORYSTATUSEX meminfo;

    meminfo.dwLength = sizeof(meminfo);

    if (GlobalMemoryStatusEx(&meminfo)) {

        cout << "[系统内存信息]" << endl;

        cout << "当前使用百分比：" << meminfo.dwMemoryLoad << "%" << endl;

        cout << "总物理内存：" << meminfo.ullTotalPhys / (1024 \* 1024) << "MB" << endl;

        cout << "当前可用物理内存：" << meminfo.ullAvailPhys / (1024 \* 1024) << "MB" << endl;

        cout << "总虚拟内存：" << meminfo.ullTotalVirtual / (1024 \* 1024) << "MB" << endl;

        cout << "当前可用虚拟内存：" << meminfo.ullAvailVirtual / (1024 \* 1024) << "MB" << endl;

    }

    else {

        cerr << "获取系统内存信息失败！" << endl;

    }

}

1. 单进程内存信息监视模式

在该模式下，内存监视器将获取当前时刻系统中正在运行的进程并显示，之后用户指定进程ID，监视器获取该进程的内存地址空间布局和工作集信息并休眠，等待刷新。

///@brief 获取系统中运行的进程的函数

void DisplayProcessInfo(){

    cout << "[系统中运行的进程]" << endl;

    DWORD Processes[1024], returnBytes;

    if (!EnumProcesses(Processes, sizeof(Processes), &returnBytes)) {

        cerr << "获取进程列表失败！" << endl;

        return;

    }

    int ProcessCount = returnBytes / sizeof(DWORD);

    int viableProcessCount = 0;

    for (int i = 0; i < ProcessCount; i++) {

        if (Processes[i] != 0) {

            HANDLE hProcess = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, Processes[i]);

            if (hProcess) {

                wchar\_t processName[MAX\_PATH] = L"unknown";

                HMODULE hMod;

                DWORD cbNeeded;

                if (EnumProcessModules(hProcess, &hMod, sizeof(hMod), &cbNeeded)) {

                    GetModuleBaseNameW(hProcess, hMod, processName, sizeof(processName) / sizeof(wchar\_t));

                    wcout << Processes[i]<<" "<< processName << endl;

                    viableProcessCount++;

                }

            }

            CloseHandle(hProcess);

        }

    }

    cout<<"共计"<<viableProcessCount<<"个可访问进程"<<endl;

    return;

}

///@brief 根据进程句柄（ID）获得内存地址空间布局的函数

void DisplayMemoryLayout(HANDLE processHandle){

    SYSTEM\_INFO sysinfo;

    GetSystemInfo(&sysinfo);

    cout << "[进程地址空间布局]" << endl;

    cout << "系统地址空间起始地址：" << sysinfo.lpMinimumApplicationAddress << endl;

    cout << "系统地址空间结束地址：" << sysinfo.lpMaximumApplicationAddress << endl;

    MEMORY\_BASIC\_INFORMATION meminfo;

    PVOID baseAddress = sysinfo.lpMinimumApplicationAddress;

    while (baseAddress < sysinfo.lpMaximumApplicationAddress) {

        if (VirtualQueryEx(processHandle, baseAddress, &meminfo, sizeof(meminfo)) == 0) {

            DWORD dwError = GetLastError();

            cerr << "获取内存信息失败！错误代码：" << dwError << endl;

            return;

        }

        cout << "基地址：" << meminfo.BaseAddress << endl;

        cout << "区域大小：" << meminfo.RegionSize / 1024 << "KB" << endl;

        if (meminfo.State == MEM\_COMMIT) {

            cout << "状态：已提交" << endl;

        } else if (meminfo.State == MEM\_FREE) {

            cout << "状态：空闲" << endl;

        } else if (meminfo.State == MEM\_RESERVE) {

            cout << "状态：保留" << endl;

        }

        baseAddress = (PVOID)((long long)meminfo.BaseAddress + meminfo.RegionSize);

    }

}

///@brief 根据句柄获得工作集信息的函数

void DisplayWorkingSetInfo(HANDLE processHandle) {

    PSAPI\_WORKING\_SET\_INFORMATION\* wsInfo=(PSAPI\_WORKING\_SET\_INFORMATION\*)malloc(1024\*1024);

    if(!wsInfo){

        cerr<<"内存分配失败！"<<endl;

        return;

    }

    if (QueryWorkingSet(processHandle, wsInfo, 1024\*1024)) {

        cout << "[工作集信息]" << endl;

        cout << "工作集内物理页数: " << wsInfo->NumberOfEntries << endl;

        cout << "工作集大小: " << (wsInfo->NumberOfEntries \* 4096) / (1024 \* 1024) << "MB" << endl;

    } else {

        DWORD dwError = GetLastError();

        cerr << "无法获取工作集信息! 错误代码：" << dwError << endl;

    }

}

1. 主模块

该模块主要实现工作模式的选择和信息的自动化刷新。

int main(){

    system("chcp 65001");

    int control=-1;

    cout<<"请选择您的使用方式："<<endl;

    cout<<"0.查看系统内存信息"<<endl;

    cout<<"1.查看进程内存信息"<<endl;

    cin>>control;

    system("cls");

    if(control==0){

        while(true){

            DisplayGlobalMemoryInfo();

            Sleep(2000);

            system("cls");

        }

        return 0;

    }

    else if(control==1){

        DisplayProcessInfo();

        cout << "请输入进程ID：";

        DWORD processID;

        cin >> processID;

        system("cls");

        HANDLE processHandle = OpenProcess(PROCESS\_QUERY\_INFORMATION | PROCESS\_VM\_READ, FALSE, processID);

        if (processHandle == NULL) {

            DWORD dwError = GetLastError();

            cerr << "打开进程失败！错误代码：" << dwError << endl;

            return -1;

        }

        while(true){

            DisplayMemoryLayout(processHandle);

            Sleep(3000);

            system("cls");

            DisplayWorkingSetInfo(processHandle);

            Sleep(3000);

            system("cls");

        }

        CloseHandle(processHandle);

        return 0;

    }

    else{

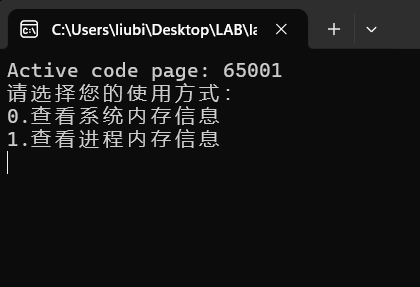
        cerr<<"无效的输入！"<<endl;

        return -1;

    }

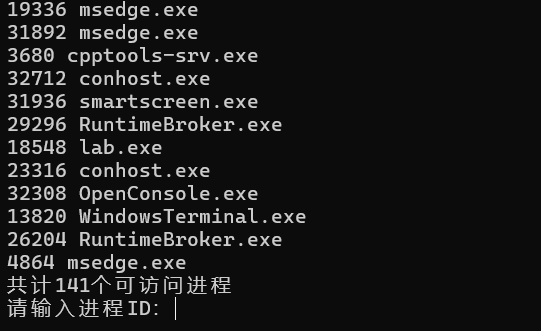
}

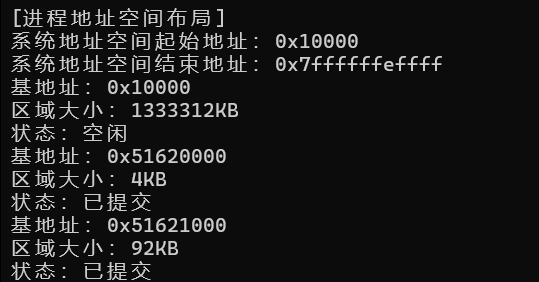
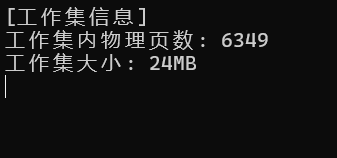
四、实验结果及分析

运行程序，进入选择页面

1. 进入系统内存信息监视模式

观察到系统内存使用信息正常输出并且以一定频率不断刷新

1. 单进程内存信息监视模式

进程列表正常输出，并等待用户输入目标进程ID，这里以4864进程为例

观察到系进程地址空间布局信息和工作集信息正常输出并且以一定频率不断刷新。

五、实验收获与体会

本次实验，我通过自行查询资料并学习，了解了如何在Windows环境下调用Windows API实现对进程和系统的内存信息进行访问，进一步加深了对操作系统内存管理相关知识的了解。

过程中我主要出现了以下问题：

1. 使用wcout输出宽字符到控制台时，不支持中文，这个我通过查询相关论坛，通过设置程序使用英文字符规避了问题
2. 使用cmd作为显示终端时，中文输出为乱码，这是因为cmd默认使用GBK输出简体中文，变更编码方式为utf-8即可
3. 使用OpenProcess函数时，出现了部分进程无法打开的问题，查询后得知这部分进程为系统进程，需要指定更高的访问权限来进行访问

附录：程序清单及说明

1.lab.cpp：实验程序源代码