实验二 内存管理

学号: 2017526019 姓名: 刘禾子 班级: 2015211307

一、 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。主要了解以下几个方面:

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

二、 Windows 2000/XP 虚拟内存机制简介

内存管理是 Windows 2000/XP 执行体的一部分,位于 Ntoskrnl. exe 文件中,是整个操作系统的重要组成部分。

默认情况下,32 位 Windows 2000/XP 上每个用户进程可以占有2GB的私有地址控件,操作系统占有剩下的2GB。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。页面大小及页面表项的宽度决定了页目录和页表索引的宽度。比如,在x86系统中,因为一页包含4096字节,于是字节索引被确定为12位宽(2¹²⁼⁴⁰⁹⁶)。

应用程序有三种使用内存方法:

- 以页为单位的虚拟内存分配方法,适合于大型对象或结构数组;
- 内存映射文件方法,适合于大型数据流文件以及多个进程之间的数据共享:
- 内存堆方法,适合于大量的小型内存申请。

本次实验主要是针对第一种使用方式。应用程序通过 API 函数 VirtualAlloc 和 VirtualAllocEx 等实现以页为单位的虚拟内存分配方法。首先保留地址空间,然后向此地址空间提交物理页面,也可以同时保留和提交。保留地址空间是为线程将来使用保留一块虚拟地址。在已保留的区域中,提交页面必须指出将物理存储器提交到何处以及提交多少。提交页面在访问时会转变为物理内存中的有效页面。

三、 实验内容

1. 与实验相关的 API

可以通过 GetSystemInfo, GlobalMemoryStatus 和 VirtualQuery 来查询进程虚空间的状态。主要的信息来源如下:

VOID GetSystemInfo (LPSYSTEM_INFO lpSystemInfo); 结构 SYSTEMINFO 定义如下:

typedef struct SYSTEM INFO {

DWORD dwOemld;

DWORD dwPageSize;

LPVOID lpMinimumApplicationAddress;

LPVOID 1pMaximumApplicationAddress;

```
DWORD dwActiveProcessorMask;
DWORD dwNumberOfProcessors:
DWORD dwProcessorType;
DWORD dwAllocationGranularity;
DWORD dwReserved;
} SYSTEM_INFO, *LPSYSTEM_INFO;
函数 VOID GlobalMemoryStatus (LPMEMORYSTATUS lpBuffer);
数据结构 MEMORYSTATUS 定义如下:
typedef struct MEMORYSTATUS {
DWORD dwLength;
DWORD dwMemoryLoad;
DWORD dwTotalPhys;
DWORD dwAvailPhys;
DWORD dwTotalPageFile;
DWORD dwAvailPageFile;
DWORD dwTotalVirtual:
DWORD dwAvailVirtual;
} MEMORYSTATUS. * LPMEMORYSTATUS:
函数 DWORD VirtualQuery (LPCVOLD lpAddress,
PMEMORY_BASIC_INFORMATION lpBuffer, DWORD dwLength);
主要数据结构 MEMORY BASIC INFORMAT 定义如下:
typedef struct _ MEMORY_BASIC_INFORMATION {
PVOID BaseAddress:
PVOID AllocationBase:
DWORD AllocationProtect;
DWORD RegionSize;
DWORD State:
DWORD Protect:
DWORD Type;
} MEMORY BASIC INFORMATION;
typedef MEMORY BASIC INFORMATION *
PMEMORY BASIC INFORMATION;
还有一些函数,例如 VirtualAlloc, VirtualAllocEx, VirtualFree
和 VirtualFreeEx 等,用于虚拟内存的管理,详情请见 Microsoft
的 Win32 API Reference Manual。
```

2. 具体步骤

使用这些 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:

- 时间:开始执行的时间;
- 块数:分配内存的粒度;
- 操作:包括保留一个区域、提交一个区域、释放一个区域、回收 以及锁与解锁一个区域:可以将这些操作编号,存放于文件中。

- 大小: 指块的大小;
- 访问权限: 共五种 PAGE_READONLY、PAGE_READWRITE、PAGE_EXCUTE、PAGE_EXECUTE_READ 和PAGE_EXECUTE_READWRITE。可以将这些权限编号,存放于文件中。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。

3. 程序说明

首先执行 makefile. exe, 生成 opfile 文件, 里面保存了模拟的内存操作。然后执行 memory-op. exe, 产生两个线程, 一个从 opfile 文件里读取内存操作,模拟内存活动,另一个跟踪第一个的内存行为,将结果输出并保存在 out. txt 文件中。两个线程通过信号量实现同步。

四、 实验结果

执行 makefile. exe 之后生成 opfile 文件:

Debug	2018/1/4 16:57	文件夹	
🔁 makefile.vcxproj	2018/1/4 16:55	VC++ Project	7 KB
makefile.vcxproj.filters	2018/1/4 16:55	VC++ Project Fil	1 KB
📄 opfile 🔫 新生成的opfile	2018/1/4 16:57	文件	1 KB
⊆ 源.cpp	2018/1/4 16:54	JetBrains CLion	1 KB

opfile 文件内容:

opfile ⊅ X	空	[往]	时间	1	#	, L	小				操1	乍			ŧΩi	REL	
	20	_	_		00	00	00	00		20			00	000	17.5		001
00000000	29	00		00	03			00		00 00	00	00			00		00
00000010 00000020	4E A9	01 00	00	00	01 05	00	00	00		00	00	00		01 02	00	00	00
00000020	DE	01	00	00	03	00	00	00		00	00	00	00	03	00	00	00
00000030	C2	03	00	00	05	00	00	00		00	00	00	00	04	00	00	00
00000050	C1	02	00	00	01	00	00	00		01	00	00	00	00	00	00	00
00000060	19	01	00	00	03	00	00	00		01	00	00	00	01	00	00	00
00000070	Ċ1	03	00	00	02	00	00	00		01	00	00	00	02	00	00	00
00000080	Ĕ3	03	00	00	03	00	00	00		01	00	00	00	03	00	00	00
00000090	3B	03	00	00	02	00	00	00		01	00	00	00	04	00	00	00
000000a0	87	01	00	00	05	00	00	00		02	00	00	00	00	00	00	00
000000ь0	86	03	00	00	04	00	00	00		02	00	00	00	01	00	00	00
000000c0	24	01	00	00	03	00	00	00	- 0	02	00	00	00	02	00	00	00
000000d0	A5	01	00	00	02	00	00	00		02	00	00	00	03	00	00	00
000000e0	CE	02	00	00	01	00	00	00		02	00	00	00	04	00	00	00
000000f0	BF	01	00	00	02	00	00	00		03	00	00	00	00	00	00	00
00000100	03	03	00	00	04	00	00	00	- 0	03	00	00	00	01	00	00	00
00000110	65	03	00	00	03	00	00	00		03	00	00	00	02	00	00	00
00000120	9B	02	00	00	05	00	00	00		03	00	00	00	03	00	00	00
00000130	23	00	00	00	05	00	00	00		03	00	00	00	04	00	00	00
00000140	BF	02	00	00	02	00	00	00		04	00	00	00	00	00	00	00
00000150	42	01	00	00	04	00	00	00		04	00	00	00	01	00	00	00
00000160	A1	02	00	00	05	00	00	00		04	00	00		02	00	00	00
00000170	8D	00	00	00	02	00	00	00		04	00	00		03	00	00	00
00000180	FD	00	00	00	04	00	00	00		04	00	00	00	04	00	00	00
00000190	23	02	00	00	05	00	00	00		05	00	00	00	00	00	00	00
000001a0	96	02	00	00	03	00	00	00		05	00	00	00	01	00	00	00
000001b0	25	00	00	00	05	00	00	00		05	00	00	00	02	00	00	00
000001c0 000001d0	D3	02 02	00	00	02	00	00	00		05 05	00	00	-	03	00	00	00
000001d0	11	UZ	00	00	04	00	00	00	Ľ	oo_	00	00	00	04	00	00	00
00000160																	

执行 memory-op. exe:

■ F:\好东西\操作系统\实验\2017526019-刘禾子-实验二\memory-op.exe						
starting address:02D20000 4:reserve now	size:16384					
starting address:03500000 5:commit now	size:20480					
starting address:009E0000 6:commit now	size:12288					
starting address:009F0000 7:commit now	size:4096					
starting address:02D10000 8:commit now	size:20480					
starting address:02D20000 9:commit now	size:16384					
	size:20480					
starting address:009E0000	size:12288					
11:lock now starting address:009F0000	ciza:4006					
12: lock now						
starting address:02D10000 998	size:20480					
	size:16384					
14:lock now starting address:03500000	size:20480					
998 15:unlock now						
	size:12288					
158 GetLastError()返回擦 16:unlock now						
starting address:009F0000 17:unlock now	size:4096					
	size:20480					
18:unlock now						
19:unlock now	size:16384					
158	size:20480					
	size:12288					
87 21:decommit now						
starting address:009F0000 22:decommit now	size:4096					
starting address:02D10000	size:20480					
23:decommit now	-11/004					
starting address:02D20000 24:decommit now	size:16384					
starting address:03500000 87	size:20480					
25:release now	0170:1220					
starting address:009E0000 26:release now	size:12288					
	size:4096					

同时向 out. txt 写内容(以第 0 个记录为例具体参考文件中 out. txt): 分别输出当前计算机系统信息、可用物理和虚拟内存信息、有关进程的虚拟地址空间中的一系列页面信息。

🎒 out.txt - 记事本 文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H) dwActiveProcessorMask dwAllocationGranularity 65536 dwNumberOfProcessors 4 dwOemId 0 dwPageSize 4096 dwProcessorType 586 lpMaximumApplicationAddress 7FFEFFFF lpMinimumApplicationAddress 00010000 wProcessorArchitecture 0 wProcessorLevel 6 wProcessorRevision 40457 wReserved 0 *************************** dwAvailPageFile 4294967295 dwAvailPhys 2147483647 dwAvailVirtual 2092498944 dwLength 32 dwMemoryLoad 45 dwTotalPageFile 4294967295 dwTotalPhys 2147483647 dwTotalVirtual 2147352576 AllocationBase 00000000 AllocationProtect 0 BaseAddress 00010000 Protect 1 RegionSize 8323072 State 65536 Type 0

五、 源代码

```
//文件生成程序
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
struct operation
   int time;//起始时间
   int block://内存页数
   int oper;//操作
   int protection;//权限
};
int main()
   FILE* file;
   file = fopen("opfile", "wb");// "opfile" 为二进制用以确定内存操作
   operation op;
   for (int j = 0; j<6; j++) //0-保留; 1-提交; 2-锁; 3-解锁; 4-回收; 5-释放
       for (int i = 0; i < 5; i++)
          //O-PAGE READONLY;
           //1-PAGE_READWRITE;
          //2-PAGE EXECUTE;
          //3-PAGE_EXECUTE_READ;
          //4-PAGE EXECUTE READWRITE;
           op. time = rand() % 1000;//随机生成等待时间
           op. block = rand() % 5 + 1;//随机生成块大小
           op. oper = j;
           op.protection = i;
           fwrite(&op, sizeof(operation), 1, file);//将生成的结构写入文件
       }
   return 0;
memory-op.cpp
//内存管理实习
//将程序从文件读入每次的操作,并将结果输入到out.txt文件中
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <iostream>
```

```
using namespace std;
struct operation
    int time;//起始时间
    int block;//内存页数
    int oper;//操作
    int protection;//权限
};
struct trace //跟踪每一次分配活动的数据结构
   LPVOID start;//起始地址
   long size; //分配的大小
};
HANDLE allo, trac; //信号量的句柄
DWORD Tracker (LPDWORD 1pdwparm) //跟踪allocator线程的内存行为,并输出必要信息
    ofstream outfile;//输出文件
    outfile.open("out.txt");
    for (int i = 0; i \le 30; i++)
        WaitForSingleObject(trac, INFINITE); //等待allocator一次内存分配活动结束
                                             //打印内存状况和系统状况
        outfile << i << endl;
        //以下一段显示系统信息,每次执行操作后系统信息不变
        //如果要查看系统信息,可以取消注释
        SYSTEM_INFO info; //系统信息
        GetSystemInfo(&info);
        outfile << "dwActiveProcessorMask" << '\t' << info.dwActiveProcessorMask <<
end1;
        outfile << "dwAllocationGranularity" << '\t' << info. dwAllocationGranularity
<< endl;</pre>
        outfile << "dwNumberOfProcessors" << '\t' << info.dwNumberOfProcessors <<
endl:
        outfile << "dw0emId" << '\t' << info.dw0emId << endl;</pre>
        outfile << "dwPageSize" << '\t' << info.dwPageSize << endl;
        outfile << "dwProcessorType" << '\t' << info. dwProcessorType << endl;
        outfile << "lpMaximumApplicationAddress" << '\t' <<
info.lpMaximumApplicationAddress << endl;</pre>
        outfile << "lpMinimumApplicationAddress" << '\t' <<
```

```
info.lpMinimumApplicationAddress << endl;</pre>
        outfile << "wProcessorArchitecture" << '\t' << info.wProcessorArchitecture <<
endl;
        outfile << "wProcessorLevel" << '\t' << info. wProcessorLevel << endl;
        outfile << "wProcessorRevision" << '\t' << info.wProcessorRevision << endl;</pre>
        outfile << "wReserved" << '\t' << info.wReserved << endl;</pre>
<< endl;</pre>
        //内存状况
        MEMORYSTATUS status; //内存状态
        GlobalMemoryStatus(&status);
        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;</pre>
        //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 mem;//内存基本信息
        VirtualQuery (info. lpMinimumApplicationAddress, &mem,
            sizeof(MEMORY BASIC INFORMATION));
        outfile << "AllocationBase" << '\t' << mem. AllocationBase << endl;
        outfile << "AllocationProtect" << '\t' << mem. AllocationProtect << endl;
        outfile << "BaseAddress" << '\t' << mem. BaseAddress << endl;
        outfile << "Protect" << '\t' << mem. Protect << endl;
        outfile << "RegionSize" << '\t' << mem. RegionSize << endl;
        outfile << "State" << '\t' << mem. State << endl;</pre>
        outfile << "Type" << '\t' << mem. Type << endl;
        outfile <<
~~~~~" << endl;</pre>
        //释放信号量通知allocator可以执行下一次内存分配活动
        ReleaseSemaphore(allo, 1, NULL);
```

```
}
    return 0;
void Allocator() //模拟内存分配活动的线程
    trace traceArray[5];
    int index = 0;
    FILE* file;
    file = fopen("opfile", "rb");//读入文件
    operation op;
    SYSTEM INFO info;
    DWORD temp;
    GetSystemInfo(&info);
    for (int i = 0; i < 30; i++)
    {
        WaitForSingleObject(allo, INFINITE); //等待tracker打印结束的信号量
        cout << i << ':';
        fread(&op, sizeof(operation), 1, file);
        Sleep(op. time); //执行时间, 如果想在指定时间执行可以取消注释
        GetSystemInfo(&info);
        switch (op. protection) //根据文件内容确定权限
        {
        case 0:
            index = 0;
            temp = PAGE_READONLY;
            break;
        }
        case 1:
            temp = PAGE_READWRITE;
            break;
        case 2:
            temp = PAGE_EXECUTE;
            break;
        case 3:
            temp = PAGE_EXECUTE_READ;
            break:
        case 4:
            temp = PAGE_EXECUTE_READWRITE;
            break;
        default:
            temp = PAGE_READONLY;
        switch (op. oper)
```

```
{
         case 0://保留一个区域
              cout << "reserve now" << endl;</pre>
              traceArray[index].start = VirtualAlloc(NULL, op.block*info.dwPageSize,
                  MEM RESERVE, PAGE NOACCESS);
              traceArray[index++].size = op.block*info.dwPageSize;
              cout << "starting address:"</pre>
                  << traceArray[index - 1].start << '\t' << "size:" << traceArray[index</pre>
- 1].size << endl;
             break;
         case 1://提交一个区域
              cout << "commit now" << endl;</pre>
              traceArray[index].start = VirtualAlloc(traceArray[index].start,
traceArray[index].size, MEM_COMMIT, temp);
              index++;
              cout << "starting address:"</pre>
                  << traceArray[index - 1].start << '\t' << "size:" << traceArray[index</pre>
- 1].size << endl;
             break;
         case 2: //锁一个区域
              cout << "lock now" << endl;</pre>
             cout << "starting address:" << traceArray[index].start << '\t' << "size:"</pre>
<< traceArray[index].size << endl;</pre>
              if (!VirtualLock(traceArray[index].start, traceArray[index++].size))
                  cout << GetLastError() << endl;//GetLastError()函数返回错误号
             break;
         }
         case 3: //解锁一个区域
             cout << "unlock now" << endl;</pre>
             cout << "starting address:" << traceArray[index].start << '\t' << "size:"</pre>
<< traceArray[index].size << endl;</pre>
              if (!VirtualUnlock(traceArray[index].start, traceArray[index++].size))
                  cout << GetLastError() << endl;</pre>
             break;
         case 4: //回收一个区域
```

```
{
            cout << "decommit now" << endl:</pre>
            cout << "starting address:" << traceArray[index].start << '\t' << "size:"</pre>
<< traceArray[index].size << endl;</pre>
            if (!VirtualFree(traceArray[index].start, traceArray[index++].size,
MEM DECOMMIT))
                cout << GetLastError() << endl;</pre>
            break:
        }
        case 5: //释放一个区域
            cout << "release now" << endl;</pre>
            cout << "starting address:" << traceArray[index].start << '\t' << "size:"</pre>
<< traceArray[index].size << endl;</pre>
            if (!VirtualFree(traceArray[index++].start, 0, MEM_RELEASE))
                cout << GetLastError() << endl;</pre>
            break;
        }
        default:
            cout << "error" << endl;</pre>
        ReleaseSemaphore(trac, 1, NULL); //释放信号量通知tracker可以打印信息
    }
int main()
    DWORD dwThread;
    HANDLE handle[2];
    //生成两个线程
    handle[0] = CreateThread(NULL, 0, (LPTHREAD START ROUTINE)Tracker, NULL, 0,
&dwThread);
    handle[1] = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Allocator, NULL, 0,
&dwThread);
    //生成两个信号量
    allo = CreateSemaphore(NULL, 0, 1, (LPCWSTR) "allo");
    trac = CreateSemaphore(NULL, 1, 1, (LPCWSTR)"trac");
    //等待线程执行的执行结束后,再退出
    WaitForMultipleObjects(2, handle, TRUE, INFINITE);
    system("pause");
//The End
```