加载器的实现

自己实现了一个加载器，可以加载自己编写的程序，很简单，给出代码：

```
// PELoader.cpp : 定义控制台应用程序的入口点。
//
#include "stdafx.h"
#include<Windows.h>
#define FILE_PATH "C:\\Users\\AVATAR\\Documents\\Visual Studio 2013\\Projects\\PELoader\\Debug\\PETest.exe"
typedef int (WINAPI *MAIN)(HINSTANCE, HINSTANCE, LPSTR, int);
LPVOID InitPE(LPSTR szPEPath)
{
	LPVOID lpDsk;
	FILE *fp = fopen(szPEPath,"rb");
	if (fp == NULL)
	{
		return NULL;
	}
	fseek(fp,0,SEEK_END);
	DWORD dwPESize = ftell(fp);
	fseek(fp, 0, SEEK_SET);
	lpDsk = malloc(dwPESize);
	fread(lpDsk, dwPESize, 1, fp);
	fclose(fp);
	return lpDsk;
}
BOOL MemoryLoader(LPVOID lpBuff)
{
	BYTE * pMem = (BYTE*)lpBuff;
	if (pMem == NULL)
	{
		return FALSE;
	}
	DWORD dwIgnore;
	IMAGE_DOS_HEADER ImgDosHdr = { 0 };
	IMAGE_NT_HEADERS ImgNtHdr = { 0 };
	IMAGE_SECTION_HEADER * ImgSecHdrList = NULL;
	memcpy(&ImgDosHdr, pMem, sizeof(IMAGE_DOS_HEADER));
	memcpy(&ImgNtHdr, pMem + ImgDosHdr.e_lfanew, sizeof(IMAGE_NT_HEADERS));
	if (ImgDosHdr.e_magic != 0x5a4d)
	{
		return FALSE;
	}
	if (ImgNtHdr.Signature != 0x4550)
	{
```

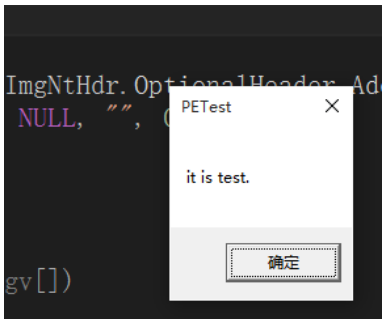
```
		return FALSE;
	}
	BYTE* pMemMap = (BYTE*)VirtualAlloc(NULL, ImgNtHdr.OptionalHeader.SizeOfImage, MEM_COMMIT,
PAGE_READWRITE);
	VirtualProtect(pMemMap, ImgNtHdr.OptionalHeader.SizeOfImage, PAGE_EXECUTE_READWRITE, &dwIgnore);
	memcpy(pMemMap, pMem, ImgNtHdr.OptionalHeader.SizeOfHeaders);
	DWORD dwSecHdrsSize = ImgNtHdr.FileHeader.NumberOfSections*sizeof(IMAGE_SECTION_HEADER);
	ImgSecHdrList = (IMAGE_SECTION_HEADER*)malloc(dwSecHdrsSize);
	memcpy(ImgSecHdrList, pMem + ImgDosHdr.e_lfanew + sizeof(IMAGE_NT_HEADERS), dwSecHdrsSize);
	DWORD dwRelocIndex = -1;
	int i = 0,j = 0;
	for (i = 0; i < ImgNtHdr.FileHeader.NumberOfSections; i++)
	{
		IMAGE_SECTION_HEADER sec = ImgSecHdrList[i];
		DWORD rva = sec.VirtualAddress;
		if (sec.PointerToRawData == NULL)
		{

		}
		else
		{
			memcpy(pMemMap + rva, pMem + sec.PointerToRawData,sec.SizeOfRawData);
		}
		if (strcmp((char*)sec.Name,".reloc")==0)
		{
			dwRelocIndex = i;
		}
	}
	//reloc
	BYTE * RelocMem = NULL;
	RelocMem = pMemMap +
ImgNtHdr.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].VirtualAddress;
	DWORD dwRelocValue = (DWORD)pMemMap - ImgNtHdr.OptionalHeader.ImageBase;
	dwRelocIndex = 0;
	while (true)
	{
		DWORD RvaOfBlock = *(DWORD*)&RelocMem[dwRelocIndex];
		if (RvaOfBlock == 0)
		{
			break;
		}
		dwRelocIndex += 4;
		DWORD SizeOfBlock = *(DWORD*)&RelocMem[dwRelocIndex] / 2 - 4;
		dwRelocIndex += 4;
```

```
            for (i = 0; i<SizeOfBlock; i++)
            {
                short TypeRVA = *(short*)&RelocMem[dwRelocIndex];
                dwRelocIndex += 2;
                TypeRVA = TypeRVA & 0xfff;
                if (TypeRVA != 0)
                {
                    *(DWORD*)(pMemMap + TypeRVA + RvaOfBlock) += dwRelocValue;
                }
            }
        }
        //import table
        DWORD dwImpRVA = ImgNtHdr.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress;
        IMAGE_IMPORT_DESCRIPTOR * pIMP = (IMAGE_IMPORT_DESCRIPTOR *)(pMemMap + dwImpRVA);
        while (true)
        {
            if (pIMP->Name == NULL)
            {
                break;
            }
            DWORD dwINTRVA = pIMP->OriginalFirstThunk;
            DWORD dwNameRVA = pIMP->Name;
            DWORD dwIATRVA = pIMP->FirstThunk;
            LPSTR szDllName = (LPSTR)(pMemMap + dwNameRVA);
            HMODULE hmDll = LoadLibraryA(szDllName);
            DWORD * pIAT = (DWORD*)(pMemMap + dwIATRVA);
            while (true)
            {
                if (*pIAT == 0)
                {
                    break;
                }
                DWORD dwIATMARK = (DWORD)(*pIAT + pMemMap + 2);
                if ((dwIATMARK & IMAGE_ORDINAL_FLAG) != 0)
                {
                    dwIATMARK = dwIATMARK & 0xffff;
                    *pIAT = (DWORD)GetProcAddress(hmDll, (LPSTR)dwIATMARK);
                }
                else
                {
                    LPSTR szFuncName = (LPSTR)dwIATMARK;
                    *pIAT = (DWORD)GetProcAddress(hmDll, szFuncName);
                }
```

```
                    pIAT++;
                }
                pIMP++;
        }
        DWORD dwLoadConfigDirRVA = ImgNtHdr.OptionalHeader.DataDirectory[10].VirtualAddress;
        IMAGE_LOAD_CONFIG_DIRECTORY * ilcd = (IMAGE_LOAD_CONFIG_DIRECTORY *)(pMemMap +
dwLoadConfigDirRVA);
        ilcd->SecurityCookie = (DWORD)(ilcd->SecurityCookie - ImgNtHdr.OptionalHeader.ImageBase + pMemMap);
        MAIN ptrMain;
        ptrMain = (MAIN)(pMemMap + ImgNtHdr.OptionalHeader.AddressOfEntryPoint);
        ptrMain((HINSTANCE)pMemMap, NULL, "", 0xa);
        return TRUE;
}
int _tmain(int argc, _TCHAR* argv[])
{
        LPVOID lpPEBuf;
        lpPEBuf = InitPE(FILE_PATH);
        MemoryLoader(lpPEBuf);
        free(lpPEBuf);
        return 0;
}
```

下面给出截图：

```
154     MAIN ptrMain;
155     ptrMain = (MAIN)(pMemMap + ImgNtHdr.OptionalHeac
156     ptrMain((HINSTANCE)pMemMap, NULL, "", 0xa);
```



其中 petest 作为被加载的程序，只有一个功能，就是显示这个对话框。

然后，我就希望将这个程序扩展到其他场景，比如说 notepad.exe，然后，我加载了，然后运行了，程序没有报错，但是运行的结果是加载器直接退出了。

所以我就通过 od 调试并与自己写的加载器进行比较的时候，找到了加载器退出的原因：

```
00155B35  .  6A 00          PUSH 0                                   Arg2 = 0
00155B37  .  68 00001400    PUSH OFFSET <STRUCT IMAGE_DOS_HEADER>    Arg1 = notepad.<ST
00155B3C  .  E8 8F02FFFF    CALL 00145DD0                            notepad.00145DD0
00155B41  .  A3 A0771500    MOV DWORD PTR DS:[1577A0],EAX
00155B46  .  833D B877150(  CMP DWORD PTR DS:[1577B8],0
00155B4D  .v 75 60          JNE SHORT 00155BAF
00155B4F  .  50             PUSH EAX                                 status
00155B50  .  FF15 78A3150(  CALL DWORD PTR DS:[<&msvcrt.exit>]       MSVCRT.exit
00155B56  >  80F9 22        CMP CL,22
00155B59  .v 75 0C          JNE SHORT 00155B67
```

在正常情况下，在执行完我打断点的位置后，即 00155B3C，会显示 notepad 主窗体，而不继续向下执行，然而，在我的加载器中，程序并没有停下来显示窗体，而是继续向下执

行，执行 msvcrt.exit，然后程序就退出了。

所以，加载器退出的原因是被加载程序触发了退出条件，然后执行 msvcrt.exit 函数退出。那么，就需要进一步寻找触发的退出条件的原因是什么。

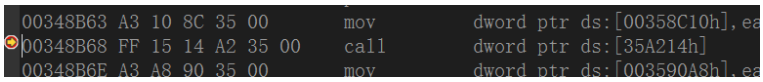
我找到了退出的逻辑：

在 base+8b68 位置处，call 了一个函数，这个函数是 LoadAcceleratorsw：

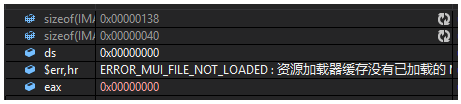
> 函数功能：调入加速键表。该函数调入指定的加速键表。
>
> 函数原型：HACCEL LoadAccelerators（HINSTANCE hInstance，LPCTSTR lpTableName）；
>
> 参数：
>
> hInstance:模块的一个实例的句柄，该模块的可执行文件中包含将要调入的加速键表。
>
> lpTableName:指向一个以空结尾的字符串的指针，该字符串包含了即将调入的加速键表的名字。另一种可选的方案是，该参数可以在加速键表资源的低位字中指定资源标识符，而高位字中全零。MADEINTRESOURCE 宏可被用于创建该值。
>
> 返回值：若函数调用成功，则返回所加载的加速键表句柄[1] 。若函数调用失败，则返回值为 NULL。若要获得更多的错误信息，可以调用 GetLastError 函数。
>
> 备注：若加速键表尚未装入，该函数可从指定的可执行文件中将它装入。从资源中装入的加速键表，在程序结束时可自动释放。Windows CE：资源不被拷贝到 RAM 中，因而不能被修改。
>
> 速查：Windows NT：3.1 及以上版本；Windows：95 及以上版本；Windows CE：1.0 及以上版本；头文件：winuser.h;库文件：user32.lib；Uncode：在 Windows NT 上实现为 Unicode 和 ANSI 两种版本。

在调用的时候，hInstance 为模块基地址，lpTableName 指向 MainAcc 字符串。返回值在 od 调试下不为 0，在我的加载器中为 0 。在这里有个个判断逻辑，返回值是否为 0。如果为 0，那么程序将返回调用方，然后继续执行使得程序退出。

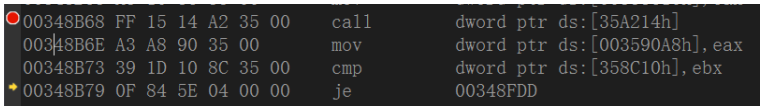
然后通过 getlasterror，发现在运行中有这个错误：资源加载器缓存没有已加载的 mui 项。首先找到这个位置偏移 8B68 处：



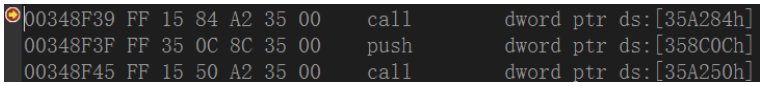
然后 step over



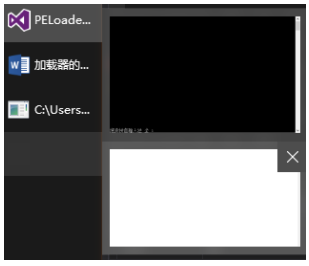
在这里修改 eax，使得 eax 不等于 0



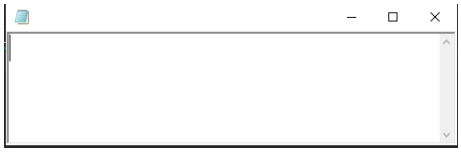
这里 je 就会跳另外一个分支，其中显示窗体函数在偏移 8F39 处被调用



执行 step over 后，显示一个白色窗体：



继续执行，显示出来了 notepad 的主界面：



但似乎是少点什么，对，没有菜单栏了。后来上网发现，mui 文件是与语言资源图标相关的，所以 notepad 没有找到对应的 mui 项在内存中的映射，所以菜单的图标就无法显示。

那么下面我将寻找 mui 项是如何映射到内存，以及 notepad 如何定位该内存地址的。