PEB结构(Process Envirorment Block Structure)其中文名是进程环境块信息，进程环境块内部包含了进程运行的详细参数信息，每一个进程在运行后都会存在一个特有的PEB结构，通过附加进程并遍历这段结构即可得到非常多的有用信息。

在应用层下，如果想要得到PEB的基地址只需要取 fs:[0x30] 即可，TEB线程环境块则是 fs:[0x18]，如果在内核层想要得到应用层进程的PEB信息我们需要调用特定的内核函数来获取。

在内核层要获取应用层进程的PEB结构，可以通过以下步骤实现：

- 1.调用内核函数 PsGetCurrentProcess 获取当前进程的 EPROCESS 结构。
- 2.调用内核函数 KeStackAttachProcess，附加到目标进程。
- 3.调用内核函数 PsGetProcessWow64Process，获取目标进程的PEB结构信息。
- 4.通过 PEB 结构的 Ldr 成员可以访问到该进程加载的所有模块，遍历整个 Ldr 链表即可得到需要的模块信息。
- 5.遍历完成后，通过调用 KeUnstackDetachProcess 函数脱离进程空间。

首先在开始写代码之前需要先定义好 PEB 进程环境快结构体，用于对内存指针解析，新建 peb.h 文件并保存如下代码，这些是微软的结构定义分为32位与64位，官方定义规范而已不需要费工夫。

```
#pragma once
#include <ntifs.h>

typedef struct _CURDIR              // 2 elements, 0x18 bytes (sizeof)
{
    /*0x000*/     struct _UNICODE_STRING DosPath; // 3 elements, 0x10 bytes (sizeof)
    /*0x010*/     VOID*          Handle;
}CURDIR, *PCURDIR;

typedef struct _RTL_DRIVE_LETTER_CURDIR // 4 elements, 0x18 bytes (sizeof)
{
    /*0x000*/     UINT16         Flags;
    /*0x002*/     UINT16         Length;
    /*0x004*/     ULONG32        TimeStamp;
    /*0x008*/     struct _STRING DosPath;             // 3 elements, 0x10 bytes (sizeof)
}RTL_DRIVE_LETTER_CURDIR, *PRTL_DRIVE_LETTER_CURDIR;

typedef enum _SYSTEM_DLL_TYPE  // 7 elements, 0x4 bytes
{
    PsNativeSystemDll = 0 /*0x0*/,
    PsWowX86SystemDll = 1 /*0x1*/,
    PsWowArm32SystemDll = 2 /*0x2*/,
    PsWowAmd64SystemDll = 3 /*0x3*/,
    PsWowChpeX86SystemDll = 4 /*0x4*/,
    PsVsmEnclaveRuntimeDll = 5 /*0x5*/,
    PsSystemDllTotalTypes = 6 /*0x6*/
}SYSTEM_DLL_TYPE, *PSYSTEM_DLL_TYPE;

typedef struct _EWOW64PROCESS        // 3 elements, 0x10 bytes (sizeof)
{
    /*0x000*/     VOID*          Peb;
    /*0x008*/     UINT16         Machine;
    /*0x00A*/     UINT8          _PADDING0_[0x2];
```

```
    /*0x00C*/     enum _SYSTEM_DLL_TYPE NtdllType;
}EWOW64PROCESS, *PEWOW64PROCESS;

typedef struct _RTL_USER_PROCESS_PARAMETERS                 // 37 elements, 0x440 bytes
(sizeof)
{
    /*0x000*/     ULONG32      MaximumLength;
    /*0x004*/     ULONG32      Length;
    /*0x008*/     ULONG32      Flags;
    /*0x00C*/     ULONG32      DebugFlags;
    /*0x010*/     VOID*        ConsoleHandle;
    /*0x018*/     ULONG32      ConsoleFlags;
    /*0x01C*/     UINT8        _PADDING0_[0x4];
    /*0x020*/     VOID*        StandardInput;
    /*0x028*/     VOID*        StandardOutput;
    /*0x030*/     VOID*        StandardError;
    /*0x038*/     struct _CURDIR CurrentDirectory;                        // 2 elements, 0x18
bytes (sizeof)
    /*0x050*/     struct _UNICODE_STRING DllPath;                         // 3 elements, 0x10
bytes (sizeof)
    /*0x060*/     struct _UNICODE_STRING ImagePathName;                   // 3 elements, 0x10
bytes (sizeof)
    /*0x070*/     struct _UNICODE_STRING CommandLine;                     // 3 elements, 0x10
bytes (sizeof)
    /*0x080*/     VOID*        Environment;
    /*0x088*/     ULONG32      StartingX;
    /*0x08C*/     ULONG32      StartingY;
    /*0x090*/     ULONG32      CountX;
    /*0x094*/     ULONG32      CountY;
    /*0x098*/     ULONG32      CountCharsX;
    /*0x09C*/     ULONG32      CountCharsY;
    /*0x0A0*/     ULONG32      FillAttribute;
    /*0x0A4*/     ULONG32      WindowFlags;
    /*0x0A8*/     ULONG32      ShowWindowFlags;
    /*0x0AC*/     UINT8        _PADDING1_[0x4];
    /*0x0B0*/     struct _UNICODE_STRING WindowTitle;                     // 3 elements, 0x10
bytes (sizeof)
    /*0x0C0*/     struct _UNICODE_STRING DesktopInfo;                     // 3 elements, 0x10
bytes (sizeof)
    /*0x0D0*/     struct _UNICODE_STRING ShellInfo;                       // 3 elements, 0x10
bytes (sizeof)
    /*0x0E0*/     struct _UNICODE_STRING RuntimeData;                     // 3 elements, 0x10
bytes (sizeof)
    /*0x0F0*/     struct _RTL_DRIVE_LETTER_CURDIR CurrentDirectores[32];
    /*0x3F0*/     UINT64       EnvironmentSize;
    /*0x3F8*/     UINT64       EnvironmentVersion;
    /*0x400*/     VOID*        PackageDependencyData;
    /*0x408*/     ULONG32      ProcessGroupId;
    /*0x40C*/     ULONG32      LoaderThreads;
    /*0x410*/     struct _UNICODE_STRING RedirectionDllName;              // 3 elements, 0x10
bytes (sizeof)
    /*0x420*/     struct _UNICODE_STRING HeapPartitionName;               // 3 elements, 0x10
bytes (sizeof)
```

```
    /*0x430*/     UINT64*      DefaultThreadpoolCpuSetMasks;
    /*0x438*/     ULONG32      DefaultThreadpoolCpuSetMaskCount;
    /*0x43C*/     UINT8        _PADDING2_[0x4];
}RTL_USER_PROCESS_PARAMETERS, *PRTL_USER_PROCESS_PARAMETERS;

typedef struct _PEB_LDR_DATA                            // 9 elements, 0x58 bytes (sizeof)
{
    /*0x000*/     ULONG32      Length;
    /*0x004*/     UINT8        Initialized;
    /*0x005*/     UINT8        _PADDING0_[0x3];
    /*0x008*/     VOID*        SsHandle;
    /*0x010*/     struct _LIST_ENTRY InLoadOrderModuleList;           // 2 elements, 0x10
bytes (sizeof)
    /*0x020*/     struct _LIST_ENTRY InMemoryOrderModuleList;         // 2 elements, 0x10
bytes (sizeof)
    /*0x030*/     struct _LIST_ENTRY InInitializationOrderModuleList; // 2 elements, 0x10
bytes (sizeof)
    /*0x040*/     VOID*        EntryInProgress;
    /*0x048*/     UINT8        ShutdownInProgress;
    /*0x049*/     UINT8        _PADDING1_[0x7];
    /*0x050*/     VOID*        ShutdownThreadId;
}PEB_LDR_DATA, *PPEB_LDR_DATA;

typedef struct _PEB64
{
    UCHAR InheritedAddressSpace;
    UCHAR ReadImageFileExecOptions;
    UCHAR BeingDebugged;
    UCHAR BitField;
    ULONG64 Mutant;
    ULONG64 ImageBaseAddress;
    PPEB_LDR_DATA Ldr;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    ULONG64 SubSystemData;
    ULONG64 ProcessHeap;
    ULONG64 FastPebLock;
    ULONG64 AtlThunkSListPtr;
    ULONG64 IFEOKey;
    ULONG64 CrossProcessFlags;
    ULONG64 UserSharedInfoPtr;
    ULONG SystemReserved;
    ULONG AtlThunkSListPtr32;
    ULONG64 ApiSetMap;
} PEB64, *PPEB64;

#pragma pack(4)
typedef struct _PEB32
{
    UCHAR InheritedAddressSpace;
    UCHAR ReadImageFileExecOptions;
    UCHAR BeingDebugged;
    UCHAR BitField;
    ULONG Mutant;
```

```
    ULONG ImageBaseAddress;
    ULONG Ldr;
    ULONG ProcessParameters;
    ULONG SubSystemData;
    ULONG ProcessHeap;
    ULONG FastPebLock;
    ULONG AtlThunkSListPtr;
    ULONG IFEOKey;
    ULONG CrossProcessFlags;
    ULONG UserSharedInfoPtr;
    ULONG SystemReserved;
    ULONG AtlThunkSListPtr32;
    ULONG ApiSetMap;
} PEB32, *PPEB32;

typedef struct _PEB_LDR_DATA32
{
    ULONG Length;
    BOOLEAN Initialized;
    ULONG SsHandle;
    LIST_ENTRY32 InLoadOrderModuleList;
    LIST_ENTRY32 InMemoryOrderModuleList;
    LIST_ENTRY32 InInitializationOrderModuleList;
    ULONG EntryInProgress;
} PEB_LDR_DATA32, *PPEB_LDR_DATA32;

typedef struct _LDR_DATA_TABLE_ENTRY32
{
    LIST_ENTRY32 InLoadOrderLinks;
    LIST_ENTRY32 InMemoryOrderModuleList;
    LIST_ENTRY32 InInitializationOrderModuleList;
    ULONG DllBase;
    ULONG EntryPoint;
    ULONG SizeOfImage;
    UNICODE_STRING32 FullDllName;
    UNICODE_STRING32 BaseDllName;
    ULONG Flags;
    USHORT LoadCount;
    USHORT TlsIndex;
    union
    {
        LIST_ENTRY32 HashLinks;
        ULONG SectionPointer;
    }u1;
    ULONG CheckSum;
    union
    {
        ULONG TimeDateStamp;
        ULONG LoadedImports;
    }u2;
    ULONG EntryPointActivationContext;
    ULONG PatchInformation;
} LDR_DATA_TABLE_ENTRY32, *PLDR_DATA_TABLE_ENTRY32;
```

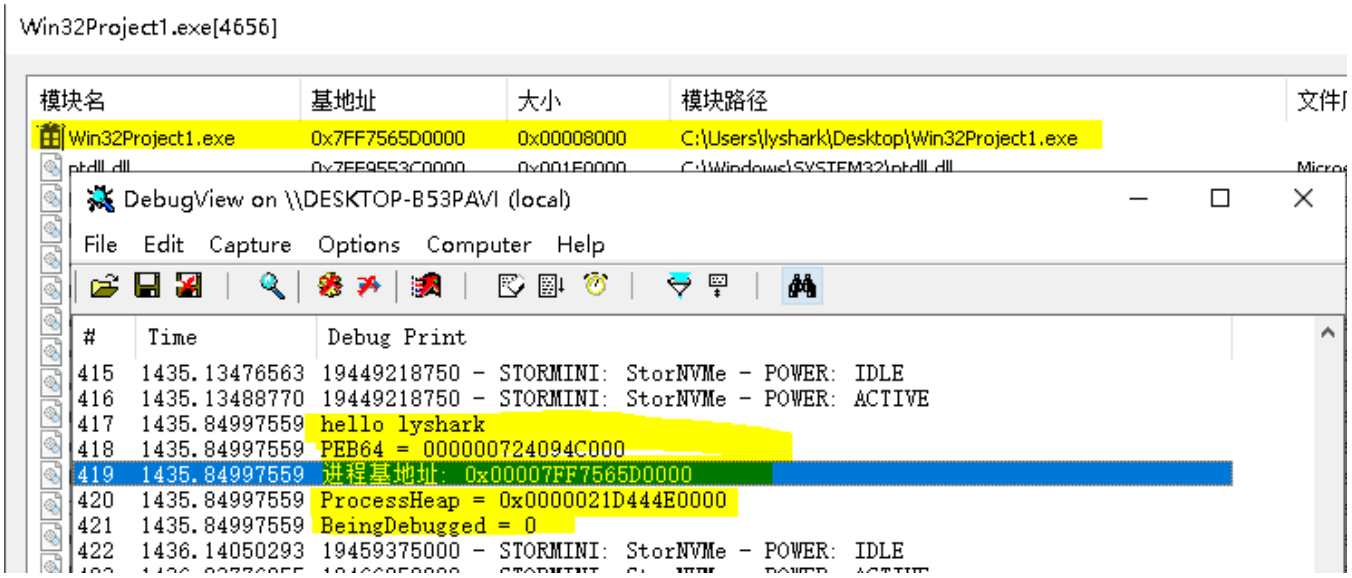
```
#pragma pack()
```

接着就来实现对PEB的获取操作，以`64位`为例，我们需要调用`PsGetProcessPeb()`这个内核函数，因为该内核函数没有被公开所以调用之前需要头部导出，该函数需要传入用户进程的`EProcess`结构，该结构可用`PsLookupProcessByProcessId`函数动态获取到，获取到以后直接`KeStackAttachProcess()`附加到应用层进程上，即可直接输出进程的PEB结构信息，如下代码。

```
#include "peb.h"
#include <ntifs.h>

// 定义导出
NTKERNELAPI PVOID NTAPI PsGetProcessPeb(_In_ PEPROCESS Process);

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("Uninstall Driver Is OK \n"));
}
// LyShark
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark \n");

    NTSTATUS status = STATUS_UNSUCCESSFUL;
    PEPROCESS eproc = NULL;
    KAPC_STATE kpc = { 0 };

    PPEB64 pPeb64 = NULL;

    __try
    {
        // HANDLE)4656 进程PID
        status = PsLookupProcessByProcessId((HANDLE)4656, &eproc);

        // 得到64位PEB
        pPeb64 = (PPEB64)PsGetProcessPeb(eproc);

        DbgPrint("PEB64 = %p \n", pPeb64);

        if (pPeb64 != 0)
        {
            // 验证可读性
            ProbeForRead(pPeb64, sizeof(PEB32), 1);

            // 附加进程
            KeStackAttachProcess(eproc, &kpc);

            DbgPrint("进程基地址: 0x%p \n", pPeb64->ImageBaseAddress);
            DbgPrint("ProcessHeap = 0x%p \n", pPeb64->ProcessHeap);
            DbgPrint("BeingDebugged = %d \n", pPeb64->BeingDebugged);

            // 脱离进程
```

```
            KeUnstackDetachProcess(&kpc);
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        Driver->DriverUnload = UnDriver;
        return STATUS_SUCCESS;
    }

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

PEB64代码运行后，我们加载驱动即可看到如下结果:

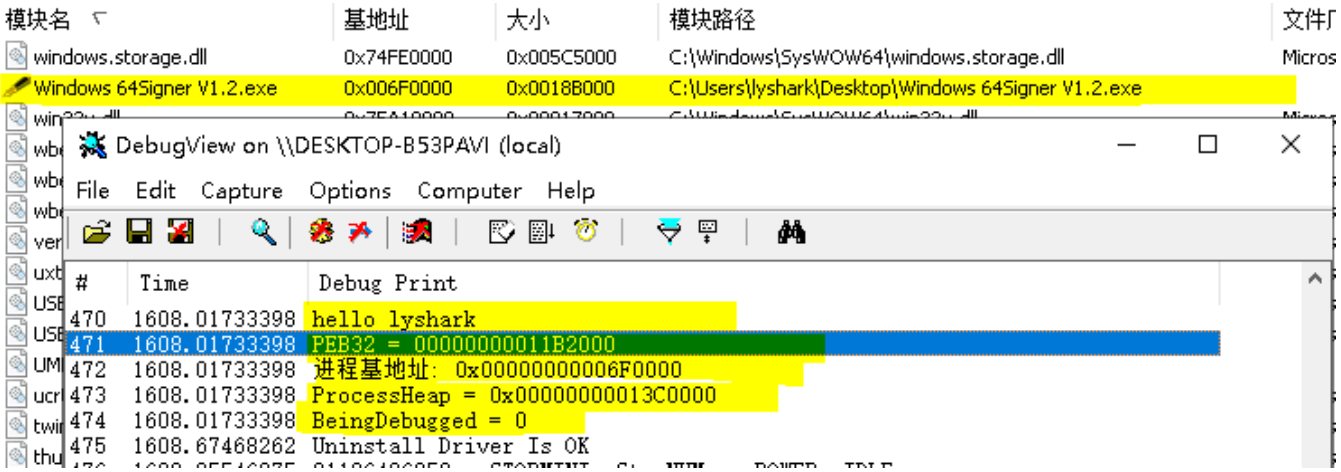


而相对于64位进程来说，获取 32位 进程的PEB信息可以直接调用 `PsGetProcessWow64Process()` 函数得到，该函数已被导出可以任意使用，获取PEB代码如下。

```
#include "peb.h"
#include <ntifs.h>

// 定义导出
NTKERNELAPI PVOID NTAPI PsGetProcessPeb(_In_ PEPROCESS Process);

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint(("Uninstall Driver Is OK \n"));
}

// LyShark
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark \n");

    NTSTATUS status = STATUS_UNSUCCESSFUL;
    PEPROCESS eproc = NULL;
    KAPC_STATE kpc = { 0 };
```

```
    PPEB32 pPeb32 = NULL;

    __try
    {
        // HANDLE)4656 进程PID
        status = PsLookupProcessByProcessId((HANDLE)6164, &eproc);

        // 得到32位PEB
        pPeb32 = (PPEB32)PsGetProcessWow64Process(eproc);

        DbgPrint("PEB32 = %p \n", pPeb32);

        if (pPeb32 != 0)
        {
            // 验证可读性
            ProbeForRead(pPeb32, sizeof(PEB32), 1);

            // 附加进程
            KeStackAttachProcess(eproc, &kpc);

            DbgPrint("进程基地址: 0x%p \n", pPeb32->ImageBaseAddress);
            DbgPrint("ProcessHeap = 0x%p \n", pPeb32->ProcessHeap);
            DbgPrint("BeingDebugged = %d \n", pPeb32->BeingDebugged);

            // 脱离进程
            KeUnstackDetachProcess(&kpc);
        }

    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        Driver->DriverUnload = UnDriver;
        return STATUS_SUCCESS;
    }

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

PEB32代码运行后，我们加载驱动即可看到如下结果:


模块名
基地址
大小
模块路径
文件
windows.storage.dll
0x74FE0000
0x005C5000
C:\Windows\SysWOW64\windows.storage.dll
Micros
Windows 64Signer V1.2.exe
0x006F0000
0x0018B000
C:\Users\lyshark\Desktop\Windows 64Signer V1.2.exe
DebugView on \\DESKTOP-B53PAVI (local)
File Edit Capture Options Computer Help
Time Debug Print
470 1608.01733398 hello lyshark
471 1608.01733398 PEB32 = 00000000011B2000
472 1608.01733398 进程基地址: 0x00000000006F0000
473 1608.01733398 ProcessHeap = 0x00000000013C0000
474 1608.01733398 BeingDebugged = 0
475 1608.67468262 Uninstall Driver Is OK