

本章将探索驱动程序开发的基础部分，了解驱动对象 `DRIVER_OBJECT` 结构体的定义，一般来说驱动程序 `DriverEntry` 入口处都会存在这样一个驱动对象，该对象内所包含的就是当前所加载驱动自身的一些详细参数，例如驱动大小，驱动标志，驱动名，驱动节等等，每一个驱动程序都会存在这样的一个结构。

首先来看一下微软对其的定义，此处我已将重要字段进行了备注。

```
typedef struct _DRIVER_OBJECT {
    CSHORT Type; // 驱动类型
    CSHORT Size; // 驱动大小
    PDEVICE_OBJECT DeviceObject; // 驱动对象
    ULONG Flags; // 驱动的标志
    PVOID DriverStart; // 驱动的起始位置
    ULONG DriverSize; // 驱动的大小
    PVOID DriverSection; // 指向驱动程序映像的内存区对象
    PDRIVER_EXTENSION DriverExtension; // 驱动的扩展空间
    UNICODE_STRING DriverName; // 驱动名字
    PUNICODE_STRING HardwareDatabase;
    PFAST_IO_DISPATCH FastIoDispatch;
    PDRIVER_INITIALIZE DriverInit;
    PDRIVER_STARTIO DriverStartIo;
    PDRIVER_UNLOAD DriverUnload; // 驱动对象的卸载地址
    PDRIVER_DISPATCH MajorFunction[IRP_MJ_MAXIMUM_FUNCTION + 1];
} DRIVER_OBJECT;
```

`DRIVER_OBJECT`结构体是 windows 操作系统内核中用于表示驱动程序的基本信息的结构体。它包含了一系列的字段，用于描述驱动程序的特定属性。

以下是 `DRIVER_OBJECT` 结构体中的一些重要字段：

- `Type`：该字段标识该结构体的类型，始终设置为 `DRIVER_OBJECT_TYPE`。
- `Size`：该字段表示该结构体的大小，以字节为单位。
- `DeviceObject`：该字段是一个指针，指向驱动程序所创建的设备对象链表的头部。每个设备对象代表着一个设备或者驱动程序创建的一种虚拟设备。
- `DriverStart`：该字段是一个指针，指向驱动程序代码的入口点，也就是驱动程序的 `DriverEntry` 函数。该函数会在驱动程序被加载时被调用。
- `DriverSize`：该字段表示驱动程序代码的大小，以字节为单位。
- `DriverName`：该字段是一个 `UNICODE_STRING` 结构体，用于表示驱动程序的名称。
- `Flags`：该字段是一个32位的位掩码，用于表示驱动程序的一些属性。例如，可以设置 `DO_BUFFERED_IO` 标志表示驱动程序支持缓冲 I/O。

如果我们想要遍历出当前自身驱动的一些基本信息，我们只需要在驱动的头部分析 `_DRIVER_OBJECT` 即可得到全部的数据，这段代码可以写成如下样子，其中的 `IRP_MJ_` 这一系列则是微软的调用号，不同的RIP代表着不同的涵义，但一般驱动也就会用到如下这几种调用号。

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com
```

```

#include <ntifs.h>

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

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

    Driver->DriverUnload = UnDriver;

    DbgPrint("驱动名字 = %wZ \n", Driver->DriverName);
    DbgPrint("驱动起始地址 = %p | 大小 = %x | 结束地址 %p \n", Driver->DriverStart, Driver->DriverSize, (ULONG64)Driver->DriverStart + Driver->DriverSize);

    DbgPrint("卸载地址 = %p\n", Driver->DriverUnload);
    DbgPrint("IRP_MJ_READ地址 = %p\n", Driver->MajorFunction[IRP_MJ_READ]);
    DbgPrint("IRP_MJ_WRITE地址 = %p\n", Driver->MajorFunction[IRP_MJ_WRITE]);
    DbgPrint("IRP_MJ_CREATE地址 = %p\n", Driver->MajorFunction[IRP_MJ_CREATE]);
    DbgPrint("IRP_MJ_CLOSE地址 = %p\n", Driver->MajorFunction[IRP_MJ_CLOSE]);
    DbgPrint("IRP_MJ_DEVICE_CONTROL地址 = %p\n", Driver->MajorFunction[IRP_MJ_DEVICE_CONTROL]);

    // 输出完整的调用号
    for (auto i = 0; i < IRP_MJ_MAXIMUM_FUNCTION; i++)
    {
        DbgPrint("IRP_MJ调用号 = %d | 函数地址 = %p \r\n", i, Driver->MajorFunction[i]);
    }

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

编译这段程序，签名并运行，我们即可看到如下输出信息，此时当前自身驱动的详细参数都可以被输出；

DebugView on \\DESKTOP-B53PAVI (local)		
File Edit Capture Options Computer Help		
#	Time	Debug Print
262	4.02525377	hello lyshark
263	4.02525568	驱动名字 = \Driver\WinDDK
264	4.02525711	驱动起始地址 = FFFFF8051FFE0000   大小 = 5000   结束地址 FFFFF8051FFE5000
265	4.02525806	卸载地址 = FFFFF8051FFE1000
266	4.02525902	IRP_MJ_READ地址 = FFFFF8051B11E8A0
267	4.02525997	IRP_MJ_WRITE地址 = FFFFF8051B11E8A0
268	4.02526093	IRP_MJ_CREATE地址 = FFFFF8051B11E8A0
269	4.02526140	IRP_MJ_CLOSE地址 = FFFFF8051B11E8A0
270	4.02526236	IRP_MJ_DEVICE_CONTROL地址 = FFFFF8051B11E8A0
271	4.02526379	IRP_MJ调用号 = 0   函数地址 = FFFFF8051B11E8A0
272	4.02526474	IRP_MJ调用号 = 1   函数地址 = FFFFF8051B11E8A0
273	4.02526569	IRP_MJ调用号 = 2   函数地址 = FFFFF8051B11E8A0
274	4.02526617	IRP_MJ调用号 = 3   函数地址 = FFFFF8051B11E8A0
275	4.02526712	IRP_MJ调用号 = 4   函数地址 = FFFFF8051B11E8A0
276	4.02526808	IRP_MJ调用号 = 5   函数地址 = FFFFF8051B11E8A0

当然运用 `_DRIVER_OBJECT` 对象中的 `DriverSection` 字段我们完全可以遍历输出当前系统下所有的驱动程序的具体信息，`DriverSection` 结构指向了一个 `_LDR_DATA_TABLE_ENTRY` 结构，结构的微软定义如下；

```
typedef struct _LDR_DATA_TABLE_ENTRY {
    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID DllBase;
    PVOID EntryPoint;
    ULONG SizeOfImage;
    UNICODE_STRING FullDllName;
    UNICODE_STRING BaseDllName;
    ULONG Flags;
    USHORT LoadCount;
    USHORT TlsIndex;
    union {
        LIST_ENTRY HashLinks;
        struct {
            PVOID SectionPointer;
            ULONG CheckSum;
        };
    };
    union {
        struct {
            ULONG TimeDateStamp;
        };
        struct {
            PVOID LoadedImports;
        };
    };
};
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
```

为了能够遍历出所有的系统驱动，我们需要得到 pLdr 结构，该结构可通过 Driver->DriverSection 的方式获取到，获取到之后通过 pLdr->InLoadOrderLinks.Flink 得到当前驱动的入口地址，而每一次调用 pListEntry->Flink 都将会指向下一个驱动对象，通过不断地循环 CONTAINING\_RECORD 解析，即可输出当前系统内所有驱动的详细信息。这段程序的写法可以如下所示；

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

#include <ntifs.h>

typedef struct _LDR_DATA_TABLE_ENTRY {
    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID DllBase;
    PVOID EntryPoint;
    ULONG SizeOfImage;
    UNICODE_STRING FullDllName;
    UNICODE_STRING BaseDllName;
    ULONG Flags;
    USHORT LoadCount;
    USHORT TlsIndex;
    union {
        LIST_ENTRY HashLinks;
        struct {
            PVOID SectionPointer;
            ULONG CheckSum;
        };
    };
};

union {
    struct {
        ULONG TimeDateStamp;
    };
    struct {
        PVOID LoadedImports;
    };
};
}LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;

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

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

    Driver->DriverUnload = UnDriver;
```

```

PLDR_DATA_TABLE_ENTRY pLdr = NULL;
PLIST_ENTRY pListEntry = NULL;
PLIST_ENTRY pCurrentListEntry = NULL;

PLDR_DATA_TABLE_ENTRY pCurrentModule = NULL;
pLdr = (PLDR_DATA_TABLE_ENTRY)Driver->DriverSection;
pListEntry = pLdr->InLoadOrderLinks.Flink;
pCurrentListEntry = pListEntry->Flink;

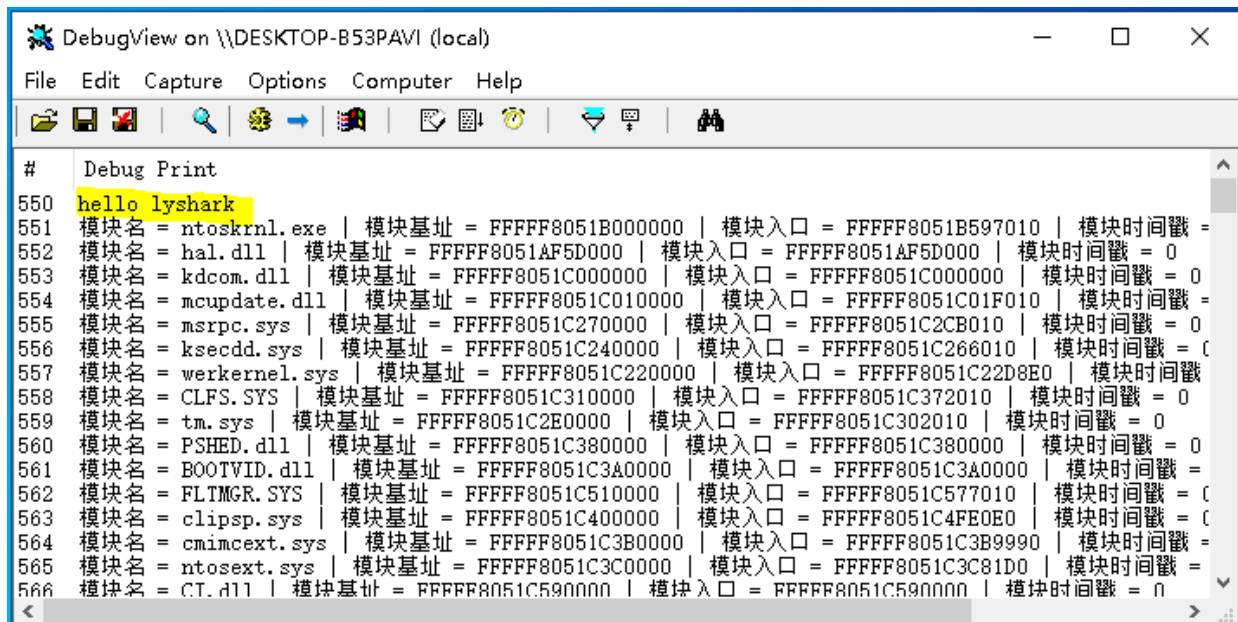
// 判断是否结束
while (pCurrentListEntry != pListEntry)
{
    // 获取LDR_DATA_TABLE_ENTRY结构
    pCurrentModule = CONTAINING_RECORD(pCurrentListEntry, LDR_DATA_TABLE_ENTRY,
InLoadOrderLinks);

    if (pCurrentModule->BaseDllName.Buffer != 0)
    {
        DbgPrint("模块名 = %wZ | 模块基址 = %p | 模块入口 = %p | 模块时间戳 = %d \n",
            pCurrentModule->BaseDllName,
            pCurrentModule->DllBase,
            pCurrentModule->EntryPoint,
            pCurrentModule->TimeDateStamp);
    }
    pCurrentListEntry = pCurrentListEntry->Flink;
}

Driver->DriverUnload = UnDriver;
return STATUS_SUCCESS;
}

```

编译这段程序，签名并运行，我们即可看到如下输出信息，此时当前自身驱动的详细参数都可以被输出；



The screenshot shows the DebugView application window titled "DebugView on \\DESKTOP-B53PAVI (local)". The window has a menu bar with "File", "Edit", "Capture", "Options", "Computer", and "Help". Below the menu bar is a toolbar with various icons. The main area displays a "Debug Print" window with a list of modules loaded by the system. The list is as follows:

#	Module Name	Module Base Address	Module Entry Point	Module Time Stamp
550	hello lyshark	FFFFF8051B000000	FFFFF8051B597010	0
551	ntoskrnl.exe	FFFFF8051AF5D000	FFFFF8051AF5D000	0
552	hal.dll	FFFFF8051C000000	FFFFF8051C000000	0
553	kdcom.dll	FFFFF8051C010000	FFFFF8051C01F010	0
554	mcupdate.dll	FFFFF8051C270000	FFFFF8051C2CB010	0
555	msrpc.sys	FFFFF8051C240000	FFFFF8051C266010	0
556	ksecdd.sys	FFFFF8051C220000	FFFFF8051C22D8E0	0
557	werkernel.sys	FFFFF8051C310000	FFFFF8051C372010	0
558	CLFS.SYS	FFFFF8051C302010	FFFFF8051C302010	0
559	tm.sys	FFFFF8051C380000	FFFFF8051C380000	0
560	PSHED.dll	FFFFF8051C3A0000	FFFFF8051C3A0000	0
561	BOOTVID.dll	FFFFF8051C510000	FFFFF8051C577010	0
562	FLTMGR.SYS	FFFFF8051C400000	FFFFF8051C4FE0E0	0
563	clipsys.sys	FFFFF8051C3B0000	FFFFF8051C3B9990	0
564	cmimcext.sys	FFFFF8051C3C0000	FFFFF8051C3C81D0	0
565	ntosext.sys	FFFFF8051C590000	FFFFF8051C590000	0
566	CT.dll			0

通过使用上一篇文章《驱动开发：内核字符串拷贝与比较》中所介绍的 `RtlCompareUnicodeString` 函数，还可用于对比与过滤特定结果，以此来实现通过驱动名返回驱动基址的功能。

```
LONGLONG GetModuleBaseByName(PDRIVER_OBJECT pDriverObj, UNICODE_STRING ModuleName)
{
    PLDR_DATA_TABLE_ENTRY pLdr = NULL;
    PLIST_ENTRY pListEntry = NULL;
    PLIST_ENTRY pCurrentListEntry = NULL;

    PLDR_DATA_TABLE_ENTRY pCurrentModule = NULL;
    pLdr = (PLDR_DATA_TABLE_ENTRY)pDriverObj->DriverSection;
    pListEntry = pLdr->InLoadOrderLinks.Flink;
    pCurrentListEntry = pListEntry->Flink;

    while (pCurrentListEntry != pListEntry)
    {
        // 获取LDR_DATA_TABLE_ENTRY结构
        pCurrentModule = CONTAINING_RECORD(pCurrentListEntry, LDR_DATA_TABLE_ENTRY,
        InLoadOrderLinks);

        if (pCurrentModule->BaseDllName.Buffer != 0)
        {
            // 对比模块名
            if (RtlCompareUnicodeString(&pCurrentModule->BaseDllName, &ModuleName,
            TRUE) == 0)
            {
                return (LONGLONG)pCurrentModule->DllBase;
            }
        }
        pCurrentListEntry = pCurrentListEntry->Flink;
    }
    return 0;
}
```

上这段代码的使用也非常简单，通过传入一个 `UNICODE_STRING` 类型的模块名，即可获取到模块基址并返回，至于如何初始化 `UNICODE_STRING` 则在《驱动开发：内核字符串转换方法》中有详细的介绍，此处你只需要这样来写。

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

    UNICODE_STRING unicode;

    // 获取winDDK驱动基地址
    RtlUnicodeStringInit(&unicode, L"winDDK.sys");
    LONGLONG winddk_address = GetModuleBaseByName(Driver, unicode);
    DbgPrint("winDDK模块基址 = %p \n", winddk_address);

    // 获取ACPI驱动基地址
    RtlUnicodeStringInit(&unicode, L"ACPI.sys");
```

```
LONGLONG acpi_address = GetModuleBaseByName(Driver, unicode);
DbgPrint("ACPI模块基址 = %p \n", acpi_address);

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

运行这段驱动程序，即可分别输出 winDDK.sys 以及 ACPI.sys 两个驱动模块的基地址；

