注册表是Windows中的一个重要的数据库，用于存储系统和应用程序的设置信息，注册表是一个巨大的树形结构，无论在应用层还是内核层操作注册表都有独立的API函数可以使用，而在内核中读写注册表则需要使用内核装用API函数，如下将依次介绍并封装一些案例，实现对注册表的创建，删除，更新，查询等操作。

在Windows内核中，注册表是一种存储系统配置信息的机制，包括应用程序、硬件、驱动程序和操作系统的各种设置。内核提供了一些API函数，可以让驱动程序通过代码访问和修改注册表，以实现系统的配置和管理。下面简单介绍一下内核中的注册表增删改查操作：

注册表查询

- 在内核中，可以使用ZwQueryValueKey或ZwEnumerateValueKey函数查询指定键的值。其中，ZwQueryValueKey函数可以查询指定键的值，而ZwEnumerateValueKey函数可以枚举指定键下的所有值。这两个函数都需要指定键的句柄和要查询的值的名称，查询结果将返回在指定的缓冲区中。

注册表修改

- 在内核中，可以使用ZwSetValueKey函数修改指定键的值。该函数需要指定键的句柄、要修改的值的名称、值的类型和值的数据。在修改注册表时，需要注意权限和安全性问题，以避免潜在的安全问题。

注册表添加

- 在内核中，可以使用ZwCreateKey函数创建一个新的键。该函数需要指定要创建键的父键的句柄、新键的名称、新键的属性等信息。如果成功创建了新键，则可以使用ZwSetValueKey函数向其添加值。

注册表删除

- 在内核中，可以使用ZwDeleteValueKey函数删除指定键的值，或使用ZwDeleteKey函数删除指定键及其下面的所有子键和值。这两个函数都需要指定要删除的键的句柄或路径。在删除注册表时，同样需要注意权限和安全性问题，以避免潜在的安全问题。

需要注意的是，对注册表的操作可能会对系统的稳定性产生影响。因此，在实现这些技术时，需要遵循操作系统和安全软件的规定，以确保系统的安全和稳定。

ZwCreateKey

创建注册表Key键，内核函数`ZwCreateKey`可用于创建新的注册表项或打开现有注册表项。

ZwCreateKey是Windows内核中的一个函数，用于创建一个新的注册表键（registry key）。它通常被驱动程序使用来添加新的配置信息或者修改已有的配置信息。

以下是ZwCreateKey函数的一般形式：

```
NTSTATUS ZwCreateKey(
  _Out_ PHANDLE            KeyHandle,
  _In_  ACCESS_MASK        DesiredAccess,
  _In_  POBJECT_ATTRIBUTES ObjectAttributes,
  _Reserved_ ULONG         TitleIndex,
  _In_  PUNICODE_STRING    Class,
  _In_  ULONG              CreateOptions,
  _Out_ PULONG             Disposition
);
```

参数说明：

- KeyHandle： 输出参数，指向新创建的注册表键的句柄（handle）。

- DesiredAccess： 指定新创建的键所需的访问权限，比如KEY_QUERY_VALUE等，具体请参考MSDN文档。
- ObjectAttributes： 指向一个OBJECT_ATTRIBUTES结构体的指针，该结构体包含了注册表键的一些属性信息，比如名称、路径等。
- TitleIndex： 指定键的标题索引。
- Class： 指向一个UNICODE_STRING结构体的指针，它用于指定新创建的键的类名。
- CreateOptions： 指定创建键的选项，比如REG_OPTION_NON_VOLATILE等。
- Disposition： 输出参数，指向一个ULONG类型的指针，返回创建的键的状态信息，比如REG_CREATED_NEW_KEY等。

函数执行成功时，将返回STATUS_SUCCESS，否则返回相应的错误代码。需要注意的是，在使用ZwCreateKey函数之前，必须先初始化OBJECT_ATTRIBUTES结构体，以包含要创建的注册表键的完整路径。

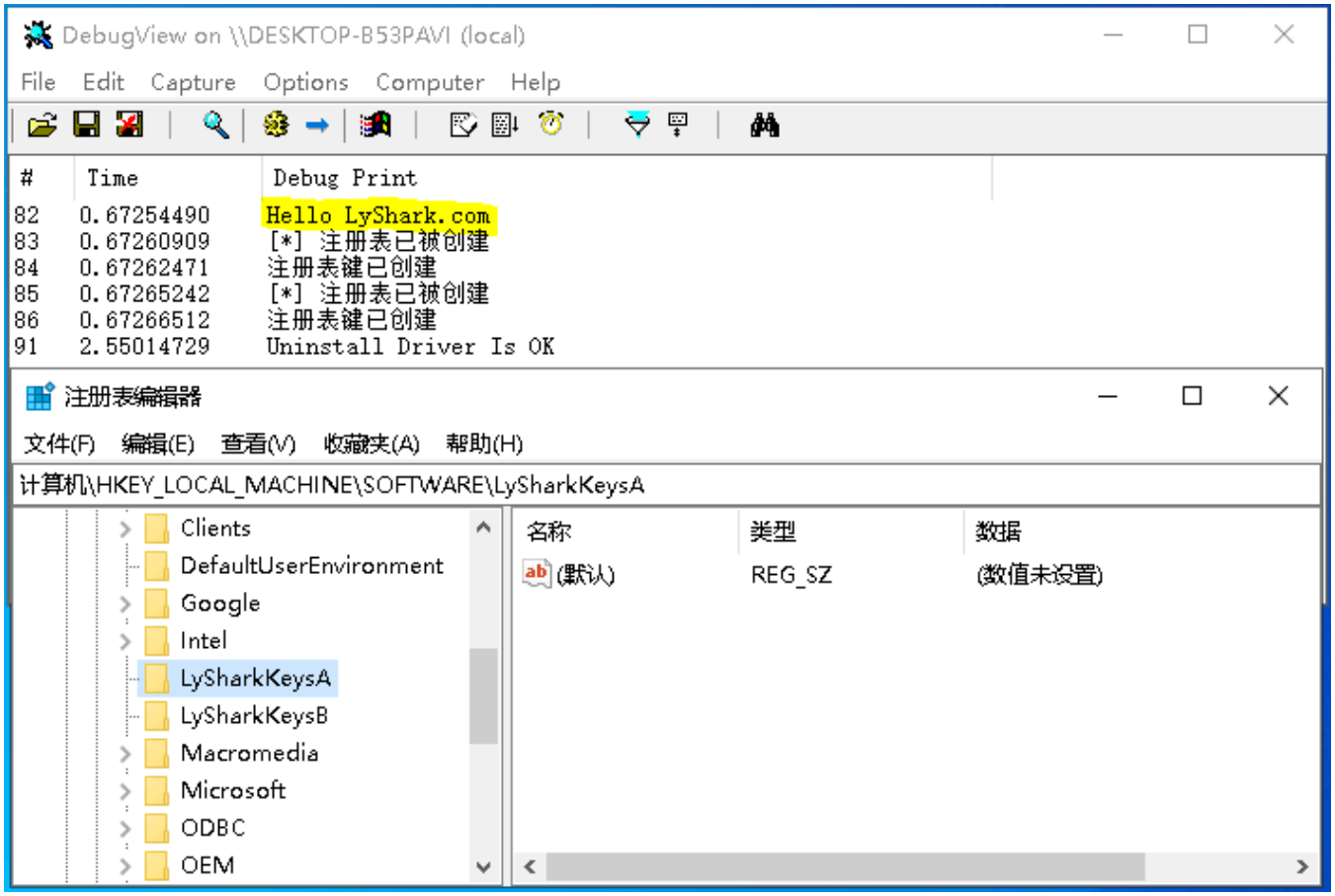
在使用ZwCreateKey函数时，需要注意权限和安全性问题，以避免潜在的安全问题。同时，需要仔细考虑键的类名、访问权限和创建选项等参数的设置，以确保所创建的键能够正确地满足应用程序的需求。

```
#include <ntifs.h>

// 创建或者打开已存在注册表键
BOOLEAN MyCreateRegistryKeyA(UNICODE_STRING ustrRegistry)
{
    HANDLE hRegister = NULL;
    OBJECT_ATTRIBUTES objectAttributes = { 0 };
    ULONG ulResult = 0;
    NTSTATUS status = STATUS_SUCCESS;

    // 创建或者打开已存在注册表键
    InitializeObjectAttributes(&objectAttributes, &ustrRegistry, OBJ_CASE_INSENSITIVE, NULL,
NULL);

    // 创建Key
    status = ZwCreateKey(&hRegister,
        KEY_ALL_ACCESS,
        &objectAttributes,
        0,
        NULL,
        REG_OPTION_NON_VOLATILE,
        &ulResult);
    if (!NT_SUCCESS(status))
    {
        return FALSE;
    }
    if (REG_CREATED_NEW_KEY == ulResult)
    {
        DbgPrint("[*] 注册表已被创建 \n");
    }
    else if (REG_OPENED_EXISTING_KEY == ulResult)
    {
        DbgPrint("[*] 注册表打开 \n");
    }

    // 关闭注册表键句柄
```

```
    ZwClose(hRegister);
    return TRUE;
}

// 创建键值对
BOOLEAN MyCreateRegistryKeyB(LPWSTR KeyName)
{
    OBJECT_ATTRIBUTES objectAttributes;
    UNICODE_STRING usKeyName;
    NTSTATUS ntStatus;
    HANDLE hRegister;

    RtlInitUnicodeString(&usKeyName, KeyName);

    // 初始化
    InitializeObjectAttributes(&objectAttributes, &usKeyName, OBJ_CASE_INSENSITIVE, NULL,
NULL);

    // 创建Key
    ntStatus = ZwCreateKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes, 0, NULL,
REG_OPTION_NON_VOLATILE, NULL);
    if (NT_SUCCESS(ntStatus))
    {
        DbgPrint("[*] 注册表已被创建 \n");
        ZwClose(hRegister);
        return TRUE;
    }
    else
    {
        DbgPrint("[*] 注册表创建失败 \n");
        return FALSE;
    }
    return FALSE;
}

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

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark.com \n");

    BOOLEAN flag = FALSE;

    // 创建注册表键
    UNICODE_STRING ustrRegistry;
    RtlInitUnicodeString(&ustrRegistry, L"\\Registry\\Machine\\Software\\LySharkKeysA");
    flag = MyCreateRegistryKeyA(ustrRegistry);
    if (flag == TRUE)
    {
        DbgPrint("注册表键已创建 \n");
```

```
    }

    // 创建注册表键
    flag = MyCreateRegistryKeyB(L"\\Registry\\Machine\\Software\\LySharkKeysB");
    if (flag == TRUE)
    {
        DbgPrint("注册表键已创建 \n");
    }

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

运行如上代码即可在`计算机\HKEY_LOCAL_MACHINE\SOFTWARE\`目录下分别创建`LySharkKeysA`和`LySharkKeysB`两个空目录，输出效果如下图；



ZwDeleteKey

删除注册表Key键，内核函数`ZwDeleteKey`可从注册表中删除打开的项。

ZwDeleteKey是Windows内核中的一个函数，用于删除指定的注册表键（registry key）。它通常被驱动程序使用来删除不再需要的配置信息或者清理无用的键。

以下是ZwDeleteKey函数的一般形式：

```
NTSTATUS ZwDeleteKey(
  _In_ HANDLE             KeyHandle
);
```

参数说明：

- KeyHandle：要删除的键的句柄（handle）。

函数执行成功时，将返回STATUS_SUCCESS，否则返回相应的错误代码。需要注意的是，在使用ZwDeleteKey函数之前，需要先打开要删除的键，获取其句柄。

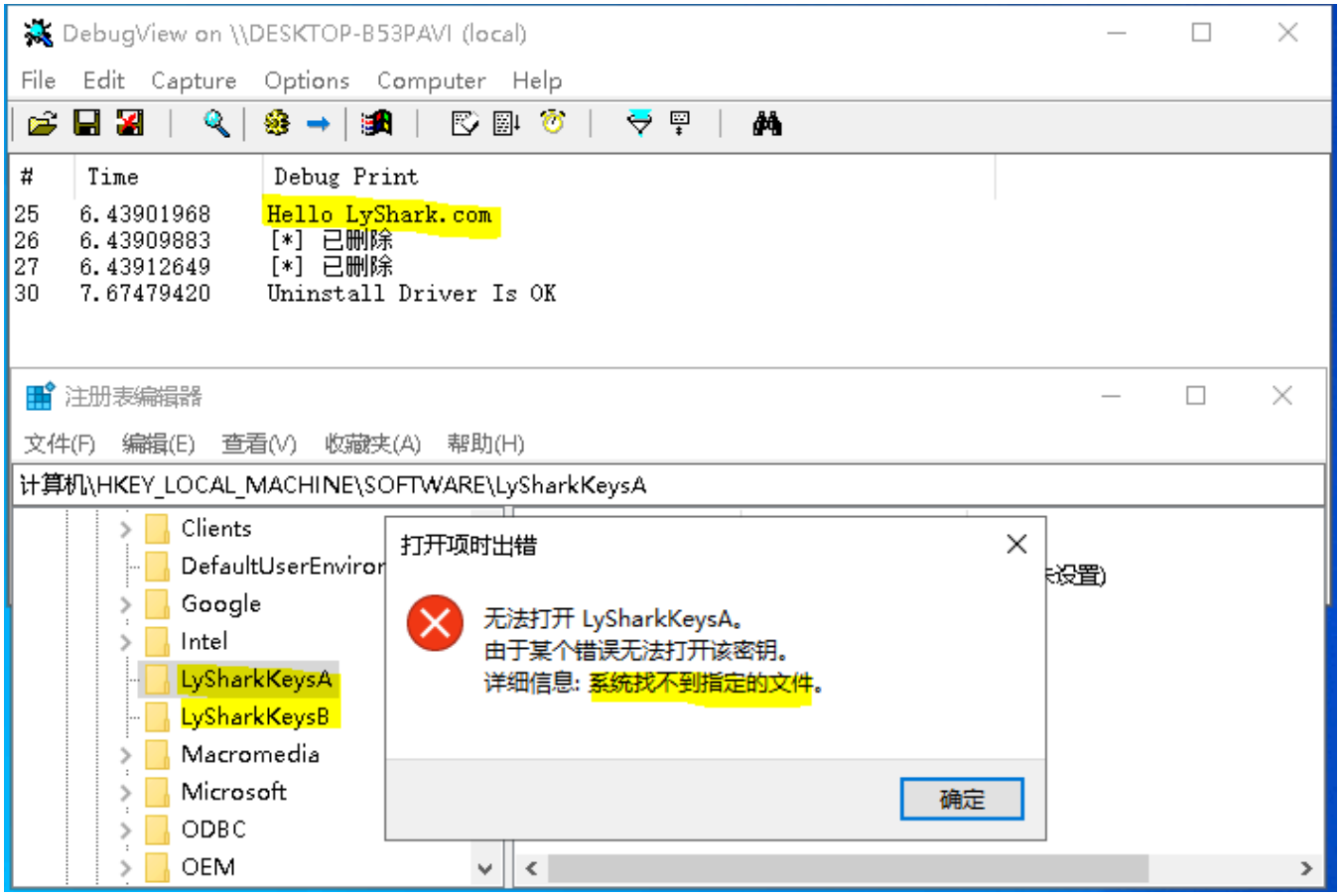
在使用ZwDeleteKey函数时，需要注意权限和安全性问题，以避免潜在的安全问题。同时，需要仔细考虑键的名称和路径等信息，确保要删除的键是正确的，并且不会对系统造成不良影响。

另外，需要注意的是，ZwDeleteKey函数只能用于删除空的注册表键。如果要删除非空的键，需要先递归地删除该键下的所有子键和值。

```
#include <ntifs.h>

// 删除注册表键
BOOLEAN MyDeleteRegistryKeyA(UNICODE_STRING ustrRegistry)
{
    HANDLE hRegister = NULL;
    OBJECT_ATTRIBUTES objectAttributes = { 0 };
    NTSTATUS status = STATUS_SUCCESS;

    // 打开注册表键
    InitializeObjectAttributes(&objectAttributes, &ustrRegistry, OBJ_CASE_INSENSITIVE, NULL,
NULL);
    status = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (!NT_SUCCESS(status))
    {
        return FALSE;
    }
    // 删除注册表键
    status = ZwDeleteKey(hRegister);
    if (!NT_SUCCESS(status))
    {
        ZwClose(hRegister);
        return FALSE;
    }
    // 关闭注册表键句柄
    ZwClose(hRegister);
    return TRUE;
}

// 删除注册表键
BOOLEAN MyDeleteRegistryKeyB(LPWSTR KeyName)
{
    OBJECT_ATTRIBUTES objectAttributes;
    UNICODE_STRING usKeyName;
    NTSTATUS ntStatus;
    HANDLE hRegister;

    RtlInitUnicodeString(&usKeyName, KeyName);

    // 初始化
```

```
    InitializeObjectAttributes(&objectAttributes, &usKeyName, OBJ_CASE_INSENSITIVE, NULL,
NULL);

    // 打开Key
    ntStatus = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (NT_SUCCESS(ntStatus))
    {
        ntStatus = ZwDeleteKey(hRegister);
        ZwClose(hRegister);
        return TRUE;
    }
    else
    {
        return FALSE;
    }
    return FALSE;
}

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

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark.com \n");

    BOOLEAN flag = FALSE;

    // 删除注册表键
    UNICODE_STRING ustrRegistry;
    RtlInitUnicodeString(&ustrRegistry, L"\\Registry\\Machine\\Software\\LySharkKeysA");
    flag = MyDeleteRegistryKeyA(ustrRegistry);
    if (flag == TRUE)
    {
        DbgPrint("[*] 已删除 \n");
    }

    // 删除注册表键
    flag = MyDeleteRegistryKeyB(L"\\Registry\\Machine\\Software\\LySharkKeysB");
    if (flag == TRUE)
    {
        DbgPrint("[*] 已删除 \n");
    }

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

编译并运行如上程序，则可将`ZwCreateKey`创建的Key键删除，当尝试再次打开`LySharkKeysB`则会提示打开失败，输出效果如下所示；



ZwRenameKey

重命名注册表Key键，内核函数 `ZwRenameKey` 可修改特定注册表键名，此函数需要自行导出。

ZwRenameKey是Windows内核中的一个函数，用于重命名一个指定的注册表键。它通常被驱动程序使用来更改配置信息或者重命名键。

以下是ZwRenameKey函数的一般形式：

```
NTSTATUS ZwRenameKey(
  _In_ HANDLE            KeyHandle,
  _In_ PUNICODE_STRING   NewName
);
```

参数说明：

- KeyHandle： 要重命名的键的句柄（handle）。
- NewName： 新键名称的Unicode字符串。

函数执行成功时，将返回STATUS_SUCCESS，否则返回相应的错误代码。需要注意的是，在使用ZwRenameKey函数之前，需要先打开要重命名的键，获取其句柄。

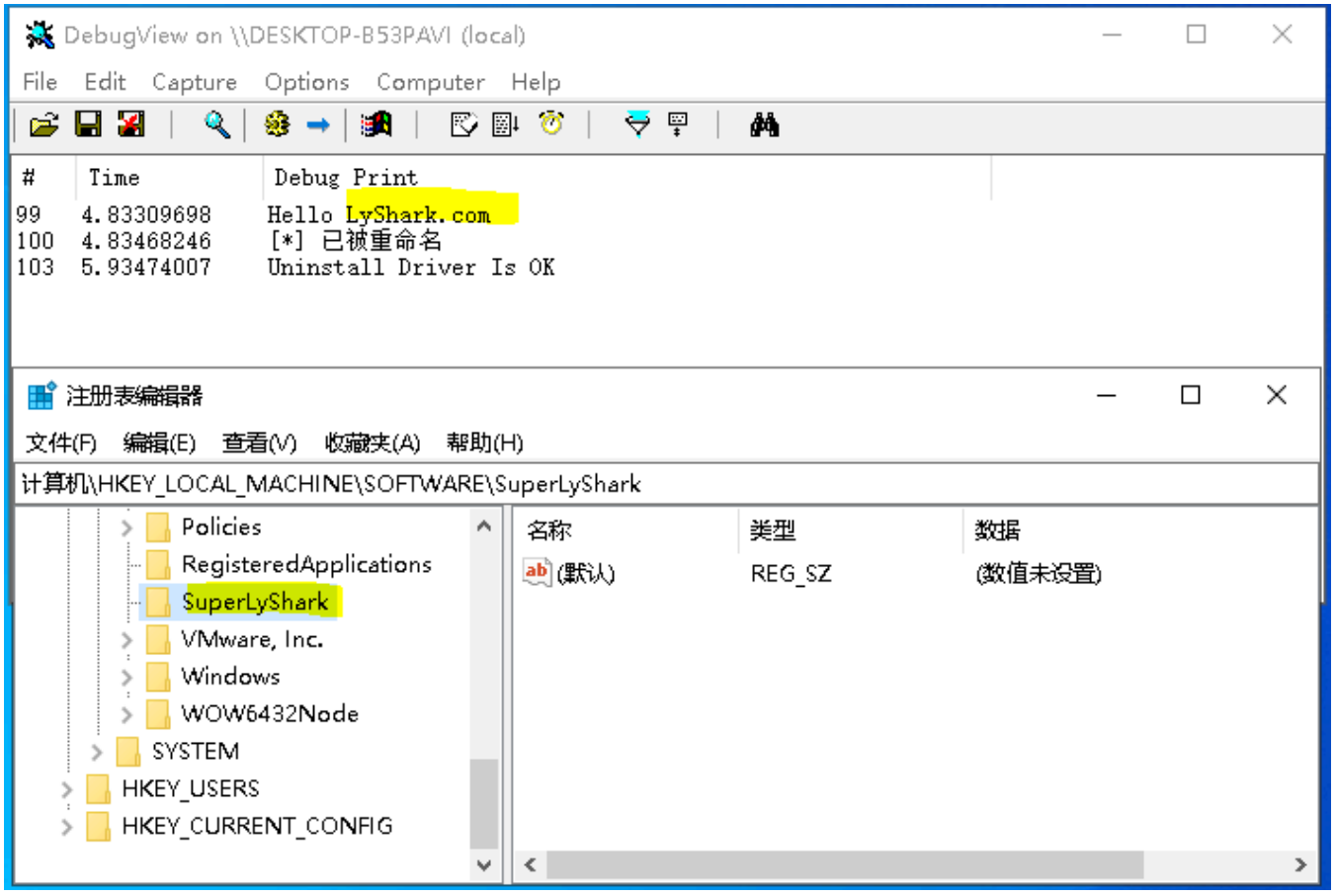
在使用ZwRenameKey函数时，需要注意权限和安全性问题，以避免潜在的安全问题。同时，需要仔细考虑键的名称和路径等信息，确保要重命名的键是正确的，并且不会对系统造成不良影响。另外，需要确保新键名称是唯一的，且符合注册表键名称的规范。

需要注意的是，ZwRenameKey函数只能用于重命名单个键，如果需要批量重命名键，则需要自行实现递归操作。

```
#include <ntifs.h>

// ZwRenameKey 需要自己导出
typedef NTSTATUS(__fastcall *ZWRENAMEKEY)(HANDLE KeyHandle, PUNICODE_STRING NewName);

ZWRENAMEKEY MyZwRenameKey = NULL;

// 根据函数名得到函数内存地址
PVOID GetFunctionAddr(PCWSTR FunctionName)
{
    UNICODE_STRING UniCodeFunctionName;
    RtlInitUnicodeString(&UniCodeFunctionName, FunctionName);
    return MmGetSystemRoutineAddress(&UniCodeFunctionName);
}

// 重命名注册表Key
BOOLEAN RegRenameKey(LPWSTR OldKeyName, LPWSTR NewKeyName)
{
    OBJECT_ATTRIBUTES objectAttributes;
    HANDLE hRegister;
    NTSTATUS ntStatus;
    UNICODE_STRING usOldKeyName, usNewKeyName;

    RtlInitUnicodeString(&usOldKeyName, OldKeyName);
    RtlInitUnicodeString(&usNewKeyName, NewKeyName);

    InitializeObjectAttributes(&objectAttributes, &usOldKeyName, OBJ_CASE_INSENSITIVE, NULL,
NULL);

    // 得到函数内存地址
    MyZwRenameKey = (ZWRENAMEKEY)GetFunctionAddr(L"ZwRenameKey");

    ntStatus = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (NT_SUCCESS(ntStatus))
    {
        // 重命名Key键
        ntStatus = MyZwRenameKey(hRegister, &usNewKeyName);
        ZwFlushKey(hRegister);
        ZwClose(hRegister);
        return TRUE;
    }
    else
    {
        return FALSE;
    }
    return FALSE;
}

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

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark.com \n");

    BOOLEAN flag = FALSE;

    // 重命名键
    flag = RegRenameKey(L"\\Registry\\Machine\\Software\\LySharkKeysA", L"SuperLyShark");
    if (flag == TRUE)
    {
        DbgPrint("[*] 已被重命名 \n");
    }

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

编译并运行这段驱动程序，自动将`LySharkKeysA`改名为`SuperLyShark`，输出效果如下所示；



ZwSetValueKey

在键中创建Value值，在一个Key中增加一个新的值。

ZwSetValueKey是Windows内核中的一个函数，用于向指定的注册表键中写入值。它通常被驱动程序使用来修改或添加配置信息或者键值。

以下是ZwSetValueKey函数的一般形式：

```
NTSTATUS ZwSetValueKey(
  _In_ HANDLE            KeyHandle,
  _In_ PUNICODE_STRING   ValueName,
  _In_opt_ ULONG         TitleIndex,
  _In_ ULONG             Type,
  _In_opt_ PVOID         Data,
  _In_ ULONG             DataSize
);
```

参数说明：

- KeyHandle： 要写入值的键的句柄（handle）。
- ValueName： 要写入值的名称的Unicode字符串。
- TitleIndex： 零基索引，用于在键的名称列表中查找与ValueName相对应的索引值。
- Type： 要写入的值的类型。
- Data： 要写入的数据的指针。
- DataSize： 要写入的数据的长度。

函数执行成功时，将返回STATUS_SUCCESS，否则返回相应的错误代码。需要注意的是，在使用ZwSetValueKey函数之前，需要先打开要写入值的键，获取其句柄。

在使用ZwSetValueKey函数时，需要注意权限和安全性问题，以避免潜在的安全问题。同时，需要仔细考虑键的名称和路径等信息，确保要写入值的键是正确的，并且不会对系统造成不良影响。另外，需要确保写入的数据类型和长度正确，以避免造成不必要的问题。

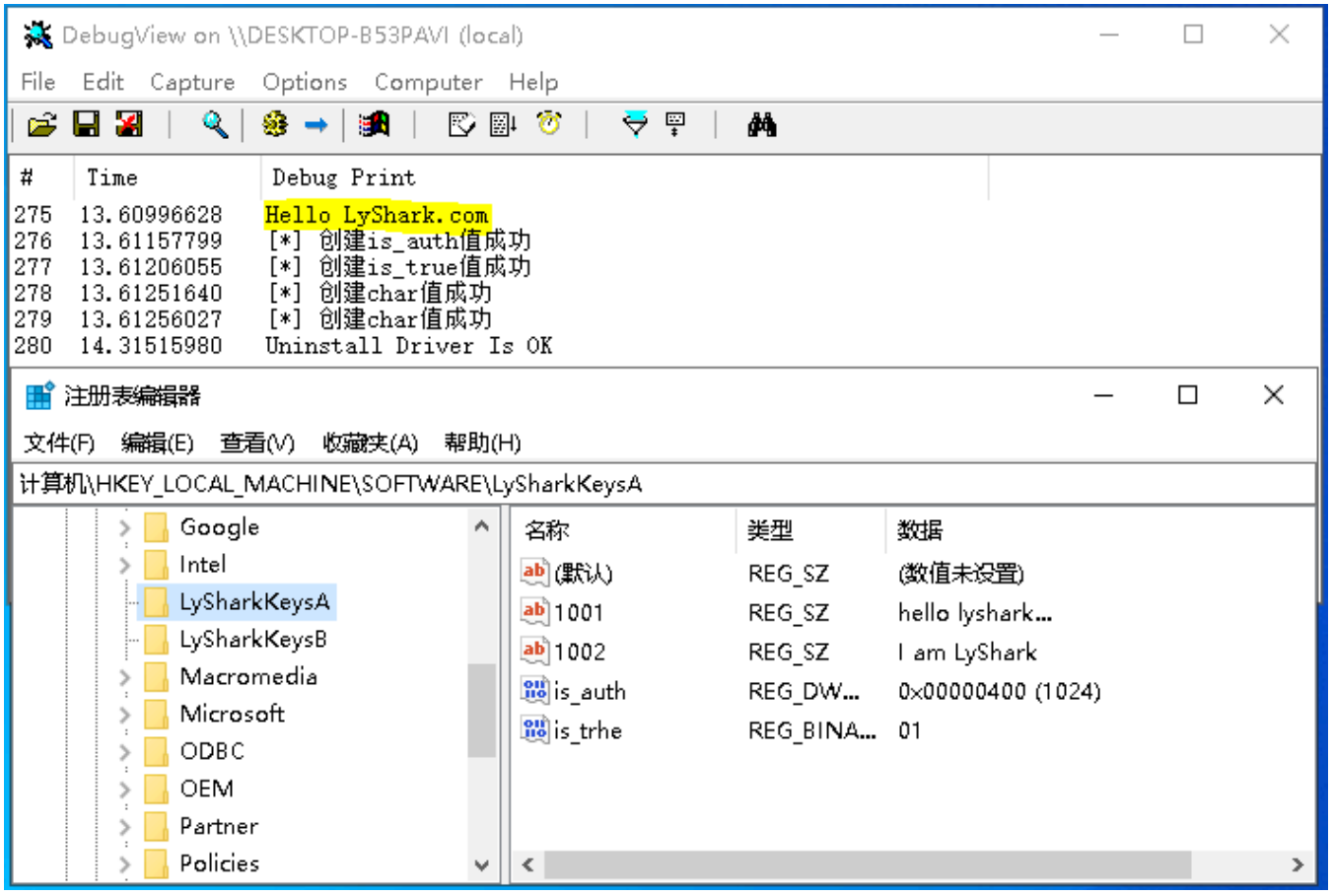
需要注意的是，ZwSetValueKey函数只能用于向单个键写入单个值，如果需要批量写入值，则需要自行实现循环操作。

```
#include <ntifs.h>
#include <windef.h>

// 在键中增加值
BOOLEAN RegSetValueKey(LPWSTR KeyName, LPWSTR ValueName, DWORD DataType, PVOID DataBuffer,
DWORD DataLength)
{
    OBJECT_ATTRIBUTES objectAttributes;
    UNICODE_STRING usKeyName, usValueName;
    NTSTATUS ntStatus;
    HANDLE hRegister;
    RtlInitUnicodeString(&usKeyName, KeyName);
    RtlInitUnicodeString(&usValueName, ValueName);

    InitializeObjectAttributes(&objectAttributes, &usKeyName, OBJ_CASE_INSENSITIVE, NULL,
NULL);

    // 打开
    ntStatus = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (NT_SUCCESS(ntStatus))
    {
        // 设置注册表
```

```
        ntStatus = ZwSetValueKey(hRegister, &usValueName, 0, DataType, DataBuffer,
DataLength);

        // 将请求刷新到磁盘
        ZwFlushKey(hRegister);
        ZwClose(hRegister);
        return TRUE;
    }
    else
    {
        return FALSE;
    }
    return FALSE;
}

// 添加或者修改注册表键值
BOOLEAN MySetRegistryKeyValue(UNICODE_STRING ustrRegistry, UNICODE_STRING ustrKeyValueName,
ULONG ulKeyValueType, PVOID pKeyValueData, ULONG ulKeyValueDataSize)
{
    HANDLE hRegister = NULL;
    OBJECT_ATTRIBUTES objectAttributes = { 0 };
    NTSTATUS status = STATUS_SUCCESS;

    InitializeObjectAttributes(&objectAttributes, &ustrRegistry, OBJ_CASE_INSENSITIVE, NULL,
NULL);

    // 打开注册表键
    status = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (!NT_SUCCESS(status))
    {
        return FALSE;
    }

    // 添加或者修改键值
    status = ZwSetValueKey(hRegister, &ustrKeyValueName, 0, ulKeyValueType, pKeyValueData,
ulKeyValueDataSize);
    if (!NT_SUCCESS(status))
    {
        ZwClose(hRegister);
        return FALSE;
    }

    // 关闭注册表键句柄
    ZwClose(hRegister);
    return TRUE;
}

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

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
```

```
{
    DbgPrint("Hello LyShark.com \n");

    BOOLEAN flag = FALSE;

    DWORD set_dw = 1024;
    BOOLEAN is_true = TRUE;
    WCHAR sz_char[256] = L"hello lyshark";

    // 新建设置value
    flag = RegSetValueKey(L"\\Registry\\Machine\\Software\\LySharkKeysA", L"is_auth",
REG_DWORD, &set_dw, sizeof(set_dw));
    if (flag == TRUE)
    {
        DbgPrint("[*] 创建is_auth值成功 \n");
    }

    // 新建设置bool
    flag = RegSetValueKey(L"\\Registry\\Machine\\Software\\LySharkKeysA", L"is_trhe",
REG_BINARY, &is_true, sizeof(is_true));
    if (flag == TRUE)
    {
        DbgPrint("[*] 创建is_true值成功 \n");
    }

    // 新建设置char
    flag = RegSetValueKey(L"\\Registry\\Machine\\Software\\LySharkKeysA", L"1001", REG_SZ,
&sz_char, sizeof(sz_char));
    if (flag == TRUE)
    {
        DbgPrint("[*] 创建char值成功 \n");
    }

    // 添加注册表键值
    UNICODE_STRING ustrRegistry;
    UNICODE_STRING ustrKeyValueName;

    WCHAR wstrKeyValueData[] = L"I am LyShark";
    RtlInitUnicodeString(&ustrKeyValueName, L"1002");
    RtlInitUnicodeString(&ustrRegistry, L"\\Registry\\Machine\\Software\\LySharkKeysA");

    flag = MySetRegistryKeyValue(ustrRegistry, ustrKeyValueName, REG_SZ, wstrKeyValueData,
sizeof(wstrKeyValueData));
    if (flag == TRUE)
    {
        DbgPrint("[*] 创建char值成功 \n");
    }

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

编译并运行如上代码，即可在`\\Registry\\Machine\\Software\\LySharkKeysA`分别创建一个整数，布尔值，字符串类型，效果图如下；



ZwQueryValueKey

查询某个Key键中的值，调用后可输出特定键中的值。

ZwQueryValueKey是Windows内核中的一个函数，用于从指定的注册表键中读取指定值的数据。它通常被驱动程序使用来获取配置信息或者键值。

以下是ZwQueryValueKey函数的一般形式：

```
NTSTATUS ZwQueryValueKey(
  _In_ HANDLE             KeyHandle,
  _In_ PUNICODE_STRING    ValueName,
  _In_ KEY_VALUE_INFORMATION_CLASS KeyValueInformationClass,
  _Out_opt_ PVOID         KeyValueInformation,
  _In_ ULONG              Length,
  _Out_ PULONG            ResultLength
);
```

参数说明：

- KeyHandle： 要读取值的键的句柄（handle）。
- ValueName： 要读取值的名称的Unicode字符串。
- KeyValueInformationClass： 指定要获取的键值的信息类型。
- KeyValueInformation： 存储读取的键值信息的缓冲区。

- Length： KeyValueInformation缓冲区的大小。
- ResultLength： 实际读取的键值信息的大小。

函数执行成功时，将返回STATUS_SUCCESS，否则返回相应的错误代码。需要注意的是，在使用ZwQueryValueKey函数之前，需要先打开要读取值的键，获取其句柄。

在使用ZwQueryValueKey函数时，需要注意权限和安全性问题，以避免潜在的安全问题。同时，需要仔细考虑键的名称和路径等信息，确保要读取值的键是正确的，并且不会对系统造成不良影响。另外，需要确保KeyValueInformation缓冲区的大小足够，以存储读取的键值信息。

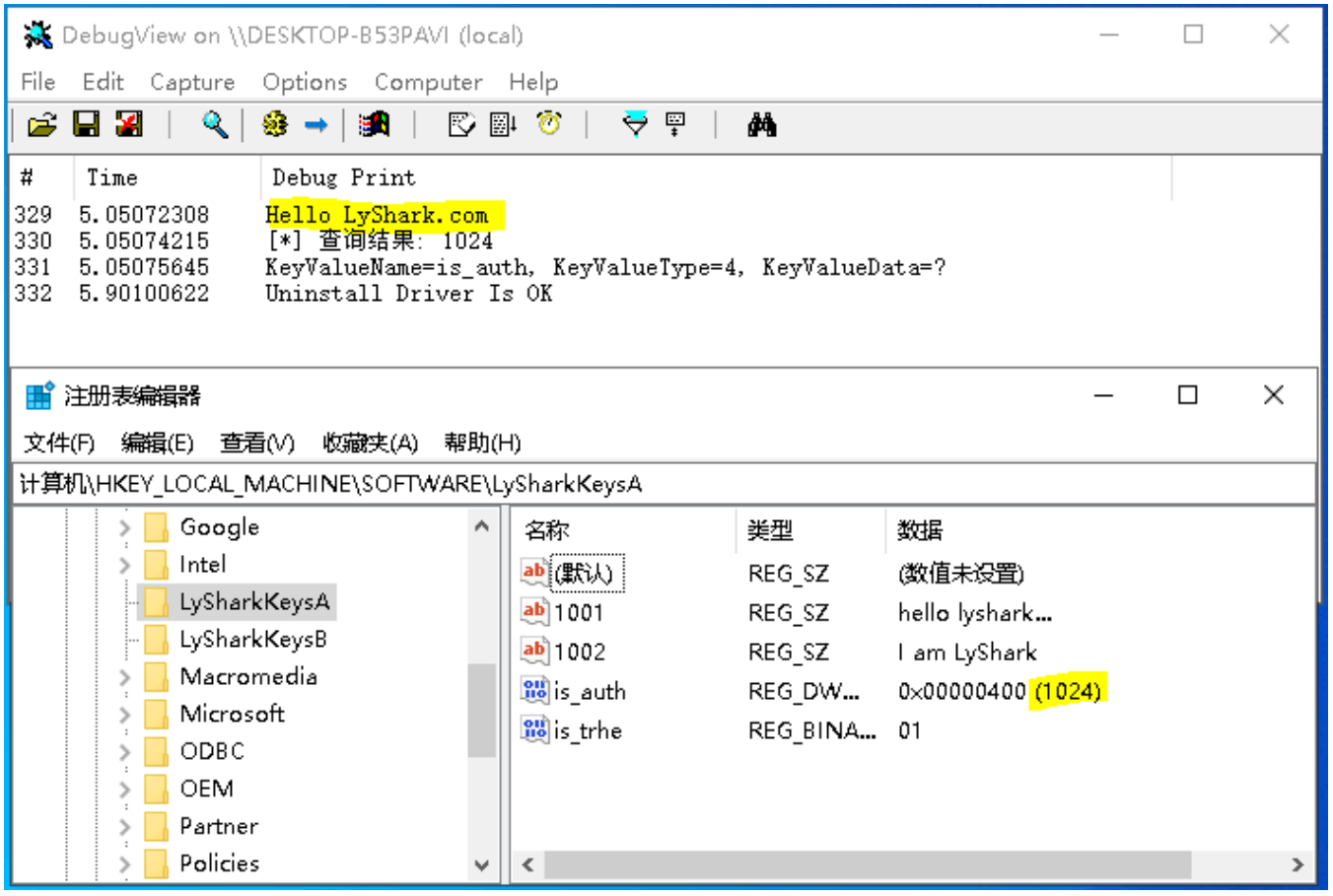
需要注意的是，ZwQueryValueKey函数只能用于读取单个键的单个值，如果需要读取多个键的值，则需要自行实现循环操作。

```
#include <ntifs.h>
#include <windef.h>

// 查询Key键中的Value值
BOOLEAN RegQueryValueKey(LPWSTR KeyName, LPWSTR ValueName, PKEY_VALUE_PARTIAL_INFORMATION
*pkvpi)
{
    ULONG ulSize;
    NTSTATUS ntStatus;
    PKEY_VALUE_PARTIAL_INFORMATION pvpi;
    OBJECT_ATTRIBUTES objectAttributes;
    HANDLE hRegister;
    UNICODE_STRING usKeyName;
    UNICODE_STRING usValueName;

    RtlInitUnicodeString(&usKeyName, KeyName);
    RtlInitUnicodeString(&usValueName, ValueName);

    // 初始化
    InitializeObjectAttributes(&objectAttributes, &usKeyName, OBJ_CASE_INSENSITIVE, NULL,
NULL);

    // 打开注册表Key
    ntStatus = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (!NT_SUCCESS(ntStatus))
    {
        return FALSE;
    }

    // 查询长度
    ntStatus = ZwQueryValueKey(hRegister, &usValueName, KeyValuePartialInformation, NULL, 0,
&ulSize);
    if (ntStatus == STATUS_OBJECT_NAME_NOT_FOUND || ulSize == 0)
    {
        return FALSE;
    }

    // 分配空间保存查询结果
    pvpi = (PKEY_VALUE_PARTIAL_INFORMATION)ExAllocatePool(PagedPool, ulSize);
```

```
    ntStatus = ZwQueryValueKey(hRegister, &usValueName, KeyValuePartialInformation, pvpi,
ulSize, &ulSize);
    if (!NT_SUCCESS(ntStatus))
    {
        return FALSE;
    }

    // 这里的pvpi未被释放，可在外部释放
    // 执行 ExFreePool(pvpi); 释放
    *pkvpi = pvpi;
    return TRUE;
}

// 查询注册表键值
BOOLEAN MyQueryRegistryKeyValue(UNICODE_STRING ustrRegistry, UNICODE_STRING
ustrKeyValueName)
{
    HANDLE hRegister = NULL;
    OBJECT_ATTRIBUTES objectAttributes = { 0 };
    NTSTATUS status = STATUS_SUCCESS;
    ULONG ulBufferSize = 0;
    PKEY_VALUE_PARTIAL_INFORMATION pKeyValuePartialInfo = NULL;

    // 初始化
    InitializeObjectAttributes(&objectAttributes, &ustrRegistry, OBJ_CASE_INSENSITIVE, NULL,
NULL);

    // 打开注册表Key
    status = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (!NT_SUCCESS(status))
    {
        return FALSE;
    }
    // 先获取查询注册表键值所需缓冲区的大小
    status = ZwQueryValueKey(hRegister, &ustrKeyValueName, KeyValuePartialInformation, NULL,
0, &ulBufferSize);
    if (0 == ulBufferSize)
    {
        ZwClose(hRegister);
        return FALSE;
    }

    // 申请缓冲区
    pKeyValuePartialInfo = (PKEY_VALUE_PARTIAL_INFORMATION)ExAllocatePool(NonPagedPool,
ulBufferSize);

    // 查询注册表键值并获取查询结果
    status = ZwQueryValueKey(hRegister, &ustrKeyValueName, KeyValuePartialInformation,
pKeyValuePartialInfo, ulBufferSize, &ulBufferSize);
    if (!NT_SUCCESS(status))
    {
        ExFreePool(pKeyValuePartialInfo);
        ZwClose(hRegister);
```

```
        return FALSE;
    }
    // 显示查询结果
    DbgPrint("KeyValueName=%wZ, KeyValueType=%d, KeyValueData=%S\n", &ustrKeyValueName,
pKeyValuePartialInfo->Type, pKeyValuePartialInfo->Data);

    // 释放内存，关闭句柄
    ExFreePool(pKeyValuePartialInfo);
    ZwClose(hRegister);
    return TRUE;
}

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

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark.com \n");

    BOOLEAN flag = FALSE;
    DWORD get_dw = 0;

    PKEY_VALUE_PARTIAL_INFORMATION pkvi;

    // 查询设置
    flag = RegQueryValueKey(L"\\Registry\\Machine\\Software\\LySharkKeysA", L"is_auth",
&pkvi);
    if (flag == TRUE)
    {
        // 拷贝查询结果
        RtlCopyMemory(&get_dw, pkvi->Data, pkvi->DataLength);

        // 输出结果
        DbgPrint("[*] 查询结果: %d \n", get_dw);
        ExFreePool(pkvi);
    }

    // 第二种查询方式
    UNICODE_STRING ustrRegistry;
    UNICODE_STRING ustrKeyValueName;

    RtlInitUnicodeString(&ustrRegistry, L"\\Registry\\Machine\\Software\\LySharkKeysA");
    RtlInitUnicodeString(&ustrKeyValueName, L"is_auth");

    MyQueryRegistryKeyValue(ustrRegistry, ustrKeyValueName);

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

编译并运行这段程序，将会查询 `\\Registry\\Machine\\Software\\LySharkKeysA` 下面的 `is_auth` 字段中的值，输出效果如下图所示；



ZwEnumerateKey

枚举某个主键底部的子键值，实现对指定主键中所有的子键的枚举。

ZwEnumerateKey是Windows内核中的一个函数，用于列举指定注册表键下的子键。它通常被驱动程序使用来获取键列表，以及子键的数量和名称等信息。

以下是ZwEnumerateKey函数的一般形式：

```
NTSTATUS ZwEnumerateKey(
  _In_ HANDLE                 KeyHandle,
  _In_ ULONG                  Index,
  _In_ KEY_INFORMATION_CLASS KeyInformationClass,
  _Out_ PVOID                 KeyInformation,
  _In_ ULONG                  Length,
  _Out_ PULONG                ResultLength
);
```

参数说明：

- KeyHandle： 要列举子键的键的句柄（handle）。
- Index： 指定要列举的子键的索引。
- KeyInformationClass： 指定要获取的子键信息类型。
- KeyInformation： 存储读取的子键信息的缓冲区。

- Length： KeyInformation缓冲区的大小。
- ResultLength： 实际读取的子键信息的大小。

函数执行成功时，将返回STATUS_SUCCESS，否则返回相应的错误代码。需要注意的是，在使用ZwEnumerateKey函数之前，需要先打开要列举子键的键，获取其句柄。

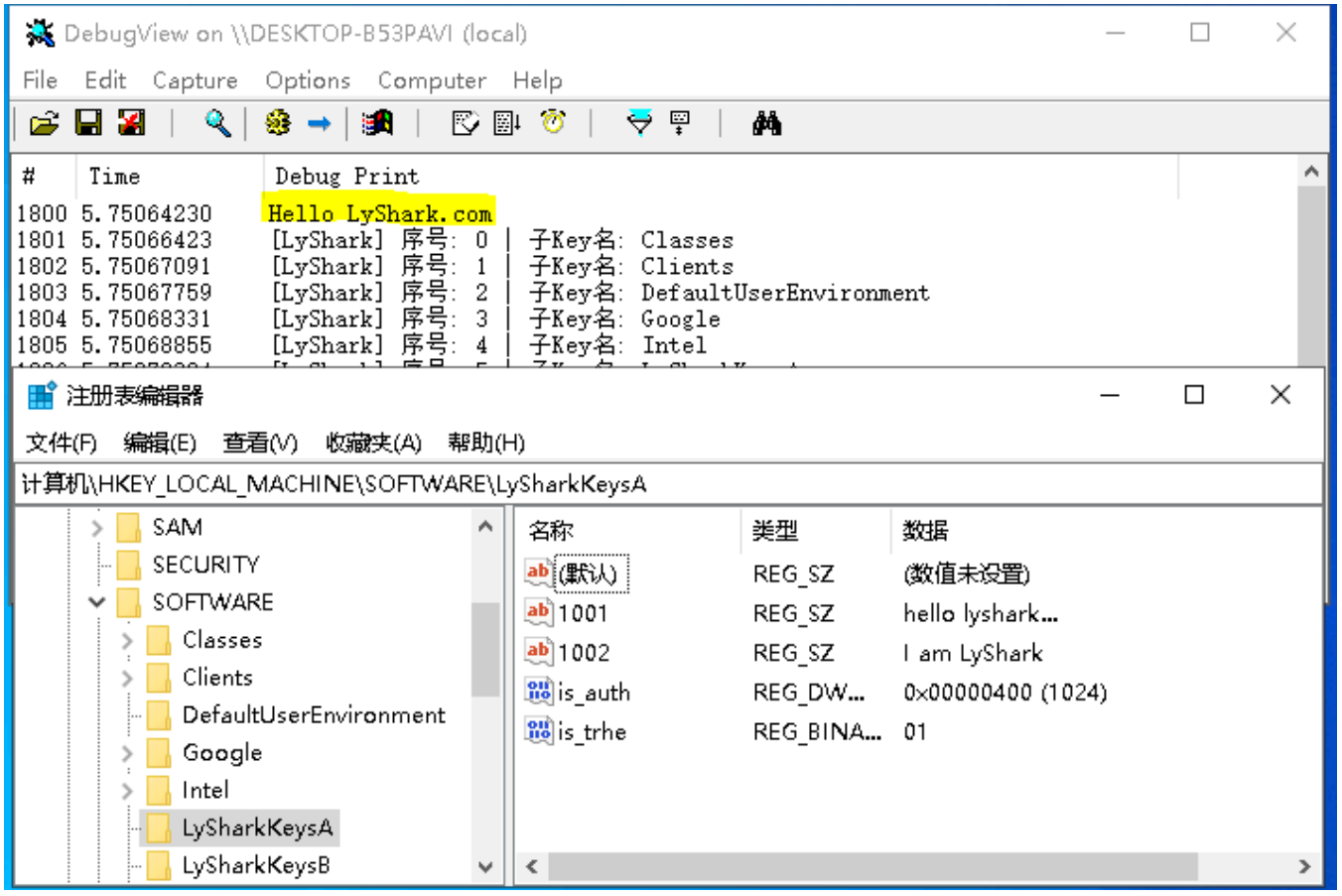
在使用ZwEnumerateKey函数时，需要注意权限和安全性问题，以避免潜在的安全问题。同时，需要仔细考虑键的名称和路径等信息，确保要列举子键的键是正确的，并且不会对系统造成不良影响。另外，需要确保KeyInformation缓冲区的大小足够，以存储读取的子键信息。

需要注意的是，ZwEnumerateKey函数只能用于列举单个键下的子键，如果需要列举多个键的子键，则需要自行实现循环操作。

```
#include <ntifs.h>
#include <windef.h>

// 枚举子键
BOOLEAN EnumRegistrySubKey(WCHAR *MY_KEY_NAME)
{
    UNICODE_STRING RegUnicodeString;
    HANDLE hRegister;
    OBJECT_ATTRIBUTES objectAttributes;
    NTSTATUS ntStatus;
    ULONG ulSize, i;
    UNICODE_STRING uniKeyName;
    PKEY_FULL_INFORMATION pfi;

    // 初始化UNICODE_STRING字符串
    RtlInitUnicodeString(&RegUnicodeString, MY_KEY_NAME);

    // 初始化objectAttributes OBJ_CASE_INSENSITIVE(大小写敏感)
    InitializeObjectAttributes(&objectAttributes, &RegUnicodeString, OBJ_CASE_INSENSITIVE,
NULL, NULL);

    // 打开注册表
    ntStatus = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (!NT_SUCCESS(ntStatus))
    {
        return FALSE;
    }

    // 第一次调用获取KEY_FULL_INFORMATION数据的长度
    ZwQueryKey(hRegister, KeyFullInformation, NULL, 0, &ulSize);
    pfi = (PKEY_FULL_INFORMATION)ExAllocatePool(PagedPool, ulSize);

    // 第二次调用获取KEY_FULL_INFORMATION数据的数据
    ZwQueryKey(hRegister, KeyFullInformation, pfi, ulSize, &ulSize);

    // 循环输出子键
    for (i = 0; i<pfi->SubKeys; i++)
    {
        PKEY_BASIC_INFORMATION pbi;
```

```
        // 第一次调用获取KEY_BASIC_INFORMATION数据的长度
        ZwEnumerateKey(hRegister, i, KeyBasicInformation, NULL, 0, &ulSize);
        pbi = (PKEY_BASIC_INFORMATION)ExAllocatePool(PagedPool, ulSize);

        // 第二次调用获取KEY_BASIC_INFORMATION数据的数据
        ZwEnumerateKey(hRegister, i, KeyBasicInformation, pbi, ulSize, &ulSize);

        uniKeyName.Length = (USHORT)pbi->NameLength;
        uniKeyName.MaximumLength = (USHORT)pbi->NameLength;
        uniKeyName.Buffer = pbi->Name;

        DbgPrint("[LyShark] 序号: %d | 子Key名: %wZ \n", i, &uniKeyName);
        ExFreePool(pbi);
    }
    ExFreePool(pfi);
    ZwClose(hRegister);
    return TRUE;
}

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

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark.com \n");

    WCHAR MY_KEY_NAME[] = L"\\Registry\\Machine\\Software";
    BOOLEAN flag = EnumRegistrySubKey(MY_KEY_NAME);

    if (flag == TRUE)
    {
        DbgPrint("[*] 枚举结束 \n");
    }

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

编译并运行如上代码片段，则会枚举`\\Registry\\Machine\\Software`底部的所有子键值，输出效果图如下所示；



ZwEnumerateValueKey

用于枚举子键下所有键值对的值，原理与上方枚举子键类似。

ZwEnumerateValueKey是Windows内核中的一个函数，用于列举指定注册表键下的所有值。它通常被驱动程序使用来获取键值列表，以及每个键值的名称、类型和数据等信息。

以下是ZwEnumerateValueKey函数的一般形式：

```
NTSTATUS ZwEnumerateValueKey(
  _In_ HANDLE                 KeyHandle,
  _In_ ULONG                  Index,
  _In_ KEY_VALUE_INFORMATION_CLASS KeyValueInformationClass,
  _Out_ PVOID                 KeyValueInformation,
  _In_ ULONG                  Length,
  _Out_ PULONG                ResultLength
);
```

参数说明：

- KeyHandle： 要列举值的键的句柄（handle）。
- Index： 指定要列举的值的索引。
- KeyValueInformationClass： 指定要获取的值信息类型。
- KeyValueInformation： 存储读取的值信息的缓冲区。
- Length： KeyValueInformation缓冲区的大小。
- ResultLength： 实际读取的值信息的大小。

函数执行成功时，将返回STATUS_SUCCESS，否则返回相应的错误代码。需要注意的是，在使用ZwEnumerateValueKey函数之前，需要先打开要列举值的键，获取其句柄。

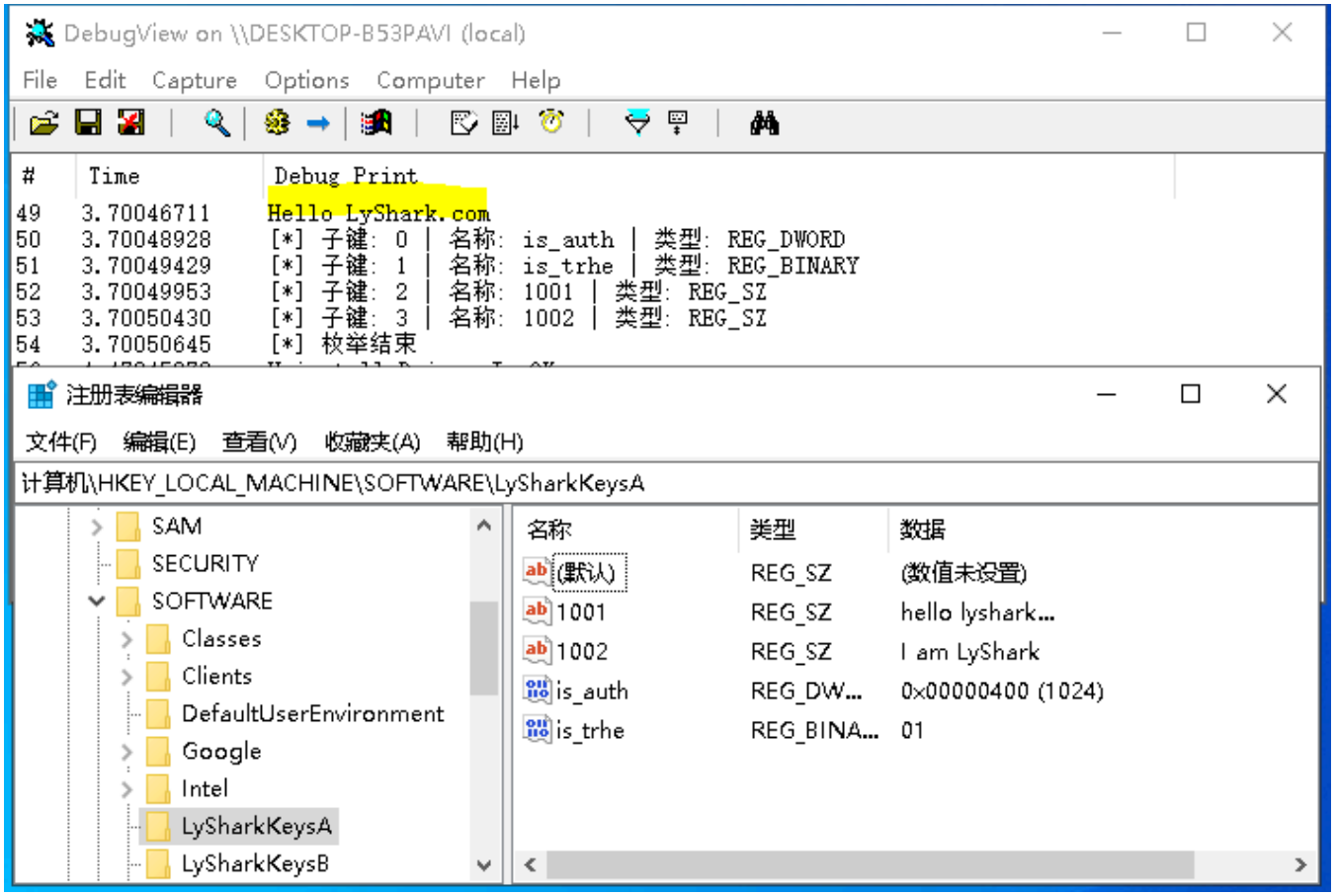
在使用ZwEnumerateValueKey函数时，需要注意权限和安全性问题，以避免潜在的安全问题。同时，需要仔细考虑键的名称和路径等信息，确保要列举值的键是正确的，并且不会对系统造成不良影响。另外，需要确保KeyValueInformation缓冲区的大小足够，以存储读取的值信息。

需要注意的是，ZwEnumerateValueKey函数只能用于列举单个键下的所有值，如果需要列举多个键的所有值，则需要自行实现循环操作。

```
#include <ntifs.h>
#include <windef.h>

// 枚举子键
BOOLEAN EnumegistrySubValue(WCHAR *MY_KEY_NAME)
{
    UNICODE_STRING RegUnicodeString;
    HANDLE hRegister;
    OBJECT_ATTRIBUTES objectAttributes;
    ULONG ulSize, i;
    UNICODE_STRING uniKeyName;
    PKEY_FULL_INFORMATION pfi;
    NTSTATUS ntStatus;

    // 初始化UNICODE_STRING字符串
    RtlInitUnicodeString(&RegUnicodeString, MY_KEY_NAME);

    // 初始化objectAttributes
    InitializeObjectAttributes(&objectAttributes, &RegUnicodeString, OBJ_CASE_INSENSITIVE,
NULL, NULL);

    // 打开注册表
    ntStatus = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (!NT_SUCCESS(ntStatus))
    {
        return FALSE;
    }

    // 查询VALUE的大小
    ZwQueryKey(hRegister, KeyFullInformation, NULL, 0, &ulSize);
    pfi = (PKEY_FULL_INFORMATION)ExAllocatePool(PagedPool, ulSize);
    ZwQueryKey(hRegister, KeyFullInformation, pfi, ulSize, &ulSize);
    for (i = 0; i<pfi->Values; i++)
    {
        PKEY_VALUE_BASIC_INFORMATION pvbi;

        // 查询单个VALUE的大小
        ZwEnumerateValueKey(hRegister, i, KeyValueBasicInformation, NULL, 0, &ulSize);
        pvbi = (PKEY_VALUE_BASIC_INFORMATION)ExAllocatePool(PagedPool, ulSize);

        // 查询单个VALUE的详情
        ZwEnumerateValueKey(hRegister, i, KeyValueBasicInformation, pvbi, ulSize, &ulSize);
        uniKeyName.Length = (USHORT)pvbi->NameLength;
```

```
        uniKeyName.MaximumLength = (USHORT)pvbi->NameLength;
        uniKeyName.Buffer = pvbi->Name;

        DbgPrint("[*] 子键: %d | 名称: %wZ | ", i, &uniKeyName);
        if (pvbi->Type == REG_SZ)
        {
            DbgPrint("类型: REG_SZ \n");
        }
        else if (pvbi->Type == REG_MULTI_SZ)
        {
            DbgPrint("类型: REG_MULTI_SZ \n");
        }
        else if (pvbi->Type == REG_DWORD)
        {
            DbgPrint("类型: REG_DWORD \n");
        }
        else if (pvbi->Type == REG_BINARY)
        {
            DbgPrint("类型: REG_BINARY \n");
        }
        ExFreePool(pvbi);
    }

    ExFreePool(pfi);
    ZwClose(hRegister);
    return TRUE;
}

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

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark.com \n");

    WCHAR MY_KEY_NAME[] = L"\\Registry\\Machine\\Software\\LySharkKeysA";
    BOOLEAN flag = EnumegistrySubValue(MY_KEY_NAME);

    if (flag == TRUE)
    {
        DbgPrint("[*] 枚举结束 \n");
    }

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

编译并运行如上这段代码，则可枚举出`\\Registry\\Machine\\Software\\LySharkKeysA`底部的所有子键以及该子键的键值，输出效果如下图所示；



ZwDeleteValueKey

用于删除指定键里面键值对的某个值。如果使用函数 `RegDeleteKey` 则删除键包括里面的所有值。

ZwDeleteValueKey是Windows内核中的一个函数，用于删除指定注册表键下的一个值。它通常被驱动程序使用来删除指定键下的一个值，以及释放该值占用的空间。

以下是ZwDeleteValueKey函数的一般形式：

```
NTSTATUS ZwDeleteValueKey(
  _In_ HANDLE          KeyHandle,
  _In_ PUNICODE_STRING ValueName
);
```

参数说明：

- KeyHandle： 要删除值的键的句柄（handle）。
- ValueName： 要删除的值的名称，为Unicode字符串指针。

函数执行成功时，将返回STATUS_SUCCESS，否则返回相应的错误代码。需要注意的是，在使用ZwDeleteValueKey函数之前，需要先打开要删除值的键，获取其句柄。

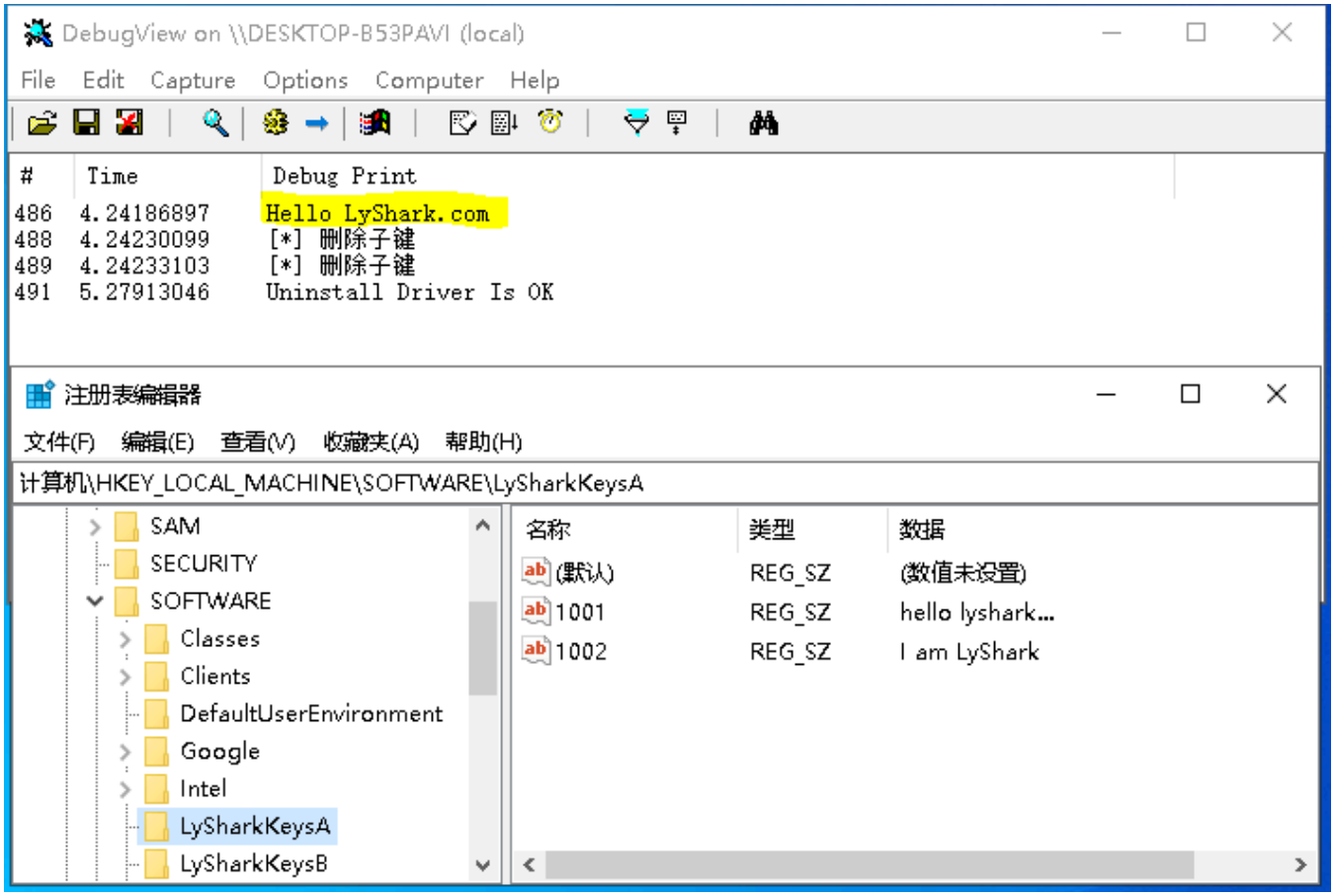
在使用ZwDeleteValueKey函数时，需要注意权限和安全性问题，以避免潜在的安全问题。同时，需要仔细考虑键的名称和路径等信息，确保要删除的值是正确的，并且不会对系统造成不良影响。

需要注意的是，ZwDeleteValueKey函数只能用于删除单个键下的一个值，如果需要删除多个键的多个值，则需要自行实现循环操作。

```
#include <ntifs.h>
#include <windef.h>

// 删除键中的值
BOOLEAN RegDeleteValueKey(LPWSTR KeyName, LPWSTR ValueName)
{
    OBJECT_ATTRIBUTES objectAttributes;
    UNICODE_STRING usKeyName, usValueName;
    NTSTATUS ntStatus;
    HANDLE hRegister;
    RtlInitUnicodeString(&usKeyName, KeyName);
    RtlInitUnicodeString(&usValueName, ValueName);

    InitializeObjectAttributes(&objectAttributes, &usKeyName, OBJ_CASE_INSENSITIVE, NULL,
NULL);

    // 打开注册表
    ntStatus = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (NT_SUCCESS(ntStatus))
    {
        ntStatus = ZwDeleteValueKey(hRegister, &usValueName);
        ZwFlushKey(hRegister);
        ZwClose(hRegister);
        return TRUE;
    }
    else
    {
        return FALSE;
    }
    return FALSE;
}

// 删除注册表键值
BOOLEAN MyDeleteRegistryKeyValue(UNICODE_STRING ustrRegistry, UNICODE_STRING
ustrKeyValueName)
{
    HANDLE hRegister = NULL;
    OBJECT_ATTRIBUTES objectAttributes = { 0 };
    NTSTATUS status = STATUS_SUCCESS;
    // 打开注册表键
    InitializeObjectAttributes(&objectAttributes, &ustrRegistry, OBJ_CASE_INSENSITIVE, NULL,
NULL);
    status = ZwOpenKey(&hRegister, KEY_ALL_ACCESS, &objectAttributes);
    if (!NT_SUCCESS(status))
    {
        return FALSE;
    }

    // 删除注册表键
    status = ZwDeleteValueKey(hRegister, &ustrKeyValueName);
    if (!NT_SUCCESS(status))
    {
        ZwClose(hRegister);
```

```
        return FALSE;
    }

    // 关闭注册表键句柄
    ZwClose(hRegister);
    return TRUE;
}

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

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark.com \n");

    // 删除值
    BOOLEAN flag = RegDeleteValueKey(L"\\Registry\\Machine\\Software\\LySharkKeysA",
L"is_auth");

    if (flag == TRUE)
    {
        DbgPrint("[*] 删除子键 \n");
    }

    UNICODE_STRING ustrRegistry;
    UNICODE_STRING ustrKeyValueName;

    RtlInitUnicodeString(&ustrRegistry, L"\\Registry\\Machine\\Software\\LySharkKeysA");
    RtlInitUnicodeString(&ustrKeyValueName, L"is_trhe");
    flag = MyDeleteRegistryKeyValue(ustrRegistry, ustrKeyValueName);
    if (flag == TRUE)
    {
        DbgPrint("[*] 删除子键 \n");
    }

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

编译并运行如上驱动程序，则会将`\\Registry\\Machine\\Software\\LySharkKeysA`里面的`is_trhe`以及`is_auth`删除，效果图如下所示；


DebugView on \\DESKTOP-B53PAVI (local)
File Edit Capture Options Computer Help
Time Debug Print
486 4.24186897 Hello LyShark.com
488 4.24230099 [*] 删除子键
489 4.24233103 [*] 删除子键
491 5.27913046 Uninstall Driver Is OK
注册表编辑器
文件(F) 编辑(E) 查看(V) 收藏夹(A) 帮助(H)
计算机\HKEY_LOCAL_MACHINE\SOFTWARE\LySharkKeysA
SAM
SECURITY
SOFTWARE
Classes
Clients
DefaultUserEnvironment
Google
Intel
LySharkKeysA
LySharkKeysB
名称 类型 数据
(默认) REG_SZ (数值未设置)
1001 REG_SZ hello lyshark...
1002 REG_SZ I am LyShark