在内核编程中字符串有两种格式 ANSI\_STRING 与 UNICODE\_STRING ，这两种格式是微软推出的安全版本的字符串结构体，也是微软推荐使用的格式，通常情况下 ANSI\_STRING 代表的类型是 char 也就是



ANSI多字节模式的字符串，而 UNICODE\_STRING 则代表的是 wchar 也就是UNCODE类型的字符，如下文章将介绍这两种字符格式在内核中是如何转换的。



# 在内核开发模式下 初始化字符串 也需要调用专用的初始化函数，如下分别初始化ANSI和UNCODE字符串，我们来看看代码是如何实现的。

#include <ntifs.h> #include <ntstrsafe.h>



VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动卸载成功 \n");

}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

// 定义内核字符串ANSI\_STRING ansi; UNICODE\_STRING unicode;

UNICODE\_STRING str;

// 定义普通字符串

char char\_string = "hello lyshark";

wchar\_t wchar\_string = (WCHAR )"hello lyshark";

// 初始化字符串的多种方式

RtlInitAnsiString(&ansi, char\_string); RtlInitUnicodeString(&unicode, wchar\_string); RtlUnicodeStringInit(&str, L"hello lyshark");

// 改变原始字符串（乱码位置，此处仅用于演示赋值方式）

char\_string[0] = (CHAR)"A"; // char类型每个占用1字节char\_string[1] = (CHAR)"B";

wchar\_string[0] = (WCHAR)"A"; // wchar类型每个占用2字节wchar\_string[2] = (WCHAR)"B";

// 输出字符串 %Z

DbgPrint("输出ANSI: %Z \n", &ansi);

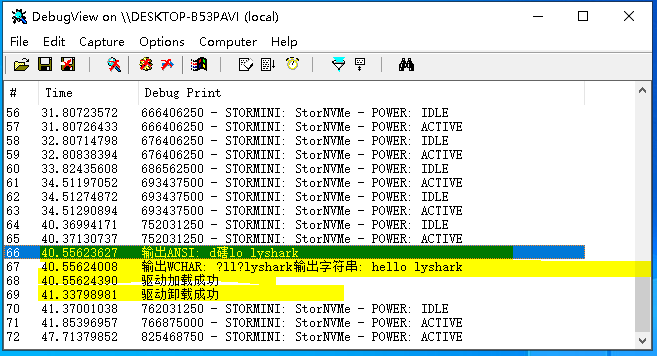
DbgPrint("输出WCHAR: %Z \n", &unicode);

DbgPrint("输出字符串: %wZ \n", &str);

DbgPrint(" 驱 动 加 载 成 功 \n"); Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 代码输出效果：



内核中还可实现 字符串与整数 之间的灵活转换，内核中提供了 RtlUnicodeStringToInteger 这个函数来实现 字符串转整数 ，与之对应的 RtlIntegerToUnicodeString 则是将 整数转为字符串 这两个内核函数也是非常常用的。

#include <ntifs.h> #include <ntstrsafe.h>

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动卸载成功 \n");

}

// Power: lyshark

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

NTSTATUS flag; ULONG number;

DbgPrint("hello lyshark \n");

UNICODE\_STRING uncode\_buffer\_source = { 0 }; UNICODE\_STRING uncode\_buffer\_target = { 0 };

// 字符串转为数字

// By：LyShark RtlInitUnicodeString(&uncode\_buffer\_source, L"100");

flag = RtlUnicodeStringToInteger(&uncode\_buffer\_source, 10, &number);

if (NT\_SUCCESS(flag))

{

DbgPrint("字符串 -> 数字: %d \n", number);

}

// 数字转为字符串

uncode\_buffer\_target.Buffer = (PWSTR)ExAllocatePool(PagedPool, 1024); uncode\_buffer\_target.MaximumLength = 1024;

flag = RtlIntegerToUnicodeString(number, 10, &uncode\_buffer\_target);

if (NT\_SUCCESS(flag))

{

DbgPrint("数字 -> 字符串: %wZ \n", &uncode\_buffer\_target);

}

// 释放堆空间

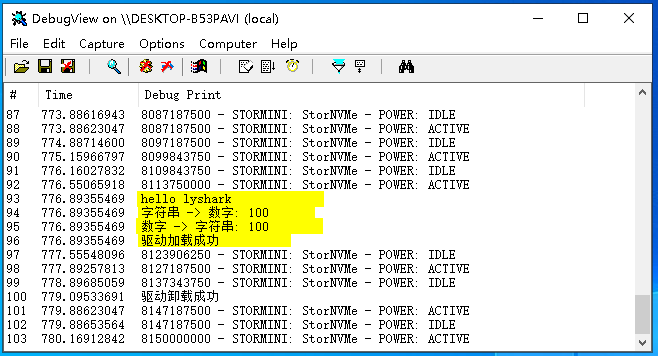
RtlFreeUnicodeString(&uncode\_buffer\_target);

DbgPrint("驱动加载成功 \n");

Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 代码输出效果：



继续看另一种转换模式，将 UNICODE\_STRING 结构转换成 ANSI\_STRING 结构，代码中调用了

RtlUnicodeStringToAnsiString 内核函数，该函数也是微软提供的。

#include <ntifs.h> #include <ntstrsafe.h>

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动卸载成功 \n");

}

// Power: lyshark

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint("hello lyshark \n");

UNICODE\_STRING uncode\_buffer\_source = { 0 }; ANSI\_STRING ansi\_buffer\_target = { 0 };

// 初始化 UNICODE 字符串

RtlInitUnicodeString(&uncode\_buffer\_source, L"hello lyshark");

// 转换函数

NTSTATUS flag = RtlUnicodeStringToAnsiString(&ansi\_buffer\_target, &uncode\_buffer\_source, TRUE);

if (NT\_SUCCESS(flag))

{

DbgPrint("ANSI: %Z \n", &ansi\_buffer\_target);

}

// 销毁ANSI字符串

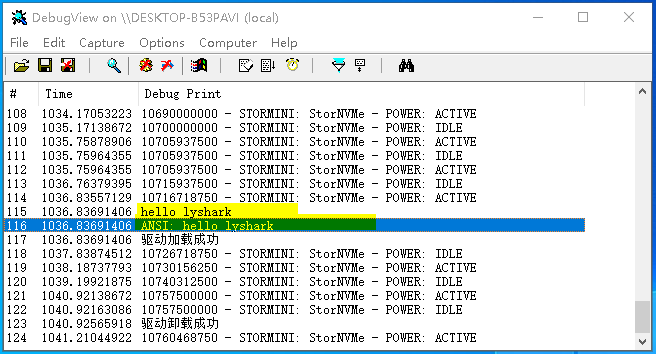
RtlFreeAnsiString(&ansi\_buffer\_target);

DbgPrint(" 驱 动 加 载 成 功 \n"); Driver->DriverUnload = UnDriver;

return STATUS\_SUCCESS;

}

# 代码输出效果：



如果将上述过程反过来，将 ANSI\_STRING 转换为 UNICODE\_STRING 结构，则需要调用

RtlAnsiStringToUnicodeString 这个内核专用函数实现。

#include <ntifs.h> #include <ntstrsafe.h>

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动卸载成功 \n");

}

// Power: lyshark

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint("hello lyshark \n");

UNICODE\_STRING uncode\_buffer\_source = { 0 }; ANSI\_STRING ansi\_buffer\_target = { 0 };

// 初始化字符串

RtlInitString(&ansi\_buffer\_target, "hello lyshark");

// 转换函数

NTSTATUS flag = RtlAnsiStringToUnicodeString(&uncode\_buffer\_source, &ansi\_buffer\_target, TRUE);

if (NT\_SUCCESS(flag))

{

DbgPrint("UNICODE: %wZ \n", &uncode\_buffer\_source);

}

// 销毁UNICODE字符串

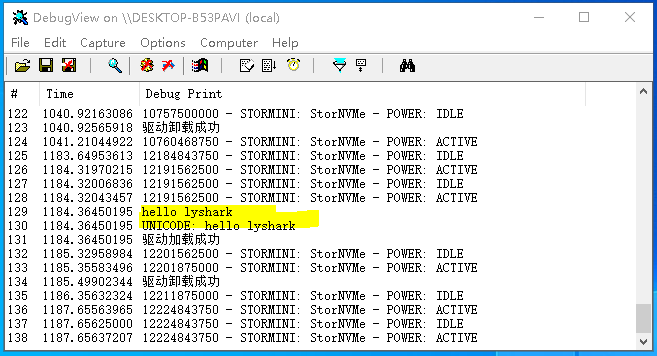
RtlFreeUnicodeString(&uncode\_buffer\_source);

DbgPrint(" 驱 动 加 载 成 功 \n"); Driver->DriverUnload = UnDriver;

return STATUS\_SUCCESS;

}

# 代码输出效果：



如上代码是内核通用结构体之间的转换类型，又是还需要将各类结构体转为普通的字符类型，例如下方 的两个案例：

例如将 UNICODE\_STRING 转为 类型。



CHAR

#define \_CRT\_SECURE\_NO\_WARNINGS #include <ntifs.h>

#include <windef.h>

#include <ntstrsafe.h>

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动卸载成功 \n");

}



// powerBY: LyShark

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint("hello lyshark \n");

UNICODE\_STRING uncode\_buffer\_source = { 0 }; ANSI\_STRING ansi\_buffer\_target = { 0 };

char szBuf[1024] = { 0 };

// 初始化 UNICODE 字符串

RtlInitUnicodeString(&uncode\_buffer\_source, L"hello lyshark");

// 转换函数

NTSTATUS flag = RtlUnicodeStringToAnsiString(&ansi\_buffer\_target, &uncode\_buffer\_source, TRUE);

if (NT\_SUCCESS(flag))

{

strcpy(szBuf, ansi\_buffer\_target.Buffer); DbgPrint("输出char 字符串: %s \n", szBuf);

}

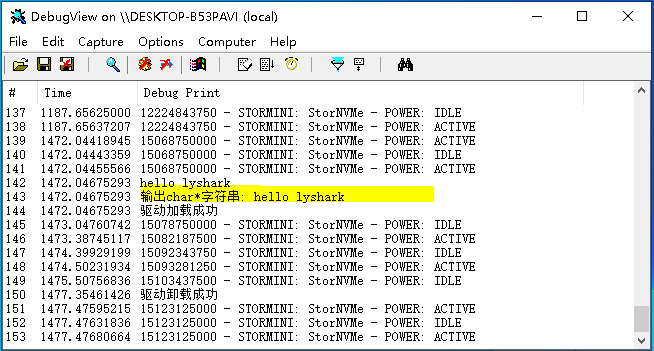
// 销毁ANSI字符串

RtlFreeAnsiString(&ansi\_buffer\_target);

DbgPrint(" 驱 动 加 载 成 功 \n"); Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 代码输出效果：



如果反过来，将构体。



CHAR

类型转为 UNICODE\_STRING 结构呢，可以进行中转最终转为 UNICODE\_STRING 结

#define \_CRT\_SECURE\_NO\_WARNINGS



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

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动卸载成功 \n");

}

// powerBY: LyShark

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint("hello lyshark \n");

UNICODE\_STRING uncode\_buffer\_source = { 0 }; ANSI\_STRING ansi\_buffer\_target = { 0 };

// 设置CHAR

char szBuf[1024] = { 0 }; strcpy(szBuf, "hello lyshark");

// 初始化ANSI字符串

RtlInitString(&ansi\_buffer\_target, szBuf);

// 转换函数

NTSTATUS flag = RtlAnsiStringToUnicodeString(&uncode\_buffer\_source, &ansi\_buffer\_target, TRUE);

if (NT\_SUCCESS(flag))

{

DbgPrint("UNICODE: %wZ \n", &uncode\_buffer\_source);

}

// 销毁UNICODE字符串

RtlFreeUnicodeString(&uncode\_buffer\_source);

DbgPrint(" 驱 动 加 载 成 功 \n"); Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 代码输出效果：

