# 在上一篇文章 《驱动开发：内核字符串转换方法》 中简单介绍了内核是如何使用字符串以及字符串之间的转换方法，本章将继续探索字符串的拷贝与比较，与应用层不同内核字符串拷贝与比较也需要使用内核 专用的API函数，字符串的拷贝往往伴随有内核内存分配，我们将首先简单介绍内核如何分配堆空间，然 后再以此为契机简介字符串的拷贝与比较。

首先内核中的堆栈分配可以使用 ExAllocatePool() 这个内核函数实现，此外还可以使用

ExAllocatePoolWithTag() 函数，两者的区别是，第一个函数可以直接分配内存，第二个函数在分配时需要指定一个标签，此外内核属性常用的有两种 NonPagedPool 用于分配非分页内存，而 PagePool 则用于分配分页内存，在开发中推荐使用非分页内存，因为分页内存数量有限。

内存分配使用 ExAllocatePool 函数，内存拷贝可使用 RtlCopyMemory 函数，需要注意该函数其实是对 Memcpy 函数的包装。

#include <ntifs.h>



VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动已卸载 \n");

}

// PowerBy: LyShark

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

{

UNICODE\_STRING uncode\_buffer = { 0 };

DbgPrint("hello lyshark \n");

wchar\_t wchar\_string = L"hello lyshark";

// 设置最大长度

uncode\_buffer.MaximumLength = 1024;

// 分配内存空间

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

// 设置字符长度 因为是宽字符，所以是字符长度的 2 倍

uncode\_buffer.Length = wcslen(wchar\_string) 2;

// 保证缓冲区足够大，否则程序终止

ASSERT(uncode\_buffer.MaximumLength >= uncode\_buffer.Length);

// 将 wchar\_string 中 的 字 符 串 拷 贝 到 uncode\_buffer.Buffer RtlCopyMemory(uncode\_buffer.Buffer, wchar\_string, uncode\_buffer.Length);

// 设置字符串长度 并输出

uncode\_buffer.Length = wcslen(wchar\_string) 2;

DbgPrint("输出字符串: %wZ \n", uncode\_buffer);

// 释放堆空间

ExFreePool(uncode\_buffer.Buffer); uncode\_buffer.Buffer = NULL;

uncode\_buffer.Length = uncode\_buffer.MaximumLength = 0;

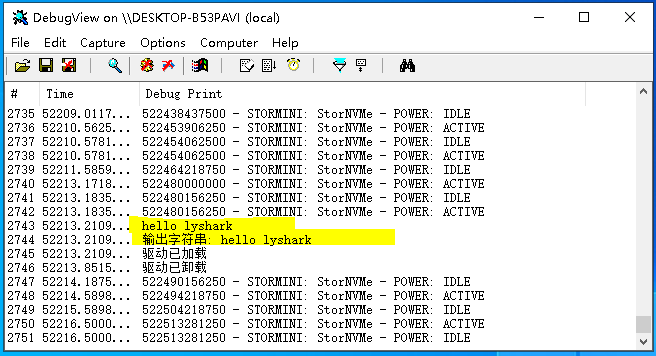
DbgPrint("驱动已加载 \n");

Driver->DriverUnload = UnDriver;

return STATUS\_SUCCESS;

}

# 代码输出效果：



实现 空间分配 ，字符串结构 UNICODE\_STRING 可以定义数组，空间的分配也可以循环进行，例如我们分配十个字符串结构，并输出结构内的参数。



#include <ntifs.h>

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动已卸载 \n");

}

// PowerBy: LyShark

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

{

UNICODE\_STRING uncode\_buffer[10] = { 0 }; wchar\_t wchar\_string = L"hello lyshark";

DbgPrint("hello lyshark \n");

int size = sizeof(uncode\_buffer) / sizeof(uncode\_buffer[0]);

DbgPrint("数组长度: %d \n", size);

for (int x = 0; x < size; x++)

{

// 分配空间

uncode\_buffer[x].Buffer = (PWSTR)ExAllocatePool(PagedPool, 1024);

// 设置长度

uncode\_buffer[x].MaximumLength = 1024;

uncode\_buffer[x].Length = wcslen(wchar\_string) sizeof(WCHAR);



ASSERT(uncode\_buffer[x].MaximumLength >= uncode\_buffer[x].Length);

// 拷贝字符串并输出

RtlCopyMemory(uncode\_buffer[x].Buffer, wchar\_string, uncode\_buffer[x].Length);

uncode\_buffer[x].Length = wcslen(wchar\_string) sizeof(WCHAR);

DbgPrint("循环: %d 输出字符串: %wZ \n", x, uncode\_buffer[x]);

// 释放内存

ExFreePool(uncode\_buffer[x].Buffer); uncode\_buffer[x].Buffer = NULL;

uncode\_buffer[x].Length = uncode\_buffer[x].MaximumLength = 0;

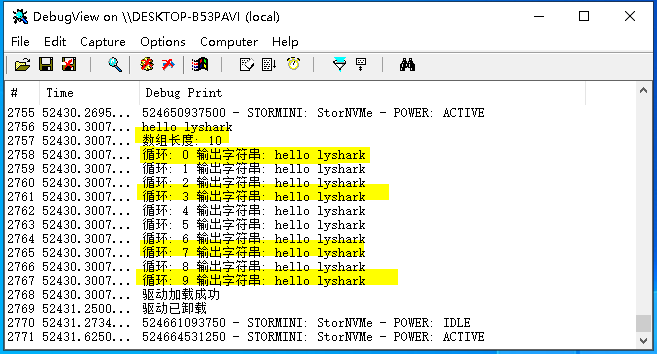
}

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

return STATUS\_SUCCESS;

}

# 代码输出效果：



实现 字符串拷贝 ，此处可以直接使用 RtlCopyMemory 函数直接对内存操作，也可以调用内核提供的

RtlCopyUnicodeString 函数来实现，具体代码如下。

#include <ntifs.h>

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动已卸载 \n");

}

// PowerBy: LyShark

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

{

DbgPrint("hello lyshark \n");

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

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

// 该函数可用于初始化字符串

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

// 初始化target字符串,分配空间

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

// 将source中的内容拷贝到target中

RtlCopyUnicodeString(&uncode\_buffer\_target, &uncode\_buffer\_source);

// 输出结果

DbgPrint("source = %wZ \n", &uncode\_buffer\_source); DbgPrint("target = %wZ \n", &uncode\_buffer\_target);

// 释放空间 source 无需销毁

// 如果强制释放掉source则会导致系统蓝屏,因为source是在栈上的

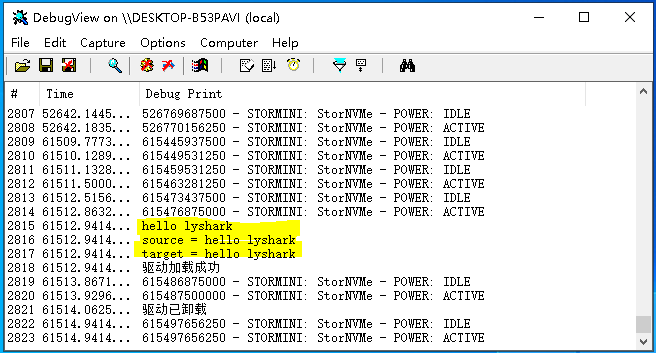
RtlFreeUnicodeString(&uncode\_buffer\_target);

DbgPrint("驱动加载成功 \n");

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

}

# 代码输出效果：



实现 字符串比较 ，如果需要比较两个 UNICODE\_STRING 字符串结构体是否相等，那么可以使用

RtlEqualUnicodeString 这个内核函数实现，该函数第三个参数是返回值类型，如果是TRUE则默认返回真，否则返回假，具体代码如下。

#include <ntifs.h>

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动已卸载 \n");

}

// PowerBy: LyShark

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

{

DbgPrint("hello lyshark \n");

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

// 该函数可用于初始化字符串RtlInitUnicodeString(&uncode\_buffer\_source, L"hello lyshark"); RtlInitUnicodeString(&uncode\_buffer\_target, L"hello lyshark");

// 比较字符串是否相等

if (RtlEqualUnicodeString(&uncode\_buffer\_source, &uncode\_buffer\_target, TRUE))

{

DbgPrint("字符串相等 \n");

}

else

{

DbgPrint("字符串不相等 \n");

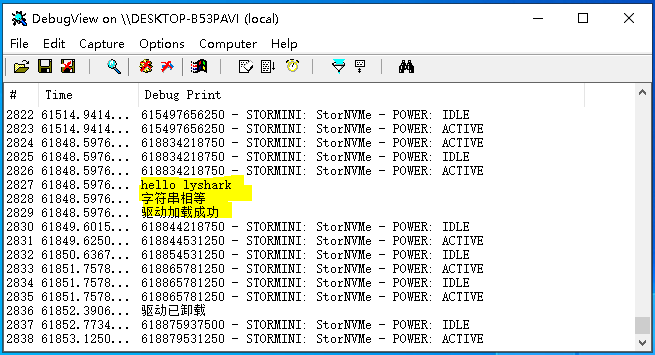
}

DbgPrint("驱动加载成功 \n");

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

}

# 代码输出效果：



有时在字符串比较时需要统一字符串格式，例如全部变大写以后在做比较等，此时可以使用

RtlUpcaseUnicodeString 函数将小写字符串为大写，然后在做比较，代码如下。

#include <ntifs.h>

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动已卸载 \n");

}

// PowerBy: LyShark

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

{

DbgPrint("hello lyshark \n");

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

// 该函数可用于初始化字符串

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

// 字符串小写变大写

RtlUpcaseUnicodeString(&uncode\_buffer\_target, &uncode\_buffer\_source, TRUE);

DbgPrint("小写输出: %wZ \n", &uncode\_buffer\_source); DbgPrint("变大写输出: %wZ \n", &uncode\_buffer\_target);

// 销毁字符串

RtlFreeUnicodeString(&uncode\_buffer\_target);

DbgPrint("驱动加载成功 \n");

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

}

# 代码输出效果：

