

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 unicode_buffer = { 0 };

    DbgPrint("hello lyshark \n");

    wchar_t * wchar_string = L"hello lyshark";

    // 设置最大长度
    unicode_buffer.MaximumLength = 1024;

    // 分配内存空间
    unicode_buffer.Buffer = (PWSTR)ExAllocatePool(PagedPool, 1024);

    // 设置字符串长度 因为是宽字符，所以是字符长度的 2 倍
    unicode_buffer.Length = wcslen(wchar_string) * 2;

    // 保证缓冲区足够大，否则程序终止
    ASSERT(unicode_buffer.MaximumLength >= unicode_buffer.Length);

    // 将 wchar_string 中的字符串拷贝到 unicode_buffer.Buffer
    RtlCopyMemory(unicode_buffer.Buffer, wchar_string, unicode_buffer.Length);

    // 设置字符串长度 并输出
    unicode_buffer.Length = wcslen(wchar_string) * 2;
    DbgPrint("输出字符串: %wZ \n", unicode_buffer);

    // 释放堆空间
    ExFreePool(unicode_buffer.Buffer);
    unicode_buffer.Buffer = NULL;
    unicode_buffer.Length = unicode_buffer.MaximumLength = 0;
```

```

    DbgPrint("驱动已加载 \n");
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

代码输出效果:

#	Time	Debug Print
2735	52209.0117...	522438437500 - STORMINI: StorNVMe - POWER: IDLE
2736	52210.5625...	522453906250 - STORMINI: StorNVMe - POWER: ACTIVE
2737	52210.5781...	522454062500 - STORMINI: StorNVMe - POWER: IDLE
2738	52210.5781...	522454062500 - STORMINI: StorNVMe - POWER: ACTIVE
2739	52211.5859...	522464218750 - STORMINI: StorNVMe - POWER: IDLE
2740	52213.1718...	522480000000 - STORMINI: StorNVMe - POWER: ACTIVE
2741	52213.1835...	522480156250 - STORMINI: StorNVMe - POWER: IDLE
2742	52213.1835...	522480156250 - STORMINI: StorNVMe - POWER: ACTIVE
2743	52213.2109...	hello lyshark
2744	52213.2109...	输出字符串: hello lyshark
2745	52213.2109...	驱动已加载
2746	52213.8515...	驱动已卸载
2747	52214.1875...	522490156250 - STORMINI: StorNVMe - POWER: IDLE
2748	52214.5898...	522494218750 - STORMINI: StorNVMe - POWER: ACTIVE
2749	52215.5898...	522504218750 - STORMINI: StorNVMe - POWER: IDLE
2750	52216.5000...	522513281250 - STORMINI: StorNVMe - POWER: ACTIVE
2751	52216.5000...	522513281250 - STORMINI: StorNVMe - POWER: IDLE

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 unicode_buffer[10] = { 0 };
    wchar_t * wchar_string = L"hello lyshark";

    DbgPrint("hello lyshark \n");

    int size = sizeof(unicode_buffer) / sizeof(unicode_buffer[0]);
    DbgPrint("数组长度: %d \n", size);

    for (int x = 0; x < size; x++)
    {
        // 分配空间
        unicode_buffer[x].Buffer = (PWSTR)ExAllocatePool(PagedPool, 1024);

        // 设置长度
        unicode_buffer[x].MaximumLength = 1024;
        unicode_buffer[x].Length = wcslen(wchar_string) * sizeof(WCHAR);
    }
}

```

```

        ASSERT(unicode_buffer[x].MaximumLength >= unicode_buffer[x].Length);

        // 拷贝字符串并输出
        RtlCopyMemory(unicode_buffer[x].Buffer, wchar_string,
unicode_buffer[x].Length);
        unicode_buffer[x].Length = wcslen(wchar_string) * sizeof(WCHAR);
        DbgPrint("循环: %d 输出字符串: %wZ \n", x, unicode_buffer[x]);

        // 释放内存
        ExFreePool(unicode_buffer[x].Buffer);
        unicode_buffer[x].Buffer = NULL;
        unicode_buffer[x].Length = unicode_buffer[x].MaximumLength = 0;
    }

    DbgPrint("驱动加载成功 \n");
    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

代码输出效果:

#	Time	Debug Print
2755	52430.2695...	524650937500 - STORMINI: StorNVMe - POWER: ACTIVE
2756	52430.3007...	hello lyshark
2757	52430.3007...	数组长度: 10
2758	52430.3007...	循环: 0 输出字符串: hello lyshark
2759	52430.3007...	循环: 1 输出字符串: hello lyshark
2760	52430.3007...	循环: 2 输出字符串: hello lyshark
2761	52430.3007...	循环: 3 输出字符串: hello lyshark
2762	52430.3007...	循环: 4 输出字符串: hello lyshark
2763	52430.3007...	循环: 5 输出字符串: hello lyshark
2764	52430.3007...	循环: 6 输出字符串: hello lyshark
2765	52430.3007...	循环: 7 输出字符串: hello lyshark
2766	52430.3007...	循环: 8 输出字符串: hello lyshark
2767	52430.3007...	循环: 9 输出字符串: hello lyshark
2768	52430.3007...	驱动加载成功
2769	52431.2500...	驱动已卸载
2770	52431.2734...	524661093750 - STORMINI: StorNVMe - POWER: IDLE
2771	52431.6250...	524664531250 - STORMINI: StorNVMe - POWER: ACTIVE

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 unicode_buffer_source = { 0 };
}

```

```

UNICODE_STRING unicode_buffer_target = { 0 };

// 该函数可用于初始化字符串
RtlInitUnicodeString(&unicode_buffer_source, L"hello lyshark");

// 初始化target字符串,分配空间
unicode_buffer_target.Buffer = (PWSTR)ExAllocatePool(PagedPool, 1024);
unicode_buffer_target.MaximumLength = 1024;

// 将source中的内容拷贝到target中
RtlCopyUnicodeString(&unicode_buffer_target, &unicode_buffer_source);

// 输出结果
DbgPrint("source = %wZ \n", &unicode_buffer_source);
DbgPrint("target = %wZ \n", &unicode_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(&unicode_buffer_target);

DbgPrint("驱动加载成功 \n");

Driver->DriverUnload = UnDriver;

return STATUS_SUCCESS;
}

```

代码输出效果:

#	Time	Debug Print
2807	52642.1445...	526769687500 - STORMINI: StorNVMe - POWER: IDLE
2808	52642.1835...	526770156250 - STORMINI: StorNVMe - POWER: ACTIVE
2809	61509.7773...	615445937500 - STORMINI: StorNVMe - POWER: IDLE
2810	61510.1289...	615449531250 - STORMINI: StorNVMe - POWER: ACTIVE
2811	61511.1328...	615459531250 - STORMINI: StorNVMe - POWER: IDLE
2812	61511.5000...	615463281250 - STORMINI: StorNVMe - POWER: ACTIVE
2813	61512.5156...	615473437500 - STORMINI: StorNVMe - POWER: IDLE
2814	61512.8632...	615476875000 - STORMINI: StorNVMe - POWER: ACTIVE
2815	61512.9414...	hello lyshark
2816	61512.9414...	source = hello lyshark
2817	61512.9414...	target = hello lyshark
2818	61512.9414...	驱动加载成功
2819	61513.8671...	615486875000 - STORMINI: StorNVMe - POWER: IDLE
2820	61513.9296...	615487500000 - STORMINI: StorNVMe - POWER: ACTIVE
2821	61514.0625...	驱动已卸载
2822	61514.9414...	615497656250 - STORMINI: StorNVMe - POWER: IDLE
2823	61514.9414...	615497656250 - STORMINI: StorNVMe - POWER: ACTIVE

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:

DebugView on \\DESKTOP-B53PAVI (local)

#	Time	Debug Print
2822	61514.9414...	615497656250 - STORMINI: StorNVMe - POWER: IDLE
2823	61514.9414...	615497656250 - STORMINI: StorNVMe - POWER: ACTIVE
2824	61848.5976...	618834218750 - STORMINI: StorNVMe - POWER: ACTIVE
2825	61848.5976...	618834218750 - STORMINI: StorNVMe - POWER: IDLE
2826	61848.5976...	618834218750 - STORMINI: StorNVMe - POWER: ACTIVE
2827	61848.5976...	hello lyshark
2828	61848.5976...	字符串相等
2829	61848.5976...	驱动加载成功
2830	61849.6015...	618844218750 - STORMINI: StorNVMe - POWER: IDLE
2831	61849.6250...	618844531250 - STORMINI: StorNVMe - POWER: ACTIVE
2832	61850.6367...	618854531250 - STORMINI: StorNVMe - POWER: IDLE
2833	61851.7578...	618865781250 - STORMINI: StorNVMe - POWER: ACTIVE
2834	61851.7578...	618865781250 - STORMINI: StorNVMe - POWER: IDLE
2835	61851.7578...	618865781250 - STORMINI: StorNVMe - POWER: ACTIVE
2836	61852.3906...	驱动已卸载
2837	61852.7734...	618875937500 - STORMINI: StorNVMe - POWER: IDLE
2838	61853.1250...	618879531250 - STORMINI: StorNVMe - POWER: ACTIVE

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 `RtlUppcaseUnicodeString` 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);

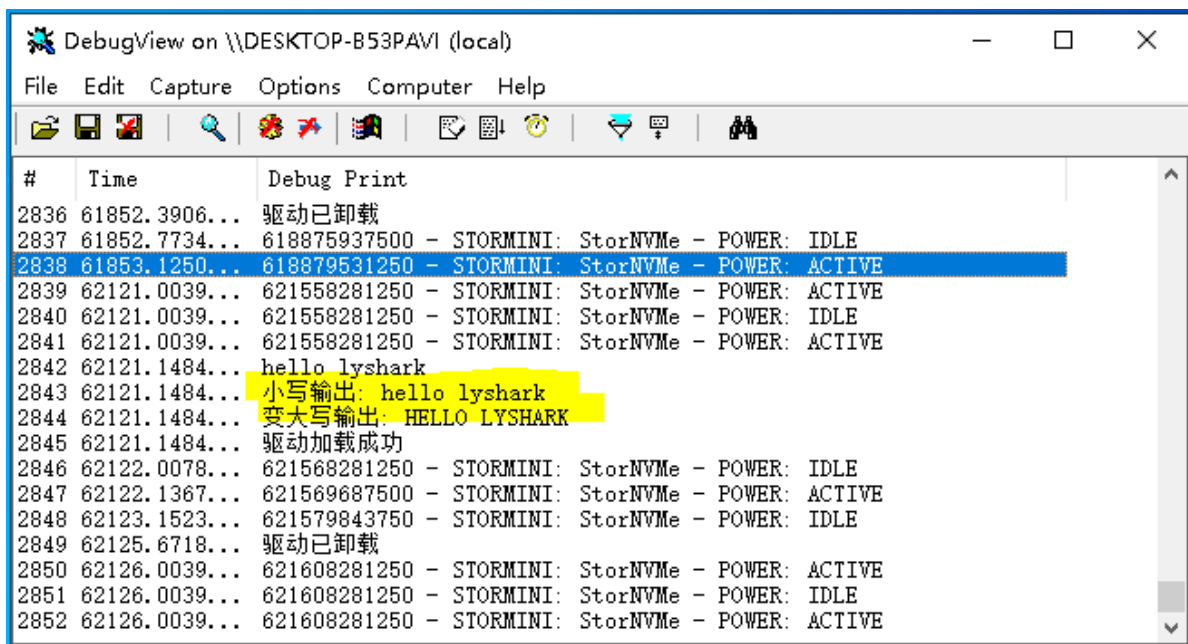
    // 销毁字符串
    RtlFreeUnicodeString(&uncode_buffer_target);

    DbgPrint("驱动加载成功 \n");

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}

```

代码输出效果:



#	Time	Debug Print
2836	61852.3906...	驱动已卸载
2837	61852.7734...	618875937500 - STORMINI: StorNVMe - POWER: IDLE
2838	61853.1250...	618879531250 - STORMINI: StorNVMe - POWER: ACTIVE
2839	62121.0039...	621558281250 - STORMINI: StorNVMe - POWER: ACTIVE
2840	62121.0039...	621558281250 - STORMINI: StorNVMe - POWER: IDLE
2841	62121.0039...	621558281250 - STORMINI: StorNVMe - POWER: ACTIVE
2842	62121.1484...	hello lyshark
2843	62121.1484...	小写输出: hello lyshark
2844	62121.1484...	变大写输出: HELLO LYSHARK
2845	62121.1484...	驱动加载成功
2846	62122.0078...	621568281250 - STORMINI: StorNVMe - POWER: IDLE
2847	62122.1367...	621569687500 - STORMINI: StorNVMe - POWER: ACTIVE
2848	62123.1523...	621579843750 - STORMINI: StorNVMe - POWER: IDLE
2849	62125.6718...	驱动已卸载
2850	62126.0039...	621608281250 - STORMINI: StorNVMe - POWER: ACTIVE
2851	62126.0039...	621608281250 - STORMINI: StorNVMe - POWER: IDLE
2852	62126.0039...	621608281250 - STORMINI: StorNVMe - POWER: ACTIVE