在上一篇文章《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。

`ExAllocatePool` 用于在内核空间分配内存。它的作用是向系统申请一块指定大小的内存，并返回这块内存的起始地址，供内核使用。需要注意的是，使用 `ExAllocatePool` 分配的内存是在内核空间中，因此不能被用户空间的代码直接访问。

`RtlCopyMemory` 也是Windows内核开发中的一个函数，用于在内存中 `拷贝` 数据。它的作用是将指定长度的数据从源地址拷贝到目标地址，可以用于在内核空间中拷贝数据。需要注意的是，`RtlCopyMemory` 实际上是对 `memcpy` 函数的封装，但是它提供了更加严格的参数检查和更好的错误处理机制，因此在内核开发中建议使用 `RtlCopyMemory` 而不是直接使用 `memcpy`。

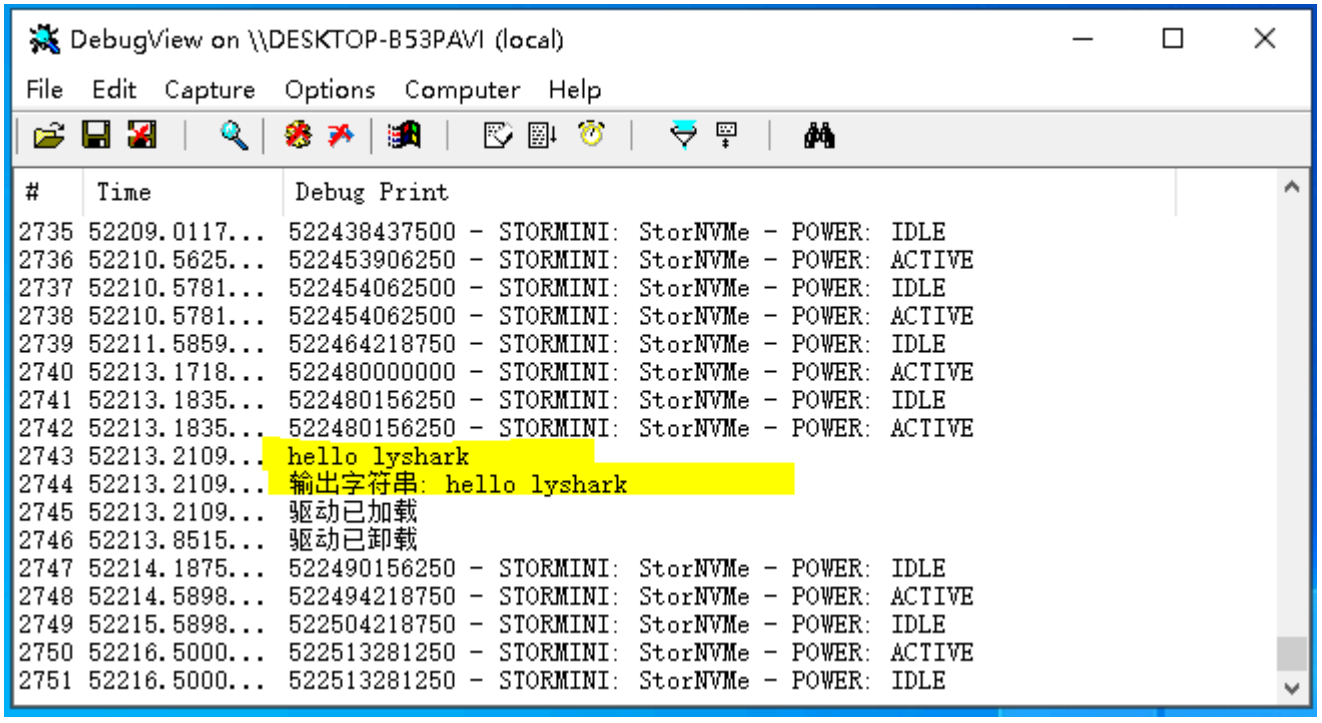
在使用这两个函数时需要注意以下几点：

- ExAllocatePool分配的内存必须在使用完后及时释放，否则会导致内存泄漏。可以使用 `ExFreePool` 函数来释放内存。
- ExAllocatePool分配的内存是非连续的，因此不能使用指针算术运算来访问内存块中的某个元素。如果需要在内存块中访问某个元素，可以使用数组下标的方式来访问。
- RtlCopyMemory函数需要确保源地址和目标地址所指向的内存块不会重叠，否则会导致数据的不确定性。可以使用 `RtlMoveMemory` 函数来处理源地址和目标地址重叠的情况。

```
#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;
}
```

代码输出效果如下图所示：

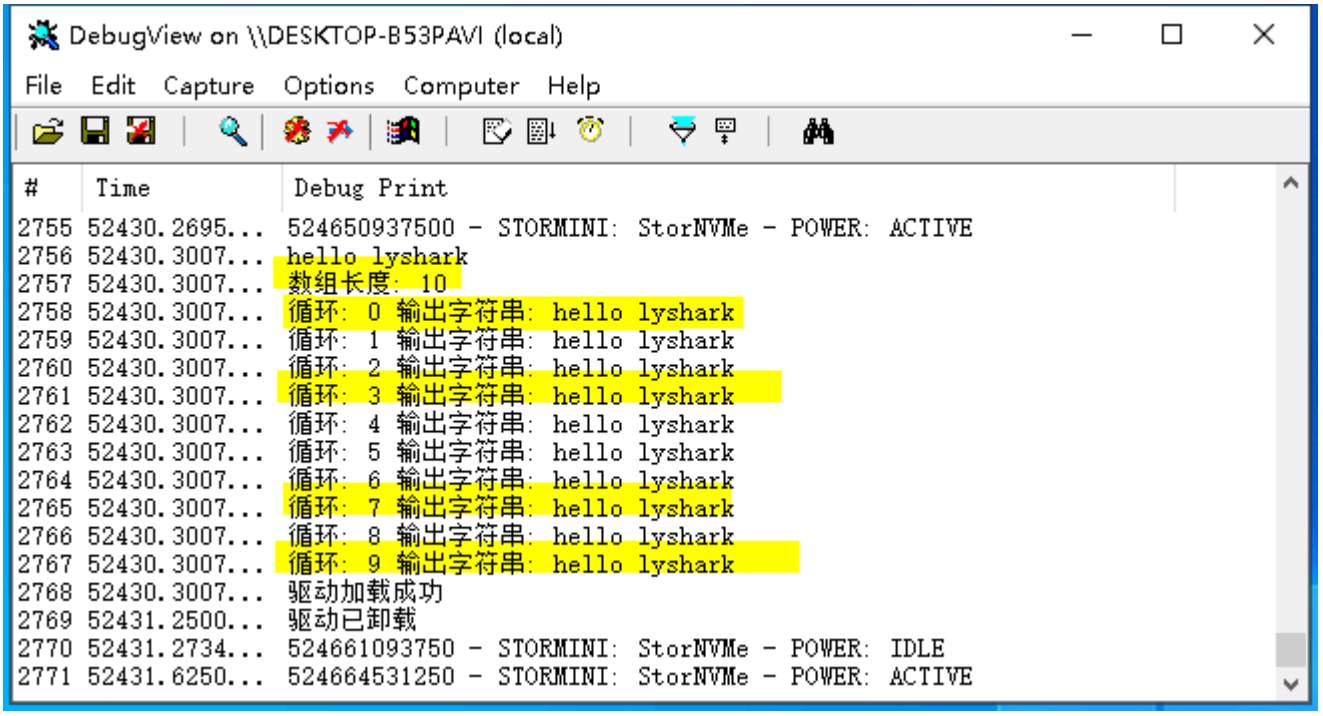


实现 空间分配，字符串结构 UNICODE_STRING 可以定义数组，空间的分配也可以循环进行，例如我们分配十个字符串结构，并输出结构内的参数。

```
#include <ntifs.h>

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("驱动已卸载 \n");
```

```
}

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;
}
```

代码输出效果如下图所示:

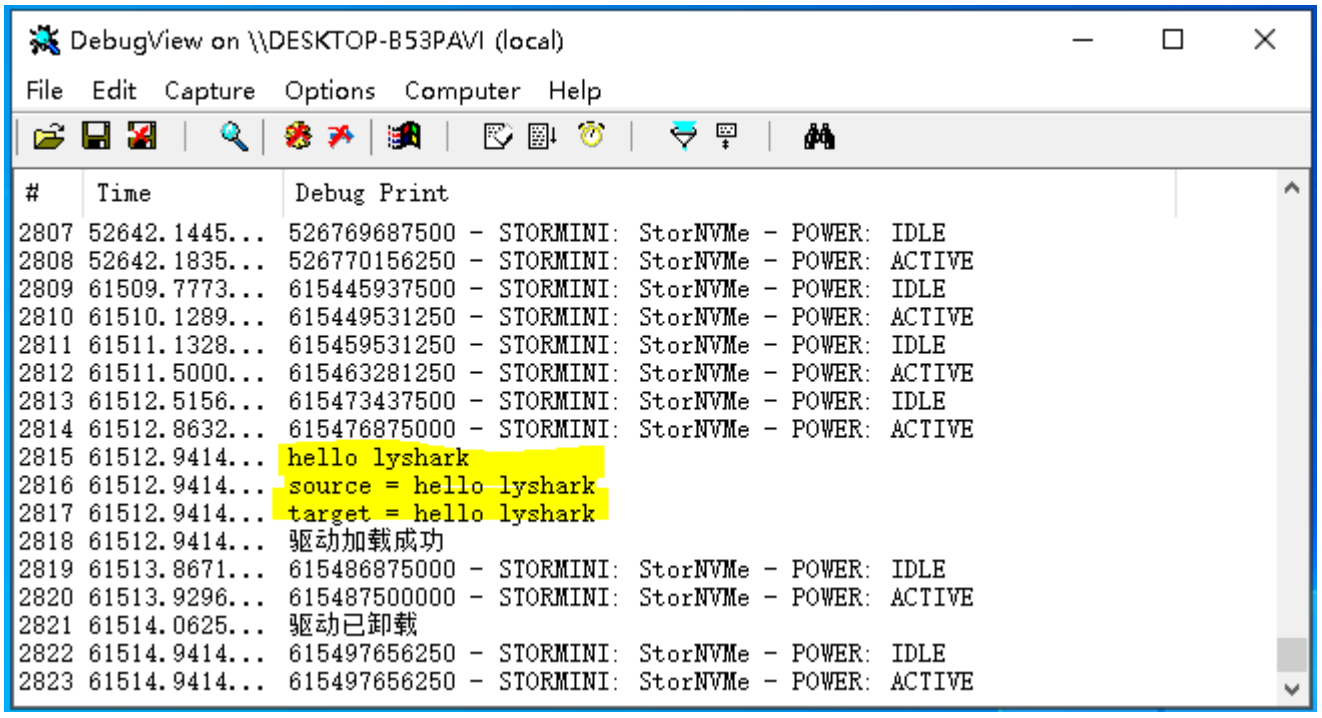


实现 `字符串拷贝`，此处可以直接使用 `RtlCopyMemory` 函数直接对内存操作，也可以调用内核提供的 `RtlCopyUnicodeString` 函数来实现，具体代码如下。

```
#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;
}
```

代码输出效果如下图所示:



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。

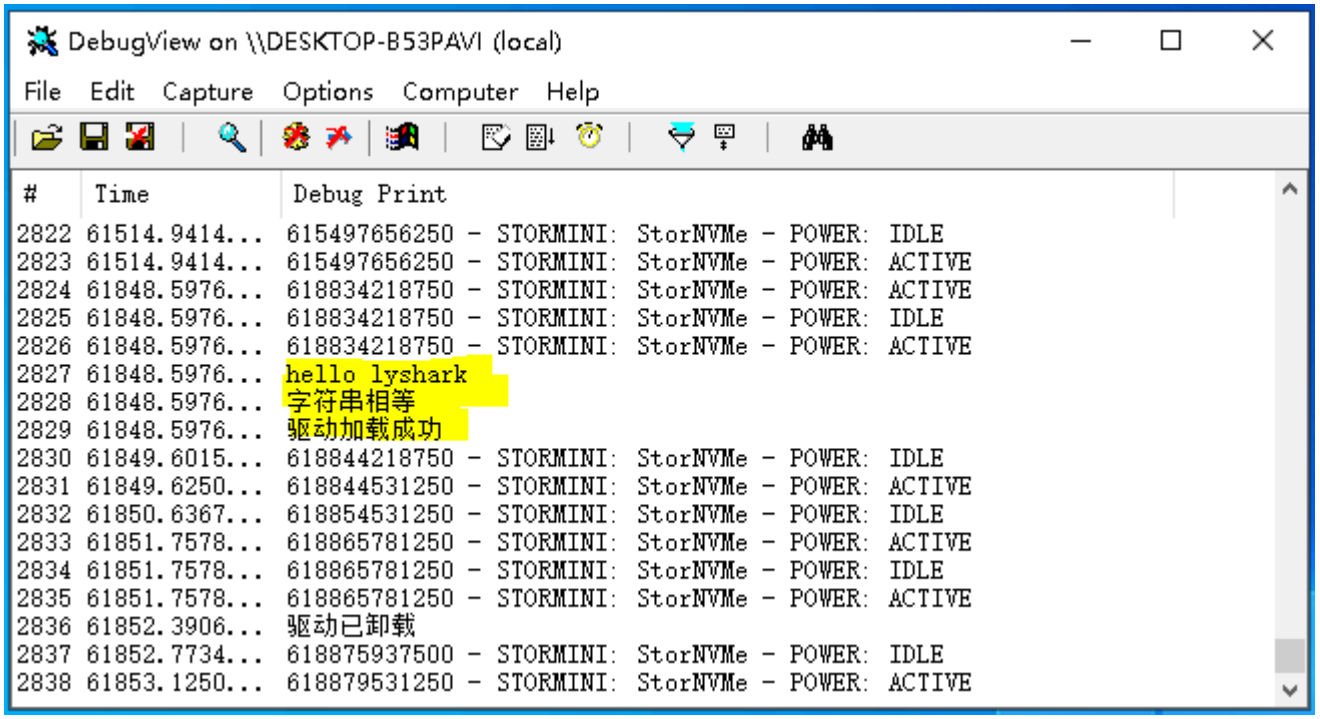
`RtlEqualUnicodeString`用于比较两个`UNICODE_STRING`字符串结构体是否相等。该函数的第一个参数是指向要比较的第一个字符串结构体的指针，第二个参数是指向要比较的第二个字符串结构体的指针，第三个参数是指定比较的方式，如果该参数为TRUE，则函数会在相等的情况下返回TRUE，否则会在不相等的情况下返回FALSE。

下面是一个使用`RtlEqualUnicodeString`函数比较两个字符串结构体是否相等的示例代码:

```
#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;
}
```

代码输出效果如下图所示：



有时在字符串比较时需要统一字符串格式，例如将所有字符全部转换为大写之后再做比较，此时可以使用`RtlUpcaseUnicodeString`函数将小写字符串为大写。

`RtlUpcaseUnicodeString`用于将`UNICODE_STRING`字符串结构体中的字符转换为大写字符。该函数的第一个参数是指向要转换的字符串结构体的指针，第二个参数是指向要存储结果的字符串结构体的指针，第三个参数指定转换的方式。

下面是一个使用`RtlUpcaseUnicodeString`函数大小写字符串转换的示例代码：

```
#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;
}
```

代码输出效果如下图所示: