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Windows内核中，字符串的处理十分重要。不同于用户态程序，内核中的字符串必须遵循严格的安全规则，以确保不会引发各种安全漏洞。

`ANSI_STRING`和`UNICODE_STRING`是微软在内核中推出的两种安全版本的字符串结构体，`ANSI_STRING`代表的是`ANSI`多字节模式的字符串，而`UNICODE_STRING`则代表的是`UNCODE`类型的字符。这两种字符串类型可以相互转换，因此在内核编程中，需要经常进行类型转换。

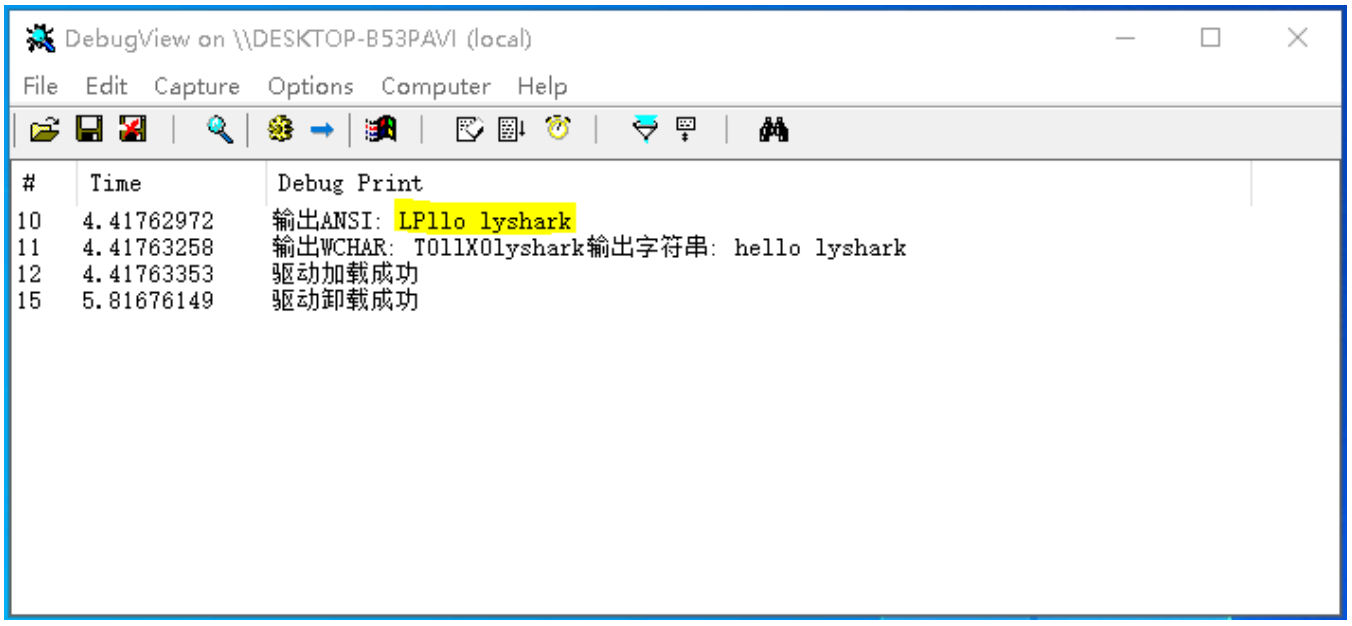
`ANSI_STRING`和`UNICODE_STRING`之间的转换可以通过内核中提供的一系列函数实现。其中，最常用的是`RtlUnicodeStringToAnsiString`和`RtlAnsiStringToUnicodeString`这两个函数。这两个函数分别用于将`UNICODE_STRING`类型的字符串转换成`ANSI_STRING`类型的字符串，以及将`ANSI_STRING`类型的字符串转换成`UNICODE_STRING`类型的字符串。

初始化字符串: 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，使用`ANSI`字符串时需要调用`RtlInitAnsiString`函数进行初始化，而使用`Unicode`字符串时则需要调用`RtlInitUnicodeString`函数进行初始化。这两个函数都需要传入要初始化的字符串和字符串长度，初始化完成后就可以对字符串进行使用了。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;
}
```

代码输出效果如下图所示;



字符串与整数转换: 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。

通常使用 `RtlUnicodeStringToInteger` 函数来将 `Unicode` 字符串转换为整数，函数原型为:

```
NTSYSAPI NTSTATUS NTAPI RtlUnicodeStringToInteger(
  PCUNICODE_STRING   String,
  ULONG              Base,
  PULONG             Value
);
```

其中，`String` 参数为输入的 `Unicode` 字符串，`Base` 参数为进制数（通常为10进制），`Value` 参数为输出的整数。返回值为函数执行状态，如果成功则返回 `STATUS_SUCCESS`。

与之对应的是 `RtlIntegerToUnicodeString` 函数，用于将整数转换为 `Unicode` 字符串，函数原型为:

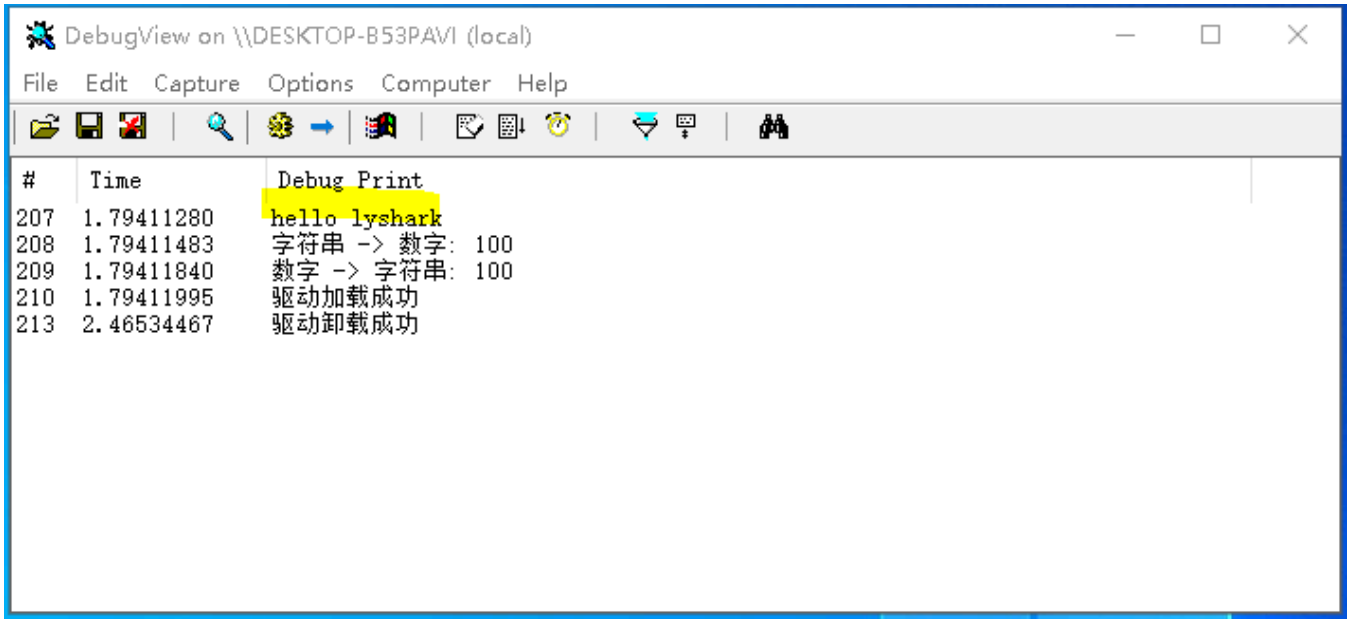
```
NTSYSAPI NTSTATUS NTAPI RtlIntegerToUnicodeString(
  ULONG  Value,
  ULONG  Base,
  PUNICODE_STRING  String
);
```

其中，`Value`参数为输入的整数，`Base`参数为进制数，`String`参数为输出的`Unicode`字符串。返回值同样为函数执行状态，如果成功则返回`STATUS_SUCCESS`。

```
#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)
{
  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
  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;
}
```

代码输出效果如下图所示;



字符串ANSI与UNICODE: 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。

将`UNICODE_STRING`结构转换成`ANSI_STRING`结构的代码，核心部分可归纳为:

```
ANSI_STRING AnsiStr;
UNICODE_STRING UniStr;
RtlUnicodeStringToAnsiString(&AnsiStr, &UniStr, TRUE);
```

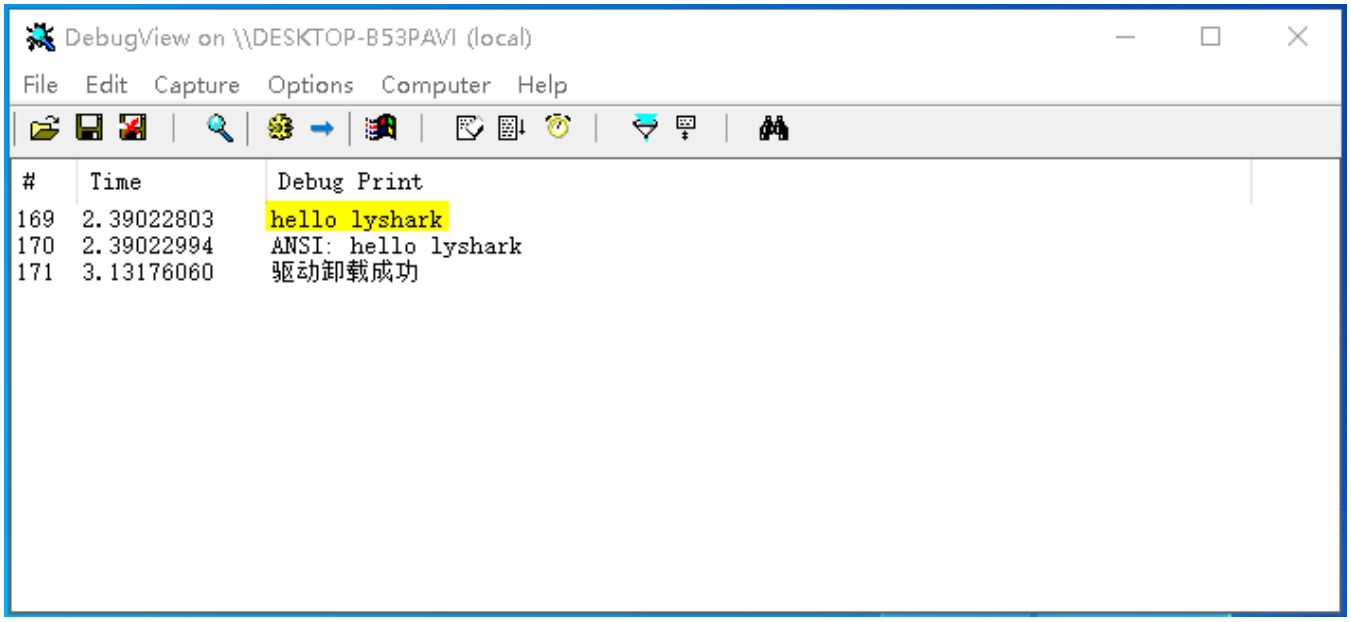
其中，`AnsiStr`是要存储转换后的`ANSI`字符串的结构体，`UniStr`是要转换的`UNICODE`字符串结构体，第三个参数`TRUE`表示要分配一个缓冲区来存储转换后的字符串。

注意，使用`RtlUnicodeStringToAnsiString`函数时，需要在使用完后调用`RtlFreeAnsiString`函数来释放所分配的缓冲区。

```
#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)
{
    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);

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

代码输出效果如下图所示;



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。

`RtlAnsiStringToUnicodeString` 函数的作用是将 `ANSI_STRING` 结构体转换成 `UNICODE_STRING` 结构体，其中 `ANSI_STRING` 代表的是 `ANSI` 格式的字符串，而 `UNICODE_STRING` 代表的是 `Unicode` 格式的字符串。具体实现过程如下:

首先需要定义一个 `ANSI_STRING` 结构体变量 `ansiStr`，并初始化其中的 `Buffer、MaximumLength` 和 `Length` 成员变量，其中 `Buffer` 成员变量指向存储 `ANSI` 格式字符串的缓冲区， `MaximumLength` 成员变量表示该缓冲区的最大长度， `Length` 成员变量表示该缓冲区中已经使用的长度。

接着需要定义一个 `UNICODE_STRING` 结构体变量 `uniStr`，并初始化其中的 `Buffer、MaximumLength` 和 `Length` 成员变量，其中 `Buffer` 成员变量指向存储 `Unicode` 格式字符串的缓冲区， `MaximumLength` 成员变量表示该缓冲区的最大长度， `Length` 成员变量表示该缓冲区中已经使用的长度。

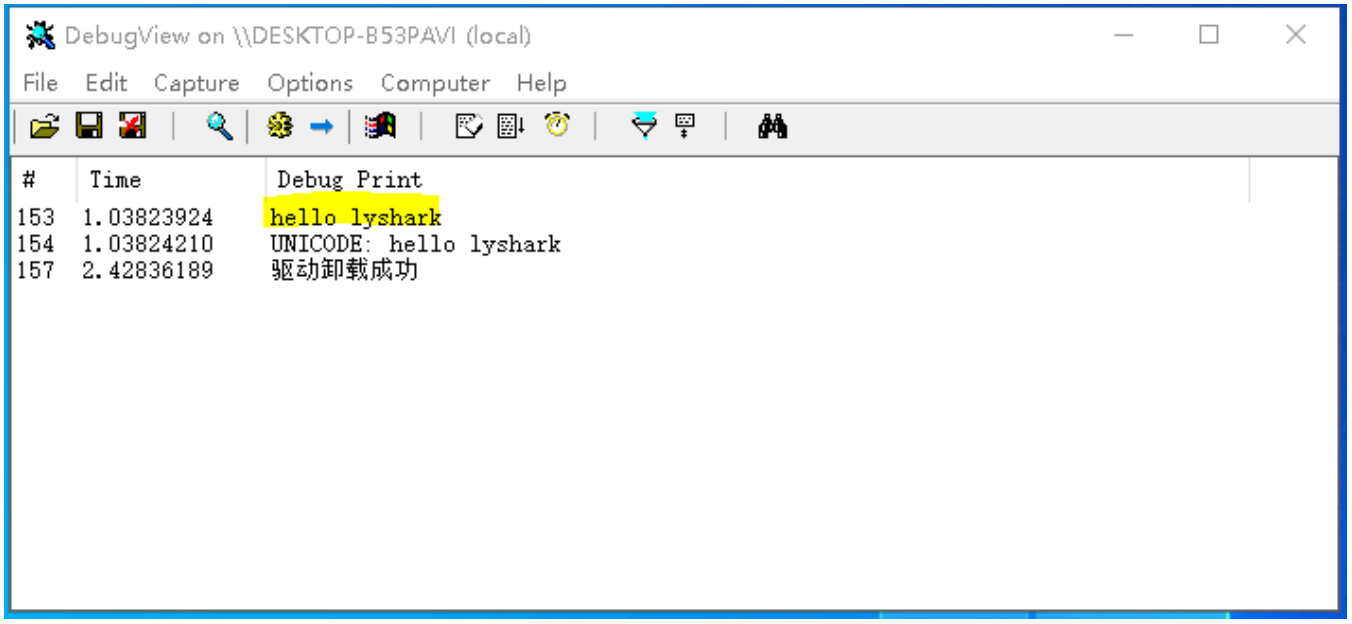
调用`RtlAnsiStringToUnicodeString`函数，传入两个参数，第一个参数为要转换的`UNICODE_STRING`结构体指针，第二个参数为要转换的`ANSI_STRING`结构体指针。函数会将`ANSI_STRING`中的内容转换为`Unicode`格式，并将结果存储在`UNICODE_STRING`结构体的`Buffer`成员变量中。

调用完成后，`uniStr.Buffer`中就存储了转换后的`Unicode`格式字符串，可以进行后续的操作。

需要注意的是，`RtlAnsiStringToUnicodeString`函数在使用完毕后，还需要调用`RtlFreeUnicodeString`函数释放内存。

```
#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)
{
    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);

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

代码输出效果如下图所示;



如上代码是内核通用结构体之间的转换类型，有时我们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 `CHAR*` 类型。将 `UNICODE_STRING` 转换为 `CHAR*` 类型需要先将 `UNICODE_STRING` 转换为 `ANSI_STRING` 类型，然后再将 `ANSI_STRING` 类型转换为 `CHAR*` 类型。

具体步骤可以总结为如下：

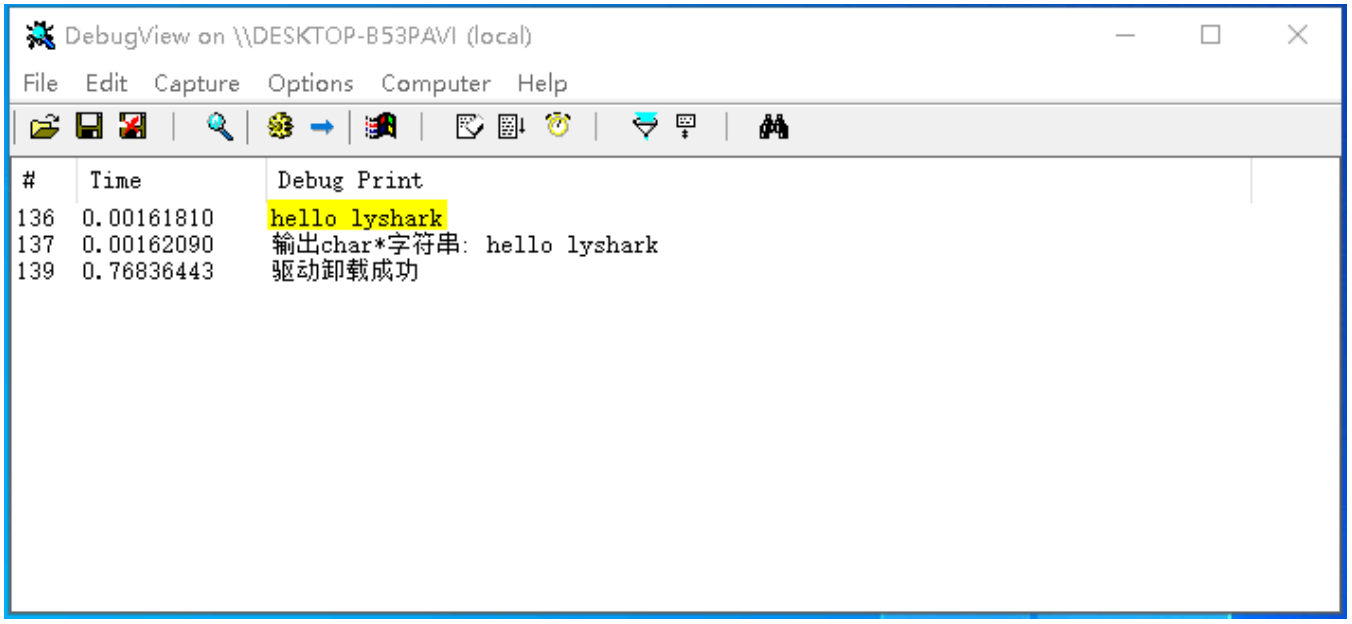
- 1.定义 `ANSI_STRING` 和 `UNICODE_STRING` 类型的变量，分别用于存储转换前后的字符串；
- 2.调用 `RtlUnicodeStringToAnsiString` 函数，将 `UNICODE_STRING` 转换为 `ANSI_STRING` 类型；
- 3.定义一个 `CHAR*` 类型的变量，用于存储转换后的字符串；
- 4.将 `ANSI_STRING` 类型转换为 `CHAR*` 类型，可以使用 `ANSI_STRING.Buffer` 指向的字符数组作为 `CHAR*` 类型的字符串。

以下是示例代码，可用于测试两者的转换模式；

```
#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");
}

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);

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

代码输出效果如下图所示：



如果我们将上述过程反过来实现，将 `CHAR*` 类型转为 `UNICODE_STRING` 结构此时有两种可行的方式；

第一种方式，可以通过调用 `RtlCreateUnicodeStringFromAsciiz` 函数来实现，该函数将 `CHAR*` 类型的字符串转换成 `UNICODE_STRING` 结构体。函数原型如下：

```
NTSYSAPI BOOLEAN RtlCreateUnicodeStringFromAsciiz(
  PUNICODE_STRING DestinationString,
  PCSZ            SourceString
);
```

函数接受两个参数，分别为目标 `UNICODE_STRING` 结构体指针和源字符串指针。函数内部将会动态分配内存并将转换后的 `UNICODE_STRING` 结构体写入到目标结构体指针所指向的内存空间中，同时返回一个布尔值表示操作是否成功。函数的具体用法如下：

```
CHAR* srcString = "Hello, lyshark!";
UNICODE_STRING destString;

RtlCreateUnicodeStringFromAsciiz(&destString, srcString);

// 对 destString 进行操作
RtlFreeUnicodeString(&destString);
```

需要注意的是，`RtlCreateUnicodeStringFromAsciiz` 函数创建的 `UNICODE_STRING` 结构体内存需要手动释放，否则会产生内存泄漏。可以使用 `RtlFreeUnicodeString` 函数来释放该内存，函数原型如下：

```
NTSYSAPI VOID RtlFreeUnicodeString(
  PUNICODE_STRING UnicodeString
);
```

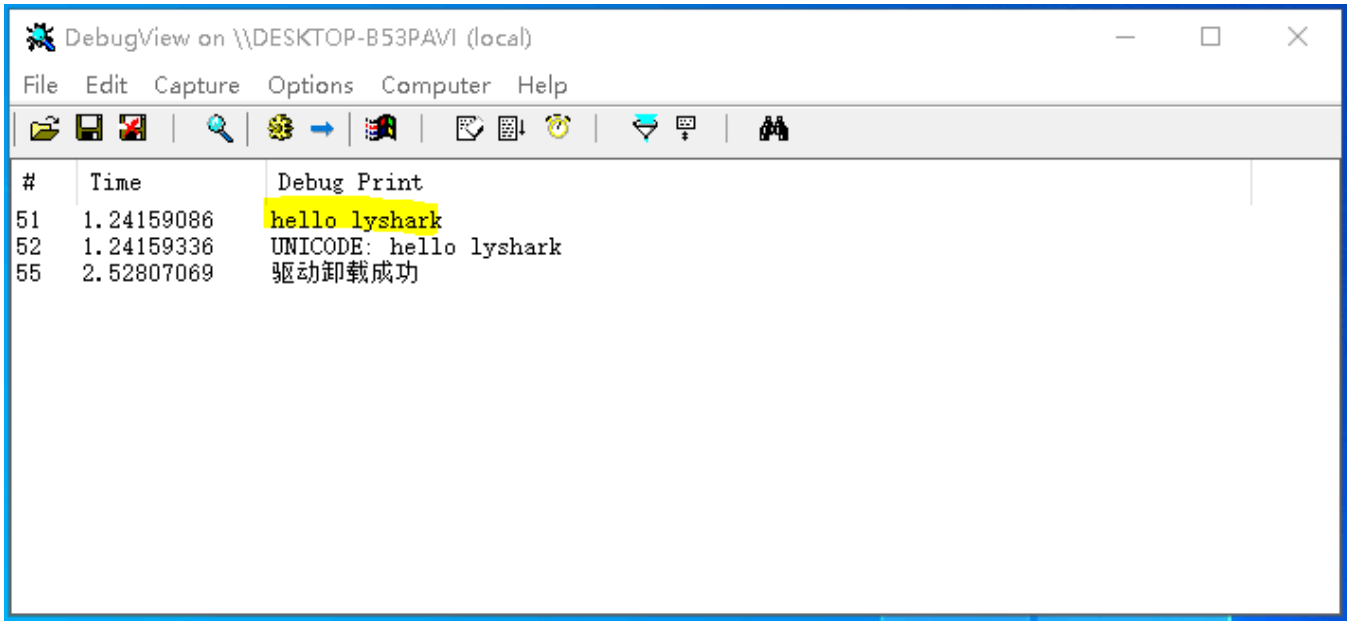
该函数接受一个 `UNICODE_STRING` 结构体指针，用于指定需要释放内存的结构体。

而第二种方法则是通过中转的方式实现，首先用户可使用 `RtlInitString` 将一个 `CHAR*` 初始化为ANSI结构，然后再使用 `RtlAnsiStringToUnicodeString` 一次性完成 `ANSI` 到 `UNICODE` 的类型转换；

```
#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");
}

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);

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

代码输出效果如下图所示：



字符串连接操作: 字符串还可以进行连接操作，例如将两个不同变量中的字符串进行合并，以此来生成一个新的字符串，通过 `RtlAppendUnicodeToString` 这个内核函数即可实现连接。

`RtlAppendUnicodeToString` 用于将 `Unicode` 字符串追加到另一个 `Unicode` 字符串的末尾。这个函数位于 `ntdll.dll` 中，可以通过 `NtDll.lib` 库来链接，函数的原型如下：

```
NTSTATUS RtlAppendUnicodeToString(
    PUNICODE_STRING DestinationString,
    PCWSTR SourceString
);
```

其中，`DestinationString` 是一个指向目标字符串的 `UNICODE_STRING` 结构体的指针，而 `SourceString` 则是一个指向源字符串的 `wchar_t` 类型的指针。

使用该函数可以很方便地将两个字符串连接起来，只需将第一个字符串作为 `DestinationString` 参数传递，第二个字符串作为 `SourceString` 参数传递即可。这个函数将会自动计算两个字符串的长度，并将第二个字符串的内容追加到第一个字符串的末尾。

以下是一个示例代码，将两个字符串 `str1` 和 `str2` 连接起来，并输出结果：

```
#include <ntifs.h>

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

```
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("hello lyshark \n");

    UNICODE_STRING dst;
    WCHAR dst_buf[256];
    NTSTATUS status;

    // 初始化字符串
    UNICODE_STRING src = RTL_CONSTANT_STRING(L"hello");

    // 字符串初始化为空串，长度为256
    RtlInitEmptyUnicodeString(&dst, dst_buf, 256 * sizeof(WCHAR));

    // 将src拷贝到dst
    RtlCopyUnicodeString(&dst, &src);

    // 在dst之后追加
    status = RtlAppendUnicodeToString(&dst, L" lyshark");

    if (status == STATUS_SUCCESS)
    {
        DbgPrint("输出链接后字符串：%wZ \n", &dst);
    }

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

最后，我们使用 `DbgPrint` 函数输出结果。在输出结果之前，我们需要使用 `%wZ` 格式化符号将 `Unicode` 字符串作为参数进行输出。