字符串本质上就是一段内存，之所以和内存使用分开讲，是因为内核里的字符串太有花样了，细数下来竟然有 4 种字符串！这四种字符串，分别是：CHAR*、WCHAR*、ANSI_STRING、UNICODE_STRING。当然，内核里使用频率最多的是 UNICODE_STRING，其次是 WCHAR*，再次是 CHAR*，而 ANSI_STRING，则几乎没见过有什么内核函数使用。

但其实这四种字符串也不是完全独立的，ANSI_STRING 可以看成是 CHAR*的安全性扩展，UNICODE_STRING 可以看成是 WCHAR*的安全性扩展。**CHAR*，可以理解成 CHAR 数组，本质上来说是一个地址，它的有效长度是多少？不知道。字符串有多长？不清楚，遇到\0 就当成是字符串的结尾**。综上所述，CHAR*的安全性是非常糟糕的，如果内核使用 CHAR*作为主要字符串，那么会非常不稳定。**WCHAR*和 CHAR*类似，和 CHAR*的唯一不同在于一个 WCHAR 占 2 个字节**，而一个 CHAR 只占 1 个字节。所以，**WCHAR*的安全性也是非常糟糕的**。微软为了安全性着想，推出了这两种字符串的扩展集，ANSI_STRING 和 UNICODE_STRING。ANSI_STRING 和 UNICODE_STRING 都是结构体，定义如下：

```
typedef struct _ANSI_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PCHAR    Buffer;
} ANSI_STRING, *PANSI_STRING;
```

```
typedef struct _UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PWCHAR    Buffer;
} UNICODE_STRING, *PUNICODE_STRING;
```

可以看到，ANSI_STRING 和 UNICODE_STRING 的结构体大小是一样的，唯一不同在于第三个成员，他们分别对应 CHAR*和 WCHAR*。它们的第一和第二个成员分别代表字符串的长度和内存的有效长度。比如 BUFFER 的长度是 260 字节，而 BUFFER 只是一个单词”FUCK”，则 ANSI_STRING 的 Length=4、MaximumLength=260；UNICODE_STRING 的 Length=8、MaximumLength=260。另外需要注意的是，**CHAR*和 WCHAR*都是以 0 结尾的（CHAR*以 1 个\0 结尾，WCHAR*以 2 个\0 结尾），但 ANSI_STRING 和 UNICODE_STRING 不一定以 0 结尾。因为有了长度的说明，就不需要用特殊标识符来表示结尾了。有些自以为是的人直接把 ANSI_STRING 或 UNICODE_STRING 的 BUFFER 当作字符串用，是极端错误的**。

在内核里，大部分的 C 语言字符串函数都是可以用的，无论是宽版的还是窄版的。比如内核里可以照样使用 strcpy 和 wcscpy，但某些字符串函数签名可能要加上一个下划线。比如 strlwr 要写成_strlwr。ANSI_STRING 和 UNICODE_STRING 有一套专门的字符串函数（在此页面：http://msdn.microsoft.com/en-us/library/windows/hardware/ff563638(v=vs.85).aspx），如果需要查询怎么使用，就用浏览器打开，搜索 AnsiString 或者 UnicodeString，并点击进去查看说明即可。关于 CHAR*和 WCHAR*的字符串操作就不讲了，不懂的请看任意的 C 语言教程。下面举几个关于 UNICODE_STRING 的例子（以下代码借用了张帆写的示例，特此声明感谢）。**1.字符串初始化、2.字符串拷贝、3.字符串比较、4.字符串变大写、5.字符串与整型相互转化、6. ANSI_STRING 字符串与 UNICODE_STRING 字符串相互转换**。

```
//1.字符串初始化
VOID StringInitTest()
{
    //(1)用 RtlInitAnsiString 初始化字符串
    ANSI_STRING    AnsiString1;
    CHAR * string1= "hello";
    //初始化 ANSI_STRING 字符串
```

```
    RtlInitAnsiString(&AnsiString1,string1);
    KdPrint(("AnsiString1:%Z\n",&AnsiString1));//打印 hello
    string1[0]='H';
    string1[1]='E';
    string1[2]='L';
    string1[3]='L';
    string1[4]='O';
    //改变 string1，AnsiString1 同样会导致变化
    KdPrint(("AnsiString1:%Z\n",&AnsiString1));//打印 HELLO
    //(2)程序员自己初始化字符串
#define BUFFER_SIZE 1024
    UNICODE_STRING UnicodeString1 = {0};
    //设置缓冲区大小
    UnicodeString1.MaximumLength = BUFFER_SIZE;
    //分配内存
    UnicodeString1.Buffer = (PWSTR)ExAllocatePool(PagedPool,BUFFER_SIZE);
    WCHAR* wideString = L"hello";
    //设置字符长度,因为是宽字符，所以是字符长度的 2 倍
    UnicodeString1.Length = 2*wcslen(wideString);
    //保证缓冲区足够大，否则程序终止
    ASSERT(UnicodeString1.MaximumLength>=UnicodeString1.Length);
    //内存拷贝，
    RtlCopyMemory(UnicodeString1.Buffer,wideString,UnicodeString1.Length);
    //设置字符长度
    UnicodeString1.Length = 2*wcslen(wideString);
    KdPrint(("UnicodeString:%wZ\n",&UnicodeString1));
    //清理内存
    ExFreePool(UnicodeString1.Buffer);
    UnicodeString1.Buffer = NULL;
    UnicodeString1.Length = UnicodeString1.MaximumLength = 0;
}
```

```
//2.字符串拷贝
VOID StringCopyTest()
{
    //初始化 UnicodeString1
    UNICODE_STRING UnicodeString1;
    RtlInitUnicodeString(&UnicodeString1,L"Hello World");
    //初始化 UnicodeString2
    UNICODE_STRING UnicodeString2={0};
    UnicodeString2.Buffer = (PWSTR)ExAllocatePool(PagedPool,BUFFER_SIZE);
    UnicodeString2.MaximumLength = BUFFER_SIZE;
    //将初始化 UnicodeString2 拷贝到 UnicodeString1
    RtlCopyUnicodeString(&UnicodeString2,&UnicodeString1);
    //分别显示 UnicodeString1 和 UnicodeString2
```

```
    KdPrint(("UnicodeString1:%wZ\n",&UnicodeString1));
    KdPrint(("UnicodeString2:%wZ\n",&UnicodeString2));
    //销毁 UnicodeString2
    //注意!!UnicodeString1 不用销毁
    RtlFreeUnicodeString(&UnicodeString2);
}
```

```
//3.字符串比较
VOID StringCompareTest()
{
    //初始化 UnicodeString1
    UNICODE_STRING UnicodeString1;
    RtlInitUnicodeString(&UnicodeString1,L"Hello World");
    //初始化 UnicodeString2
    UNICODE_STRING UnicodeString2;
    RtlInitUnicodeString(&UnicodeString1,L"Hello");
    if (RtlEqualUnicodeString(&UnicodeString1,&UnicodeString2,TRUE))
        KdPrint(("UnicodeString1 and UnicodeString2 are equal\n"));
    else
        KdPrint(("UnicodeString1 and UnicodeString2 are NOT equal\n"));
}
```

```
//4.字符串变大写
VOID StringToUpperTest()
{
    //初始化 UnicodeString1
    UNICODE_STRING UnicodeString1;
    UNICODE_STRING UnicodeString2;
    RtlInitUnicodeString(&UnicodeString1,L"Hello World");
    //变化前
    DbgPrint("UnicodeString1:%wZ\n",&UnicodeString1);
    //变大写
    RtlUpcaseUnicodeString(&UnicodeString2,&UnicodeString1,TRUE);
    //变化后
    DbgPrint("UnicodeString2:%wZ\n",&UnicodeString2);
    //销毁 UnicodeString2（UnicodeString1 不用销毁）
    RtlFreeUnicodeString(&UnicodeString2);
}
```

```
//5.字符串与整型相互转化
VOID StringToIntegerTest()
{
    //(1)字符串转换成数字
    //初始化 UnicodeString1
    UNICODE_STRING UnicodeString1;
    RtlInitUnicodeString(&UnicodeString1,L"-100");
    ULONG lNumber;
```

```
	NTSTATUS nStatus = RtlUnicodeStringToInteger(&UnicodeString1,10,&lNumber);
	if ( NT_SUCCESS(nStatus))
	{
		KdPrint(("Conver to integer succussfully!\n"));
		KdPrint(("Result:%d\n",lNumber));
	}
	else
	{
		KdPrint(("Conver to integer unsuccessfully!\n"));
	}
	//(2)数字转换成字符串
	//初始化 UnicodeString2
	UNICODE_STRING UnicodeString2={0};
	UnicodeString2.Buffer = (PWSTR)ExAllocatePool(PagedPool,BUFFER_SIZE);
	UnicodeString2.MaximumLength = BUFFER_SIZE;
	nStatus = RtlIntegerToUnicodeString(200,10,&UnicodeString2);
	if ( NT_SUCCESS(nStatus))
	{
		KdPrint(("Conver to string succussfully!\n"));
		KdPrint(("Result:%wZ\n",&UnicodeString2));
	}
	else
	{
		KdPrint(("Conver to string unsuccessfully!\n"));
	}
	//销毁 UnicodeString2
	//注意!!UnicodeString1 不用销毁
	RtlFreeUnicodeString(&UnicodeString2);
}
```

```
//6. ANSI_STRING 字符串与 UNICODE_STRING 字符串相互转换
VOID StringConverTest()
{
	//(1)将 UNICODE_STRING 字符串转换成 ANSI_STRING 字符串
	//初始化 UnicodeString1
	UNICODE_STRING UnicodeString1;
	RtlInitUnicodeString(&UnicodeString1,L"Hello World");
	ANSI_STRING AnsiString1;
	NTSTATUS nStatus = RtlUnicodeStringToAnsiString(&AnsiString1,&UnicodeString1,TRUE);
	if ( NT_SUCCESS(nStatus))
	{
		KdPrint(("Conver succussfully!\n"));
		KdPrint(("Result:%Z\n",&AnsiString1));
	}
	else
```

```
    {
        KdPrint(("Conver unsuccessfully!\n"));
    }
    //销毁 AnsiString1
    RtlFreeAnsiString(&AnsiString1);
    //(2)将 ANSI_STRING 字符串转换成 UNICODE_STRING 字符串
    //初始化 AnsiString2
    ANSI_STRING AnsiString2;
    RtlInitString(&AnsiString2,"Hello World");
    UNICODE_STRING UnicodeString2;
    nStatus = RtlAnsiStringToUnicodeString(&UnicodeString2,&AnsiString2,TRUE);
    if ( NT_SUCCESS(nStatus))
    {
        KdPrint(("Conver succussfully!\n"));
        KdPrint(("Result:%wZ\n",&UnicodeString2));
    }
    else
    {
        KdPrint(("Conver unsuccessfully!\n"));
    }
    //销毁 UnicodeString2
    RtlFreeUnicodeString(&UnicodeString2);
}
```

上述代码因为使用了 KdPrint 宏，所以必须编译成 debug 模式才能显示输出。如果要在所有模式都显示输出，则把 KdPrint 替换成 DbgPrint。另外，以上示例是我在很多书本的示例中精心挑选出来的，简单明了，当然能保证能测试成功。不过如果在实战环境下，因为操作字符串而导致蓝屏的还是非常多见的。根本原因只有两个：1.缓冲区长度溢出；2.操作的指针无效。所以大家以后在做项目时，遇到需要操作字符串的场景还是要格外当心。

最后，附上三个本人写的字符串转换函数，这些函数经过千百次使用都没有蓝屏，可以说是“久经考验”的：

```
//UNICODE_STRINGz 转换为 CHAR*
//输入 UNICODE_STRING 的指针，输出窄字符串，BUFFER 需要已经分配好空间
VOID UnicodeToChar(PUNICODE_STRING dst, char *src)
{
    ANSI_STRING string;
    RtlUnicodeStringToAnsiString(&string,dst, TRUE);
    strcpy(src,string.Buffer);
    RtlFreeAnsiString(&string);
}
```

```
//WCHAR*转换为 CHAR*
//输入宽字符串首地址，输出窄字符串，BUFFER 需要已经分配好空间
VOID WcharToChar(PWCHAR src, PCHAR dst)
```

```
{
	UNICODE_STRING uString;
	ANSI_STRING aString;
	RtlInitUnicodeString(&uString,src);
	RtlUnicodeStringToAnsiString(&aString,&uString,TRUE);
	strcpy(dst,aString.Buffer);
	RtlFreeAnsiString(&aString);
}
```

```
//CHAR*转 WCHAR*
//输入窄字符串首地址，输出宽字符串，BUFFER 需要已经分配好空间
VOID CharToWchar(PCHAR src, PWCHAR dst)
{
	UNICODE_STRING uString;
	ANSI_STRING aString;
	RtlInitAnsiString(&aString,src);
	RtlAnsiStringToUnicodeString(&uString,&aString,TRUE);
	wcscpy(dst,uString.Buffer);
	RtlFreeUnicodeString(&uString);
}
```

思考题：CharToUnicode（CHAR*转 UNICODE_STRING）函数怎么写？