本文主要介绍了`Windows`内核中的一种读写内存的方式：内存拷贝。内存拷贝的实现方式是通过调用内核API函数`MmCopyVirtualMemory`来实现的。该函数的作用是将源进程中的虚拟内存区域复制到目标进程的虚拟内存区域，从而实现内存的拷贝读写。

内存拷贝相比于其他的内存读写方式，其实现比较简单，效率也较高。但是需要注意的是，内存拷贝需要对源地址和目标地址进行合法性检查，并且需要保证源地址和目标地址的访问权限。在实现内存拷贝时，需要注意这些问题，以确保内存拷贝的正确性和安全性。

除了内存拷贝，还有其他的内存读写方式，如CR3读写和MDL读写等。这些读写方式都有其自身的优缺点，需要根据具体的应用场景选择合适的读写方式。在实际开发中，需要综合考虑内存读写的效率、安全性和可靠性等因素，选择最适合的内存读写方式。

内存读取封装

首先封装`KeReadProcessMemory()`内存读取函数，其实现方式是通过调用`MmCopyVirtualMemory`函数实现的。

具体来说，该函数首先获取当前进程的`PEPROCESS`结构体指针，然后调用`MmCopyVirtualMemory`函数，将目标进程中指定地址范围的内存数据拷贝到当前进程中指定的地址上。函数的返回值是一个`NTSTATUS`类型的状态码，用于表示读取操作的结果。

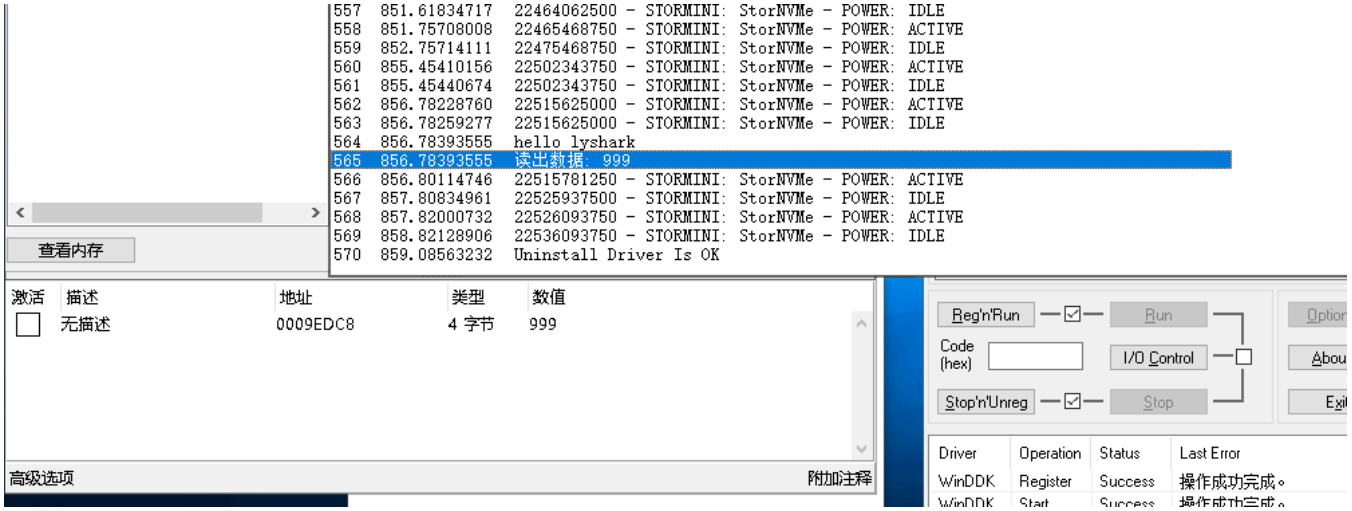
在函数的实现过程中，还包含了一个异常处理语句`__except`，用于捕获读取内存时可能发生的异常。如果发生异常，则函数会返回`STATUS_ACCESS_DENIED`表示读取操作失败。

需要注意的是，该函数只能在内核模式下使用，而不能在用户模式下使用。此外，该函数需要确保源地址和目标地址的合法性，并且需要保证当前进程具有访问目标进程内存的权限，否则将返回`STATUS_ACCESS_DENIED`表示读取操作失败。

```
#include <ntifs.h>
#include <windef.h>
#include <stdlib.h>

NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS *Process);
NTKERNELAPI CHAR* PsGetProcessImageFileName(PEPROCESS Process);
NTSTATUS NTAPI MmCopyVirtualMemory(PEPROCESS SourceProcess, PVOID SourceAddress, PEPROCESS
TargetProcess, PVOID TargetAddress, SIZE_T BufferSize, KPROCESSOR_MODE PreviousMode, PSIZE_T
ReturnSize);

// 定义全局EProcess结构
PEPROCESS Global_Peprocess = NULL;

// 普通Ke内存读取
NTSTATUS KeReadProcessMemory(PVOID SourceAddress, PVOID TargetAddress, SIZE_T Size)
{
    __try
    {
        PEPROCESS TargetProcess = PsGetCurrentProcess();
        SIZE_T Result;
        if (NT_SUCCESS(MmCopyVirtualMemory(Global_Peprocess, SourceAddress, TargetProcess,
TargetAddress, Size, KernelMode, &Result)))
            return STATUS_SUCCESS;
        else
            return STATUS_ACCESS_DENIED;
    }
```

```
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        return STATUS_ACCESS_DENIED;
    }
    return STATUS_ACCESS_DENIED;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("Uninstall Driver Is OK \n");
}

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

    // 根据PID打开进程
    DWORD PID = 6672;
    NTSTATUS nt = PsLookupProcessByProcessId((HANDLE)PID, &Global_Peprocess);

    DWORD ref_value = 0;

    // 将地址处读取4字节到ref_value中
    NTSTATUS read_nt = KeReadProcessMemory((PVOID)0x0009EDC8, &ref_value, 4);

    DbgPrint("读出数据: %d \n", ref_value);

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

编译如上所示代码，并运行即可输出读取数据，读取效果如下图所示：



内存写入封装

有了读取函数，接下来则是封装`KeWriteProcessMemory()`内存写入函数，其实现方式是通过调用`MmCopyVirtualMemory`函数实现。

具体来说，该函数首先获取当前进程的`PEPROCESS`结构体指针，作为源进程；然后将全局变量`Global_Peprocess`作为目标进程；接着调用`MmCopyVirtualMemory`函数，将当前进程中指定地址范围的内存数据拷贝到目标进程中指定的地址上。函数的返回值是一个`NTSTATUS`类型的状态码，用于表示写入操作的结果。

在函数的实现过程中，需要注意保证源地址和目标地址的合法性，并且需要保证当前进程具有向目标进程内存写入数据的权限，否则将返回`STATUS_ACCESS_DENIED`表示写入操作失败。

```
#include <ntifs.h>
#include <windef.h>
#include <stdlib.h>

NTKERNELAPI NTSTATUS PsLookupProcessByProcessId(HANDLE ProcessId, PEPROCESS *Process);
NTKERNELAPI CHAR* PsGetProcessImageFileName(PEPROCESS Process);
NTSTATUS NTAPI MmCopyVirtualMemory(PEPROCESS SourceProcess, PVOID SourceAddress, PEPROCESS
TargetProcess, PVOID TargetAddress, SIZE_T BufferSize, KPROCESSOR_MODE PreviousMode, PSIZE_T
ReturnSize);

// 定义全局EProcess结构
PEPROCESS Global_Peprocess = NULL;


// 普通Ke内存写入
NTSTATUS KeWriteProcessMemory(PVOID SourceAddress, PVOID TargetAddress, SIZE_T Size)
{
    PEPROCESS SourceProcess = PsGetCurrentProcess();
    PEPROCESS TargetProcess = Global_Peprocess;
    SIZE_T Result;

    if (NT_SUCCESS(MmCopyVirtualMemory(SourceProcess, SourceAddress, TargetProcess,
TargetAddress, Size, KernelMode, &Result)))
        return STATUS_SUCCESS;
    else
        return STATUS_ACCESS_DENIED;
}

VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("Uninstall Driver Is OK \n");
}

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

    // 根据PID打开进程
    DWORD PID = 6672;
    NTSTATUS nt = PsLookupProcessByProcessId((HANDLE)PID, &Global_Peprocess);
```

```
    DWORD ref_value = 10;

    // 将地址处写出4字节
    NTSTATUS read_nt = KeWriteProcessMemory((PVOID)0x0009EDC8, &ref_value, 4);

    DbgPrint("写入数据: %d \n", ref_value);

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

编译如上所示代码，并运行即可输出读取数据，写出效果如下图所示：