在本人前一篇博文 《驱动开发：通过ReadFile与内核层通信》 详细介绍了如何使用应用层 ReadFile 系列函数实现内核通信，本篇将继续延申这个知识点，介绍利用 PIPE 命名管道实现应用层与内核层之间的多次通信方法。

# 什么是PIPE管道?

在Windows编程中，数据重定向需要用到管道PIPE，管道是一种用于在进程间共享数据的机制，通常由 两端组成，数据从一端流入则必须从令一端流出，也就是一读一写，利用这种机制即可实现进程间直接 通信。管道的本质其实是一段共享内存区域，多数情况下管道是用于应用层之间的数据交换的，其实驱 动中依然可以使用命名管道实现应用层与内核层的直接通信。

那么如何在内核中创建一个管道？请看以下代码片段，以及MSDN针对函数的解析。

InitializeObjectAttributes

初始化一个 OBJECT\_ATTRIBUTES 结构，它设置将被打开的对象句柄的属性。然后调用方可以将一个指向该结构的指针传递给实际打开句柄的例程。

# ZwCreateFile

该函数的作用时创建或打开一个已经存在的文件，在这里其实是打开 objAttr 这个文件。

KeInitializeEvent

将事件对象初始化为同步 (单个服务) 或通知类型事件，并将其设置为已发出信号或未发出信号的状态。

HANDLE g\_hClient; IO\_STATUS\_BLOCK g\_ioStatusBlock; KEVENT g\_event;

VOID NdisMSleep(IN ULONG MicrosecondsToSleep);

// 初始化管道

void init()

{

UNICODE\_STRING uniName; OBJECT\_ATTRIBUTES objAttr;

RtlInitUnicodeString(&uniName, L"\\DosDevices\\Pipe\\LySharkPipeConn"); InitializeObjectAttributes(&objAttr, &uniName, OBJ\_CASE\_INSENSITIVE |

OBJ\_KERNEL\_HANDLE, NULL, NULL);

ZwCreateFile(&g\_hClient, GENERIC\_READ | GENERIC\_WRITE, &objAttr, &g\_ioStatusBlock, NULL, FILE\_ATTRIBUTE\_NORMAL, 0, FILE\_OPEN, FILE\_SYNCHRONOUS\_IO\_NONALERT, NULL, 0);

if (!g\_hClient)

{

return;

}

KeInitializeEvent(&g\_event, SynchronizationEvent, TRUE);

}

原理就是打开 \\DosDevices\\Pipe\\LySharkPipeConn 文件，然后将事件对象初始化为同步状态。

接下来就是如何将数据发送给应用层的问题，发送问题可以调用 ZwWriteFile 这个内核函数，如下我们实现的效果是将一个 char 类型的字符串传输给应用层。



// 将数据传到R3应用层

// LyShark

VOID ReportToR3(char m\_parameter, int lent)

{

if (!NT\_SUCCESS(ZwWriteFile(g\_hClient, NULL, NULL, NULL, &g\_ioStatusBlock, (void )m\_parameter, lent, NULL, NULL)))

{

DbgPrint("写出错误");

}

}

# 内核层的核心代码就是如上这些，将这些整合在一起完整代码如下所示：



#include <ntifs.h> #include <ndis.h> #include <stdio.h>

HANDLE g\_hClient; IO\_STATUS\_BLOCK g\_ioStatusBlock; KEVENT g\_event;

VOID NdisMSleep(IN ULONG MicrosecondsToSleep);

// 初始化管道

void init()

{

UNICODE\_STRING uniName; OBJECT\_ATTRIBUTES objAttr;

RtlInitUnicodeString(&uniName, L"\\DosDevices\\Pipe\\LySharkPipeConn"); InitializeObjectAttributes(&objAttr, &uniName, OBJ\_CASE\_INSENSITIVE |

OBJ\_KERNEL\_HANDLE, NULL, NULL);

ZwCreateFile(&g\_hClient, GENERIC\_READ | GENERIC\_WRITE, &objAttr, &g\_ioStatusBlock, NULL, FILE\_ATTRIBUTE\_NORMAL, 0, FILE\_OPEN, FILE\_SYNCHRONOUS\_IO\_NONALERT, NULL, 0);

if (!g\_hClient)

{

return;

}

KeInitializeEvent(&g\_event, SynchronizationEvent, TRUE);

}

// 将数据传到R3应用层

// LyShark

VOID ReportToR3(char m\_parameter, int lent)

{

if (!NT\_SUCCESS(ZwWriteFile(g\_hClient, NULL, NULL, NULL, &g\_ioStatusBlock, (void )m\_parameter, lent, NULL, NULL)))

{

DbgPrint("写出错误");



}

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动卸载成功 \n");

}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

init();

// 延时3秒

NdisMSleep(3000000);

DbgPrint("hello lyshark \n"); for (int x = 0; x < 10; x++)

{

// 分配空间

char report = (char )ExAllocatePoolWithTag(NonPagedPool, 4096, 'lysh'); if (report)

{

RtlZeroMemory(report, 4096);

RtlCopyMemory(report, "hello lyshark", 13);

// 发送到应用层ReportToR3(report, 4096); ExFreePool(report);

}

}

DbgPrint(" 驱 动 加 载 成 功 \n"); Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

内核中创建了命名管道，客户端就需要创建一个相同名称的管道，并通过 ReadFile 函数读取管道中的数据，应用层核心代码如下所示：



#include <iostream> #include <windows.h>

int main(int argc, char argv[])

{

HANDLE hPipe = CreateNamedPipe(TEXT("\\\\.\\Pipe\\LySharkPipeConn"), PIPE\_ACCESS\_DUPLEX | FILE\_FLAG\_OVERLAPPED, PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, PIPE\_UNLIMITED\_INSTANCES, 0, 0, NMPWAIT\_WAIT\_FOREVER, NULL);

if (INVALID\_HANDLE\_VALUE == hPipe)

{

return false;

}

const int size = 1024 10;



char buf[size]; DWORD rlen = 0; while (true)

{

//if (ConnectNamedPipe(hPipe, NULL) != NULL)

// PowerBy: LyShark.com if (1)

{

if (ReadFile(hPipe, buf, size, &rlen, NULL) == FALSE)

{

continue;

}

else

{

//接收信息

char buffer\_tmp = (char )&buf;

// 拷贝前半部分,不包括 buffer\_data char buffer = (char )malloc(size);

memcpy(buffer, buffer\_tmp, size);

printf("内核层数据: %s \n", buffer); free(buffer\_tmp);

free(buffer);

}

}

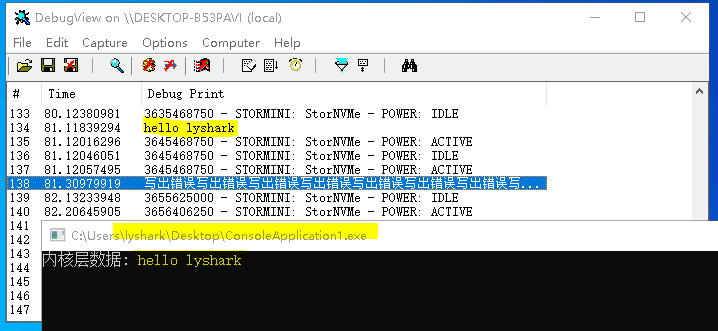
}

system("pause"); return 0;

}

至此将驱动签名后运行，并迅速打开应用层程序等待同步发送事件，即可得到如下返回结果。

此处有必要解释一下为什么会写出错误，很简单这段代码并没有控制何时触发事件，导致两边不同步， 因为只是一个案例用于演示管道的应用方法，所以大家不要太较真，如果不想出错误这段代码还有很多 需要改进的地方。



管道不仅可以传输字符串完全可以传输结构体数据，如下我们定义一个 Networkreport 结构体，并通过管道的方式多次传输给应用层，这部分传输模式适合用于驱动中一次性突出多个结构体，例如进程列表 的输出，ARK工具中的驱动列表输出等功能的实现。

驱动层完整代码

#include <ntifs.h> #include <ndis.h> #include <stdio.h>



HANDLE g\_hClient; IO\_STATUS\_BLOCK g\_ioStatusBlock; KEVENT g\_event;

typedef struct

{

int type;

unsigned long address; unsigned long buffer\_data\_len; char buffer\_data[0];

}Networkreport;

VOID NdisMSleep(IN ULONG MicrosecondsToSleep);

// 初始化管道

void init()

{

UNICODE\_STRING uniName; OBJECT\_ATTRIBUTES objAttr;

RtlInitUnicodeString(&uniName, L"\\DosDevices\\Pipe\\LySharkPipeConn"); InitializeObjectAttributes(&objAttr, &uniName, OBJ\_CASE\_INSENSITIVE |

OBJ\_KERNEL\_HANDLE, NULL, NULL);

ZwCreateFile(&g\_hClient, GENERIC\_READ | GENERIC\_WRITE, &objAttr, &g\_ioStatusBlock, NULL, FILE\_ATTRIBUTE\_NORMAL, 0, FILE\_OPEN, FILE\_SYNCHRONOUS\_IO\_NONALERT, NULL, 0);

if (!g\_hClient)

{

return;

}

KeInitializeEvent(&g\_event, SynchronizationEvent, TRUE);

}

// 将数据传到R3应用层

// PowerBy: LyShark.com

VOID ReportToR3(Networkreport m\_parameter, int lent)

{

if (!NT\_SUCCESS(ZwWriteFile(g\_hClient, NULL, NULL, NULL, &g\_ioStatusBlock, (void )m\_parameter, lent, NULL, NULL)))

{

DbgPrint("写出错误");

}

}

VOID UnDriver(PDRIVER\_OBJECT driver)



{

DbgPrint("驱动卸载成功 \n");

}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

init();

// 延时3秒NdisMSleep(3000000); DbgPrint("hello lyshark \n");

for (int x = 0; x < 10; x++)

{

// 分配空间

Networkreport report = (Networkreport )ExAllocatePoolWithTag(NonPagedPool, 4096, 'lysh');

if (report)

{

RtlZeroMemory(report, 4096);

report->type = x;

report->address = 401000 + x; report->buffer\_data\_len = 13;

// 定位到结构体最后一个元素上

unsigned char tmp = (unsigned char )report + sizeof(Networkreport); memcpy(tmp, "hello lyshark", 13);

// 发送到应用层

ReportToR3(report, 4096); ExFreePool(report);

}

}

DbgPrint(" 驱 动 加 载 成 功 \n"); Driver->DriverUnload = UnDriver;

return STATUS\_SUCCESS;

}

# 应用层完整代码



#include <iostream> #include <windows.h>

typedef struct

{

int type;

unsigned long address; unsigned long buffer\_data\_len; char buffer\_data;

}Networkreport;

int main(int argc, char argv[])

{



HANDLE hPipe = CreateNamedPipe(TEXT("\\\\.\\Pipe\\LySharkPipeConn"),

PIPE\_ACCESS\_DUPLEX | FILE\_FLAG\_OVERLAPPED, PIPE\_TYPE\_MESSAGE | PIPE\_READMODE\_MESSAGE | PIPE\_WAIT, PIPE\_UNLIMITED\_INSTANCES, 0, 0, NMPWAIT\_WAIT\_FOREVER, NULL);

if (INVALID\_HANDLE\_VALUE == hPipe)

{

return false;

}

const int size = 1024 10; char buf[size];

DWORD rlen = 0;

while (true)

{

//if (ConnectNamedPipe(hPipe, NULL) != NULL) if (1 == 1)

{

if (ReadFile(hPipe, buf, size, &rlen, NULL) == FALSE)

{

continue;

}

else

{

//接收信息

Networkreport buffer\_tmp = (Networkreport )&buf; SIZE\_T buffer\_len = sizeof(Networkreport) + buffer\_tmp-

>buffer\_data\_len;

// 拷贝前半部分,不包括 buffer\_data

Networkreport buffer = (Networkreport )malloc(buffer\_len); memcpy(buffer, buffer\_tmp, buffer\_len);

// 对后半部 分配空间

// By: LyShark

char data = (char )malloc(buffer->buffer\_data\_len); unsigned char tmp = (unsigned char )buffer +

sizeof(Networkreport) - 4;

memcpy(data, tmp, buffer->buffer\_data\_len);

printf("输出数据: %s \n", data);

printf("地址: %d \n", buffer\_tmp->address);

printf("长度: %d \n", buffer\_tmp->type);

printf("输出长度: %d \n", buffer\_tmp->buffer\_data\_len);

free(data); free(buffer);

}

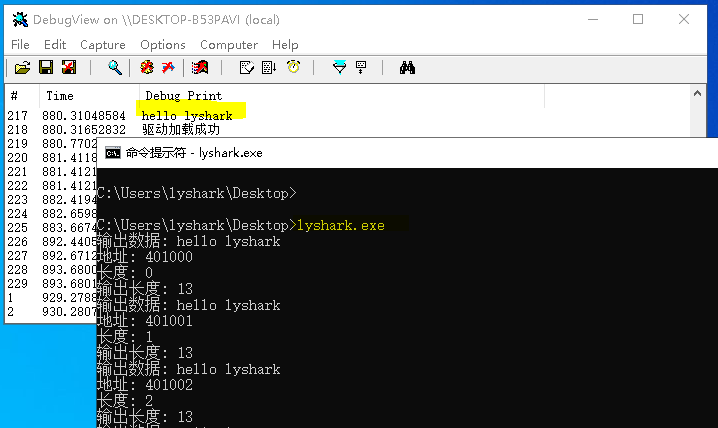
}

}

system("pause"); return 0;

}

# 结构体一次性输出多个，效果如下所示：



本书作者： 王瑞 (LyShark)

作者邮箱： m [e@lyshark.com](mailto:e@lyshark.com)

作者博客： h ttps://lyshark.cnblogs.com

团队首页： w ww.lyshark.com