在笔者前一篇文章 《驱动开发：内核文件读写系列函数》 简单的介绍了内核中如何对文件进行基本的读写操作，本章我们将实现内核下遍历文件或目录这一功能，该功能的实现需要依赖于ZwQueryDirectoryFile 这个内核API函数来实现，该函数可返回给定文件句柄指定的目录中文件的各种信息，此类信息会保存在 PFILE\_BOTH\_DIR\_INFORMATION 结构下，通过遍历该目录即可获取到文件的详细参数，如下将具体分析并实现遍历目录功能。

# 该功能也是ARK工具的最基本功能，如下图是一款通用ARK工具的文件遍历功能的实现效果；



在概述中提到过，目录遍历的核心是 ZwQueryDirectoryFile() 系列函数，该函数可返回给定文件句柄指定的目录中文件的各种信息，其微软官方定义如下；

|  |  |  |  |
| --- | --- | --- | --- |
| [in] BOOLEAN | ReturnSingleEntry, | // | 如果只应返回单个条 |
| 目，则设置为 TRUE ，否则为 FALSE 。 |  |  |  |
| [in, optional] PUNICODE\_STRING | FileName, | // | 文件路径 |
| [in] BOOLEAN | RestartScan | // | 如果扫描是在目录中 |

该函数我们需要注意 FileInformation 参数，在本例中它被设定为了 PFILE\_BOTH\_DIR\_INFORMATION

NTSYSAPI NTSTATUS ZwQueryDirectoryFile(

[in] HANDLE FileHandle,

// 返回的文件对象的句

柄，表示要为其请求信息的目录。

[in, optional] HANDLE

可选句柄。

[in, optional] PIO\_APC\_ROUTINE

调用的可选调用方提供的 APC 例程的地址。

[in, optional] PVOID

Event,

// 调用方创建的事件的

ApcRoutine,

// 请求的操作完成时要

ApcContext,

// 如果调用方提供

APC 或 I/O 完成对象与文件对象关联，则为调用方确定的上下文区域的可选指针。

[out] PIO\_STATUS\_BLOCK IoStatusBlock, // 指向

IO\_STATUS\_BLOCK 结构的指针，该结构接收最终完成状态和有关操作的信息。

[out] PVOID FileInformation, // 指向接收有关文件的所需信息的输出缓冲区的指针。

[in] ULONG Length, // FileInformation

指向的缓冲区的大小（以字节为单位）。

[in]

文件的信息类型。

FILE\_INFORMATION\_CLASS FileInformationClass,

// 要返回的有关目录中

的第一个条目开始，则设置为 TRUE 。

);

# 用于存储当前节点下文件或目录的一些属性，如文件名，文件时间，文件状态等，其次

FileInformationClass 参数也是有多种选择的，本例中我们需要遍历文件或目录则设置成

FileBothDirectoryInformation 就可以，在循环遍历文件时需要将当前目录.以及上一级目录..排除， 而 pDir->FileAttributes 则用于判断当前节点是文件还是目录，属性 FILE\_ATTRIBUTE\_DIRECTORY 代表是目录，反之则是文件，实现目录文件遍历完整代码如下所示；

// 署名权

// right to sign one's name on a piece of work



// PowerBy: LyShark

// Email: [me@lyshark.com](mailto:me@lyshark.com) #include <ntifs.h> #include <ntstatus.h>

// 遍历文件夹和文件

BOOLEAN MyQueryFileAndFileFolder(UNICODE\_STRING ustrPath)

{

HANDLE hFile = NULL;

OBJECT\_ATTRIBUTES objectAttributes = { 0 }; IO\_STATUS\_BLOCK iosb = { 0 };

NTSTATUS status = STATUS\_SUCCESS;

// 初始化结构

InitializeObjectAttributes(&objectAttributes, &ustrPath, OBJ\_CASE\_INSENSITIVE

| OBJ\_KERNEL\_HANDLE, NULL, NULL);

// 打开文件得到句柄

status = ZwCreateFile(&hFile, FILE\_LIST\_DIRECTORY | SYNCHRONIZE | FILE\_ANY\_ACCESS,

&objectAttributes, &iosb, NULL, FILE\_ATTRIBUTE\_NORMAL, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,

FILE\_OPEN, FILE\_DIRECTORY\_FILE | FILE\_SYNCHRONOUS\_IO\_NONALERT | FILE\_OPEN\_FOR\_BACKUP\_INTENT,

NULL, 0);

if (!NT\_SUCCESS(status))

{

return FALSE;

}

// 为节点分配足够的空间

ULONG ulLength = (2 4096 + sizeof(FILE\_BOTH\_DIR\_INFORMATION)) 0x2000;

PFILE\_BOTH\_DIR\_INFORMATION pDir = ExAllocatePool(PagedPool, ulLength);

// 保存pDir的首地址

PFILE\_BOTH\_DIR\_INFORMATION pBeginAddr = pDir;

// 获取信息，返回给定文件句柄指定的目录中文件的各种信息

status = ZwQueryDirectoryFile(hFile, NULL, NULL, NULL, &iosb, pDir, ulLength, FileBothDirectoryInformation, FALSE, NULL, FALSE);

if (!NT\_SUCCESS(status))

{

ExFreePool(pDir); ZwClose(hFile); return FALSE;

}

// 遍历

UNICODE\_STRING ustrTemp; UNICODE\_STRING ustrOne; UNICODE\_STRING ustrTwo;

RtlInitUnicodeString(&ustrOne, L"."); RtlInitUnicodeString(&ustrTwo, L"..");

WCHAR wcFileName[1024] = { 0 }; while (TRUE)

{

// 判断是否是..上级目录或是.本目录

RtlZeroMemory(wcFileName, 1024);

RtlCopyMemory(wcFileName, pDir->FileName, pDir->FileNameLength);

RtlInitUnicodeString(&ustrTemp, wcFileName);

// 是否是.或者是..目录

if ((0 != RtlCompareUnicodeString(&ustrTemp, &ustrOne, TRUE)) && (0 != RtlCompareUnicodeString(&ustrTemp, &ustrTwo, TRUE)))

{

// 判断是文件还是目录

if (pDir->FileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

// 目录

DbgPrint("[目录] 创建时间: %u | 改变时间: %u 目录名: %wZ \n", pDir-

>CreationTime, &pDir->ChangeTime, &ustrTemp);

}

else

{

// 文件

DbgPrint("[文件] 创建时间: %u | 改变时间: %u | 文件名: %wZ \n", pDir-

>CreationTime, &pDir->ChangeTime, &ustrTemp);

}

}

// 遍历完毕直接跳出循环

if (0 == pDir->NextEntryOffset)

{

break;

}

// 每次都要将pDir指向新的地址

pDir = (PFILE\_BOTH\_DIR\_INFORMATION)((PUCHAR)pDir + pDir-

>NextEntryOffset);

}

// 释放内存并关闭句柄ExFreePool(pBeginAddr); ZwClose(hFile);

return TRUE;

}

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.com \n");

// 遍历文件夹和文件

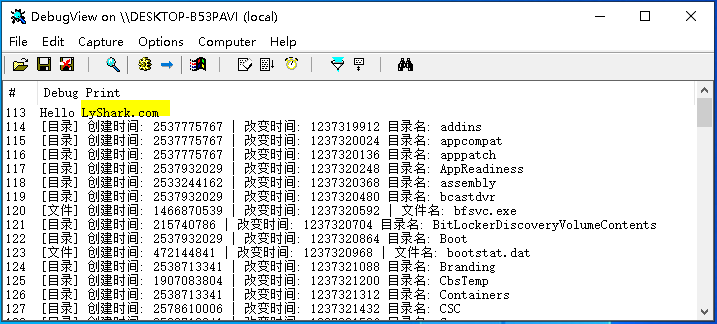
UNICODE\_STRING ustrQueryFile; RtlInitUnicodeString(&ustrQueryFile, L"\\??\\C:\\Windows"); MyQueryFileAndFileFolder(ustrQueryFile);

DbgPrint("驱动加载成功 \n");

Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 编译如上驱动程序并运行，则会输出 C:\\Windows 目录下的所有文件和目录，以及创建时间和修改时间，输出效果如下图所示；



你是否会觉得很失望，为什么不是递归枚举，这里为大家解释一下，通常情况下ARK工具并不会在内核 层实现目录与文件的递归操作，而是将递归过程搬到了应用层，当用户点击一个新目录时，在应用层只 需要拼接新的路径再次发送给驱动程序让其重新遍历一份即可，这样不仅可以提高效率而且还降低了蓝 屏的风险，显然在应用层遍历是更合理的。