RING0 操作文件和 RING3 操作文件在流程上没什么大的区别，也是“获得文件句柄->读/写/删/改->关闭文件句柄”的模式。当然了，只能用内核 API，不能用 WIN32API。在讲解具体的代码之前，先讲解一下文件系统的流程，让大家对整个文件系统有个大概的了解。

假设我们要读写一个文件，无论在 RING3 调用 ReadFile, 还是在 RING0 调用 NtReadFile，它们最终会转换为 IRP，发送到**文件系统驱动**（具体哪个驱动和分区类型相关，如果是 FAT32 分区，则是 FASTFAT.SYS；如果是 NTFS 分区，则是 NTFS.SYS）的 IRP_MJ_READ 分发函数里。文件系统驱动经过一定处理后，就把 IRP 传给**磁盘类驱动**（通常是 CLASSPNP.SYS，此驱动的源码在 WDK 里有）的 IRP_MJ_READ 分发函数处理。磁盘类驱动处理完毕后，又把 IRP 传给**磁盘小端口驱动**的 IRP_MJ_SCSI 分发函数处理。**磁盘小端口驱动太多了，网上有人用 ATAPI.SYS 来指代磁盘小端口驱动，是极端错误的说法**。ATAPI.SYS 是磁盘小端口驱动，但磁盘小端口驱动绝非只能是 ATAPI.SYS，常见的磁盘小端口驱动还有 LSI_SAS.SYS 等。如果安装了芯片组驱动，磁盘小端口驱动通常会被替换成主板厂商的驱动。比安装了英特尔 P67、HM77 的芯片组驱动后，磁盘小端口驱动就会变成 iaStroV.sys。在磁盘小端口驱动里，无论是读还是写，用的都是 IRP_MJ_SCSI 的分发函数。**IRP 被磁盘小端口驱动处理完之后，就要依靠 HAL.DLL 进行端口 IO，此时数据就真的从硬盘里读取了出来**。接下来再按照相反的方向把数据返回到调用者。另外，在内核里，文件夹和文件没啥本质的区别。比如 ZwDeleteFile 既可以删除文件，也可以删除文件夹。接下来举几个例子，让大家了解内核里读写、删除、重命名和枚举文件，以及获取文件信息。

1.复制文件（包括了读文件、写文件）：

```
BOOLEAN ZwCopyFile
(
	IN PUNICODE_STRING	ustrDestFile,	// \??\c:\1.txt
	IN PUNICODE_STRING	ustrSrcFile	// \??\c:\0.txt
)
{
	HANDLE	hSrcFile, hDestFile;
	PVOID	buffer = NULL;
	ULONG	length = 0;
	LARGE_INTEGER	offset = {0};
	IO_STATUS_BLOCK Io_Status_Block = {0};
	OBJECT_ATTRIBUTES obj_attrib;
	NTSTATUS status;
	BOOLEAN	bRet = FALSE;
	do
	{
		// 打开源文件
		InitializeObjectAttributes(	&obj_attrib,
						ustrSrcFile,
						OBJ_CASE_INSENSITIVE		|
OBJ_KERNEL_HANDLE,
						NULL,
						NULL);
```

```
	status = ZwCreateFile(	&hSrcFile,
				GENERIC_READ,
				&obj_attrib,
				&Io_Status_Block,
				NULL,
				FILE_ATTRIBUTE_NORMAL,
				FILE_SHARE_READ,
				FILE_OPEN,
				FILE_NON_DIRECTORY_FILE |
FILE_SYNCHRONOUS_IO_NONALERT,
				NULL,
				0 );
	if (!NT_SUCCESS(status))
	{
		bRet = FALSE;
		goto END;
	}
	// 打开目标文件
	InitializeObjectAttributes(	&obj_attrib,
					ustrDestFile,
					OBJ_CASE_INSENSITIVE |
OBJ_KERNEL_HANDLE,
					NULL,
					NULL);
	status = ZwCreateFile(	&hDestFile,
				GENERIC_WRITE,
				&obj_attrib,
				&Io_Status_Block,
				NULL,
				FILE_ATTRIBUTE_NORMAL,
				FILE_SHARE_READ,
				FILE_OPEN_IF,
				FILE_NON_DIRECTORY_FILE |
FILE_SYNCHRONOUS_IO_NONALERT,
				NULL,
				0 );
	if (!NT_SUCCESS(status))
	{
		bRet = FALSE;
		goto END;
	}
	// 为 buffer 分配 4KB 空间
	buffer = ExAllocatePool(NonPagedPool, 1024 * 4);
	if (buffer == NULL)
```

```
{
    bRet = FALSE;
    goto END;
}
// 复制文件
while (1)
{
    length = 4 * 1024;
    // 读取源文件
    status = ZwReadFile(hSrcFile,
                        NULL,
                        NULL,
                        NULL,
                        &Io_Status_Block,
                        buffer,
                        length,
                        &offset,
                        NULL);
    if (!NT_SUCCESS(status))
    {
        // 如果状态为 STATUS_END_OF_FILE，说明文件已经读取到末尾
        if (status == STATUS_END_OF_FILE)
        {
            bRet = TRUE;
            goto END;
        }
    }
    // 获得实际读取的长度
    length = (ULONG)Io_Status_Block.Information;
    // 写入到目标文件
    status = ZwWriteFile(   hDestFile,
                            NULL,
                            NULL,
                            NULL,
                            &Io_Status_Block,
                            buffer,
                            length,
                            &offset,
                            NULL);
    if (!NT_SUCCESS(status))
    {
        bRet = FALSE;
        goto END;
    }
```

```
                // 移动文件指针
                offset.QuadPart += length;
            }
        }
        while (0);
END:
        if (hSrcFile)
        {
            ZwClose(hSrcFile);
        }
        if (hDestFile)
        {
            ZwClose(hDestFile);
        }
        if (buffer != NULL)
        {
            ExFreePool(buffer);
        }
        return bRet;
}
```

2.删除文件/文件夹：

```
void ZwDeleteFileFolder(WCHAR *wsFileName)
{
        NTSTATUS st;
        OBJECT_ATTRIBUTES ObjectAttributes;
        UNICODE_STRING UniFileName;
        //把 WCHAR*转化为 UNICODE_STRING
        RtlInitUnicodeString(&UniFileName, wsFileName);
        //设置包 OBJECT 对象并使用 ZwDeleteFile 删除
        InitializeObjectAttributes(&ObjectAttributes,
                                   &UniFileName,
                                   OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
                                   NULL,
                                   NULL);
        st=ZwDeleteFile(&ObjectAttributes);
}
```

3.重命名文件/文件夹：

```
typedef struct _FILE_RENAME_INFORMATION
{
        BOOLEAN ReplaceIfExists;
        HANDLE    RootDirectory;
        ULONG     FileNameLength;
```

```
    WCHAR      FileName[1];
} FILE_RENAME_INFORMATION, *PFILE_RENAME_INFORMATION;

NTSTATUS
ZwRenameFile
(
    IN PWSTR SrcFileName,        // \??\x:\xxx\...\xxx.xxx
    IN PWSTR DstFileName         // \??\x:\xxx\...\xxx.xxx
)
{
#define RN_MAX_PATH 2048
#define SFLT_POOL_TAG 'fuck'
    HANDLE FileHandle = NULL;
    OBJECT_ATTRIBUTES ObjectAttributes;
    IO_STATUS_BLOCK IoStatus;
    NTSTATUS Status;
    PFILE_RENAME_INFORMATION RenameInfo = NULL;
    UNICODE_STRING ObjectName;
    //设置重命名的信息
    RenameInfo = (PFILE_RENAME_INFORMATION)ExAllocatePoolWithTag(NonPagedPool,
sizeof(FILE_RENAME_INFORMATION) + RN_MAX_PATH * sizeof(WCHAR), SFLT_POOL_TAG);
    if (RenameInfo == NULL)
    {
        return STATUS_INSUFFICIENT_RESOURCES;
    }
    RtlZeroMemory(RenameInfo, sizeof(FILE_RENAME_INFORMATION) + RN_MAX_PATH *
sizeof(WCHAR));
    RenameInfo->FileNameLength = wcslen(DstFileName) * sizeof(WCHAR);
    wcscpy(RenameInfo->FileName, DstFileName);
    RenameInfo->ReplaceIfExists = 0;
    RenameInfo->RootDirectory = NULL;
    //设置源文件信息并获得句柄
    RtlInitUnicodeString(&ObjectName, SrcFileName);
    InitializeObjectAttributes(&ObjectAttributes,
                               &ObjectName,
                               OBJ_CASE_INSENSITIVE,
                               NULL,
                               NULL);
    Status = ZwCreateFile(&FileHandle,
                          SYNCHRONIZE | DELETE,
                          &ObjectAttributes,
                          &IoStatus,
                          NULL,
                          0,
```

```
                                FILE_SHARE_READ,
                                FILE_OPEN,
                                FILE_SYNCHRONOUS_IO_NONALERT                    |
FILE_NO_INTERMEDIATE_BUFFERING,
                                NULL,
                                0);
    if (!NT_SUCCESS(Status))
    {
        ExFreePoolWithTag(RenameInfo, SFLT_POOL_TAG);
        return Status;
    }
    //最关键一步，利用 ZwSetInformationFile 来设置文件信息
    Status = ZwSetInformationFile(FileHandle,
                                  &IoStatus,
                                  RenameInfo,
                                  sizeof(FILE_RENAME_INFORMATION)               +
RN_MAX_PATH * sizeof(WCHAR),
                                  FileRenameInformation);
    if (!NT_SUCCESS(Status))
    {
        ExFreePoolWithTag(RenameInfo, SFLT_POOL_TAG);
        ZwClose(FileHandle);
        return Status;
    }
    ZwClose(FileHandle);
    return Status;
}
```

4.获取文件大小：

```
//这里传入的是文件句柄不是文件名，大家尝试把这里改成传入文件名
ULONG64 GetFileSize(HANDLE hfile)
{
    IO_STATUS_BLOCK iostatus= {0};
    NTSTATUS ntStatus=0;
    FILE_STANDARD_INFORMATION fsi= {0};
    ntStatus=ZwQueryInformationFile(hfile,
                                    &iostatus,
                                    &fsi,
                                    sizeof(FILE_STANDARD_INFORMATION),
                                    FileStandardInformation);
    if(!NT_SUCCESS(ntStatus))
        return 0;
    return fsi.EndOfFile.QuadPart;
}
```

5.枚举文件（RING3 的 FindFirstFile 和 FindNextFile 内部就是用 ZwQueryDirectoryFile 实现的，为了方便大家以后抄代码，我就把 ZwQueryDirectoryFile 封装成了 RING0 版的 FindFirstFile 和 FindNextFile）：

```
NTKERNELAPI NTSTATUS ZwQueryDirectoryFile   //最关键的 API
(
	HANDLE FileHandle,
	HANDLE Event,
	PIO_APC_ROUTINE ApcRoutine,
	PVOID ApcContext,
	PIO_STATUS_BLOCK IoStatusBlock,
	PVOID FileInformation,
	ULONG Length,
	FILE_INFORMATION_CLASS FileInformationClass,
	BOOLEAN ReturnSingleEntry,
	PUNICODE_STRING FileName,
	BOOLEAN RestartScan
);
//几个常量
#define INVALID_HANDLE_VALUE (HANDLE)-1
#define MAX_PATH2 4096
#define kmalloc(_s) ExAllocatePoolWithTag(NonPagedPool, _s, 'SYSQ')
#define kfree(_p)   ExFreePool(_p)
//枚举文件用到的结构体
typedef struct _FILE_BOTH_DIR_INFORMATION
{
	ULONG NextEntryOffset;
	ULONG FileIndex;
	LARGE_INTEGER CreationTime;
	LARGE_INTEGER LastAccessTime;
	LARGE_INTEGER LastWriteTime;
	LARGE_INTEGER ChangeTime;
	LARGE_INTEGER EndOfFile;
	LARGE_INTEGER AllocationSize;
	ULONG FileAttributes;
	ULONG FileNameLength;
	ULONG EaSize;
	CCHAR ShortNameLength;
	WCHAR ShortName[12];
	WCHAR FileName[1];
} FILE_BOTH_DIR_INFORMATION, *PFILE_BOTH_DIR_INFORMATION;
//山寨版 MyFindFirstFile
HANDLE MyFindFirstFile(LPSTR lpDirectory,PFILE_BOTH_DIR_INFORMATION pDir,ULONG uLength)
```

```
{
	char strFolder[MAX_PATH2]= {0};
	STRING astrFolder;
	UNICODE_STRING ustrFolder;
	OBJECT_ATTRIBUTES oa;
	IO_STATUS_BLOCK ioStatus;
	NTSTATUS ntStatus;
	HANDLE hFind = INVALID_HANDLE_VALUE;
	memset(strFolder,0,MAX_PATH2);
	strcpy(strFolder,"\\??\\");
	strcat(strFolder,lpDirectory);
	RtlInitString(&astrFolder,strFolder);
	if (RtlAnsiStringToUnicodeString(&ustrFolder,&astrFolder,TRUE)==0)
	{
		InitializeObjectAttributes(&oa,&ustrFolder,OBJ_CASE_INSENSITIVE,NULL,NULL);
		ntStatus = IoCreateFile(
					&hFind,
					FILE_LIST_DIRECTORY | SYNCHRONIZE | FILE_ANY_ACCESS,
					&oa,
					&ioStatus,
					NULL,
					FILE_ATTRIBUTE_NORMAL,
					FILE_SHARE_READ | FILE_SHARE_WRITE,
					FILE_OPEN,	//FILE_OPEN
					FILE_DIRECTORY_FILE | FILE_SYNCHRONOUS_IO_NONALERT |
FILE_OPEN_FOR_BACKUP_INTENT,
					NULL,
					0,
					CreateFileTypeNone,
					NULL,
					IO_NO_PARAMETER_CHECKING);
		RtlFreeUnicodeString(&ustrFolder);
		if (ntStatus==0 && hFind!=INVALID_HANDLE_VALUE)
		{
			ntStatus=ZwQueryDirectoryFile(
					hFind, // File Handle
					NULL, // Event
					NULL, // Apc routine
					NULL, // Apc context
					&ioStatus, // IoStatusBlock
					pDir, // FileInformation
					uLength, // Length
					FileBothDirectoryInformation, // FileInformationClass
					TRUE, // ReturnSingleEntry
```

```
                                NULL, // FileName
                                FALSE //RestartScan
                            );
                if (ntStatus!=0)
                {
                    ZwClose(hFind);
                    hFind=INVALID_HANDLE_VALUE;
                }
            }
        }
        return hFind;
}
//山寨版 MyFindNextFile
BOOLEAN MyFindNextFile(HANDLE hFind, PFILE_BOTH_DIR_INFORMATION pDir, ULONG uLength)
{
        IO_STATUS_BLOCK ioStatus;
        NTSTATUS ntStatus;
        ntStatus=ZwQueryDirectoryFile(
                        hFind, // File Handle
                        NULL, // Event
                        NULL, // Apc routine
                        NULL, // Apc context
                        &ioStatus, // IoStatusBlock
                        pDir, // FileInformation
                        uLength, // Length
                        FileBothDirectoryInformation, // FileInformationClass
                        FALSE, // ReturnSingleEntry
                        NULL, // FileName
                        FALSE //RestartScan
                    );
        if (ntStatus==0)
            return TRUE;
        else
            return FALSE;
}
//枚举文件夹内容的函数，输入路径，返回目录下的文件和文件夹数目
ULONG SearchDirectory(LPSTR lpPath)
{
        ULONG muFileCount=0;
        HANDLE hFind=INVALID_HANDLE_VALUE;
        PFILE_BOTH_DIR_INFORMATION pDir;
        char *strBuffer = NULL,*lpTmp=NULL;
        char strFileName[255*2];
```

```
    ULONG uLength=MAX_PATH2*2 + sizeof(FILE_BOTH_DIR_INFORMATION);
    strBuffer = (PCHAR)kmalloc(uLength);
    pDir = (PFILE_BOTH_DIR_INFORMATION)strBuffer;
    hFind=MyFindFirstFile(lpPath,pDir,uLength);
    if (hFind!=INVALID_HANDLE_VALUE)
    {
        kfree(strBuffer);
        uLength=(MAX_PATH2*2 + sizeof(FILE_BOTH_DIR_INFORMATION)) * 0x2000;
        strBuffer = (PCHAR)kmalloc(uLength);
        pDir = (PFILE_BOTH_DIR_INFORMATION)strBuffer;
        if (MyFindNextFile(hFind,pDir,uLength))
        {
            while (TRUE)
            {
                memset(strFileName,0,255*2);
                memcpy(strFileName,pDir->FileName,pDir->FileNameLength);
                if (strcmp(strFileName,"..")!=0 && strcmp(strFileName,".")!=0)
                {
                    if (pDir->FileAttributes & FILE_ATTRIBUTE_DIRECTORY)
                    {
                        DbgPrint("[目录]%S\n",strFileName);
                    }
                    else
                    {
                        DbgPrint("[文件]%S\n",strFileName);
                    }
                    muFileCount++;
                }
                if (pDir->NextEntryOffset==0) break;
                pDir = (PFILE_BOTH_DIR_INFORMATION)((char
*)pDir+pDir->NextEntryOffset);
            }
            kfree(strBuffer);
        }
        ZwClose(hFind);
    }
    return muFileCount;
}
```

6.创建文件夹（其实用 IoCreateFile 也能实现 ZwCreateFile 的功能，ZwCreateFile 不过是 IoCreateFile 的 stub 而已。下面利用 IoCreateFile 创建文件夹）：

```
void ZwCreateFolder(char *FolderPath)
{
    NTSTATUS st;
```

```
    HANDLE FileHandle;
    OBJECT_ATTRIBUTES ObjectAttributes;
    IO_STATUS_BLOCK IoStatusBlock;
    UNICODE_STRING UniFileName;
    WCHAR wsFileName[2048]= {0};
    CharToWchar(FolderPath,wsFileName);
    RtlInitUnicodeString(&UniFileName, wsFileName);
    InitializeObjectAttributes(&ObjectAttributes,
                               &UniFileName,
                               OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
                               NULL,
                               NULL);
    st=IoCreateFile(&FileHandle,
                    GENERIC_READ,
                    &ObjectAttributes,
                    &IoStatusBlock,
                    0,
                    FILE_ATTRIBUTE_NORMAL,
                    0,
                    FILE_CREATE,
                    FILE_DIRECTORY_FILE,
                    NULL,
                    0,
                    0,
                    NULL,
                    IO_NO_PARAMETER_CHECKING);
    if(NT_SUCCESS(st))
        ZwClose(FileHandle);
}
```

最后总结一下几个常见的、和文件相关的 Zw 函数的功能（详细说明可以到此处查看，用 Ctrl+F 打开搜索，然后寻找 FILE 关键字即可）：

ZwCreateFile	创建文件/文件夹、获得文件句柄
ZwDeleteFile	删除文件/文件夹
ZwOpenFile	获得文件句柄
ZwWriteFile	写入文件
ZwQueryDirectoryFile	枚举文件
ZwQueryInformationFile	查询文件信息
ZwReadFile	读文件
ZwSetInformationFile	设置文件信息
ZwWriteFile	写文件
ZwFlushBuffersFile	把磁盘缓存的内容写入到磁盘扇区里

课后作业：大家试一下把这些源码组装起来，弄成一个简易文件管理器。