Windows NT/2000 本机API 参考手册

(美) Gary Nebbett 著

齐舒创作室 译

机械工业出版社
China Machine Press

Windows NT/2000 本机 API 参考手册

（美） Gary Nebbett 著
齐舒创作室 译
毅 鸣 审校

机 械 工 业 出 版 社

本书详细地介绍了 Windows NT/2000 本机 API,清楚地给出了每个 API 的说明、成员(参数)、相关的 Win32 函数和返回值,可供从事 Windows NT/2000 应用程序编程的人员参考,以快速提高编程效率。

Gary Nebbett: Windows NT/2000 Native API Reference

Authorized translation from the English language edition published by Macmillan Technical Publishing

本书版权登记号:图字:01-2000-1311

图书在版编目(CIP)数据

Windows NT/2000 本机 API 参考手册/(美)纳比(Nebbett, G.)著;齐舒创作室译. -北京:机械工业出版社,2001.2

ISBN 7-111-08834-4

Ⅰ.W… Ⅱ.①纳…②齐… Ⅲ.计算机网络-操作系统(软件),Windows NT/2000-技术手册Ⅳ.TP316.86-62

中国版本图书馆 CIP 数据核字(2001)第 12243 号

机械工业出版社(北京市百万庄大街 22 号 邮政编码 100037)

责任编辑:何文军 封面设计:姚毅

三河市宏达印刷厂印刷·新华书店北京发行所发行

2001 年 4 月第 1 版第 1 次印刷

797mm×1092mm 1/16 ·31.75 印张·788 千字

印数:0001-4 000 册

定价:49.00 元

凡购本书,如有缺页、倒页、脱页,由本社发行部调换

本社购书热线电话(010)68993821、68326677-2527

译 者 序

Windows NT 本机应用程序编程接口是为用户模式和内核模式的程序提供的系统服务集。为了演示这些接口不应该由生产应用程序使用，Windows 2000 去除了原来存在于 Windows NT 4.0 中的一些本机 API 例程，并以与 Windows NT 4.0 程序的期望不兼容的方式改变了一些数据结构。

本书比较完整地列出了 Windows NT/2000 本机 API。详细地给出了每个 API 函数或数据结构的说明，阐明了各函数或数据结构的成员或参数，列出了相关的 Win32 函数，并进行了必要的备注。通过查阅本手册，读者可以深入地了解 Windows NT 和 Windows 2000 所提供的系统服务，从而可以深入地挖掘系统的潜力，为开发应用程序提供很好的基础。

本手册是齐舒创作室集体劳动的结晶，参加本书翻译的有刘波、吴杨、刘延、刘小明、李宏军、张世扬、赵天亮、吴齐、关汗羽、李治、李海涛、苏震、卢雪阳、张士华、陈辉、郑明峰、姚正思、沈毅明、汪宝川、高朴真、王立、顾仁、刘航、方东、许胜利、冯卫国、黄丽云和张梦天等。由于译者对 Windows NT/2000 系统内核了解得不是非常深入，所以在翻译过程中，难免在技术术语方面存在问题，望广大读者给予批评指正。

译者

引　言

本机 Windows NT 应用程序编程接口是由 Windows NT 的执行者为用户模式和内核模式的程序提供的系统服务集。本机 API 例程是 UNIX 系统调用或是 VMS 系统服务的等价形式。

在 Windows NT 说明文档中很少提到本机 API,对书中的术语没有任何的约定和固定的提法。“本机 API”和“本机系统服务”都是适当的名称,“本机(native)”这个词用于把 API 与 Win32 API 相区别开来,后者是由大多数 Windows 应用程序使用的操作系统的接口。

通常认为,Microsoft 没有完全对本机 API 进行说明,这样就可以保持灵活性,以便于在操作系统的新版本中对接口进行修改,而不必确保向后的兼容。说明完整的接口等价于所做出的开放的承诺,来支持被认为是过时了的功能的内容。的确,也许为了演示这个接口不应该由生产应用程序所使用,Windows 2000 去除了原来存在于 Windows NT 4.0 中的一些本机 API 例程,并以与 Windows NT 4.0 程序的不兼容的方式改变了一些数据结构。

本书利用 Device Driver Kit(DDK),说明了文档中描述本机 API 的一个子集习惯使用的相同方式来描述本机 API。尽管本机系统服务的参数名称和结构成员以 API 被说明子集的“样式”进行了选择,但与实际名称的任何类似之处都是巧合。所观察到的系统都是运行在 Intel 处理器上的 Windows NT 4.0 Service Pack 3 和 Windows 2000 Release Candidate 2 的自由(零售)构造。本机 API 中的有些例程是在 Intel 平台上实现的(如对 Very Large Memory[VLM]的支持),因此,本书中没有对它们进行描述,对有些其它例程(如对 Virtual DOS Machines[VDM]的支持)只是进行了简要的描述。由 win32k.sys 实现的图形系统服务完全没有被描述。

尽管样式方面存在有类似性,但 DDK 说明文档中存在主要的差别。其中最重要的差别是:本书当中的信息是从对 Windows NT 行为的观察,而不是对其源代码或开发队伍的访问而演绎出来的。

由于这里所提供的信息纯粹以严格的演绎为基础,所以不能保证其中不存在小错误。这些错误并不破坏本书的主题,本书只是提供对本机 API 的描述,以便在类似“资源包”的实用程序(也就是调试和分析工具)和其它“未支持”软件的开发中使用,不鼓励在生产应用程序中使用它们。

可以在 DDK 说明文档和 David Solomon 编写的《Inside Windows NT(第二版)》当中找到的议题的解释在本书中没有重复,并且,假设读者具备 Win32 API、C++ 和 C++ 标准库的知识。

这个引言的目的是,在读者深入研究细节之前,提供有关本机 API 的一些背景。在引言的末尾有一个对组织的简要描述,来帮助引导读者阅读。

使用本机 API

大约 10% 的本机 API 在 DDK 中进行了说明,以便供设备驱动程序编写者使用,且这些例程的定义出现在 ntddk.h 包含文件中。

在 ntddk.h 中定义的本机 API 可以由 Win32 程序调用,但不能像 winnt.h(间接地由 windows.h 包含)一样在相同的编译单元中简单地包含 ntddk.h,因为有些等同的数据结构在这

两个文件中同时进行了定义。下面这个简单的程序在编译时生成很多类型重定义和宏重定义警告和揭示错误:

```
#include <windows.h>
#include <ntddk.h>

int main( )
{
    return 0;
}
```

在 winnt.h 和 ntddk.h 中有很多有用的定义,本书当中的例子不是拷贝其中的一个文件和编辑它来去除重复定义,而是使用 C++ 的一个特征把 Win32 和本机定义置于不同的名称空间中:

```
#include <windows. h>

namespace NT {
    extern "C" {
#pragma warning (disable: 4005 )     // macro redefinition
#include <ntddk. h>
#pragma warning (default: 4005 )
    }
}
using NT:: NTSTATUS;

int main ()
{
    return 0;
}
```

使用 Microsoft C++ 编译器的第 11 个版本时(随 Visual C++ 5.0 发行),这段代码可以在不提出警告的情况下编译,但当利用这个编译器的第 12 个版本编译时(随 Visual C++ 6.0 发行),它生成了许多"注解"。这些注解扩大了由宏重定义警告返回的信息,但与警告不同的是,不能对它们进行抑制。

作为名称空间的这种使用的一个结果,来自于 ntddk.h 的类型定义的所有使用都必须在前面利用"NT::",为 NTSTATUS 形成一个例外(通过"using"语句),以便 NT_SUCCESS 宏继续工作。

为了方便起见,书中的所有例子都使用名为 ntddl.h 的单一的包含文件,这个文件包含了 windows.h、ntddk.h 和本机系统服务的所有定义。

Win32 和本机 API 之间的关系

很多 Win32 函数都是本机 API 例程周围的简单的包装器,且本书当中对本机 API 例程的描述列出了与本机例程相关的 Win32 函数。"相关"这个术语被解释为:Win32 函数使用本机例程实现其大量的功能。假设我们具备 Win32 的知识,则本机系统服务和 Win32 函数之间广泛的等效性的陈述,经常唯一地对系统服务进行了解释。例子 I.1 和 I.2 给出了本机系统服务

周围的两个典型的 Win32 包装器。

例子 I.1:本机 API 例程周围的典型的 Win32 包装

```
HANDLE Win32RootDirectory ()
{
    static HANDLE Handle;

    if (Handle = = 0) {
        NT::UNICODE_STRING ObjectName;
        NT::RtlInitUnicodeString (&ObjectName, L" \ \ BaseNamedObjects" );

        NT::OBJECT_ATTRIBUTES ObjectAttr = {sizeof ObjectAttr,0,
                    &ObjectName,

                    OBJ_CASE_INSENSITIVE};

    ACCESS_MASK DesiredAccess = READ_CONTROL | DIRECTORY_QUERY
                    | DIRECTORY_TRAVERSE
                    | DIRECTORY_CREATE_OBJECT
                    | DIRECTORY_CREATE_SUBDIRECTORY;

    NT:: ZwOpenDirectoryObject (&Handle, DesiredAccess, &ObjectAttr );
    }
    return Handle;
}

HANDLE CreateMutexW(LPSECURITY_ATTRIBUTES SecAttr, BOOL InitialOwner,
                    PCWSTR Name)
{
    NT:: UNICODE_STRING objectName;
    NT:: OBJECT_ATTRIBUTES ObjectAttr = {sizeof ObjectAttr ,0 , 0, OBJ_OPENIF};

    if (Name) {
        NT::RtlInitUnicodeString (&ObjectName, Name );
        ObjectAttr.RootDirectory = Win32RootDirectory ();
        ObjectAttr.ObjectName = &ObjectName;
    }
    if (SecAttr) {
        if (SecAttr - >bInheritHandle ) ObjectAttr.Attributes | = OBJ_INHERIT:
        ObjectAttr.SecurityDescriptor = SecAttr - >lpSecurityDescriptor;
    }

    HANDLE Handle;

    NTSTATUS rv = NT::ZwCreateMutant (&Handle, MUTANT_ALL_ACCESS,
                    &ObjectAttr, InitialOwner);
    if (NT_SUCCESS(rv ) ) {
        SetLastError (rv = = STATUS_OBJECT_NAME_EXISTS
                    ? ERROR_ALREADY_EXISTS: ERROR_SUCCESS);
        return Handle;
```

```
    }
    else {
        SetLastError (NT::RtlNtStatusToDosError (rv ) );
        return 0;
    }
}
```

例子 I.1 的 CreateMutexW 的实现中,围绕 ZwCreateMutant 的包装在典型情况下就是本机系统服务周围的 Win32 包装,它采用一个对象名作为参数。首先,创建一个 UNICODE_STRING 结构来保存名称。UNICODE_STRING 的定义如下:

```
typedef struct_UNICODE_STRING {
    USHORT Length;              // Length of string in bytes
    USHORT MaximumLength;
    PWSTR Buffer;               // Pointer to Unicode string
} UNICODE_STRING, * PUNICODE_STRING;
```

这个字符串使系统服务可以更容易地进行检查,以确保访问名称字符串时不会造成访问违法,因为,访问初期它必须确证 Buffer 和 Buffer + Length 是有效的指针。如果不告诉系统服务字符串的长度,它就将不得不仔细地扫描这个字符串,来查找终止的 null 字符。

```
typedef struct_OBJECT_ATTRIBUTES {
    ULONG Length;
    HANDLE RootDirectory;
    PUNICODE_STRING ObjectName;
    ULONG Attributes;
    PVOID SecurityDescriptor;
    PVOID SecurityQualityOfService;
} OBJECT_ATTRIBUTES, * POBJECT_ATTRIBUTES;
```

UNICODE_STRING 的指针被嵌入在 OBJECT_ATTRIBUTES 结构中,该结构允许将有关名称的附加信息供系统服务使用。后面将对这个结构进行更详细的描述,但对目前的讨论来说,最重要的成员是 RootDirectory,它含有"容器"对象的句柄;ObjectName 成员是相对于这个容器的一个名称。忽略与 Windows Terminal Server 相关的复杂因素,kernel32.dll 是创建和高速缓存名为"\BaseNameObjects"对象目录的一个句柄,并在大多数 Win32 请求中使用这个句柄,作为 RootDirectory 来创建对象(文件和注册表成为主要的例外)。

例子 I.2:本机 API 例程周围最简单的 Win32 包装

```
BOOL ResetEvent (HANDLE Handle )
{
    NTSTATUS rv = NT:: ZwClearEvent (Handle);

    return NT_SUCCESS (rv )
        ? TRUE: SetLastError (NT::RtlNtStatusToDosError(rv)), FALSE;
}
```

当 Win32 函数与本机 API 例程之间的关系类似于例子 I.2 中 ResetEvent 和 ZwClearEvent 的关系时,本机 API 例程定义了后面的注释,即"ResetEvent 揭示了 ZwClearEvent 的完整功能",因

为 ResetEvent 可以做 ZeClearEvent 所能做的所有的事情。这两个例程之间唯一的差别是 TEB 线程环境块(Thread Environment Block)中错误代码的转换(从本机编码规则到 Win32 规则的转换)和存储,如果需要,该错误代码可以由 GetLastError 来检索。

从内核模式调用本机系统服务

本机 API 例程通常可以按照它们的名称来识别,这些名称有两种形式:NtXxx 和 ZwXxx(Xxx 是 DDK 说明文档中为通配符所用的约定)。对于用户模式程序,名称的这两种模式是等价的,只是 ntdll.dll 中相同入口点的两个不同的符号而已。ntdll.dll 入口点的代码执行一条指令,可以使处理器进入内核模式。为了响应这条指令而执行的内核模式代码(系统服务调度程序)在调用实现本机系统服务的特定函数之前,把处理器以前的模式记录在当前线程的 ETHREAD 结构中。

内核模式代码是依靠 ntoskrnl.exe 而不是依靠 ntdll.dll 来链接的。两种不同形式的系统服务名称指向不同的入口点:ZwXxx 形式的入口点含有来自于 ntdll.dll 代码的一个拷贝,用以重新进入内核和使用系统服务调度程序,而 NtXxx 形式的入口点则含有系统服务的实际实现。

例子 I.3 列出了一个本机系统服务的典型序言,并帮助解释使用 ZwXxx 和 NtXxx 形式的系统服务名称之间的差别。如果已经用内核模式运行代码调用了 ZwXxx 入口点,则重新进入系统服务调度程序,该程序把前面的模式更新为内核模式。接着,如果开始执行系统服务的实际代码时,则跳过许多序言。被跳过的代码包含了对特权的检查,并且前面的模式是内核模式的这个事实意味着任何后续的打开对象的企图都将是成功的,而不管附着在对象上的 Access Control List(ACL,访问控制列表)如何。

然而,如果内核模式代码调用了 NtXxx 入口点,其结果就取决于前面运行的模式(内核模式代码对其几乎没有什么控制)。例如,设备驱动程序调度例程中,前面运行的模式可能会随调用的不同而不同。如果系统服务中有作为指针的任何参数,且这些指针指向内核模式程序中的自动或静态变量,那么对系统服务的一个调用就将失败,而提出 STATUS_ ACCESS_ VIOLATION。因为,如果前面运行的模式处于用户模式,则指针就不能成功地完成小于 MmUserProbeAddress 的测试。

如果系统服务没有任何指针参数,则使用 NtXxx 入口点就切合实际,前提是不需要绕过特权的"权限"和访问控制检查。

本书中的所有例子都使用 ZwXxx 形式的名称。

例子 I.3:本机系统服务的典型序言

```
NTSTATUS
NTAPI
NtCreatPagingFile(
    IN PUNICODE_STRING FileName,
    IN PULARGE_INTEGER InitialSize,
    IN PULARGE_INTEGER MaximumSize,
    IN ULONG Reserved
    )
{
    KPROCESSOR_MODE PreviousMode = ExGetPreviousMode();
```

```
_try {
  if (PreviousMode = = UserMode) {

      if (SeSinglePrivilegeCheck (SeCreatePagefilePrivilege,
                          PreviousMode) = = FALSE )
          return STATUS_PRIVILEGE_NOT_HELD;

      if (FileName + 1 > MmUserProbeAddress)
                          ExRaiseAccessViolation ();

      if (PCHAR (FileName - >Buffer) + FileName - >Length >
                          MmUserProbeAddress )
                          ExRaiseAccessViolation ();

      if (ULONG(InitialSize ) & 3 )
          ExRaiseDatatypeMisalignment ();

      if (InitialSize + 1 > MmUserProbeAddress )
                          ExRaiseAccessViolation ();

      if (ULONG(MaximumSize) & 3)
                          ExRaiseDatatypeMisalignment ();

      if (MaximumSize + 1 > MmUserProbeAddress )
                          ExRaiseAccessViolation ();
  }

  // Copy the arguments and file name string to kernel memory
}
  _except (PreviousMode = = UserMode
      ? EXCEPTION_EXECUTE_HANDLER: EXCEPTION_CONTINUE_SEARCH) {

      return GetExceptionCode ();
  }

  // Create the page file

  return STATUS_SUCCESS;
}
```

例子 I.3 中 NtCreatePagingFile 定义中的参数前面使用了“IN”字符串，它是展开为一个空字符串的宏。这是 DDK 说明文档所使用的一种约定的组成部分，表示一个参数是输入参数、输出参数，还是既是输入又是输出参数。如果这个参数是输入参数，则其前面就使用“OUT”，而如果它既是输入参数又是输出参数，则其前面就使用“IN OUT”。如果一个参数是指针，参数名称后面有时就跟有“OPTIONAL”的字符串，表示可以使用一个空指针来作为变元。

返回本机系统服务的值

本机系统服务的描述包含有关可能的返回值的信息，对于大多数系统服务来说，这个值是

一个状态代码。正常情况下，值的列表是不完整的，同时，又因为难于得到，且几乎无法注意到很多系统服务可能会失败，例如，提出 STATUS_INVALID_PARAMETER 警告，被指出的值旨在表示系统可能会失败的有趣方式。

本机系统服务使用的相对频率

运行 Windows NT 的一个被检查过的构造时，系统对每个系统服务的调用频率进行计数。表 I.1 为 Windows NT 4.0 系统服务调用频率排序的计数值，在采集数据之前，系统运行了几个小时。

表 I.1 本机系统服务使用的相对频率

Call count	Service name
399236	NtWaitForSingleObject
245859	NtReplyWaitReceivePort
237888	NtRequestWaitReplyPort
78260	NtReadFile
50028	NtClose
44768	NtDeviceIoControlFile
34598	NtSetEvent
27512	NtClearEvent
23878	NtOpenThreadToken
22551	NtWaitForMultipleObjects
18441	NtOpenKey
14504	NtAllocateVirtualMemory
14361	NtProtectVirtualMemory
13975	NtQueryInformationToken
13335	NtEnumerateKey
12910	NtQueryValueKey
12273	NtOpenProcessToken
11261	NtSetInformationFile
9207	NtFlushInstructionCache
8906	NtQueryInformationFile
7198	NtFreeVirtualMemory
6987	NtReleaseMutant
6637	NtWriteFile
6227	NtQuerySecurityObject
5907	NtEnumerateValueKey
5763	NtOpenFile
5541	NtQueryDirectoryFile
5346	NtQueryDefaultLocale
5062	NtSetValueKey
4716	NtRequestPort
4335	NtQueryKey
3927	NtQueryAttributesFile
3805	NtMapViewOfSection

3787	NtCreateFile
3576	NtReadVirtualMemory
3007	NtReplyPort
2858	NtQuerySystemTime
2558	NtOpenSection
2477	NtReleaseSemaphore
2461	NtFsControlFile
2414	NtQueryVolumeInformationFile
2360	NtQueryVirtualMemory
2353	NtDuplicateObject
2127	NtWriteVirtualMemory
2095	NtSetInformationProcess
2091	NtQueryInformationProcess
1909	NtCreateSection
1897	NtCreateKey
1741	NtCreateEvent
1651	NtContinue
1645	NtGetContextThread
1590	NtSetContextThread
1232	NtDeleteKey
1106	NtUnmapViewOfSection
980	NtTestAlert
899	NtSetInformationThread
869	NtQueryInformationThread
809	NtQueryEvent
717	NtQuerySystemInformation
678	NtOpenProcess
618	NtDelayExecution
583	NtFlushBuffersFile
582	NtQueryObject
576	NtSetInformationObject
568	NtCreateThread
563	NtResumeThread
555	NtRegisterThreadTerminatePort
414	NtQuerySection
385	NtTerminateProcess
350	NtOpenSymbolicLinkObject
305	NtOpenDirectoryObject
290	NtLockFile
287	NtUnlockFile
269	NtTerminateThread
234	NtAcceptConnectPort
233	NtConnectPort
232	NtCompleteConnectPort
230	NtQuerySymbolicLinkObject

209	NtCreateProcess
166	NtImpersonateClientOfPort
147	NtCreateNamedPipeFile
146	NtCreateProfile
146	NtResetEvent
146	NtStartProfile
145	NtAccessCheck
120	NtCallbackReturn
116	NtCreateSemaphore
114	NtDeleteValueKey
113	NtQueryDirectoryObject
92	NtAccessCheckAndAuditAlarm
92	NtCloseObjectAuditAlarm
85	NtCreateMutant
76	NtOpenEvent
73	NtAdjustPrivilegesToken
62	NtSetTimer
60	NtDuplicateToken
49	NtAddAtom
49	NtOpenThread
31	NtCreateSymbolicLinkObject
31	NtFindAtom
30	NtDeleteAtom
24	NtOpenSemaphore
24	NtRaiseException
22	NtQueryInformationAtom
22	NtYieldExecution
21	NtImpersonateThread
20	NtCreateMailslotFile
18	NtCreatePort
18	NtNotifyChangeDirectoryFile
17	NtLoadDriver
14	NtCreateDirectoryObject
14	NtSetSecurityObject
12	NtDisplayString
12	NtNotifyChangeKey
10	NtOpenObjectAuditAlarm
9	NtPrivilegeCheck
9	NtQueryPerformanceCounter
8	NtAllocateLocallyUniqueId
8	NtCreateTimer
8	NtPrivilegedServiceAuditAlarm
8	NtRemoveIoCompletion
7	NtFlushKey
6	NtAllocateUuids

6	NtMakeTemporaryObject
6	NtSetSystemInformation
5	NtQueryTimerResolution
4	NtListenPort
4	NtUnloadDriver
3	NtCreateToken
3	NtRaiseHardError
3	NtReadRequestData
3	NtWriteRequestData
2	NtCancelTimer
2	NtCreatePagingFile
2	NtFlushVirtualMemory
2	NtInitializeRegistry
2	NtSetDefaultLocale
1	NtCreateIoCompletion
1	NtLoadKey
1	NtPrivllegeObjectAuditAlarm
1	NtQueryFullAttributesFile
1	NtSetDefaultHardErrorPort
1	NtSetIntervalProfile
1	NtSuspendThread
1	NtUnloadKey
0	NtAdjustGroupsToken
0	NtAlertResumeThread
0	NtAlertThread
0	NtCancelIoFile
0	NtCreateChannel
0	NtCreateEventPair
0	NtDeleteFile
0	NtDeleteObjectAuditAlarm
0	NtExtendSection
0	NtFlushWriteBuffer
0	NtGetPlugPlayEvent
0	NtGetTickCount
0	NtListenChannel
0	NtLoadKey2
0	NtLockVirtualMemory
0	NtOpenChannel
0	NtOpenEventPair
0	NtOpenIoCompletion
0	NtOpenMutant
0	NtOpenTimer
0	NtPlugPlayControl
0	NtPulseEvent
0	NtQueryEaFile

```
0       NtQueryInformationPort
0       NtQueryIntervalProfile
0       NtQueryIoCompletion
0       NtQueryMultipleValueKey
0       NtQueryMutant
0       NtQueryOleDirectoryFile
0       NtQuerySemaphore
0       NtQuerySystemEnvironmentValue
0       NtQueryTimer
0       NtQueueApcThread
0       NtReadFileScatter
0       NtReplaceKey
0       NtReplyWaitReplyPort
0       NtReplyWaitSendChannel
0       NtRestoreKey
0       NtSaveKey
0       NtSendWaitReplyChannel
0       NtSetContextChannel
0       NtSetEaFile
0       NtSetHighEventPair
0       NtSetHighWaitLowEventPair
0       NtSetHighWaitLowThread
0       NtSetInformationKey
0       NtSetInformationToken
0       NtSetIoCompletion
0       NtSetLdtEntries
0       NtSetLowEventPair
0       NtSetLowWaitHighEventPair
0       NtSetLowWaitHighThread
0       NtSetSystemEnvironmentValue
0       NtSetSystemPowerState
0       NtSetSystemTime
0       NtSetTimerResolution
0       NtSetVolumeInformationFile
0       NtShutdownSystem
0       NtSignalAndWaitForSingleObject
0       NtStopProfile
0       NtSystemDebugControl
0       NtUnlockVirtualMemory
0       NtVdmControl
0       NtW32Call
0       NtWaitHighEventPair
0       NtWaitLowEventPair
0       NtWriteFileGather
```

本书的组织

这本书分为17章和4个附录。每章都阐述一组相关(有时不太相关)的系统服务,最后一章,即第17章使用了包罗万象的标题——其它系统服务。每个系统服务的描述旨在可以独立地理解,以便有可能通过目录或者索引找到一项系统服务,并找到所有相关的信息。

章节的顺序是按照例子中使用系统服务之前,就已经对它进行过描述的思路来确定的。例如,描述进程那章的内容位于有关虚拟内存和Section对象的内容之后,因为在演示进程创建的例子中需要用到虚拟内存和Section对象的系统服务。

附录的提供在很大程度上独立于特定系统服务的附加材料(也许附录C例外,这个附录提供与生产和处理异常的系统服务相关的背景信息)。对内核模式的程序员来说,特别重要的是附录A,这个附录讨论了形成可供内核模式程序使用的完整范围的系统服务的技术。

目　录