创建线程

用户模式下，最简单的线程创建函数是 CreateThread。该函数可以在当前进程中创建线程，并可以接受下列参数。

1. 可选的安全属性结构。决定了要附加给新建线程的安全描述符，此外还决定了是否以可继承的方式创建线程句柄。
2. 可选的栈大小。如果指定为零，将采用可执行文件文件头的默认大小。这一点将始终适用于用户模式下进程中创建的第一个线程。
3. 一个函数指针。将充当新线程执行所需的入口点。
4. 一个可选参数。将传递给线程的函数。
5. 可选标志。可用一个标志控制该线程是否以挂起的状态(CREATE SUSPENDED)启动，其他标志控制了有关栈大小参数的解释(初始已提交大小或最大保留大小)。一旦创建完成，系统会为这个新线程返回非零句柄，并且如果调用方请求，还将返回唯一的线程ID。

CreateRemoteThread 则是一种扩展的线程创建函数。该函数可接受一个额外的参数(第一个)，该参数是一种句柄，指向要将该线程创建到的目标进程。我们可以使用该函数将线程注入其他进程。这种技术的一种常见用法是供调试器强制中断被调试的进程。调试器注入该线程，即可立即调用 DebugBreak 函数产生断点。该技术的另一种常见用法是让一个进程获得其他进程的内部信息，如果该进程在目标进程的上下文中运行(例如整个地址空间都是可见的)，这一点将更容易实现。这种方式将可以用在合法和恶意用途中。

为了让 CreateRemoteThread 能够正常工作，进程句柄必须已经具备执行此类操作所需的足够访问权限。作为一种端范例，受保护进程就无法通过这种方式注入，因为所获得的到此类进程的句柄只能提供非常有限的权限。(获取进程句柄权限可以被内核函数ObRegisterCallbacks所注册的回调函数中执行的操作阻止，当然这些回调存于一个链表当中，可以通过清空链表的办法处理所有回调)

如果一切顺利，CreateRemoteThreadEx 最终将调用Ntdll.dll中的NtCreateThreadEx通常这会导致转换至内核模式，并继续在执行体函数NtCreateThreadEx中执行。在这里会执行线程创建过程中内核模式的工作

在内核模式创建线程是通过PsCreateSystemThread 函数(WDK 中提供了文档)实现的。对于需要在系统进程中以独立方式(即不与任何特定进程相关联)工作的驱动程序该函数非常有用。从技术上来看，该函数可用于在任何进程中创建线程，不过这一点对驱动程序而言并不是很有用。

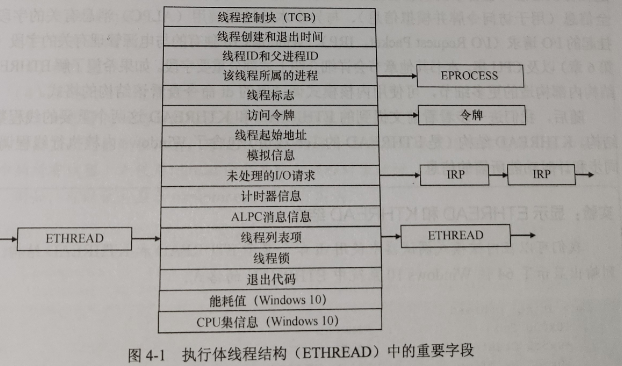
内核线程的退出函数并不能自动销毁线程对象。相反，驱动程序必须在线程函数中调用PsTerminateSystemThread 才能正确地终止线程。因此该函数永远不会返回任何结果。

创建新线程时,线程创建请求被传送至 Windows 执行体，进程管理器在这里为线程对象分配空间并调用内核初始化线程控制块 (KTHREAD)。各种线程创建函数最终都将归结于 CreateRemoteThreadEx。创建 Windows进程的过程中，Kernel32.dll 内部的该函数将执行如下步骤。

1. 该函数将 Windows API参数转换为原生标志，并构建描述对象参数(OBJECT\_ATTRIBUTE 属性)的原生结构。
2. 构建包含客户端ID和TEB 地址这两个项的属性列表。
3. 确定该线程是否要创建在调用进程的内部，或是创建到由句柄传入的另一个进程内部。如果该句柄等于GetCurrentProcess 所返回的伪(pseudo)句柄(值为“-1”)，那么就意味着是同一个进程。如果进程句柄不同，此时依然有可能是同一个进程的有效句柄因此需要调用NtQueryInformationProcess(位于Ntdll中)以确定是否属于这种情况
4. 调用NtCreateThreadEx(位于Ntdll中)转换至内核模式执行体，并在使用相同名称和参数的函数内继续运行。
5. NtCreateThreadEx(位于执行体中)创建并初始化用户模式线程上下文(其结构取决于具体架构)随后调用PspCreateThread 创建一个处于挂起状态的执行体线程对象。随后该函数返回，最终回到用户模式的CreateRemoteThreadEx。
6. CreateRemoteThreadEx为线程分配并行程序集支持所需的激活上下文。随后查询激活栈，以确定是否需要激活，并在需要时进行激活。该激活栈的指针会保存在新线程的TEB中。
7. 除非调用方在创建该线程时设置了CREATE\_SUSPENDED标志，否则至此该线程将恢复，进而可以通过调度参与执行。
8. 线程句柄和线程ID会被返回给调用方。

数据结构

在操作系统层面上，Windows 线程可以由执行体线程对象代表。该执行体线程对象的内部封装了一个ETHREAD结构，该结构所包含的第一个成员便是KTHREAD结构。ETHREAD结构和它指向的其他结构均位于系统地址空间内。唯一的例外是线程环境块(Thread Environment Block, TEB)，它位于进程地址空间内(类似于 PEB，因为用户模式组件需要访问它)。



ETHREAD:

typedef struct \_ETHREAD {

KTHREAD Tcb;

LARGE\_INTEGER CreateTime;

union {

LARGE\_INTEGER ExitTime;

LIST\_ENTRY KeyedWaitChain;

};

PVOID ChargeOnlySession;

PVOID PostBlockList;

union {

PVOID ForwardLinkShadow;

PVOID StartAddress;

ULONG\_PTR TerminationPort;

};

union {

PVOID SessionObject;

PVOID ActiveTimerListLock;

};

union {

PLARGE\_INTEGER AutoAlignmentPadding;

struct {

ULONG\_PTR : 1;

ULONG\_PTR TimerTableLock : 31;

};

};

LIST\_ENTRY ThreadListEntry;

PVOID SchedulingGroup;

ULONG\_PTR Affinity;

UCHAR Priority;

ULONG\_PTR Reserved1;

union {

ULONG\_PTR WaitBlockCount;

ULONG\_PTR SystemAffinityActive;

};

union {

ULONG\_PTR IdealProcessor;

struct {

ULONG\_PTR : 1;

ULONG\_PTR Spare1 : 15;

ULONG\_PTR DisableBoost : 1;

ULONG\_PTR DisableQuantum : 1;

ULONG\_PTR ReservedProcessOR : 12;

ULONG\_PTR ThreadFlags : 2;

};

};

UCHAR PowerState;

UCHAR NpxState;

UCHAR NodeColor;

BOOLEAN WaitBlockFlags;

union {

struct {

ULONG\_PTR : 17;

ULONG\_PTR ThreadAccountingPriority : 5;

ULONG\_PTR : 6;

ULONG\_PTR PreviousMode : 2;

ULONG\_PTR : 6;

};

ULONG\_PTR SoftSystemTrapped;

};

union {

ULONG\_PTR IoPriority : 3;

ULONG\_PTR CpuNumber : 29;

ULONG\_PTR IoQoSWaiterCount : 5;

ULONG\_PTR SharedReadyQueue : 1;

ULONG\_PTR NextProcessor : 29;

ULONG\_PTR Reserved : 1;

};

KEVENT SuspendEvent;

LIST\_ENTRY ThreadListHead;

PVOID SListFaultAddress;

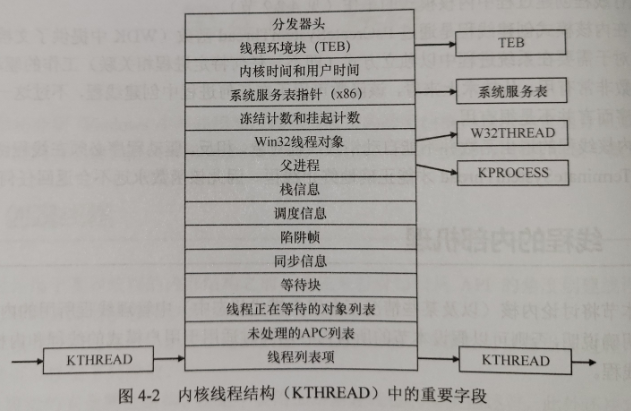
ULONG\_PTR ThreadCounters[ThreadCountersMax];

UCHAR Spare2[48];

PVOID TlsArray;

} ETHREAD, \*PETHREAD;

Windows 子系统进程(Csrss)为 Windows 子系统应用程序中创建的每个线程维护了种名为CSRTHREAD的并行结构。对于已经调用了 Windows 子系统USER或GDI函数的线程，Windows系统的内核模式部分(Win32ksys)还维护了一种每个线程数据结构(W32THREAD)，KTHREAD 结构会指向该数据结构。



KTHREAD:

typedef struct \_KTHREAD

{

DISPATCHER\_HEADER Header;

UINT64 CycleTime;

ULONG HighCycleTime;

UINT64 QuantumTarget;

PVOID InitialStack;

PVOID StackLimit;

PVOID KernelStack;

ULONG ThreadLock;

union

{

KAPC\_STATE ApcState;

UCHAR ApcStateFill[23];

};

CHAR Priority;

WORD NextProcessor;

WORD DeferredProcessor;

ULONG ApcQueueLock;

ULONG ContextSwitches;

UCHAR State;

UCHAR NpxState;

UCHAR WaitIrql;

CHAR WaitMode;

LONG WaitStatus;

union

{

PKWAIT\_BLOCK WaitBlockList;

PKGATE GateObject;

};

union

{

ULONG KernelStackResident: 1;

ULONG ReadyTransition: 1;

ULONG ProcessReadyQueue: 1;

ULONG WaitNext: 1;

ULONG SystemAffinityActive: 1;

ULONG Alertable: 1;

ULONG GdiFlushActive: 1;

ULONG Reserved: 25;

LONG MiscFlags;

};

UCHAR WaitReason;

UCHAR SwapBusy;

UCHAR Alerted[2];

union

{

LIST\_ENTRY WaitListEntry;

SINGLE\_LIST\_ENTRY SwapListEntry;

};

PKQUEUE Queue;

ULONG WaitTime;

union

{

struct

{

SHORT KernelApcDisable;

SHORT SpecialApcDisable;

};

ULONG CombinedApcDisable;

};

PVOID Teb;

union

{

KTIMER Timer;

UCHAR TimerFill[40];

};

union

{

ULONG AutoAlignment: 1;

ULONG DisableBoost: 1;

ULONG EtwStackTraceApc1Inserted: 1;

ULONG EtwStackTraceApc2Inserted: 1;

ULONG CycleChargePending: 1;

ULONG CalloutActive: 1;

ULONG ApcQueueable: 1;

ULONG EnableStackSwap: 1;

ULONG GuiThread: 1;

ULONG ReservedFlags: 23;

LONG ThreadFlags;

};

union

{

KWAIT\_BLOCK WaitBlock[4];

struct

{

UCHAR WaitBlockFill0[23];

UCHAR IdealProcessor;

};

struct

{

UCHAR WaitBlockFill1[47];

CHAR PreviousMode;

};

struct

{

UCHAR WaitBlockFill2[71];

UCHAR ResourceIndex;

};

UCHAR WaitBlockFill3[95];

};

UCHAR LargeStack;

LIST\_ENTRY QueueListEntry;

PKTRAP\_FRAME TrapFrame;

PVOID FirstArgument;

union

{

PVOID CallbackStack;

ULONG CallbackDepth;

};

PVOID ServiceTable;

UCHAR ApcStateIndex;

CHAR BasePriority;

CHAR PriorityDecrement;

UCHAR Preempted;

UCHAR AdjustReason;

CHAR AdjustIncrement;

UCHAR Spare01;

CHAR Saturation;

ULONG SystemCallNumber;

ULONG Spare02;

ULONG UserAffinity;

PKPROCESS Process;

ULONG Affinity;

PKAPC\_STATE ApcStatePointer[2];

union

{

KAPC\_STATE SavedApcState;

UCHAR SavedApcStateFill[23];

};

CHAR FreezeCount;

CHAR SuspendCount;

UCHAR UserIdealProcessor;

UCHAR Spare03;

UCHAR Iopl;

PVOID Win32Thread;

PVOID StackBase;

union

{

KAPC SuspendApc;

struct

{

UCHAR SuspendApcFill0[1];

CHAR Spare04;

};

struct

{

UCHAR SuspendApcFill1[3];

UCHAR QuantumReset;

};

struct

{

UCHAR SuspendApcFill2[4];

ULONG KernelTime;

};

struct

{

UCHAR SuspendApcFill3[36];

PKPRCB WaitPrcb;

};

struct

{

UCHAR SuspendApcFill4[40];

PVOID LegoData;

};

UCHAR SuspendApcFill5[47];

};

UCHAR PowerState;

ULONG UserTime;

union

{

KSEMAPHORE SuspendSemaphore;

UCHAR SuspendSemaphorefill[20];

};

ULONG SListFaultCount;

LIST\_ENTRY ThreadListEntry;

LIST\_ENTRY MutantListHead;

PVOID SListFaultAddress;

PVOID MdlForLockedTeb;

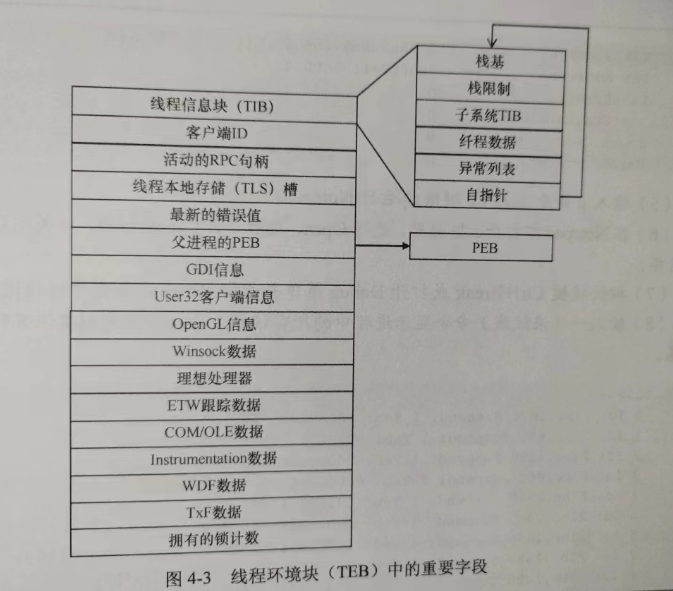
} KTHREAD, \*PKTHREAD;

注意 执行体、高级、图形相关 Win32k 线程结构由KTHREAD指向，而非由ETHREAD指向。

ETHREAD的第一个成员是线程控制块(ThreadControlBlock,TCB)这是一种类型为KTHREAD的结构。随后是线程标识信息进程标识信息(包含了到所有者进程的指针，这样才可以访问环境信息)、指针形式的安全信息(用于访问令牌并模拟信息)、与异步本地过程调用 (ALPC) 消息有关的字段挂起的IO请求(I/O Request Packet，IRP)、Windows 10独有的与电源管理有关的字段以及CPU集。可使用内核模式调试器的 dt 命令查看ETHREAD的格式。

TEB是一种位于进程地址空间(而非系统空间)中的数据结构。从内部来看，它有一个名为线程信息块 (Thread Information Block，TIB)的头部，TIB的存在主要是为了实现与OS/2和 Win9x应用程序的兼容性。在使用初始TIB创建新线程时，还可以借助它将异常和栈信息保存在一个较小的结构中。

TEB存储了映像加载器和各种 Windows DLL使用的上下文信息。由于这些组件运行在用户模式下，因此需要一种在用户模式下可写的数据结构。所以该结构存在于进程地址空间，而非系统空间中。我们可以使用内核模式调试器的!thread 命令找到TEB 的地址。



TEB:

+0x000 NtTib : \_NT\_TIB

+0x038 EnvironmentPointer : Ptr64 Void

+0x040 ClientId : \_CLIENT\_ID

+0x050 ActiveRpcHandle : Ptr64 Void

+0x058 ThreadLocalStoragePointer : Ptr64 Void

+0x060 ProcessEnvironmentBlock : Ptr64 \_PEB

+0x068 LastErrorValue : Uint4B

+0x06c CountOfOwnedCriticalSections : Uint4B

+0x070 CsrClientThread : Ptr64 Void

+0x078 Win32ThreadInfo : Ptr64 Void

+0x080 User32Reserved : [26] Uint4B

+0x0e8 UserReserved : [5] Uint4B

+0x100 WOW32Reserved : Ptr64 Void

+0x108 CurrentLocale : Uint4B

+0x10c FpSoftwareStatusRegister : Uint4B

+0x110 ReservedForDebuggerInstrumentation : [16] Ptr64 Void

+0x190 SystemReserved1 : [30] Ptr64 Void

+0x280 PlaceholderCompatibilityMode : Char

+0x281 PlaceholderHydrationAlwaysExplicit : UChar

+0x282 PlaceholderReserved : [10] Char

+0x28c ProxiedProcessId : Uint4B

+0x290 \_ActivationStack : \_ACTIVATION\_CONTEXT\_STACK

+0x2b8 WorkingOnBehalfTicket : [8] UChar

+0x2c0 ExceptionCode : Int4B

+0x2c4 Padding0 : [4] UChar

+0x2c8 ActivationContextStackPointer : Ptr64 \_ACTIVATION\_CONTEXT\_STACK

+0x2d0 InstrumentationCallbackSp : Uint8B

+0x2d8 InstrumentationCallbackPreviousPc : Uint8B

+0x2e0 InstrumentationCallbackPreviousSp : Uint8B

+0x2e8 TxFsContext : Uint4B

+0x2ec InstrumentationCallbackDisabled : UChar

+0x2ed UnalignedLoadStoreExceptions : UChar

+0x2ee Padding1 : [2] UChar

+0x2f0 GdiTebBatch : \_GDI\_TEB\_BATCH

+0x7d8 RealClientId : \_CLIENT\_ID

+0x7e8 GdiCachedProcessHandle : Ptr64 Void

+0x7f0 GdiClientPID : Uint4B

+0x7f4 GdiClientTID : Uint4B

+0x7f8 GdiThreadLocalInfo : Ptr64 Void

+0x800 Win32ClientInfo : [62] Uint8B

+0x9f0 glDispatchTable : [233] Ptr64 Void

+0x1138 glReserved1 : [29] Uint8B

+0x1220 glReserved2 : Ptr64 Void

+0x1228 glSectionInfo : Ptr64 Void

+0x1230 glSection : Ptr64 Void

+0x1238 glTable : Ptr64 Void

+0x1240 glCurrentRC : Ptr64 Void

+0x1248 glContext : Ptr64 Void

+0x1250 LastStatusValue : Uint4B

+0x1254 Padding2 : [4] UChar

+0x1258 StaticUnicodeString : \_UNICODE\_STRING

+0x1268 StaticUnicodeBuffer : [261] Wchar

+0x1472 Padding3 : [6] UChar

+0x1478 DeallocationStack : Ptr64 Void

+0x1480 TlsSlots : [64] Ptr64 Void

+0x1680 TlsLinks : \_LIST\_ENTRY

+0x1690 Vdm : Ptr64 Void

+0x1698 ReservedForNtRpc : Ptr64 Void

+0x16a0 DbgSsReserved : [2] Ptr64 Void

+0x16b0 HardErrorMode : Uint4B

+0x16b4 Padding4 : [4] UChar

+0x16b8 Instrumentation : [11] Ptr64 Void

+0x1710 ActivityId : \_GUID

+0x1720 SubProcessTag : Ptr64 Void

+0x1728 PerflibData : Ptr64 Void

+0x1730 EtwTraceData : Ptr64 Void

+0x1738 WinSockData : Ptr64 Void

+0x1740 GdiBatchCount : Uint4B

+0x1744 CurrentIdealProcessor : \_PROCESSOR\_NUMBER

+0x1744 IdealProcessorValue : Uint4B

+0x1744 ReservedPad0 : UChar

+0x1745 ReservedPad1 : UChar

+0x1746 ReservedPad2 : UChar

+0x1747 IdealProcessor : UChar

+0x1748 GuaranteedStackBytes : Uint4B

+0x174c Padding5 : [4] UChar

+0x1750 ReservedForPerf : Ptr64 Void

+0x1758 ReservedForOle : Ptr64 Void

+0x1760 WaitingOnLoaderLock : Uint4B

+0x1764 Padding6 : [4] UChar

+0x1768 SavedPriorityState : Ptr64 Void

+0x1770 ReservedForCodeCoverage : Uint8B

+0x1778 ThreadPoolData : Ptr64 Void

+0x1780 TlsExpansionSlots : Ptr64 Ptr64 Void

+0x1788 DeallocationBStore : Ptr64 Void

+0x1790 BStoreLimit : Ptr64 Void

+0x1798 MuiGeneration : Uint4B

+0x179c IsImpersonating : Uint4B

+0x17a0 NlsCache : Ptr64 Void

+0x17a8 pShimData : Ptr64 Void

+0x17b0 HeapData : Uint4B

+0x17b4 Padding7 : [4] UChar

+0x17b8 CurrentTransactionHandle : Ptr64 Void

+0x17c0 ActiveFrame : Ptr64 \_TEB\_ACTIVE\_FRAME

+0x17c8 FlsData : Ptr64 Void

+0x17d0 PreferredLanguages : Ptr64 Void

+0x17d8 UserPrefLanguages : Ptr64 Void

+0x17e0 MergedPrefLanguages : Ptr64 Void

+0x17e8 MuiImpersonation : Uint4B

+0x17ec CrossTebFlags : Uint2B

+0x17ec SpareCrossTebBits : Pos 0, 16 Bits

+0x17ee SameTebFlags : Uint2B

+0x17ee SafeThunkCall : Pos 0, 1 Bit

+0x17ee InDebugPrint : Pos 1, 1 Bit

+0x17ee HasFiberData : Pos 2, 1 Bit

+0x17ee SkipThreadAttach : Pos 3, 1 Bit

+0x17ee WerInShipAssertCode : Pos 4, 1 Bit

+0x17ee RanProcessInit : Pos 5, 1 Bit

+0x17ee ClonedThread : Pos 6, 1 Bit

+0x17ee SuppressDebugMsg : Pos 7, 1 Bit

+0x17ee DisableUserStackWalk : Pos 8, 1 Bit

+0x17ee RtlExceptionAttached : Pos 9, 1 Bit

+0x17ee InitialThread : Pos 10, 1 Bit

+0x17ee SessionAware : Pos 11, 1 Bit

+0x17ee LoadOwner : Pos 12, 1 Bit

+0x17ee LoaderWorker : Pos 13, 1 Bit

+0x17ee SkipLoaderInit : Pos 14, 1 Bit

+0x17ee SpareSameTebBits : Pos 15, 1 Bit

+0x17f0 TxnScopeEnterCallback : Ptr64 Void

+0x17f8 TxnScopeExitCallback : Ptr64 Void

+0x1800 TxnScopeContext : Ptr64 Void

+0x1808 LockCount : Uint4B

+0x180c WowTebOffset : Int4B

+0x1810 ResourceRetValue : Ptr64 Void

+0x1818 ReservedForWdf : Ptr64 Void

+0x1820 ReservedForCrt : Uint8B

+0x1828 EffectiveContainerId : \_GUID

CSR\_THREAD存储了Csrss 为该线程维护的句柄、各种标志、客户端 ID(线程ID 和进程 ID)以及线创建时间的副本。注意，如果某些API需要向 Csrss告知某种操作或状况，那么线程需要向 Csrss 发送消息，且首次发送时要向 Csrss 注册。



W32THREAD结构主要包含对 GDI子系统(笔刷和设备上下文属性)、DirectX，以及硬件供应商开发用户模式打印机驱动时所用的用户模式打印机驱动(User Mode Print Driver，UMPD)框架有用的信息。最后，它还包含了用于桌面合成和抗走样所需的渲染状态数据。

