==========

LIST_ENTRY **PsLoadedModuleList**;

[定　义] wrk\wrk-v1.2\base\ntos\mm\Sysload.c

[初始化] wrk\wrk-v1.2\base\ntos\mm\Sysload.c　[MiInitializeLoadedModuleList()]

[引　用]

- NtQuerySystemInformation()　　查询系统所有内核模块
- MiReleaseAllMemory()　　释放所有内核模块
- MiProcessLoaderEntry()　　从全局链表中插入或删除一个模块

[描　述]

内核加载的所有驱动对象链表。链表所连接结构为 KLDR_DATA_TABLE_ENTRY。使用 NtQuerySystemInformation()查询系统模块信息和进程信息时内部就是在对这个链表进行遍历。

==========

LIST_ENTRY **PsActiveProcessHead**;

[定　义] wrk\wrk-v1.2\base\ntos\ps\Psinit.c

[初始化] wrk\wrk-v1.2\base\ntos\ps\Psinit.c　[PspInitPhase0()]

[引　用]

- PsEnumProcesses()　　遍历所有进程
- PsGetNextProcess()　　获取下一个进程
- PspCreateProcess()　　创建一个新进程插入到链表中

[描　述]

内核所有进程 EPROCESS 对象链表 ,所有获取进程列表的函数都是从这里出发遍历的。该全局变量通过 EPROCESS 中的 ActiveProcessLinks 把所有进程链接在一起。

==========

PEPROCESS **PsIdleProcess**;

[定　义] wrk\wrk-v1.2\base\ntos\ps\Psinit.c

[初始化] wrk\wrk-v1.2\base\ntos\ps\Psinit.c　[PspInitPhase0()]

[引　用]

[描　述]

空闲进程的进程对象

==========

PEPROCESS ***PsInitialSystemProcess***; //导出

[定　义] wrk\wrk-v1.2\base\ntos\ps\Psinit.c

[初始化] wrk\wrk-v1.2\base\ntos\ps\Psinit.c　[PspInitPhase0()]

[引　用]

[描　述]

系统进程(SYSTEM)的进程对象，被内核导出，驱动程序使用它可以遍历全局进程链表。

==========

KSERVICE_TABLE_DESCRIPTOR

KeServiceDescriptorTable [NUMBER_SERVICE_TABLES]; //导出

[定　义] wrk\wrk-v1.2\base\ntos\ke\kernldat.c

[初始化] wrk\wrk-v1.2\base\ntos\ke\keinit.c　[KiSystemStartup ()]

[引　用]

- KeInitThread()　　线程初始化使用 SSDT 表
- KeAddSystemServiceTable()　　增加系统调用表

[描　述]

著名的 SSDT 表，包含了 Windows NT 内核的基本系统服务，供 Ring3 调用。其中宏 NUMBER_SERVICE_TABLES 随不同系统不一样。WRK 中定义为 2。

==========

KSERVICE_TABLE_DESCRIPTOR
KeServiceDescriptorTableShadow [NUMBER_SERVICE_TABLES];

[定　义] wrk\wrk-v1.2\base\ntos\ke\kernldat.c

[初始化] wrk\wrk-v1.2\base\ntos\ke\ keinit.c　[KiInitSystem ()]

[引　用]

- KeAddSystemServiceTable()　　增加系统调用表
- PsConvertToGuiThread()　　GUI 线程使用 Shadow 表
- NtQuerySystemInformation()　　查询系统调用数量

[描　述]

著名的 Shadow SSDT 表，包含了 Windows NT 内核基本服务和 Win32 子系统内核的基本系统服务 ,供 Ring3 调用。其中宏 NUMBER_SERVICE_TABLES 随不同系统不一样。WRK 中定义为 2。

==========

LIST_ENTRY **IopNotifyShutdownQueueHead**;

[定　义] wrk\wrk-v1.2\base\ntos\io\iomgr\Iodata.c

[初始化] wrk\wrk-v1.2\base\ntos\io\iomgr\Ioinit.c　[IoInitSystem()]

[引　用]

- IoRegisterShutdownNotification()　　添加关机回调
- IoUnregisterShutdownNotification()　　删除关机回调
- IoShutdownSystem()　　遍历链表发送关机消息

[描　述]

关机回调驱动对象链表，使用 IoRegisterShutdownNotification()注册的驱动会在这里。链表链接结构为：SHUTDOWN_PACKET

详情查看 WRK 源码中 IoRegisterShutdownNotification()函数实现向该结构添加新的关机回调, IoUnregisterShutdownNotification 从该结构中删除回调。在函数 IoShutdownSystem()中会遍历这个链表，向它们分别发送 IRP_MJ_SHUTDOWN 消息。

==========

LIST_ENTRY **IopNotifyLastChanceShutdownQueueHead**;

[定　义] wrk\wrk-v1.2\base\ntos\io\iomgr\Iodata.c

[初始化] wrk\wrk-v1.2\base\ntos\io\iomgr\Ioinit.c　[IoInitSystem()]

[引　用]

- IoRegisterLastChanceShutdownNotification()　　添加关机回调
- IoUnregisterShutdownNotification()　　删除关机回调
- IoShutdownSystem()　　遍历链表发送关机消息

[描　述]

关机回调驱动对象链表，使用 IoRegisterLastChanceShutdownNotification()注册的驱动会在这里。链表链接结构为：SHUTDOWN_PACKET

注意和 IoRegisterShutdownNotification()不同的是，这里从名字中可以看出 LastChance——最后机会，WDK 中给出了说明，这里注册的关机回调被调用时所有的文件系统已经关闭了，所以不能执行关于文件 IO 的相关操作。看 WRK 也可以印证这一点。

==========

```
LIST_ENTRY IopDriverReinitializeQueueHead;
```

[定　义] wrk\wrk-v1.2\base\ntos\io\iomgr\Iodata.c

[初始化] wrk\wrk-v1.2\base\ntos\io\iomgr\Ioinit.c　[IoInitSystem()]

[引　用]

- IopCallDriverReinitializationRoutines()　　调用驱动二次初始化例程
- IoRegisterDriverReinitialization()　　注册驱动为此初始化例程

[描　述]

驱动二次初始化链表头

==========

```
LIST_ENTRY IopBootDriverReinitializeQueueHead;
```

[定　义] wrk\wrk-v1.2\base\ntos\io\iomgr\Iodata.c

[初始化] wrk\wrk-v1.2\base\ntos\io\iomgr\Ioinit.c　[IoInitSystem()]

[引　用]

- IopCallBootDriverReinitializationRoutines()　调用 0 级驱动二次初始化例程
- IoRegisterBootDriverReinitialization()　　注册 0 级驱动二次初始化例程

[描　述]

0 级驱动二次初始化链表头

==========

```
EX_CALLBACK CmpCallBackVector[CM_MAX_CALLBACKS] = {0};
ULONG CmpCallBackCount = 0;
```

[定　义] wrk\wrk-v1.2\base\ntos\config\Cmhook.c

[初始化] wrk\wrk-v1.2\base\ntos\config\Cmhook.c　[CmpInitCallback()]

[引　用]

- CmRegisterCallback　　注册注册表回调
- CmUnRegisterCallback　　撤销注册表回调
- CmpCallCallBacks　　调用注册表回调

[描　述]

XP 系统上用于保存所有注册表回调的数据结构。驱动程序使用 CmRegisterCallback 注册的回调函数就保存在这里。通过对该数据结构进行处理将可以增加和删除注册表回调。注意，从 Vista 开始使用 CM_CALLBACK_CONTEXT_BLOCKEX 结构，使用 CallbackListHead 全局变量来保存节点。

==========

```
EX_CALLBACK
```

PspCreateProcessNotifyRoutine[PSP_MAX_CREATE_PROCESS_NOTIFY];
ULONG **PspCreateProcessNotifyRoutineCount**;

[定　义] wrk\wrk-v1.2\base\ntos\ps\psp.h

[初始化] wrk\wrk-v1.2\base\ntos\ps\Psinit.c　[PspInitPhase0()]

[引　用]

- PspCreateThread　　创建第一个线程时调用回调函数
- PsSetCreateProcessNotifyRoutine　　添加或删除进程创建/退出回调
- PspExitProcess　　进程退出调用回调函数

[描　述]

驱动程序使用 PsSetCreateProcessNotifyRoutine 添加和删除进程创建/退出回调函数，用于对进程诞生和消亡事件进行捕获。其添加的回调函数保存在这个全局数组中，WRK 中宏 PSP_MAX_CREATE_PROCESS_NOTIFY 定义为 8，最多支持 8 个回调。

==========

EX_CALLBACK
PspCreateThreadNotifyRoutine [PSP_MAX_CREATE_THREAD_NOTIFY];
ULONG **PspCreateThreadNotifyRoutineCount**;

[定　义] wrk\wrk-v1.2\base\ntos\ps\psp.h

[初始化] wrk\wrk-v1.2\base\ntos\ps\Psinit.c　[PspInitPhase0()]

[引　用]

- PspCreateThread　　线程创建时调用回调函数
- PsSetCreateThreadNotifyRoutine　　添加线程创建/退出回调
- PsRemoveCreateThreadNotifyRoutine　　移除线程创建/退出回调
- PspExitThread　　线程退出时调用回调函数

[描　述]

驱动程序使用 PsSetCreateThreadNotifyRoutine 添加和使用 PsRemoveCreateThreadNotifyRoutine 删除线程创建/退出回调函数，用于对线程诞生和消亡事件进行捕获。其添加的回调函数保存在这个全局数组中，WRK 中宏 PSP_MAX_CREATE_THREAD_NOTIFY 定义为 8，最多支持 8 个回调。

==========

EX_CALLBACK
PspLoadImageNotifyRoutine [PSP_MAX_LOAD_IMAGE_NOTIFY];
ULONG **PspLoadImageNotifyRoutineCount**;

[定　义] wrk\wrk-v1.2\base\ntos\ps\psp.h

[初始化] wrk\wrk-v1.2\base\ntos\ps\Psinit.c　[PspInitPhase0()]

[引　用]

- PsSetLoadImageNotifyRoutine　　添加模块加载回调
- PsRemoveLoadImageNotifyRoutine　　移除模块加载回调
- PsCallImageNotifyRoutines　　调用所有模块加载回调

[描　述]

驱动程序使用 PsSetLoadImageNotifyRoutine 添加和使用 PsRemoveLoadImageNotifyRoutine 删除模块加载回调函数，用于对模块加载事件进行

捕获。其添加的回调函数保存在这个全局数组中，WRK 中宏 PSP_MAX_LOAD_IMAGE_NOTIFY 定义为 8，最多支持 8 个回调。

==========

PHANDLE_TABLE PspCidTable;

[定　义] wrk\wrk-v1.2\base\ntos\ps\ Psinit.c

[初始化] wrk\wrk-v1.2\base\ntos\ps\Psinit.c　[PspInitPhase0()]

[引　用]

- PspCreateThread()　　创建线程插入全局句柄表
- PspCreateProcess()　　创建进程插入全局句柄表
- PsLookupThreadByThreadId()　　查找线程内核对象 ETHREAD
- PsLookupProcessByProcessId()　　查找进程内核对象 EPROCESS
- PsLookupProcessThreadByCid()　　根据 ID 同时查进程和线程
- PspProcessDelete()　　删除一个进程
- PspThreadDelete()　　删除一个线程

[描　述]

全局进程内核对象句柄表。PspCidTable 是一个句柄表,其格式与普通的句柄表是完全一样的.但它与每个进程私有的句柄表有以下不同:

1.PspCidTable 中存放的对象是系统中所有的进线程对象指针,其索引就是 PID 和 CID

2.PspCidTable 中存放是对象体(指向 EPROCESS 和 ETHREAD),而每个进程私有的句柄表则存放的是对象头(OBJECT_HEADER)

3.PspCidTable 是一个独立的句柄表,而每个进程私有的句柄表以一个双链连接起来

==========

LIST_ENTRY HandleTableListHead;

[定　义] wrk\wrk-v1.2\base\ntos\ex\Handle.c

[初始化] wrk\wrk-v1.2\base\ntos\ex\Handle.c　[ExInitializeHandleTablePackage ()]

[引　用]

- ExCreateHandleTable()　　创建进程时调用创建句柄表链入全局链表头
- ExDupHandleTable()
- ExSnapShotHandleTables()

[描　述]

系统所有进程的句柄表构成一个链表，链表节点为 HANDLE_TABLE，而上面的 PspCidTable 是个例外。

==========

PHANDLE_OBJECT ObpKernelHandleTable;

[定　义] wrk\wrk-v1.2\base\ntos\ob\obp.h

[初始化] wrk\wrk-v1.2\base\ntos\ob\obinit.c　[ObInitSystem ()]

[引　用]

- ObpCloseHandle()　　关闭句柄（内核句柄）
- ObpCreateHandle()　　创建句柄（内核句柄）
- ObpCreateUnnamedHandle()　　创建匿名句柄（内核句柄）

- ObSetHandleAttributes()　　设置句柄属性（内核句柄）
- ObReferenceObjectByHandle()　　通过句柄引用对象（内核句柄）

[描　述]

系统进程 System 的句柄表，也是内核全局句柄表。内核驱动程序所打开的句柄都存储在这里。内核句柄最高位为 1。

==========

`CCHAR` ***KeNumberProcessors***`; //导出`

[定　义] wrk\wrk-v1.2\base\ntos\ke\kernldat.c

[初始化] wrk\wrk-v1.2\base\ntos\ke\newsysbg.c　[KiSystemStartup ()]

[引　用]

[描　述]

处理器数量

==========

`PKPRCB` **KiProcessorBlock**`[MAXIMUM_PROCESSORS];`

[定　义] wrk\wrk-v1.2\base\ntos\ke\kernldat.c

[初始化] wrk\wrk-v1.2\base\ntos\ex\obinit.c　[KiSystemStartup ()]

[引　用]

- KiSwapThread()　　线程调度
- KiSetAffinityThread()　　设置线程亲和性

[描　述]

系统核心数据结构，与线程调度密切相关。所有 KPRCB 的指针数组，往前拨偏移可以得到对应 KPCR。MAXIMUM_PROCESSORS 定义为 32。而实际数组的元素个数由上面 KeNumberProcessors 决定，每个处理器对应一个 KPRCB 结构体。

==========

`ULONG_PTR` **KiSystemSharedData** `= KI_USER_SHARED_DATA;`

[定　义] wrk\wrk-v1.2\base\ntos\ke\kernldat.c

[初始化] wrk\wrk-v1.2\base\ntos\ke\ kernldat.c

[引　用]

[描　述]

常量指针，指向 Ring0 与 Ring3 共享的一个页面地址，宏 KI_USER_SHARED_DATA 定义为：0xFFDF0000，即内核态地址，用户态地址为：0x7FFE0000。该页面存储的数据结构是：KUSER_SHARED_DATA。

==========

`PVOID` **KeUserExceptionDispatcher**`;`

[定　义] wrk\wrk-v1.2\base\ntos\ke\kernldat.c

[初始化] wrk\wrk-v1.2\base\ntos\ps\kulookup.c　[PspLookupKernelUserEntryPoints ()]

[引　用]

- KiDispatchException()　　异常分发

[描　述]

函数指针，指向了用户模式异常处理入口：位于 ntdll.dll 中的 KiUserExceptionDispatcher 函数。用于系统在进行异常分发过程中向用户态分发异常。

To be update...

轩辕之风 http://www.cnblogs.com/xuanyuan/