**2021/05/10_Windows32位内核_第13课_使用未公开API枚举进程、驱动模块以及WRK源码分析、调试**

| | |
|---|---|
| **笔记本:** | Windows32位内核 |
| **创建时间:** | 2021/5/10 星期一 15:10 |
| **作者:** | ileemi |



# 课前会议

EPROCESS EX 执行层
KPROCESS KE 核心层

两个结构体的成员有重复性，首地址一致，执行层和核心层对其的解释不同。

遍历进程 0x88（偏移）：找遍历进程相关（系统相关）的未公开API，尽量不使用偏移。
ZwQuerySystemInformation -- 可以查询操作系统的大部分信息。
ZwQueryProcessInfomation -- 查询进程相关的信息

病毒分析
内核（安全软件，外挂保护）
竞品分析（协议分析）
渗透

有壳软件：看结果，分析老版本程序（可能没壳）
没壳软件：入口点分析

打造自己的驱动工具（软件不商业，可关闭驱动签名）。
PCHunter -- 看雪有学长的实战记录

# WRK 源码分析

更好的研究内核，有Windows源码可以更好的进行内核分析。早期的Windows2000源码有部分泄露。Windows10（微软官方公开一部分，比较权威）源码：WRK。

WRK：WRK的全称是“Windows Research Kernel”，它是微软为高校操作系统课程提供的可修改和跟踪的操作系统教学平台。它给出了Windows这个成功的商业操作系统的内核大部分代码，可以对其进行修改、编译，并且可以用这个内核启动Windows操作系统。可让学生将操作系统基本原理和商业操作系统内核联系起来，进一步加深对操作系统整体的理解。

- WRK是建立在真实的NT内核基础上的，实现了线程调度、内存管理、I/O管理、文件系统等操作系统所必须的组成部分。
- 可以将编译出的内核放到装有Windows 2003的机器上，通过增加启动项，指定从WRK内核启动。通过修改编译时的选项，可以支持X86和AMD64两种架构。
- 源码编译后是一个类似 ntkrxxx.exe 的可执行文件，可替换操作系统的的内核，对其进行调试分析。

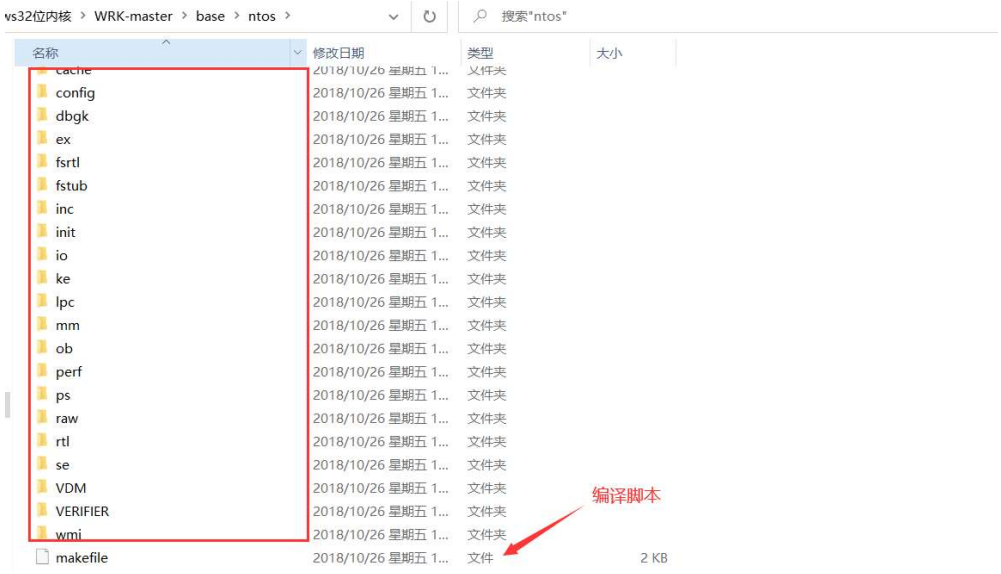
WRK包含了以下模块：

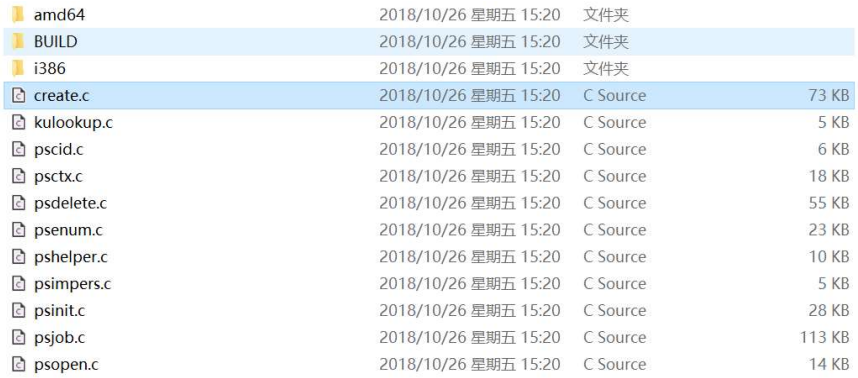
- Processes
- Threads
- Virtual memory and cache managers
- I/O management
- The registry
- Executive functions, such as the kernel heap and synchronization
- Object manager
- Local procedure call mechanism
- Security reference monitor
- Low-level CPU management

以v1.2版本为例，进行分析：

- 文件目录：


- 以各种类型名命名的API源码分类以及编译脚本：






- 内核相关的映像抽象层文件，负责硬件的通讯（区分单、多核）：

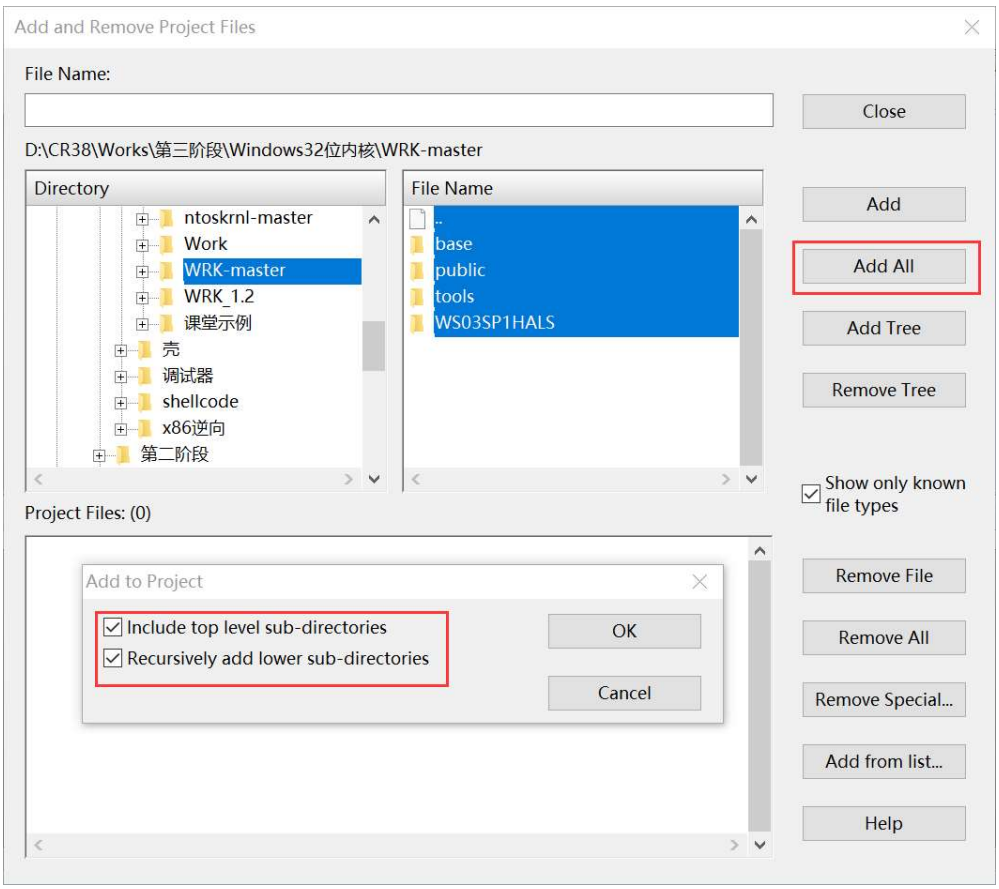

> KeGetPreviousMode -- 判断运行模式（当前函数从Ring3调用还是Ring0调用）。

# Source Insight

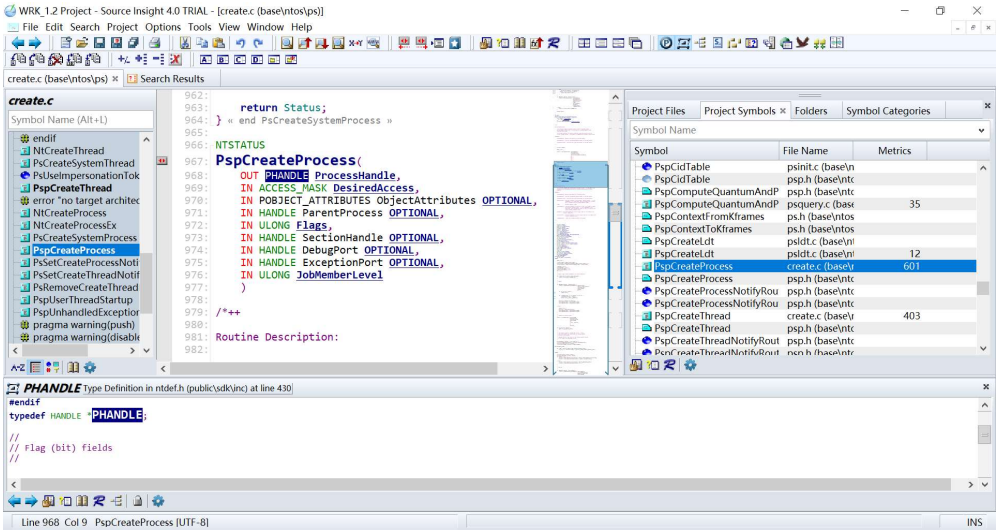
> 方便查看WRK源码中函数的定义以及实现，可以使用源码浏览工具（Source Insight），会对符号进行重建。

Source Insight的使用：

- 新建工程 --> 选择源码路径，会生成一个对应的符号文件。



- 查看对应函数的定义，以及使用：




- "Ctrl+="：跳转到定义，"Ctrl+/"：进行查找引用分析。

# 使用微软未公开函数

使用未公开函数遍历进程：ZwQuerySystemInformation。Zw... 开头的函数最终都会调用 Nt... 开头的函数，在Source Insight中查询不到ZwQuerySystemInformation的实现，所以可查询NtQuerySystemInformation函数的实现。

_SYSTEM_INFORMATION_CLASS 结构体：

```
typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemBasicInformation,
    SystemProcessorInformation,             // obsolete...delete
    SystemPerformanceInformation,
    SystemTimeOfDayInformation,
    SystemPathInformation,
    SystemProcessInformation,
    SystemCallCountInformation,
    SystemDeviceInformation,
    SystemProcessorPerformanceInformation,
    SystemFlagsInformation,
    SystemCallTimeInformation,
    SystemModuleInformation,
    SystemLocksInformation,
    SystemStackTraceInformation,
    SystemPagedPoolInformation,
    SystemNonPagedPoolInformation,
    SystemHandleInformation,
    SystemObjectInformation,
    SystemPageFileInformation,
```

...

> 使用未公开函数操作系统的兼容行更好（不在需要使用硬编码）

```
case ProcessTimes:

    if ( ProcessInformationLength != (ULONG) sizeof(KERNEL_USER_TIMES) ) {
        return STATUS_INFO_LENGTH_MISMATCH;
    }

    st = ObReferenceObjectByHandle (ProcessHandle,
                                    PROCESS_QUERY_INFORMATION,
                                    PsProcessType,
                                    PreviousMode,
                                    &Process,
                                    NULL);

    if (!NT_SUCCESS (st)) {
        return st;
    }

    //
    // Query the process kernel and user runtime.
    //

    TotalKernel = KeQueryRuntimeProcess (&Process->Pcb, &TotalUser);
    SysUserTime.KernelTime.QuadPart = UInt32x32To64(TotalKernel,
                                                    KeMaximumIncrement);

    SysUserTime.UserTime.QuadPart = UInt32x32To64(TotalUser,
                                                  KeMaximumIncrement);

    SysUserTime.CreateTime = Process->CreateTime;
    SysUserTime.ExitTime = Process->ExitTime;
    ObDereferenceObject (Process);
```

不在需要硬编码访问偏移，方便兼容操作系统版本

## 枚举进程

> 对 ZwQuerySystemInformation 函数查找引用，查看操作系统是如何遍历进程的：

```
---- ZwQuerySystemInformation Matches (6 in 4 files) ----
PerfInfoProcessRunDown in hooks.c (base\ntos\perf) :    Status = ZwQuerySystemInformation( SystemExtendedProcessInformation,
PerfInfoSysModuleRunDown in hooks.c (base\ntos\perf) :    Status = ZwQuerySystemInformation( SystemModuleInformation,
RtlpCreateStack in rtlexec.c (base\ntos\rtl) :    Status = ZwQuerySystemInformation( SystemBasicInformation,
RtlSystemTimeToLocalTime in time.c (base\ntos\rtl) :    Status = ZwQuerySystemInformation(
RtlLocalTimeToSystemTime in time.c (base\ntos\rtl) :    Status = ZwQuerySystemInformation(
zwapi.h (public\internal\base\inc) line 526 : ZwQuerySystemInformation (
NTSTATUS
PerfInfoProcessRunDown (
    )
/*++

Routine Description:

    This routine does the Process and thread rundown in the kernel mode.
    Since this routine is called only by global logger (i.e., trace from boot),
    no Sid info is collected.

Arguments:

    None.

Return Value:

    Status

--*/
{
    NTSTATUS Status;
    PSYSTEM_PROCESS_INFORMATION ProcessInfo;
    PSYSTEM_EXTENDED_THREAD_INFORMATION ThreadInfo;
    PCHAR Buffer;
    ULONG BufferSize = 4096;
    ULONG ReturnLength;

retry:
    Buffer = ExAllocatePoolWithTag(NonPagedPool, BufferSize, PERFPOOLTAG);

    if (!Buffer) {
        return STATUS_NO_MEMORY;
    }
    Status = ZwQuerySystemInformation( SystemExtendedProcessInformation,
                                       Buffer,
                                       BufferSize,
                                       &ReturnLength
                                       );
```

遍历进程线程相关信息，所以这段代码具有参考性

所需结构体，代码示例：

```
#include <ntddk.h>

typedef enum _SYSTEM_INFORMATION_CLASS {
  SystemBasicInformation,
  SystemProcessorInformation,             // obsolete...delete
  SystemPerformanceInformation,
  SystemTimeOfDayInformation,
  SystemPathInformation,
```

```
    SystemProcessInformation,
    SystemCallCountInformation,
    SystemDeviceInformation,
    SystemProcessorPerformanceInformation,
    SystemFlagsInformation,
    SystemCallTimeInformation,
    SystemModuleInformation,
    SystemLocksInformation,
    SystemStackTraceInformation,
    SystemPagedPoolInformation,
    SystemNonPagedPoolInformation,
    SystemHandleInformation,
    SystemObjectInformation,
    SystemPageFileInformation,
    SystemVdmInstemulInformation,
    SystemVdmBopInformation,
    SystemFileCacheInformation,
    SystemPoolTagInformation,
    SystemInterruptInformation,
    SystemDpcBehaviorInformation,
    SystemFullMemoryInformation,
    SystemLoadGdiDriverInformation,
    SystemUnloadGdiDriverInformation,
    SystemTimeAdjustmentInformation,
    SystemSummaryMemoryInformation,
    SystemMirrorMemoryInformation,
    SystemPerformanceTraceInformation,
    SystemObsolete0,
    SystemExceptionInformation,
    SystemCrashDumpStateInformation,
    SystemKernelDebuggerInformation,
    SystemContextSwitchInformation,
    SystemRegistryQuotaInformation,
    SystemExtendServiceTableInformation,
    SystemPrioritySeperation,
    SystemVerifierAddDriverInformation,
    SystemVerifierRemoveDriverInformation,
    SystemProcessorIdleInformation,
    SystemLegacyDriverInformation,
    SystemCurrentTimeZoneInformation,
    SystemLookasideInformation,
    SystemTimeSlipNotification,
    SystemSessionCreate,
    SystemSessionDetach,
    SystemSessionInformation,
    SystemRangeStartInformation,
    SystemVerifierInformation,
```

```
    SystemVerifierThunkExtend,
    SystemSessionProcessInformation,
    SystemLoadGdiDriverInSystemSpace,
    SystemNumaProcessorMap,
    SystemPrefetcherInformation,
    SystemExtendedProcessInformation,
    SystemRecommendedSharedDataAlignment,
    SystemComPlusPackage,
    SystemNumaAvailableMemory,
    SystemProcessorPowerInformation,
    SystemEmulationBasicInformation,
    SystemEmulationProcessorInformation,
    SystemExtendedHandleInformation,
    SystemLostDelayedWriteInformation,
    SystemBigPoolInformation,
    SystemSessionPoolTagInformation,
    SystemSessionMappedViewInformation,
    SystemHotpatchInformation,
    SystemObjectSecurityMode,
    SystemWatchdogTimerHandler,
    SystemWatchdogTimerInformation,
    SystemLogicalProcessorInformation,
    SystemWow64SharedInformation,
    SystemRegisterFirmwareTableInformationHandler,
    SystemFirmwareTableInformation,
    SystemModuleInformationEx,
    SystemVerifierTriageInformation,
    SystemSuperfetchInformation,
    SystemMemoryListInformation,
    SystemFileCacheInformationEx,
    MaxSystemInfoClass  // MaxSystemInfoClass should always be the last
enum
} SYSTEM_INFORMATION_CLASS;

// 进程
typedef struct _SYSTEM_PROCESS_INFORMATION {
    ULONG NextEntryOffset;
    ULONG NumberOfThreads;
    LARGE_INTEGER SpareLi1;
    LARGE_INTEGER SpareLi2;
    LARGE_INTEGER SpareLi3;
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    UNICODE_STRING ImageName;
    KPRIORITY BasePriority;
    HANDLE UniqueProcessId;
```

```
    HANDLE InheritedFromUniqueProcessId;
    ULONG HandleCount;
    ULONG SessionId;
    ULONG_PTR PageDirectoryBase;
    SIZE_T PeakVirtualSize;
    SIZE_T VirtualSize;
    ULONG PageFaultCount;
    SIZE_T PeakWorkingSetSize;
    SIZE_T WorkingSetSize;
    SIZE_T QuotaPeakPagedPoolUsage;
    SIZE_T QuotaPagedPoolUsage;
    SIZE_T QuotaPeakNonPagedPoolUsage;
    SIZE_T QuotaNonPagedPoolUsage;
    SIZE_T PagefileUsage;
    SIZE_T PeakPagefileUsage;
    SIZE_T PrivatePageCount;
    LARGE_INTEGER ReadOperationCount;
    LARGE_INTEGER WriteOperationCount;
    LARGE_INTEGER OtherOperationCount;
    LARGE_INTEGER ReadTransferCount;
    LARGE_INTEGER WriteTransferCount;
    LARGE_INTEGER OtherTransferCount;
} SYSTEM_PROCESS_INFORMATION, * PSYSTEM_PROCESS_INFORMATION;

// 枚举驱动模块
typedef struct _RTL_PROCESS_MODULE_INFORMATION {
    HANDLE Section;         // Not filled in
    PVOID MappedBase;
    PVOID ImageBase;
    ULONG ImageSize;
    ULONG Flags;
    USHORT LoadOrderIndex;
    USHORT InitOrderIndex;
    USHORT LoadCount;
    USHORT OffsetToFileName;
    UCHAR  FullPathName[256];
} RTL_PROCESS_MODULE_INFORMATION, * PRTL_PROCESS_MODULE_INFORMATION;

typedef struct _RTL_PROCESS_MODULES {
    ULONG NumberOfModules; // 驱动模块数量
    RTL_PROCESS_MODULE_INFORMATION Modules[1];
} RTL_PROCESS_MODULES, * PRTL_PROCESS_MODULES;
```

> 枚举进程信息，代码示例：

```
// 函数声明(使用未公开函数)
NTSTATUS NTAPI ZwQuerySystemInformation(
  __in SYSTEM_INFORMATION_CLASS SystemInformationClass,
  __out_bcount_opt(SystemInformationLength) PVOID SystemInformation,
  __in ULONG SystemInformationLength,
  __out_opt PULONG ReturnLength
);

NTSTATUS EnumProcess()
{
  NTSTATUS Status;
  PSYSTEM_PROCESS_INFORMATION ProcessInfo;
  PCHAR Buffer;
  ULONG BufferSize = 4096;
  ULONG ReturnLength;

retry:
  Buffer = ExAllocatePoolWithTag(NonPagedPool, BufferSize, '1234');

  if (!Buffer) {
    return STATUS_NO_MEMORY;
  }
  Status = ZwQuerySystemInformation(SystemExtendedProcessInformation,
    Buffer,
    BufferSize,
    &ReturnLength
  );

  if (Status == STATUS_INFO_LENGTH_MISMATCH) {
    ExFreePool(Buffer);
    BufferSize = ReturnLength;
    goto retry;
  }

  if (NT_SUCCESS(Status)) {
    ULONG TotalOffset = 0;
    ProcessInfo = (PSYSTEM_PROCESS_INFORMATION)Buffer;
    while (TRUE) {
      DbgPrint("[51asm] UniqueProcessId:%d PageDirectoryBase:%p
ImageName:%wZ\n",
        ProcessInfo->UniqueProcessId,
        ProcessInfo->PageDirectoryBase,
        &ProcessInfo->ImageName);

      if (ProcessInfo->NextEntryOffset == 0) {
```

```
        break;
      }
      else {
        TotalOffset += ProcessInfo->NextEntryOffset;
        ProcessInfo = (PSYSTEM_PROCESS_INFORMATION)&Buffer[TotalOffset];
      }
    }
  }
  ExFreePool(Buffer);
  return Status;
}
```

## 枚举操作系统驱动模块

> 和枚举进程基本类似，修改拆需要查询的信息即可：
> SystemModuleInformation

> 枚举进程信息，代码示例：

```
NTSTATUS EnumKernelModule()
{
  NTSTATUS Status;
  PRTL_PROCESS_MODULES ModuleInfo;
  PCHAR Buffer;
  ULONG BufferSize = 4096;
  ULONG ReturnLength;

retry:
  Buffer = ExAllocatePoolWithTag(NonPagedPool, BufferSize, '1234');

  if (!Buffer) {
    return STATUS_NO_MEMORY;
  }
  Status = ZwQuerySystemInformation(SystemModuleInformation,
    Buffer,
    BufferSize,
    &ReturnLength
  );

  if (Status == STATUS_INFO_LENGTH_MISMATCH) {
    ExFreePool(Buffer);
    BufferSize = ReturnLength;
    goto retry;
  }
```

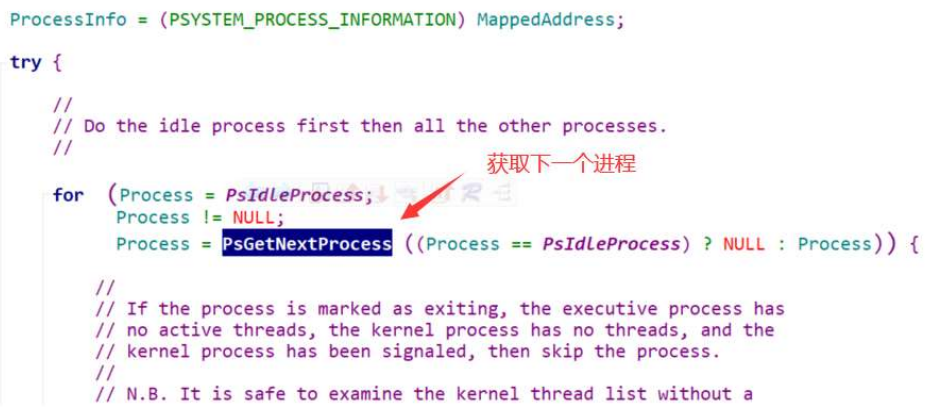
```
    if (NT_SUCCESS(Status)) {
      ModuleInfo = (PRTL_PROCESS_MODULES)Buffer;
      for (ULONG i = 0; i < ModuleInfo->NumberOfModules; i++) {
        DbgPrint("[51asm][%d] ImageBase:%p ImageSize:%p
FullPathName:%s\n",
          i,
          ModuleInfo->Modules[i].ImageBase,
          ModuleInfo->Modules[i].ImageSize,
          ModuleInfo->Modules[i].FullPathName);
      }
    }

    ExFreePool(Buffer);
    return Status;
}
```

## 操作系统内核中API的实现

> 操作系统获取进程相关信息：


ExpGetProcessInformation：



PsGetNextProcess：

```
for (ListEntry = (Process == NULL) ? PsActiveProcessHead.Flink : Process->ActiveProcessLinks.Flink;
     ListEntry != &PsActiveProcessHead;
     ListEntry = ListEntry->Flink) {

    NewProcess = CONTAINING_RECORD (ListEntry, EPROCESS, ActiveProcessLinks);

    //
    // Processes are removed from this list during process objected deletion (object reference count goes
    // to zero). To prevent double deletion of the process we need to do a safe reference here.
    //
    if (ObReferenceObjectSafe (NewProcess)) {
        break;
    }
    NewProcess = NULL;
}
PspUnlockProcessList (CurrentThread);

if (Process != NULL) {
    ObDereferenceObject (Process);
}
```

解决操作系统版本问题

> 操作系统获取模块相关信息：

```
        case SystemModuleInformation:
进入临界区  KeEnterCriticalRegion();
加载资源锁  ExAcquireResourceExclusiveLite( &PsLoadedModuleResource, TRUE );
            try {
                Status = ExpQueryModuleInformation( &PsLoadedModuleList,
                                                    &MmLoadedUserImageList,
                                                    (PRTL_PROCESS_MODULES)SystemInformation,
                                                    SystemInformationLength,
                                                    ReturnLength
                                                  );
            } except(EXCEPTION_EXECUTE_HANDLER) {
                Status = GetExceptionCode();
            }
释放资源锁  ExReleaseResourceLite (&PsLoadedModuleResource);
退出临界区  KeLeaveCriticalRegion();
            break;
```

遍历操作系统加载的所有模块信息

防止同步

资源锁在获取不到同步对象时，不会让出线程时间片（不需要切回，会独占CPU，锁线程）。

> 获取进程信息：
> ZwQueryInformationProcess --> NtQueryInformationProcess（有相关的实现代码）

```
NTSTATUS
NtQueryInformationProcess(
    __in HANDLE ProcessHandle,
    __in PROCESSINFOCLASS ProcessInformationClass,
    __out_bcount(ProcessInformationLength) PVOID ProcessInformation,
    __in ULONG ProcessInformationLength,
    __out_opt PULONG ReturnLength
    )
{
    PEPROCESS Process;
    KPROCESSOR_MODE PreviousMode;
    NTSTATUS st;
    PROCESS_BASIC_INFORMATION BasicInfo;
    VM_COUNTERS_EX VmCounters;
    IO_COUNTERS IoCounters;
    KERNEL_USER_TIMES SysUserTime;
    HANDLE DebugPort;
    ULONG ExecuteOptions;
    ULONG HandleCount;
    ULONG DefaultHardErrorMode;
    ULONG DisableBoost;
    ULONG BreakOnTerminationEnabled;
    PPROCESS_DEVICEMAP_INFORMATION DeviceMapInfo;
    PROCESS_SESSION_INFORMATION SessionInfo;
    PROCESS_PRIORITY_CLASS PriorityClass;
    ULONG_PTR Wow64Info;
    ULONG Flags;
    PUNICODE_STRING pTempNameInfo;
    SIZE_T MinimumWorkingSetSize;
    SIZE_T MaximumWorkingSetSize;
    ULONG HardEnforcement;
    KAPC_STATE ApcState;
    ULONG TotalKernel;
    ULONG TotalUser;
    KPROCESS_VALUES Values;

    PAGED_CODE();

    //
    // Get previous processor mode and probe output argument if necessary.
    //
```

# WRK源码的编译

1. 设置临时环境遍历（cmd path命令）：path D:\WRK-master\tools\x86;%path%


2. cmd 进入到源码目录：base\nots，进行编译：`nmake -nologo x86=` ，nologo 不提供警告信息，编译x86的内核可执行程序。编译成功后，对应的文件在nots\build\exe 目录下。
3. 将编译的可执行文件拷贝到被调试的操作系统中（这里以Winxp为例）对该操作系统的ntkrxxx.exe进行替换（ntkrxxx.exe文件不能删除）。通过在Boot.ini文件中添加编译选项即可。

ReactOS 是开源、自由的 Windows NT 系列克隆操作系统，保持了与 Windows 的系统级兼容性。

OpenGrok：一款大型的代码阅读工具。

# EPROCESS、KPROCESS、ETHREAD、KTHREAD 结构体的解析

> EPROCESS：

- EX_PUSH_LOCK ProcessLock;　　//EPROCESS成员保护锁(推锁)
  操作系统访问 EPROCESS 中的任意成员时，都会用到推锁（ProcessLock），所以在修改 EPROCESS 结构体中的成员时，需要先获取推锁，修改完数据后，释放推锁（防止线程同步）。
- PVOID DebugPort;　　//调试端口
  反调试：产生调试事件，会通过调试端口（DebugPort）将调试事件发送给调试器。调试端口为0，无法发送调试事件。调试器也就接收不到调试事件（这样软件就可以做到反调试）。
  反反调试：在 DebugPort 地址上下一个内存访问断点（> ba xxx）。ba命令设置一个数据断点，访问指定内存时会触发此断点。bm：符号断点；bp：地址断点；bu：延时断点。
  示例：dt _EPROCESS xxx(PROCESS_ADDR)
  dd xxx(PROCESS_ADDR) + 0xbc
  ed xxx(PROCESS_ADDR) + 0xbc 0
- EX_FAST_REF Token;　　//令牌
  影响到应用程序的权限，可将普通进程修改为系统进程（权限也将提升）。

> KPROCESS：比EPROCESS处理的事情少

> KTHREAD：

- KAPC_STATE ApcState;　　//指向APC信息
  异步过程调用（内部保存的是函数指针），线程调用，分配到时间片时会先调用这里。

- ULONG ContextSwitches;　　//线程进行了多少次环境切换

# 微软Hook库

> detours：支持32位、64位的内联Hook。64位不能内联汇编（可联合编译，编写完成的汇编函数）。

# Windows版本的判断

可对指定的结构内存进行扫描，找特征（定位成员偏移）。