DbgPrint 函数流程分析

前言

Windows 下编写内核驱动时经常用到 DbgPrint 函数输出一些调试信息，用来辅助调试。当正在用 WinDbg 内核调试时，调试信息会输出到 WinDbg 中。或者利用一些辅助工具也能看到输出的调试信息，比如 Sysinternals 公司的 DebugView 工具。本文分析了 Vista 系统上 DbgPrint 系列函数的执行流程，并揭示了 DebugView 工具的实现原理。

DbgPrint函数流程

先看一下 WDK 中 DbgPrint 函数的原型：

```
ULONG
DbgPrint (
    IN PCHAR Format,
    ...
    );
```

和 printf 的参数一样，可以格式化字符串。

```
.text:0049E123 ; ULONG DbgPrint(PCH Format,...)
.text:0049E123              public _DbgPrint
.text:0049E123 _DbgPrint      proc near             ; CODE XREF: sub_4046B2+11↑p
.text:0049E123
.text:0049E123 Format         = dword ptr  8
.text:0049E123 arglist        = byte ptr  0Ch
.text:0049E123
.text:0049E123              mov     edi, edi
.text:0049E125              push    ebp
.text:0049E126              mov     ebp, esp
.text:0049E128              push    TRUE
.text:0049E12A              lea     eax, [ebp+arglist]
.text:0049E12D              push    eax
.text:0049E12E              push    [ebp+Format]
.text:0049E131              mov     ecx, offset ??_C@_00CNPNBAHC@?$AA@FNODOBFM@
.text:0049E136              push    3             ; DPFLTR_INFO_LEVEL = 3
.text:0049E138              push    65h           ; DPFLTR_DEFAULT_ID = 101 = 0x65
.text:0049E13A              call    vDbgPrintExWithPrefixInternal(x,x,x,x,x,x)
.text:0049E13F              pop     ebp
.text:0049E140              retn
.text:0049E140 _DbgPrint      endp
```

从反汇编代码来看，DbgPrint 函数很简单，传递参数直接调用 vDbgPrintExWithPrefixInternal 函数。传递的 ComponentId 为 DPFLTR_DEFAULT_ID，Level 为 DPFLTR_INFO_LEVEL。查看 DbgPrintEx 函数的文档可以知道这两个参数的意义。

```
.text:0046EBF4 __stdcall vDbgPrintExWithPrefixInternal(x, x, x, x, x, x) proc near
.text:0046EBF4                                         ; CODE XREF: _DbgPrintEx+19↑p
.text:0046EC11
.text:0046EC17                 push    [ebp+ulLevel]
.text:0046EC1A                 push    [ebp+ulComponentId]
.text:0046EC1D                 call    NtQueryDebugFilterState(x,x)
.text:0046EC22                 test    eax, eax
.text:0046EC24                 jnz     short loc_46EC2D
.text:0046EC26
.text:0046EC26 loc_46EC26:
.text:0046EC26                 xor     eax, eax
.text:0046EC28                 jmp     _exit

.text:0046EBA8 __stdcall NtQueryDebugFilterState(x, x) proc near
.text:0046EBA8
.text:0046EBA8 ulComponentId   = dword ptr  8
.text:0046EBA8 ulLevel         = dword ptr  0Ch
.text:0046EBA8
.text:0046EBA8                 mov     edi, edi
.text:0046EBAA                 push    ebp
.text:0046EBAB                 mov     ebp, esp
.text:0046EBC0                 mov     ecx, [ebp+ulLevel]
.text:0046EBCC                 xor     eax, eax
.text:0046EBCE                 inc     eax
.text:0046EBCF                 shl     eax, cl
.text:0046EBD1                 test    _Kd_WIN2000_Mask, eax
.text:0046EBD7                 jnz     short loc_46EBE8
.text:0046EBD9                 mov     ecx, _KdComponentTable[edx*4]
.text:0046EBE0                 test    [ecx], eax
.text:0046EBE2                 jnz     short loc_46EBE8
.text:0046EBE4                 xor     eax, eax
.text:0046EBE6                 jmp     short loc_46EBEB
```

vDbgPrintExWithPrefixInternal 函数首先调用 NtQueryDebugFilterState 函数检查 ComponentId 和 Level 值判断当前输出是否需要屏蔽。DbgPrint 传入的值分别是 65h 和 3， 65h 定为的 nt!Kd_DEFAULT_Mask 的值和 3 被移位后的 8 比较，从而确定此次输出是否需要屏蔽。所以 Vista 系统上用 WinDbg 内核调试时缺省看不到 DbgPrint 输出的调试字符串，可以用 ed nt!Kd_DEFAULT_Mask 0x8 命令或者修改注册表打开 DbgPrint 调试输出。

```
.text:0046EC5D                push    [ebp+ntStatus2] ; cbDest
.text:0046EC63                push    [ebp+pszDest]   ; pszDest
.text:0046EC69                mov     ecx, 512
.text:0046EC6E                sub     ecx, esi
.text:0046EC70                lea     edi, [ebp+esi+szBuffer]
.text:0046EC77                call    RtlStringCbVPrintfA(x,x,x,x)

.text:0046ECB2                lea     ecx, [ebp+szBuffer]
.text:0046ECB8                mov     [ebp+asBuffer.Buffer], ecx
.text:0046ECBE                mov     [ebp+asBuffer.Length], ax
.text:0046ECC5                cmp     _KeBugCheckActive, 0
.text:0046ECCC                jnz     short loc_46ED09
.text:0046ECCC
.text:0046ECCE                mov     esi, _RtlpDebugPrintCallback
.text:0046ECD4                test    esi, esi
.text:0046ECD6                jz      short loc_46ED09
.text:0046ECD6
.text:0046ECD8                call    ds:KeGetCurrentIrql()
.text:0046ECDE                mov     bl, al
.text:0046ECE0                cmp     bl, PROFILE_LEVEL
.text:0046ECE3                jnb     short loc_46ECED
.text:0046ECE3
.text:0046ECE5                mov     cl, PROFILE_LEVEL
.text:0046ECE7                call    ds:KfRaiseIrql(x)
.text:0046ECED
.text:0046ECED loc_46ECED:
.text:0046ECED                push    [ebp+ulLevel]
.text:0046ECF0                push    [ebp+ulComponentId]
.text:0046ECF3                lea     eax, [ebp+asBuffer]
.text:0046ECF9                push    eax
.text:0046ECFA                call    esi
; DbgPrintCallback(PANSI_STRING pasBuffer, ULONG ulComponentId, ULONG ulLevel);
.text:0046ECFC                cmp     bl, PROFILE_LEVEL
.text:0046ECFF                jnb     short loc_46ED09
.text:0046ECFF
.text:0046ED01                mov     cl, bl
.text:0046ED03                call    ds:KfLowerIrql(x)
```

如果此次输出不需要屏蔽，vDbgPrintExWithPrefixInternal 继续执行。调用 RtlStringCbVPrintfA 函数格式化字符串，再判断是否正在蓝屏过程中，然后提高 IRQL 调用输出回调函数。调试输出回调是 Vista 的新增功能，XP 中没有见到。NTSTATUS DbgSetDebugPrintCallback(PDBGPRINT_CALLBACE pDbgCallback, BOOLEAN bSet)函数用来设置和取消回调函数，只能设置一个函数，函数地址保存在 RtlpDebugPrintCallback 内部变量中。回调函数返回后降低 IRQL。

```
.text:0046ED09              movzx   eax, [ebp+asBuffer.Length]
.text:0046ED10              mov     [ebp+pszDest], eax
.text:0046ED16              mov     eax, [ebp+asBuffer.Buffer]
.text:0046ED1C              mov     [ebp+var_234], eax
.text:0046ED22              push    edi
.text:0046ED23              push    ebx
.text:0046ED24              mov     eax, 1          ; eax = BREAKPOINT_PRINT
.text:0046ED29              mov     ecx, [ebp+var_234] ; ecx = pszBuffer
.text:0046ED2F              mov     edx, [ebp+pszDest] ; edx = ulBufLength
.text:0046ED35              mov     ebx, [ebp+ulComponentId]
.text:0046ED38              mov     edi, [ebp+ulLevel]
.text:0046ED3B              int     2Dh             ; Internal routine for MSDOS (IRET)
.text:0046ED3D              int     3               ; Trap to Debugger
```

vDbgPrintExWithPrefixInternal 函数接着执行，通过 int2d 调用调试服务把字符串输出到调试器。int2d 的服务例程为 KiDebugService 函数。

```
.text:0044737C _KiDebugService:                     ; DATA XREF: INIT:0070A35C↓o
.text:004473EA              inc     dword ptr [ebp+68h]
.text:004473EA              inc     dword ptr [ebp+68h] ; [_KTRAP_FRAME].Eip
.text:004473ED              mov     eax, [ebp+44h]  ; [_KTRAP_FRAME].Eax
.text:004473F0              mov     ecx, [ebp+40h]  ; [_KTRAP_FRAME].Ecx
.text:004473F3              mov     edx, [ebp+3Ch]  ; [_KTRAP_FRAME].Edx
.text:004473F6              jmp     loc_447527

.text:00447527
.text:00447527 loc_447527:                          ; CODE XREF: .text:004473F6↑j
.text:00447527
.text:00447543              mov     esi, ecx
.text:00447545              mov     edi, edx
.text:00447547              mov     edx, eax
.text:00447549              mov     ebx, [ebp+68h] ; [_KTRAP_FRAME].Eip
.text:0044754C              dec     ebx
.text:0044754D              mov     ecx, 3
.text:00447552              mov     eax, STATUS_BREAKPOINT
.text:00447557              call    CommonDispatchException
```

```
.text:00446D70 CommonDispatchException proc near
.text:00446D70
.text:00446D70 stExceptionRecord= EXCEPTION_RECORD ptr -50h
.text:00446D70
.text:00446D70              sub     esp, 50h
.text:00446D73              mov     [esp+50h+stExceptionRecord.ExceptionCode], eax
.text:00446D76              xor     eax, eax
.text:00446D78              mov     [esp+50h+stExceptionRecord.ExceptionFlags], eax
.text:00446D7C              mov     [esp+50h+stExceptionRecord.ExceptionRecord], eax
.text:00446D80              mov     [esp+50h+stExceptionRecord.ExceptionAddress], ebx
.text:00446D84              mov     [esp+50h+stExceptionRecord.NumberParameters], ecx
.text:00446D88              cmp     ecx, 0
```

```
.text:00446D8B              jz      short loc_446D99
.text:00446D8D              lea     ebx,
[esp+50h+stExceptionRecord.ExceptionInformation]
.text:00446D91              mov     [ebx], edx
.text:00446D93              mov     [ebx+4], esi
.text:00446D96              mov     [ebx+8], edi
.text:00446D99
.text:00446D99
.text:00446DA8              mov     eax, [ebp+6Ch] ; [_KTRAP_FRAME].SegCs
.text:00446DAB
.text:00446DAB loc_446DAB:                          ; CODE XREF: CommonDispatchException+36↑j
.text:00446DAB              and     eax, 1
.text:00446DAE              push    1               ; char
.text:00446DB0              push    eax             ; int
.text:00446DB1              push    ebp             ; BugCheckParameter3
.text:00446DB2              push    0               ; int
.text:00446DB4              push    ecx             ; void *
.text:00446DB5              call    KiDispatchException(x,x,x,x,x)
```

KiDebugService 先构建一个陷阱帧（KTRAP_FRAME），然后设置参数调用 CommonDispatchException，CommonDispatchException 会构建一个异常纪录（EXCEPTION_RECORD），然后调用 KiDispatchException 函数走异常处理流程，异常代码为 STATUS_BREAKPOINT。从 int2d->KiDebugService->CommonDispatchException->KiDispatchException 这个流程一路看下来，会发现 int2d 时提供的三个参数存放在异常纪录的 ExceptionInformation 数组里面，分别是：

ExceptionInformation[0]表示 BREAKPOINT_PRINT 功能号。

ExceptionInformation[1]表示调试信息字符串地址。

ExceptionInformation[2]表示调试信息字符串长度。

进入 KiDispatchException 后的代码比较复杂，当前只是分析 DbgPrint 的流程，其他代码暂时不管，只需要知道 KiDispatchException 会调用 KiDebugRoutine 把异常提交给内核调试引擎处理。当处于内核调试时，KiDebugRoutine 指向 KdpTrap 函数，没有调试时，KiDebugRoutine 指向 KdpStub 函数。先来看看 KdpStub 函数。

```
.text:0042A2DC __stdcall KdpStub(x, x, x, x, x, x) proc near
.text:0042A2DC
.text:0042A2DC TrapFrame      = dword ptr  8
.text:0042A2DC ExceptionFrame  = dword ptr  0Ch
.text:0042A2DC ExceptionRecord = dword ptr  10h
.text:0042A2DC ContextRecord   = dword ptr  14h
.text:0042A2DC PreviousMode    = dword ptr  18h
.text:0042A2DC bSecondChance   = dword ptr  1Ch
.text:0042A2DC
.text:0042A2DC              mov     edi, edi
.text:0042A2DE              push    ebp
.text:0042A2DF              mov     ebp, esp
.text:0042A2E1              push    ebx
.text:0042A2E2              push    esi
.text:0042A2E2
.text:0042A2E3              mov     esi, [ebp+ExceptionRecord]
.text:0042A2E6              xor     ebx, ebx
.text:0042A2E8              cmp     [esi+EXCEPTION_RECORD.ExceptionCode],
STATUS_BREAKPOINT
```

```
.text:0042A2EE                jnz     short _elseif
.text:0042A2EE
.text:0042A2F0                cmp     [esi+EXCEPTION_RECORD.NumberParameters], ebx
.text:0042A2F3                jbe     short _elseif
.text:0042A2F3
.text:0042A2F5                mov     eax, [esi+EXCEPTION_RECORD.ExceptionInformation]
.text:0042A2F8                cmp     eax, BREAKPOINT_LOAD_SYMBOLS
.text:0042A2FB                jz      short loc_42A30C
.text:0042A2FD                cmp     eax, BREAKPOINT_UNLOAD_SYMBOLS
.text:0042A300                jz      short loc_42A30C
.text:0042A302                cmp     eax, BREAKPOINT_COMMAND_STRING
.text:0042A305                jz      short loc_42A30C
.text:0042A307                cmp     eax, BREAKPOINT_PRINT
.text:0042A30A                jnz     short _elseif
.text:0042A30C
.text:0042A30C                mov     eax, [ebp+ContextRecord]
.text:0042A30F                inc     [eax+CONTEXT._Eip]
.text:0042A315                mov     al, 1           ; return TRUE;
.text:0042A317                jmp     short _exit
```

KdpStub 先判断异常代码是不是 STATUS_BREAKPOINT（int3 断点异常也是这个异常代码，但第一个参数是 BREAKPOINT_BREAK），然后判断参数个数。对于当前支持的四种调试服务，包括输出调试字符串，都是把 eip 加一，跳过 int2d 后面带的 int3 指令，然后从异常处理中返回，继续执行。

当正在调试时，KiDispatchException 调用的就是 KdpTrap 函数。

```
PAGEKD:006AB5EB __stdcall KdpTrap(x, x, x, x, x, x) proc near
PAGEKD:006AB6A1 loc_6AB6A1:                           ; CODE XREF: KdpTrap(x,x,x,x,x,x)+3A↑j
PAGEKD:006AB6A1                mov     edx, [ebx+CONTEXT._Ebx]
PAGEKD:006AB6A7                lea     ecx, [ebp+bReturn]
PAGEKD:006AB6AA                push    ecx
PAGEKD:006AB6AB                push    [ebp+ExceptionFrame]
PAGEKD:006AB6AE                movzx   ecx, word ptr [eax+1Ch] ; ExceptionInformation[2]
PAGEKD:006AB6B2                push    [ebp+TrapFrame]
PAGEKD:006AB6B5                push    dword ptr [ebp+PreviousMode]
PAGEKD:006AB6B8                push    ecx
PAGEKD:006AB6B9                push    dword ptr [eax+18h] ; ExceptionInformation[1]
PAGEKD:006AB6BC                mov     ecx, [ebx+CONTEXT._Edi]
PAGEKD:006AB6C2                call    KdpPrint(x,x,x,x,x,x,x,x)
```

KdpTrap 也会和 KdpStub 一样判断异常代码和参数个数，以及调试服务号，根据调试服务号的不同调用不同的处理函数。针对 BREAKPOINT_PRINT 输出调试信息的情况，调用的是 KdpPrint 函数。

KdpPrint 也会根据 ComponentId 和 Level 值判断一下是否需要屏蔽此次输出。然后判断特权模式，如果是用户模式还需要探测字符串内存，保证可读。

```
PAGEKD:006AC921              mov     [ebp+asBuffer.Buffer], edi
PAGEKD:006AC924              mov     [ebp+asBuffer.Length], bx
PAGEKD:006AC928              lea     eax, [ebp+asBuffer]
PAGEKD:006AC92B              push    eax
PAGEKD:006AC92C              call    KdpLogDbgPrint(x)
PAGEKD:006AC931              cmp     _KdDebuggerNotPresent, 0
PAGEKD:006AC938              jnz     short loc_6AC984
PAGEKD:006AC93A              push    [ebp+ExceptionFrame]
PAGEKD:006AC93D              push    [ebp+TrapFrame]
PAGEKD:006AC940              call    KdEnterDebugger(x,x)
PAGEKD:006AC945              mov     [ebp-20h], al
PAGEKD:006AC948              lea     eax, [ebp+asBuffer]
PAGEKD:006AC94B              call    KdpPrintString(x)
```

KdpPrint 接着调用 KdpLogDbgPrint 在一个循环缓冲区里记录调试字符串，然后判断是否挂接了调试器，调用 KdpPrintString 输出调试字符串。

KdpPrintString 构造一个调试包，通过 KdSendPacket 函数发送给调试器。

DebugView实现原理

上一节详细介绍了 DbgPrint 输出调试字符串的流程，现在来看看 DebugView 工具的实现原理。

在 Vista 系统上，DebugView 设置了调试输出回调函数，从而截获调试字符串。

```
kd> dps nt!RtlpDebugPrintCallback L 1
818f41b8  00000000
kd> g

ModLoad: 919ef000 919f2d00   Dbgv.sys

kd> dps nt!RtlpDebugPrintCallback L 1
818f41b8  919efa86 Dbgv+0xa86
```

在 Vista 以前的系统，比如 2003 系统上，DbgPrint 函数调用 vDbgPrintExWithPrefixInternal 函数，在 vDbgPrintExWithPrefixInternal 函数里面不是直接 int2d 调用调试服务，而是通过 DebugPrint 函数再调用调试服务把字符串输出。DebugView 通过 Hook 函数 DebugPrint 截获调试字符串。

```
kd> u nt!DebugPrint
nt!DebugPrint:
808356b6 8bff            mov     edi,edi
808356b8 55              push    ebp
808356b9 8bec            mov     ebp,esp
808356bb ff7510          push    dword ptr [ebp+10h]
808356be 8b4508          mov     eax,dword ptr [ebp+8]
808356c1 ff750c          push    dword ptr [ebp+0Ch]
808356c4 0fb708          movzx   ecx,word ptr [eax]
808356c7 51              push    ecx
808356c8 ff7004          push    dword ptr [eax+4]
808356cb 6a01            push    1
808356cd e86f460100      call    nt!DebugService (80849d41)
```

```
808356d2 5d              pop     ebp
808356d3 c20c00          ret     0Ch
kd> g

ModLoad: f6a46000 f6a49d00   Dbgv.sys

kd> u nt!DebugPrint
nt!DebugPrint:
808356b6 ff258c7da4f6    jmp     dword ptr [Dbgv+0x1d8c (f6a47d8c)]
808356bc 7510            jne     nt!DebugPrint+0x18 (808356ce)
808356be 8b4508          mov     eax,dword ptr [ebp+8]
808356c1 ff750c          push    dword ptr [ebp+0Ch]
808356c4 0fb708          movzx   ecx,word ptr [eax]
808356c7 51              push    ecx
808356c8 ff7004          push    dword ptr [eax+4]
808356cb 6a01            push    1
kd> dps 0xf6a47d8c L 1
f6a47d8c  f6a469ac Dbgv+0x9ac
```

==========

小喂

2007-12-01