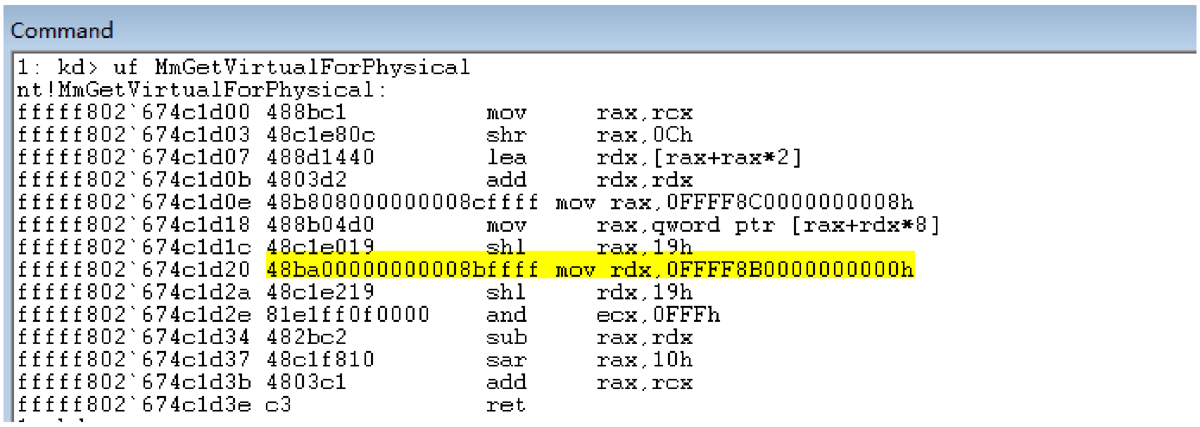
当今操作系统普遍采用64位架构，CPU最大寻址能力虽然达到了64位，但其实仅仅只是用到了48位进行寻址，其内存管理采用了`9-9-9-9-12`的分页模式，`9-9-9-9-12`分页表示物理地址拥有四级页表，微软将这四级依次命名为PXE、PPE、PDE、PTE这四项。

关于内存管理和分页模式，不同的操作系统和体系结构可能会有略微不同的实现方式。9-9-9-9-12的分页模式是一种常见的分页方案，其中物理地址被分成四级页表：PXE（Page Directory Pointer Table Entry）、PPE（Page Directory Entry）、PDE（Page Table Entry）和PTE（Page Table Entry）。这种分页模式可以支持大量的物理内存地址映射到虚拟内存地址空间中。每个级别的页表都负责将虚拟地址映射到更具体的物理地址。通过这种层次化的页表结构，操作系统可以更有效地管理和分配内存。

首先一个PTE管理1个分页大小的内存也就是`0x1000`字节，PTE结构的解析非常容易，打开WinDBG输入`!PTE 0`即可解析，如下所示，当前地址0位置处的PTE基址是`FFFF898000000000`，由于PTE的一个页大小是`0x1000`所以当内存地址高于`0x1000`时将会切换到另一个页中，如下`FFFF898000000008`则是另一个页中的地址。

```
0: kd> !PTE 0
                                           VA 0000000000000000
PXE at FFFF89C4E2713000    PPE at FFFF89C4E2600000    PDE at FFFF89C4C0000000
 PTE at FFFF898000000000
contains 8A0000000405F867  contains 0000000000000000
pfn 405f      ---DA--UW-V  not valid

0: kd> !PTE 0x1000
                                           VA 0000000000001000
PXE at FFFF89C4E2713000    PPE at FFFF89C4E2600000    PDE at FFFF89C4C0000000
 PTE at FFFF898000000008
contains 8A0000000405F867  contains 0000000000000000
pfn 405f      ---DA--UW-V  not valid
```

由于PTE是动态变化的，找到该地址的关键就在于通过`MmGetSystemRoutineAddress`函数动态得到`MmGetVirtualForPhysical`的内存地址，然后向下扫描特征寻找`mov rdx,0FFFF8B0000000000h`并将内部的地址提取出来。



```
1: kd> uf MmGetVirtualForPhysical
nt!MmGetVirtualForPhysical:
fffff802`674c1d00 488bc1          mov     rax,rcx
fffff802`674c1d03 48c1e80c        shr     rax,0Ch
fffff802`674c1d07 488d1440        lea     rdx,[rax+rax*2]
fffff802`674c1d0b 4803d2          add     rdx,rdx
fffff802`674c1d0e 48b80800000000008cffff mov rax,0FFFF8C0000000008h
fffff802`674c1d18 488b04d0        mov     rax,qword ptr [rax+rdx*8]
fffff802`674c1d1c 48c1e019        shl     rax,19h
fffff802`674c1d20 48ba0000000000008bffff mov rdx,0FFFF8B0000000000h
fffff802`674c1d2a 48c1e219        shl     rdx,19h
fffff802`674c1d2e 81e1ff0f0000    and     ecx,0FFFh
fffff802`674c1d34 482bc2          sub     rax,rdx
fffff802`674c1d37 48c1f810        sar     rax,10h
fffff802`674c1d3b 4803c1          add     rax,rcx
fffff802`674c1d3e c3              ret
```

这段代码完整版如下所示，代码可动态定位到PTE的内存地址，然后将其取出；

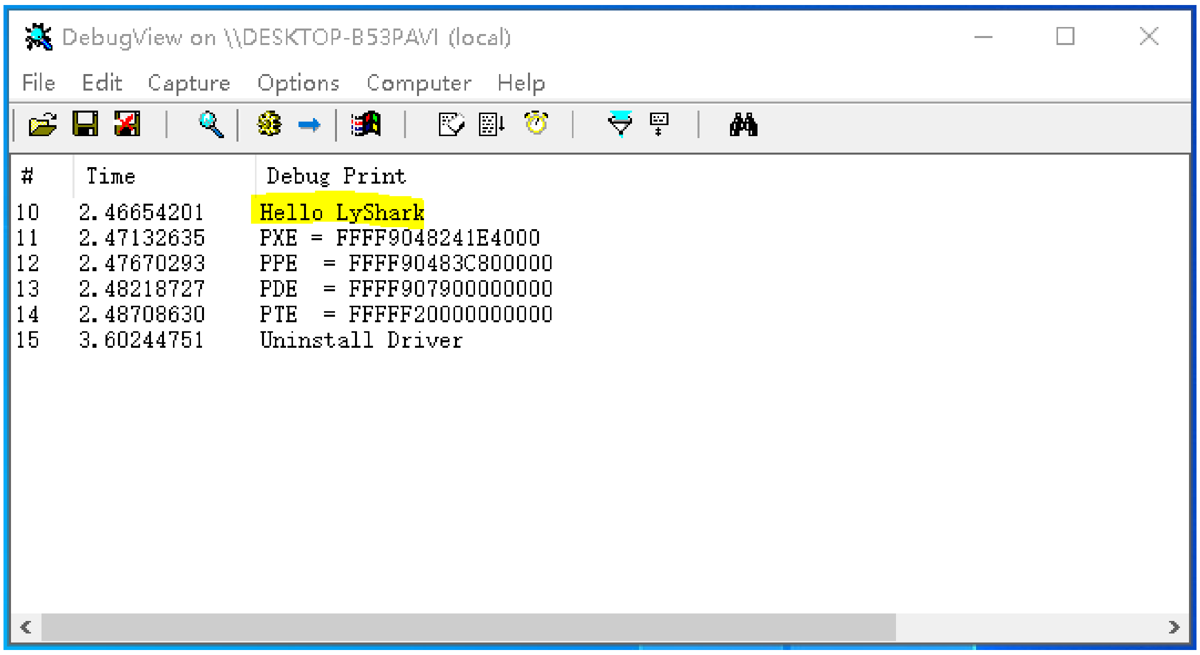
```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

#include <ntifs.h>
#include <ntstrsafe.h>
```

```
// 指定内存区域的特征码扫描
PVOID SearchMemory(PVOID pStartAddress, PVOID pEndAddress, PUCHAR pMemoryData,
ULONG ulMemoryDataSize)
{
    PVOID pAddress = NULL;
    PUCHAR i = NULL;
    ULONG m = 0;

    // 扫描内存
    for (i = (PUCHAR)pStartAddress; i < (PUCHAR)pEndAddress; i++)
    {
        // 判断特征码
        for (m = 0; m < ulMemoryDataSize; m++)
        {
            if (*(PUCHAR)(i + m) != pMemoryData[m])
            {
                break;
            }
        }
        // 判断是否找到符合特征码的地址
        if (m >= ulMemoryDataSize)
        {
            // 找到特征码位置, 获取紧接着特征码的下一地址
            pAddress = (PVOID)(i + ulMemoryDataSize);
            break;
        }
    }

    return pAddress;
}

// 获取到函数地址
PVOID GetMmGetVirtualForPhysical()
{
    PVOID VariableAddress = 0;
    UNICODE_STRING uioiTime = { 0 };

    RtlInitUnicodeString(&uioiTime, L"MmGetVirtualForPhysical");
    VariableAddress = (PVOID)MmGetSystemRoutineAddress(&uioiTime);
    if (VariableAddress != 0)
    {
        return VariableAddress;
    }
    return 0;
}

// 驱动卸载例程
VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("Uninstall Driver \n");
}

// 驱动入口地址
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark \n");

    // 获取函数地址
    PVOID address = GetMmGetVirtualForPhysical();

    DbgPrint("GetMmGetVirtualForPhysical = %p \n", address);

    UCHAR pSecondSpecialData[50] = { 0 };
    ULONG ulFirstSpecialDataSize = 0;

    pSecondSpecialData[0] = 0x48;
    pSecondSpecialData[1] = 0xc1;
    pSecondSpecialData[2] = 0xe0;
    ulFirstSpecialDataSize = 3;

    // 定位特征码
    PVOID PTE = SearchMemory(address, (PVOID)((PUCHAR)address + 0xFF),
pSecondSpecialData, ulFirstSpecialDataSize);
    __try
    {
        PVOID lOffset = (ULONG)PTE + 1;
        DbgPrint("PTE Address = %p \n", lOffset);
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        DbgPrint("error");
    }

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

运行如上代码可动态获取到当前系统的PTE地址，然后将PTE填入到`g_PTEBASE`中，即可实现解析系统内的四个标志位，完整解析代码如下所示；

```
// 署名权
// right to sign one's name on a piece of work
// PowerBy: LyShark
// Email: me@lyshark.com

#include <ntifs.h>
#include <ntstrsafe.h>

INT64 g_PTEBASE = 0;
INT64 g_PDEBASE = 0;
INT64 g_PPEBASE = 0;
INT64 g_PXEBASE = 0;

PULONG64 GetPteBase(PVOID va)
{
    return (PULONG64)((((ULONG64)va & 0xFFFFFFFFFFFF) >> 12) * 8) + g_PTEBASE;
}

PULONG64 GetPdeBase(PVOID va)
{
    return (PULONG64)((((ULONG64)va & 0xFFFFFFFFFFFF) >> 12) * 8) + g_PDEBASE;
}

PULONG64 GetPpeBase(PVOID va)
{
    return (PULONG64)((((ULONG64)va & 0xFFFFFFFFFFFF) >> 12) * 8) + g_PPEBASE;
}

PULONG64 GetPxeBase(PVOID va)
{
    return (PULONG64)((((ULONG64)va & 0xFFFFFFFFFFFF) >> 12) * 8) + g_PXEBASE;
}

// 驱动卸载例程
VOID UnDriver(PDRIVER_OBJECT driver)
{
    DbgPrint("Uninstall Driver \n");
}

// 驱动入口地址
NTSTATUS DriverEntry(IN PDRIVER_OBJECT Driver, PUNICODE_STRING RegistryPath)
{
    DbgPrint("Hello LyShark \n");

    g_PTEBASE = 0XFFFFF20000000000;

    g_PDEBASE = (ULONG64)GetPteBase((PVOID)g_PTEBASE);
    g_PPEBASE = (ULONG64)GetPteBase((PVOID)g_PDEBASE);
    g_PXEBASE = (ULONG64)GetPteBase((PVOID)g_PPEBASE);

    DbgPrint("PXE = %p \n", g_PXEBASE);
    DbgPrint("PPE  = %p \n", g_PPEBASE);
    DbgPrint("PDE  = %p \n", g_PDEBASE);
    DbgPrint("PTE  = %p \n", g_PTEBASE);

    Driver->DriverUnload = UnDriver;
    return STATUS_SUCCESS;
}
```

我的系统内PTE地址为`0XFFFFF20000000000`，填入变量内解析效果如下图所示；


DebugView on \\DESKTOP-B53PAVI (local)
File Edit Capture Options Computer Help

```
#   Time         Debug Print
10  2.46654201   Hello LyShark
11  2.47132635   PXE = FFFF9048241E4000
12  2.47670293   PPE  = FFFF90483C800000
13  2.48218727   PDE  = FFFF907900000000
14  2.48708630   PTE  = FFFFF20000000000
15  3.60244751   Uninstall Driver
```