初始化流程分析(不涉及内存管理)

通过代码调用流程进行分析，也可参考代码注释

DriverEntry(Newbp.c)

```
if (!NT_SUCCESS (Status = HvmInit ())) {
  _KdPrint (("NEWBLUEPILL: HvmInit() failed with status 0x%08hX\n", Status));
```

-调用 HvmInit()
由于 nbp 实现了两套体系结构，因此最初要对处理器属于那种体系结构加以判断，然后为 Hvm 赋予一套对应的函数指针。如果处理器不支持硬件虚拟技术则返回 STATUS_NOT_SUPPORTED。
函数最后初始化了一个互斥锁，将来用于对不能重入的代码进行保护。

```
if (!NT_SUCCESS (Status = HvmSwallowBluepill ())) {
  _KdPrint (("NEWBLUEPILL: HvmSwallowBluepill() failed with status 0x%08hX\n", Status));
```

-调用 HvmSwallowBluepill()
首先获取一个互斥锁对象，之后代码开始是不能重入的。
然后对每个处理器进行设置。

```
Status = CmDeliverToProcessor (cProcessorNumber, CmSubvert, NULL, &CallbackStatus);
```

--调用 CmDeliverToProcessor
主要执行的是 CallbackStatus = CallbackProc (CallbackParam);//执行 CmSubvert

---调用 CmSubvert()
CmSubvert 首先保存了 15 个寄存器，然后调用函数 HvmSubvertCpu

----调用 HvmSubvertCpu()
开始还是进行了一系列的初始化，包括 Traps 的注册，大部份初始化和内存相关，此处跳过。然后执行

```
Status = Hvm->ArchInitialize (Cpu, CmSlipIntoMatrix, GuestRsp);
```

-----调用 ArchInitialize(VmxInitialize)

```
  Cpu->Vmx.OriginaVmxonR = MmAllocateContiguousPages (VMX_VMXONR_SIZE_IN_PAGES, &Cpu->Vmx.OriginalVmxonRPA);
//分 配 VMXON Regin的 页 ， 详 见 3B 20.10.4
```

在执行 VMXON 之前，软件必须分配一块内存作为 VMXON region，处理器用它来支持 VMX operation。

```
//Allocate VMCS
Cpu->Vmx.OriginalVmcs = MmAllocateContiguousPages (VMX_VMCS_SIZE_IN_PAGES, &Cpu->Vmx.OriginalVmcsPA);
```

分配 VMCS 的内存空间。

```
if (!NT_SUCCESS (VmxEnable (Cpu->Vmx.OriginaVmxonR))) {
  _KdPrint (("VmxInitialize(): Failed to enable Vmx\n"));
  return STATUS_UNSUCCESSFUL;
}
```

------调用 VmxEnable()

```
if (!(cr4 & X86_CR4_VMXE))//3B 19.8 support vmx operation;windy
  return STATUS_NOT_SUPPORTED;

vmxmsr = MsrRead (MSR_IA32_FEATURE_CONTROL);//3B 19.7 VMXON is controlled by it;windy
if (!(vmxmsr & 4)) {//check: enable VMXON outside SMX operation ;windy
  _KdPrint (("VmxEnable(): VMX is not supported: IA32_FEATURE_CONTROL is 0x%llx\n", vmxmsr));
  return STATUS_NOT_SUPPORTED;
}
//Software can discover the VMCS revision identifier that a processor uses by reading the VMX capability MSR IA32_VMX_BASIC.3B 20.2;windy
vmxmsr = MsrRead (MSR_IA32_VMX_BASIC);
*((ULONG64 *) VmxonVA) = (vmxmsr & 0xffffffff);          //set up vmcs_revision_id
//Before executing VMXON, software should write the VMCS revision identifier to the VMXON region. 20.10.4
VmxonPA = MmGetPhysicalAddress (VmxonVA);
_KdPrint (("VmxEnable(): VmxonPA:  0x%llx\n", VmxonPA.QuadPart));
VmxTurnOn (MmGetPhysicalAddress (VmxonVA));
flags = RegGetRflags ();//获 取 rflags内 容
_KdPrint (("VmxEnable(): vmcs_revision_id: 0x%x  Eflags: 0x%x \n", vmxmsr, flags));
return STATUS_SUCCESS;
```

前几步都是检查性的，判断是否符合标准，过程与 3B，25.5 对应。这里要注意的是 set up vmcs_revision_id 的方法，软件必须把 VMCS revision identifier 写到 VMXON region 中作为 VMXON region 的初始化(3B 20.10.4)

VmxTurnOn 由汇编实现，用于执行 VMXON

```
//VMXON之 后 ， set up and launch a guest VM
  *((ULONG64 *) (Cpu->Vmx.OriginalVmcs)) = (MsrRead (MSR_IA32_VMX_BASIC) & 0xffffffff); //set up vmcs_revision_id
//VMCS revision identirfier在 VMCS中 offset 0的 位 置 ， 这 步 设 置 是 执 行 VMXON必 需 的 ， 必 要 步 骤 3B 25.5
  if (!NT_SUCCESS (Status = VmxSetupVMCS (Cpu, GuestRip, GuestRsp))) {
    _KdPrint (("Vmx(): VmxSetupVMCS() failed with status 0x%08hX\n", Status));
    VmxDisable ();
    return Status;
  }
```

------调用 VmxSetupVMCS()(3B 25.6)

……参考手册，细节有待研究

----回到 HvmSubvertCpu()

```
Status = Hvm->ArchVirtualize (Cpu);
```

-----调用 ArchVirtualize(VmxVirtualize)

执行 VmxLaunch()

这里由于要切换到 guest VM 下，因此会读取 VMCS 的相应位到 rip，rsp 等寄存器。

```
VmxWrite (GUEST_RIP, (ULONG64) GuestRip);      //setup guest ip:CmSlipIntoMatrix //VMLAUNCH调 用 时 执 行 设 置
```

VmxSetupVMCS()中已经把 Guest_rip 赋值为 CmSlipIntoMatrix 的地址，因此 VmxLaunch 会跳到这里来执行。另外同样的 RSP 的值通过追踪传参可以发现是指向 CmSubvert 执行的位置。因此调用 VmxLaunch 后，相当于从 CmSlipIntoMatrix 函数开始执行，栈也从 CmSubvert 调用保存寄存器并为函数调用分配空间后开始继续增长。

CmSlipIntoMatrix 首先调用了 HvmResumeGuest 来打印了一些信息，然后紧接着取消函数调用参数空间，恢复寄存器。于是相当于完成了 CmSubvert。

回到 DriverEntry

完成驱动安装部分的最后工作，结束。