虚拟地址空间布局

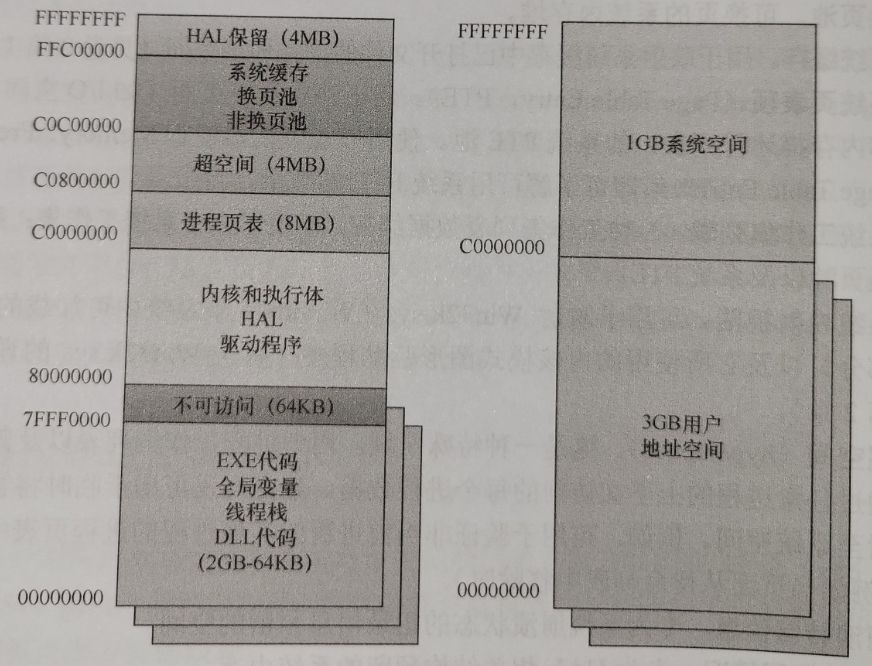
在 Windows中，主要有3 类数据会被映射到虚拟地址空间。

1. **每个进程的私有代码和数据。**跨进程内存函数(ReadProcessMemory和WriteProcessMemory)是由目标进程上下文中运行的内核模式代码来执行的。进程虚拟地址空间也叫作页表(page table)，每个进程都有自己的一套页表，该页表会存储在仅内核模式可以访问的页面中，因此进程中用户模式的线程无法修改自己的地址空间布局。
2. **整个会话范围的代码和数据。**会话由进程以及其他系统对象(例如窗口站、桌面、窗口) 组成，代表一个用户的登录会话。每个会话包含一个该会话专用的换页池区域，Windows 子系统 (Win32k.sys)内核模式的部分会使用该区域分配会话私有的 GUI 数据结构。此外，每个会话都有自己的Windows 子系统进程 (Csrss.exe) 副本和登录进程(Winlogon.exe) 副本。会话管理器程(Smss.exe)负责创建新会话，包括加载会话的私有 Win32k.sys 副本，创建会话私有对象管理器命名空间，并为该会话创建Csrss.exe和 Winlogon.exe进程实例。为了实现会话的虚拟化，所有会话范围的数据结构会映射至一个名为会话空间的系统空间区域。在创建进程时，该地址范围会被映射至与该进程所属会话相关联的页面。
3. **整个系统范围的代码和数据。**系统空间包含仅内核模式代码(无论当前执行的进程是什么)可见的全局操作系统代码和数据结构。系统空间由下列组件组成。
   * **系统代码**。包含操作系统映像、HAL 和用于引导系统的设备驱动程序
   * **非换页池**。不可换页的系统内存堆。
   * **换页池**。可换页的系统内存堆。
   * **系统缓存**。用于映射系统缓存中已打开文件的虚拟地址空间系统页表项(Page Table Entry，PTE)。用于映射系统页面(如I/O空间、内核栈和内存描述符列表)的系统PTE池。使用性能监视器检查Memory: Free System Page Table Entries 值即可了解可用系统 PTE的数量。
   * **系统工作集列表**。一种工作集列表数据结构，描述了3个系统工作集:系统缓存换页池以及系统 PTE。
   * **系统映射视图**。可用于映射 Win32k.sys以及它所使用的内核模式图形驱动程序。
   * **超空间**(hyperspace)。这是一种特殊区域，用于进程工作集列表以及其他不需要通过任意进程的上下文访问的每个进程数据。超空间还可用于临时将物理页面映射至系统空间。例如，可用于验证非当前进程的其他进程的进程页表中的页表项(例如当页面从待命列表中移除时)。
   * **崩溃转储信息**。专为系统崩溃状态的记录信息预留的空间。
   * **HAL使用情况**。专为HAL相关结构预留的系统内存。

x86地址空间布局

默认情况下，32位Windows 中每个用户进程可以获得2GB私有地址空间。(剩余2GB由操作系统使用。)然而对于x86体系结构，可为系统配置BCD的increaseuserva引导选项分配最大3GB的用户地址空间，另外进程映像文件头必须设置IMAGE\_FILE\_LARGE\_ADDRESS\_AWARE标志。

如果使用3GB 地址空间，可能会无意中截断了值大于 2GB的指针，进而导致程序出错甚至可能造成数据损坏。若要设置该标志，可在构建可执行文件时指定/LARGEADDRESSAWARE链接器标志。或者可以在 Visual Studio中使用Property页面(选择Linker，选择System，单击 Enable Large Addresses。)如果体用诸如Editbin.exe(包含在 Windows SDK 工具中)等工具，甚至可以在不进行构建(也无须源代码)的情况下为可执行映像文件添加该标志(前提是文件不包含数字签名)。如果在具备 2GB 用户地址空间的系统中运行应用程序，该标志将不产生任何效果。



默认情况下使用 VirtualAlloc、VirtualAllocEx和 VirtualAllocExNumastart 执行的内存分配是从低位虚拟地址开始，并向着高位地址增长的。除非进程分配了大量内存,或虚拟地址空间的碎片化问题非常严重，否则永远不会达到非常高位的地址上。因此出于测试目的，我们可以使用 VirtualAlloc函数的MEM\_TOP\_DOWN 标志，或在注册表HKLM\SYSTEM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management键下创建一个名为AllocationPreference 的 DWORD值，并将其数值设置为0x100000，借此强制从高位地址开始分配内存。

我们可以在以管理员身份运行的命令提示符窗口中运行/set increaseuserva 3072命令切换至3GB 地址空间，该命令允许我们(以MB 为单位)指定介于2048(默认的2GB)和3072(最大3GB)之间的任何数值。重新启动系统后改动即可生效

如果希望将系统恢复为默认的每个进程2GB 地址空间，请运行 bcdedit/deletevalue increaseuserva 命令

x86系统地址空间布局

32位版 Windows 使用虚拟地址分配器实现了一种动态系统地址空间布局。不过依然会保留一些特定区域(如上图)。然而很多内核模式结构使用了动态地址空间分配因此这些结构的虚拟地址并不一定是连续的。它们的每个地址都可能位于系统地址空间同区域内不连续的多个内存片段里。通过这种方式分配系统地址空间，主要用于非换页池换页池、特殊池、系统 PTE、系统映射视图、文件系统缓存、PFN 数据库和会话空间等场景。

x86会话空间

对于包含多个会话的系统(通常几乎所有系统都是如此，系统进程和服务使用会话0，首个登录用户使用会话 1),每个会话独有的代码和数据会映射至系统地址空间，并被该会话中的进程共享。下图展示了会话空间的常规布局(未按比例绘制)。会话空间中组件的大小与内核系统地址空间的其他部分一样，可由内存管理器根据需要动态配置并调整大小。

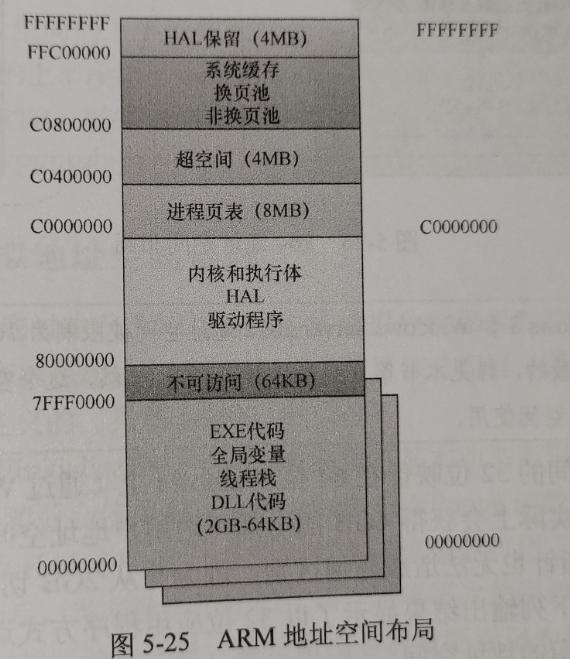
|  |
| --- |
| 会话数据结构和工作集 |
| 会话视图映射 |
| 会话换页池 |
| 会话驱动程序映像 |
| Win32k.sys |

系统页表项

系统页表项(PTE)可用于动态地映射系统页面(如I/O空间、内核栈、内存描述符列表)。系统PTE资源是有限的。在32位Windows中，系统PTE的可用数量在理论上可以供系统描述2GB大小的持续系统虚拟地址空间。在64位Windows10和Windows Server 2016中，系统PTE最多可描述16TB的连续虚拟地址空间。可以使用调试器中的!sysptes 或!vm 命令。

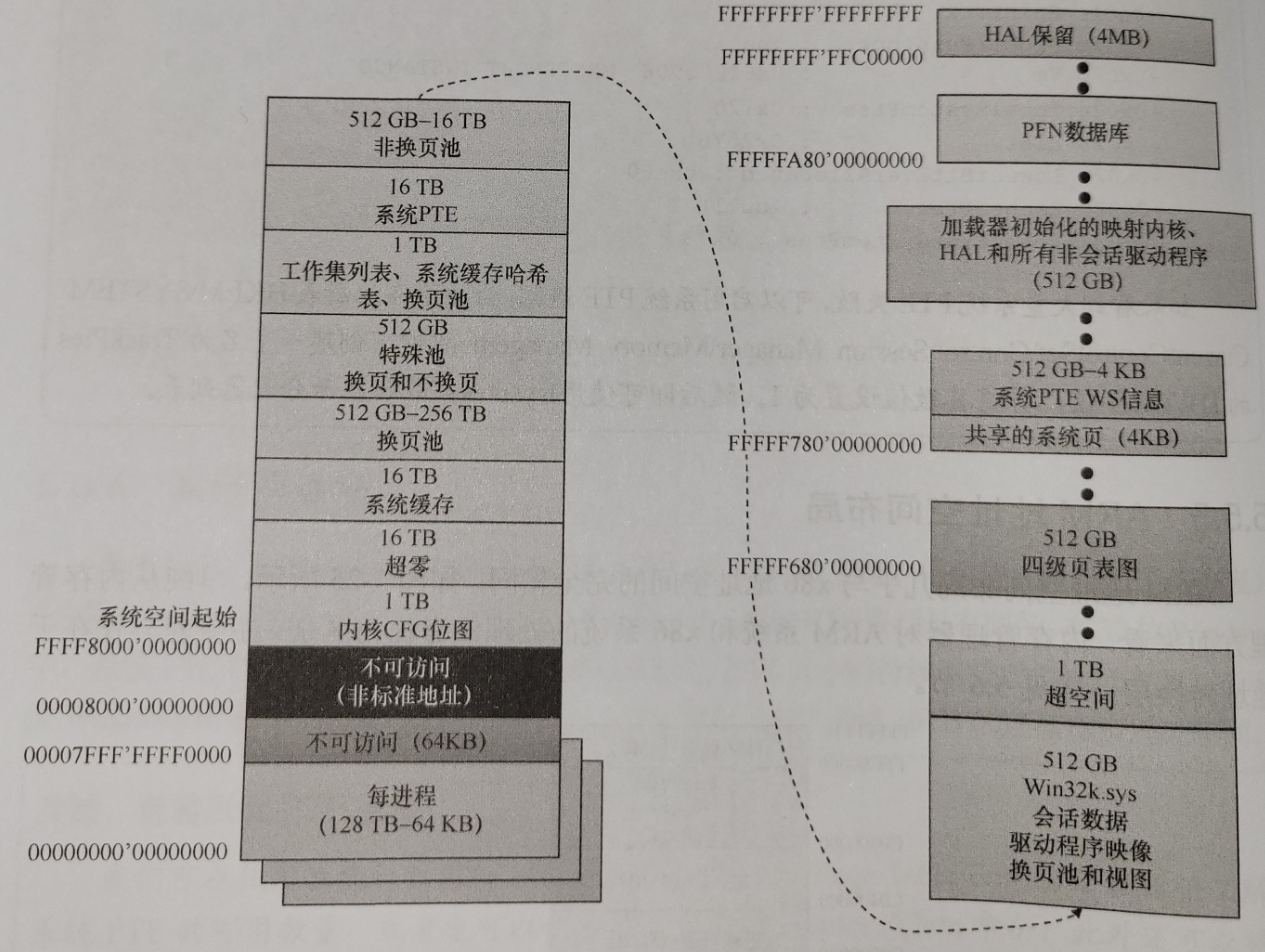
ARM地址空间布局

ARM 地址空间布局几乎与x86 地址空间的完全相同，如下图所示。单纯从内存管理方面来看，内存管理器对ARM系统和x86系统的处理没有任何区别。主要差异仅在地址转换层



64位地址空间布局

理论上，64 位虚拟地址空间可达 16EB (exabyte)，目前在处理器方面的限制导致最多只能使用 48 位地址，因此地址空间最大可以达到256TB(2的48 次方)。该地址空间会对半划分，低位的 128TB 供私有用户进程使用,高位的128TB供系统空间使用。(在 Windows 10和 Windows Server 2016 中)系统空间将进一步划分为多个不同大小的区域，如下图所示。很明显，相比 32 位，从地址空间大小的角度来看，64位代表着一次重大的飞跃。但是注意，由于最新版 Windows 为内核空间使用了ASLR(地址空间布局随机化)机制，不同内核节 (section)的实际起始地址并不一定是下图所示的地址。



可感知大地址空间的32位映像在64位 Windows上(通过Wow64)运行时还能获得额外收益。此类映像实际上会获得 4GB 的全部可用用户地址空间。毕竟如果映像可以支持3GB 指针，4GB 指针也无法造成任何区别，因为与从2GB 切换至3GB 时不同，此并不涉及更多位数。

x64 虚拟寻址的局限

因此为了简化芯片的体系架构并避免不必要的开销(尤其是地址转换方面的开销)，AMD和Intel目前的x64处理器均只实现了256TB 的虚拟地址空间。也就是说，在完整的 64 位虚拟地址中，只实现了最低的 48 位地址。然而虚拟地址依然是64位宽的，在寄存器或内存中存储时需要占用8个字节。接下来的高16位(48到63位)必须设置为与所实现的最高位(第47 位)相同的值，这也是一种类似于二进制补码运算(complement arithmetic)符号扩展的过程。符合这种规则的地址也可称为规范的(canonical)地址。

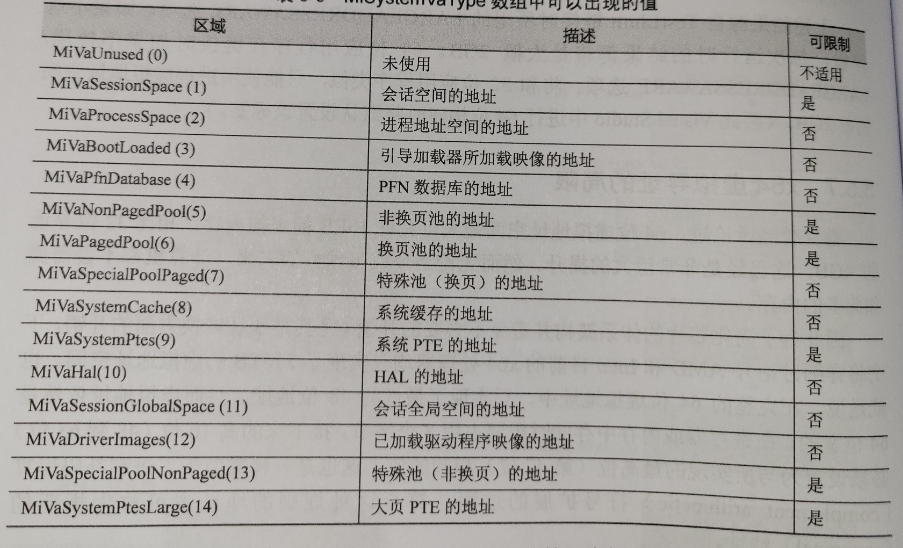
根据这些规则，地址空间的下半段按预期始于 0x0000000000000000，止于 0x00007FFFFFFFFFFF;地址空间的上半段始于 0xFFFF800000000000，止于 0xFFFFFFFFFFFFFFF。每个规范的区域均为 128TB。随着以后的新处理器开始实现更多地址位，内存的下半段将向上扩展至 0x7FFFFFFFFFFFFFFF，而上半段将向下扩展至 0x8000000000000000。

动态系统虚拟地址空间管理

当系统初始化时，MilnitializeDynamicVa函数会将基本动态范围和可用虚拟地址设置为所有可用的内核空间。随后会使用MilnitializeSystemVaRange函数,为引导加载器映像进程空间(超空间)以及 HAL初始化地址空间范围，该函数可用于设置硬编码的地址范围(仅限 32 位系统)。随后，当非换页池初始化后，还将再次使用该函数为非换页池保留虚拟地址范围。最后在加载驱动程序时，地址范围会被重新标记为驱动程序映像的范围，而不再是引导加载的范围。

在这之后，系统虚拟地址空间的其他部分可通过 MiObtainSystemVa(及与它类似的MiObtainSessionVa)和MiReturnSystemVa 以动态方式请求并释放。如扩展系统缓存、系统 PTE、非换页池、换页池或特殊池等操作，使用大空间映射内存的操作，创建PFN操作，及新建会话操作，这些操作都会导致为特定范围分配动态虚拟地址。

每次内核虚拟地址空间分配器通过某类虚拟地址获取要使用的虚拟内存范围时，还会更新 MiSystemVaType数组，该数组中包含了新分配范围的虚拟地址类型信息。MiSystemVaType数组中可以出现的值见下图



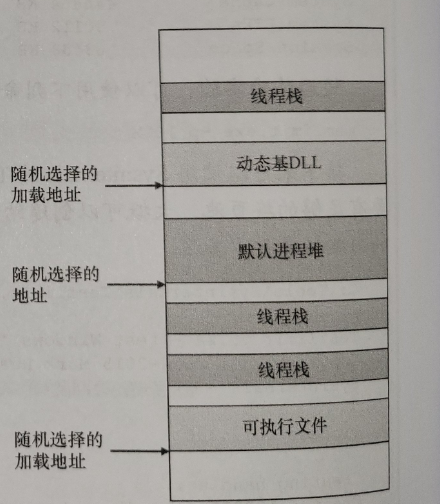
组件也可以通过MiReclaimSystemVa回收内存，这需要当可用虚拟地址空间低于 128MB 时，能够(通过解除对节的引用来)清空与系统缓存相关的虚拟地址。如果初始非换页池被释放，也将进行这样的回收。

系统虚拟地址空间配额

可以在注册表HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management键下配置 PagedPoolQuota、NonPagedPoolQuota、PagingFileQuota和 WorkingSetPagesQuota 值，借此限制特定进程每个类型的地址可以使用多少内存。该信息会在系统初始化时读取，同时还会生成默认的系统配额块并将其分配给所有系统进程。

要启用每个用户配额，我们可以在注册表HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Quota System键下创建子键，每个子键对应一个特定用户SID。随后即可在特定SID 子键下创建前文提到的值，借此为该用户创建的进程强制施加限制。下表显示了这些值的配置方式(能否在运行时配置) 以及所需特权。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **值名称** | **描述** | **值类型** | **动态配置** | **特权** |
| PagedPoolQuota | 此进程可分配换页池的最大大小 | MB为单位的大小 | 仅限以System令牌运行的进程 | SeIncreaseQuota Privilege |
| NonPagedPool Quota | 此进程可分配非换页池的最大大小 | MB为单位的大小 | 仅限以System令牌运行的进程 | SeIncreaseQuota Privilege |
| PagingFileQuota | 此进程可包含的、由页面文件支撑的页面数量最大值 | 页面数 | 仅限以System令牌运行的进程 | SeIncreaseQuota Privilege |
| WorkingSetPages Quota | 此进程可在自己工作集(物理内存)中使用的页面数量最大值 | 页面数 | 支持 | SelncreaseBasePriorityPrivilege ,除非请求的是清空操作 |



用户地址空间布局

在操作系统层面上，用户地址空间可分为几个明确定义的内存区域，如右图所示。可执行文件和 DLL 本身均表现为内存映射的映像文件随后是进程堆以及进程中的线程栈。除了这些区域(以及一些保留的系统结构，例如TEB和PEB)外，所有其他内存分配都依赖于运行时并生成于运行时。ASLR 会负责处理所有依赖于运行时的区域位置，并与 DEP 一起提供了一种机制，让通过远程操纵内存挖掘系统数据的做法变得难以成功。由于Windows代码和数据都放置在动态位置上，攻击者将无法在程序或系统提供的 DLL 中通过典型的硬编码方式获得有意义的偏移量。

ASLR始于映像层面，会应用于进程的可执行文件及其依赖的 DLL。任何映像文件如果在 PE 头(IMAGE\_DLL\_CHARACTERISTICS\_DYNAMIC\_BASE) 指定了对ASLR的支持(例如通常可在 Microsoft Visual Studio 中指定/DYNAMICBASE 链接器标志)，并且包含重定位内存区域，那么就会由 ASLR 来处理。在找到此类映像后，系统会选择个对当前引导来说全局有效的映像偏移量，该偏移量是从一个包含 256 个值的桶中选出的，所有值均以64KB 为边界对齐。

1. **映像随机化**

对于可执行文件，可通过计算可执行文件每次加载时的增量值得到加载的偏移量。该增量值是一个介于0x10000和0xFE0000之间的8比特伪随机数，其计算方式为将当前处理器的时间戳计数器(Time Stamp Counter，TSC)右移4位后模254再加1，随后将得出的数值乘以分配粒度，即 64KB。因为加了1，内存管理器可以确保该值绝对不会为0，进而确保了启用ASLR的情况下，可执行文件绝对不会加载到PE头中记录的地址中。随后该增量值会与可执行文件的首选加载地址相加，从而获得 PE 头中映像地址之后 16MB 范围内共256 个可能位置中的一个位置。

对于 DLL，加载偏移量的计算会在每次引导时从系统范围内一个名为映像偏差(image bias)的值开始进行。这个计算由 MiInitializeRelocations 函数负责，结果会存储在全局内存状态结构(MI\_SYSTEM\_INFORMATION)中的 MiState.Sections.ImageBias 字段内 (Windows 8.x、Windows Server 2012/2012 R2 则会保存在全局变量 MilmageBias中)。该值对应着引导过程中调用 MilnitializeRelocations函数时当前CPU的TSC值，会被移位并掩码为8位值，借此可在 32位系统中提供256个可能的值；地址空间更大的64位系统也会进行类似的计算并获得更多可能的值。与可执行文件不同，该值只在系统每次引导时计算一次，随后即可与整个系统共享，以便在物理内存中维持 DLL 的共享，并确保DLL只需要重定位一次。如果 DLL 被重新映射至另一个进程中的不同位置，此时代码将无法共享。加载器必须针对每个进程进行不同的地址引用修正，借此让原本可共享的只读代码成为进程的私有数据。每个使用特定DLL的进程必须在物理内存中针对该DLL维持一个自己私有的副本。

计算出偏移量后，内存管理器会初始化一个名为ImageBitMap (对于 Windows 8.x、Windows Server 2012/2012 R2，初始化的是MilmageBitMap 全局变量)的位图，该位图也是MISECTION STATE 结构的一部分。在 32位系统中，这个位图可用于代表从0x50000000到0x78000000的地址范围 64位系统的相应数字见下文)，其中每一位代表个分配单位(前文提到过，每个分配单位大小为 64KB)。当内存管理器加载 DLL时,会设置相应的位，借此在系统中标识其位置。当同一个 DLL 再次加载时，内存管理器会为节对象共享已经重定向后的信息。

在加载每个 DLL 时，系统会在位图中从上到下扫描空闲位。此时会使用早先计算而来的ImageBias值作为从顶部开始的起始索引，借此即可像前文介绍的那样在每次引导实现随机化的加载。由于在加载第一个DLL(始终为 Ntdll.dll)时该位图是空的，因此很容易即可算出它的加载地址。(64 位系统也有自己的偏差。)

32位。0x78000000-(ImageBias +NtDllSizein64KBChunks)\* 0x10000。

64位。0x7FFFFFFF0000-(ImageBias64High +NtDllSizein64KBChunks)\* 0x10000。

1. **栈随机化**

ASLR接下来需要对初始线程(及后续每个新线程)的栈位置进行随机化。除非为进程启用了 StackRandomizationDisabled 标志，否则该随机化会默认启用，并从32个可能的栈位置(64KB或256KB 分隔)中选择一个作为第一个位置。这个基址的选择方法为:首先找出适合的第一个空闲内存区域，随后选择第x个可用区域，这里的“x”也是通过当前处理器的 TSC移位并掩码为一个5位值计算而来的。(借此可获得32个可能的位置。)

选出该基址后，还将计算一个由 TSC 派生出的新值，该值长度为9位。随后将该值乘以4以实现对齐，这也意味着这个值最大可以为 2048 字节(半个页面)。将其与基址相加即可得到最终的栈基址。

1. **堆随机化**

用户模式下创建的初始进程堆和后续堆的地址也会进行ASLR 随机化。此时 RtlCreateHeap函数会使用由 TSC 派生的另一个伪随机值来确定堆的基址。这个伪随机值长度为5位，将其与 64KB 相乘可得到最终的基址。该基址从 0 开始，因此初始堆的可能范围介于0x00000000和0x001F0000之间。此外堆基址前方的地址范围还会通过手动方式撤销分配这样，当攻击者试图对所有可能的堆地址范围进行暴力扫描时，可迫使产生访问冲突。

1. **内核地址空间中的ASLR**

ASLR还可作用于内核地址空间。32位驱动程序可以有64 个可能的加载地址，64位驱动程序则可能有256个。如果要重定位用户空间中的映像，需要用到内核空间中的大量工作区域，但如果内核空间很紧张，ASLR 可以把System 进程的用户模式地址空间用于这个工作区域。在 Windows 10(版本 1607)和Windows Server 2016 中，大部分系统内存区域均实现了 ASLR，例如换页和非换页池、系统缓存、页表以及 PFN 数据库(由MiAssignTopLevelRanges 初始化)。

1. **控制安全缓解措施**

为了让企业用户与个人能对这些功能更可见和可控，微软提供了增强缓解体验工具包(Enhanced Mitigation Experience Toolkit，EMET)。EMET可用于集中控制Windows提供的缓解措施,并提供了几个尚未包含在Windows产品中的缓解措施。EMET还可以通过事件日志提供通知能力，让管理员了解某个软件何时因为所使用的缓解措施而遇到了访问错误。最后，EMET 还可供用户在某些环境中，为某些可能会遇到兼容性问题的应用程序选择性地关闭某些缓解措施(哪怕应用程序开发者选择性地启用了这些措施)

虽然存在ASLR机制，但是仍然可以通过PE头中的信息确定可执行文件和DLL文件的基址以及入口点和重定位信息，可以通过fs(x86)或gs(x64)寄存器(x64为0x60，x86为0x30)获取PEB结构进而获取堆和栈的基址