堆管理器

堆管理器存在于两个位置:Ntdll.dll 和 Ntoskrnl.exe。子系统API (例如 Windows API)调用 Ntdll.dll 中的函数，而各种执行体组件和设备驱动程序会调用 Ntoskrnl.exe的函数。其原生接口(带有 Rtl 前缀)仅限 Windows 内部组件或内核模式设备驱动使用。与堆有关的文档化 Windows API接口(前为Heap)实际上是到Ntdll.dll中原函数的转发器。此外，提供遗留 API(前缀为 Local 或 Global) 还可为老版本 Window应用程序提供支持。它们也在内部调用堆管理器，使用自己的一些特殊接口为遗留行为提供支持。最常见的Windows 堆函数如下。

1. **HeapCreate或 HeapDestroy**。分别负责创建或删除堆。创建时可指定初始保留大小和已提交大小。
2. **HeapAIloc**。分配堆块，可转发至Ntdll.dll 中的 RtlAllocateHeap。
3. **HeapFree**。可释放之前由 HeapAlloc 分配的块。
4. **HeapReAIloc**。可更改现有分配的大小，借此增大或缩小现有块。可转发至 Ntdll.dll中的 RtlReAllocateHeap。
5. **HeapLock 和HeapUnlock**。控制了堆操作的互斥(mutual exclusion)
6. **HeapWalk**。枚举堆中的项和区域。

进程堆

每个进程至少有一个堆——默认进程堆。默认进程堆是在进程启动时创建的，在进程生命周期内永远不会删除。其大小默认为1MB，但可在映像文件中使用/HEAP链接器标志指定更大的初始大小。不过该大小仅仅是初始时的保留大小，随着需求的变化，其大小也会自动变化。我们还可以在映像文件中指定初始已提交大小。

默认进程堆可被程序显式使用，或被某些 Windows 内部函数隐式使用。应用程序可调用Windows的GetProcessHeap 函数来查询默认进程堆。进程还可以使用HeapCreate函数创建额外的私有堆。当进程不再需要某个私有堆时，可调用 HeapDestroy 恢复自己的虚拟地址空间。每个进程都维持了所有堆信息的数组，线程可以通过 Windows 的 GetProcessHeap函数查询该数组。

通用Windows平台(UWP)应用的进程至少包含如下3个堆

1. 前文所述的默认进程堆。
2. 一个用于将大型参数传递给进程会话 Csrss.exe 实例的共享堆。这是通过 Ntdll.dll的 CsrClientConnectToServer 函数创建的，Ntdll.dll 会在进程初始化的早期阶段执行该作。堆句柄可通过(Ntdll.dll中的)全局变量 CsrPortHeap 使用。
3. 一个由Microsoft C运行时库创建的堆。其柄会存储在 (msvcrt 模块内部的)全局变量\_crtheap 中。该堆将被 malloc、free、operat 或 new/delete 等 C/C++内存分配函数在内部使用。

堆可以管理由内存管理器通过 VirtualAlloc 保留的大内存区域分配，也可管理进程地址空间中映射的内存映射文件对象的内存分配。实践中，后一种方法很罕见(且 Windows API未暴露这种做法)，但在需要跨越两个进程共享，或在内核模式与用户模式组件间共享内存块的内容时，这也是一种适合的做法。Win32 GUI子系统驱动程序(Win32k.sys) 会使用这种堆与用户模式共享GDI和USER 对象。

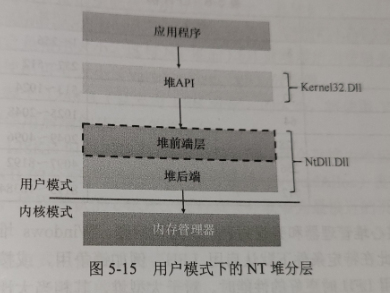
堆的类型

在Windows 10和Windows Server 2016之前，只有一种类型的堆，即NT堆。NT堆可通过一种可选的前端层进一步扩展，在使用这种扩展的情况下，NT 堆还将包含低碎片堆(Low-fragmentation Heap，LFH)。

Windows 10引入了一种名为段堆(segment heap)的全新堆类型。这两种类型的堆包含了一些通用元素，但在结构和实现上各有不同。默认情况下，段堆主要被所有UWP应用和某些系统进程使用，其他所有进程则使用NT 堆。不过这种行为可通过注册表进行调整

NT堆

用户模式的NT堆由一个前端层和一个堆后端(有时也叫作堆核心)两层组成。后端承担了堆的基本功能，这包括段内部内存块管理、段管理、堆扩展策略、内存提交和解除提交以及大块管理。



堆同步

堆管理器默认支持多个线程的并发访问。然而，如果进程是单线程的，或使用了外部同步机制，即可在创建堆的时候或在每次分配时指定 HEAP\_NO\_SERIALIZE 标志，借此告诉堆管理器，以避免同步所造成的开销。如果启用堆同步，每个堆还可获得一个用于保护内部堆结构的锁。

进程也可以将整个堆锁定起来,借此防止其他线程执行可能需要跨越多个堆调用实现一致状态的堆操作。例如，使用 Windows 函数 HeapWalk 枚举堆内部的堆块时，如果多线程可同时执行堆操作，就需要具备堆锁。堆的锁定和解锁可分别通过 HeapLock 和HeapUnlock 函数实现。

低碎片堆

Windows中运行的很多应用程序往往有着较小的堆内存用量(通常不会超过1MB)对于此类应用程序，堆管理器的最佳匹配策略会确保每个进程只占用最少量的内存。然而这种策略并不能很好地适应大型进程和多处理器计算机。此时，由于堆碎片的存在，堆可用的内存可能会减少。某些情况下，调度到不同处理器的多个线程常常需要并发地使用特定大小的块，这也会导致性能进一步退化。原因在于多个处理器需要同时修改相同内存位置(例如特定大小的旁视列表头)，进而在相应处理器的缓存线上产生激烈的争用。

LFH(低碎片堆)通过将已分配的内存块映射为事先确定了不同大小范围的桶 (bucket)，可以有效避免碎片的出现。当进程从堆分配内存时，LFH 会选择映射到足以包含所需大小的最小内存块所对应的桶。下表总结了不同的桶，它们的粒度，以及可映射的大小范围。

|  |  |  |
| --- | --- | --- |
| **桶** | **粒度** | **范围** |
| 1~32 | 8 | 1~256(一个桶8) |
| 33~48 | 16 | 257~512(一个桶16) |
| 49~64 | 32 | 513~1024 |
| 65~80 | 64 | 1025~2048 |
| 81~96 | 128 | 2049~4096 |
| 97~112 | 256 | 4097~8192 |
| 113~128 | 512 | 8193~16384 |

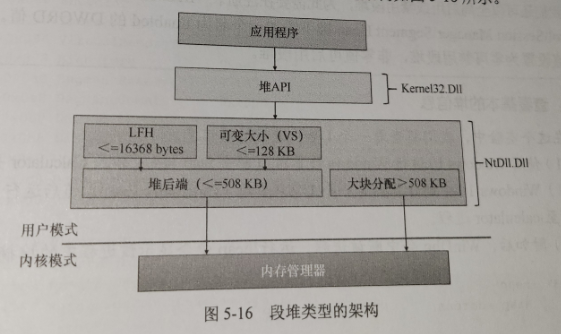
为了解决可伸缩性问题，LFH 还会对频繁访问的内部结构进行扩展，将其变为多个槽(slot)，而槽的数量等同于计算机中当前处理器数量的两倍。将线程分配到这些槽的工作是由一个名为相关性管理器 (affinity manager)的LFH组件进行的。初始时，LFH会为堆分配使用第一个槽，如果在访问某些内部数据时检测到争用，LFH 会将当前线程切换到一个不同的槽。如果随后还出现争用，则会进一步将线程分散给更多槽。对这些槽的控制是针对每个大小的桶分别进行的，这也是为了提高“局部性”并尽可能降低整体内存用量。

低碎片堆通过以下方式优化内存分配，以减少内存碎片：

1. 对象分组：LFH 将具有相似大小的对象分组在一起。这样，当一个对象被释放时，它附近的内存块很可能具有相似的大小，从而提高了内存重用的可能性。
2. 分配策略：LFH 使用一种称为“最佳适应”的分配策略，它试图在所有可用内存块中找到与请求大小最接近的内存块。这有助于减少由于分配过大内存块而产生的内存浪费。
3. 碎片合并：LFH 会在分配内存时尝试合并相邻的空闲内存块，从而减少碎片的产生。
4. 分离空闲内存：LFH 将空闲内存块与正在使用的内存块分离。这可以防止空闲内存块被频繁访问，从而提高内存使用效率。

段堆

Windows 10中引入的段 (segment heap)类型的架构如下图所示。



实际负责管理所有分配的层取决于所分配的大小，具体如下。

1. 对于较小的分配大小(小于等于 16368 字节)，将使用LFH分配器 (allocator),前提是该大小被确定是常用大小。这与NT 堆的LFH前端层所用逻辑类似。如果LFH尚未引入，则会转为使用可变大小 (Variable Size，VS)分配器。
2. 对于小于等于 128KB(且未由 LFH 分配)的大小，将使用 VS 分配器。VS 和LFH分配器会酌情使用后端创建所需的堆子段。
3. 对于大于128KB但小于等于 508KB 的分配，则会直接由堆后端进行分配。
4. 大于508KB的分配将通过调用内存管理器 (VirtualAlloc)直接实现，因为这种分配实在是太大了，使用默认的64KB 分配粒度(并取整为最接近的页大小)往往是更适合的做法。

堆的这两种实现方式可做如下简要对比。

1. 某些场景中，段堆可能比 NT 堆慢，但很可能未来版本的 Windows 会通过改进使其实现与 NT 堆类似的速度。
2. 段堆的元数据内存占用更小，因此更适合手机等小内存设备。
3. 段堆的元数据会与实际数据分隔，而NT 堆的元数据会与实际数据掺杂在一起因此段堆更安全，因为更难以仅通过块地址就从分配中获取元数据。
4. 段堆只能用于可增长堆，无法用于用户提供的内存映射文件。如果企图用该方式创建段堆，则会转为创建 NT 堆
5. 这两种堆都支持 LFH 类型的分配，但两者的内部实现完全不同。

NT堆的LFH实现：

NT堆的LFH在Windows XP和Windows Server 2003中首次引入。在NT堆中，LFH是作为一个可选策略实现的，当检测到一定数量的碎片时，才会自动启用。LFH为小型内存分配提供了专门的子堆，这些子堆按大小划分为多个桶（bucket）。每个桶存储特定大小的内存块。在分配时，NT堆会选择与请求大小最接近的桶，并尝试从中分配内存块。为了保持连续性并减少碎片，NT堆会尽量使用相邻的空闲块来满足内存请求。

段堆的LFH实现：

段堆是在Windows 10版本2004（20H1）中引入的一种新型堆策略。段堆的LFH实现与NT堆有很大不同。首先，段堆是默认启用的，而不是根据内存碎片情况自动切换。其次，段堆的内部结构更加高效，以便更快地满足内存请求。段堆中的内存分配使用了所谓的“段”来存储内存块。这些段按大小分组，每组段只存储特定大小范围的内存块。段堆中的LFH实现还包括了对并发分配的优化，以提高多线程应用程序的性能。

若要为特定可执行文件启用或禁用段堆，可以设置映像文件执行选项中名为 FrontEndHeapDebugOptions的DWORD值: Bit 2(4) 可禁用段堆；Bit 3(8) 可启用段堆。

我们还可以全局启用或禁用段堆，为此需要在注册表 HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap 键下添加一个名为 Enabled的DWORD值。将该值的数值设置为零可禁用段堆，非零值可启用段堆

堆的安全特性

堆用来实现内部管理所用的元数据采用了一种高度随机化的封装方式，使得通过修改内部结构以防止崩溃或隐藏攻击企图的做法变得更加困难。这些内存块还可通过针对块头进行完整性检查来检测一些简单的破坏，如缓冲区溢出。最后，堆的基址或句柄也进行了少量程度的随机化处理。通过使用HeapSetInformation API以及HeapEnableTerminationOnCorruption类，进程可以在检测到不一致时选择性地启用自动终止功能，进而避免执行未知代码。

与段堆有关的安全特性

段堆的实现使用了很多安全机制，使其难以破坏内存或被攻击者注入代码，如下所示。

1. **链接列表节点出错时快速失败**。段堆使用链接列表跟踪段和子段。与NT 类列表节点的插入和删除增加了检查机制，以防止列表节点出错导致的随意内存写入。如检测到出错的节点，进程将调用 RtlFailFast 并终止。
2. **红黑 (Red-black，RB)树节点出错时快速失败**。段堆使用 RB 树跟踪空闲的端和 VS 分配。节点插入和删除函数会验证所涉及的节点，如果节点出错，会调用快速失败机制。
3. **函数指针解码**。段堆的某些部分可允许回调(位于\_SEGMENT\_HEAP 结构中VsContext 和 LfhContext 结构内)。攻击者可以覆盖这些回调并指向自己的代码。然而数指针会通过XOR函数使用一个内部随机堆密钥和上下文地址进行编码，这两个因素都是无法事先猜测的。
4. **保护页**。当分配了 LFH 和 VS 子段及大块时，会在末尾添加一个守护页。这有助于检测毗邻数据的溢出和错误

堆的调试特性

堆管理器提供的一些特性有助于使用下列堆设置检测 bug。

1. **启用尾部检查。**每个块的尾部包含一个签名，释放该块的时候会检查该签名。如果缓冲区溢出彻底/部分销毁了该签名，堆将上报这个错误。
2. **启用空闲检查。**空闲块会被填充为一种特定模式，而堆管理器在需要访问该块(例如从空闲列表中删除，以满足分配请求)之前会在多个点检查这种模式。如果在释放之后依然有进程继续写入该块，堆管理器会检测到模式的变化并上报错误
3. **参数检查。**该功能包含了作为参数传递给堆函数的大量检查。
4. **堆验证。**每次调用每个堆之前，都会对整个堆进行验证。
5. **堆标记和栈跟踪支持。**该功能可以为每个分配指定标记，并/或为堆调用捕获用户模式栈追踪记录，以便在遇到造成堆错误的问题时缩小可能的原因范围。

页堆

前文提及的尾部检查和空闲检查可能会检测到内存破坏，而实际造成的问题会在检测到之后很久才出现，因此系统提供了一种名为页堆(page heap)的堆调试能力。页堆会将到堆的全部或部分调用定向至另一个堆管理器。当开启页堆时，系统在每个内存分配之前和之后放置一个特殊的内存页面，这个页面的访问权限设置为“无访问”。这意味着如果程序试图访问这些页面，将触发访问异常。通过这种方式，页堆可以立即检测到缓冲区溢出、释放后使用等问题。页堆还可以保护已释放的页面，防止堆块被释放后对其中任何页面的引用企图。

容错堆

堆元数据的损坏已经成为应用程序失败的最常见原因之一。为了缓解此类问题并为需要解决此类问题的应用程序开发者提供更有用的资源，Windows 提供了一种名为容错堆(Fault-tolerant Heap，FTH)的功能。FTH通过两个主要组件来实现：检测组件(FTH服务器)和缓解组件 (FTH客户端)。

检测组件是一个名为 Fthsvc.dll的DLL，位于 Windows 安全中心服务(Wscsvc.dll)中，而安全中心服务使用 Local Service 账户运行在一个共享服务进程中。Windows 错误报告 (WER)服务会将应用程序崩溃事件通知给该服务。

当一个应用程序在 Ntdll.dll 中崩溃且其错误状态指出问题源自访问冲突或堆损坏异常时，如果该应用程序尚未包含在 FTH服务的监视应用程序列表中，FTH服务将为这个应用程序创建一个用于保存FTH数据的“工单”。如果随后该应用程序崩溃的频率超过每小时 4次，FTH 服务会配置此应用程序以后开始使用FTH客户端。

FTH客户端是一种应用程序兼容性填充码。自Windows XP开始就在使用该机制来让依赖老版本 Windows 中特定行为的应用程序顺利运行在新版操作系统中。在本例中，填充码机制会拦截对堆例程的调用，并将其重定向至FTH自己的代码。FTH代码会通过多种缓解措施试图让应用程序从各种与堆有关的错误中恢复出来。

例如，为了预防小缓冲区溢出错误，FTH 会为每个内存分配添加 8 字节的填充(padding)和一个 FTH 保留区域。为解决堆块在释放后又被访问这种常见问题，只有在延迟一段时间后才会调用 HeapFree。“被释放”的块会被包含在一个列表中，并且仅在该列表中块的总大小超过 4MB 后才会被释放。如果试图释放不属于堆的区域，或试图释放并未由 HeapFree 的堆句柄参数所识别的堆，这些企图会被忽略。此外如果调用了 exit 或RtlExitUserProcess，任何块都将不再被真正释放。

在安装缓解措施后，FTH 服务器会持续监视应用程序的失败率。如果失败率没有观，则缓解措施会被取消。

我们可以在事件查看器中观察到容错堆的行为。为此请执行如下步骤。

* 1. 打开Run对话框并输入eventvwr.msc。
  2. 在左侧窗格中选择 Event Viewer，选择 Applications and Services，选择Microsoft选择Windows，单击Fault-Tolerant-Heap。
  3. 单击 Operational日志。
  4. FTH可以通过注册表彻底禁用，在注册表HKLM\Software\Microsoft\FTH键下点Enabled值设置为0即可。

该注册表键还包含了与FTH有关的不同设置，如前文提到的延迟以及可执行文件的排除列表(该列表默认包含了 smss.exe、csrss.exe、wininit.exe、services.exe、winlogon.exe和taskhost.exe 等系统进程)。此外该键下还包含一个规则列表(RuleList值)，其中列出了为让FTH生效而需要监视的模块和异常类型(以及一些标志)。默认情况下这里只有一条规则对应着Ntdll.dll 中类型为STATUS\_ACCESSVIOLATION(0xc0000005)的堆问题。

FTH通常并不会作用于服务，并且出于性能方面的考虑，在Windows Server 系统是禁用的。系统管理员也可以通过应用程序兼容性工具包(application compatibility toolkit)手动为应用程序或服务的可执行文件设置填充码。