一个 Windows 应用程序的垃圾回收器

苗宏伟[1],钟云鹏[2]
(1.天津大学 微软.net 实验室,天津 300072;
2.装甲兵工程学院 控制系,北京 100072)

摘 要:为减少长期运行的 Windows 应用程序持续堆内存泄漏而造成的系统性能损失,设计并实现了一个运行时自动垃圾回收器 RT-AGC。RT-AGC 采用传统 Mark-Sweep 算法的一个变种,对回收触发条件进行改进,使其具有更大的灵活性。利用 Windows 下应用程序的存储结构和系统 API 函数,通过扫描可能存储指针变量的区域,RT-AGC 可以在应用程序运行时检测并处理内存泄漏,而不需要对目标应用程序重新编译或链接。文中描述了所采用的关键算法流程和主要实现技术,并编写了测试程序对 RT-AGC 的回收效果进行验证。实验证明可以有效地抑制应用程序的内存泄漏。

关键词:Windows 系统;动态内存分配;内存泄漏;垃圾回收;堆内存
中图分类号:TP399 **文献标识码**:A **文章编号**:1673-629X(2012)01-0017-04

A Garbage Collector for Windows Applications

MIAO Hong-wei[1],ZHONG Yun-peng[2]
(1. Microsoft . net Research Lab,Tianjin University,Tianjin 300072,China;
2. Dept. of Control Engineering,Academy of Armored Force Engineering,Beijing 100072,China)

Abstract:It proposes a garbage collector (RT-AGC) to resolve memory leaks in long running Windows applications. RT-AGC uses Mark-Sweep algorithm and makes it have more flexibility by improving the trigger conditions. This collector detects and reclaims garbage memory for non-cooperative applications at run time using PE file structures and Windows API functions by scanning program space. Key algorithm and implementation for this collector is presented. A defective test application is written for verifying the effect of RT-AGC. Experiment proves that the collector can prevent memory leaks effectively.

Key words:Windows;dynamic memory allocation;memory leaks;garbage collection;heap storage

0 引 言

堆内存泄漏是一类内存管理问题,指已分配的堆内存没有得到正确地释放。持续的内存泄漏会严重地降低程序的性能,甚至导致程序崩溃。

在 Windows 下,很多需要长期、大量的动态申请内存的应用程序由 C/C++实现。C/C++语言要求程序员管理动态内存的分配与回收,因此,不可避免的会出现一定程度的内存泄漏。针对这类程序,文中设计并实现了一个基于 Windows 平台应用程序的运行时自动垃圾回收器(RT-AGC),RT-AGC 对运行中的应用程序进行检测并回收已失去管理的堆内存,而不需要目标应用程序的合作。

收稿日期:2011-06-07;**修回日期**:2011-09-15
基金项目:天津市科技支撑计划重点项目(10ZCGYSF01300)
作者简介:苗宏伟(1984-),男,吉林洮南人,硕士研究生,主要研究方向为系统安全、网络信息安全;导师:许林英,副教授,主要从事计算机软件工程和网络信息安全方向的教学与研究。

1 垃圾回收算法

在描述算法之前,首先明确以下几个定义:

定义1 活动指针:程序空间中的指针变量,且指向已分配的堆内存。

定义2 活动堆内存:已分配的堆内存,且通过当前的活动指针可以直接或间接访问。

RT-AGC 所采用的算法核心是 Mark-Sweep 技术[1]的一个变种。与传统的 Mark-Sweep 算法不同,文中的算法既不是等到堆内存耗尽才进行回收操作,也不是即时回收,而是满足以下两个条件之一时触发回收操作:

条件1 阈值触发条件:当已分配的堆内存总数达到或超过某一预设的阈值;

条件2 周期触发条件:当目标应用程序运行时间达到某一预设的回收周期。

除此之外,算法与传统的 Mark-Sweep 算法[1]基于同一规则:遍历活动、可写的程序空间,以标记活动堆内存,其它未被标记的堆内存视为垃圾内存,可被安全

的回收。

Mark-Sweep 算法分为 Mark 和 Sweep 两个阶段，Mark 阶段将所有已分段的堆内存段划分为两个集合：活动集和不可达集，活动集中的元素会被标记，不可达集表示已经失去管理的堆内存，可以被视为垃圾进行回收。Sweep 阶段回收所有未标记的堆内存段。

回收操作由 RT-AGC 根据触发条件调度，算法如下：

```
PROCUDERE collector( )
BEGIN
WHILE RT-AGC 正在运行 DO
阻塞以等待回收操作条件
中止目标应用程序所有线程
执行回收操作
恢复所有线程
END
```

collector( )运行在目标应用程序的一个独立线程中，在不满足回收条件时，线程处于阻塞状态；当触发条件满足时，RT-AGC 首先将目标应用程序中所有的线程挂起，然后执行回收操作。最后再将所有线程恢复使目标应用程序继续执行。回收线程继续阻塞以等待被触发。

两个触发条件中，如果当前已分配的堆内存大小超过阈值，且不能通过回收操作使其减小到阈值以下，则阈值触发条件暂时失效，而由回收周期单独触发回收操作。

1.1 Mark 算法

Mark 阶段的算法基于保守垃圾回收规则[2~4]。精确的垃圾回收需要有编译器地协助，即垃圾回收器可以确定程序空间中各对象的布局；与此相反，保守的垃圾回收器不能从编译器获得这些信息，因此必须遍历程序空间，以获得对象的可能布局。

另外，传统的回收器为了标识活动指针和活动堆内存信息，需要程序员在分配堆内存时遵循某种协议，或者重新链接目标应用程序以替换原有的内存分配库函数。文中的 Mark 算法对此进行了改进：RT-AGC 作用于运行中的目标应用程序，不需获得目标程序的源代码，也不需要对目标程序重新编译或连接。

算法对目标应用程序作以下假设[5]，如果目标应用程序中假设不成立，则 Mark 算法将失败：

a)活动指针存储于寄存器或内存中；

b)活动指针在内存中的存储方式都是按字对齐。

根据以上假设，算法只需扫描目标应用程序的寄存器、全局数据区、栈区和活动堆内存。同时因为算法在进行回收操作前会暂停目标应用程序地运行，所有正在使用的寄存器中的值会被保存至栈空间，因此，Mark 算法只需扫描程序全局数据区，各个线程的栈空间以及活动堆内存，算法如下所示：

```
PROCEDURE  mark_active_heap_segment( )
BEGIN
①FOR 栈区中的所有字 DO
node = BTree_Search( word )
IF 找到结点 node THEN
标记结点 node
scan_heap_segment( node )
②FOR 全局数据区中的所有字 DO
node = BTree_Search( word )
IF 找到结点 node THEN
标记结点 node
scan_ heap_segment( node )
END
```

如前所述，保守垃圾回收假设程序空间中的每个字都是潜在的指针，因此需遍历整个程序空间中可能存在指针变量的地方，以标识活动指针。代码中循环①和②分别按字扫描目标进程的栈空间和全局数据区，在每一次循环中，算法首先判断当前字中所包含的值是否为活动指针，如果结果为真，则找到它所指向的堆内存并对其调用 scan_heap_segment 函数，在活动堆内存中递归的扫描内部活动指针。

判断一个值是否为活动指针的工作由函数 BTree_Search 完成，对于每一块已分配的堆内存，RT-AGC 都为其维护一个结点，结点包含这块内存的起始地址、大小、标记域等信息。这些结点又构成一棵平衡排序二叉树，BTree_Search 函数利用平衡二叉树来判别活动指针[6]。

指针的判断过程分为两个步骤：首先判断当前字所包含的值是否在已分配的堆空间地址范围内，此阶段只是简单地将值与已分配堆内存首地址的最小值和最大值进行比较，如果结果为真，则认为当前值为“潜在指针”，否则可以认定非活动指针。这一步骤可以淘汰大部分非活动指针的字；第二步在平衡排序二叉树中检索该“潜在指针”，如果找到对应的结点，则返回结点，否则返回空值。

算法在遍历程序空间的时候，可能会因为巧合将某个不含活动指针的字误识别为指针，即“指针标识错误”[4]，发生这种错误时，回收器可能误将已失去管理的堆内存段当作活动堆内存予以保留，但不会将活动堆内存回收而造成目标程序运行错误，同样也不会对回收器的性能造成影响。

目标应用程序的活动堆内存中可能包含有指针变量指向其它的活动堆内存，要标识所有的活动堆内存，对程序空间的遍历应该是递归过程。给定一块活动堆

内存,scan_heap_segment 函数在其上递归的遍历内部指针。

1.2 Sweep 算法

在算法的 Sweep 阶段,算法使用一个栈结构辅助完成回收操作:遍历前文提到的平衡排序二叉树,将所有已标记结点的标记位复位,将未作标记的结点压入栈中,并从二叉树中删除。当遍历结束,将栈中的结点一一弹出并回收其对应的空间,算法伪码如下所示。其中以 SA_为前缀的一组函数代表对辅助栈的操作,包括元素入栈,出栈以及判断栈是否为空等。

```
PROCEDURE  sweep_heap_segments( )
BEGIN
FOR 二叉树中的所有结点 DO
IF 结点被标记 THEN
取消标记
ELSE 结点入栈
WHILE 辅助栈不为空 DO
弹出一个结点
回收垃圾内存
END
```

2 RT-AGC 的实现

根据上一节描述的算法,要完成回收操作,RT-AGC 需要获得以下几项信息:

a) 目标应用程序中栈空间的地址范围;

b) 目标应用程序中全局数据区的地址范围;

c) 目标应用程序中已分配的堆内存信息(包括起始地址,大小等)。

以上信息的获取是高度平台相关的。在 Windows 下,这些信息可以通过分析 PE 文件结构和调用 Windows API 函数来获取[7]。

另外,由于目标应用程序运行在自己独立的进程空间中,很多操作难以通过进程间的通信实现。因此,RT-AGC 在运行时将远程线程和动态链接库(DLL)注入目标程序,以获得操作权限[8~10]。

RT-AGC 采用 C 语言实现,运行在 Windows 2000 及以上版本操作系统。

2.1 栈空间地址范围

在 Windows 应用程序中,一个进程里可能有多个线程并发执行。每个线程都有自己独立的栈空间。RT-AGC 需枚举目标应用程序中的所有线程,分别获得其栈空间。

Windows API 函数 VirtualQuery 提供一段内存区域的页面信息,原型如下:

DWORD VirtualQuery (LPVOID lpAddress, PMEMORY_BASIC_INFORMATION lpBuffer, DWORD dwLength)

Windows 给应用程序分配内存时是以页为单位的,每个内存页面具有各自的属性。VirtualQuery 函数给出 lpAddress 这个地址所在的页面以及与它相邻的具有相同属性的页面信息,这些信息由 MEMORY_BASIC_INFORMATION 结构返回,包括页面的首地址和长度等[8]。将目标应用程序的线程栈空间中某一地址作为 lpAddress,利用 VirtualQuery 函数,可以获得与线程栈空间内存属性相同的内存块信息。这个内存块至少包含线程栈空间的地址范围。

文中利用栈顶指针寄存器 ESP 取得栈空间中某一地址,ESP 指向当前线程的栈顶,当发生线程切换或挂起,Windows 会将当前 ESP 连同其它线程上下文内容保存至 CONTEXT 结构[8],利用 API 函数 GetThreadContext 可以读取线程 CONTEXT,原型如下所示:

BOOL GetThreadContext (HANDLE hThread, LPCONTEXT lpContext);

其中 hThread 参数指定线程句柄,lpContext 返回线程上下文 CONTEXT 结构,其中包含寄存器 ESP 的内容。函数成功执行返回 true,否则返回 false[8]。

2.2 全局数据区地址范围

Windows 中,可执行文件都是 PE 文件格式。PE 文件被组织为一个线性的数据流。每个 PE 文件包含多个段,段是一块连续的内存,没有大小限制,包含可执行代码或者数据。同时每个段都有自己的属性,例如 MEM_READ(读)、MEM_WRITE(写)等。PE 文件为它所包含的所有段保存一张段表,段表中的每项代表一个段,保存该段的类型、性质、大小和地址等信息[11]。

应用程序的全局变量和静态局部变量的地址在程序编译时即已经分配,存储在 PE 文件的段中[11]。基于保守回收规则,文中遍历所有同时具有可读写属性的段查找全局/静态活动指针。

2.3 已分配堆内存信息

在 Windows 下,应用程序可以通过 C 运行时(CRT)函数或 API 函数分配/释放堆内存。调用层次[12]如图 1 所示。

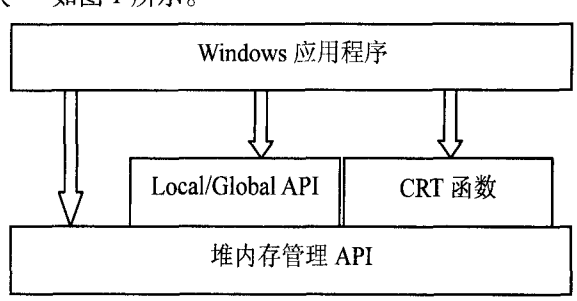

图 1　堆内存服务程序调用层次

要获得目标应用程序中已分配的堆内存信息,RT

-AGC 拦截目标程序对堆内存管理服务程序的调用，需要拦截的函数如表1所示。

表1 堆内存服务程序列表

CRT 函数	Window API 函数
malloc/free	HeapAlloc/HeapFree
realloc/free	GlobalAlloc/GlobalFree
calloc/free	LocalAlloc/LocalFree

以上函数的实现代码都在动态链接库中，由所有 Windows 应用程序共享。当应用程序中出现对以上函数的调用，连接器会将函数所在的动态链接库导入该应用程序空间。当应用程序调用一个来自其它模块的函数，由编译器发出的调用指令不会将控制直接转移到该模块中的函数，而是利用一条 JMP 指令跳转到真实的函数入口点。函数入口点的地址保存在 PE 文件的.idata 段中。

RT-AGC 修改目标应用程序.idata 段中的函数地址，将需要被拦截的服务函数地址指向托管函数，在托管函数中执行信息的收集工作之后，再调用实际的服务函数，以获得已分配的堆内存信息。

3 RT-AGC 的运行测试

为了测试 RT-AGC 的运行效果，编写了测试目标应用程序 TestTargetAppl：TestTargetAppl 使用 C 语言编写，每隔固定时间间隔申请堆内存但从不释放。活动指针分布于全局数据区、栈区和堆空间。在以下两种情况下运行 TestTargetAppl 并查看其内存使用情况：

1. TestTargetAppl 在没有 RT-AGC 干预的情况下独立运行：内存使用量随时间呈线性增长，其增长率与每次申请的堆内存大小相关，说明 TestTargetAppl 发生了严重的内存泄漏；

2. 在 RT-AGC 干预的情况下，TestTargetAppl 的内存使用量在经过一段时间的线性增长后，逐渐趋于稳定。

以上实验说明 RT-AGC 虽然不能完全清除内存泄漏情况，但是可以有效地抑制内存泄漏。这一点在需要长期、大量的申请堆内存的应用程序上表现的尤为明显。

4 结束语

文中描述了一种在 Windows 下应用程序垃圾内存的回收器的设计与实现。回收器采用 Mark-Sweep 算法，其中关键信息的获取通过分析 PE 文件结构和调用 API 函数实现。最后通过实验证明了其有效性。

但是文中只是从较高层对 Windows 应用程序的垃圾回收器的设计与实现进行描述，尚有一些细节问题需要考虑：文中实现的 RT-AGC 无法有效的作用于已加壳的应用程序；另外，对于应用程序中由其它模块分配的堆内存，RT-AGC 无法进行拦截和回收。在这两方面进行改进将是下一步工作。

参考文献：

[1] Jones R, Lins R. Garbage Collection: Algorithms for Automatic Dynamic Memory Management[M]. [s. l.]: John Wiley & Sons, 1996.

[2] Boehm H J, Weiser M. Garbage collection in an uncooperative environment[J]. Software Practice and Experience, 1988, 18(9): 807-820.

[3] Wentworth E P. Pitfalls of conservative garbage collection[J]. Software Practice and Experience, 1990, 20(7): 719-727.

[4] Willard B, Frieder O. Autonomous garbage collection: resolving memory leaks in long-running server applications[J]. Computer Communications, 2000, 23(10): 887-900.

[5] Aho A V, Sethi R, Ullman J D, et al. 编译原理[M]. 李建中, 译. 北京: 机械工业出版社, 2003.

[6] Horowitz E, Sahni S, Anderson-Freed S, et al. 数据结构(C语言版)[M]. 李建中, 译. 北京: 机械工业出版社, 2006.

[7] Michael G. Easy Detection of Memory Leaks[EB/OL]. 2005-08-04[2011-04-26]. http://www.codeproject.com/KB/cpp/MemoryHooks.aspx.

[8] 杰夫瑞, 克里斯托夫. Windows 核心编程[M]. 葛子昂, 周靖, 廖 敏, 译. 北京: 清华大学出版社, 2008.

[9] Robert K. Three Ways to Inject Your Code into Another Process[EB/OL]. 2003-08-21[2011-04-20]. http://www.codeproject.com/threads/winspy.asp.

[10] 王 峥, 娄渊胜. 远程线程注入技术在监控系统中的应用[J]. 计算机技术与发展, 2010, 20(3): 207-210.

[11] Pietrek M. An In-Depth Look into the Win32 Portable Executable File Format[EB/OL]. 2002-02[2011-05-10]. http://msdn.microsoft.com/en-us/magazine/cc301805.aspx.

[12] Randy K. Managing Heap Memory[EB/OL]. 1993-04-03[2011-05-03]. http://msdn.microsoft.com/en-us/library/ms810603.aspx.

(上接第16页)

[9] 王艳慧. 基于 CMM 的软件过程改进实践[J]. 计算机技术与发展, 2008, 18(5): 141-143.

[10] 龚 波. 能力成熟度模型集成及其应用[M]. 北京: 中国水利水电出版社, 2003.

[11] 李红梅. 软件项目跟踪与监控[J]. 科技论坛, 2005(17): 17-17.

[12] 邓治文, 商惠华, 戴汇川. 基于需求分析的软件质量目标策划方法[J]. 微计算机信息, 2010, 26(18): 187-188.