·计算机工程与应用·

引用计数与时间戳的混合垃圾搜集器算法

张　宁，熊光泽

(电子科技大学计算机学院　成都　610054)

【摘要】实时垃圾搜集器(GC)的任务是及时回收系统中无用内存，并保证实时任务不超过其时限，在此基础上还应尽可能降低系统内存需求。提出了引用计数与时间戳的混合GC算法，用引用计数算法回收非循环垃圾，用时间戳算法回收循环垃圾，并参与系统并发调度。不仅能回收全部无用内存，与标记清除混合GC相比还可进一步降低系统内存需求，该算法适用于大规模系统。

关 键 词　算法；　垃圾搜集器；　内存需求；　实时；　时间戳

中图分类号　TP316.2;　V557^{+}.1　　**文献标识码**　A　　doi:10.3969/j.issn.1001-0548.2010.04.024

Hybrid Garbage Collection of Reference Counting and Timestamp

ZHANG Ning and XIONG Guang-ze

(School of Computer Science and Engineering, University of Electronic Science and Technology of China　Chengdu　610054)

Abstract　Real-time garbage collection should collect unused memory and guarantee real-time tasks to meet their deadlines. Furthermore, system memory requirement should be considered. A hybrid garbage collection is proposed in this paper. Reference counting algorithm is used to collect acyclic garbage and timestamp algorithm is used to collect cyclic garbage. GC and real-time tasks are scheduled concurrently. The proposed GC not only collects all unused memory but also reduces memory requirement compared with hybrid GC based on mark-sweep algorithm. It fits for large-scale system.

Key words　algorithms;　garbage collection;　memory requirements;　real-time;　timestamp

近年来，高级程序开发语言JAVA[1]和C#被广泛应用，其中一个重要原因在于它们都采用了垃圾搜集器(GC)自动回收无用内存空间，可避免传统开发语言(如C语言)采用人工管理内存方式所导致的内存泄露、指针悬挂、内存碎片等潜在危险，极大地提高了系统的安全可靠性。随着嵌入式实时系统的日益规模化及复杂化，很多研究着重于如何把垃圾搜集器的优点应用于嵌入式实时系统的开发。

传统的GC在执行时需要暂停系统的所有任务，显然不适合于实时系统。文献[2]提出了渐进式GC，即基于工作的GC，把GC任务分成很多小片与实时任务交替执行，大大降低了任务暂停时间。然而，一段集中的内存操作可能使大量GC片的时间叠加，导致任务超过其时限。因此文献[2]的算法只适合于软实时系统。为解决该问题，近年来出现了从基于工作的GC到基于时间的GC[3-5]的转变，GC任务被作为一个单独的实时任务参与调度，可保证系统内实时任务的硬时限要求。此外，由于嵌入式实时系统往往应用于资源有限的环境中，因此在保证任务实时性的条件下，使系统内存需求量降低成为一个重要的课题[6-8]。

传统GC算法主要有引用计数算法[9]和追踪类算法[10]，追踪类算法中最常用的是标记清除算法。引用计数算法和追踪类算法两者各有优缺点：(1) 引用计数算法简单易行，易于并发调度，并且回收的内存可以及时供系统重用，最大的缺点在于无法回收循环垃圾；(2) 标记清除算法可以回收所有垃圾包括循环垃圾，缺点是不适合于大规模系统，所有无用内存只能在整个GC循环结束时一次性回收供系统重用，导致系统需要较多的冗余内存。有人提出了结合两者优点的混杂GC算法[11]，用引用计数算法回收非循环垃圾，而用标记清除算法回收循环垃圾，既能保证所有垃圾都被回收，又可以降低系统内存需求。然而，对于大规模系统，所有循环垃圾仍然

收稿日期：2008－11－20;　修回日期：2009－12－13
基金项目：国家863计划(2006AA01Z137)
作者简介：张　宁(1975－)，男，博士，主要从事嵌入式实时系统、动态内存管理方面的研究.

需要保留至GC循环，最后才被回收。本文提出用引用计数和时间戳算法相混杂的GC算法，可以在GC循环中及时回收循环垃圾供系统重用，从而降低系统的内存需求。

1 算法描述

1.1 引用计数

所有垃圾搜集器的判断方法是，在系统中有一组根对象，任何从根对象可到达(即被根对象引用或间接引用)的对象都是活动对象，而不能从根对象到达的对象都是垃圾对象，应被回收。

引用计数算法要求每个对象头部保留一个值，记录该对象被其他对象引用的数值，当指向该对象的指针被删除时，则该计数值减1，如果该计数值为0，说明该对象没有被任何其他对象引用，则该对象成为垃圾被系统回收。该算法不仅易于实现，可以迅速回收垃圾对象所占用的内存供系统重用，更重要的是它适合于与实时任务并发运行。但是，该引用计数算法的一个最大的缺陷在于其无法回收循环垃圾。

如图1所示，当根对象指向对象*O*的指针r_1被删除时，对象*A*、*B*、*O*都成为垃圾应该被回收，然而该图显示O的引用计数值永远不会变为0，因此单纯的引用计数算法无法回收循环垃圾。

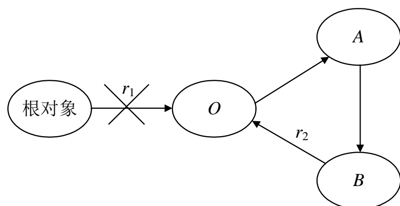

图1 循环垃圾示例

1.2 时间戳算法回收循环垃圾

标记清除算法也可以回收循环垃圾。如图1所示，当指针r_1被删除后，标记清除GC启动，从根对象开始把所有被引用的对象标记为黑色，而没有被根对象引用的对象标记为白色，因此*O*、*A*、*B*均为白色，标记阶段结束后回收所有未被根对象引用的对象即白色对象，因此循环垃圾也可被标识并回收。然而，该算法需要扫描标记系统的全部对象，当系统规模较大时，需要占用较长时间，所有内存垃圾只有在GC循环结束时才会被统一回收重用，在回收前它们成为浮动垃圾占用系统冗余空间。

为解决标记清除算法的缺陷，本文采用时间戳算法回收循环垃圾。每个对象头部不仅保留引用计数值，还保留一个时间戳，其初始值等于对象的生成时间，即第一个引用指向对象的时间。系统还为每个对象保留一张表，记录指向对象的所有引用和引用时间。例如对图1中的对象*O*，有如图2所示的引用表。

引用	引用时间
r_1	Time_r_1
r_2	Time_r_2

图2 对象引用表

如果在任务运行中某对象的引用被删除，则表中相应的记录也被删除，新指向某对象的引用也会相应地在引用表中被增加，且其引用时间为当前时间。如图3所示，当新增加引用r_3时，删除r_1，对象*O*的引用表为((r_2,Time_r_2),(r_3,Time_r_3))，其中Time_r_3为当前时间，必然大于Time_r_2。即新增加引用的时间总是大于以前引用的时间。

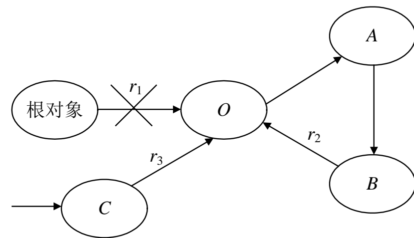

图3 增加及删除引用图

从图中可以看出，循环垃圾的形成总是由删除老的引用所引起的，如图1中Time_r_2<Time_r_1，当新引用r_2被删除时，*A*、*B*和*O*仍然都是活跃的，而当老引用r_1被删除时，3个对象都变成垃圾。因此，当对象的引用被删除时，系统比较该引用的时间戳是否为最小，如果是则说明有可能形成循环垃圾，需启动判断。而如果该引用的时间戳不是最小，则说明该引用不是最老的引用，不会形成循环垃圾，则不需启动判断。

当时间戳最小的引用即最老引用被删除时，GC启动判断，判断是否产生循环垃圾。判断方法可以使用局部标记清除算法依次扫描[12]，也可以使用计算循环外部引用计数算法[13-14]进行扫描。前者可以回收全部循环垃圾，但需要对每个对象扫描3次，复杂度较高，后者的优点是只需要对每个对象扫描一次，但完备性不足，只能回收部分循环垃圾。

需要注意的是，在扫描某对象时，应当同时更新对象引用表，使其时间为当前扫描时间，否则如果有新增引用r_3后，r_1被删除，当r_3又被删除形成循环垃圾时，由于r_3为最新引用，会导致GC无法正确启动判断，无法回收循环垃圾，如图3所示。因此，当r_1被删除，启动GC判断时，在扫描过程中需要更

新各对象引用的引用时间为当前时间，扫描结束时，r_3即为最老引用，当r_3被删除时，GC就可以正常启动判断。

2　内存分析

本文采用基于时间的策略，GC作为一个单独的任务和实时任务并发调度，调度必须保证所有实时任务都在时限内完成，并且系统内存不会被耗尽。考虑系统中有一组实时任务$M=\{M_1,M_2,\cdots,M_i\}$，H为内存堆大小，F为系统中比GC更高优先权的任务所预留内存量，$L_{\max}$为系统中所有活动内存量的上限，CGG_i和$\mathrm{CGG}_{\max}$分别是GC任务的第i次循环中系统所产生的全部循环垃圾及其最大值，MGG_i是任务M_i产生的循环垃圾中最大循环所占的内存量。

引用计数算法可以及时回收非循环垃圾，并使其立刻被系统重用，因而不会造成浮动垃圾而占用系统冗余内存。对于用全局标记清除的混杂GC算法，全部循环垃圾要保留到GC循环结束才能被回收，系统所需内存不会超过$L_{\max}+\mathrm{CGG}_{\max}$。此外，为了保证GC任务和系统实时任务的同步，在系统开始时，系统内存H中必须预留$F+\mathrm{CGG}_{\max}$的内存空间，因此可得：

$$F+\mathrm{CGG}_{\max}=H-L_{\max}-\mathrm{CGG}_{\max} \tag{1}$$

满足式(1)意味着在最坏情况下，系统所保留的冗余内存都能满足系统任务的分配需求。由式(1)可得：

$$H=2\mathrm{CGG}_{\max}+F+L_{\max} \tag{2}$$

为便于分析，本文在调度中使GC优先权最高，则F为零，而$L_{\max}$为固定值，因此系统内存需求H只取决于$\mathrm{CGG}_{\max}$。如果采用本文的基于时间戳的混杂GC算法，则循环垃圾也可局部回收，并使其内存空间及时为系统重用，则系统只需保留$\mathrm{MAX}(\mathrm{MGG}_i)$的冗余空间，即：

$$H=2\mathrm{MAX}(\mathrm{MGG}_i)+F+L_{\max} \tag{3}$$

显然$\mathrm{MAX}(\mathrm{MGG}_i)$远小于$\mathrm{CGG}_{\max}$，因此采用本文的混杂GC算法可进一步降低内存需求。在具体的实时调度中，由于GC的调度及运行情况不同，各任务的内存分配和其中循环垃圾量的不同，与离线分析相比，系统可能需要更多的冗余内存以保证某些额外开销，但不会改变本文的结论。

3　仿真实验

本文对基于时间戳的混杂GC算法进行仿真实验，使用如表1所示的任务集。表中，C_i为周期任务M_i的执行时间；T_i为任务的周期，其值等于任务的时限D_i；A_i为任务在一个运行周期中分配的最大内存量；α_i为活动内存所占任务总分配内存的最大百分比；cgg_i为任务的每次循环中所产生的最大循环垃圾量。使用Bacon的固定时间间隔的GC调度策略[5]，一个完整的GC循环执行时间为3 ms，GC开始运行时每4 ms执行一个GC片段，时间为1 ms，共3个GC片段执行完后，一个GC循环结束，GC周期为12 ms。

表1　任务集

任务	C_i/ms	$T_i(D_i)$/ms	A_i/Byte	α_i	cgg_i/byte
M_1	1	5	320	0.53	75
M_2	2	10	960	0.46	260
M_3	5	50	1 920	0.38	595
M_4	12	120	3 120	0.57	671

为了方便仿真并且不妨碍验证结论，作如下假设：任务之间相互独立，没有阻塞存在；上下文切换的调度开销忽略不计；任务内存分配量均匀，即分配内存量和其执行时间成正比，并与产生的内存垃圾量也成正比；在GC的时间片内都能正常执行完相应的GC任务。

仿真结果如图4所示。图中，由点构成的线表示使用纯粹的标记清除算法(mark-sweep)下系统的内存变化；虚线表示引用计数与标记清除混杂GC算法下系统的内存变化；实线表示本文的引用计数与时间戳混杂GC算法下系统的内存变化。

可以看出，在纯粹的标记清除算法下，全部无用内存都要等待GC循环一次，结束时统一回收，因此系统内存需求最大。引用计数与标记清除混杂GC算法下，每个GC工作片段都可以及时回收非循环垃圾，但循环垃圾在GC循环结束时才统一回收，因此系统内存需求比纯粹的标记清除算法下降很多。而如果采用本文的基于时间戳的混杂GC算法，不仅非循环垃圾可以被及时回收，而且循环垃圾也可由时间戳算法及时回收，浮动垃圾减少，系统内存需求进一步降低。与标记清除混杂GC算法相比，基于时间戳的混杂GC算法在最坏情况下也可以使系统内存需求降低约4.5%。

内存需求与实时任务的内存量及循环垃圾在其中所占的比重有关。图中，每隔12 ms，3种算法的图线重合，这是因为GC的完整周期为12 ms，当一个GC周期结束时，所有无用内存都会被回收。

与以往的GC算法相比，本文的基于时间戳的混杂GC算法在保证系统实时任务时限及内存不被耗

尽的基础上，主要有两方面优点。

(1) 可以减小系统内存需求。由于引用计数算法回收系统中的非循环垃圾，以及时间戳算法及时判断并回收系统中的循环垃圾供系统重用，因此系统不需要过多的冗余内存保存浮动垃圾。

(2) 适合于对象较多的大规模复杂系统。传统的标记清除算法必须在全局标记所有对象，整个GC循环结束时再统一回收无用内存，对于大规模系统，GC的执行时间过长，可能导致内存耗尽，而基于时间戳的算法可以局部回收循环垃圾，其执行时间与系统对象数无关，因此适合于大规模系统。

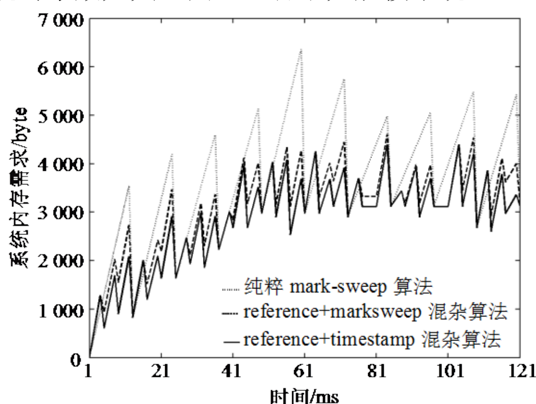

图4 基于时间戳的混杂GC与纯标记清除GC及标记清除混杂GC的系统内存需求对比

4 结 论

本文提出引用计数和基于时间戳算法相结合的混杂GC策略。通过对算法的描述和内存分析，以及相应的仿真实验，可知，与基于标记清除算法的混杂GC相比，该算法由于可以在GC循环中局部回收循环垃圾并及时供系统重用，因而可以进一步减小系统内存需求。该算法的执行时间与系统全局对象数无关，因此适用于大规模系统。

然而，该算法也有其缺点，即算法比较复杂，本文在分析时对任务调度开销、任务独立性及无堵塞等情况作了假设。以后的工作将着重于减少这些假设条件，并实现一个基于时间戳的混杂GC模型。

参 考 文 献

[1] WU Yue, WU Jing, ZHOU Ming-tian. Research on event handling models of Java[J]. Journal of Electronic Science and Technology of China, 2004, 2(2): 42-47.

[2] BAKER H G. List processing in real time on a serial computer[J]. Communications of the ACM, 1978, 21(4): 280-294.

[3] KIM T, CHANG N, KIM N, et al. Joint scheduling of garbage collector and hard real-time tasks for embedded applications[J]. Journal of Systems and Software, 2001, 58(3): 247-260.

[4] HENRIKSSON R. Scheduling garbage collection in embedded systems[D]. [S.l.]: Lund University, 1998.

[5] BACON D F, CHENG P, RAJAN V T. A real-time garbage collector with low overhead and consistent utilization [C]//Conference Record of the Thirtieth Annual ACM Symposium on Principles of Programming Languages. New Orleans, Louisiana, USA: ACM Press, 2003: 285-298.

[6] KIM T, CHANG N, SHIN H. Bounding worst case garbage collection time for embedded real-time systems [C]//Proceedings of the Sixth IEEE Real Time Technology and Applications Symposium (RTAS 2000). Washington, DC, USA: IEEE Press, 2000: 46-55.

[7] ZHANG Ning, XIONG Guang-ze. Minimizing GC work by analysis of live objects[J]. ACM Sigplan Notices, 2006, 41(3): 20-29.

[8] XIAN Yu-qiang, XIONG Guang-ze. Minimizing memory requirement of real-time systems with concurrent garbage collector[J]. ACM SIGPLAN Notices, 2005, 40(3): 40-48.

[9] CHRISTOPHER T W. Reference count garbage collection [J]. Software Practice and Experience, 1984, 16(4): 503-507.

[10] McCARTHY J. Recursive functions of symbolic expressions and their computation by machine[J]. Communications of the ACM, 1960, 3(4): 184-195.

[11] YANG Chang, WELLINGS A. Hard real-time hybrid garbage collection with low memory requirements [C]//Proceedings of 27th IEEE International Real-Time Systems Symposium (RTSS). Washington, D C, USA: IEEE Press, 2006: 77-86.

[12] BACON D F, RAJAN V T. Concurrent cycle collection in reference counted systems[C]//Proceedings of 15th European Conference on Object-Oriented Programming (ECOOP '01). Budapest, Hungary: Springer-Verlag, LNCS 2072, 2001: 207-235.

[13] LIN Chin-yang, HOU Ting-wei. A lightweight cyclic reference counting algorithm[C]//Proceedings of the first International Conference on Grid and Pervasive Computing. Taichung, Taiwan, Springer-Verlag, LNCS 3947, 2006: 346-359.

[14] LIN Chin-yang, HOU Ting-wei. A simple and efficient algorithm for cycle collection[J]. ACM Sigplan Notices, 2007, 42(3): 7-13.

编 辑 蒋 晓