ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2010/04(01)-0029-07
DOI:10.3778/j.issn.1673-9418.2010.01.003

E-mail:fcst@public2.bta.net.cn
http://www.ceaj.org
Tel:+86-10-51616056

内存泄漏检测工具与评估方法*

李 倩+,潘敏学,李宣东
南京大学 计算机科学与技术系,南京 210093

Benchmark of Tools for Memory Leak*

LI Qian+, PAN Minxue, LI Xuandong
Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China
+ Corresponding author: E-mail: qianjie@seg.nju.edu.cn

LI Qian, PAN Minxue, LI Xuandong. Benchmark of tools for memory leak. Journal of Frontiers of Computer Science and Technology, 2010,4(1):29-35.

Abstract: Memory leak, which is a substantial source of errors in software systems, consumes system memory, degrading performance and eventually resulting in program crashes. There are two kinds of tools for detecting memory leaks, which are static tools based on program static analysis technique, and dynamic tools recording allocation of heaps during program running. So far, benchmarks for evaluating their ability are still nonexistent. An approach to build this benchmark is proposed, starting from comprehension and recognition of memory leak and ability of those detecting tools.

Key words: memory leak; dynamic tools; static tools; benchmark

摘 要: 内存泄漏是软件系统中常见的一种错误,会持续消耗内存,致使系统运行效率下降,甚至导致系统崩溃。内存泄漏的检测工具主要可以分为两类:一类是使用基于程序扫描分析技术的静态工具;另一类则是监视实时内存分配状态进行判别的动态工具。如何评估工具检测内存泄漏的能力,相关的标准并不明确。通过对内存泄漏的认识与了解,对相关工具能力进行了调研与分析,提出了一个内存泄漏工具的评估标准。

* The National Natural Science Foundation of China under Grant No.60603036 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2007AA010302,2009AA01Z148 (国家高技术研究发展计划(863)); the Natural Science Foundation of Jiangsu Province of China under Grant No.BK2007139 (江苏省自然科学基金).
Received 2009-05, Accepted 2009-07.

关键词:内存泄漏;动态工具;静态工具;评估方法
文献标识码:A **中图分类号**:TP333

1 引言

随着计算机硬件性能的不断提高,计算机能为人们处理更多复杂的问题。然而软件可靠性的发展并没有跟上,软件本身存在的缺陷频繁地造成系统出错,成为计算机处理问题能力提升的桎梏。怎样去提高软件的可靠性,减少软件存在的缺陷已经成为软件开发生产中的一个必不可少的环节。特别是安全敏感的项目,系统出错可能导致致命失败,软件的可靠性成为整个系统的关键属性。

如何去提升软件可靠性?这项工作贯穿整个软件开发过程。其中,最重要的工作集中在测试与验证阶段。程序员利用可以获得的工具对程序进行测试与分析,期望发现程序中存在的所有错误并进行修正。理想情况下,由检测结果定位到程序中的错误,继而修复,同时避免引入新的错误,这项工作需要耗费大量的时间与精力。特别是测试工作,为了检测到小概率路径中存在的错误,需要运行大量测试用例,可能需要花费几天或者几星期才能完成。每一个软件开发团队使用的检测方法不尽相同,付出的时间与精力也不一样,怎样去恒量所使用的方法产生的结果?换句话说,为了提高软件的可靠性所进行的工作效果评估标准应该是什么?使用这些标准,一方面,可以指导检测人员根据系统的需求,以及开发条件,选择合理有效的方法来提高软件的可靠性,另一方面,也可以为软件的可靠性打分或者评级。这是一个复杂又有挑战性的工作。程序中可能存在的缺陷种类繁多,针对各种缺陷的方法与工具也各有不同,深入了解各类错误在程序中的表现以及成因,综合分析各种工具的能力,总结出评估标准,是一个可行的方法。

这里选择内存泄漏作为代表,研究内存泄漏的成因、分类,综合评估如今比较流行的内存漏泄的专项工具,最后给出了针对内存泄漏这一缺陷,程序员使用的检测手段在程序可靠性这一项上的评估标准。

文章安排如下:第2章介绍了评估标准建立的过程与方法;第3章介绍了内存泄漏的基本概念以及相关工具;第4章是实验及评估结果;第5章对研究工作进行总结。

2 评估标准建立的过程与方法

软件中最常使用的标准是系统的性能标准。性能标准从起初直观的速度比较,发展至今,已经细化到系统性能特性和领域。Standard Performance Evaluation Corporation(SPEC)制定的性能标准涵盖了CPU、高性能计算、e-mail、Web服务等各个领域[1]。这些性能评估标准已经广泛应该于系统的开发与软件销售中,衡量和比较各个不同的系统的性能表现,给客户在选择系统时提供参考。

然而,单纯的性能比较不能完整地评估系统。在安全敏感的核心系统中,可靠性成为决定系统优劣的关键属性。不同于性能标准,可靠性标准的发展仍处于探索阶段。近十年来,可靠性标准研究取得了很大的进步,但是,对于可靠性的定义,以及标准的形式,仍然没能统一。

Debench[2]项目中提出了一个可靠性标准制定的框架。其中,可靠性标准指的是该系统通用的,用来描述和比较系统在一定的工作负荷下,引入可能的错误负荷时的表现方法。这些可靠性标准的制定参考了性能标准制定的过程,采用实验和模型模拟相结合的方式,定性和定量地给出系统的评分。

该文提出的可靠性标准从另一角度衡量了系统的可靠性。所谓可靠系统,在设计与实现时,本身携带的显示或者隐藏的缺陷应该是越少越好。基于这种认知,定义系统可靠性为系统摆脱缺陷的程度。如何标准化系统摆脱缺陷的程度呢?直观地,理想情况下,如果系统绝对不存在缺陷,那么它摆脱缺陷程度即为最高。然而实际的软件系统中,绝对的零缺陷仍是不可

能的。希望可以建立一系列可以实际使用且广泛应用的基于缺陷摆脱程度的可靠性评估标准,使用此标准,客户可以比较各种软件系统,从而选择满足自身可靠性需求的产品。

从缺陷出发,通过分析检测缺陷工具的特点与能力来间接地获得可靠性标准。首先,得到的是工具与方法的能力标准,进而结合系统使用工具以及检测结果与报告,给出系统摆脱内存泄漏的程度。因此,此评估标准是与工具相关的。

系统可能的缺陷种类繁多,例如,死锁、缓存区溢出、内存泄漏等等。这些错误出现的原因,对系统的影响以及检测的手段大相径庭,使用同一个标准来刻画显然是不可能的。文章选择了内存泄漏作为代表,尝试制定软件系统内存泄漏相关的可靠性标准,以此提供刻画与比较系统摆脱内存泄漏程度的参考方法。

3　内存泄漏的基本概念

3.1　内存泄漏的定义

受限于内存资源的有限性,很多编程语言提供了手动动态分配内存的机制,这样会有较大的灵活性,相应地也为内存管理带来了一系列的严重问题,内存泄漏则是其中一个极为突出的错误。内存泄漏是指动态分配的内存在使用结束后,由于某种原因没有被及时释放,使得这些动态内存不能再被程序使用,导致可用内存数量减少,甚至耗尽。内存泄漏直接影响程序的稳定性和可用性,是影响程序安全性的一个重要缺陷。

不同于一般程序错误,内存泄漏本身问题的症状不明显,很多情况下,只有通过长时间的运行累积,问题才会表现出来。内存泄漏造成资源的不合理浪费,致使程序运行变慢,某些核心模块,如果无法获得需求的内存去处理就会导致整个系统崩溃。对于需要长时间运行的如服务器程序,内存泄漏带来的问题以及针对内存泄漏进行的攻击所造成的危害应更加受到重视。内存泄漏可以由远程输入触发,造成大量的拒绝服务攻击,更有甚者,还会造成未经授权执行任意代码。

3.2　内存泄漏的起因

常见的导致内存泄漏的原因很多。例如,不适当的释放操作(leak free),当释放一块含有对另一块内存持有唯一引用的内存;不适当的返回值(leak return),使用函数返回值持有唯一引用的内存地址;不适当的局部引用(leak scope),函数局部变量指向一块内存,函数返回前,没有将这个指引用传递到全局变量,也没有返回给函数的调用者。另外,类似悬摆指针(dangling pointer),对未曾分配内存的读/写操作等等,这些都有可能导致内存泄漏。使用垃圾收集机制的程序语言,如 Java,可以自动回收不再使用的内存,这在一定程度上减轻了程序员的负担,但是,垃圾收集不能解决内存泄漏的问题。垃圾收集机制对游荡对象无能为力。所谓的游荡对象是指,可以从根部通过某个引用访问到的,但对程序来说已经无用的对象。造成游荡对象的原因一般有:无意义的容器持有引用;无意义的事件监听器(这种情况的存在使得事件发生时,虚拟机要对更多的监听器传播事件,严重影响了程序的性能);一些过渡性的引用被延迟删除等。

图 1 给出了 C/C++、Java 中的内存泄漏现象的不同表现。

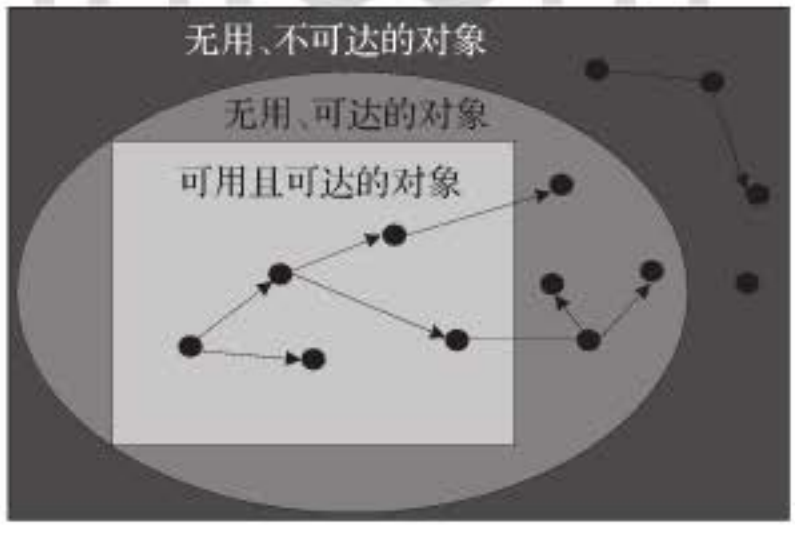

Fig.1　Behaviors of memory leak

图 1　内存泄漏的不同表现

3.3　内存泄漏的分类

图 2 给出了内存泄漏的基本分类。

这个划分是对工具进行实验评估的基础。分类包括两个分支:分支 1 是泄漏产生的常规原因分类;分

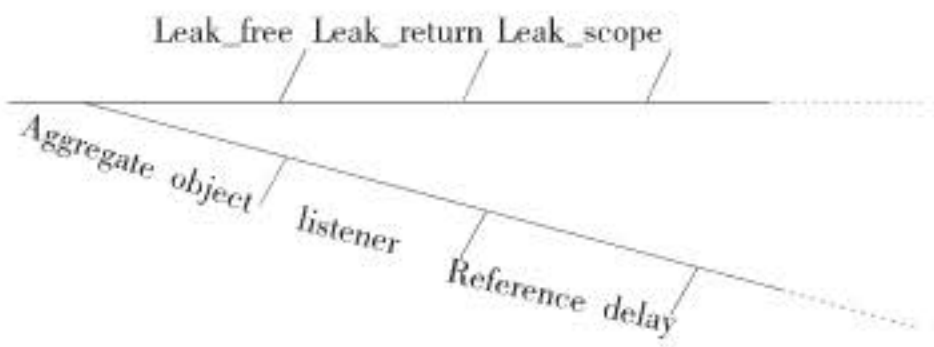

Fig.2 Classification of memory leak

图 2 内存泄漏的分类

支 2 是针对类似 Java 这一类具有垃圾收集机制语言特有的内存泄漏类型。

3.4 内存泄漏检测方法概述

要保证一个程序在内存泄漏方面的可靠性,程序员在开发阶段应养成良好的习惯,注意对动态分配的内存及时释放,合理使用异常处理方法,在技术上避免内存泄漏的发生。比如,Java 中提供了弱引用机制,可以用来避免不必要的引用关系阻止垃圾收集器对游荡对象的回收。调试时,除了使用工具,VC 本身也提供了专门用于内存监测的 Debug 方法,辅助程序员在最早时刻发现错误。

目前,检测内存泄漏的方法大致可以分为两类:动态方法和静态方法。最根本的区别在于是否需要执行目标程序。不需要执行程序的测试方法称为静态方法,需要执行程序的称为动态方法。

静态检测方法的最大优点是可以直接检查程序,而不用驱动程序运行。传统上静态分析方法根据流敏感、路径敏感以及上下文敏感进行分类,相应地有 flow-sensitive/flow-insensitive(FS/FI)、path-sensitive/path-insensitive(PS/PI)以及 context-sensitve/context-insensitive(CS/CI)。不同类型的方法所能达到的效率以及精确程度不同。

现存的、可用的静态工具较少。比如 PREfix[3]、Matal[4],这些工具基于从内存分配点开始的分析,识别出可能发现内存泄漏的路径。根据路径分析的结果,可以给出原来发生泄漏的对象应该在什么地方进行释放。类似如 LCLint[5],它需要程序员在编程的时候加入相应的注释标记,分析工具就根据这些标记来对程序进行检查。

静态检测方法的核心是基于对错误模式的描述。每一种检测工具,检测方法都预定义了相关的错误模式,或者是程序应当遵守断言,以此为基础进行检测工作,判断程序是否存在缺陷。针对内存泄漏,各类工具相应地定义了检测的基准,依据对内存漏泄这一现象本质的剖析,总结得到可能出现泄漏的错误模式。

例如,斯坦福大学的 SATURN 工具提供的内存泄漏检测,是基于所谓逃离分析(escape analysis)假设进行分析的。任何在进程 P 中分配的内存,没有在进程 P 中释放,也没有传递到其他进程,那么可以判定,这个分配的内存产生了泄漏。Stanford 大学 Monica S.Lam 教授领导的小组开发的 Clouseau[6-8],是一个检测 C++和 C 代码中内存泄漏和对象多次删除的工具,是基于隶属关系模型(ownership model)进行分析的半自动工具。

由于静态分析过程中,前端语法分析存在驼鸟式保守的猜测,使得最后的分析结果包含了很多的不精确成分,出现了大量错误的警告。过高的假阳性需要程序员手动地对分析结果进行筛选。Clouseau 在 3 个著名的开源 C 代码 binutils、openssh、apache 上运行的结果报告指出,检测工具进行过程内分析后给出的错误警告有 82 个,其中真正的错误是 26 个,准确率为 32%,而过程间分析后,警告的准确率为 21%。

动态检测方法通过驱动软件系统运行,或者是系统模型运行,得出针对当次运行结果的一个明确结论。许多工具提供了对程序本身的插装,监视程序运行时刻的状态,得到的数据用于追踪错误的起源。内存泄漏的动态检测工具,最基本的工作是监视程序中堆、栈的分配与释放,以及程序在运行过程中路径和方法的调用。动态检测工具可以分为 3 类。

第一类以监测内存使用为主,如 Jprobe、Jprofiler。这些工具提供了丰富的观察视角选项,可以从不同的角度描述程序运行时内存使用情况的变化。专业的程序员通过比较这些视图,可以比较清楚地观察泄漏的对象。例如,Jprobe 工具提供了多达 13 种的视图。

Instance 视图展示了所有在运行过程中分配的对象的相关数据;Instance Detail 视图可以针对指定的对象查看它持有哪些对象的引用,哪些对象持有对它的引用,以及这个对象在堆栈中分配的轨迹。

第二类是基于设定的堆增长率算法,比较运行时多个时刻的内存快照,可以自动判定是否发生了内存泄漏。这类的工具代表有 Cork[9]、LeakPot[10]和 Sleigh[11]。Cork 定义的描述视图为 TPFG(type points-from graph),通过比较 TPFG 中每个类型点的分配值的增加趋势来判定是否发生了内存泄漏。这一类工具的判定算法是一种经验算法,错报与漏报不可避免。

第三类是比较综合全面的商业工具。例如 Purify[12]、Insure++、BoundsChecker。这一类商业工具提供了完整的内存监测手段、错误判定以及追溯定位错误的方法。以 Numega 公司的 BoundsChecker 为例,BoundsChecker 提供了两种检测模式 ActiveCheck 和 FinalCheck,区别在于是否插装代码,两种模式的检测能力和检测开销因此也有差别。基本的做法是监测程序运行时刻内存的分配和释放状态,分析诊断是否出现内存使用错误。

理想的情况下,使用充分多的测试用例,获取完全的程序覆盖度,也就是说,每一个可能出现内存泄漏的场景都在检测过程中出现,假设每一个被发现的引起内存泄漏的程序缺陷都被修正,而且不会引入新的错误,那么内存泄漏错误就被最大化地消除了。实际情况下,一方面,测试用例的缺乏,使得系统的覆盖度不足。测试用例无法遍历所有的场景,对于存在于未运行到路径和模块中的错误,工具无能为力。另一方面,工具收集和描绘的运行时刻内存使用状况,有助于最后进行判断,以及错误的定位。这是检测工具能力的一个重要指标。

4 检测工具的评估方法与实验结果

4.1 检测工具的评估方法

错误植入是评估工具能力常用的手段,这里设计了 15 段不同的错误代码,覆盖内存泄漏的基本类型。通过对这 15 段代码进行检测,其结果可以评估工具的能力。

选取了 Stanford 大学开发的一个用于检测 C++和 C 代码中内存泄漏的 Clouseau 作为静态工具的代表,Numega 公司的 BoundsChecker 作为动态工具的代表进行第 2 章中描述的变异测试的实验。表 1 给出了实验结果。

Table 1 Experiment results

表 1 实验结果

	Leak_free	Leak_return	Leak_scope	Report warning	Confirmed errors
Clouseau	5/5	2/5	4/5	70	11
BoundsChecker	5/5	5/5	5/5	15	15

这两个工具处理的内存泄漏分类只在划分的第一分支上覆盖。采用了手动确定的方式对工具给出的报告进行了真伪辨认,可以看出,动态工具检测得比较精确,前提是实验提供的测试用例覆盖到了所有的错误场景。而静态工具 Clouseau 是在一个 C 和 C++的类似子语言的基础上开发的,这个子集包括多继承、静态函数、多构造函数以及模版,但是不包括并发进程、异常处理、指针运算、函数指针等 C++的语言特性。如果被检查的程序使用了这些特性,工具检测的有效性就无法保证,很多潜在的错误无法检测出来,但仍得到了一个比较理想的检测结果,警告的准确率为 15.7%。

4.2 实验结果分析

通过分析检测工具对这些程序的检测结果,获得其检测错误的能力。两种工具检测内存泄漏的能力如表 2 所示。

Table 2 Results analysis

表 2 实验结果分析

	f_1=Leak_free	f_2=Leak_return	f_3=Leak_scope
Clouseau	1.0	0.4	0.8
BoundsChecker	1.0	1.0	1.0

每个分类的能力值定义为检测到的错误与实际存在的错误的比值。

实验结果是对两种工具的内存泄漏检测能力的初步评估。

5 结论

内存泄漏不可避免地出现在所有使用动态分配内存机制语言的程序中,错误表现隐蔽,累积之后又会造成系统崩溃。如何有效地在编程阶段防范内存泄漏,在测试阶段查找内存泄漏这个错误,一直是保障软件可靠性的一个重要课题。

现阶段存在的针对内存泄漏的工具,根据是否要执行代码可以分为两类:静态工具以及动态工具。静态工具不会对程序执行造成额外的负担,效率高,但是由于分析方法的桎梏,静态工具都是不完备的,存在较高的漏报率和误报率。动态工具通过对目标代码的插装,在程序执行时测控内存的分配和释放状态,以此为据查找是否存在内存泄漏。动态工具检测错误的准确率高,同时根据程序执行时收集的信息,可以比较精确地定位到发生泄漏的对象。另一方面,监控行为受限于测试用例的选择,不能运行到的代码部分无法检查到,更为重要的是动态工具的使用为程序的执行带来了较大的额外开销,对于某些实时程序来说是不可忍受的。

References:

[1] Elling R, Pramnaick I, Mauro J, et al. Analytical reliablility, availability, and serviceability benchmarks[C]//Kanoun K, Spainhower L. Dependability Benchmarking for Computer Systems. [S.l.]: IEEE Computer Society, 2008:23-33.

[2] Coleman J, Lau T, Lokhande B, et al. The autonomic computing benchmark[M]//Kanoun K, Spainhower L. Dependability Benchmarking for Computer Systems. [S.l.]: IEEE Computer Society, 2008:1-21.

[3] Bush W R, Pincus J D, Sielaff D J. A static analyzer for finding dynamic programming errors[J]. Software-Practice and Experience, 2000,30(7):775-802.

[4] Engler D, Chen D Y, Hallem S, et al. Bugs as deviant behavior: A general approach to inferring errors in systems code[C]// Proceedings of 18th ACM Symposium on Operating System Principles, 2001:57-72.

[5] Evans D, Guttag J, Horning J, et al. LCLint: A tool for using specifications to check code[C]//Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 1994:87-86.

[6] Heine D L. Static memory leak detection[D]. Stanford University, 2004-12.

[7] Heine D L, Lam M S. A practical flow-sensitive and context-sensitive C and C++ memory leak detector[C]//Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03), 2003:168-181.

[8] Heine D L, Lam M S. Static detection of leaks in polymorphic containers[C]//Proceedings of the 28th International Conference of Software Engineering (ICSE'06), 2006:252-261.

[9] Jump M, McKinley K S. Cork: Dynamic memory leak detection for garbage-collected languages[C]//Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'07), 2007:31-38.

[10] Mitchell N, Sevitsky G. LeakBot: An automated and lightweight tool for diagnosing memory leaks in large Java applications[C]//LNCS 2743: European Conference on Object-Oriented Programming, 2003:351-377.

[11] Band M D, McKinley K S. Bell: Bit-encoding online memory leak detection[C]//Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, 2006:61-72.

[12] Hastings R, Joyce B. Purify: Fast detection of memory leaks and access errors[C]//Proceeding of the Winter USENIX Conference, 1992:125-136.

LI Qian was born in 1982. She received her B.S. degree in Computer Science and Technology from Nanjing University in 2005. She is currently a Ph.D. candidate at Nanjing University. Her research interests include code analysis and dependable software, etc.

李倩(1982-),女,江苏苏州人,2005 年于南京大学获计算机科学与技术专业理学学士学位,目前为南京大学计算机软件与理论专业博士研究生,主要研究领域为程序分析,可信软件等。

PAN Minxue was born in 1983. He received his B.S. degree in Computer Science and Technology from Nanjing University in 2006. He is currently a Ph.D. candidate at Nanjing University. His research interests include software model checking, design and analysis of real-time and embedded systems, etc.

潘敏学(1983-),男,江苏苏州人,2006 年于南京大学获计算机科学与技术专业理学学士学位,目前为南京大学计算机软件与理论专业博士研究生,主要研究领域为模型验证,实时和并发系统的设计和分析等。

LI Xuandong was born in 1963. He received his M.S. and Ph.D. degrees in Computer Science from Nanjing University. He is currently a full professor and Ph.D. supervisor at Nanjing University. His research interests include software modeling and analysis, software testing and verification, etc.

李宣东(1963-),男,博士,南京大学计算机科学与技术系教授、博士生导师,国家杰出青年科学基金获得者,主要研究领域为软件建模与分析,软件测试与验证等。主持承担了国家自然科学基金项目、国家重点基础研究计划(973)项目、国家高技术研究发展计划(863)项目在内的多项国家和省、部级科研项目,在国际期刊、国际会议和国内一级学报上发表论文 60 余篇。