第35卷 第20期 计 算 机 工 程 2009年10月
Vol.35 No.20 Computer Engineering October 2009
·软件技术与数据库· 文章编号：1000—3428(2009)20—0044—03 文献标识码：A 中图分类号：N945

嵌入式软件动态内存检测工具的设计与实现

刘发贵[1]，温宇龙[2]，罗春威[1]
(1. 华南理工大学计算机科学与工程学院，广州 510006；2. 华南理工大学软件工程学院，广州 510006)

摘 要：针对嵌入式软件中的内存泄露、内存写溢出等问题，提出嵌入式Linux 平台下数据采集和测试分离的交叉测试方法，设计实现一个嵌入式软件动态内存的检测工具。该工具可以检测软件的内存泄露、内存写溢出、释放野指针和内存管理函数不匹配等问题，通过一个实例验证其有效性和可靠性。

关键词：动态内存检测；嵌入式软件；内存泄露；交叉测试；可靠性

Design and Implementation of Dynamic Memory Detection Tool for Embedded Software

LIU Fa-gui[1], WEN Yu-long[2], LUO Chun-wei[1]
(1. School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006;
2. School of Software Engineering, South China University of Technology, Guangzhou 510006)

【Abstract】Aiming at the problems of memory leakage and memory overflow in embedded software, this paper proposes a cross-test method on embedded Linux platform, in which data collection and test are separated to improve the test efficiency. A dynamic memory detection tool is implemented, which can be used to detect memory leakage, memory overflow and mismatch of memory manage function. An instance is presented to show the effectiveness and reliability of the tool.

【Key words】dynamic memory detection; embedded software; memory leakage; cross-test; reliability

1 概述

嵌入式软件的质量和开发周期对产品的最终质量和上市时间起到决定性的作用，因此，嵌入式软件的测试成为研究的热点[1]。嵌入式平台内存不大、实时性要求高，嵌入式软件如果出现内存泄露等问题，势必导致系统可用内存减少，严重的可能引起系统崩溃。如何有效解决内存泄露、内存碎片和内存崩溃等问题对于嵌入式软件开发越来越重要。同时，嵌入式软件的测试比一般商用软件的测试更为复杂，可以说，嵌入式软件是最难测试的一种软件[2]。

目前，国内外已经有一些内存检测软件。国内的有FENSE,这个工具只能检测标准C的内存泄露和写越界违例，并且在易用性、附加开销和性能等方面对于嵌入式平台都不太理想[3]。国外的有 Purify, Valgrind, Mpatro, Memtrace, Memprof 等。这些工具都是在同一个主机上采集和分析数据的，对于资源紧缺的嵌入式平台而言，它们会影响系统资源的利用率[1]。其次，对于那些长时间处于运行状态的嵌入式软件，程序结束再检测内存问题就显得非常不方便[4]。

最常用的内存错误检测方法有源代码插装法、目标码插装法以及添加保护字节法[5]。本文基于源代码插装法和添加保护字节法技术，提出了数据采集和测试分离的交叉测试方法，并实现了一个嵌入式软件动态内存检测工具，该工具可以保证测试准确度、可靠性、及时性以及嵌入式平台资源的利用率。

2 内存问题分析

内存问题包括内存泄露、内存写溢出、分配和释放函数不匹配、野指针问题、内存申请失败、空指针解引用、内存未初始化、指针空挂和内存重复释放、栈内地址外传、内存使用函数接口错误等。其中4种常见的内存错误如下：

(1)内存泄露。内存泄露是指程序运行时动态分配的、使用完毕后没有释放、不能被操作系统回收的内存空间。

(2)内存写溢出。内存写溢出即内存写越界。写内存时超越所分配的内存空间，就发生了内存写越界。

(3)内存分配和释放函数不匹配。例如使用 new()分配内存，使用 free()释放。有这种错误的程序的运行情况将是不可预知的。

(4)野指针问题。野指针不是空指针，是指向垃圾内存的指针。

3 测试工具总体设计

考虑到嵌入式平台的资源问题，本文采用数据采集和测试分离的交叉测试方法。被测文件在开发机上交叉编译生成可执行文件，然后运行于嵌入式系统的目标机，同时采集数据。通过TCP/IP协议把采集到的数据传输到开发机上，再进行数据分析[1]，通过图形界面直观显示内存的各个问题。

3.1 总体框架

图 1 为测试的总体框架。下面分别从开发机(Service)和目标机(Client)介绍各个模块的功能：

(1)Service 包含 Memory Detection Service 和 Insert Library 2个模块：

基金项目：2007 年粤港关键领域重点突破基金资助项目(2007A010101003)；广东省教育部产学研结合示范基地基金资助项目(2007B090200018)

作者简介：刘发贵(1963－)，女，教授、博士、博士生导师，主研方向：操作系统，嵌入式软件；温宇龙、罗春威，硕士研究生
收稿日期：2009-01-23 **E-mail：**fgliu@scut.edu.cn

1)Memory Detection Service 又分为:与 Client 上的 Agent Testing 建立连接的 Communicate 模块；发送可执行文件到 Agent Testing，接收 Agent Testing 发送来数据的 Manage data 模块；及时分析数据和图形显示的 Display result 模块。

2)Insert Library 是 Service 静态库。

(2)目标机 Agent Testing 分 3 个部分:

1)Receive and Execute file 模块与 Service 建立通信，接收 Service 端发送来的可执行文件，把 Mirco Testing 和 Marco Testing 所获数据及时传输给 Service。

2)Micro Testing 模块捕捉可执行文件的内存申请和释放等操作的行为。

3)Macro Testing 模块只用于观测系统内存使用情况，根据系统内存的变化判断是否出现内存问题。

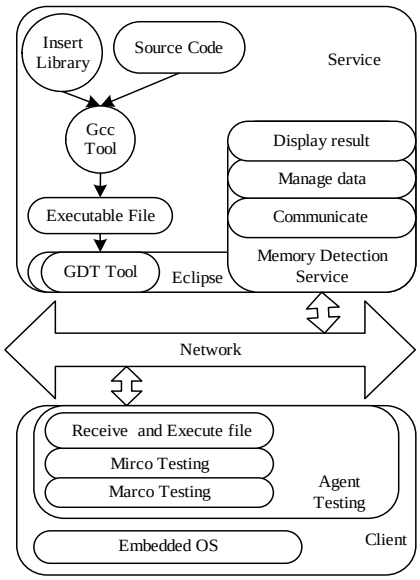

图 1　测试的总体框架

测试流程如下:

(1)在 Service 端，把 Insert Library 和嵌入式软件一起编译生成相对于 Client 的可执行文件。

(2)Service 端把可执行文件发送到 Receive and Execute file 模块并且执行。

(3)Micro Testing 和 Macro Testing 收集数据，再通过 Agent Testing 把数据发送 Manage data，再通过 Display result 分析，图形界面显示数据。

3.2　Micro Testing 模块设计

Micro Testing 模块采集的数据与 Service 端 Insert Library 关系密切，Insert Library 是通过改写 malloc 与 free 等函数并在这些函数里包装了对内存操作信息发送到 Service 端的代码，在 Linux 下为 Insert Library 设置环境变量 LD_PRELOAD 就可以完成对这个库的优先加载。

(1)Micro Testing 模块流程

在 Client 端执行测试文件，当遇到内存操作的函数时，流程如图 2 所示，具体如下:

1)如果是内存操作函数，会被 Insert Library 的函数替换，接着调用 Client 上标准库里的相应函数。

2)如果是申请内存函数，就把申请成功的内存的开始地址存入全局链表，如果是释放内存函数，就搜索全局链表看是否存在该内存，如果没有就报错。

3)将获得的内存操作信息发送给 Agent Testing。

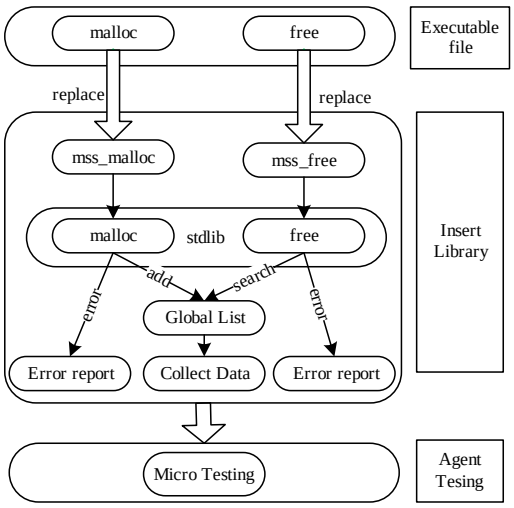

图 2　插装库流程

Insert Library 里一个全局链表 List 保存每次申请的内存地址；当释放内存时，在链表里搜索给定内存地址，然后从链表里删除，保证数据一致性。

(2)函数实现

下面给出 2 个主要函数 malloc 和 free 的实现原理，其他函数的实现类似。

1)malloc 函数

函数的实现如图 3 所示，具体流程为: ①设置函数类型，主要是方便 Service 分析。②前后多分配 MSS_WATCH_SIZE 字节，并且写入特殊的值。③把申请成功的内存的开始地址保存到全局链表。④统计信息，包括调用 malloc 函数的文件名、函数名、行号、申请时间、具体大小、进程号等，并发送给测试代理。其中，__FILE__, __PRETTY_FUNCTION__, __LINE__是 Linux 下的宏，用来获得文件名、函数名以及行号；MSS_BY_MALLOC 用来区别函数类型。

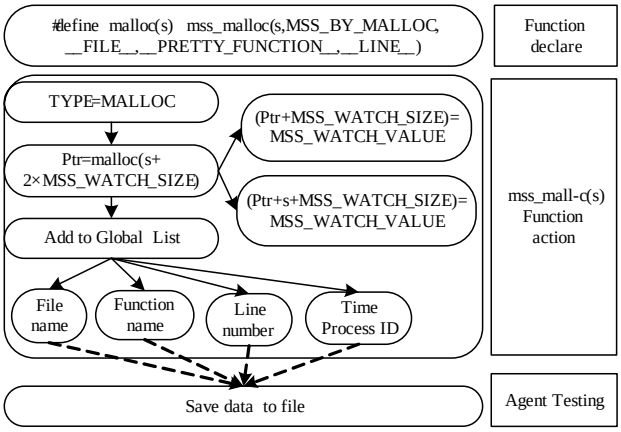

图 3　malloc 函数实现流程

2)free 函数

函数声明中的宏与 malloc 一样，流程为: ①设置函数类型。②搜索全局链表，如果找到相应的地址，就调用标准库里的 free 函数，释放空间。③统计信息，包括调用 malloc 函数的文件名、函数名、行号、申请时间、具体大小、进程号等，并发送给测试代理。Service 端接收数据保存，然后及时采用以下算法对数据进行分析。

(3)数据分析算法

1)访问越界。当用户需要分配 *s* 字节的内存时，实际分

配的内存是 s+2×MSS_WATCH_SIZE 字节，在实际分配的 s 字节内存的前面和后面分别分配 MSS_WATCH_SIZE 字节的额外内存，并在这 2 个额外内存里写入特殊数据 MSS_WATCH_VALUE。然后检查是否出现越界访问。每次释放内存时，判断这块内存区域中额外内存的内容是否与 MSS_WATCH_LIMITS_ VALUE 的值一致，如果不一致，说明出现了越界访问。

2)内存泄露。遍历记录内存申请信息的链表，如果链表不为空，则链表中的每个节点都代表一个泄露的内存块。

3)野指针。遍历记录保存内存申请信息的链表，取出链表中的每个节点，然后在循环体内部判断指针的地址是否在链表的节点的开始或内部。

3.3 Macro Testing 模块设计

宏观检测是指从宏观上确定一个进程有没有内存泄露，不考虑这个进程内存泄露的具体信息。从宏观上看，如果进程有内存泄露，它占用的内存就会不断增大，那么它占用内存的最大值就会不断被刷新。应用程序一般都是在堆上申请内存的，所以，通过获得应用程序堆空间、虚拟内存和实际内存的变化，可以判断是否出现内存问题。

(1)实现原理

为了获得用户进程的内存使用情况，需要读取 Linux 内核数据结构的实时数据，内核态和用户态数据交互流程如下：1)Agent Testing 执行被测试程序，获得被测试程序的进程号，再把这个进程号传送给内核态的 Macro Testing。2)根据进程号，Macro Testing 的 kernel_user 模块搜索整个进程链表(内核用于保存所有进程号的链表)，找到相应的 PCB(Process Control Block)。3)根据 PCB，可以获取进程的所有内核态信息，包括内存信息。同时把获得的内存信息写入字符设备驱动文件。4)应用态的 Agent Testing 读该字符设备驱动文件里的数据，并且发送给 Service。

(2)kernel_user 模块实现

内核里有 2 个主要的数据结构 task_struct 和 mm_struct，这 2 个结构体成员变量比较多，这里只说明对测试有用的成员。如图 4 所示，Macro Testing 下的 kernel_user 模块获得进程号后就可以找到相应的 PCB，PCB 结构体 mm 成员是指向 mm_struct 结构体的指针。mm_struct 结构体包含了进程的内存使用情况。

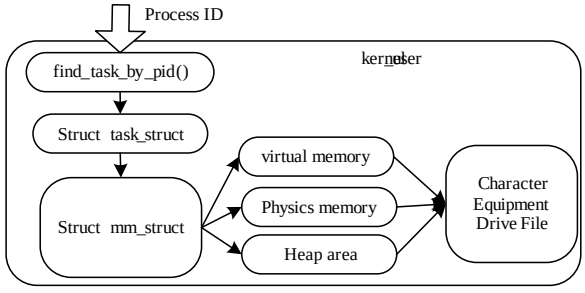

图 4 kernel_user 工作流程

4 实例与结果分析

本节给出了一个数字媒体适配器(DMA)中模块 GWsearchDir 的实现及测试实例。为了方便测试，把该模块从 DMA 中取出形成一个独立的程序。其中，main()函数遍历服务器共享目录下的文件，过滤出具体的文件类型和文件名，同时用 ptr 保存文件类型，再调用 fun1()，根据文件类型保存文件。同时在 main()函数里调用 create_list()函数把服务器的文件保存在链表中以方便查找。部分被测试的源代码如下：

```
Void fun1()
{ Int * p;   Int i=3;
  …
 While(i-->0)   { p=(int *)malloc(1024);
  …   }}
int main ()
{   Char *ptr;
  …
    ptr=(char*) malloc(10);
    *(ptr+11)='c';
     free(ptr)
     fun1();
     …}
 Void create_list()
{ int i;
 Struct list   *p,*list;
 L1=(struct list *)malloc(sizeof(struct list ));
…
for(i=0;i<Max;i++)
   {   P=(struct list *)malloc(sizeof(struct list ));
 …   }
 …}
```

main()函数执行时会检测到指针 ptr 申请内存情况以及 fun1()函数、create_list()函数内的内存使用记录。图 5 只给出了 create_list()函数申请内存情况，包含申请函数类型、文件名、函数名、行号、大小等。在 main()函数中，指针 ptr 出现了访问越界，执行 fun1()，检测到了内存泄露，并且显示了具体泄露的文件名、函数名、具体行等信息。宏观测试将显示被测试程序虚拟内存、物理内存、堆栈空间的使用情况。

内存操作记录 | 当前申请内存 | 内存越界记录 | 无效内存记录 | 内存泄漏记录

类型	文件名	函数名	行号	地址	大小	时间
Malloc	test2/test3.c	create_list	21	0x8051cd0	8	2004-01-04 03:38:19
Malloc	test2/test3.c	create_list	21	0x8051d48	8	2004-01-04 03:38:19
Malloc	test2/test3.c	create_list	21	0x8051dc0	8	2004-01-04 03:38:19
Malloc	test2/test3.c	create_list	21	0x8051e38	8	2004-01-04 03:38:19

图 5 申请内存记录

5 结束语

本文实现了一个针对嵌入式软件的动态内存检测的测试工具。该工具能准确、可靠、及时地测试嵌入式软件内存问题。从测试工具的运行情况来看，该工具自动化程度高、可扩展性好、界面友好，分析结果直观而且精确，可有效降低嵌入式软件测试成本。

参考文献

[1] 刘发贵, 李绍华, 陆　璐. 嵌入式软件测试平台的框架设计[J]. 系统工程, 2007, 25(增刊): 8-11.

[2] Broekman B, Notenboom E. 嵌入式软件测试[M]. 北京: 电子工业出版社, 2004.

[3] 何杭军, 朱　利, 李青山, 等. 基于 Linux 的动态内存检测工具的设计与实现[J]. 计算机工程, 2005, 31(21): 30-31.

[4] Liu Fagui, Luo Chunwei. Service-oriented Software Testing Platform[C]//Proc. of International Federation for Information Processing. Wuhan, China: [s. n.], 2007.

[5] 欧阳志强. 一种与平台无关的 C 程序内存错误检测工具的设计[J]. 中国高新技术企业, 2007, 30(7): 47-49.

编辑　张　帆