面向TD协议栈的内存管理技术研究

石 坚,雷咏梅
(上海大学 计算机工程与科学学院,上海 200072)

摘 要:内存管理是一个一直值得研究和优化的问题。在TD-SCDMA高层协议栈软件跨平台技术的研究过程中,文中提出了一种内存管理方法:在软件和操作系统之间设立抽象层,抽象层的内存池在系统初始化时集中申请,软件运行时的内存申请释放由抽象层进行调度,抽象层采用大小最匹配算法将内存池中的内存块给软件使用。抽象层提供link函数以提高软件执行效率,并提供内存泄露和内存越界使用的检测。与传统内存管理相比,使用文中介绍的内存管理技术的高层协议栈软件,一次内存调度平均速度可以提高1微秒,并能完全屏蔽下层不同操作系统之间的差异。

关键词:抽象层;面向协议栈的内存管理;连接
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2009)11-0053-04

Buffer Management Technologies for High-Layer Protocol

SHI Jian, LEI Yong-mei
(School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

Abstract: The buffer management has been one of the problems which is worthy of being researched and improved all the time. In the development of the operation-system-abstract for the TD-SCDMA high layer protocol software, present a buffer management method: to create an abstraction layer between the software and operating system. The abstraction layer apply for physical memory while initialization. When software applies a memory block, the abstraction layer provides it which matches the size best. The abstraction layer provides the Link function in order to enhance the software efficiency. And it provides the methods for checking buffer leak and cross-border use of memory. Compared with conventional memory management, the software using the abstraction layer can increase the average speed of one memory block application by 1 microsecond and can completely mask the differences between different operating systems.

Key words: abstraction layer; buffer management for protocol layer; link

0 引 言

关于抽象层的技术,微软曾最先提出硬件抽象层HAL(Hardware Abstraction Layer)的思想[1],便于操作系统在不同硬件结构上进行移植。之后,抽象层技术就广泛地应用于各类跨平台软件的开发[2]。

在开发TD-SCDMA高层协议栈[3]软件的过程中,文中也使用抽象层思想,在协议栈软件和底层操作系统增加一个操作系统抽象层。抽象层对上层软件提供一系列基本和高效函数,屏蔽下层不同操作系统之间的差异。基于文中所介绍的抽象层的内存管理技术,TD协议栈软件的运行速度有一定的提高,一次内存调度平均能节省1微秒的时间,并且在内存泄漏和越界的保护上也有改进。

收稿日期:2009-03-13;**修回日期**:2009-06-27
作者简介:石 坚(1982-),男,硕士研究生,研究方向为软件工程;雷咏梅,博士,教授,研究方向为高性能计算技术、网格计算与应用及信息安全。

1 面向TD协议栈的抽象层内存管理技术

1.1 当前OS内存管理现状

虽然目前主流操作系统的内存管理机制都已比较成熟,但是对于TD协议栈软件,还是存在着下面的一些不足[4]:

(1)Host OS的内存管理函数涉及大量保护,要耗费较多的系统调用时间。

(2)不同的Host OS在内存管理上有很大差异(比如内存碎片[5]的处理)、性能参差不齐[6],会给移植带来隐患。

(3)若内存上发生错误,如果不熟悉Host OS的内部结构(尤其是闭源码的操作系统),将很难追查错误的源头。

1.2 面向TD协议栈的抽象层内存管理的结构

基于上述的问题,文中针对TD协议栈软件,在操作系统抽象层中开发了内存管理模块[7]。图1为内存管理模块结构图。

该文所介绍的内存管理技术,对内存使用频繁的

系统有明显的性能优化。

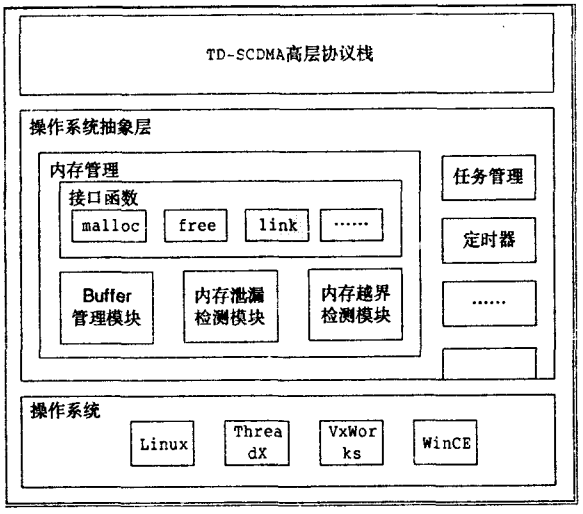

图 1 内存管理模块图

2 内存管理技术的原理与实现

2.1 Buffer 管理的原理

操作系统抽象层自己实现了一套内存分配调度机制,称之为 Buffer 管理[8]。

Buffer 管理的基本原理是:抽象层首先从操作系统分配出若干块较大的内存区(称为 Memory Pool),每块内存区又分为若干组大小不等的子内存块(Memory Block),其中每一组成为一个簇(Cluster)。应用程序需要分配内存时,抽象层就从应用程序指定的内存池中找到一块比所需内存大的最小 Block,将其指针返回给应用程序。而应用程序释放内存时,抽象层则找出这块内存属于哪个 Memory Pool,然后将其归还给该 Pool。如图 2 所示。

具体来讲,每个簇包含一条由尚未分配出去的 Block 组成的空闲簇链。每次上层应用申请一段内存时,抽象层就在其指定的 Memory Pool 里搜索各 Cluster,直到找到一个簇满足:

(1)它的 Cluster Size 不小于应用申请的内存大小。

(2)它是该内存池中满足条件 1 的所有簇中 size 最小的。

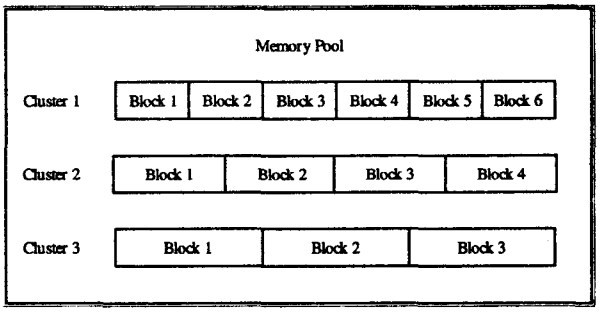

图 2 内存池结构

抽象层会到这个簇的空闲簇链中"摘"出第一个 Block,将其地址返回给应用[9]。

相应地,上层应用释放一段内存时,抽象层会将这个 Block 返还给这条空闲簇链的尾部,这块内存下次就可以被别的应用申请了。

2.2 Buffer 控制块设计

抽象层对 Buffer 的管理依赖于若干个 Buffer 控制块。所谓 Buffer 控制块,就是记载 Buffer 各种信息的一组数据结构,包括内存池描述表、内存池控制块、簇控制块、Block 控制块。如图 3 所示。

内存池描述表(Memory Pool Description Table):每个内存池都有这样一张描述表。这张表静态地描述了这个内存池有多少个簇,每个簇多大,以及每个簇里有多少个 Block。

比如

```
struct Osa_mem_desc_tbl_st mem_pool_1[] =
```

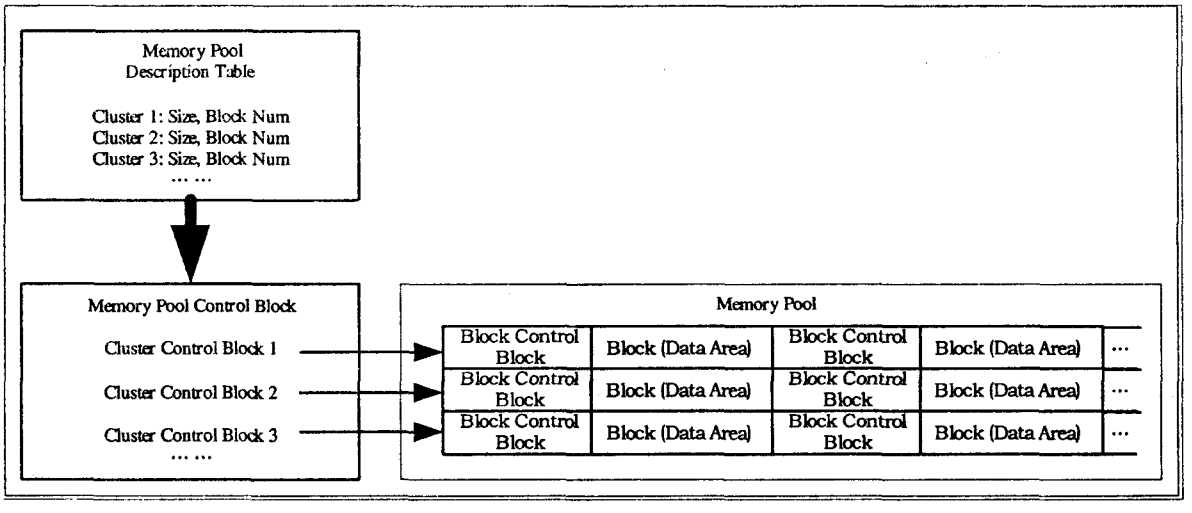

图 3 抽象层 Buffer 控制块数据结构

```
{
    {8, 100},
    {512, 512},
    {32, 1200},
    {1024,5},
    {0,0}
};
```

上面这张表表示,mem_pool_1 这个内存池中有 4 个簇,其中 8 字节大小的簇有 100 个 Block,32 字节大小的簇有 1200 个 Block 等等。其中最后一行{0, 0}是表结束的标记符。创建内存池时,正是根据这张表。需要注意的是,内存池描述表本身是一个全局变量,它并不占用 Memory Pool 的内存。

内存池控制块(Memory Pool Control Block):每个内存池都有一个动态的控制块,用来存放各簇控制块信息。内存池控制块本身是单独一块内存,不占用 Memory Pool 的内存。

簇控制块(Cluster Control Block):簇控制块存放了属于该簇的 Block 的信息,包括簇的大小,Block 的首地址、空闲簇链的首尾地址等。簇控制块本身属于内存池控制块的一部分。

Block 控制块(Block Control Block):Block 控制块登记了每个 Block 自身的信息,包括这个 Block 属于哪个内存池的哪个簇(这使得释放内存时,该 Block 可以被放回到正确的空闲簇链里),等等。与其它 Buffer 控制块不同,Block 控制块占用 Memory Pool 的内存,其位置就在每个 Block 之前。

2.3　Memory Link 功能

根据 TD 协议栈的实际特点,抽象层的内存管理模块对用户提供了一套"Link"接口函数。

一般情况下,一次内存申请应该和一次内存释放相对应。但是,在某些情况下却会因此而碰到异常情况。比如,下面是某个模块的 task_main()函数:

```
Osa_status task_main(Osa_fsm_message * fsm_msg_ptr)
{
    switch(FSM_PRIMITIVE_ID(fsm_msg_ptr)
    {
        case: PRIMITIVE_1:
            handle_prim_1(fsm_msg_ptr);
            break;
        case: PRIMITIVE_2:
            handle_prim_2(fsm_msg_ptr);
            break;
        case: PRIMITIVE_3:
            handle_prim_3(fsm_msg_ptr);
            break;
        default:
            do something;
        break;
    }
    /* Always attempt to free the memory */
    osa_free_fsm_msg(fsm_msg_ptr);
    return somevalue;
}
```

假设 handle_prim_1()和 handle_prim_2()结束后要释放参数 fsm_msg_ptr,handle_prim_3()结束后不能释放 fsm_msg_ptr,上面的代码就得修改。有两种该法:方法 1 是重新申请一块内存,把这条消息的内容拷贝一遍,保存下来;方法 2 是把 task_main()函数末尾的 osa_free_fsm_msg()移到 handle_prim_1()和 handle_prim_2()里面去释放它们处理完的消息。

但是这两种解决方法都不是很恰当。对于方法 1,若这条消息很大,那么每次都拷贝整条消息会消耗太多的时间和内存。而对于方法 2,switch 很多的时候,每个出口都要有 osa_free_fsm_msg()调用,不仅代码量增加了不少,而且很容易发生遗漏。

Link 函数的引入正是为了处理该类问题。Link 函数的实际作用是,抵消后续的一次 free 操作。对于前面那个 task_main(), handle_prim_1()、handle_prim_2()和 task_main()本身都不再需要任何更改,只要 handle_prim_3()里面调用一次 osa_link_fsm_msg(),就可以抵消 task_main()结尾处的 osa_free_fsm_msg()调用。

基于抽象层实现 link 的方法即在每个 Block Control Block 里,设置一个计数器。空闲簇链里的 Block,它的计数值是 0,当它被分配出去,计数值就增长为 1。如果它被 link,那么每次 link,计数值都再加 1。free 的时候,首先会把计数值减 1,若减 1 之后计数值仍然大于 0,那么 free 函数不会真正释放这块内存,直到某次调用 free 之后,计数值减到 0 才释放。

2.4　内存泄漏与越界检查

抽象层提供了一套简单易行的内存泄漏与越界检测机制。

内存泄漏的检测使用位于 Block Control Block 里的计数值。因为这个计数值表示了这块内存当前的状态:未分配、已分配或者已分配且被 link 过。通过统计各个内存块里的这个计数值,若发现某些内存块的计数值增长过之后,就一直没有回到过 0,那么这块内存就有可能是发生泄露的内存。抽象层能够记录每块内存块每次 get、link 和 free 的操作来自那个文件的哪一行(这些信息也存放在 Block Control Block 里,不过只在打开内存泄漏检测开关的时候才会有),因此可以找

到这块内存是如何发生泄漏的。

内存越界的检测依赖于 Block Control Block 里的保留字。从图 2 中可以知道，每个 Block 的数据区和下一个 Block 的控制块是紧挨着的。如果前一个 Block 的数据区发生越界，那么就会覆盖下一个 Block Control Block 的内容。在每个 Block Control Block 开头的位置放置一个保留字，它被初始化为是一个罕用数值(0xCF)。每次 get、link 和 free 的时候，都会检查下一个 Block 控制块的这个保留字有没有发生更改。如果值变了，那么说明当前这个 Block(或更前面的 Block)的数据区发生过越界。和内存泄漏检测一样，检测到越界后，可通过查询发生越界的那个 Block 的控制块里存储的文件和行号信息，来推测发生越界的代码。

必须明白的指出，内存泄漏与越界检测机制，实际上只是一种告警机制。它并不能阻止内存泄漏或越界的发生，也不能防止泄漏或越界造成的破坏。

3 性能测试与分析

该文在 VxWorks 系统上对内存管理进行了性能测试[10,11]。

测试环境：VxWorks 版本为 5.5.1，系统内存大小为 32MB，除去系统必要的内存开销，剩余可供动态分配的内存有 20MB 左右。

系统的硬件配置如下：

CPU：主频 551239583Hz；

66MHz 系统总线。

测试程序包括内存申请、释放，共申请内存 4086 次，每次申请大小按正态分布，申请的内存总数超过系统内存的最大值。测试程序分别运行于 VxWorks(直接调用 malloc/free 函数)和基于 VxWorks 的抽象层(调用 osa_malloc 和 osa_free)之上。

运行结果如表 1 所示，VxWorks 申请一次内存的平均时间开销在 1.46 微秒，而抽象层申请一次内存的平均时间开销只要 0.82 微秒。

表 1 测试数据

场景	次数	均值	方差
VxWorks	4086	1.468751	0.14371
抽象层	4086	0.82759	0.40757

从图 4 可以看出，申请时间主要集中在 1.5 微秒附近，形成一条趋势线，还有一部分申请时间在 0.5 微秒左右，并且有零星几次的申请时间在 3 微秒以上。

从图 5 可以看出，大部分的点落在 1.5 微秒附近和 0.5 微秒附近，而在 0.5 微秒附近的点形成一条趋势线。和场景 1 的统计图对比来看，最大值超过了 7 微秒，而且多了很多超过 3 微秒的点。

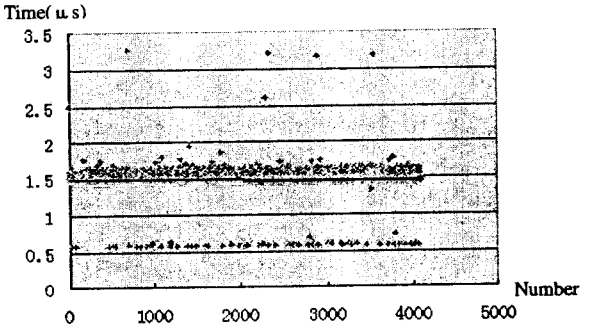

图 4 VxWorks 内存申请耗时统计图

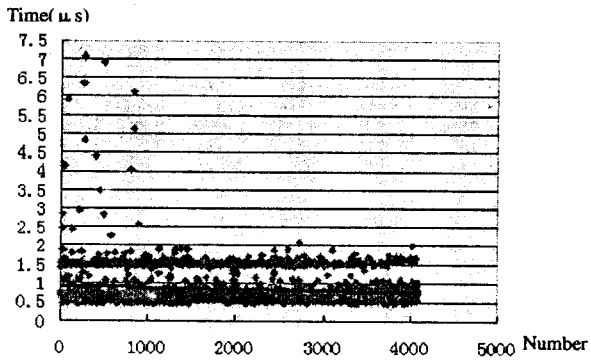

图 5 抽象层内存申请耗时统计图

从试验数据可以看出，抽象层的内存调度算法，在性能上是优于 Host OS 的。但是一次内存申请时间的波动区间较大，仍需要改进。

对比图 4 和图 5 可以看出，使用了抽象层的内存管理技术之后，能够将相当一部分的内存调度时间从 1.5 微秒降至 0.5 微秒。这是因为内存管理的实质是将内存申请和分配工作都放到了抽象层的初始化流程中去。之后上层软件的内存申请，只是在使用抽象层已经申请好的内存，所以 Buffer 管理去掉了一些非必须的内存边界保护和内存碎片处理，因此抽象层的内存管理在性能上优于直接使用 VxWorks 的内存管理。

4 结束语

在 TD 协议栈软件的实际使用过程中，因为内存的动态申请工作全是在系统初始化时完成，所以协议栈软件启动需要花费更多的时间，但这个时间相对于手机的开机流程，用户是无法察觉到的。但是在协议栈运行过程中，很多流程比如说测量上报的周期都是微秒级的，能够在内存调度上平均节省 1 微秒，在性能上确实有很大的提高。

TD 协议栈软件目前能够完美地运行于各种操作系统之上，完全归功于操作系统抽象层。作为抽象层核心模块之一的内存管理模块，起着至关重要的作用。在性能上，它可以提高内存申请释放的速度，而且它所

(下转第 60 页)

小时。根据图 6 所示 SPN 模型,该软件计算的性能数据结果是:该 SPN 模型共 12 个标识状态,例如标识 0 为"2 0 0 0 0 1 1 1",稳定状态概率是 0.2590。节点 P0 的平均标识数是 0.88。

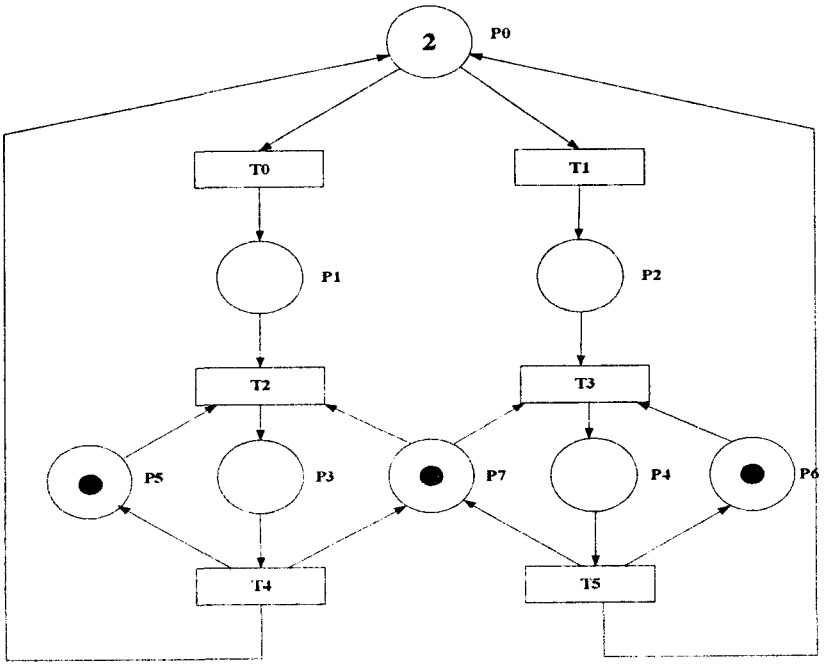

图 6 奶牛体征采集 SPN 模型图

3 结束语

通过研究可视化随机 Petri 网性能计算软件模型,分别建立了该软件的 CIM,PIM,PSM 模型,最后通过一个电子农务平台应用案例,给出了软件运行实例。由于采用了独立于随机 Petri 网的软件设计方法,而且

(上接第 56 页)

提供的内存泄漏与越界检测也简化了软件的开发流程,增强了软件的稳定性。

参考文献:

[1] 王力生,仇志付,唐军敏.嵌入式操作系统通用硬件层的设计[J/OL].单片机与嵌入式系统应用,2006-10. http://www.mesnet.com.cn/htm/magazine_view.asp? id=2029.

[2] Serpanos D N,Karakonstantis P. Efficient Memory Management for High-Speed ATM Systems[J/OL]. Design Automation for Embedded Systems,2004-11. http://www.library.utoronto.ca/pdf_test/kl08132001/354145.pdf.

[3] 李小文. TD-SCDMA 第三代移动通信系统、信令及实现[M]. 北京:人民邮电出版社,2005.

[4] 斯托林斯.操作系统——精髓与设计原理[M].第 5 版.北京:电子工业出版社,2006.

[5] 黄贤英,王 越,陈 媛.嵌入式实时系统内存管理策略[J].计算机工程与设计,2004(10):182-184.

[6] 李 毅.Slab 内存分配策略与移植[J].计算机技术与发展,2007,17(10):168-170.

[7] Friedrich L F,Stankovic J. A Survey of Configurable, Component-based Operating System for Embedded Applications [J]. IEEE MICRO, 2001,5(6):54-68.

[8] 董庆丰,黄迪明.一种适用于嵌入式系统的动态内存管理技术[J].微型机与应用,2004(8):52-55.

[9] Leeman M. Methodology for Refinement and Optimisation of Dynamic Memory Management for Embedded Systems in Multimedia Applications[J]. The Journal of VLSI Signal Processing,2005,40(3):383-396.

[10] 刘小军,李秀娟.嵌入式操作系统 VxWorks 的内存管理技术研究[J].电子科技,2008,21(6): 62-65.

[11] 陈 洋.VxWorks 下的内存管理[J].计算机工程,2007,33(8):94-96.

该性能计算功能是同类性能计算软件的基础功能,因此同类软件的设计亦可适用此开发方法。下一步的研究方向是如何解决原始输入参数的合法性验证等问题,以提高软件的可计算性。

参考文献:

[1] Petri C A. Kommunkation mit automaten[M]. Bonn: Institut fur Lnstrum Entelle Mathematik, Schriften des IIM Nr.3,1962.

[2] 梅 宏,陈 锋,冯耀东,等. ABC:基于体系结构、面向构件的软件开发方法[J]. 软件学报,2003,14(4):721-732.

[3] OMG unified modeling language specification (Version 1.4)[S]. Needham: Object Management Group, Inc.,2001.

[4] 邵维忠,杨芙清. 面向对象的系统分析[M].第 2 版. 北京:清华大学出版社,2004.

[5] 邵维忠,刘 昕.可视化编程环境下人机界面的面向对象设计[J].软件学报,2002,13(8):1494-1499.

[6] 杨 健. 基于 MVC 的论坛网站的设计与实现[J].计算机技术与发展,2006,16(6):81-83.

[7] 徐婷婷,段凡丁. MVC 设计模式在 B/S 开发中的研究与应用[J].计算机技术与发展,2007,17(6):119-121.

[8] 隋 永,周家纪.MVC 在 J2EE 框架中的应用研究[J].计算机技术与发展,2006,16(12):235-238.