第 40 卷 第 9 期 计 算 机 工 程 2014 年 9 月
Vol. 40 No. 9 Computer Engineering September 2014

·开发研究与工程应用· 文章编号:1000-3428(2014)09-0291-04 文献标识码:A 中图分类号:TP316.81

一种提高 Linux 内存管理实时性的设计方案

王兆文,蒋泽军,陈进朝

(西北工业大学计算机学院,西安 710129)

摘 要:针对 Linux 系统在内存管理方面实时性支持不够的问题,设计一种提高 Linux 内存管理实时性的方案。从3 个方面改进 Linux 系统内存管理的实时性,包括建立内存映射来减少用户态和内核态之间的模式转换,将内存锁定避免换页操作,改进系统原有的内存管理算法来消除内存操作的不确定性。改进后的内存管理算法基于分区管理和最佳适配的原理,时间复杂度为 $O(1)$。实验结果表明,该方案可以提高 Linux 内存管理的时间性能,特别是在内存使用紧张的环境下效果更加明显,性能提高率可达 49.5%,能够满足实时性的要求。

关键词:Linux 系统;实时性;内存映射;内存锁定;内存管理算法;分区管理

A Design Scheme for Improving Real-time Property of Memory Management in Linux

WANG Zhao-wen, JIANG Ze-jun, CHEN Jin-chao

(School of Computer Science, Northwestern Polytechnical University, Xi'an 710129, China)

【Abstract】 To the problem of imperfection in real-time property of memory management under Linux system, this paper designs a solution to improve the timeliness. It works in three aspects: establishing a mapping relationship between virtual address and physical address to reduce the switch between the user mode and kernel mode, locking memory to avoid page exchanging, improving the original algorithm of memory management to remove the nondeterministic operations. The modified memory management algorithm is based on the principle of partitioned management and best fit, whose time complexity is $O(1)$. Experimental results show that this solution is a good way to improve the performance of memory management, in the environment of memory tension, its effect is more obvious, and performance improvement rate can reach 49.5%. It meets the requirement of real-time.

【Key words】 Linux system; real-time property; memory mapping; memory locking; memory management algorithm; partitioned management

DOI:10.3969/j.issn.1000-3428.2014.09.058

1 概述

随着信息技术的飞速发展,实时系统得到了越来越广泛和深入的应用。实时性的含义并不意味着“快”,它是指系统的时间响应特性。具体主要有以下 3 个要求:(1)时间约束,任务响应时间在要求的期限内。(2)可预测性,任务的执行时间可以预先判断,是有界的,没有不确定因素影响。(3)可靠性。

Linux2.6 系统在实时性方面做了许多改进,但是在内核可抢占性、中断机制、虚拟内存技术等方面还是不能满足越来越高的实时要求[1],特别是在内存管理方面,Linux 更注重于空间效率与时间效率的平衡,对实时性的支持还不够完善。需要深入分析影响内存管理实时性的原因并对其加以改进。因此,本文设计一种提高 Linux 内存管理实时性的方案。

2 影响内存管理实时性的因素分析

Linux 在内存管理方面影响系统实时性的因素主要有以下 3 个方面:

(1)用户态和内核态之间的模式转换

在 Linux 中,实时任务运行在用户态,而出于系统安全的目的,用户态和内核态之间的程序不能直接通信。当有数据需要传输和处理时,系统需要不断地在用户态和内核态之间切换,不断复制数据,这样就增加了任务完成时间,有可能造成超时。

(2)页面交换机制

Linux 采用了虚拟内存技术,当系统需要更多内

基金项目:航空科学基金资助项目(2012ZC53040);国家部委基金资助项目;西北工业大学研究生创业种子基金资助项目。

作者简介:王兆文(1978-),男,工程师、硕士,主研方向:分布式实时嵌入式系统;蒋泽军,教授;陈进朝,博士研究生。

收稿日期:2013-07-10 **修回日期**:2013-09-17 **E-mail**:xxkhw@tom.com

存时,它将暂时不使用的页面从内存换出到磁盘,增大可用内存空间,然后再将需要使用的页面换入到内存中,这就是页面交换机制[2]。对于实时任务而言,缺页换页操作会带来延时,更重要的是,它会带来操作时间的不确定,而这些对实时任务来说都是不能容忍的。

(3)内存操作的不确定因素

应用程序在调用 malloc/free 函数进行内存申请/释放时,系统采用动态内存分配,即从堆上分配内存[3],由于申请大小和申请释放时机不同,当频繁进行这些操作时会产生大量的内存碎片,导致系统性能降低,花费时间不确定。

3 内存管理实时性改进方案设计

针对以上非实时性因素,从 3 个方面改进系统实时性:

(1)建立虚拟地址和物理地址之间的映射,使得用户层和内核层之间的通信无需经过模式转换,避免模式转换带来的时间消耗。

(2)将内存页面锁定在物理内存中,使得实时任务不会产生缺页换页。

(3)改进操作系统原有的内存管理方法,设计新的内存管理算法代替系统原有的方法,避免原内存操作带来的不确定性[4]。改进方案如图 1 所示。

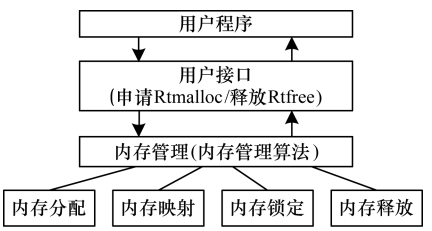

图 1 内存管理实时性的改进方案

系统首先用 kmalloc( )函数在驱动程序中申请足够大的一块内存,之后每个用户程序需要使用内存时,先对这块区域进行映射,然后用 SetPageReserved( )函数将这块内存区域设置为保留(即内存锁定),用户通过内存管理模块导出的用户接口(分配(RTmalloc)、释放(RTfree))来使用内存。具体的内存分配释放算法由内存管理模块实现,这部分功能对用户是透明的,用户程序通过用户接口直接使用内存。

3.1 内存映射

进程用户空间的一部分可以和磁盘文件的某一部分或者设备文件相关联,因此,内核把对线性区中页内某个字节的访问转换成文件中相应字节的操作。这种技术称为内存映射[5]。进程发出一个 mmap( )系统调用来创建一个新的内存映射。一旦创建了这种映射,进程就可以从这个新线性区的内存单元读取数据,也就等价于读取了文件中存放的数据,如图 2 所示。

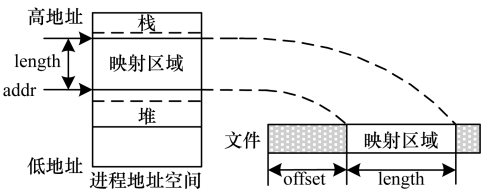

图 2 内存映射的原理

mmap( )系统调用的原型:

void * mmap(void * addr, size_t length, int prot, int flags, int fd, off_t offsize);

返回值:成功则返回映射区起始地址。失败则返回 MAP_ FAILED(-1)。

参数:

addr:指定映射的起始地址;

length:将文件的多大长度映射到内存;

prot:映射区的保护方式;

flags:映射对象的共享标志;

fd:文件描述符,代表要映射的文件;

offset:以文件开始处的偏移量;

根据 fd 参数指向文件类型的不同,内存映射可以分为不同的方式。当 fd 指向类型为设备文件时,为内存映射设备方式[6];当指向类型为普通磁盘文件时,为内存映射文件方式,2 种方式的原理基本相同,但是实现方式有所不同,本文主要介绍内存映射设备方式。

映射的实现过程:首先,驱动程序先分配好一段内存;接着,用户进程通过系统调用 mmap( )告诉内核需要将多大的内存映射到内核空间,内核经过一系列处理后调用对应的驱动程序的 file_operation 结构中的(* mmap)( )方法,在该方法中调用 remap_pfn_range( )方法建立映射关系,即建立映射区的页表。简单点说就是驱动程序在(* mmap)( )中利用 remap_pfn_range( )函数将内核空间的一段内存与用户空间的一段内存建立映射关系,如图 3 所示。

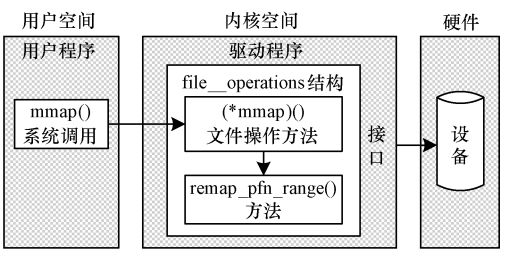

图 3 内存映射的实现过程

在不使用内存映射的情况下,应用程序使用 read( )/ write( )等函数与设备传输数据,需要先切

换到内核模式,使用 copy_to_user/copy_from_user 等内存复制函数将数据复制到内核空间,然后输出到设备。使用内存映射以后,在应用程序的进程地址空间上映射了设备驱动程序提供的物理地址,用户程序就可以不用切换到内核空间拷贝数据而直接访问物理地址,这样就大大提高了用户程序访问磁盘文件或者硬件设备的效率。

内存映射通常应用在那些内核和用户空间需要快速大量交互数据的情况下,特别是那些对实时性要求较强的应用。实际上,它是把内核中特定部分的内存空间映射到用户级程序的内存空间去[7]。服务器需要对视频内存进行大量数据交换,可以被看作是内存映射用法的一个典型例子,它将视频内存直接映射到了用户空间,而视频内存在内核空间是有特定位置的。也就是说,内存映射是为某个进程映射特定的内核空间区域,把这块区域留作专用[8],因此,它对整个系统安全性的影响是很小的。

3.2　内存锁定

针对换页操作带来的延迟和不确定,采用内存锁定的方法加以解决。内存锁定是一种将进程保留在内存而不产生页面交换的方法,它允许程序在物理内存上锁住它的部分或全部地址空间。内存锁定以页为基本单位,被指定区间所包含的每个内存页都会被锁定。Linux 提供了内存锁定和解锁的 API,用户程序中可使用 mlock()/munlock()函数,驱动程序中可使用 SetPageReserved()/ClearPageReserved()函数[9]。内存锁定可以有效解决换页操作带来的延迟和不确定,但是要避免同时锁定大量的内存页面,否则将会因为内存紧张而造成系统的整体性能降低。

3.3　对内存管理算法的改进

改进后的内存管理算法采用二级分区的思想,将之前锁定的内存区域分成若干个分区,每个分区又分成若干大小相同的内存单元,不同分区的内存单元大小各不相同,系统以分区为单位来管理内存,而用户程序以单个内存单元为单位来获得和释放内存[10],每次收到用户申请时系统找出大小最合适的一个空闲单元来满足申请,释放时将其放回原来所属的分区中。

系统使用 m_partition 结构来表示每个分区,这样整个区域就可以用一个 m_partition 结构数组来维护,m_partition 结构中的空闲链表和已使用链表用来标记各个分区中的内存单元使用情况,其结构示意图如图 4 所示。

系统还预留一部分空间作为备用空间,当某个分区的内存单元需求量很大不够用时,从备用空间补充分配一些空间给这个分区;当有个别用户申请超过最大块限制时,直接从备用空间分配[11]。

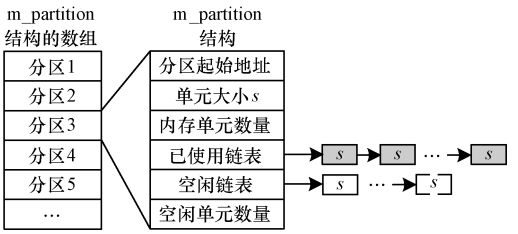

图 4　分区的数据结构示意图

算法的运行过程如下:

(1) 初始化过程。系统先映射 16 MB 大小的内存区域,将这块区域分别以 1 KB,4 KB,8 KB,16 KB,32 KB,64 KB 为单元分成 6 个不同的分区,每个分区分成大小相同的若干单元并建立一个空闲链表和一个已使用链表。

(2) 分配过程。用户申请一块内存,系统根据用户申请的大小确定从哪个分区中分配内存单元,例如此时有一个用户程序申请 25 KB 的内存,因为 16 < 25 < 32,所以选择从 32 KB 的分区中分配,将这个分区的空闲链表的第一个节点对应的内存单元首地址返回给用户,将此节点从空闲链表删除并插入到已使用链表尾部。

(3) 使用过程。用户程序使用内存。

(4) 释放过程。使用完毕,将此节点单元从已使用链表删除并插入空闲链表头部。

采用这种管理方法的优点是:(1)算法的原理与实现简单。(2)响应时间快,分配时分区查找时间是常数 N(分区的数量,此时为 6),其时间复杂度为 $O(1)$,单个链表节点的删除和插入基本都在链表头部或者尾部,不需要花费过多的时间去查找定位。(3)时间性能稳定可靠,分区查找时间和单个链表节点操作时间相对固定,整个过程没有不确定因素影响。(4)因为每次最多只分配一个单元,不存在外部碎片;同时每次查找大小最合适的内存单元,所以内部碎片也相对较少[12]。

4　性能测试与结果分析

分别对实时方案改进前和改进后申请相同大小内存的性能进行对比测试:改进前,使用系统原有的 malloc 函数申请一块内存,并对其读写数据,然后由 free 函数释放这块内存,测试整个过程所花费的时间;改进后,使用 RTmalloc 函数申请同样大小的内存并且读写数据,然后由 RTfree 函数释放,测试所花费的时间。

测试环境:处理器为 Intel(R) Core(TM)2 Duo,频率为 2.70 GHz,物理内存为 2 GB,页面交换空间

为 2 GB,操作系统为 CentOS6.0(Linux3.6.0 内核)平台。当物理内存使用率较高并且内存碎片较多时,系统原有的方法效率低下且花费时间不确定。而改造方案中的内存锁定与内存管理算法主要是解决这个问题的,因此,测试时制造了内存紧张的环境,使内存使用率达到 90%,页面交换率达到 50%。

4.1 对不同大小内存的测试

多次申请不同大小的内存,测试结果如表 1 所示。对测试结果进行对比,如图 5 所示。

表 1 方案改进前后内存性能测试结果

申请内存大小/KB	耗费时间平均值/μs		性能提高率/%
	改进前	改进后	
1	79	40	49.4
4	95	48	49.5
8	124	64	48.4
16	150	84	44.0
32	210	139	33.8
64	327	214	34.6

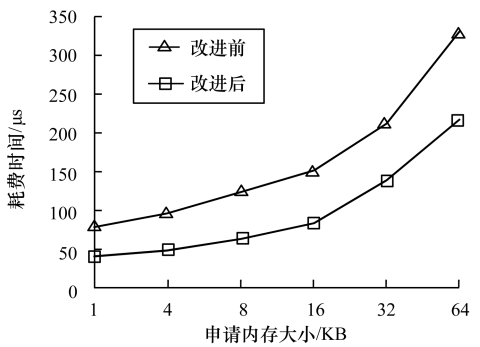

图 5 方案改进前后耗费时间平均值对比

4.2 对不同内存环境的测试

在内存紧张的环境下(内存使用率 90%,页面交换率 50%),申请 16 KB 大小内存,测试方案改进前后所花费的时间结果如图 6 所示。

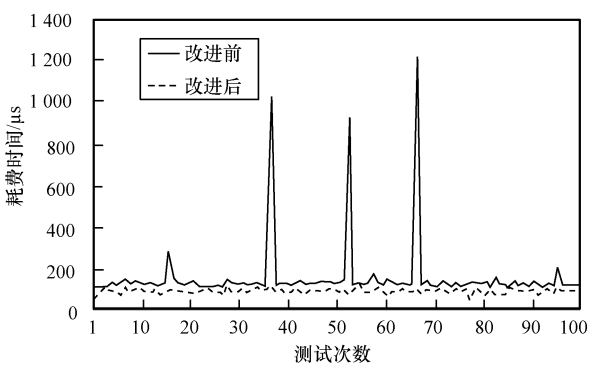

图 6 内存紧张环境下耗费时间对比

在内存不紧张的环境下(内存使用率 15%,页面交换率 0),申请 16 KB 大小内存,测试方案改进前后所花费的时间结果如图 7 所示。将不同环境的性能测试结果进行对比,如表 2 所示。

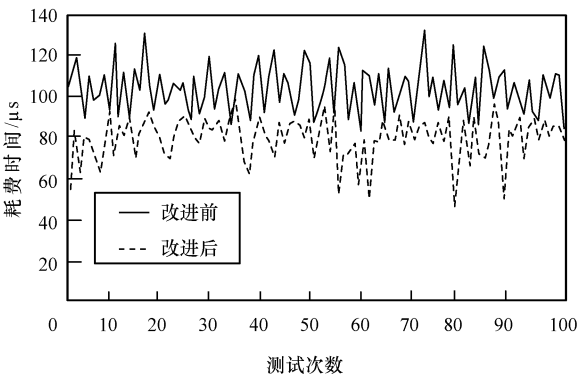

图 7 内存不紧张环境下耗费时间对比

表 2 不同环境下内存性能测试时间平均值 μs

环境	改进前	改进后	耗费时间差
内存紧张环境	150.2	84.0	66.2
内存不紧张环境	103.6	79.0	27.6

从实验结果可以看出,在对相同大小的内存进行多次分配、读写、释放操作时,当内存环境紧张时,改进后的方案所花费的平均时间比改进前所花费的平均时间少,时间性能提高率最高可达 49.5%,时间差的绝对值大,而且改进后的时间稳定性好;当内存环境不紧张时,虽然改进后的方案所花费时间也比改进前少,但是绝对值小,时间稳定性对比效果也不明显(因为改进前后都没有页面交换)。也就是说,内存环境紧张时,改进方案的效果要比内存环境不紧张时的效果更加明显。

综合起来看,改进后的内存方案较好地提高了 Linux 系统在内存管理方面的实时性,可以应用在对实时性要求较高的任务中。

5 结束语

本文提出了一种 Linux 系统内存管理实时化改进方案,主要通过内存映射、内存锁定及新的内存管理算法改进内存管理中的非实时性因素。实验测试结果表明,该方案能够满足 Linux 系统中实时任务对内存操作的实时性要求,适用于系统内存使用率高且又对实时性要求较高的场合。由于在内存管理算法中初始化时限制了可分配内存大小,使得该方案不够灵活,下一步需要解决分配过程中可能出现的一些特殊情况,扩大算法的适用范围。

(下转第 299 页)

图 8　本文方法实验结果

表 1　系统占用 FPGA 的资源利用情况

Slice Logic 利用数	已用	可用	利用率/%
Number of Slice Registers	1 875	11 440	16
Number of Slice LUTs	2 067	5 720	36
Number of RAM16BWERs	19	32	59
Number of MCBs	1	2	50
Number of PLL_ADVs	1	2	50

6　结束语

本文利用 FPGA 在设计上可实现硬件并行和流水线技术,提出一种在 FPGA 上实现四路 CIF 视频合成为一路 D1 视频的设计方法,对合成的一路 D1 视频进行中值滤波去噪。中值滤波采用文献[12]提出的改进算法,本文给出了该算法在 FPGA 上实现的逻辑结构,实现的滤波算法只有 6 个时钟周期的延时。实验结果表明,该方法利用 FPGA 实现了硬件并行和流水线技术,从而保证了视频的实时处理,没有出现丢帧的现象。在保证视频处理的实时性情况下,对中值滤波算法进行改进,加入阈值比较或者自适应处理,是今后研究的内容。

参考文献

[1] 沈淦松,叶玉堂,刘　霖,等. FPGA 软硬件协同处理实时图像处理系统[J]. 光电工程,2012,39(10):143-150.

[2] 李　波,葛宝珊,李　炜,等. 基于通用 DSP 的多模式视频编码器[J]. 计算机学报,2004,27(12):1648-1656.

[3] 樊兆宾,史忠科,杨　珺. 基于视频的车辆检测系统设计[J]. 计算机工程,2008,34(6):255-257.

[4] Iakovidou C, Vonikakis V, Andreadis I. FPGA Implementation of a Real-time Biologically Inspired Image Enhancement Algorithm[J]. Journal of Real-time Image Processing,2008,3(4):269-287.

[5] 周文晖,杜　歆,叶秀清,等. 基于 FPGA 的双目立体视觉系统[J]. 中国图象图形学报,2005,10(9):1166-1170.

[6] 冯伟昌,林玉池,何　冬,等. 基于 FPGA 的双通道实时图像处理系统[J]. 传感技术学报,2010,23(8):1118-1122.

[7] 王春平,张晓华,赵　翔. Xilinx 可编程逻辑器件设计与开发(基础篇)[M]. 北京:人民邮电出版社,2011.

[8] Hu Y, Ji H. Research on Image Median Filtering Algorithm and Its FPGA Implementation[C]//Proc. of Global Congress on Intelligent Systems. Xiamen, China: [s. n.],2009:226-230.

[9] Gundam M, Charalampidis D. Median Filter on FPGAs [C]//Proc. of the 44th IEEE Southeastern Symposium on System Theory. Jacksonville, USA: IEEE Systems Theory Press,2012:83-87.

[10] Fahmy S A, Cheung P Y K, Luk W. High-throughput One-dimensional Median and Weighted Median Filters on FPGA[J]. IET Computers & Digital Techniques, 2009,3(4):384-394.

[11] Arce G R, McLoughlin M. Theoretical Analysis of the Max/Median Filter [J]. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1987, 35(1): 60-69.

[12] Nieminen A, Neuvo Y. Comments on Theoretical Analysis of the Max/median Filter [J]. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1988, 36(5): 826-827.

编辑　索书志

(上接第 294 页)

参考文献

[1] 周保余,孔德刚,赵宏伟,等. 嵌入式 Linux 实时性研究[J]. 吉林大学学报:信息科学版,2011,29(4):339-340.

[2] Mauerer W. 深入 Linux 内核架构[M]. 郭　旭,译. 北京:人民邮电出版社,2010.

[3] 杨　峰. 基于 Linux 内核的动态内存管理机制的实现[J]. 计算机工程,2010,36(9):85-86.

[4] Cwiklinski L, Kicinski W. Management of Memory in a Real-time Measurement System Based on a Signal Processor[J]. Metrology and Measurement Systems,2010, (4):589-598.

[5] Bovet D P, Cesati M. 深入理解 Linux 内核[M]. 陈莉君,张琼声,张宏伟,译. 3 版. 北京:中国电力出版社,2007.

[6] 任桥伟. Linux 内核修炼之道[M]. 北京:人民邮电出版社,2010.

[7] 康　华. Linux 内核空间与用户空间信息交互方法[EB/OL]. (2011-11-23). http://www. kerneltravel. net/jiaoliu/005. htm.

[8] 刘　斌,朱程荣. Linux 内核与用户空间通信机制研究[J]. 电脑知识与技术,2012,8(16):70-71,103.

[9] Gorman M. 深入理解 Linux 虚拟内存管理[M]. 北京:北京航空航天大学出版社,2006.

[10] 俞勤丰,孙　涌. μC/OS-Ⅱ中内存管理方法的分析及改进[J]. 计算机工程,2009,35(11):280-281.

[11] 王小银,陈莉君. Linux 内核中内存池的实现及应用[J]. 西安邮电学院学报,2011,16(4):40-43.

[12] 李满丽. 复杂嵌入式系统内存管理方案的研究与实现[D]. 厦门:厦门大学,2009.

编辑　顾逸斐