嵌入式系统的高速缓存管理

钟锐[1],方文楷[2]

(东南大学 电子科学与工程学院,江苏 南京 210096;东南大学 集成电路学院,江苏 南京 210096)

摘要:本文针对嵌入式文件系统提出了一套基于最久未使用页面替换算法的高速缓存管理方案。该高速缓存管理模块的使用,较好地提高文件系统的读写性能。以 NorFlash 操作为例,使用高速缓存以后,可以使多数量、小文件的写入速度提升 20%左右,读出速度提升 30%- 40%。对于大容量数据传输,适当地调整缓存的容量,也可使得写入速度提升 2%,读出速度提升 10%左右。

关键词:高速缓存管理方案;回写;最久未使用页面替换算法;块设备

中图分类号:TP311　　**文献标识码**:A　　**文章编号**:1009- 3044(2008)12- 20393- 05

Embedded System Cache Management

ZHONG Rui[1],FANG Wen- kai[2]

(1.School of Electronics Science and Engeering, Southeast University, Nanjing 210096,China;2.School of Integrated Circuit, Southeast University,Nanjing 210096,China)

Abstract: A new file cache management based on Least Recently Used (LRU) was bring forward and realized in this paper. The impact of this cache management on embedded system files is proved to be positive. NorFlash as an example, after the use of this cache management, the write speed of mass and little files upgrade 20%, and read speed upgrade from 30% to 40%. To large- capacity data transmission, proper adjustment of cache capacity, we can also makes writing speed upgrade 2% and read rate increase of around 10%.

Key words: Cache Management; Write- back; Least Recently Used; Block device

1 论文研究背景

嵌入式系统的运行环境远不同于 PC 机;嵌入式系统的存储介质多为块设备,其存储原理与磁盘设备差异很大;桌面文件系统为提高读写性能尽可能使用较大的文件缓存,嵌入式系统必须针对资源限制设计特定的缓存管理单元,并尽量适应于块设备的操作特点。

目前,嵌入式领域有多种文件系统可供选择:RomFS 是只读文件系统,可以放在 ROM 空间,也可以放在系统 RAM 中,通常用作嵌入式 Linux 的根文件系统;JFFS/JFFS2 是专为 Flash 设计的日志文件系统,它们提供了很好的掉电保护措施,并能平衡读写 Flash 设备,但是它们需要占用大量 RAM 空间,在文件系统接近满时,它们速度会变得很慢,同时启动的时候需要扫描日志节点,不太适合大容量存储应用;Yaffs 是专为 NandFlash 设计的日志文件系统,其针对 NandFlash 访问的特点进行优化(如块的读写、坏块管理等),但欠缺通用性;Ext2 是 Linux 下使用的文件系统,具有支持存储容量大,访问速度快等优点,但是对 Flash 存储设备支持不好;FAT 文件系统是目前桌面系统中使用最广泛的文件系统,其文件管理方便、通用性好,但是同样不能直接用于嵌入式应用,它的缓存管理单元主要针对 pc 机的磁盘管理,运用于块存储设备并不能有效提升文件的读写速度。我们由此进行了专项开发。东南大学国家专用集成电路工程技术中心针对这一需求进行了将 FAT 文件系统运用到嵌入式系统的研究工作。

2 高速缓存管理原理

高速缓冲存储器(Cache),是一种加速内存或磁盘存取的技术,主要用来提升系统响应的速度。其作用的原理在于使用较快速的存储装置保留一份从慢速存储装置中所读取的数据,当需要再从较慢的存储体中读写数据时,Cache 能够使读写的动作先在快速的装置上完成。这样可以使得系统的响应较为快速。

这种技术如果仅仅使用在读数据这一方面,则没有任何问题。如果还要用在写上,即写入 Cache 中的数据不立即回写真正的存储体,则一旦电源中断或出现其它意外会导致数据的流失。而每次都将数据写入真正的存储体,会使 Cache 只能发挥加速读取的功能,而不能加速写入。这样的情况使 Cache 的写入方式分为两类:

(1)直写式(Write- through):每次遇到写入 Cache 的同时,也把数据写入真正的存储设备,以保证 Cache 和真正的存储设备中的单元数据的一致性。直写式系统简单可靠,但每次更新 Cache 时都要对真正的存储设备写入,因此速度受到了影响。

(2)回写式(Write- back):数据一般只写到 Cache,这样可能出现 Cache 中的数据得到更新而存储体中的数据不变的情况。但此时可在 Cache 中设一标志表示数据是否需要回写,等系统有空或这部分 Cache 需要存储其它数据或等到一定的时间后,再将数据写回存储设备,这种做法是用承担一点风险来换取效率的。

在嵌入式文件系统中,同样需要使用高速缓冲技术来提升文件系统的性能。由于嵌入式系统的资源有限,同时针对嵌入式系统块设备的存储特点(块读、块写),需要设计针对嵌入式应用特点的高速缓存管理。本论文设计了一种针对嵌入式应用特点的高速缓存管理单元,其具有如下特点:(1)软件模块化设计,提供统一接口,便于移植到不同平台;(2)采用回写方式提升写入效率,同时具有一定的掉电保护措施;(3)设计最久未使用页面替换算法(LRU)提升 Cache 系统的命中率;(4)以块设备的块大小为缓冲单元,实现对块设备读写的优化;下面将具体介绍本论文设计的高速缓存管理模块的实现方法。

3 实现代码

3.1 文件系统 Cache 的主数据结构

收稿日期:2007- 12- 17

本论文的 Cache 管理系统将 Cache 设计成保存在内存中的一个结构体数组。数组的每一个元素缓冲了存储设备的一个簇的数据,同时还保存了为便于操作的其他必要信息。Cache 结构定义的代码清单如 List 3.1 所示:

List 3.1 文件系统 Cache 的主要数据结构

```
typedef struct _Disk_Cache{
    U8  DriveNo;  //设备号,
    U8  Flag;     //状态
    U16  RW_ID;  //读/写标志
    U32  CluNo;   //缓冲的簇号
    U8  buf[DISK_CACHE_SIZE]; //缓冲//数据区
}DISK_CACHE;
```

Cache 的大小可以采用静态分配的方法,其代码见 List 3.2。

List 3.2 Cache 存储大小分配

```
DISK_CACHE DiskCache[MAX_DISK_CACHE] //Cache 大小
```

其中 MAX_DISK_CACHE 的值,如果太小则 Cache 性能提升不明显,太大则占据大量系统内存资源。在实际应用中,需要根据系统资源和性能要求作出合理的折衷。

3.2 Cache 管理的主要操作

本论文设计的 Cache 管理可以作为一个独立的 Cache 管理模块,与文件系统本身有一定的独立性。Cache 管理模块对文件系统提供了统一的 API 接口。表 1 为所使用 API 的列表,本节将介绍这些函数的实现过程。

表 1 Cache 管理的 API 函数

函数名	函数功能
CacheInit	Cache 管理系统初始化
OpenClu	打开一个文件簇
GetCache	获取一个 Cache
WriteClu	将 Cache 中的一项数据置需回写标志
CloseClu	关闭簇
CacheWriteBack	以 Cache 数组下标回写指定文件簇
AllCacheWriteBack	回写所有 Cache 项
WriteBackDrive	回写指定设备上的所有在 Cache 中项
WriteBackClu	回写指定设备的指定簇
CluinCache	查看指定设备的指定簇是否在 Cache 中

3.2.1 初始化操作

在使用 Cache 前必须要对各 Cache 元素进行初始化操作. Cache 初始化是调用函数 CacheInit()完成的。函数 CacheInit()的代码如 List 3.3 所示。代码很简单,仅仅是把 Cache 数组所有元素的所有成员赋一定的初值。

3.2.2 打开存储设备的簇

在读写存储设备的扇区之前必须打开它,这时通过调用函数 OpenClu()实现的。函数 OpenClu()有两个参数,其中 DriverNo 指出需要操作的设备号,index 指出在指定设备上需要操作的簇号。此函数将返回一个指针指向簇数据的读写位置。如果返回值为 NULL,则表示簇打开失败。函数原型如 List 3.4 所示。

List 3.4 函数 OpenClu()的原型

```
U8 * OpenClu(U8 DriveNo, U32 index)
```

3.2.3 分配 Cache 项

在前面的打开簇函数中使用到了为簇分配 Cache 项函数 GetCache(),该函数是 Cache 管理的核心,它提供了一种 Cache 页面替换算法。

所谓 Cache 页面替换算法,就是当 Cache 空间已满时,文件系统必须从 Cache 中选择一项删除掉以便为即将被调入的数据让出空间。被删除的项将根据被修改与否决定是否直接从 Cache 删除还是回写如存储设备。

Cache 的效能依算法的不同有好坏之分,估量的单位通常使用命中率。很明显把最不常使用的页面替换出缓存是提高命中率的好方法。为了找出最不常使用的缓存项替换掉,人们已经在理论和实践上对页面替换算法进行了深入的研究。

目前研究的页面替换算法主要有:最近未使用页面替换算法、先进先出页面替换算法、时钟页面替换算法、最久未使用页面替换算法等。

本论文使用的是最久未使用页面替换算法(LRU)。该算法是基于这样的设想:很久没有使用的数据簇可能在未来较长一段时间内也不会被使用。因此 LRU 算法的替换策略是替换那些最久未被使用的数据簇。

本论文分配 Cache 项函数 GetCache()核心的工作就是使用 LRU 算法找出可以被替换的数据簇。该函数结合函数 OpenClu()中每次访问 Cache 都把所有 Cache 元素 RW_ID 成员加 1 的功能,找出 RW_ID 参数值最大的一个 Cache 元素(可能有多个,选择最先找到的),将其回写存储设备(如果需要的话)并替换出去。

List 3.5 函数 GetCache()

```
U16 GetCache(void){
    U16 Max;
    U16 i,j;
    Max = 0;
    j = 0;
    for(i = 0; i<MAX_DISK_CACHE; i++){ ){
        if(Max<=DiskCache[i].RW_ID){
            Max = DiskCache[i].RW_ID;
            j = i;
        }
        if(Max == (U16)(-1)){
            break; //如 Max 已是最大值,
//则不要继续搜索
        }
    }
    if(j<MAX_DISK_CACHE)
    {
        if(DiskCache[j].DriveNo ! =
EMPTY_DRIVE){    (1)
          if((DiskCache[j].Flag&
CACHE_WRITE)! =0) {
              CacheWriteBack(j);
          }
        }
    }
    return j;}
```

List 3.5(1)中 DiskCache[j]即为要替换的 Cache 数组元素。CacheWriteBack()函数实现了将缓冲数据写回物理存储设备的功能。Cache 管理系统中,此部分需要跟物理设备操作有关,移植的时候需要加以改写。

3.2.4 从设备读数据到 Cache

函数 OpenClu()仅仅为设备的簇分配了 Cache,如果程序需要事先把簇中的数据读出来,需要调用函数 ReadClu()。函数 ReadClu()的参数同样是需要读取簇的设备号和设备内簇号。此函数将返回一个状态码,其中 RETURN_OK 表示成功,SECTOR_READ_ERR 表示底层驱动读操作失败,NOT_FIND_DISK 表明没有找到指定的设备等。

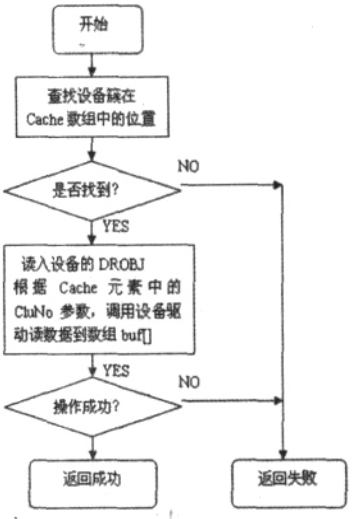

图 1　读设备簇到 Cache 流程

3.2.5 置写标志函数

根据前面对于 Cache 数据结构的介绍,需要回写的 Cache 块的 Flag 元素需标记为已被修改。函数 WriteClu()就是完成这样的功能。它有两个参数,设备号和要操作的设备簇号,该函数首先根据设备号和簇号查找 Cache 数组 DiskCache[],找到设备簇在 Cache 中的缓存数组元素,然后将该元素的 Flag 标志置为 CACHE_WRITE(或操作)。

3.2.6 回写存储设备

函数 CacheWriteBack()实现改动数据的回写。该函数只有一个参数 index,为 Cache 数组的下标号,即根据下标号,读取指定 Cache 项的设局结构,根据 DriveNo 成员决定的设备号,读取设备的 DROBJ 结构(在其它文件系统中可能不同)以获取操作设备的信息,然后将 Cache 的 Flag 标记置回无需回写状态并调用设备驱动完成回写操作。

3.3 Cache 管理的掉电处理

本论文采用回写式的 Cache 管理。前面提到这种方法为提升性能而承担了一定的风险,即有可能在 Cache 中数据尚未回写时发生了掉电等异常情况导致系统失去响应,这样新写入的数据将丢失,之前的写入操作失败。为了在一定程度上解决此问题,本论文采用如下规则:(1)在设备关闭时 Cache 中所有与之相关的项必须回写;(2)在系统空闲的时候,可以运行一个优先级较低的回写任

务,该任务完成查找 Cache 数组中所有成员标志为已改写的元素,将其回写。

另外,文件系统掉电保护还包括 FAT 表与文件数据一致性的问题,即在文件数据操作和 FAT 表更新操作之间发生掉电等异常,导致文件数据与 FAT 表数据不一致。对于此问题的处理,本论文采用如下的规则:

(1)有修改完 FAT 表后,才允许删除对应簇的无效数据;(2)应用程序调用了一个文件操作接口之后返回之前,必须完成 FAT 表的更新以及 FAT 表的备份;(3)文件的新数据在写入物理存储设备之后,才允许修改 FAT 表。

采用上述措施后,即使写入某簇的时候发生错误,也可以采取错误处理进行补救,或者直接返回给上层的应用,或者重新申请一个新的簇。总之,不至于在 FAT 表上出现不合事实的错误。

3.4 Cache 管理在文件系统中的应用

上述 Cache 管理操作是以文件簇为单位的,而文件系统最终通过设备驱动以簇为单位操作设备上的数据的,因此只需对设备驱动作一定的修改,在驱动中调用表 3.1 中的 Cache 管理 API 函数就可以实现用上述 Cache 管理方法管理设备上的数据了。

以 NorFlash 驱动为例,介绍 Cache 管理的添加。根据 2.5.1 节关于驱动的介绍可知 NorFlash 驱动中与存储介质进行数据交互的函数是 nor_rd_io(),需要对此函数进行改写。为便于说明,将此函数的原型列如下:

int nor_rd_io(U16 driveno, U32 page, void *buffer, U16 count, int do_read)

函数中 driveno 为操作扇区号,page 为操作的簇号,buffer 为读写数据的缓冲区,count 为操作的次数,do_read 为读写的标志(1 为读,0 为写)。

首先根据函数的读写标志确定操作类型。如果是读操作,则先调用函数 CluinCache()判断要读取的文件簇是否在 Cache 中已经缓存,如已经缓存,则根据缓存的数组下标,直接读 Cache 项 buf[]中存储的数据到 buffer 中;若不在缓存中,则通过替换算法在 Cache 中找到一项用于替换,调用 NorFlash 驱动,将设备上的数据同时写入 buffer 和 Cache 项的 buf[]元素中,以便于下次访问使用方便。对于写操作,可以首先判断所写的簇是否在 Cache 中,如在,则更新 Cache 项中 buf[]内容;如不在,则根据替换算法找到新的 Cache 项,并将 buffer 写入新 Cache 项中 buf[]。写操作采用的是 Cache 回写策略,此时并没有回写物理设备,回写物理设备的操作是在 Cache 回写函数中完成的。

上述高速缓存的使用,有效地减少了设备的存取次数,提高了系统的通过能力并降低平均响应时间,特别是在做数据传输时,数据停留在高速缓存中,直到系统认为适当的时候才进行数据传输。另外,最久未使用页面替换算法(LRU)的使用大大提高了 Cache 系统的命中率,提高了文件系统的读写性能。

4 测试与评估

4.1 测试平台

本调试平台由基于 SEP3203 的实验开发板,JTAG 仿真器和运行于主机上的集成开发环境(IDE)组成。将开发板通过 JTAG 仿真器连接到 PC 机上的集成调试环境(IDE)软件平台,在 IDE 中统一完成汇编/C 语言的编辑、编译、连接。IDE 选择 ARM 公司的开发软件(SDT2.5、ADS1.2),利用处理机的 Embedded- ICE 性能,通过 JTAG 接口实现实时的仿真调试。使用 ARM ADS,并通过 JTAG 仿真器,就可以实现针对 ARM 处理器开发板的板极调试。目前 ARM ADS 的最新版本为 1.2。本论文使用 ADS 的集成开发环境 CodeWarrior IDE 作为代码编译的工具,运用 ARM 扩展调试器 AXD 进行代码的调试和结果观察。

为了全面测试加上 Cache 管理模块后文件系统性能的提升,现以 NorFlash 的读写为例,将测试过程分为如下三步骤:

4.1.1 小文件写入测试

将 Cache 数组的元素参数 MAX_DISK_CACHE 设为 20,根据第三章介绍的文件系统 API 函数,在设备上建立 10 个文件,每个文件中写入 512 字节的任意数据,写完之后关闭上述文件。测试整个过程的完成时间。将文件数目增加到 20,文件大小增加到 1024 字节,重复测试得下表所示结果。

表 2 小文件写入测试(使用 Cache)

	10 个文件	20 个文件
每文件 512 字节	29.9ms	55.3ms
每文件 1024 字节	30.2ms	56.2ms

未使用 Cache 管理系统的情况下,重复上述测试过程结果如下。

表 3 小文件写入测试(未使用 Cache)

	10 个文件	20 个文件
每文件 512 字节	56.3ms	120.5ms
每文件 1024 字节	60.4	122.4ms

通过比较两次测试的结果可知,采用回写式 Cache 管理的文件系统在写入操作时减少了对存储设备的访问次数,大大提升了对多次、小文件的写入速度。上述两表中对于写入相同文件数目而文件大小不同情况下,写入速度差别并不大。这是因为文件大小都没有超过 NorFlash 操作的基本单位——块。

4.1.2 小文件读出测试

对于上面创建的文件再调用文件系统 API 函数,测试文件的读出速度。测试分两种情况。首先,在写入后立即读出,此时数据仍在 Cache 中,可以作为使用 Cache 情况下读出速度的测试;然后,将设备掉电,Cache 中数据清除,再测量读出时间,此时可以作为未使用 Cache 下的读速度。按照这样的测试过程,得到如下两表。

表 4 小文件读出测试(使用 Cache)

	10 个文件	20 个文件
每文件 512 字节	12.1ms	25.8ms
每文件 1024 字节	12.6ms	26.1ms

表 5 小文件读出测试(未使用 Cache)

	10 个文件	20 个文件
每文件 512 字节	20.3ms	35.6ms
每文件 1024 字节	21.4	36.3ms

根据上面两表结果可以看出在使用 Cache 和未使用 Cache 的情况下,小文件的读出速度提高明显,约有 30%-40%的提升。

4.1.3 大容量数据的读写测试

在 Flash 上只建立一个文件,向文件中分别写入大小不等的数据,测试文件的写入速度如下表所示。

表 6 NorFlash 写速度(KB/s)比较

测试容量	128K	512K	1M	2M	4M
未用 Cache	50.47	54.86	51.40	51.81	54.73
使用 Cache	55.62	56.41	53.56	59.23	56.52

从上表可以看出,当写入的容量较大时,写入速度提升的并不明显。这是因为当写入的数据比 Cache 的容量大的多时,写入数据 Cache 的缓冲作用将不大,数据近似地等价于是直接地写入物理设备。测试的结果与理论分析相吻合。

将写入的大容量数据进行读出测试,测试的结果如表 7 所示。

表 7 NorFlash 读速度(KB/s)比较

测试容量	128K	512K	1M	2M	4M
未使用 Cache	5013.3	5217.5	5114.2	5308.2	5335.6
使用 Cache	5977.2	5815.3	5869.6	5623.4	5423.5

可见读出与 Cache 容量相差越大的数据,读速度差别越小。从理论上分析可知,Cache 中不能够为大容量的数据缓存,每次读取数据时 Cache 的命中率很低,均须访问实际物理设备,因而读速度提升不明显。

实际应用中,应该首先知道系统可供文件系统使用的内存大小,根据可用内存情况设置高速缓存的大小。同时还应该知道,应用中需要经常传输文件的大小,如容量较小,则缓存单元可选用块的大小,如需要传输的文件都很大,也可将数块作为基本缓冲单元。总之,实际应用中,可以根据需要合理定制缓存参数,最终达到性能和资源消耗的较好折衷。

5 结束语

本文介绍了一种针对嵌入式应用的高速缓存管理方案。该高速缓存管理具有模块化、易于移植的特点,且采用了最久未使用页面替换算法(LRU)作为缓存替换策略,方法简洁、资源占有率低且具有较高的缓存命中率,同时该缓存管理还具有一定的掉电保护措施。经实践测试本方案能有效的提升嵌入式系统的读写速度。

参考文献:

[1] 卓越. FAT 文件系统在嵌入式块设备系统中的优化[D].东南大学,2007.
[2] 硬盘 FAT 文件系统原理的详细分析[DB/OL]. http://www.vshj.com/.
[3] Microsoft Crop. FAT32 File System Specification, 2000.
[4] 汤铮.基于嵌入式技术的电力装置文件系统及逻辑组态的研究[D].南京 东南大学电力系统及其自动化系,2005.
[5] 孙涛. 基于 ARM 的嵌入式闪存文件系统的研究与实践[D].湖北工业大学计算机系,2005.
[6] 赵俊才.DOS 文件系统在 Flash 存储介质上的实现[D].中国科学院计算机计数研究所,1999.
[7] 张延虎.嵌入式设备中文件系统的研究与实现[D].北京科技大学控制理论与控制工程系,2001.
[8] 冯福香.LINUX 文件系统的性能、可扩展性和缓冲技术研究[D].南京大学软工程系,2004.
[9] Bart Broekman,Edwin Notenboom(美) .嵌入式软件测试[M].电子工业出版社,2004.
[10] Patton(美) .软件测试(原书第 2 版) [M].机械工业出版社.2001.