NAND FLASH文件系统的设计与实现

张 雪 杨春林 黄 娟
(四川师范大学计算机科学学院 四川 成都 610068)

【摘 要】:本文在现有的 Flash文件系统的基础上,设计并实行了一个专门针对于 NAND型 Flash的文件系统。此文件系统的基本结构类似于 YAFFS,但是,此文件系统对数据结点管理、垃圾回收和扫描加载等方面做了改进,使其性能有了很大提高。

【关键词】:文件系统; Flash; NAND Flash; YAFFS

1.概述

现今,电子产品层出不穷,随着嵌入式技术在各种电子产品中广泛而深入的应用,越来越多的数据需要被暂时或长期的保存起来。磁盘是最常用的数据存储,但是嵌入式系统有其自身的特点,一般它要求设备具有体积小、功耗低、便于携带、防震、能适应恶劣的环境等特点。由此可见,嵌入式数据存储的首选介质应该是 Flash存储器,而不是磁盘。

Flash存储器的最大特点是每个数据位只能由 1改写为 0,而不能由 0改写为 1。如果要把一个数据位由 0变为 1,需要擦除此位所在的块,但每个块的擦除此时是有限的,为 10万次到 100万次不等。

NAND Flash存储器是 Flash的一种,以高密度、大容量、高数据存储速率以及更多的擦除次数等特点,逐渐成为大容量嵌入式数据存储设备应用的主流。NAND Flash的基本组成单位是页,页是 NAND Flash数据存储的基本单位。每个数据页由两部分组成数据区(date space)和附加区(spare space),数据区用于常规数据的存储,附加区可用来存储数据页的相关属性。

2. Flash文件系统介绍

目前存在多种可以应用于 NAND Flash的文件系统,按其结构主要分为两类:基于块设备的常规文件系统和日志结构文件系统。

基于块设备的文件系统是磁盘存储器最常用的文件系统,最典型的就是的 FAT文件系统。Flash设备并不是块设备,要想在 Flash上运行像 FAT这样的文件系统,需要通过一个中间层把 Flash模拟成一个块设备,这个中间层就是转译层(FTL),它把 Flash模拟成与磁盘相似的块设备,由此像 FAT这样的文件系统才能够应用在 Flash上。

虽然通过 FTL可以在 Flash上应用像 FAT这样的文件系统,但是会存在很多问题。首先,文件系统通过 FTL来访问 Flash设备,进行数据传输,这会在很大程度上降低文件系统的效率。更主要的是,在像 FAT这样的文件系统中,所有文件的索引信息都集中存放在像文件分配表这样的指定位置,当有数据需要更新时,必需修改其对应的索引信息。频繁的修改索引信息势必会导致存放索引信息的存储块很快损坏。由此看来,针对 Flash存储器的特性,设计新型的文件系统是非常必要的。

日志结构文件系统就是专门针对 Flash设备而设计的文件系统,在日志型文件系统中,文件的基本组成单位是数据结点。当文件有新的数据需要添加或是更新时,新的数据结点写入空闲空间,不是覆写原有的数据,只是通过数据结点中的一个标记位将原有的过期的数据结点标记为无效,这也保证了意外情况下数据存储的完整性。通过循环使用各个擦除块也实现了存储区的磨损均衡(wear leveling),由于文件更新时不进行覆写操作,不进行较耗时的擦除操作,数据的写入速度将有所提高。由此看来,日志结构文件系统更适合于 Flash设备。

目前基于 NAND Flash的日志结构文件系统主要是 YAFFS系列。YAFFS系列的文件系统是第一个专门针对 NAND型 Flash而设计的日志结构的文件系统,它考虑到了 NAND型 Flash的结构属性,提高了文件系统的加载速度、降低了内存的损耗。本文就是在现有的文件系统的基础上,设计了一种新的 NAND型 Flash文件系统,以提高各方面的性能。

4. NAND FLASH文件系统的设计

本文件系统(ZFFS)整体结构如图所示:

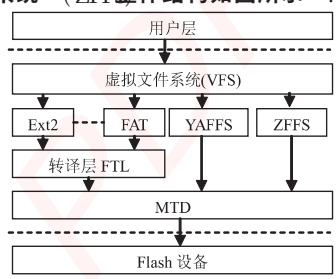

图 1 文件系统的整体结构

ZFFS对文件系统以下的几个方面做了改进和优化:数据结点的管理、文件系统的启动、垃圾回收磨损平衡。

4数据节点的管理

本文件系统是在 linux下实现的,所以每个文件的属性信息仍然保存在 inode结点中,包括 inode号、用户 id、组 id、时间戳等信息,除此外,inode还包含了文件名和指向父 inode的指针。这样就去处了目录项结构,在一定程度上降低了数据更新的频率。在 ZFFS中,一个文件由多个数据结点组成,每个结点的大小是一个数据页。数据页的数据区存储数据,附加区存储相应的属性信息。在每个数据页附加区中的属性信息有几项属性信息是非常重要的:块号(chunk number),inode号(inode number)和错误校验信息(EDC/ECC),每个 inode结点在 Flash存储器上也占有一个数据页。和 EXT2类似,inode中包含文件数据所在页的索引(index entry),由于一个 inode结点占一个数据页,那么 inode结点中可以包含很多对文件数据的索引,比如一个页的大小是 512B,除索引外的其他信息占 256B,一个索引项占 4B,那么一个 inode就可以有(512-256)/4=64个数据索引。由此我们也可以计算出一个文件的最大值应为:64*512=32K,这个值对于一个较大的文件来说显然是不够的,本文件系统把最后一个索引项设置成一个二级索引。这样一个文件的最大值就可以达到无限大,如图 2所示:

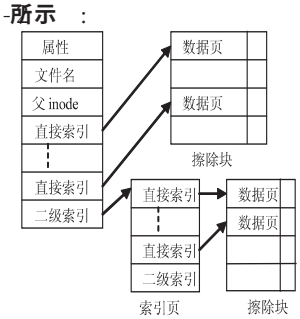

图 2 ZFFS文件索引结构

在 YAFFS文件系统中,文件的属性信息(inode结点)和文件的数据并没有分开存储,一个擦除块上即存储了属性信息又存储了数据信息,ZFFS并没有采取这种方式,而是将文件系统的控制信息、inode结点和数据分别存储于不同的擦除块中,即一个擦除块要么完全用于存储 inode结点,要么完全用于存储文件的数据。inode结点信息和数据不同时存储于一个擦除块。文件

的数据索引信息都存储在 inode结点中 所以在系统启动的过程中 只需要扫描存储 inode结点的擦除块就可以建立文件系统的整体结构 不必 YAFFS一样 扫描每一个擦除块 这就大大缩短文件系统的启动时间。

系统在启动时需要知道哪些擦除块存储的是 inode结点 这样才能对其进行扫描。 ZFFS将存储 inode结点的擦除块的地址存储在 Flash的第一个块 这个块被称作 inode映射块 (inode map block), inode映射块共有 部分组成 如下图所示 :

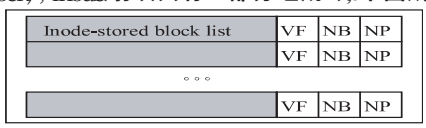

图 -3 inode映射块结构

页的数据区存储 inode结点块的地址 当一个页耗尽时 向下一个相邻的页继续存储。 ZFFS用两个字节存储 inode块的块号 这样 一个页可以存储的 inode块就是页尺寸的一半 那么 inode映射块可以存储的 inode块的总数就是 :

(number of pages)*(size of page/2)

通常情况下这个值远大于用于存储 inode结点的擦除块的数目。 VF, NB, N这三个部分的内容存储在页的附加区 , V用于标志 ZFFS上次运行时是否被正确的卸载 如果被正确的卸载 那么 inode映射块中的信息就是有效的。 N表示的 inode结点块的总数。 N表示的是用于存储 inode结点块地址的总页数 即在系统启动时只需要读取 inode映射块的前 N个页。为了保持系统的一致性 这三个部分被重复写入每一个页的附加区。

4文件系统的加载

ZFFS的加载分两步。首先 检查 inode映射块 查看附加区的 V是否被置位。 V被置位 就说明文件系统在上一次使用后是被正确卸载的 , inode映射块包含的是合法的信息。如果 VF被置位 就根据 N的值把 inode映射块中的有效页读入到内存 建立相应的 hash表。 hash函数就是简单的 inode块号对 hash表的大小进行取模操作。接下来就是根据内存中的 hash表的内容扫描各 inode结点块。系统对 hash表需要进行两种操作 当有新的 inode结点块写入 Flash时 要将相应的信息插入 hash表 , 当有 inode结点块由于垃圾回收被删除时 要将其相应的信息从 hash表中删除。在读取 inode结点块点的数据页时 先核对页附加区的校验信息 看此页上存储的是否是正确的 inode结点。如果校验信息正确 就在内存中为此 inode结点接了相应的内存映象 zffs_inode_cache结构体 此结构的定义如下 :

```
struct zffs_inode_cache{
int ino_number;
char * filename;
... .
struct zffs_inode_cache* parent;
struct zffs_inode_cache* children;
struct zffs_inode_cache* sibling;
... .
};
```

各个映象通过的 parent, children和 sibling这三个域建立这个文件系统的目录结构。位于同一层目录下的文件是兄弟关系 , 他们在内存映象中通过 sibling链成一个链表。

如果 inode映射块中的 V没有被置位 则表明 ZFFS在上次使用后没有被正确的卸载 我们必须扫描 Flash存储器的每一个擦除块。如果被扫描的块是数据块则跳过 如果被扫描的块是 inode块 操作的方法上面已经描述。即使扫描每个块 , ZFFS加载也比其他的文件系统要快 因为对于那些被跳过的块 系统只扫描了第一个页的附加区以确定其是数据块 而不是 inode结点块。

当文件系统被卸载时 要将内存中的 inode hash表写入的 inode映射块 正确写入后对其 V置位 以供下一次加载的使用。

4垃圾回收和磨损平衡

废弃的 inode结点和数据结点占用的空间是垃圾空间 随着系统的运行 垃圾空间越来越多 空白空间会越来越少 当空白空间的大小达到指定的下限后 必须进行垃圾回收以维护系统继续运行。 inode块存储的是文件的属性信息 数据块存储的是数据信息 通常情况下 文件的属性信息被修改的概率远大于数据被修改的概率。文件重命名 或是移动文件都需要修改文件的属性 当有信息被修改是就会有垃圾空间产生。由此可见 , inode块更容易产生垃圾空间 对其进行垃圾回收的概率就比数据块要大。为了平衡各个擦除块被用于存储 inode结点和数据的概率 以达到磨损平衡 , ZFFS将空闲块分成两个链表 分别是 : inode_free_list和 data_free_list inode_free_list保存对 inode块进行回收得到的空闲块 , data_free_list保存对数据块进行回收得到的空闲块。这样就不会导致某些块经常被用于 inode结点的存储 过早的结束其寿命。

结论

本文在现有的文件系统的基础上 重新设计了基于 NAND Flash的文件系统 对数据管理 垃圾回收等方面进行了优化 经测试后 本文件系统的启动时间和磨损平衡要高于同类的 JFFS系列和 YAFFS系列。不足之处是对实时性支持的不好 今后会在现有的基础上对其改进 使其满足实时性要求。

参考文献 :

1. ERAN GAL and SIVAN TOLEDO. Algorithms and Data Structures for Flash Memories. ACM Journal Name. Vol. TBD, No. TBD, TBD 20TBD, 2005: 1- 30.

2. Aleph One Ltd. Yaffs: A NAND- Flash Filesystem. http://www.aleph1.co.uk/yaffs/, 2006- 12- 05.

3. Woodhouse D. Red Hat Inc. JFFS: The Journalling Flash File System. http://sourceware.org/jffs2/jffs2- html, 2007- 01- 11.

李庆诚 孙明达 基于 NAN型闪存的嵌入式文件系统设计 计算机应用研究 . 2006- 04: 231- 233.

(上接第 14页)

熟悉数据库设计方法 并能在教师的指导下完成毕业设计工作。只有在具体技术的应用中 才能真正深入地理解《数据库系统概论》里的相关知识和内容 也才真正达到了学以致用的目的。

4加强实践教学

计算机课程是一门实践性很强的课程 通过上机操作才能使学生真正理解、消化课堂上的理论知识 这一点对数据库的学习尤为重要 为提高上机实习课的质量 要做到如下几点 :

、课堂教学与上机实习环节紧密衔接。既不能提前 也不能滞后 随讲随练 趁热打铁 会取得比较理想的效果。

、作好辅导工作。学生上机时 应随时注意学生的实习过程和情况 并及时给予指导 有问题及时纠正。

总之 在计算机技术和信息处理技术日新月异的今天 对于本课程的教学和数据库技术的发展和应用 已远远不止于此。除了因人而异 因材施教以外 更要博览群书 拓宽视野 迎头赶上。

参考文献 :

萨师煊 王珊 数据库系统概论 第三版)[M北京 高等教育出版社 , 2001.

钟乐海 微机原理及接口技术课程改革探讨 [J]四川师范学院学报 自然科学版), 1999, 20(3): 282 285.

王珊 陈红 数据库系统原理教程 [M北京 清华大学出版社 , 1998.

王珊 数据库系统基础 [M北京 铁道出版社 , 1998.

王珊 陈红 文继荣 数据库和数据库管理系统 [M北京 电子工业出版社 , 1995.

石树刚 郑振楣 关系数据库 [M北京 清华大学出版社 , 1993.

施伯乐 数据库系统导论 [M北京 高等教育出版社 , 1995.