日志文件系统在嵌入式存储设备上的实现*)

The Implementation of Journaling File System on Embedded Memory Device

郑良辰　孙玉芳

(中国科学院软件研究所　北京100080)(北京中科红旗软件技术有限公司　北京100086)

Abstract In embedded systems, unexpected power-off often causes the corruption of the file system and the lose of data. It is necessary to develop a special kind of file system to prevent such corruption. As a kind of journaling file system specially for embedded memory devices, JFFS is just the files system we need. In order to make use of JFFS more extensively, Redflag Software Ltd. Co. has successfully solved the problem about JFFS' implementation on DiskOnChip, a special kind of embedded memory device. This paper mostly discusses the design of JFFS and its implementation on DiskOnChip.

Keywords Embedded memory devices, Journaling file systems, JFFS, DiskOnChip

1. 引言

随着计算机技术的发展，嵌入式系统已成为计算机领域的一个重要组成部分。对嵌入式系统来说，不可预知的非正常断电随时可能发生。对于传统的文件系统来说，非正常的断电会带来文件系统的不一致性，导致文件系统的破坏，严重的时候甚至会导致信息的大量丢失。因此，在嵌入式存储设备上，实现一种特殊的防断电文件系统具有特别重要的意义。JFFS日志文件系统正是这样一种在嵌入式存储设备 Flash 上设计并实现的防断电的文件系统。下面我们将重点讨论 JFFS 的设计原理和它在特殊的 Flash 设备 DiskOnChip 上的实现。

2. 日志文件系统原理分析

2.1 日志文件系统概述

传统的文件系统都是用数据缓冲区来进行文件系统数据的读写，这样能够大大地提高文件系统的效率。但是这样也带来严重的隐患。当非正常断电发生时，缓冲区中的数据可能还没有来得及写回磁盘，引起部分数据的丢失。缓冲区的数据可能包含了文件系统的管理信息，如文件系统超级块，inode 表，inode 节点等数据，如果这些信息丢失的话，会导致这个文件系统数据组织的混乱，严重时甚至会导致文件系统的完全崩溃。

日志文件系统采用记日志的方法来克服这个缺陷。其基本思想是在磁盘上维持一个只允许扩展的日志文件。每次对文件系统的修改都以追加的形式，直接写回磁盘上的日志文件的尾部。因此，日志尾部总是记录着文件系统的最新的状态。这样，在非正常断电导致文件系统破坏后，我们只要查看日志文件系统的尾部，就能够找到文件系统断电前的状态，迅速地恢复系统。

日志文件系统主要分为两大类：Log-Structured 日志文件系统(日志结构文件系统)和 Meta-Data 日志文件系统(元数据日志文件系统)。下面对这两种类型的日志文件系统进行简要的介绍。

2.2 Log-Structured 日志文件系统

4. 4BSD Unix 操作系统中的日志文件系统 BSD-LFS 是 Log-Structured 日志文件系统的典型代表。它的设计思想是：整个磁盘都由日志来管理，日志是文件系统在磁盘上的唯一代表。所有的写操作都在日志的尾部进行。日志是允许扩展的，每次磁盘块被修改后，它总要写往一个新的地址。这就意味着日志中的一些老的数据块变得过时，每隔一定的时间，清除程序就会把它们回收，同时对这个磁盘的日志数据进行整理：清除所有的过时块，把有效的数据集中到日志的逻辑意义上的尾部，这样日志逻辑上的前部就能够被重新利用了。

BSD-LFS 保存了传统的文件系统的一些数据结构，如目录和 Inode 结构以及为了对文件进行逻辑编址而设置的间接块。这样，我们可以通过传统的文件系统一样的方式来检索文件的数据。但是，这样也带来设计的复杂性。比如说，在 BSD-LFS 中，inode 结构也是作为日志的一个部分写入磁盘的，并没有固定的地址。每次修改了 Inode 结构后，文件系统都把它发在日志的不同的位置。这样 BSD-LFS 就需要一个附加的数据结构 Inode 映射表来保存每个 Inode 当前的磁盘位置。所以 BSD-LFS 系统的文件系统的结构也变得十分复杂。BSD-LFS 的文件的写过程如图1所示。

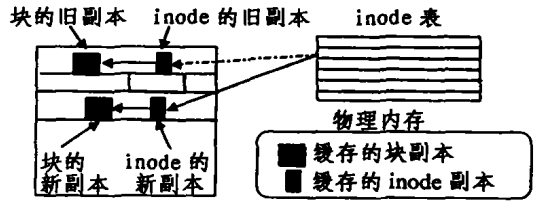

图1　BSD-LFS 文件的写过程

2.3 Meta-Data 日志文件系统

在元数据日志文件系统中，日志只是扩充了文件系统的正常的表示方式。或者在磁盘上划分一块单独的区域，或者在文件系统内部指定一个特殊的文件来保存日志，把元数据(我们把文件系统的管理结构如 Inode、目录块和间接块的修改，超级块、磁盘块组和磁盘分配结构等统称为元数据)的修改记录在日志的尾部。

*)本文得到国家自然科学基金项目60073022、国家863项目8633-306-2D12-14-2和中科院知识创新重大项目 KGCX1-09的资助。郑良辰　硕士研究生，研究方向：系统软件。孙玉芳　博士生导师，研究员，研究方向：系统软件，中文信息处理，大型数据库与网络工程。

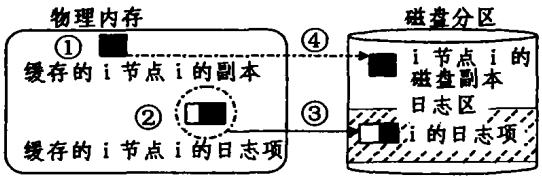

图2 元数据日志的实现

元数据日志文件系统记录日志的过程如图2所示。在元数据日志文件系统中，如果我们对文件系统的操作改变了某个 Inode 的数据，如 setattr 操作，系统的内核将会做如下的四步操作(如图2中①②③④所示)：

①更新缓冲区中 Inode(I 节点的副本)，将其标识为已修改。

②建立一个新的日志项，它的头部是日志项的控制信息，后面是修改后的 Inode 的新的内容。

③把这个 Inode 的日志项直接写回到磁盘日志的尾部。

④在一定的时间缓冲区的管理进程(在 Linux 中是核心线程)将修改后的 Inode 写回磁盘。

这样当非正常断电时，缓存中的 Inode 的副本可能没有来得及写回磁盘，导致了文件系统的不一致性。这时只要检查日志的尾部，就能够迅速地恢复系统。

2.4 两类日志文件系统的优劣简单比较

Meta-Data 日志文件系统的优点是：元数据日志文件系统是对正常的文件系统的扩展，实现起来相对要简单；而且由于元数据的数据量很小，这样系统崩溃后只要检索日志就能够迅速地恢复系统。

Meta-Data 日志文件系统的缺点是：由于元数据日志文件系统只是记录的元数据的变化，它在系统崩溃后能够保证文件系统的完好性，但是无法防止在缓冲区内没有写回磁盘的用户数据的丢失；由于会带来一些附加的管理开销，相对正常的文件系统增加的设计也会导致系统性能的降低。

Log-Structured 日志文件系统的优点是：它能够把文件系统的所有变化都忠实地记录下来，这样在系统崩溃的时候，我们只要找到日志的尾部，就能够准确地把系统恢复到断电前一时刻的状态。

Log-Structured 日志文件系统的缺点是：BSD-LFS 文件系统把这个磁盘当作一个日志，在这个日志内部保持原来文件系统的结构，这样的实现比较复杂，文件系统的效率也会降低；对于大磁盘来说，把这个磁盘作为一个日志，进行垃圾块搜索和清除要整理这个磁盘，必定会耗费大量的系统开销。日常管理的系统开销也是很大的。

综上所述，相对以 BSD-LFS 为代表的日志 Log-Structured 日志文件系统来说，元数据日志文件系统由于实现简单，效率相对要高，渐渐发展成为日志文件系统的主流趋势。然而，在某些特定的应用场合，Log-Structured 日志文件系统是一种更合理的选择。下面要介绍的 JFFS 就是 Log-Structured 日志文件系统的一个特殊版本。

3. JFFS 日志文件系统设计原理

上面提到的各种日志文件系统都有充分的发展，而且有的甚至已经十分成熟。我们是不是能够直接在嵌入式存储设备上使用它们呢？答案是否定的。首先，这些日志文件系统都是基于大磁盘来设计的，无论它们怎样简化其设计，都比正常的文件系统的要复杂。如果把这些日志文件系统直接放到容量很小的嵌入式存储设备上去运行，其效率之低可想而知。而且它们一般都需要较大的空间进行日志管理，这对嵌入式设备来说也是不可能的。其次，嵌入式系统一般都使用 Flash 作为存储设备。Flash 设备有一个显著的特点就是它不能同在一块地方连续写两次。我们如果在某块做了修改，要想写回原来的地址，必须先擦除那一块，然后才能把修改后的信息写回相同的位置。这就决定了 Flash 上的文件系统必须采用特殊的设计方式。

JFFS 对日志文件系统的结构进行了简化，而且结合了 Flash 特殊的属性，是在 Flash 设备上保存经常修改的数据的文件系统的理想选择。下面简要介绍它的设计原理。

3.1 JFFS 的设计思想

JFFS 的全称是 Journaling Flash File System，其性质属于 Log-Structured 日志文件系统：它把整个 Flash 看作成为一个日志，每次对文件的修改都写在日志尾部。同时，它针对 Flash 设备较小的特点，简化了文件系统的表示，避免了已有 Log-Structured 日志文件系统的缺点。

·JFFS 的文件的表示方式 JFFS 简化了文件系统的结构，在 JFFS 中，文件由若干块的数据区组成。在 Flash 设备上，这些数据区的物理地址可能相邻，也可能不相邻。每个数据区最大不超过32kB。

每个数据区由一个固定大小的文件控制结构 jffs_raw_node 和文件的部分数据组成。jffs_raw_node 定义了文件的 inode 号、文件的父亲节点的 inode 号、文件的名称、该数据区版本号、数据区的偏址、数据的尺寸等等。jffs_raw_node 结构如图3所示。

对同一个文件的每一个数据区的控制结构 jffs_raw_node 来说，它们的文件的 inode 号、文件的双亲节点的 inode 号、文件的名称都相同，但具有不同的版本号。一般来讲，每块数据区存放了从偏址开始的该文件的一段数据。

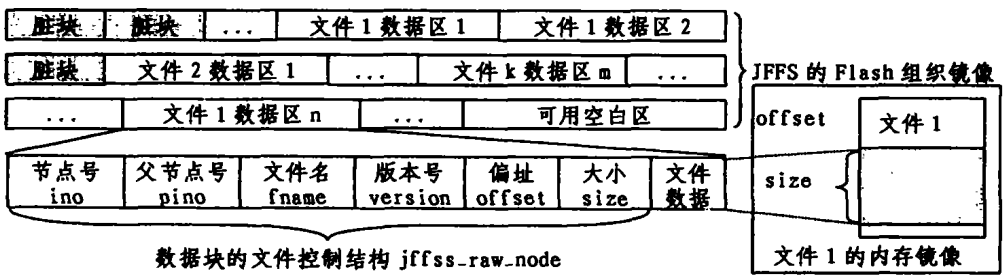

图3 JFFS 的内部设计原理示意图

·JFFS 的写文件方式 在文件的某个数据区中的数据进行了修改后，我们不是直接把修改的数据写回原来的地址，而

是在Flash上的日志尾部申请一块新的可用区域来存放修改后的数据。并把原来的数据区标识成"脏"区。

这样做能够把文件的变化都记录在日志的尾部，在日志尾部维持系统的最新状态，实现系统的防断电的机制。同时它结合了Flash的存储特性。如果要把修改后的数据写回原来的区域，就必须通过费时的擦除过程来擦除原来区域，然后才能把数据写回去。现在我们把修改后的数据直接写到日志尾部新的可用空间上，就不用执行这样的擦除操作。原来的区域仅仅标识为"脏"，只有等到必要时才对它进行擦除，这样能够提高系统的效率。

当Flash上的"脏区"占用的空间达到了一定的数量，或者系统中的可用空间小于某个数量，系统就会激活一个垃圾清除线程，擦除日志头部的"脏区"，使Flash能够作为一个循环的日志使用。

·JFFS的Flash组织　Flash由一些脏区、大量文件的数据区和可用空间组成，它的组织结构如图3所示。

·JFFS的文件的检索　当需要检索一个文件的信息的时候，系统要扫描整个Flash，找出该文件的所有的数据区，在内存中重组，形成该文件的内存镜像结构(不一定把文件的所有信息都读入内存)。由于Flash由很多具有一定大小的各种区域(脏区、文件数据区、可用空间)组成，对Flash扫描时可以以"区域"为单位跳跃着进行，而且Flash本身的容量有限，因此，尽管检索一个文件时需要对着整个Flash进行扫描，但是进行这样扫描的系统开销并不大，系统的效率也不会因此受到影响。

3.2　JFFS文件系统在Flash上的镜像动态变化

开始时，我们可以通过一个工具程序把需要放在Flash上的原始数据做成一个JFFS镜像，写入到Flash上去。这个初始的镜像的所有文件的数据区都集中在日志文件逻辑上前部。

在JFFS运行了一段时间后，会在日志的嵌部形成大量连续的脏数据区。在日志的有用的数据区中间也会有少许的脏数据区。

当所有脏区的大小达到一定的数量，或者系统的可用空间小于一定的数量，系统会激活垃圾清除线程，进行垃圾清除，它会把集中的日志头部的所有的脏区回收。整个变化过程如图4所示。图中灰色块表示文件数据，黑色块表示脏区，空白块表示可用空白区。

当日志的尾部到达了Flash的尾部时，它会跳到日志尾部继续开始。这样它可以循环地使用Flash。

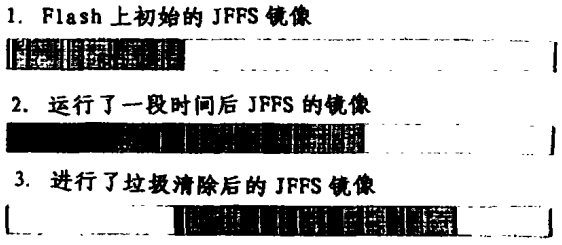

图4　Flash上JFFS镜像变化图

4. JFFS在嵌入式存储设备DiskOnChip上的实现

4.1　DiskOnChip的特殊性

嵌入式存储设备大多为Flash设备。DiskOnChip是一种特殊的Flash设备，在各种嵌入式设备中被广泛使用。它和一般的Flash有着很大的不同：在DOC芯片中，它有一个特殊的控制器。从DOC芯片物理地址0开始的48k这样的区域里，存放了一段特殊的防火墙的程序代码，它会在系统BIOS自检时被调入并驻留内存，这段程序会对DOC的读写进行监控。它能够协助在DOC上模拟传统的磁盘块设备，是我们能够像使用传统的磁盘一样使用DOC，这也是DiskOnChip名称的来由。一般的Flash是不能像磁盘一样使用的。

在Linux操作系统中，使用DOC不仅仅需要DOC的硬件驱动，而且还需要一个中间的协议实现层NFTL (NAND FLASH Translation Layer)。NFTL层在DOC驱动上实现块设备的模拟，向上层提供了一个可分区的块设备接口。传统的文件系统如EXT2就可以通过NFTL来建立在DOC上。而JFFS则不同，JFFS在文件系统内部之间对Flash设备进行管理，它直接访问Flash的底层驱动，不需要任何中间层来为它模拟块设备。传统的文件系统、JFFS、NFTL、DOC驱动的关系如图7所示。一般来说，DOC的驱动接口函数只考虑了如何和NFTL层来交互，没有考虑为JFFS提供支持的问题。这就为我们在DOC上实现JFFS带来了一定的困难。

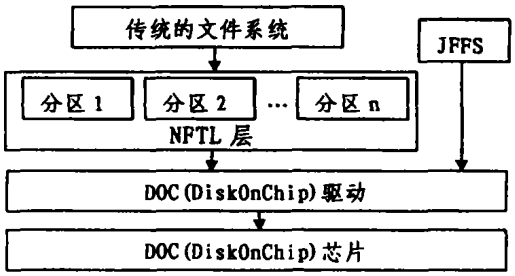

图5　传统文件系统、JFFS、NFTL、DOC驱动关系

4.2　JFFS在DiskOnChip上的问题一和解决方法

JFFS在DOC上的问题之一是：NFTL层和DOC的驱动的关系有两个重要的特点：NFTL向DOC驱动提交的写请求一次最多不超过512个字节；NFTL每次提交一块(512大小)的擦除请求。因此，DOC的驱动在它的写函数DOC_WRITE()中默认只处理不超过512个字节的请求。在它的擦除函数DOC_Erase()默认也只擦除一个512个字节的块。这对DOC的驱动的实现也是很方便的。因为在系统启动后，DOC把它的读写寄存器(CDSN_IO_Register)映射到了系统内存扩展内存区的一个512节的内存窗口上，所以对于不超过512个字节IO请求来说，直接通过对这个窗口一个序列的操作就可以完成，实现起来很简单。但是对于JFFS来说，JFFS需要直接访问DOC驱动，它向DOC提交的写请求很多情况下肯定会超过512个字节，而且肯定会提交多块的擦除请求。这就带来了矛盾。

上述问题的解决办法是：修改DOC_Write()函数，在其中加入判断，如果写请求的字节数大于512，就进行多次循环，每个循环的一系列的操作仅仅处理不超过512个字节，最后一次循环处理的字节可能小于512个字节。对DOC_Erase也要做类似的修改。具体实现是繁琐的，鉴于篇幅，就不讨论实现细节了。

4.3　JFFS在DiskOnChip上实现的问题二和解决方法

JFFS在DOC上的问题之二是：在嵌入式系统中，绝大部分的系统信息是不需要修改的，只有很少的一部分系统数据用户可能经常修改。这样出于效率的考虑，我们可以在D-

(下转第52页)

动画效果?

结束语 作为世界上第一个基于对象的多媒体压缩标准,MPEG-4首次定义了专门的人脸动画工具,使得人脸动画可以更加广泛地应用在电影与广告、视频会议、可视电话、人机接口、虚拟环境、游戏娱乐等诸多领域中。由于MPEG-4只给出了人脸动画的功能描述,并没有规定人脸动画的具体实现方法,这就使得我们可以设计一个基于MPEG-4的人脸动画系统,在该系统中研究人脸动画技术。除了人脸动画技术之外,进一步的研究包括基于MPEG-4的人体动画技术,最终形成一套完整的虚拟人物系统。

参 考 文 献

1 The MPEG Home Page. http://www.cselt.it/mpeg/

2 ISO/IEC IS 14496-1 Systems. 1999

3 ISO/IEC IS 14496-2 Visual. 1999

4 ISO/IEC IS 14496-3 Audio, 1999

5 Ekman P,Friesen W V. Facial Action Coding System. Consulting Psychologists Press, Palo Alto, CA, 1978

6 Kala P,et al. Simulation of facial muscle actions based on rational free form deformations, Proc. Eurographics'92, 1992. 59～69

7 Waters K. A muscle model for animating three-dimensional facial expression. Computer Graphics (Proc. SIGGRAPH'87),1987,21(4):17～24

8 Lee Y C, Terzopoulos D, Waters K. Realistic modeling for facial animation. In:Proc. SIGGRAPH'95, 1995. 55～62

9 Lavagetto F, Pockaj R. The facial animation engine: toward a high-level interface for the design of MPEG-4 compliant animated faces, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY,1999,9(2):277～289

10 Lavagetto F,Pockaj R,Costa M. Smooth surface interpolation and texture adaptation for MPEG-4 compliant calibration of 3D head models. Image and Vision Computing,2000,18:345～353

11 Pighin F,Hecker J,Lischinski D,Szeliski R,Salesin D H. Synthesizing realistic facial expressions from photographs. In: Proc. SIGGRAPH'98, 1998. 75～84

12 Ostermann J,Haratsch E. An animation definition interface: Rapid design of MPEG-4 compliant animated faces and bodies. In:Proc. Intl. Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging, 1997. 216～219

13 Hartman J,Wernecke J. The VRML Handbook, Addison-Wesley, Reading, MA, 1996

14 ISO/IEC 14772-1. Information Technology-Computer Graphics and Image Processing-The Virtual Reality Modeling Language-Part 1: Functional specification and UTF-8 encoding, 1997

15 Murakami S, Tadokoro H. Generation of facial expressions based on time functions. In: Proc. Intl. Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging,1997. 212～215

(上接第74页)

OC上使用只读的传统的文件系统来保存所占比重很大的那部分不变的信息。而另外划分一块相对较小的区域来使用JFFS存储经常修改的数据。这样做的好处是:对JFFS来说,在一定程度上,文件系统的容量越小,它的效率会越高。而相对较大的使用传统文件系统的区域,由于它的属性是只读的,非正常的断电也不可能对它造成破坏。这样组合起来使用系统的效率就会提高很多。

但是对于DOC来说,我们如果要使用传统的文件系统,必须使用NFTL层,NFTL需要使用整个硬件设备,而JFFS也需要使用一个硬件设备。DOC驱动在检测DOC硬件时,一块DOC只是当作一个逻辑设备加入到系统。那么,我们如何才能让JFFS和正常文件系统在一块DOC上共存呢?

上述问题的解决办法是:在原理上也不复杂,我们可以修改DOC的驱动,使DOC驱动在检测硬件时,把一块DOC作为两块逻辑设备加入到系统中来,同时修改驱动接口函数,让这两个逻辑设备对上层来说,好像是一个独立的设备,这样我们就能够在其中一块上通过NFTL来使用正常的文件系统,而在另外一块上使用JFFS。鉴于篇幅,我们略去具体的实现。

4.4 DiskOnChip上的JFFS和正常文件系统的性能比较

①通常的性能比较:经过我们的测试,DOC上的JFFS和DOC上的传统文件系统在读写速度上它们没明显的差异。但是当垃圾清除的时间到来时,JFFS会锁住整块设备,会有数秒的等待时间。只是垃圾清除动作发生的频度不高,整体上对JFFS系统的系统没有太大的影响。

②防断电性能比较:DOC上的传统文件系统只要有一次非正常断电就会报告文件系统错误。如果不经过修复,连续非正常断电数次就可能导致文件系统的严重破坏。即便是每次断电以后都进行修复,连续非正常的断电后的次数大于10次以上就肯定会导致文件系统的严重破坏。

DOC上的JFFS在经历了几天断电试验,非正常断电几百次后系统还能够保持完好。

从上面可以看出,JFFS在防断电方面有优良的性能,而且相对于传统的文件系统来说,性能并没有明显的降低,是嵌入式存储设备中用来存放经常修改的数据的理想选择。

小结 JFFS是一个正在发展的文件系统。它能够在嵌入式存储设备上做到防止非正常断电导致的文件系统破坏和数据丢失。但是目前它还有很多的不足,比如说垃圾清除耗时过长,系统还不是足够稳定等等。在DOC上JFFS的实现也仅仅是刚刚完成,还需要长时间的测试和错误修复。不过随着时间的推移,JFFS也会日益成熟,也必将会在嵌入式领域广泛使用。

参 考 文 献

1 http://www.linux-mtd.infradead.org/ (MTD主页)

2 http://oss.software.ibm.com/developerworks/opensource/jfs/ (JFS主页)

3 http://www.freebsd.org/docs/en/books/design-44bsd/ (4.4BSD的网上技术文档)

4 Leffler S J,McKusick M K,Quarterman J S. The Design and Implementation of the 4.3BSD Unix Operating System, Addison-Wesley, Reading ,MA, 1989