嵌入式 Linux 文件系统剪裁方法研究*)

姜春茂[1] 段 莹[2] 黄春梅[1]
(哈尔滨师范大学数学与计算机学院 哈尔滨 150500)[1] (哈尔滨工业大学软件学院 哈尔滨 150001)[2]

摘 要 从三个方面(初始化、装载及内部实现)介绍了 VFS 的实现机制,提出嵌入式 Linux 文件系统的剪裁方法,并给出一个 EXT2 文件系统精简实例。

关键词 嵌入式 Linux,操作系统,剪裁

Research on Tailoring Embedded Linux File System

JIANG Chun-mao[1] DUAN Ying[2] HUANG Chun-mei[1]
(College of Math and Computer, Harbin Normal University, Harbin 150500, China)[1]
(College of Software, Harbin Institute of Technology, Harbin 150001, China)[2]

Abstract Introduces the VFS realization mechanism from three aspects, which include initialization, loading and internal realization, proposes the tailor method of embedded Linux file system, and gives a simplification example of EXT2 file system.

Keywords Embedded Linux, Operating system, Tailor

1 引言

文件系统是操作系统的重要组成部分之一。它负责管理外存上的文件,并为操作系统和用户提供文件的存取、共享和保护等功能。文件系统设计的好坏对系统安全和性能有着很大影响。

Linux 是一个迅速发展的、性能卓越的、具有巨大发展潜力的操作系统。它的重要特征之一就是虚拟文件系统抽象机制。Linux 文件系统的总体结构可分为三个部分:

第一部分是 Virtual File System Switch(VFS),这是 Linux 文件系统对外的接口,任何要使用文件系统的程序都必须经由这层接口;

第二部分是 Cache,Linux 通过这一层缓冲机制来加快速度;

第三部分就是实际的文件系统,像 Ext2,VFAT 等等。

为避免混淆,通常将 VFS 及缓冲机制统称为 VFS(Virtual File System)。VFS 是用户程序空间与实际文件系统之间的一层接口软件,它将不同的文件系统的具体实现统一起来,并向上提供一层统一的系统调用接口[1]。VFS 只存在于内存空间,它在系统启动时建立,在系统关闭时消亡。Linux 就是以这样的结构得以支持多达几十种文件系统。

Linux 的文件系统是一个结构清晰的文件系统。在用户进程对文件系统提出操作请求后,虚拟文件系统将内存的数据结构与具体的文件系统的数据结构关联起来,同时将调用具体的文件系统的操作函数,启动设备的输入/输出操作,实现设备上文件的读取、写回、查找、更改、更新等操作。当然,由虚拟文件系统提供的内存节点缓冲区、内存目录项缓冲区、数据块缓冲区提供了内存中操作节点、目录、数据块的手段,实现文件系统尽量在内存中处理文件,减少读取外设的操作次数。在操作完成之后,文件系统在适当的时机将调用虚拟文件系统的更新例程,将改变的数据从内存中全部写回外部设备。

2 VFS 实现机制

2.1 文件系统的初始化

在操作系统启动并进行初始化的过程中,系统调用 file-system_setup()例程进行文件系统初始化,它根据系统的配置参数调用各实际文件系统的初始化例程——init_fsname_fs(),此过程向 VFS 进行注册 register_filesystem。

VFS 在内存中维护一个文件系统类型列表,全局指针变量为 file_systems,新注册的文件系统将一个描述该文件系统的数据结构 file_systemtype 传入 VFS,将它插入链表 file_systems 的结尾。文件系统的初始化执行过程如图 1 所示。

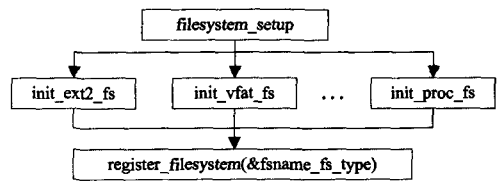

图 1 文件系统的初始化执行过程

这样,VFS 就可以遍历 file_systems 链表来决定内核支持哪些文件系统类型。

2.2 文件系统的装载

实际的文件系统是内建到系统内核中,也可作为可载入模块在需要时进行装载。系统在内存中维护一个已装载的文件系统列表,全局指针变量为 vfsrrntlist。另一个指针 vfsrrn-ttail 指向表中最后一项,mru_vfsrrnt 指针指向最近使用的文件系统[2]。装载例程执行如下几个步骤:

(1)搜索文件系统类型列表,若未找到则返回错误码。

(2)检查物理设备是否存在且还没有安装,是则找到装载

*)黑龙江省教育厅项目(基于 pda 的 linux 裁剪技术研究与应用,11531449),哈尔滨师范大学校级科研项目 KM2006-28。姜春茂 硕士,副教授,研究方向为操作系统、ERP 等。

点的目录 i 节点，检查是否是目录类型且尚未装载其它的文件系统。

(3)在 super_blocks 中搜索空闲的 super_block 结构，若没有搜索到，则分配一个超级块结构，将它初始化，然后加入到 super_blocks 向量中。

(4)运行所装载文件系统的读超级块例程，这个读例程从物理设备读取信息，填充 VFS 超级块结构的各个域。fstype→read_super(s,data,silent)。

(5)增加新的装载点。分配新的装载点结构，进行初始化设置，然后加入到 vfsrrntlist 表中，并修改 vfsrrnttail 指针。

2.3 VFS 的内部实现

Linux 文件系统的数据结构类似 EXT2 文件系统，即采用超级块与 inode 节点数据结构进行管理。超级块结构驻留在内存空间，存放了该文件系统的重要信息。从超级块中可以取到该文件系统中任何一个文件的 inode 节点，从 inode 节点则可对该文件进行读写等操作，这就实现了对磁盘中任一文件的控制。

2.3.1 超级块

super block 结构存放本文件系统的重要信息，包括一些基本信息：一组 super block 结构的操作函数；一组与 quota 管理有关的函数；管理 super block 所属文件系统 inode 的信息；一些处理 superblock 的同步字段及各个文件系统本身所特有的信息。

(1) super block 的同步机制管理

```
unsigned char s_lock;
struct wait_queue * s_wait;
```

上面这两个字段是用来处理 super block 的同步。s_lock 记录着目前 super block 是否被锁住，如果是，其值为 1；若不是则为 0。s_wait 是一个 wait queue 的结构，被放到 queue 里的进程将会进入 sleep 的状态，直到被叫醒为止，并且，如果要改变 super block 的内容，需要先调用 lock_super()锁住 super block，以免产生竞争条件。操作完之后则要调用 unlock_super()将 lock 释放掉。

(2)文件系统本身特定信息的管理

super_block 结构是所有文件系统所共同使用的一个结构，但是，除了共同的部分之外，不同的文件系统之间也有相当的差异性，因此，在 super_block 结构有一个字段是专门用来存放各个文件系统所独有的信息。这些信息是在注册文件系统时调用 read_super()所填入的。

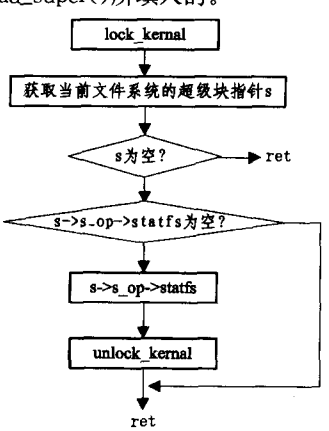

图 2 执行过程图

因为每个 super_block 在同一时间内最多只会记录一个文件系统的资料，所以，这个字段是 union。例如 EXT2 文件系统 ext2_sb 就是专门存放 ext2 文件系统本身所额外需要的信息，由 ext2_read__super()函数填入的。

(3)超级块操作指针

Linux 在注册文件系统时，将操作指针添入，当进行超级块操作时，通过操作指针指向实际文件系统的具体执行函数，实现 VFS 向下一级文件系统的映射。

以系统调用 statfs 为例，该调用的执行过程如图 2 所示。

2.3.2 inode 节点

inode 节点唯一地表示文件系统中某一个目录或文件，是 Linux 管理文件系统的最基本单位。inode 结构包括 inode 节点链表管理字段、inode 基本信息、inode 同步管理字段、与内存管理有关的字段、Quota 管理字段、跟 file lock 有关的字段以及一组用来操作 inode 的函数。

inode 结构前三个字段是用来将 inode 链接起来的字段，分别是：

```
struct list_head i_hash;
struct list_head i_list;
struct list_head i_dentry;
```

在 VFS 里，用几个链表来管理 inode。inode_unused 用来将目前未使用的 inode 链接在一起[3]。inode_in_use 用来将目前正在使用的 inode 链接在一起，当一个 inode 被使用时，它会从 inode_unused 中被取出来，接着它会利用 i_list 字段放到 inode_in_use 中。sb→s_dirty 用来将标志为“脏”的 inode 链接在一起。这个链表的表头位于 super block 的 s_dirty 字段，使用 i_list 链接。另外，所有正在使用中的 inode 都可以经由 inode_in_use 链表找到，但是，因为系统的 node 太多，该链可能会很长，查找的速度较慢，因此，Linux 构造了 inode 节点哈希表 inode_hashtable，每个使用中的 inode 都会计算出其 hash value，并且放到 hash table 中。

2.3.3 文件描述符管理

Linux 通过几个间接的指针从文件描述符指向 inode 节点结构，例如，当一个进程执行 open 系统调用时，内核返回一个非负整数值，这个整数就是文件结构指针数组的索引值，而每一个文件结构指针通过 file→f_dentry 指向 dentry 结构，dentry 结构又通过 dentry→d_inode 指向 inode 节点结构。

3 文件系统的选择与改进

我们知道，在标准的 Linux 系统中，对块设备的存取使用的是逻辑文件系统，一个逻辑文件系统被装配在虚拟文件系统的一个装配点上。Linux 支持几十种逻辑文件系统，但是在嵌入式系统中，这些功能不是必须的，这些是可以裁减的。同时，因为嵌入式系统的应用范围非常广泛，所以不可能有一种文件系统在嵌入式 Linux 系统中一统天下，适用于大到嵌入式服务器、工控系统，小到嵌入式手表、PDA 机顶盒等的所有情况。真正有经验的开发者必须根据系统应用环境、目标、配置等信息来选择构建合适的文件系统。例如需要存储大量数据的场合，还是需要在系统中配置普通硬盘，搭建一个大容量、高吞吐率的文件系统。而在以 Flash 为主要存储设备的便携型嵌入式系统中，为了尽量节省空间，延长 Flash 使用寿命，则需要选择 cramfs 等文件系统。要想成功地开发完成一个嵌入式 Linux 系统应用，就必须在平时积累关于 Linux 各种文件系统的知识，熟悉其性能和操作。

过去，除了 ext2 等通用目的文件系统以外，没有专门针

对嵌入式环境的文件系统，因此使用 Linux 作为嵌入式设备的操作系统时，有些情况下就需要自己修改文件系统代码，来满足特定目的的需要。随着 cramfs，JFFS2 等文件系统的成熟，在嵌入式环境下，也有了很多好用的 Linux 文件系统可供选用，降低了开发难度，减少了开发时间。

上面已经论述了在 Linux 中，文件系统是如何挂上系统并且和系统进行通信的。Linux 既然通过 VFS 来和各种不同文件系统发生联系，不难考虑只要将 super_block 数据结构中的联合中的成员使用自己文件系统的数据结构去替代就可以了。同样，要把 VFS inode 的联合使用自己文件系统的数据结构去替代。然后编写自己的系统调用，就可以了。

在建立自己的文件系统的时候，最好仿照 EXT2 文件系统的方法，组织自己的逻辑文件，组织自己的数据结构。EXT2 文件系统在组织文件上的优秀性是有目共睹的，比如支持长文件名、高扩展性等。但是对于一个嵌入式系统，这些功能和实现这些功能的数据结构反而显得多余，如 MINIX 文件系统，要求文件名不超过 14 个字符，这就能够满足嵌入式环境的需要了[4]。所以改进的较好的方法是在 EXT2 文件系统中去掉对嵌入式系统来说多余的、繁琐的信息，是切实可行的做法，也是一种节省工作量的做法。

然而修改一个文件系统，不是很容易的事情，因为涉及到很多方面的事情，所有关系到打开文件列表，进程创建、运行和结束等一系列的过程和文件系统相关联，相应的代码都需要修改。内存方面，凡是与交换文件有关的代码也需要改动。当然，也可以采用 MINIX 文件系统。

4 EXT2 文件系统精简实例

在本例中我们使用了两个 Ext2 文件系统分区，一个存放 Linux 系统，作为 root 分区，另一个主要存放应用程序和数据。Ext2 文件系统映像存放在 Flash 中。系统运行时，这两个文件系统映像被复制到内存中，并装载到 Linux 的虚拟文件系统上。所以这两个文件系统完全运行在内存内。

根据上面的分析，我们对 Ext2 文件系统进行了简化。主要针对 Ext2 文件系统的数据块预分配机制。Ext2 文件系统中的文件占用若干数据块。文件扩展时，首先向最后一个数据块的空闲部分写入数据。如果不够，则申请空闲数据块。Ext2 数据块预分配机制是在从文件系统分区分配数据块时，文件系统为该文件一次预分配若干连续的数据块；当文件被再次扩展，并要申请新数据块时，文件系统首先查看该文件是否有预分配的数据块。如果有，则直接使用其中第一个预分配数据块；如果没有，才真正从文件系统分区获取空闲数据块。为了避免文件系统碎片，提高访问效率，预分配的数据块尽量接近最后一个数据块。

预分配机制的主要目的是减少由于文件扩展造成的文件系统碎片，并且提高分配数据块的效率。使用预分配机制的必要性是：在低速外部存储设备上访问若干非连续数据块的速度比访问连续数据块慢很多。这在硬盘之类需要机械运动的存储设备上非常明显。使用预分配机制的可行性是：外部存储设备的存储资源非常丰富，不会因为预分配使得文件系统空闲数据块不足。

显然，对于我们使用的 Strong ARM 平台嵌入式 Linux 而言，情况正好相反：在内存中访问连续数据块与非访问连续数据块的速度差别不大；而 Ext2 文件系统分区却很小，由于整个 Ext2 文件系统都运行在内存中，碎片不会对文件访问效率造成影响；而预分配数据块却会提前耗尽系统空闲数据块。例如，用于存放用户应用数据的分区通常只有 128kB－512kB，视用户应用程序和用户数据大小而定。而默认情况下，Ext2 文件系统的数据块大小是 1 kB，预分配机制每次预分配 8 个数据块，这样平均每个文件多占用 4kB。因为系统中大部分文件仅数十 kB，所以这部分开销很可观。数据块预分配机制对于我们使用的 Strong ARM 平台的 Linux 环境是不适用的。我们在 Ext2 文件系统实现中去除了数据块预分配机制。这样，当文件扩展并要申请新数据块时，文件系统不再查找是否有预分配的数据块，而是直接在该文件最后一个数据块相邻区域内查找并分配且只分配一个空闲数据块。

实验表明，我们使用的 Strong ARM 平台上使用去除了数据块预分配机制的 Ext2 文件系统，文件操作的效率没有受到明显影响。以受预分配机制影响最大的文件写操作为例，我们采用每次写 1kB 的方式连续扩展文件，对采用和不采用预分配机制的 Ext2 文件系统进行比较，以执行文件操作的时间长度作为考察目标。jiffies 是 Linux 内核的毫秒级计数器，我们将扩展写入操作前后记录的 jiffies 的差值作为文件写操作的时间开销，写 16kB 以下的文件时，文件系统用时小于或接近 1 毫秒，计时存在较大相对误差。写 64kB 以上文件用时基本呈线性增长。但是无论哪一种大小，使用预分配机制的 Ext2 文件系统和不采用预分配机制的 Ext2 文件系统进行扩展写入操作的时间开销基本一致，效率没有明显差别。

结束语 Linux 是一个功能强大的操作系统，并且拥有大量的应用程序，将 Linux 移植到嵌入式系统，能够充分利用这些成熟的技术，提高开发效率和开发质量。为了使 Linux 成为真正的嵌入式操作系统，除了开发驱动程序，让 Linux 运行在嵌入式硬件平台上，还需要针对嵌入式系统应用的具体需求对 Linux 进行裁减和优化，为此，一方面我们利用 Linux 模块化的特点配置 Linux 内核，在我们使用的 Strong ARM 平台已经建立起一个功能类似桌面 Linux 的嵌入式 Linux 环境，而其体积和运行开销很小，能够满足手持移动设备的应用需求。另一方面，根据应用的特点对 Linux 实现进行优化。实验表明，我们对 Ext2 文件系统预分配机制修改适用于我们的基于 Strong ARM 平台的嵌入式 Linux 环境，在对性能没有明显影响的情况下节省系统开销。当然，在这个方面我们还有许多工作要做，例如为了进一步提高文件系统效率，需要对缓冲缓存机制进行优化，以及使用更加紧凑的文件系统。此外，进程管理、内存管理等模块也需要针对嵌入式应用的特点加以优化。而对于数据管理简单的应用，可以把文件系统以设备驱动的形式实现。

参 考 文 献

[1] 沈玉利. Linux 的虚拟文件系统中数据结构的研究[J]. 湛江海洋大学学报，2001，21(3)：61-62

[2] 杨益，郭庆平. Linux 虚拟文件系统实现技术剖析[J]. 交通与计算机，2001，19(101)：46-49

[3] 刘章仪. Linux 文件系统分析[J]. 贵州工业大学学报(自然科学版)，2002，31(4)：135-137

[4] Rosenblum M，Ousterhout J K. The Design and Implementation of a Log-Structured File System // Proc. Thirteenth Symp. on Operating System Principles. ACM，1991：1-15