第34卷 第23期 计 算 机 工 程 2008年12月
Vol.34 No.23 Computer Engineering December 2008
·工程应用技术与实现· 文章编号：1000—3428(2008)23—0249—03 文献标识码：A 中图分类号：TP311

可移植嵌入式文件系统的设计与实现

白瑞林，张 道
(江南大学智能控制研究所，无锡 214122)

摘 要：嵌入式系统需要处理大容量数据，其存储介质具有多样性。该文提出一种遵循FAT标准、能在多种介质上移植的嵌入式文件系统(EFS)。分析EFS的体系结构，给出底层驱动的设计方法。实验结果表明，EFS支持FAT12/16/32格式文件且可靠性高，它采用模块化设计，提供标准API函数，具有较高可移植性。该系统已应用于实际产品中。

关键词：嵌入式；文件系统；可移植系统

Design and Implementation of Transplantable Embedded File System

BAI Rui-lin, ZHANG Dao
(Institute of Intelligent Control, Jiangnan University, Wuxi 214122)

【Abstract】 Mass data processing and variety of memory medium is existed in the embedded system. This paper designs an Embedded File System(EFS) is which followed by the File Allocation Table(FAT) specification and can be transplanted to variety of the medium. It analyzes the system structure of EFS and lists the bottom-driven design method based on it. Experimental results show that EFS supports FAT12/16/32 format, has high reliability, and adopts modular design to provide standard API function so that it can be transplanted easily. This system is used in the actual products.

【Key words】 embedded; file system; transplantable system

1 概述

嵌入式系统被越来越广泛地应用，数据存储与管理已成为该领域的重要研究课题之一。存储介质的多样性和复杂性为嵌入式系统的数据管理提供了可实现的灵活载体，但也对数据管理规范化提出了更高要求。因此，出现了嵌入式文件系统。

目前已有多种嵌入式文件系统，主要包括TureFFS, JFFS2和文件分配表(File Allocate Table, FAT)等。TrueFFS不是严格意义上的文件系统，它只提供了中间层，必须在上层配合其他文件系统使用。JFFS2是日志结构文件系统，包括写平衡、垃圾收集、回滚操作、支持数据压缩等功能。但JFFS2文件系统加载时需要进行全存储区扫描，启动时间长并会占用大量内存资源[1]。FAT文件系统以其技术成熟、结构简单、系统资源开销小等特点被广泛应用于通用计算机系统中。

因为嵌入式系统资源有限，所以要求嵌入式系统软件必须尽可能小巧、稳定且高效。嵌入式系统与通用计算机系统的运行环境具有明显差异，因此，通用计算机系统中成熟的文件系统并不适合直接用于嵌入式系统，必须对其进行相应修改。基于上述考虑，本文提出一种可在多种存储介质中使用的嵌入式文件系统(Embedded File System, EFS)，它与FAT标准完全兼容，支持在运行中动态添加存储设备驱动[2]。EFS完全采用ANSI C语言来实现，与应用程序的接口仅为少量符合可移植操作系统接口(Portable Operating System Interface, POSIX)标准的应用程序接口(Application Programming Interface, API)函数调用。开发人员只要针对不同存储设备编写少量设备驱动函数，就可以将EFS方便地移植到多种嵌入式系统中。

2 EFS的体系结构

为了方便移植，需要尽量减少软件实现之间的耦合性，EFS采用模块化结构设计，使用底层驱动程序来兼容不同硬件结构和不同介质。EFS按功能可以分为2个模块，即文件系统模块和设备驱动管理模块。EFS的体系结构如图1所示，其中，箭头表示模块之间的调用关系。

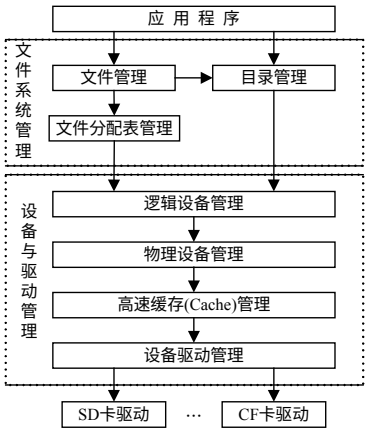

图1 EFS的体系结构

应用程序不属于嵌入式文件系统，它是EFS的使用者，通过文件系统提供的API函数进行文件和设备管理操作。为了增强应用的可移植性，并便于程序员进行快速开发，文件系统提供的API函数应符合POSIX标准。

作者简介：白瑞林(1955－)，男，教授，主研方向：智能控制与嵌入式系统；张 道，硕士研究生

收稿日期：2008-03-05 **E-mail：**bairuilin@hotmail.com

文件是文件系统中的一个对象，它在文件系统控制的内存中具有一个逻辑数据结构，文件系统通过对这个数据结构的访问，实现对文件的操作。目录是一个特殊文件，它存放的不是用户数据，而是文件目录项，这个目录中的每个文件和下一层的目录都拥有一个目录项。该目录项记录了文件或下一层目录名字、文件长度、目录数据块位置等信息。文件系统管理模块主要实现对文件和目录的管理，大部分与文件和目录操作相关的 API 函数都在这个模块中定义，它是直接与应用程序接口的模块。

存储文件数据的设备称为文件系统设备，文件系统设备向文件系统提供数据的物理存储服务。设备与驱动管理模块记录了设备驱动程序和设备本身的信息，该模块从逻辑和物理 2 个层次描述了文件数据的存储设备。物理设备控制块描述物理设备的数据结构，用以记录设备驱动程序和设备本身的信息。逻辑设备是对物理设备进行划分或变换后的、可以被文件系统以一定逻辑进行理解的抽象设备。

EFS 主要通过底层驱动程序实现对多种介质和硬件的兼容。底层驱动为上层程序提供与设备无关的接口，EFS 通过这个接口访问实际硬件。由于嵌入式系统硬件差别很大，因此需要为不同硬件结构编写相应底层驱动。

3 EFS 的设计

3.1 文件系统管理模块设计

文件系统管理模块主要包括文件管理(file.c)、目录管理(dir.c)和文件分配表管理(fat.c)等文件。

文件管理提供给应用程序的 API 函数包括 FileOpen(打开文件)、FileClose(关闭文件)、FileRead(读文件)和 FileWrite(写文件)等。以 FileRead 为例说明函数的原型，其功能是从文件中读取指定大小的内容并将其放入指定的缓冲区，具体如下：

输入 file，指向以 read 方式打开的文件的指针；size，从文件中读取的字节数；buf，指向存放数据缓冲区的指针

输出 实际读取的字节数

uint32 FileRead (File *file, uint32 size, uint8 *buf)

目录管理提供的函数包括 MakeDir(创建目录)、OpenDir(打开目录)、ChangeDir(改变目录)等，以 MakeDir()为例说明函数原型，其功能是创建一个新的目录，具体如下：

输入 fs，指向 FileSystem(结构体)类的指针；dirname，指向新创建目录(该目录名为 dirname)的指针

输出 0，操作成功；1，目录已经存在；2，路径错误

int8 MakeDir(FileSystem *fs,int8 * dirname)

文件分配表管理 FAT 文件系统的重要数据结构，EFS 用到的一个较重要的数据结构体如下(根据结构体成员名称可以知道各个成员代表的含义)：

```
struct FileSystem{
        Partition *part;
        VolumeId volumeId;     // VolumeId 本身也是一个结构体
        uint32 DataClusterCount;
        uint32 FatSectorCount;
        uint32 SectorCount;
        uint32 FirstSectorRootDir;
        uint32 FirstClusterCurrentDir;
        uint32 FreeClusterCount;
        uint32 NextFreeCluster;
        uint8 type;
};
```

3.2 设备与驱动管理模块设计

设备与驱动管理模块主要包括逻辑设备管理(efs.c)、物理设备管理(disc.c)、高速缓存管理(cache.c)和设备驱动管理(efsdriver.c)等文件。

逻辑设备管理模块位于设备与驱动管理模块的最上层，通过调用下层函数实现 EFSInit(文件系统初始化)。物理设备管理模块主要包括 2 个函数：DiscLoadMBR(加载主引导记录)和 DiscInit(初始化逻辑盘)。

高速缓存管理是文件系统针对块存储设备(如硬盘，Flash 等)的可选组件，在目标计算机物理内存足够大的情况下，启用高速缓存管理将极大提高应用程序对存储设备的访问速度，但会以内存开销为代价。在内存较小的嵌入式系统中，需要根据实际目标板来配置需要使用的 Cache 的大小(在 fsconfig.h 中配置)。

设备驱动管理层的作用是隐藏不同存储设备之间的差异和实现细节，为上层提供统一的驱动入口。在这种统一的框架下，通过调用的函数 InterfaceInit(底层接口驱动初始化)可以方便地在运行时动态增加或卸载设备驱动程序。

4 EFS 移植实例

下文通过一个实例来说明如何移植 EFS。目标板的 MCU 选用 Philips 的 LPC2148，它是一款基于 ARM7TDMI-S 内核的微控制器，该芯片是 LPC2000 系列的新品，提供了增强型 GPIO、1 个同步串行接口(Serial Peripheral Interface, SPI)和 1 个 SSP 接口，存储介质选用 SanDisk 公司 512 MB 的 SD 卡。

4.1 连接硬件电路

目标板上LPC2148 与SD卡的硬件连接如图 2 所示。LPC2148 通过SPI总线访问SD卡，LPC2148 在通信过程中充当SPI总线上的主机[3]。

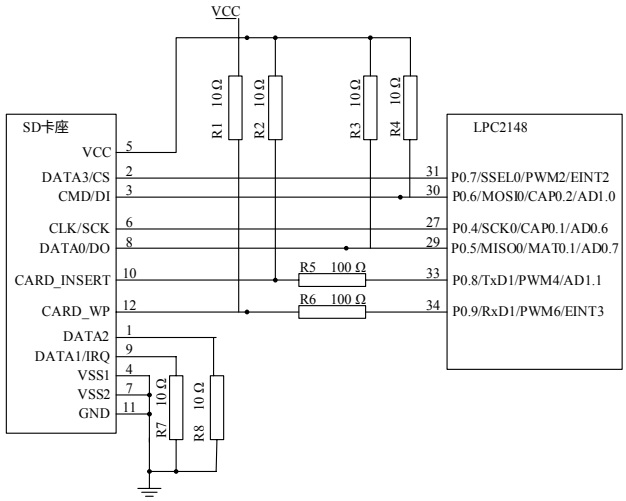

图 2 LPC2148 与 SD 卡的硬件连接

4.2 软件设计

4.2.1 编写底层驱动

在目标板上移植EFS之前，需要针对SD存储卡和SPI接口来编写相应底层驱动函数。本例中用到的驱动函数都在SD.c 文件中，文件包括SDInit(SD卡初始化函数)、GetSD Drive(获得SD卡驱动函数)和SDCommand(SD命令解析函数)等。其中，提供 InterfaceInit 函数直接调用的函数有 SDInit 和 GetSDDrive。SDInit函数的功能是初始化SPI、配置LPC2148 为SPI总线的主机、设置SPI时钟等。GetSDDrive函数的作用是获得逻辑盘的容量，将驱动程序地址传递给逻辑盘的信息管理程序等。SDCommand函数主要完成对SD卡操作命令的解析并执行相应操作[4]。

4.2.2 功能配置

EFS 所有可选功能配置都在 fsconfig.h 头文件中设置，对不同目标板的配置要求不同，本例中的主要功能配置如下：

```
(1)#define HW_LPC2000_SPI 0
//配置 LPC2148 SPI0 与 SD 卡连接
(2)#define CACHE_NUM_BUFFER 6
//使用 3KB 的 RAM 供 Cache 使用
(3)#define LITTLE_ENDIAN 1
//处理器配置为小端模式
(4)#define DATE_TIME_SUPPORT 0
//不支持时间和日期功能
```

对上述配置说明如下：(1)为了说明 EFS 的通用性，本文选择 LPC2148 的 SPI0 口，使这个代码不加任何修改就可以在 LPC2000 系列的其他 MCU 上使用，笔者在 LPC2148 微控制器的 SPI1(SSP)口上也做了相应测试，SPI1 口对文件的操作速度高于 SPI0。(2)由于 SD 卡需要按块(每块 512 Byte)操作，因此本文定义 CACHE_NUM_BUFFER 为 6，共分配 3 KB(512×6 Byte)的 RAM 区供 Cache 使用。Cache 设置的合理与否对 EFS 效率的影响很大，考虑到系统的整体性能，通常需要通过试验来决定如何设置该参数。(3)由于 LPC2000 微控制器把 ARM7 配置为小端模式，本文选择小端模式，与目标板 MCU 一致。(4)由于嵌入式系统的资源有限，文件的时间日期在嵌入式系统中的意义不大，因此定义 DATE_TIME_SUPPORT 为 0 来裁剪掉一些不必要的函数，以减小代码量。

4.3 EFS 的实现与性能测试

完成硬件设计和连接、驱动程序的编写且移植成功后，就可以使用 EFS。基本操作步骤如下：初始化 EFS→打开文件→读文件(写文件)→关闭文件。读文件的测试代码如下：

```
    if
( EFSInit(&efs,0 )==0){                                  (1)
    if
( FileOpen( &file_r,&efs.Fs,"test.txt",'r')==0){         (2)
    while ( e=FileRead(&file_r,512,buf ))                (3)
    LEDtest( buf,e );                                    (4)
      }
    FileClose ( &file_r );                               (5)
    }
```

对上述代码说明如下：(1)如果 EFS 初始化成功，则进行后继操作。(2)以只读方式打开文件 test.txt，如果成功则进行后继操作。(3)从打开的文件中读取 512 Byte，并将其发送到首地址，由 buf 指向的缓冲区中。e 为实际读取的字节数，最后一次执行该函数时 $e \leq 512$，当 $e=0$ 时，test.txt 文件读取结束，后面的 LEDtest(读出数据测试函数)不再执行。(4)读出数据测试函数，可以观察读出数据是否正确。(5)关闭文件 test.txt。

笔者在该目标板上进行了速度和正确性测试。读 20 MB 大小的文件所需时间约为 80 s，写 20 MB 文件所需时间约为 100 s。经过长时间可靠性测试，没有出现读写错误。该 EFS 在 ADS 编译环境下，编译之后的代码为 16 KB，SD 卡的驱动程序编译后的代码量约为 1 KB。整个 EFS 结构紧凑、代码量较小，且可以根据应用需求做进一步裁剪，适用于多种嵌入式系统。

5 结束语

本文提出一种 EFS，给出了较详细的设计过程。该 EFS 具有以下优点：(1)采用模块化设计，结构清晰；(2)提供标准 API 函数，移植方便；(3)加入可选 Cache 模块，文件读写速度极大提高；(4)符合 FAT 标准，支持 FAT12/16/32 格式的文件，可以方便地移植到多种嵌入式系统中。目前，该 EFS 已成功用于大数据量管理的实时应用中。下一步的工作是扩充驱动程序库，使之支持更多外部存储设备。

参考文献

[1] 秦婷婷, 罗玉平. 一种应用于多媒体嵌入式系统的文件系统的设计与实现[J]. 电子技术, 2007, 2(1): 64-68.

[2] Microsoft 公司. Microsoft Extensible Firmware Initiative FAT32 File System Specification. Version 1.03[Z]. 2000.

[3] Philips 公司. Accessing SD/MMC Card Using SPI on LPC2000[Z]. 2007.

[4] SanDisk 公司. SanDisk Secure Digital Card Product Manual Version 2.2[Z]. 2004.

(上接第 248 页)

文献[4]指出提高速度必然会导致芯片面积大量增加，因此，主张使用串行乘法器。从表 1 可以看出，4 bit 串并混合乘法器的速度比全串行乘法器速度提高了 4 倍，占用的空间却增加不到 1 倍；8 bit 串并混合乘法器的速度比全串行乘法器的速度提高了 8 倍，占用的空间增加不到 2 倍，比 4 bit 串并混合乘法器的速度提高 2 倍，空间只增加了约 1/2；16 bit 串并混合乘法器的速度比 8 bit 串并混合乘法器的速度提高 2 倍，占用空间也将近是它的 2 倍。速度提高得越快，空间增加得也越快(表 1 中 8 bit 以后)。因此，8 bit 串并混合乘法器在速度和面积协调方面相对较好，即为本文采用的最终设计。

5 结束语

本文实现了几个并行度不同的混合算法，并对它们的面积和速度进行比较，找出相对合适的设计，但该设计仅针对典型的几个并行度值，今后的工作将针对其他值进行设计，以便更精确地找到使速度和面积协调较好的设计方案。

参考文献

[1] Wollinger T. Computer Architectures for Cryptosystems Based on Hyperelliptic Curves[D]. Worchester, New York, USA: Worchester Polytechnic Institute, 2001.

[2] Sakai Y, Sakurai K. On the Practical Performance of Hyperelliptic Curve Cryptosystems in Software Implementations[J]. IEICE Transaction Fundamentals, 2000, 83(4): 692-703.

[3] Francisco R. New Algorithms and Architectures for Arithmetic in GF(2(m)) Suitable for Elliptic Curve Cryptography[D]. Corvallis, OR, USA: Oregon State University, 2000.

[4] Clancy T. Analysis of FPGA-based Hyperelliptic Curve Cryptosystems[D]. Chicago, USA: University of Illinois, 2002.