UFFS介绍文档

##### 概述

UFFS是一个针对或非闪存存储器（NOR Flash）设计的执行文件系统，其目标是能在嵌入式SylixOS环境中运行、实现文件系统的各项功能。

##### 设计思路

a）储存格式

NOR Flash具有读数据快，可按字节寻址等特点，但写速度较慢，且写之前需要进行擦除操作，这使得在NOR Flash上更适合使用日志型结构的文件系统。

基于NOR Flash可以按字节写入数据的特点，在UFFS中，最小记录信息的单位——节点是不固定长度的。

b）设计方案

接口上我们选择使用注册虚表的方式和SylixOS进行对接，这样可以减少和操作系统的耦合，以便于脱机的调试工作。

实现上我们参考了JFFS2文件系统的储存方式，选择使用以节点为单位的存储方案设计。

##### 实现方案

a）物理储存格式：

在UFFS中，具有储存可识别数据单元的最小单位是节点。每个文件的存储节点按照建立的时间先后，顺序地存储在Flash上，并且在可利用的存储空间里严格地按线性增长。存储空间由若干擦写块组成，每个擦写块之间相互独立，存储节点不能超越擦写块的边界存放。

在一个可擦除块中，节点从头开始，按4字节对齐紧密排布。

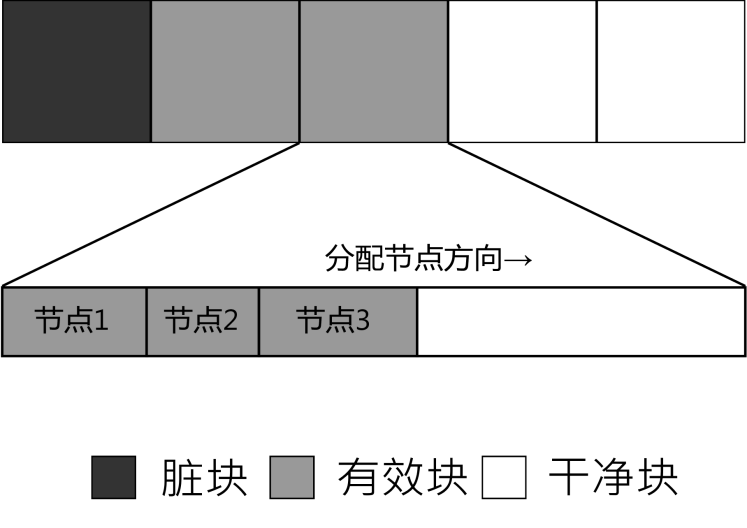


图 1闪存结构

节点有不同的类型，目前UFFS中两种主要的节点类型是INODE和DIRENT，其中INODE表示索引节点，它包括了一个对象的元数据和数据；DIRENT表示连接项，它记录了一个索引节点所建立的连接关系数据。

所有各类节点公有的头部信息如下，其中magic是用于表示UFFS节点的魔数，class是节点类型，tsize表示该节点的总长度，head CRC是这一头部的校验码，当校验码出错误时，视为无效的头部。

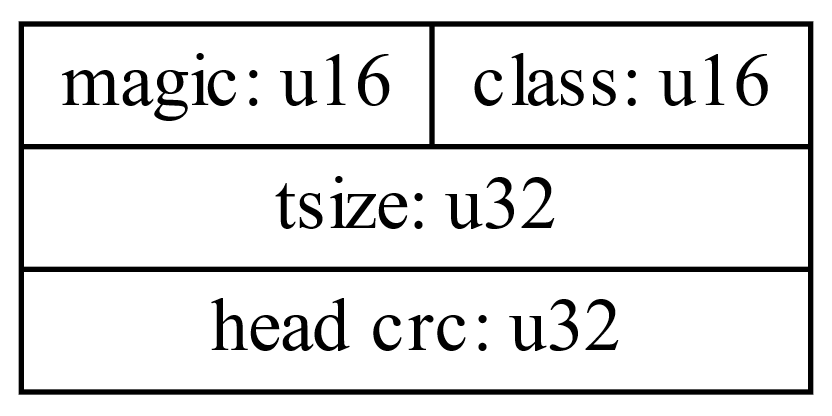


图 2所有各类节点共有的头部信息

INODE类型的节点，主要包括节点头和数据区，节点头储存了文件的元数据，而后面的数据区储存了一个文件真实储存信息的一个片段。

DIRENT类型的节点主要包括连接的信息，包括当前文件的ino和pino，其中pino指该节点连接到的父文件夹的ino，以及这个连接的名称。

由于闪存中可擦写块是最小的擦除单位，所以节点总是不会跨越一个块的边界。为了便于操作系统的读写过程，在UFFS中，一个INODE的数据部分不会超越一个页面的大小。

b）文件的逻辑结构：

在UFFS中，一个普通的文件对象由两级结构映射到实际的物理空间中，下图展示的是一个文件数据构成的实例：

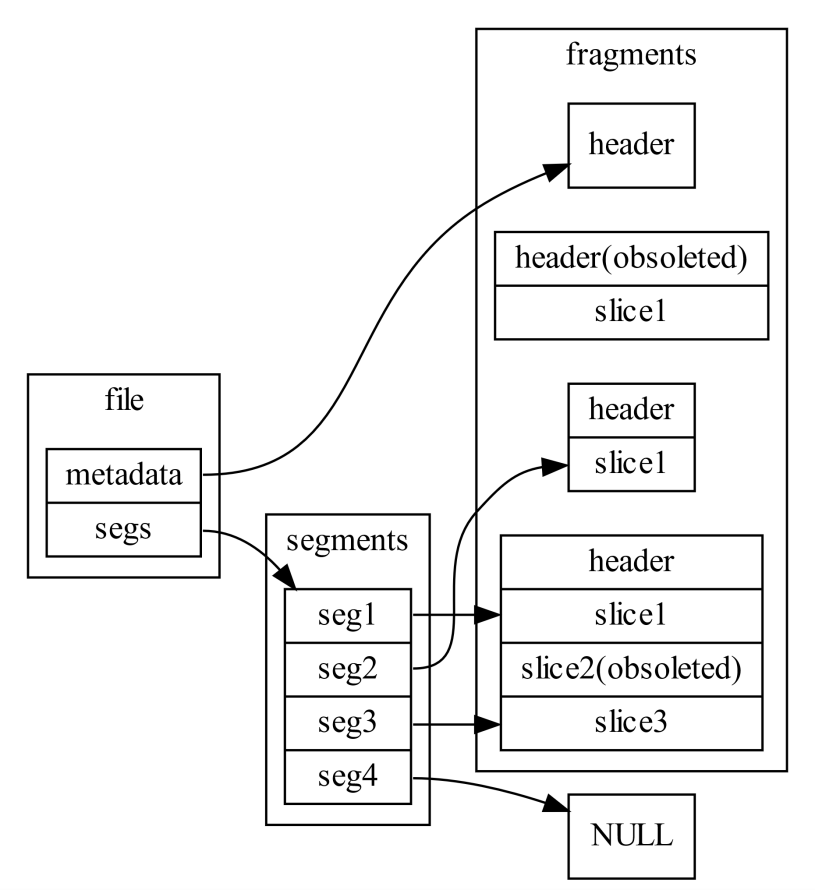


图 3文件层次

其中segment是文件中一段数据，fragment是一个闪存中储存的INODE节点中数据段的内容，它被分割为多个slice。一个segment可以指向一个slice，如果没有的话说明这里存在一个空洞，读出的二进制数据全部为0x00。

对于文件夹类型，一个文件夹类型的所有数据由整个闪存中所有连接到该节点的DIRENT节点构成。

c）文件系统的运行状态

文件系统在启动后首先会进行挂载操作，在这一操作时启动一个GC线程，在这之后这两部分开始一起进行工作。

在挂载操作中，UFFS首先会扫描所有可擦写块，检查节点的有效性，将具有相同ino的INODE节点连接起来，形成链表以便后续对文件的读取；将DIRENT节点构建成树状结构，剔除无效的连接。在检查和构建完毕后保留最小的恢复信息，以保证在读写节点时的效率同时减小文件系统的内存开销。

UFFS文件系统主要通过u\_dev和u\_file外部进行交互。其中u\_dev表示一个闪存设备句柄，用户可以通过这一句柄进行关于整个文件系统的操作；u\_file表示一个文件句柄，用户可以通过这一句柄进行关于一个文件或文件夹的操作，在UFFS中，多次打开同一个文件时产生的是一个共享的文件句柄。

在打开文件时，UFFS系统会利用节点保留的信息重新加载进一步检查文件的有效性并得到更详细的信息，这些信息会缓存在u\_file中。在对文件进行操作时，UFFS会利用u\_file中缓存的信息进行文件操作。

##### 问题与解决方案

**问题1**：

目前SylixOS中已存在的是YAFFS2文件系统，这个文件系统是基于NAND Flash设计的，这一文件系统不能让NOR Flash的特点得到充分发挥。

解决方案：对于SylixOS里的YAFFS文件系统，由于与我们的需设计的文件系统差异过大，我们可以参考其与翼辉的接口设计。但是在搭建整个文件系统的过程中，我们主要参照Linux中的JFFS2文件系统的操作，如它的文件存储结构、挂载过程、文件读写形式等等。

**问题2：**

直接使用Sylix OS提供的接口不方便对闪存进行仿真测试或脱机测试。

解决方案：为了在尚未完整实现文件系统时进行部分测试，我们使用了以下方案进行提供接口：由用户注册虚表，虚表中包括闪存系统的打开、关闭、读、写、擦除等操作，UFFS通过调用虚表方法进行对闪存驱动的调用。这样可以实现使用文件虚拟闪存的方法：将对闪存的操作替换成普通文件的操作，以方便在没有真实闪存的情况下进行调试。

**问题3：**

当擦除的时候产生了掉电的情况，这时一个可擦除块中就会产生大量连续的0xFF，但是整块的数据并没有被完全擦除。

解决方案：我们使用擦除标记来解决这个问题，在每次擦除完成之后在一个可擦除块的开头写入一个CLEAN\_MARKER节点，当挂载或检查的过程中读到CLEAN\_MARKER，同时读取它后面跟随着一定量连续的0xFF，则认为整个块已经被完全擦除。

##### 未来展望：

1. 由于挂载过程要扫描全盘的效率很低下，所以我们打算加入检查点来加速挂载过程。在可擦除块中加入CHECK\_POINT节点，在其中记录上这一块上部分的运行时信息，当挂载读取到这个信息时。可以通过扫描这个节点把运行时的信息记录在闪存上，利用的是以空间换时间的思路。
2. UFFS在未来可以加入闪存的地址映射的支持，让闪存直接映射到一部分地址空间中，便于读操作。
3. GC目前还未展开过多的工作。在擦写过程如何去延长闪存寿命，是研究的关键。