

WS63V100 文件系统

使用指南

文档版本 01

发布日期 2024-04-10

前言

概述

本文档主要针对 WS63V100 中 LITTLE FILE SYSTEM（下简称 LFS）文件系统模块的使用进行介绍。用于指导工程人员能够快速使用文件系统模块进行二次开发。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
WS63	V100

读者对象





本文档主要适用于以下工程师：

- 技术支持工程师
- 软件工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
----	----

符号	说明
 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2024-04-10	第一次正式版本发布。
00B01	2024-02-22	第一次临时版本发布。

目 录

前言i

1 LFS 简介1

2 LFS 编译预置.....2

3 LFS 接口 API.....3

3.1 API.....3

3.2 编程实例4

1 LFS 简介

LFS 文件系统是为小型嵌入式系统创建的一个文件系统，为用户提供了文件的打开、关闭、读取、写入、删除等功能，用户可以通过使用这些接口，将数据存储到 NOR FLASH。

- 掉电恢复能力- LFS 旨在处理随机掉电。所有文件操作都有强大的写时拷贝保证，如果断电，文件系统将回退到最后一个已知的良好状态。
- 动态磨损均衡- LFS 在设计时考虑到了 flash，并在动态块上提供了磨损均衡。此外，LFS 可以检测坏块并解决它们。
- 有界 RAM/ROM - LFS 设计用于使用少量内存。RAM 的使用是有严格限制的，这意味着 RAM 的使用不会随着文件系统的增长而改变。文件系统不包含无界递归，动态内存仅限于可以静态提供的可配置缓冲区。

2 LFS 编译预置

当前 SDK 中提供了 LFS 特性，但默认不编译打开，如需使用，需按以下指导进行编译及特性配置

- 步骤 1 加入编译。在需要 LFS 特性的编译目标中加入组件 'littlefs_adapt_ws63'
- 步骤 2 特性宏。使用 menuconfig，在需要 LFS 特性的编译目标中打开该特性对应的特性宏，特性宏配置路径：middleware->chip->Choose Chip (ws63)-> Chip Configurations for ws63->打开 littlefs adapt，即可打开 CONFIG_MIDDLEWARE_SUPPORT_LFS 特性宏，打开宏后，LFS 会在运行过程中自行完成挂载操作
- 步骤 3 FLASH 分配，关注代码（文件路径：middleware/chips/ws63/littlefs/littlefs_adapt.c）接口 little_adapt_get_block_info 读取分区表信息对应分区为 PARTITION_FOTA_DATA，该宏需要替换或修改为分区表 ID 文件 (middleware/chips/ws63/partition/include/partition_resource_id.h)中其他的分区 ID 宏，并在分区表配置文件 (build/config/target_config/ws63/param_sector/param_sector.json) 中对该宏 ID 对应的 FLASH 地址与大小进行配置，作为 LFS 的运行基础，需要注意的是，每次调整 LFS 的地址后，LFS 中的内容会丢失；
- 步骤 4 其他适配，为适配不同的上层 VFS，对打开文件传入的 oflag 参数进行了转化，使得能够适配 LFS 的下层实现；为保证正常使用，需要重新实现 fs_adapt_flag_format 接口来适配上层 flag。

----结束

3 LFS 接口 API

- 3.1 API
- 3.2 编程实例

3.1 API

当前提供的 API 不代表 LFS 的所有能力，请按需使用

表3-1

功能分类		说明
文件系统挂载/ 去挂载	fs_adapt_mount	挂载文件系统。
	fs_adapt_unmount	去挂载文件系统。
文件 I/O 操作	fs_adapt_open	打开一个文件，如果不存在，根据传入的 oflag 参数决定是否创建文件，该 oflag 是已进行转换后的参数，如果 path 参数中包含的路径不存在，则会默认创建该路径。
	fs_adapt_close	关闭一个文件句柄。
	fs_adapt_read	读文件操作，成功返回读取的字节数，失败返回-1 并设置错误码，如果在调用该接口前已到达文件末尾，则此次 read 返回 0。
	fs_adapt_write	成功返回写入的字节数，出错返回-1 并设置错误码。
	fs_adapt_delete	通过路径名删除一个文件。

功能分类		说明
设置读/写偏移	fs_adapt_seek	设置读/写文件偏移。 <ul style="list-style-type: none">LFS_SEEK_SET: 从文件头部开始偏移 offset 个字节。LFS_SEEK_CUR: 从文件当前读写的指针位置开始, 增加 offset 个字节的偏移量。LFS_SEEK_END: 文件偏移量设置为文件的大小加上偏移量字节, 偏移量 offset 只允许为负值
获取文件大小	fs_adapt_stat	通过文件名获取文件大小。
同步文件内容	fs_adapt_sync	同步内存和片外存储, 将缓冲数据写入到片外存储。
创建路径	fs_adapt_mkdir	创建路径。

3.2 编程实例

代码实现中提供一段基础调用示例 lfs_test, 打开根目录下名为 lfs_test 的文件, 读取首个字节并按照整型数打印, 字节加一后再写入到文件头, 最后关闭文件;

```
void lfs_test(void)
{
    // read current count
    char boot_count = 0;
    int fp = fs_adapt_open("/lfs_test", LFS_O_RDWR | LFS_O_CREAT);
    if (fp < 0) {
        return;
    }
    int ret = fs_adapt_read(fp, &boot_count, sizeof(boot_count));
    lfs_debug_print_info("lfs_test read, ret = 0x%x\r\n", ret);
    // print the boot count
    lfs_debug_print_info("====boot_count: %d====\r\n", boot_count);
    // update boot count
    boot_count = (char)((uint8_t)boot_count + 1);
    ret = fs_adapt_seek(fp, 0, LFS_SEEK_SET);
```



```
lfs_debug_print_info("lfs_test seek, ret = 0x%x\r\n", ret);
if (ret < LFS_ERR_OK) {
    return;
}
ret = fs_adapt_write(fp, &boot_count, sizeof(boot_count));
lfs_debug_print_info("lfs_test write, ret = 0x%x\r\n", ret);
// remember the storage is not updated until the file is closed successfully
ret = fs_adapt_close(fp);
lfs_debug_print_info("lfs_test close, ret = 0x%x\r\n", ret);
if (ret < LFS_ERR_OK) {
    return;
}
}
```