|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 深圳市同洲电子股份有限公司 | | 文 档 编 号 | 版本号 | 密级 |
|  | V0.1 | 内部公开 |
| 文档名称 |  | | 日期 | 2014-09-9 |

Android系统MhddFS移植

|  |  |  |  |
| --- | --- | --- | --- |
| **文档作者：** | **王洋** | **日期：** | **2014-9-09** |
| **审 核：** |  | **日期：** |  |
| **批 准：** |  | **日期：** |  |



**深圳市同洲电子股份有限公司 版权所有**

**内部资料 注意保密**

# Android系统MhddFS移植

### 背景

在Android系统中安装大型游戏时，存在存储空间不足问题，希望通过外部存储卡或者U盘实现空间扩容。

### 目的

研究使用MhddFS来实现空间扩容的方法。

#### 工具介绍

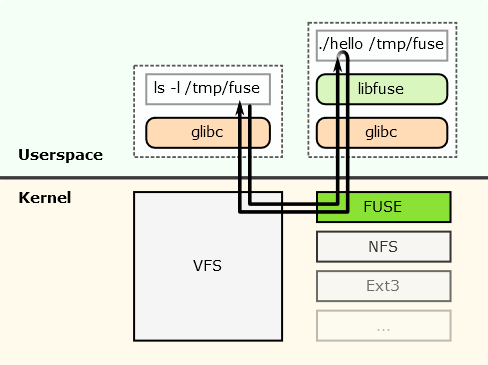
The driver combines a several mount points into the single one.

（通过fuse接口，在应用层实现多个目录合并到一个目录的功能）。其中涉及两个第三方库：libfuse和uthash。

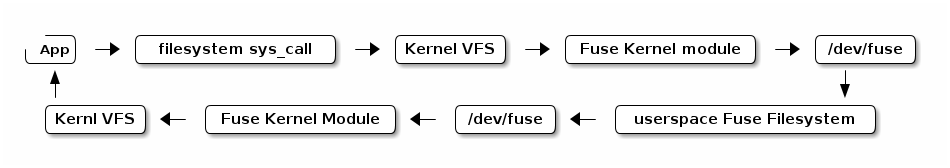
Uthash： 通过C的宏定义实现Hash功能，向外基本功能的API接口。

Libfuse：用户空间的fuse库，发送到用户空间的请求会先被libfuse进行接收和解析，翻译成具体的操作，然后调用hook的实现函数，完成请求后再进行FUSE协议通信的方式返回给内核的FUSE模块。所以libfuse充当了用户空间的FUSE协议的另一端。负责和内核空间的FUSE模块进行通信。

Fuse的实现原理：



一次用户使用Fuse类型文件系统的过程如下：



### 参考网址

<https://romanrm.net/mhddfs>

### 过程

下载源码：

# mhddfs

svn co http://svn.uvw.ru/mhddfs/trunk mhddfs

# libfuse

git clone ssh://10.10.5.66:29418/cm/CyanogenMod/android\_external\_fuse.git

# uthash

git clone <https://github.com/troydhanson/uthash.git>

创建Android.mk

LOCAL\_PATH:= $(call my-dir)

include $(CLEAR\_VARS)

LOCAL\_SRC\_FILES := \

src/main.c \

src/debug.c \

src/flist.c \

src/parse\_options.c \

src/tools.c \

src/usage.c

LOCAL\_CFLAGS := -g -O0 -Wall -DFUSE\_USE\_VERSION=26 -MMD -DWITHOUT\_XATTR -D\_FILE\_OFFSET\_BITS=64

LOCAL\_STATIC\_LIBRARIES := libfuse

LOCAL\_SHARED\_LIBRARIES := libc libcutils liblog libselinux libdl

LOCAL\_C\_INCLUDES := \

external/fuse/include \

external/fuse/android \

$(LOCAL\_PATH)/src

LOCAL\_MODULE := mhddfs

include $(BUILD\_EXECUTABLE)

解决编译错误：

diff --git a/src/usage.c b/src/usage.c

index 33b2dba..024a2f1 100644

--- a/src/usage.c

+++ b/src/usage.c

@@ -43,7 +43,7 @@ void usage(FILE \* to)

"\n"

" see fusermount(1) for information about other options\n"

"";

- fprintf(to, usage);

+ fprintf(to, "%s", usage);

if (to==stdout) exit(0);

exit(-1);

}

#### 使用说明

命令行参数使用说明：

Multi-hdd FUSE filesystem

Copyright (C) 2008, Dmitry E. Oboukhov <dimka@avanto.org>

Usage:

mhddfs dir1,dir2.. mountpoint [ -o OPTIONS ]

OPTIONS:

mlimit=xxx - limit of the disk free space (if the disk

has the free space more than specified - it is

considered as the empty one). Default is 4Gb,

but 100Mb at least.

logfile=/path/to/file - path to a file where the logs

will be stored.

loglevel=x - level for log-messages:

0 - debug

1 - info

2 - default messages

see fusermount(1) for information about other options }

命令行参数的使用：

mhddfs /storage/external\_storage/sdcard1,/cache /mnt/shell/emulated \

-o allow\_other \

-o nonempty \

-o mlimit=100M \

-o logfile=/tmp/mhddfs.log \

-o loglevel=0

1. allow\_other必须设置，允许其它用户可访问。
2. mlimit限制最低磁盘空间大小，单位可以是M，G， 看介绍至少要100M。
3. 程序限制了输入源的目录至少为2个，在系统启动后如果没有sdcard或者u盘，可挂载的目录只有一个，因此将该参数限制关闭掉。
4. nonempty：fuse: if you are sure this is safe, use the 'nonempty' mount option，不清楚该参数的具体作用，如果只是想确认挂载的目录是否一定存在的，代码里已经做了检测，没有则会创建。

如何卸载：

umount /mnt/shell/emulated

查看挂载点是被进程占用：

fuser -m /storage/external\_storage/mhddfs

kill掉占用的进程：

fuser -k /storage/external\_storage/mhddfs

在vold中封装了一个管理mhddfs的类:

class MhddfsManager

{

public:

MhddfsManager();

~MhddfsManager();

enum Action { ADD\_BRANCH, DEL\_BRANCH };

#ifdef DEBUG\_MHDDFS

void logArgs(std::vector<const char\*>& args);

#else

void logArgs(std::vector<const char\*>& args) {};

#endif

int doAddDelBranch(Action act, const char\* branch);

int doMount();

int unmount();

int ctlSdcardServer(bool enable);

bool isMounted();

private:

const static std::string mMountPoint;

static std::vector<std::string> mBranches;

MhddfsManager(const MhddfsManager&);

void operator=(const MhddfsManager&);

};

其中如果增加或者删除一个分支，都必须按该顺序:

StopSdcard -> FuserKill -> umount -> mount -> StartSdcard 流程。一级级关联，先入后出原则。

### 问题

在使用拇指玩应用测试安装游戏时，偶尔会出现mhddfs崩溃的问题：

D/SystemInfoUtil( 1625): The surplus space in SD card is 1986359296

I/SystemInfoUtil( 1625): totalCount===5944991744++++++remaidCount===730771456

I/AppDownloadTask( 1625): flag===3

I/AppDownloadTask( 1625): url======http://appstatic.coship.com:80/upload/signedapp/ed36471e-5962-4e8a-b35c-80f45df57f44.apk

F/libc ( 694): Fatal signal 11 (SIGSEGV) at 0x00000000 (code=1), thread 2674 (mhddfs)

I/DEBUG ( 121): \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\* \*\*\*

I/DEBUG ( 121): AM write failure (32 / Broken pipe)

I/DEBUG ( 121): Build fingerprint: 'MBX/k200/k200:4.4.2/KOT49H/20140814:user/test-keys'

I/DEBUG ( 121): Revision: '10'

I/DEBUG ( 121): pid: 694, tid: 2674, name: mhddfs >>> /system/bin/mhddfs <<<

I/DEBUG ( 121): signal 11 (SIGSEGV), code 1 (SEGV\_MAPERR), fault addr 00000000

I/DEBUG ( 121): r0 00000000 r1 00000000 r2 00000000 r3 b8f104c8

I/DEBUG ( 121): r4 b8f10c88 r5 b8f10cf8 r6 00000000 r7 c024e7b1

I/DEBUG ( 121): r8 a0d954a8 r9 00000006 sl 006a6b00 fp 00000034

I/DEBUG ( 121): ip 00000001 sp b6b26af8 lr b6f3ecc5 pc b6fa65e2 cpsr 60070030

I/DEBUG ( 121): d0 0000000000000000 d1 0000000000000000

I/DEBUG ( 121): d2 0000000000000000 d3 0000000000000000

I/DEBUG ( 121): d4 0000000000000000 d5 6361632f65726f74

I/DEBUG ( 121): d6 00000001752f6568 d7 0000000100000000

I/DEBUG ( 121): d8 0000000000000000 d9 0000000000000000

I/DEBUG ( 121): d10 0000000000000000 d11 0000000000000000

I/DEBUG ( 121): d12 0000000000000000 d13 0000000000000000

I/DEBUG ( 121): d14 0000000000000000 d15 0000000000000000

I/DEBUG ( 121): d16 41efffffffe00000 d17 3f50624dd2f1a9fc

I/DEBUG ( 121): d18 41c6c71f94000000 d19 0000000000000000

I/DEBUG ( 121): d20 0000000000000000 d21 0000000000000000

I/DEBUG ( 121): d22 0000000000000000 d23 0000000000000000

I/DEBUG ( 121): d24 0000000000000000 d25 0000000000000000

I/DEBUG ( 121): d26 0000000000000000 d27 0000000000000000

I/DEBUG ( 121): d28 0000000000000000 d29 0000000000000000

I/DEBUG ( 121): d30 0000000000000000 d31 0000000000000000

I/DEBUG ( 121): scr 80000010

W/NativeCrashListener( 513): Couldn't find ProcessRecord for pid 694

I/DEBUG ( 121):

I/DEBUG ( 121): backtrace:

I/DEBUG ( 121): #00 pc 000035e2 /system/bin/mhddfs

I/DEBUG ( 121): #01 pc 00007faf /system/bin/mhddfs

I/DEBUG ( 121): #02 pc 0000807d /system/bin/mhddfs

I/DEBUG ( 121): #03 pc 0000b2e1 /system/bin/mhddfs

I/DEBUG ( 121): #04 pc 0000bec7 /system/bin/mhddfs

I/DEBUG ( 121): #05 pc 0000cd33 /system/bin/mhddfs

I/DEBUG ( 121): #06 pc 0000a607 /system/bin/mhddfs

I/DEBUG ( 121): #07 pc 0000d228 /system/lib/libc.so (\_\_thread\_entry+72)

I/DEBUG ( 121): #08 pc 0000d3c0 /system/lib/libc.so (pthread\_create+240)

I/DEBUG ( 121): #09 pc 000003b4 [heap]

通过valgrind工具测试，mhddfs存在内存越界和内存泄漏的问题：内存越界问题是程序本身的问题，访问使用已经释放的内存地址（悬浮指针），其结果可能是正确的，也有可能会导致崩溃。

修正如下：

HASH\_ITER(hh, items\_ht, item, tmp) {

free(item->name);

item->name = NULL;

free(item->st);

item->st = NULL;

// 当数据链表为最后一个时，清除HASH表空间

if (item->hh.next == NULL) {

uthash\_free(item->hh.tbl->buckets,

item->hh.tbl->num\_buckets\*sizeof(struct UT\_hash\_bucket));

HASH\_BLOOM\_FREE(item->hh.tbl);

uthash\_free(item->hh.tbl, sizeof(UT\_hash\_table));

}

free(item);

item = NULL;

}

内存泄露修正如下：

void mhdd\_freeconfig()

{

int i = 0;

if (mhdd.mount)

free(mhdd.mount);

if (mhdd.debug)

fclose(mhdd.debug);

if (mhdd.debug\_file)

free(mhdd.debug\_file);

if (mhdd.mlimit\_str)

free(mhdd.mlimit\_str);

if (mhdd.dirs) {

for (i = 0; mhdd.dirs[i]; i++)

free(mhdd.dirs[i]);

free(mhdd.dirs);

}

}

}

### 成果

通过Vold进程管理扩展存储设备的挂载功能，在增加或者移除时都会激活mhddfs重新挂载现有分区到指定挂载点。