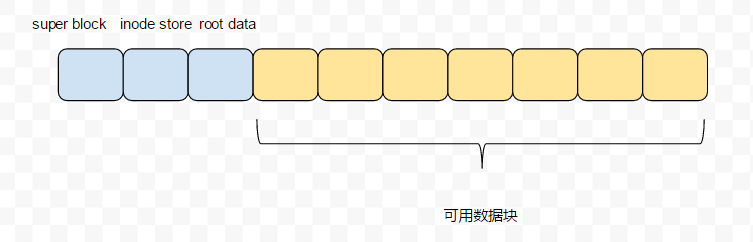
Simplefs unlink实现和缓存设计概要

By Ackerman

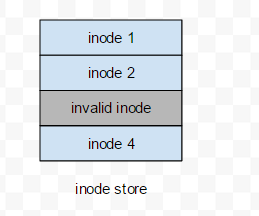
# Simplefs layout

下面是simplefs的layout，其中第一个块是super block。第二个块是inode store，等价于ext2的inode table。第3个块保存的是根目录的数据。asf

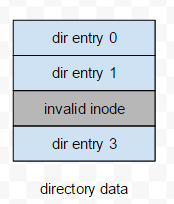


# Simplefs的缺陷

1. Simplefs认为在inode store中所有有效的inode都是连续存放的，所以在遍历的时候根据文件个数遍历inode store中的inode。而将文件删除会造成可用inode的不连续，simplefs无法处理这种情况。如下图



1. 和inode的保存方式一样，simplefs也认为有效目录项是连续存放的，如果一个文件删除，simplefs无法将这个空闲部分继续分配给下个文件。



# 解决方案

在不改变磁盘layout的情况下，可以采用缓存的方式解决上述两个问题：

1. 对于inode，最经典的方式莫过于inode bitmap了，所以在super block的私有数据中保存一份这样的bitmap可以有效利用inode store的空间，也可以快速查找到已经使用的inode。挂载的时候填充这个bitmap，后续查找inode可以通过bitmap精确定位。

相关数据结构：

struct simplefs\_sb\_info {

struct simplefs\_super\_block \*sb;

unsigned long imap;

struct buffer\_head \*bh;

};

1. 对于目录项，采用两个链表分别管理已经使用的目录项和空闲的目录项。将缓存的结构挂接到dentry的私有指针。每次lookup的时候，为扫描一个目录的数据块，将目录项的缓存加到dentry的cache中。

每一个dentry的一个cache:

struct simplefs\_dir\_cache {

uint64\_t dir\_children\_count;

struct list\_head used;

struct list\_head free;

};

每一个目录项的缓存中记录目录项的位置（第几个目录项）

struct simplefs\_cache\_entry {

struct simplefs\_dir\_record record;

struct list\_head list;

int entry\_no;

};

# 相关函数行为的改变

struct dentry \*ovl\_lookup(struct inode \*dir, struct dentry \*dentry,

unsigned int flags)

lookup函数负责建立缓存，一个目录第一次被访问的时候就会扫描所有的目录项，将已经使用的目录项放到used链表中，空闲的目录项放到free链表中。

static int simplefs\_create\_fs\_object(struct inode \*dir, struct dentry \*dentry,

umode\_t mode)

函数在创建的时候需要从其目录的缓存中取得一个可用的目录项缓存，从缓存中查找创建文件时候需要写入的目录项位置。同时，将目录项缓存从目录的free链表中移动到used链表中。

static int simplefs\_unlink(struct inode \*dir, struct dentry \*dentry)

删除文件对缓存的操作截然相反，将一个目录项的缓存从used移动到free链表上。

static int simplefs\_iterate(struct file \*filp, struct dir\_context \*ctx)

有了缓存，就不需要从磁盘上依次读取目录项了，直接将used链表中的目录项返回即可。

# Patch

From 1f2274dde77626a2ede45ad1034e7f81a929b031 Mon Sep 17 00:00:00 2001

From: Rock Li <rli@sierrawireless.com>

Date: Tue, 24 Apr 2018 10:45:43 +0800

Subject: [PATCH] add unlink API and cache layer.

From 02d645bcea362bc775b54c12da152a940992b1bb Mon Sep 17 00:00:00 2001

From: Ackerman <rockdotlee@gmail.com >

Date: Tue, 24 Apr 2018 10:45:43 +0800

Subject: [PATCH] add unlink API and cache layer.

Signed-off-by: Ackerman <rockdotlee@gmail.com>

---

simple.c | 508 ++++++++++++++++++++++++++++++++++++++++++-------------

simple.h | 9 +-

super.h | 2 +-

3 files changed, 404 insertions(+), 115 deletions(-)

mode change 100644 => 100755 simple.c

diff --git a/simple.c b/simple.c

old mode 100644

new mode 100755

index 54a5628..95c8a35

--- a/simple.c

+++ b/simple.c

@@ -16,7 +16,6 @@

#include <linux/version.h>

#include "super.h"

-

#define f\_dentry f\_path.dentry

/\* A super block lock that must be used for any critical section operation on the sb,

\* such as: updating the free\_blocks, inodes\_count etc. \*/

@@ -33,11 +32,110 @@ static DEFINE\_MUTEX(simplefs\_inodes\_mgmt\_lock);

static DEFINE\_MUTEX(simplefs\_directory\_children\_update\_lock);

static struct kmem\_cache \*sfs\_inode\_cachep;

+static struct kmem\_cache \*sfs\_entry\_cachep;

+

+struct simplefs\_cache\_entry {

+   struct simplefs\_dir\_record record;

+   struct list\_head list;

+   int entry\_no;

+};

+

+struct simplefs\_dir\_cache {

+   uint64\_t dir\_children\_count;

+   struct list\_head used;

+   struct list\_head free;

+};

+

+

+static struct simplefs\_dir\_cache \*simplefs\_cache\_alloc(void)

+{

+ struct simplefs\_dir\_cache \*dir\_cache;

+ dir\_cache = kzalloc(sizeof(struct simplefs\_dir\_cache), GFP\_KERNEL);

+ if (!dir\_cache)

+ ERR\_PTR(-ENOMEM);

+

+ INIT\_LIST\_HEAD(&dir\_cache->free);

+ INIT\_LIST\_HEAD(&dir\_cache->used);

+

+ return dir\_cache;

+}

+

+static int dir\_cache\_build(struct simplefs\_dir\_cache \*dir\_cache, struct buffer\_head \*bh)

+{

+   struct simplefs\_dir\_record \*record;

+   struct simplefs\_cache\_entry \*cache\_entry;

+   int i;

+

+   record = (struct simplefs\_dir\_record \*)bh->b\_data;

+   for (i = 0; i < SIMPLEFS\_MAX\_CHILDREN\_CNT; i++, record++) {

+       cache\_entry = kmem\_cache\_alloc(sfs\_entry\_cachep, GFP\_KERNEL);

+       if (!cache\_entry)

+           return -ENOMEM;

+

+       cache\_entry->entry\_no = i;

+

+       if (record->inode\_no != 0) {

+           memcpy(&cache\_entry->record, record, sizeof(struct simplefs\_dir\_record));

+           list\_add\_tail(&cache\_entry->list, &dir\_cache->used);

+       } else {

+           list\_add\_tail(&cache\_entry->list, &dir\_cache->free);

+       }

+   }

+

+   return 0;

+}

+

+#ifdef SIMPLEFS\_DEBUG

+static void travers\_dir\_cache(struct simplefs\_dir\_cache \*dir\_cache)

+{

+   struct simplefs\_cache\_entry \*cache\_entry;

+   pr\_info("print used record:\n");

+   list\_for\_each\_entry(cache\_entry, &dir\_cache->used, list)

+       pr\_info("record name %s, ino %lld, entry\_no %d",

+           cache\_entry->record.filename, cache\_entry->record.inode\_no, cache\_entry->entry\_no);

+

+   pr\_info("print free record:\n");

+   list\_for\_each\_entry(cache\_entry, &dir\_cache->free, list)

+       pr\_info("record entry\_no %d", cache\_entry->entry\_no);

+}

+#endif

+

+static struct simplefs\_cache\_entry \*used\_cache\_entry\_get(struct simplefs\_dir\_cache \*dir\_cache,

+                                       struct dentry \*dentry)

+{

+   struct simplefs\_cache\_entry \*cache\_entry;

+

+   list\_for\_each\_entry(cache\_entry, &dir\_cache->used, list) {

+       if (!strcmp(cache\_entry->record.filename, dentry->d\_name.name)) {

+           return cache\_entry;

+       }

+   }

+

+   return NULL;

+}

+

+static void cache\_entry\_insert(struct list\_head \*head, struct simplefs\_cache\_entry \*cache\_entry)

+{

+   struct simplefs\_cache\_entry \*tmp\_entry;

+   list\_del(&cache\_entry->list);

+

+   list\_for\_each\_entry(tmp\_entry, head, list) {

+       if (cache\_entry->entry\_no < tmp\_entry->entry\_no)

+           break;

+   }

+

+   list\_add\_tail(&cache\_entry->list, &tmp\_entry->list);

+}

+

+static struct simplefs\_cache\_entry \*free\_cache\_entry\_get(struct simplefs\_dir\_cache \*dir\_cache)

+{

+   return list\_first\_entry(&dir\_cache->free, struct simplefs\_cache\_entry, list);

+}

void simplefs\_sb\_sync(struct super\_block \*vsb)

{

   struct buffer\_head \*bh;

-   struct simplefs\_super\_block \*sb = SIMPLEFS\_SB(vsb);

+   struct simplefs\_super\_block \*sb = SIMPLEFS\_SB(vsb)->sb;

   bh = sb\_bread(vsb, SIMPLEFS\_SUPERBLOCK\_BLOCK\_NUMBER);

   BUG\_ON(!bh);

@@ -53,8 +151,8 @@ struct simplefs\_inode \*simplefs\_inode\_search(struct super\_block \*sb,

       struct simplefs\_inode \*search)

{

   uint64\_t count = 0;

-   while (start->inode\_no != search->inode\_no

-           && count < SIMPLEFS\_SB(sb)->inodes\_count) {

+   int icount = SIMPLEFS\_DEFAULT\_BLOCK\_SIZE / sizeof(struct simplefs\_inode);

+   while (start->inode\_no != search->inode\_no && count < icount) {

       count++;

       start++;

   }

@@ -68,7 +166,7 @@ struct simplefs\_inode \*simplefs\_inode\_search(struct super\_block \*sb,

void simplefs\_inode\_add(struct super\_block \*vsb, struct simplefs\_inode \*inode)

{

-   struct simplefs\_super\_block \*sb = SIMPLEFS\_SB(vsb);

+   struct simplefs\_sb\_info \*sb\_info = SIMPLEFS\_SB(vsb);

   struct buffer\_head \*bh;

   struct simplefs\_inode \*inode\_iterator;

@@ -88,10 +186,10 @@ void simplefs\_inode\_add(struct super\_block \*vsb, struct simplefs\_inode \*inode)

   }

   /\* Append the new inode in the end in the inode store \*/

-   inode\_iterator += sb->inodes\_count;

-

+   inode\_iterator += inode->inode\_no;

   memcpy(inode\_iterator, inode, sizeof(struct simplefs\_inode));

-   sb->inodes\_count++;

+   sb\_info->sb->inodes\_count++;

+   set\_bit(inode->inode\_no, &sb\_info->imap);

   mark\_buffer\_dirty(bh);

   simplefs\_sb\_sync(vsb);

@@ -111,7 +209,7 @@ void simplefs\_inode\_add(struct super\_block \*vsb, struct simplefs\_inode \*inode)

\* will still be marked as non-free. You need fsck to fix this.\*/

int simplefs\_sb\_get\_a\_freeblock(struct super\_block \*vsb, uint64\_t \* out)

{

-   struct simplefs\_super\_block \*sb = SIMPLEFS\_SB(vsb);

+   struct simplefs\_super\_block \*sb = SIMPLEFS\_SB(vsb)->sb;

   int i;

   int ret = 0;

@@ -150,7 +248,7 @@ end:

static int simplefs\_sb\_get\_objects\_count(struct super\_block \*vsb,

                    uint64\_t \* out)

{

-   struct simplefs\_super\_block \*sb = SIMPLEFS\_SB(vsb);

+   struct simplefs\_super\_block \*sb = SIMPLEFS\_SB(vsb)->sb;

   if (mutex\_lock\_interruptible(&simplefs\_inodes\_mgmt\_lock)) {

       sfs\_trace("Failed to acquire mutex lock\n");

@@ -162,35 +260,20 @@ static int simplefs\_sb\_get\_objects\_count(struct super\_block \*vsb,

   return 0;

}

-#if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(3, 11, 0)

static int simplefs\_iterate(struct file \*filp, struct dir\_context \*ctx)

-#else

-static int simplefs\_readdir(struct file \*filp, void \*dirent, filldir\_t filldir)

-#endif

{

   loff\_t pos;

   struct inode \*inode;

   struct super\_block \*sb;

-   struct buffer\_head \*bh;

   struct simplefs\_inode \*sfs\_inode;

-   struct simplefs\_dir\_record \*record;

-   int i;

+   struct dentry \*dentry = filp->f\_path.dentry;

+   struct simplefs\_dir\_cache \*dir\_cache = dentry->d\_fsdata;

+   struct simplefs\_cache\_entry \*cache\_entry;

-#if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(3, 11, 0)

   pos = ctx->pos;

-#else

-   pos = filp->f\_pos;

-#endif

   inode = filp->f\_dentry->d\_inode;

   sb = inode->i\_sb;

-   if (pos) {

-       /\* FIXME: We use a hack of reading pos to figure if we have filled in all data.

-        \* We should probably fix this to work in a cursor based model and

-        \* use the tokens correctly to not fill too many data in each cursor based call \*/

-       return 0;

-   }

-

   sfs\_inode = SIMPLEFS\_INODE(inode);

   if (unlikely(!S\_ISDIR(sfs\_inode->mode))) {

@@ -201,24 +284,19 @@ static int simplefs\_readdir(struct file \*filp, void \*dirent, filldir\_t filldir)

       return -ENOTDIR;

   }

-   bh = sb\_bread(sb, sfs\_inode->data\_block\_number);

-   BUG\_ON(!bh);

+   if (pos) {

+       /\* FIXME: We use a hack of reading pos to figure if we have filled in all data.

+        \* We should probably fix this to work in a cursor based model and

+        \* use the tokens correctly to not fill too many data in each cursor based call \*/

+       return 0;

+   }

-   record = (struct simplefs\_dir\_record \*)bh->b\_data;

-   for (i = 0; i < sfs\_inode->dir\_children\_count; i++) {

-#if LINUX\_VERSION\_CODE >= KERNEL\_VERSION(3, 11, 0)

-       dir\_emit(ctx, record->filename, SIMPLEFS\_FILENAME\_MAXLEN,

-           record->inode\_no, DT\_UNKNOWN);

+   list\_for\_each\_entry(cache\_entry, &dir\_cache->used, list) {

+       dir\_emit(ctx, cache\_entry->record.filename, SIMPLEFS\_FILENAME\_MAXLEN,

+           cache\_entry->record.inode\_no, DT\_UNKNOWN);

       ctx->pos += sizeof(struct simplefs\_dir\_record);

-#else

-       filldir(dirent, record->filename, SIMPLEFS\_FILENAME\_MAXLEN, pos,

-           record->inode\_no, DT\_UNKNOWN);

-       filp->f\_pos += sizeof(struct simplefs\_dir\_record);

-#endif

       pos += sizeof(struct simplefs\_dir\_record);

-       record++;

   }

-   brelse(bh);

   return 0;

}

@@ -228,12 +306,11 @@ static int simplefs\_readdir(struct file \*filp, void \*dirent, filldir\_t filldir)

struct simplefs\_inode \*simplefs\_get\_inode(struct super\_block \*sb,

                    uint64\_t inode\_no)

{

-   struct simplefs\_super\_block \*sfs\_sb = SIMPLEFS\_SB(sb);

+   struct simplefs\_sb\_info \*sfs\_sb\_info = SIMPLEFS\_SB(sb);

   struct simplefs\_inode \*sfs\_inode = NULL;

   struct simplefs\_inode \*inode\_buffer = NULL;

-

-   int i;

   struct buffer\_head \*bh;

+   int found = 0;

   /\* The inode store can be read once and kept in memory permanently while mounting.

    \* But such a model will not be scalable in a filesystem with

@@ -243,23 +320,15 @@ struct simplefs\_inode \*simplefs\_get\_inode(struct super\_block \*sb,

   sfs\_inode = (struct simplefs\_inode \*)bh->b\_data;

-#if 0

-   if (mutex\_lock\_interruptible(&simplefs\_inodes\_mgmt\_lock)) {

-       printk(KERN\_ERR "Failed to acquire mutex lock %s +%d\n",

-        \_\_FILE\_\_, \_\_LINE\_\_);

+   /\*no 1 is the bit 0 in bitmap\*/

+   found = (sfs\_sb\_info->imap >> (inode\_no - 1)) & 1;

+   if (!found)

       return NULL;

-   }

-#endif

-   for (i = 0; i < sfs\_sb->inodes\_count; i++) {

-       if (sfs\_inode->inode\_no == inode\_no) {

-           inode\_buffer = kmem\_cache\_alloc(sfs\_inode\_cachep, GFP\_KERNEL);

-           memcpy(inode\_buffer, sfs\_inode, sizeof(\*inode\_buffer));

-           break;

-       }

-       sfs\_inode++;

-   }

-// mutex\_unlock(&simplefs\_inodes\_mgmt\_lock);

+   sfs\_inode += inode\_no - 1;

+   inode\_buffer = kmem\_cache\_alloc(sfs\_inode\_cachep, GFP\_KERNEL);

+   BUG\_ON(!inode\_buffer);

+   memcpy(inode\_buffer, sfs\_inode, sizeof(\*inode\_buffer));

   brelse(bh);

   return inode\_buffer;

@@ -341,12 +410,36 @@ int simplefs\_inode\_save(struct super\_block \*sb, struct simplefs\_inode \*sfs\_inode

   }

   brelse(bh);

-

   mutex\_unlock(&simplefs\_sb\_lock);

   return 0;

}

+static void simplefs\_dentry\_release(struct dentry \*dentry)

+{

+   struct simplefs\_dir\_cache \*dir\_cache = dentry->d\_fsdata;

+   struct simplefs\_cache\_entry \*tmp, \*cache\_entry;

+

+   if (dir\_cache) {

+       list\_for\_each\_entry\_safe(cache\_entry, tmp, &dir\_cache->free, list) {

+           list\_del(&cache\_entry->list);

+           kmem\_cache\_free(sfs\_entry\_cachep, cache\_entry);

+       }

+

+       list\_for\_each\_entry\_safe(cache\_entry, tmp, &dir\_cache->used, list) {

+           list\_del(&cache\_entry->list);

+           kmem\_cache\_free(sfs\_entry\_cachep, cache\_entry);

+       }

+   }

+

+   kfree(dir\_cache);

+   dentry->d\_fsdata = NULL;

+}

+

+static const struct dentry\_operations simplefs\_dentry\_operations = {

+   .d\_release = simplefs\_dentry\_release,

+};

+

/\* FIXME: The write support is rudimentary. I have not figured out a way to do writes

\* from particular offsets (even though I have written some untested code for this below) efficiently. \*/

ssize\_t simplefs\_write(struct file \* filp, const char \_\_user \* buf, size\_t len,

@@ -364,10 +457,12 @@ ssize\_t simplefs\_write(struct file \* filp, const char \_\_user \* buf, size\_t len,

   int retval;

+#if 0

   retval = generic\_write\_checks(filp, ppos, &len, 0);

   if (retval) {

       return retval;

   }

+#endif

   inode = filp->f\_path.dentry->d\_inode;

   sfs\_inode = SIMPLEFS\_INODE(inode);

@@ -441,10 +536,13 @@ static int simplefs\_create(struct inode \*dir, struct dentry \*dentry,

static int simplefs\_mkdir(struct inode \*dir, struct dentry \*dentry,

            umode\_t mode);

+static int simplefs\_unlink(struct inode \*dir, struct dentry \*dentry);

+

static struct inode\_operations simplefs\_inode\_ops = {

   .create = simplefs\_create,

   .lookup = simplefs\_lookup,

   .mkdir = simplefs\_mkdir,

+   .unlink = simplefs\_unlink,

};

static int simplefs\_create\_fs\_object(struct inode \*dir, struct dentry \*dentry,

@@ -452,18 +550,26 @@ static int simplefs\_create\_fs\_object(struct inode \*dir, struct dentry \*dentry,

{

   struct inode \*inode;

   struct simplefs\_inode \*sfs\_inode;

-   struct super\_block \*sb;

+   struct super\_block \*sb = dir->i\_sb;

   struct simplefs\_inode \*parent\_dir\_inode;

   struct buffer\_head \*bh;

   struct simplefs\_dir\_record \*dir\_contents\_datablock;

+   struct dentry \*parent\_dentry = dentry->d\_parent;

+   struct simplefs\_cache\_entry \*cache\_entry;

+   struct simplefs\_dir\_cache \* dir\_cache;

+   struct simplefs\_sb\_info \*sb\_info = SIMPLEFS\_SB(sb);

   uint64\_t count;

   int ret;

+   BUG\_ON(parent\_dentry->d\_inode != dir);

+

+   dir\_cache = (struct simplefs\_dir\_cache \*)parent\_dentry->d\_fsdata;

+   BUG\_ON(!dir\_cache);

+

   if (mutex\_lock\_interruptible(&simplefs\_directory\_children\_update\_lock)) {

       sfs\_trace("Failed to acquire mutex lock\n");

       return -EINTR;

   }

-   sb = dir->i\_sb;

   ret = simplefs\_sb\_get\_objects\_count(sb, &count);

   if (ret < 0) {

@@ -486,6 +592,7 @@ static int simplefs\_create\_fs\_object(struct inode \*dir, struct dentry \*dentry,

       return -EINVAL;

   }

+   /\* create inode \*/

   inode = new\_inode(sb);

   if (!inode) {

       mutex\_unlock(&simplefs\_directory\_children\_update\_lock);

@@ -494,8 +601,9 @@ static int simplefs\_create\_fs\_object(struct inode \*dir, struct dentry \*dentry,

   inode->i\_sb = sb;

   inode->i\_op = &simplefs\_inode\_ops;

-   inode->i\_atime = inode->i\_mtime = inode->i\_ctime = CURRENT\_TIME;

-   inode->i\_ino = (count + SIMPLEFS\_START\_INO - SIMPLEFS\_RESERVED\_INODES + 1);

+   inode->i\_atime = inode->i\_mtime = inode->i\_ctime = current\_time(inode);

+   //inode->i\_ino = (count + SIMPLEFS\_START\_INO - SIMPLEFS\_RESERVED\_INODES + 1);

+   inode->i\_ino = ffz(sb\_info->imap);

   sfs\_inode = kmem\_cache\_alloc(sfs\_inode\_cachep, GFP\_KERNEL);

   sfs\_inode->inode\_no = inode->i\_ino;

@@ -527,18 +635,28 @@ static int simplefs\_create\_fs\_object(struct inode \*dir, struct dentry \*dentry,

   }

   simplefs\_inode\_add(sb, sfs\_inode);

-

+   /\* Read directory \*/

   parent\_dir\_inode = SIMPLEFS\_INODE(dir);

+   /\* get a free place for record \*/

+   cache\_entry = free\_cache\_entry\_get(dir\_cache);

+   BUG\_ON(!cache\_entry);

+

   bh = sb\_bread(sb, parent\_dir\_inode->data\_block\_number);

   BUG\_ON(!bh);

   dir\_contents\_datablock = (struct simplefs\_dir\_record \*)bh->b\_data;

   /\* Navigate to the last record in the directory contents \*/

-   dir\_contents\_datablock += parent\_dir\_inode->dir\_children\_count;

+   dir\_contents\_datablock += cache\_entry->entry\_no;

   dir\_contents\_datablock->inode\_no = sfs\_inode->inode\_no;

   strcpy(dir\_contents\_datablock->filename, dentry->d\_name.name);

+   memcpy(&cache\_entry->record, dir\_contents\_datablock, sizeof(struct simplefs\_dir\_record));

+   cache\_entry\_insert(&dir\_cache->used, cache\_entry);

+

+#ifdef SIMPLEFS\_DEBUG

+   travers\_dir\_cache(dir\_cache);

+#endif

   mark\_buffer\_dirty(bh);

   sync\_dirty\_buffer(bh);

@@ -571,6 +689,87 @@ static int simplefs\_create\_fs\_object(struct inode \*dir, struct dentry \*dentry,

   return 0;

}

+static int simplefs\_unlink(struct inode \*dir, struct dentry \*dentry)

+{

+   struct buffer\_head \*bh;

+   struct super\_block \*sb = dir->i\_sb;

+   int err = -ENOENT;

+   struct inode \*inode = d\_inode(dentry);

+   struct simplefs\_inode \*sfs\_inode = SIMPLEFS\_INODE(inode);

+   struct simplefs\_inode \*dir\_sfs\_inode = SIMPLEFS\_INODE(dir);

+   struct simplefs\_dir\_record \*record;

+   struct dentry \*parent\_dentry = dentry->d\_parent;

+   struct simplefs\_cache\_entry \*cache\_entry;

+   struct simplefs\_dir\_cache \* dir\_cache;

+   struct simplefs\_sb\_info \*sb\_info = SIMPLEFS\_SB(sb);

+

+   BUG\_ON(parent\_dentry->d\_inode != dir);

+

+   dir\_cache = (struct simplefs\_dir\_cache \*)parent\_dentry->d\_fsdata;

+   BUG\_ON(!dir\_cache);

+

+   if (mutex\_lock\_interruptible(&simplefs\_directory\_children\_update\_lock))

+   {

+       sfs\_trace("Failed to acquire mutex lock\n");

+       return -EINTR;

+   }

+

+   bh = sb\_bread(sb, dir\_sfs\_inode->data\_block\_number);

+   BUG\_ON(!bh);

+

+   cache\_entry = used\_cache\_entry\_get(dir\_cache, dentry);

+   if (!cache\_entry)

+       goto end\_unlink;

+

+   record = (struct simplefs\_dir\_record \*)bh->b\_data;

+   record += cache\_entry->entry\_no;

+   record->inode\_no = 0;

+

+   memset(&cache\_entry->record, 0, sizeof(struct simplefs\_dir\_record));

+   cache\_entry\_insert(&dir\_cache->free, cache\_entry);

+

+   dir->i\_ctime = dir->i\_mtime = current\_time(dir);

+

+   mark\_buffer\_dirty(bh);

+   sync\_dirty\_buffer(bh);

+   brelse(bh);

+

+   if (mutex\_lock\_interruptible(&simplefs\_inodes\_mgmt\_lock))

+   {

+       sfs\_trace("Failed to acquire mutex lock\n");

+       mutex\_unlock(&simplefs\_directory\_children\_update\_lock);

+       return -EINTR;

+   }

+

+   dir\_sfs\_inode->dir\_children\_count--;

+   err = simplefs\_inode\_save(sb, dir\_sfs\_inode);

+   if (err)

+   {

+       mutex\_unlock(&simplefs\_inodes\_mgmt\_lock);

+       mutex\_unlock(&simplefs\_directory\_children\_update\_lock);

+       return err;

+   }

+

+   clear\_bit(sfs\_inode->inode\_no, &sb\_info->imap);

+   sfs\_inode->inode\_no = 0;

+

+   err = simplefs\_inode\_save(sb, sfs\_inode);

+   if (err)

+   {

+       mutex\_unlock(&simplefs\_inodes\_mgmt\_lock);

+       mutex\_unlock(&simplefs\_directory\_children\_update\_lock);

+       return err;

+   }

+

+   mutex\_unlock(&simplefs\_inodes\_mgmt\_lock);

+   mutex\_unlock(&simplefs\_directory\_children\_update\_lock);

+   inode->i\_ctime = dir->i\_ctime;

+   return 0;

+

+end\_unlink:

+   return err;

+}

+

static int simplefs\_mkdir(struct inode \*dir, struct dentry \*dentry,

            umode\_t mode)

{

@@ -591,56 +790,76 @@ struct dentry \*simplefs\_lookup(struct inode \*parent\_inode,

   struct simplefs\_inode \*parent = SIMPLEFS\_INODE(parent\_inode);

   struct super\_block \*sb = parent\_inode->i\_sb;

   struct buffer\_head \*bh;

-   struct simplefs\_dir\_record \*record;

-   int i;

+   struct dentry \*parent\_dentry;

+   struct simplefs\_dir\_cache \*dir\_cache;

+   struct simplefs\_cache\_entry \*cache\_entry;

+   struct inode \*inode;

+   struct simplefs\_inode \*sfs\_inode;

-   bh = sb\_bread(sb, parent->data\_block\_number);

-   BUG\_ON(!bh);

+   parent\_dentry = child\_dentry->d\_parent;

+ //BUG\_ON(parent\_dentry->d\_inode != parent\_inode);

+ if (parent\_dentry->d\_inode != parent\_inode)

+       return ERR\_PTR(-ENOENT);

-   record = (struct simplefs\_dir\_record \*)bh->b\_data;

-   for (i = 0; i < parent->dir\_children\_count; i++) {

-       if (!strcmp(record->filename, child\_dentry->d\_name.name)) {

-           /\* FIXME: There is a corner case where if an allocated inode,

-            \* is not written to the inode store, but the inodes\_count is

-            \* incremented. Then if the random string on the disk matches

-            \* with the filename that we are comparing above, then we

-            \* will use an invalid uninitialized inode \*/

-

-           struct inode \*inode;

-           struct simplefs\_inode \*sfs\_inode;

-

-           sfs\_inode = simplefs\_get\_inode(sb, record->inode\_no);

-

-           inode = new\_inode(sb);

-           inode->i\_ino = record->inode\_no;

-           inode\_init\_owner(inode, parent\_inode, sfs\_inode->mode);

-           inode->i\_sb = sb;

-           inode->i\_op = &simplefs\_inode\_ops;

-

-           if (S\_ISDIR(inode->i\_mode))

-               inode->i\_fop = &simplefs\_dir\_operations;

-           else if (S\_ISREG(inode->i\_mode))

-               inode->i\_fop = &simplefs\_file\_operations;

-           else

-               printk(KERN\_ERR

-                "Unknown inode type. Neither a directory nor a file");

-

-           /\* FIXME: We should store these times to disk and retrieve them \*/

-           inode->i\_atime = inode->i\_mtime = inode->i\_ctime =

-            CURRENT\_TIME;

-

-           inode->i\_private = sfs\_inode;

-

-           d\_add(child\_dentry, inode);

-           return NULL;

-       }

-       record++;

+ dir\_cache = (struct simplefs\_dir\_cache \*)parent\_dentry->d\_fsdata;

+

+   if (!dir\_cache) {

+       parent\_dentry->d\_fsdata = simplefs\_cache\_alloc();

+       if (IS\_ERR(parent\_dentry->d\_fsdata))

+        return parent\_dentry->d\_fsdata;

+

+       bh = sb\_bread(sb, parent->data\_block\_number);

+       BUG\_ON(!bh);

+

+       dir\_cache = (struct simplefs\_dir\_cache \*)parent\_dentry->d\_fsdata;

+       dir\_cache\_build(dir\_cache, bh);

   }

+#ifdef SIMPLEFS\_DEBUG

+   travers\_dir\_cache(dir\_cache);

+#endif

+

+   cache\_entry = used\_cache\_entry\_get(dir\_cache, child\_dentry);

+   if (!cache\_entry)

+       goto out;

+

+   /\* FIXME: There is a corner case where if an allocated inode,

+    \* is not written to the inode store, but the inodes\_count is

+    \* incremented. Then if the random string on the disk matches

+    \* with the filename that we are comparing above, then we

+    \* will use an invalid uninitialized inode \*/

+   sfs\_inode = simplefs\_get\_inode(sb, cache\_entry->record.inode\_no);

+   if (!sfs\_inode)

+       return ERR\_PTR(-ENOENT);

+

+   inode = new\_inode(sb);

+   inode->i\_ino = cache\_entry->record.inode\_no;

+   inode\_init\_owner(inode, parent\_inode, sfs\_inode->mode);

+   inode->i\_sb = sb;

+   inode->i\_op = &simplefs\_inode\_ops;

+

+   if (S\_ISDIR(inode->i\_mode))

+       inode->i\_fop = &simplefs\_dir\_operations;

+   else if (S\_ISREG(inode->i\_mode))

+       inode->i\_fop = &simplefs\_file\_operations;

+   else

+       printk(KERN\_ERR

+        "Unknown inode type. Neither a directory nor a file");

+

+   /\* FIXME: We should store these times to disk and retrieve them \*/

+   inode->i\_atime = inode->i\_mtime = inode->i\_ctime =

+    current\_time(inode);

+

+   inode->i\_private = sfs\_inode;

+

+   d\_add(child\_dentry, inode);

+#if 0

   printk(KERN\_ERR

    "No inode found for the filename [%s]\n",

    child\_dentry->d\_name.name);

+#endif

+out:

   return NULL;

}

@@ -661,6 +880,43 @@ static const struct super\_operations simplefs\_sops = {

   .destroy\_inode = simplefs\_destory\_inode,

};

+#ifdef SIMPLEFS\_DEBUG

+static void imap\_dump(struct super\_block \*sb)

+{

+   struct simplefs\_sb\_info \*sb\_info = sb->s\_fs\_info;

+   pr\_info("starting imap dump: imap %lu\n", sb\_info->imap);

+}

+#endif

+

+static void fill\_imap(struct super\_block \*sb)

+{

+   int i;

+   struct simplefs\_sb\_info \*sb\_info = sb->s\_fs\_info;

+   struct simplefs\_inode \*simple\_inode;

+   struct buffer\_head \*bh;

+   int icount = SIMPLEFS\_DEFAULT\_BLOCK\_SIZE / sizeof(struct simplefs\_inode);

+

+   bh = sb\_bread(sb, SIMPLEFS\_INODESTORE\_BLOCK\_NUMBER);

+   simple\_inode = (struct simplefs\_inode \*)bh->b\_data;

+

+   for (i = 0; i < SIMPLEFS\_START\_INO; i++)

+       set\_bit(i, &sb\_info->imap);

+

+   simple\_inode += SIMPLEFS\_START\_INO;

+   for (i = SIMPLEFS\_START\_INO; i < icount; i++) {

+       if (simple\_inode->inode\_no != 0) {

+           pr\_err("func %s, line %d, ino %lld\n", \_\_func\_\_, \_\_LINE\_\_, simple\_inode->inode\_no);

+           set\_bit(i, &sb\_info->imap);

+       }

+       simple\_inode++;

+   }

+

+   brelse(bh);

+#ifdef SIMPLEFS\_DEBUG

+   imap\_dump(sb);

+#endif

+}

+

/\* This function, as the name implies, Makes the super\_block valid and

\* fills filesystem specific information in the super block \*/

int simplefs\_fill\_super(struct super\_block \*sb, void \*data, int silent)

@@ -668,6 +924,7 @@ int simplefs\_fill\_super(struct super\_block \*sb, void \*data, int silent)

   struct inode \*root\_inode;

   struct buffer\_head \*bh;

   struct simplefs\_super\_block \*sb\_disk;

+   struct simplefs\_sb\_info \*sb\_info;

   int ret = -EPERM;

   bh = sb\_bread(sb, SIMPLEFS\_SUPERBLOCK\_BLOCK\_NUMBER);

@@ -698,10 +955,20 @@ int simplefs\_fill\_super(struct super\_block \*sb, void \*data, int silent)

   sb->s\_magic = SIMPLEFS\_MAGIC;

   /\* For all practical purposes, we will be using this s\_fs\_info as the super block \*/

-   sb->s\_fs\_info = sb\_disk;

+   sb\_info = kzalloc(sizeof(struct simplefs\_sb\_info), GFP\_KERNEL);

+   if (!sb\_info) {

+       brelse(bh);

+       return -ENOMEM;

+   }

+

+   sb\_info->sb = sb\_disk;

+   sb\_info->bh = bh;

+   sb->s\_fs\_info = sb\_info;

+   fill\_imap(sb);

   sb->s\_maxbytes = SIMPLEFS\_DEFAULT\_BLOCK\_SIZE;

   sb->s\_op = &simplefs\_sops;

+   sb->s\_d\_op = &simplefs\_dentry\_operations;

   root\_inode = new\_inode(sb);

   root\_inode->i\_ino = SIMPLEFS\_ROOTDIR\_INODE\_NUMBER;

@@ -710,7 +977,7 @@ int simplefs\_fill\_super(struct super\_block \*sb, void \*data, int silent)

   root\_inode->i\_op = &simplefs\_inode\_ops;

   root\_inode->i\_fop = &simplefs\_dir\_operations;

   root\_inode->i\_atime = root\_inode->i\_mtime = root\_inode->i\_ctime =

-    CURRENT\_TIME;

+    current\_time(root\_inode);

   root\_inode->i\_private =

    simplefs\_get\_inode(sb, SIMPLEFS\_ROOTDIR\_INODE\_NUMBER);

@@ -726,13 +993,13 @@ int simplefs\_fill\_super(struct super\_block \*sb, void \*data, int silent)

   if (!sb->s\_root) {

       ret = -ENOMEM;

+       kfree(sb\_info);

       goto release;

   }

-   ret = 0;

+   return 0;

release:

   brelse(bh);

-

   return ret;

}

@@ -755,11 +1022,14 @@ static struct dentry \*simplefs\_mount(struct file\_system\_type \*fs\_type,

static void simplefs\_kill\_superblock(struct super\_block \*sb)

{

+   struct simplefs\_sb\_info \*sb\_info = sb->s\_fs\_info;

   printk(KERN\_INFO

    "simplefs superblock is destroyed. Unmount succesful.\n");

   /\* This is just a dummy function as of now. As our filesystem gets matured,

    \* we will do more meaningful operations here \*/

+   brelse(sb\_info->bh);

+   kfree(sb\_info);

   kill\_block\_super(sb);

   return;

}

@@ -785,6 +1055,17 @@ static int simplefs\_init(void)

       return -ENOMEM;

   }

+   sfs\_entry\_cachep = kmem\_cache\_create("sfs\_entry\_cachep",

+                                       sizeof(struct simplefs\_cache\_entry),

+                                       0,

+                                       (SLAB\_RECLAIM\_ACCOUNT| SLAB\_MEM\_SPREAD),

+                                       NULL);

+

+   if (!sfs\_entry\_cachep) {

+       kmem\_cache\_destroy(sfs\_inode\_cachep);

+       return -ENOMEM;

+   }

+

   ret = register\_filesystem(&simplefs\_fs\_type);

   if (likely(ret == 0))

       printk(KERN\_INFO "Sucessfully registered simplefs\n");

@@ -799,6 +1080,7 @@ static void simplefs\_exit(void)

   int ret;

   ret = unregister\_filesystem(&simplefs\_fs\_type);

+   kmem\_cache\_destroy(sfs\_entry\_cachep);

   kmem\_cache\_destroy(sfs\_inode\_cachep);

   if (likely(ret == 0))

diff --git a/simple.h b/simple.h

index e7f1f96..acaa7e3 100644

--- a/simple.h

+++ b/simple.h

@@ -1,15 +1,16 @@

-

#define SIMPLEFS\_MAGIC 0x10032013

#define SIMPLEFS\_DEFAULT\_BLOCK\_SIZE 4096

#define SIMPLEFS\_FILENAME\_MAXLEN 255

#define SIMPLEFS\_START\_INO 10

+#define SIMPLEFS\_ICOUNT 256

/\*\*

\* Reserver inodes for super block, inodestore

\* and datablock

\*/

#define SIMPLEFS\_RESERVED\_INODES 3

+#define SIMPLEFS\_MAX\_CHILDREN\_CNT (SIMPLEFS\_DEFAULT\_BLOCK\_SIZE / sizeof(struct simplefs\_dir\_record))

#ifdef SIMPLEFS\_DEBUG

#define sfs\_trace(fmt, ...) { \

   printk(KERN\_ERR "[simplefs] %s +%d:" fmt, \

@@ -75,3 +76,9 @@ struct simplefs\_super\_block {

   char padding[SIMPLEFS\_DEFAULT\_BLOCK\_SIZE - (5 \* sizeof(uint64\_t))];

};

+

+struct simplefs\_sb\_info {

+   struct simplefs\_super\_block \*sb;

+   unsigned long imap;

+   struct buffer\_head \*bh;

+};

diff --git a/super.h b/super.h

index ddfa713..40da6bb 100644

--- a/super.h

+++ b/super.h

@@ -1,6 +1,6 @@

#include "simple.h"

-static inline struct simplefs\_super\_block \*SIMPLEFS\_SB(struct super\_block \*sb)

+static inline struct simplefs\_sb\_info \*SIMPLEFS\_SB(struct super\_block \*sb)

{

   return sb->s\_fs\_info;

}

-- hgr

2.17.0

# 附件

