

Re: [PATCH 1/2] LogFS proper

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-05/msg03853.html>

- *From:* Jörn Engel <joern@xxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 8 May 2007 22:25:20 +0200
-

Before I forget this again: thanks for the review! It really is appreciated.

On Tue, 8 May 2007 20:00:41 +0200, Thomas Gleixner wrote:

On Tue, 2007-05-08 at 18:32 +0200, Jörn Engel wrote:

Please sort includes alphabetically and separate the
#include <linux/mtd/mtd.h> from the #include <linux/...>
ones

Sort: will do.
Separation: Any particular reason for that?

Easier to see the different <include/xxx> categories

I'm not convinced, but neither do I care enough to argue.

```
+#define packed
__attribute__((__packed__))
```

Please use the `__attribute__((__packed__))` on your structs
instead of
creating some extra "needs lookup" magic.

Actually I would prefer to understand what that attribute actually does.

It ensures that gcc does not align things according to its own idea of
optimized access.

All structure members should be properly aligned, so having this

Re: [PATCH 1/2] LogFS proper

attribute is pure paranoia. The definition is just there to make my eyes tear less.

Would anything potentially break if I just ripped that out?

It's gcc :)

Sounds like I'm not the only paranoid around. Oh well!

Please comment the structure with kernel doc comments and avoid the tail comments.

I'd like to hear your rationale.

Kernel doc comments as:

```
/**
 * struct hrtimer – the basic hrtimer structure
 * @node: red black tree node for time ordered insertion
 * @expires: the absolute expiry time in the hrtimers internal
 * representation. The time is related to the clock on
 * which the timer is based.
```

give you a nice overview with enough space for good explanations and can be converted to kernel doc as well.

That makes sense, at least for anything that can be described as an interfaces. As for the kernel-only header – not sure yet.

enum please

I don't care much one way or another. Do enums have a significant advantage?

yes, type checking

```
+
+struct logfs_segment_header {
+ be32 crc; /* checksum */
+ be16 len; /* length of object, header not
```

Re: [PATCH 1/2] LogFS proper

Re: [PATCH 1/2] LogFS proper

```
included */
+ u8 type; /* node type */
+ u8 level; /* GC level */
+ be32 segno; /* segment number */
+ be32 ec; /* erase count */
+ be64 gec; /* global erase count (write
time) */
+}packed;
+
+enum {
+ COMPR_NONE = 0,
+ COMPR_ZLIB = 1,
+};
```

Please name the enums and use the same enum for the
according fields and
the function arguments.

Does sparse check on that? That would be quite useful and stop my
ambivalence.

also the compiler complains

Reason enough to use it for the simple cases.

Not sure. Those constants are actually in groups of 16, so they are a
weird mixture of bitfields and enums. There is code roughly along these
lines:

```
switch (i >> 4) {
case 0:
switch (i & 0xf) {
case JE_COMMIT:
case JE_ABORT:
...
case 1:
...
}
```

I'll have to check whether enums support this.

Hmm, ok. But this needs some comment then

Sure.

Re: [PATCH 1/2] LogFS proper

Re: [PATCH 1/2] LogFS proper

I can see the point for an inline function. But lowercase would change a style that appears to be common in Linux filesystems.

Well, we have uppercase MACROs and lower case function names.

Will you send the janitorial patches for existing code?

:)

Then I will leave the casing for some energetic janitor as well. :)

Istr enums having severe problems for anything larger than int. LogFS inodes are 64bit. Hmm. And how do enums behave wrt. cpu_to_beXX and sparse?

Hmm, good question.

And these will remain macros as well. Without type checking there doesn't seem to be a compelling reason left.

```
+ u64 s_free_bytes; /* number of free bytes
*/
```

```
+#define journal_for_each(__i) for (__i=0;
__i<LOGFS_JOURNAL_SEGS; __i++)
```

```
__i = 0; __i < LOGFS_JOURNAL_SEGS;
```

Will that make the code look better or just slavishly follow indentation guidelines? Adding spaces where you suggested weakens the grouping of the three for(;;) parameters, imo.

```
(__i = 0; __i < LOGFS_JOURNAL_SEGS; __i++)
```

is way simpler to parse than

```
(__i=0; __i<LOGFS_JOURNAL_SEGS; __i++)
```

Re: [PATCH 1/2] LogFS proper

Re: [PATCH 1/2] LogFS proper

If that statement was meant to be generic, it failed for me. Now, I happen to be used to one thing and you are used to another, so a large part of that may only be habit. Still, I have have thought about what I'm doing and believe to have a slightly more objective reason (better grouping).

```
+void logfs_crash_dump(struct super_block *sb);  
  
+ #define LOGFS_BUG(sb) do { \  
+ struct super_block *__sb = sb; \  
+ }
```

Why do we need a local variable here ?

Trying to add type safety. It cannot be an inline function if without making the file/line information useless.

```
#define LOGFS_BUG(sb) logfs_bug(sb, __FUNCTION__, __LINE__)
```

Also the BUG itself will give you enough clue where it happened, so having the function/line info is not really necessary

If that were true, why are function and line included in BUG() then?

And after looking up BUG(), why is the loglevel missing from every printk in it? Looks like a "naked" printk is far from uncommon. Plus, in my testing, something magically added a default (<4>) loglevel to every printk.

This leaves me a bit puzzled and wondering whether I should change each and every printk in my code. After killing the bogus ones, of course.

```
+static inline u8 logfs_type(struct inode  
+inode)  
+{  
+ return (inode->i_mode >> 12) & 15;
```

What's 12 and 15 ? Constants perhaps ?

There should be a generic function doing just the same. At least this is better than the open-coded variants elsewhere:

```
fs/jffs2/dir.c: type = (old_dentry->d_inode->i_mode & S_IFMT) >> 12;
```

Re: [PATCH 1/2] LogFS proper

```
fs/jffs2/dir.c: type = (old_dentry->d_inode->i_mode & S_IFMT) >> 12;  
fs/libfs.c: return (inode->i_mode >> 12) & 15;  
fs/nfs/dir.c: return (inode->i_mode >> 12) & 15;  
fs/proc/base.c: type = inode->i_mode >> 12;
```

Maybe the libfs version could get moved to a header somewhere.

Yes please

Does anyone have a header suggestion? fs.h is the obvious one, although it looks like the last thing it needs is even more content.

```
+int logfs_memcpy(void *in, void *out,  
size_t inlen, size_t outlen);  
+int logfs_compress(void *in, void *out,  
size_t inlen, size_t outlen);  
+int logfs_compress_vec(struct kvec *vec,  
int count, void *out, size_t outlen);  
+int logfs_uncompress(void *in, void *out,  
size_t inlen, size_t outlen);  
+int logfs_uncompress_vec(void *in, size_t  
inlen, struct kvec *vec, int count);
```

are those global ? If yes, please add extern, else remove

What purpose does "extern" have? To my understanding it makes zero difference. About half the headers use it, the other half doesn't.

and yours uses it in one place and not in the other.

Hey, I _do_ use it. But only for logfs_*_ops. So clearly I did it once and then copied one from another.

extern is an empty macro, but it makes it clear that this is a global function declaration

Anyone living in any doubt that a function declaration in a header is not meant to be global has my deepest sympathy. I'll kill the existing "extern"s.

Re: [PATCH 1/2] LogFS proper

```
+static inline u64 dev_ofs(struct super_block
*sb, u32 segno, u32 ofs)
+{
+ struct logfs_super *super =
LOGFS_SUPER(sb);
```

Separate variables and code by an empty line please

In general: sure. But for 1–2 line functions the empty lines seem to hurt more than they help.

No, it's about pattern recognition. Consistent patterns allow faster parsing.

You have a point there.

As much as I agree with the kernel coding style, I have never liked to slavishly follow any written doctrine. The overall goal should be easy to read. If "easy to read" would match the wording 100%, someone should adjust the Lindent parameters and run the whole kernel through.

```
+ LOGFS_BUG_ON(err, sb);
```

Please open code this instead of nesting mtdread into device_read and therefor avoid the error handling pathes in those places where device_read is used.

Open code the LOGFS_BUG_ON()? What purpose would that serve?

No, open code device_read and add the error path at the place where device_read is used and put a bug in the error path for now.

But that is pointless. In particular the "for now" part is pointless. "For now" is exactly what I have already. And the less I waste my time with cosmetics the sooner I can spend time to fix it "for good".

```
+static s64 dir_seek_data(struct inode
*inode, s64 pos)
```

Re: [PATCH 1/2] LogFS proper

Re: [PATCH 1/2] LogFS proper

```
+ return 0;
+ if (ret == -ENODATA) { /* deleted dentry
*/
```

Please move the comment away. It makes parsing hard

ENOPARSE

Do you want an extra space or tab?

No, please remove the tail comment after the {

You have to come up with a better reason than your personal preference.
In particular when your suggestion is to remove useful documentation.

What is the rationale here?

Pattern recognition

What was the context again? Oh, yes, comments. Hmm.

```
/* fairly short one-line comment; is just barely within 80 columns */
/*
* slightly longer two-line comment; would be just barely over 80
* columns
*/
```

I think it is unfortunate that the second comment is just 6 characters longer and yet has to fill four lines instead of one. Oh well, consistency wins.

```
+ if (dest) /* symlink */
+ ret = logfs_inode_write(inode, dest,
destlen, 0);
+ else /* creat/mkdir/mknod */
+ ret = __logfs_write_inode(inode);
```

Please remove this confusing tail comments

?!?

Re: [PATCH 1/2] LogFS proper

Re: [PATCH 1/2] LogFS proper

Imo they explain what is going on in either of those cases. Do you consider that to be self-explanatory?

if you think you need comments, then please use new lines, i.e:

```
if (dest) {
/* symlink */
ret = logfs_inode_write(inode, dest, destlen, 0);
} else {
/* creat/mkdir/mknod */
ret = __logfs_write_inode(inode);
}
```

That `_does_` look better. Consider me convinced.

```
+static int logfs_delete_dd(struct inode *dir,
struct logfs_disk_dentry *dd,
+ loff_t pos)
+{
+ int err;
+
+ err = read_dir(dir, dd, pos);
+ if (err == -EOF) /* don't expose internal
errno */
+ err = -EIO;
```

Interesting. Why is EOF morphed to EIO ?

Because deleting something beyond EOF is indeed an error. Although in two cases, this should be a `BUG()` instead, if anything at all.

Journal replay is special. Garbage and/or malicious data on the medium cause this error. The journal CRCs should protect us against garbage, which leaves only the prepared filesystem image to worry about.

I guess I'll just `BUG` in any case.

At least provide a comment for the ignorami.

Can do.

Just like for any other filesystem, fuzzing an image will uncover many bugs. More than in other filesystems simply because LogFS is younger. The only exceptions are JFFS2 and ZFS – and even those only if the

Re: [PATCH 1/2] LogFS proper

attacker^Wresearcher didn't bother to recalculate checksums after fuzzing.

Hardly material for 8 o'clock news.

```
+#include "logfs.h"
+
+
+static int logfs_prepare_write(struct file
+file, struct page *page,
+ unsigned start, unsigned end)
+{
+ if (PageUptodate(page))
+ return 0;
+
+ if ((start == 0) && (end ==
+PAGE_CACHE_SIZE))
+ return 0;
```

Self explaining logic ?

Boilerplate code that every filesystem uses.

I know, I have seen it elsewhere. It still does not make much sense.

Then it should be generically implemented and commented somewhere once, so that other filesystems can just use the functionality.

I haven't closely followed Nick Piggin's work, but it could be entirely possible that some of his patches make this obsolete. Might be a bad time for such a cleanup.

```
+#if 0
```

Can you please remove this ?

Nope. That code will get used in the future.

So why don't you add it when it you start to use it ?

Why do you want to see this code gone? Unlike many of the #if 0 Adrian

Re: [PATCH 1/2] LogFS proper

Re: [PATCH 1/2] LogFS proper

added, this code has a maintainer that cares about it. Surely there must be better candidates for removal.

Interestingly enough this unused function is better commented than anything else in this patch.

With the exception of dir.c. In both cases I was documenting the algorithm used, which is far from obvious. Most other things are fairly straightforward for people used to existing filesystems.

Hmm. Comments are of general use and it's way easier to understand code when it has comments to functions and tricks used in the code. You don't write code for people used to existing filesystems. You write code which is understandable and allows debugging without twisting the brain for non filesystem wizzards who use it and trap into the occasional problem

Then please ask specific questions. My abilities to guess what others consider obvious are quite limited. Doubly so because I am more familiar with many basic (to LogFS) concepts than any potential reader. So familiar in fact, I usually don't even notice.

```
+ sh = (void*)&h;
```

Please use proper type casting !

How would that improve the code? (void*) clearly states that "I don't care what the base type it, just cast this thing to the new pointer type." (struct logfs_segment_header*) would state the same but be less concise.

Hell no. It documents that you actually want to do this IMHO.

That implies I would also do this without actually wanting to. Or maybe not me but some other kernel hackers. Is that realistic? Have such bugs occurred?

```
+ /* This is a blatant copy of alloc_inode  
code. We'd need alloc_inode  
+ * to be nonstatic, alas. */  
+ {
```

Re: [PATCH 1/2] LogFS proper

```
+ static const struct
address_space_operations empty_aops;
+ struct address_space * const mapping =
&inode->i_data;
```

Please remove the brackets and move the variables to the top of the function

Erm? Did you read the comment? I have copied the code from alloc_inode() without changes. That is bad enough as it is. If I were to change the code format, chances of detecting changes in one function not followed in the other would increase even more.

I read the comment, but it did not make any sense versus the brackets.

I'm sure this particular gem can use some discussion, as long as it's not limited to formatting issues.

well, both.

Then let us discuss the more important issue of potentially exporting alloc_inode() first, please.

```
+ level = i & 0xf;
```

what is 0xf ?

```
+ area = super->s_area[level];
+ switch (i & ~0xf) {
+ case JEG_BASE:
+ switch (i) {
```

Represents I an enum or a bitfield or both ?

Both. High nibble groups the journal entries. High nibble 0 are the normal journal entries. High nibble 1 are the summaries for all levels.

"Levels" is something I should document, seeing that most people haven't watched my LCA presentation.

Re: [PATCH 1/2] LogFS proper

Re: [PATCH 1/2] LogFS proper

I know roughly how it works. It just is not obvious and really needs some comments.

Ack.

```
+static void journal_get_free_segment(struct
logfs_area *area)
+{
+ struct logfs_super *super =
LOGFS_SUPER(area->a_sb);
+ int i;
+
+ journal_for_each(i) {
+ if (area->a_segno !=
super->s_journal_seg[i])
+ continue;
+empty_seg:
+ i++;
+ if (i == LOGFS_JOURNAL_SEGS)
+ i = 0;
+ if (!super->s_journal_seg[i])
+ goto empty_seg;
```

Does this loop for ever or is there a guranteed exit ?
Please use a do while loop instead of the goto

There is a guaranteed exit. mkfs can specify up to four segments (read erase blocks) for the journal to live in. Two are the required minimum. In order to specify just two segments, the array will be initialized like {1, 2, 0, 0}.

This code shall find the current segment from that array, then pick the next one and skip over any entries that are zero.

I thought that, but it needs a comment as well

Ack.

Send me a testcase. :)

Use nand error injection :)

I'll inject errors in ramtd then. If nothing else, at least I'm familiar with that beast.

As above, I prefer explicitly stating "this has never happened, I have no clue what should be done" over some half-assed "I hope this works, even though noone ever tested it".

Both are lame, one just happens to be slightly less wicked and a lot more honest.

Well, at least it would be good to return the problem back to the place, where it actually would do damage and BUG there, so it is more obvious where you need to work on error handling. Bugs in the middle of nowhere are not really helpful

Helpful to whom? If you volunteered to do this testing, I will gladly change the code to your liking. If, as expected, I will do this work then I actually like it as it is "for now". :)

```
+ ret = mtdwrite(sb, ofs, sb->s_blocksize,  
block);  
+ if (ret)  
+ return ret;  
+ return 0;
```

Interesting way to rely on compiler smartness

Que?

```
if (ret)  
return ret;  
return 0;
```

might be optimized by a smart compiler to

```
return ret;
```

but you should do it yourself, as gcc is not always smart

Ah, yes. One day I should go through all my patches, interdiff them and see how many lines I actually wrote. There has been a huge amount of churn and this is not the first case where I missed an obvious cleanup

Re: [PATCH 1/2] LogFS proper

after some other code change.

Any comments to used code you would like to see? Your pattern appears to be "remove comment". :)

No, "move comment away from the tail" :)

Yup. I'm converted.

Comments to functions and tricky non obvious code would be really appreciated.

Sometimes I get this feeling that none of my code is obvious. Maybe I should add a file giving a rough design overview and what part of the design each file is supposed to deal with.

```
+ if (*ppos >= size)
+ return 0;
+ if (count > size - *ppos)
+ count = size - *ppos;
+
+ BUG_ON(logfs_index(*ppos) !=
logfs_index(*ppos + count - 1));
+
+ block_data =
kzalloc(LOGFS_BLOCKSIZE,
GFP_KERNEL);
+ if (!block_data)
+ goto fail;
+
+ err = logfs_read_block(inode,
logfs_index(*ppos), block_data,
+ read_zero);
+ if (err)
+ goto fail;
+
+ memcpy(buf, block_data + (*ppos %
LOGFS_BLOCKSIZE), count);
+ *ppos += count;
+ kfree(block_data);
+ return count;
```

err = count; and fall trough ?

Re: [PATCH 1/2] LogFS proper

Then I would change *ppos.

err ?

Lack of coffee (or sleep, since I don't drink coffee). Now I see what you mean.

```
+ ret = ret==n ? 0 : -EIO;
```

```
return ret == n ? ..... perhaps ?
```

Again I consider the lack of spaces to give better grouping. It is similar to brackets. In general they help, but then there is Lisp...

dickhead :)

:)

```
+int mtderase(struct super_block *sb, loff_t  
ofs, size_t len)  
{  
+ struct mtd_info *mtd =  
LOGFS_SUPER(sb)->s_mtd;  
+ struct inode *inode =  
LOGFS_SUPER(sb)->s_dev_inode;  
+ struct erase_info ei;  
+ int ret;  
+  
+ BUG_ON(len % mtd->erasesize);  
+  
+  
truncate_inode_pages_range(&inode->i_data,  
ofs, ofs+len-1);  
+ if (mtd->block_isbad(mtd, ofs))  
+ return -EIO;
```

this actually leads to a double check of block_isbad for blocks which are not bad.

Re: [PATCH 1/2] LogFS proper

Does it? Where is the second check happening?

in mtd->erase()

Does not seem to be documented either. Not sure if I can trust every driver on it. But I should be able to trust my own code, which is tracking bad blocks as well. Will kill.

No. This one will print a little statistic about segment usage. Something like:

```
0 0 0 0 20000 12345 01234 ...
```

It is useful as-is for fsck purposes, except that the lines wrap since I count bytes instead of blocks now. "blocks" is a strange concept once they get compressed.

Still something like:

```
LOGFS 0 0 0 0 20000 12345 01234 ...  
LOGFS 0 0 0 0 20000 12345 01234 ...
```

makes it easier to find in the logs

Finding it in the logs when looking for it is definitely not a problem.

What could be a problem is that people not looking for this could find it in their logs. So the fsck as a whole should be hidden behind a big sign forbidding civilians and children to enter. Or just moved to userspace.

Not anymore, this can go. But since we are on the subject, what is the difference between yield() and cond_resched()? Those two functions could also use slightly better comments.

cond_resched() calls schedule, when the need_resched flag of the task is set. yield() goes through schedule always and should not be used in the kernel.

Thanks.

Re: [PATCH 1/2] LogFS proper

Re: [PATCH 1/2] LogFS proper

Humm. So far those functions are unused. And I'm starting to doubt their usefulness. The commented-out code should be pure paranoia, but that hardly matters now, does it.

In a review it matters, as it raises questions, doesn't it.

Raising questions definitely matters. What doesn't matter (anymore) are those comments, as the functions are on my black list.

```
+static void ostore_get_free_segment(struct
logfs_area *area)
+{
+ struct logfs_super *super =
LOGFS_SUPER(area->a_sb);
+ struct logfs_segment *seg;
+
+
BUG_ON(list_empty(&super->s_free_list));
+
+ seg = list_entry(super->s_free_list.prev,
struct logfs_segment, list);
+ list_del(&seg->list);
+ area->a_segno = seg->segno;
+ kfree(seg);
+ super->s_free_count -= 1;
```

get_free_segment actually kfree's a segment ? Please use a less misleading function name

It actually gets a free segment. It also kfree's an object that happens to be called logfs_segment. Both names make sense on their own. The combination... can be confusing.

I'm not exactly sure what to do here.

At least add a comment !

Will do.

Re: [PATCH 1/2] LogFS proper

```
+++ linux-2.6.21logfs/fs/logfs/memtree.c
2007-05-07 13:32:12.000000000 +0200
@@ -0,0 +1,199 @@
+/* In-memory B+Tree. */
```

license and a little bit more description

For sure. This could potentially move to lib/

yup

```
+ if (fill-1 < BTREE_NODES/2) {
+ /* XXX */
```

YYYY perhaps ?

Or maybe even so actual code?

Might be even better.

As it is, this is a somewhat generic btree implementation using lazy removal (or else there must be code here). I hacked it up just for learning purposes, but later found it to be useful. And while I haven't done any tests, it should significantly beat rbtree's performance-wise.

Put this explanation into the comment with a FIXME. Is far better than "XXX" :)

At least you ask this year, while I still have a faint memory of what I did. :)

Will do.

Jörn

—

When in doubt, punt. When somebody actually complains, go back and fix it...
The 90% solution is a good thing.

— Rob Landley

—

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in

Re: [PATCH 1/2] LogFS proper

Re: [PATCH 1/2] LogFS proper

the body of a message to majordomo@xxxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>