Subject: [PATCH 2/2] mm: incorrect direct io error handling (v6)
Posted by Dmitriy Monakhov on Mon, 12 Mar 2007 07:57:50 GMT
View Forum Message <> Reply to Message

I realy don't want to be annoying by sending this patcheset over and over
again, i just want the issue to be solved. If anyone think this solution
is realy cappy, please comment what exectly is bad. Thank you.

Changes:
  - patch was split in two patches.
  - comments added. I think now it is clearly describe things.
  - patch prepared against 2.6.20-mm3

How this patch tested:
  - fsstress test.
  - manual direct_io tests.

LOG:
 - Trim off blocks after generic_file_direct_write() has failed.
 - Update out of date comments about direct_io locking rules.

Signed-off-by: Monakhov Dmitriy <dmonakhov@openvz.org>
---
 mm/filemap.c | 32 ++++++++++++++++++++++++++++----
 1 files changed, 28 insertions(+), 4 deletions(-)

diff --git a/mm/filemap.c b/mm/filemap.c
index 0aadf5f..8959ae3 100644
--- a/mm/filemap.c
+++ b/mm/filemap.c
@@ -1925,8 +1925,9 @@ generic_file_direct_write(struct kiocb *iocb, const struct iovec *iov,
  /*
   * Sync the fs metadata but not the minor inode changes and
   * of course not the data as we did direct DMA for the IO.
- * i_mutex is held, which protects generic_osync_inode() from
- * livelocking. AIO O_DIRECT ops attempt to sync metadata here.
+ * i_mutex may not being held, if so some specific locking
+ * ordering must protect generic_osync_inode() from livelocking.
+ * AIO O_DIRECT ops attempt to sync metadata here.
   */
  if ((written >= 0 || written == -EIOCBQUEUED) &&
      ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
@@ -2240,6 +2241,29 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct iovec
*iov,
  mutex_lock(&inode->i_mutex);
  ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
    &iocb->ki_pos);
+ /*

```
+  * If __generic_file_aio_write_nolock has failed.
+  * This may happen because of:
+  * 1) Bad segment found (failed before actual write attempt)
+  * 2) Segments are good, but actual write operation failed
+  *    and may have instantiated a few blocks outside i_size.
+  *   a) in case of buffered write these blocks was already
+  *    trimmed by generic_file_buffered_write()
+  *   b) in case of O_DIRECT these blocks weren't trimmed yet.
+  *
+  * In case of (2b) these blocks have to be trimmed off again.
+  */
+ if (unlikely( ret < 0 && file->f_flags & O_DIRECT)) {
+  unsigned long nr_segs_avail = nr_segs;
+  size_t count = 0;
+  if (!generic_segment_checks(iov, &nr_segs_avail, &count,
+     VERIFY_READ)) {
+   /*It is (2b) case, because segments are good*/
+   loff_t isize = i_size_read(inode);
+   if (pos + count > isize)
+    vmtruncate(inode, isize);
+  }
+ }
   mutex_unlock(&inode->i_mutex);

   if (ret > 0 && ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
@@ -2254,8 +2278,8 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct iovec
*iov,
 EXPORT_SYMBOL(generic_file_aio_write);

 /*
- * Called under i_mutex for writes to S_ISREG files.   Returns -EIO if something
- * went wrong during pagecache shootdown.
+ * May be called without i_mutex for writes to S_ISREG files.
+ * Returns -EIO if something went wrong during pagecache shootdown.
   */
 static ssize_t
 generic_file_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
--
1.5.0.1
```

Subject: Re: [PATCH 2/2] mm: incorrect direct io error handling (v6)
Posted by Nick Piggin on Mon, 12 Mar 2007 08:20:28 GMT

View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 10:58:10AM +0300, Dmitriy Monakhov wrote:
> I realy don't want to be annoying by sending this patcheset over and over
> again, i just want the issue to be solved. If anyone think this solution

> is realy cappy, please comment what exectly is bad. Thank you.

If you don't get feedback, then you have to keep posting. If you still
don't get feedback, try cc'ing a few more lists (eg. linux-fsdevel).

> Changes:
> - patch was split in two patches.
> - comments added. I think now it is clearly describe things.
> - patch prepared against 2.6.20-mm3
>
> How this patch tested:
> - fsstress test.
> - manual direct_io tests.
>
> LOG:
> - Trim off blocks after generic_file_direct_write() has failed.
> - Update out of date comments about direct_io locking rules.

It can be nice to expand on what the problem was, and how you fixed it...
but I guess you do quite a good job in the C comments.

>
> Signed-off-by: Monakhov Dmitriy <dmonakhov@openvz.org>
> ---
> mm/filemap.c | 32 ++++++++++++++++++++++++++++----
> 1 files changed, 28 insertions(+), 4 deletions(-)
>
> diff --git a/mm/filemap.c b/mm/filemap.c
> index 0aadf5f..8959ae3 100644
> --- a/mm/filemap.c
> +++ b/mm/filemap.c
> @@ -1925,8 +1925,9 @@ generic_file_direct_write(struct kiocb *iocb, const struct iovec *iov,
> 	/*
> 	 * Sync the fs metadata but not the minor inode changes and
> 	 * of course not the data as we did direct DMA for the IO.
> -	 * i_mutex is held, which protects generic_osync_inode() from
> -	 * livelocking. AIO O_DIRECT ops attempt to sync metadata here.
> +	 * i_mutex may not being held, if so some specific locking
> +	 * ordering must protect generic_osync_inode() from livelocking.
> +	 * AIO O_DIRECT ops attempt to sync metadata here.
> 	 */

This wasn't exactly clear to me. Did you mean:

 "may be held, which protects generic_osync_inode() from livelocking. If it
  is not held, then the filesystem must prevent this livelock"?

> 	if ((written >= 0 || written == -EIOCBQUEUED) &&

>   ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
> @@ -2240,6 +2241,29 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct iovec
*iov,
>   mutex_lock(&inode->i_mutex);
>   ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
>   &iocb->ki_pos);
> + /*
> + * If __generic_file_aio_write_nolock has failed.
> + * This may happen because of:
> + * 1) Bad segment found (failed before actual write attempt)
> + * 2) Segments are good, but actual write operation failed
> + *   and may have instantiated a few blocks outside i_size.
> + *   a) in case of buffered write these blocks was already
> + *   trimmed by generic_file_buffered_write()
> + *   b) in case of O_DIRECT these blocks weren't trimmed yet.
> + *
> + * In case of (2b) these blocks have to be trimmed off again.
> + */
> + if (unlikely(ret < 0 && file->f_flags & O_DIRECT)) {
> + unsigned long nr_segs_avail = nr_segs;
> + size_t count = 0;
> + if (!generic_segment_checks(iov, &nr_segs_avail, &count,
> +   VERIFY_READ)) {
> +   /*It is (2b) case, because segments are good*/
> +   loff_t isize = i_size_read(inode);
> +   if (pos + count > isize)
> +   vmtruncate(inode, isize);
> + }
> + }

OK, but wouldn't this be better to be done in the actual direct IO
functions themselves? Thus you could be sure that you have the 2b case,
and the code would be less fragile to something changing?

And a minor nit: extra space after "if (unlikely("

>   mutex_unlock(&inode->i_mutex);
>
>   if (ret > 0 && ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
> @@ -2254,8 +2278,8 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct iovec
*iov,
> EXPORT_SYMBOL(generic_file_aio_write);
>
> /*
> - * Called under i_mutex for writes to S_ISREG files.   Returns -EIO if something
> - * went wrong during pagecache shootdown.
> + * May be called without i_mutex for writes to S_ISREG files.

> + * Returns -EIO if something went wrong during pagecache shootdown.
> */

These comments updates are for DIO_OWN_LOCKING, right? In that case, you
should mention that.

Subject: Re: [PATCH 2/2] mm: incorrect direct io error handling (v6)
Posted by Dmitriy Monakhov on Mon, 12 Mar 2007 08:55:14 GMT
View Forum Message <> Reply to Message

Nick Piggin <npiggin@suse.de> writes:

> On Mon, Mar 12, 2007 at 10:58:10AM +0300, Dmitriy Monakhov wrote:
>> I realy don't want to be annoying by sending this patcheset over and over
>> again, i just want the issue to be solved. If anyone think this solution
>> is realy cappy, please comment what exectly is bad. Thank you.
>
> If you don't get feedback, then you have to keep posting. If you still
> don't get feedback, try cc'ing a few more lists (eg. linux-fsdevel).
>
>> Changes:
>> - patch was split in two patches.
>> - comments added. I think now it is clearly describe things.
>> - patch prepared against 2.6.20-mm3
>>
>> How this patch tested:
>> - fsstress test.
>> - manual direct_io tests.
>>
>> LOG:
>> - Trim off blocks after generic_file_direct_write() has failed.
>> - Update out of date comments about direct_io locking rules.
>
> It can be nice to expand on what the problem was, and how you fixed it...
> but I guess you do quite a good job in the C comments.
If generic_file_direct_write() has fail (ENOSPC condition) inside
__generic_file_aio_write_nolock() it may have instantiated
a few blocks outside i_size. And fsck will complain about wrong i_size
(ext2, ext3 and reiserfs interpret i_size and biggest block difference as error),
after fsck will fix error i_size will be increased to the biggest block,
but this blocks contain gurbage from previous write attempt, this is not
information leak, but its silence file data corruption. This issue affect
fs regardless the values of blocksize or pagesize.
We need truncate any block beyond i_size after write have failed , do in simular
generic_file_buffered_write() error path.
TEST_CASE:
open("/mnt/test/BIG_FILE", O_WRONLY|O_CREAT|O_DIRECT, 0666) = 3

write(3, "aaaaaaaaaaaaaaa"..., 104857600) = -1 ENOSPC (No space left on device)

#stat /mnt/test/BIG_FILE
  File: `/mnt/test/BIG_FILE'
  Size: 0   Blocks: 110896   IO Block: 1024   regular empty file
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<^^^^^^^^file size is less than biggest block idx

Device: fe07h/65031d   Inode: 14   Links: 1
Access: (0644/-rw-r--r--) Uid: (0/   root)   Gid: (0/   root)
Access: 2007-01-24 20:03:38.000000000 +0300
Modify: 2007-01-24 20:03:38.000000000 +0300
Change: 2007-01-24 20:03:39.000000000 +0300

#fsck.ext3 -f /dev/VG/test
e2fsck 1.39 (29-May-2006)
Pass 1: Checking inodes, blocks, and sizes
Inode 14, i_size is 0, should be 56556544. Fix<y>? yes
Pass 2: Checking directory structure
....
>
>>
>> Signed-off-by: Monakhov Dmitriy <dmonakhov@openvz.org>
>> ---
>> mm/filemap.c | 32 ++++++++++++++++++++++++++++----
>> 1 files changed, 28 insertions(+), 4 deletions(-)
>>
>> diff --git a/mm/filemap.c b/mm/filemap.c
>> index 0aadf5f..8959ae3 100644
>> --- a/mm/filemap.c
>> +++ b/mm/filemap.c
>> @@ -1925,8 +1925,9 @@ generic_file_direct_write(struct kiocb *iocb, const struct iovec *iov,
>>   /*
>>   * Sync the fs metadata but not the minor inode changes and
>>   * of course not the data as we did direct DMA for the IO.
>> - * i_mutex is held, which protects generic_osync_inode() from
>> - * livelocking. AIO O_DIRECT ops attempt to sync metadata here.
>> + * i_mutex may not being held, if so some specific locking
>> + * ordering must protect generic_osync_inode() from livelocking.
>> + * AIO O_DIRECT ops attempt to sync metadata here.
>>   */
>
> This wasn't exactly clear to me. Did you mean:
>
> "may be held, which protects generic_osync_inode() from livelocking. If it
>   is not held, then the filesystem must prevent this livelock"?
Yep.. my english is not realy good :(
>
>>   if ((written >= 0 || written == -EIOCBQUEUED) &&

```
>>       ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
>> @@ -2240,6 +2241,29 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct iovec
*iov,
>>    mutex_lock(&inode->i_mutex);
>>    ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
>>      &iocb->ki_pos);
>> + /*
>> +  * If __generic_file_aio_write_nolock has failed.
>> +  * This may happen because of:
>> +  * 1) Bad segment found (failed before actual write attempt)
>> +  * 2) Segments are good, but actual write operation failed
>> +  *    and may have instantiated a few blocks outside i_size.
>> +  *   a) in case of buffered write these blocks was already
>> +  *    trimmed by generic_file_buffered_write()
>> +  *   b) in case of O_DIRECT these blocks weren't trimmed yet.
>> +  *
>> +  * In case of (2b) these blocks have to be trimmed off again.
>> +  */
>> + if (unlikely( ret < 0 && file->f_flags & O_DIRECT)) {
>> +  unsigned long nr_segs_avail = nr_segs;
>> +  size_t count = 0;
>> +  if (!generic_segment_checks(iov, &nr_segs_avail, &count,
>> +    VERIFY_READ)) {
>> +   /*It is (2b) case, because segments are good*/
>> +   loff_t isize = i_size_read(inode);
>> +   if (pos + count > isize)
>> +    vmtruncate(inode, isize);
>> +  }
>> + }
>
> OK, but wouldn't this be better to be done in the actual direct IO
> functions themselves? Thus you could be sure that you have the 2b case,
> and the code would be less fragile to something changing?
Ohh, We can't just call vmtruncate() after generic_file_direct_write()
failure while __generic_file_aio_write_nolock() becase where is no guarantee
what i_mutex held. In fact all existing fs always invoke
__generic_file_aio_write_nolock() with i_mutex held in case of S_ISREG files,
but this was't explicitly demanded and documented. I've proposed to do it in
previous versions of this patch, because it this just document current state
of affairs, but David Chinner wasn't agree with it.
>
> And a minor nit: extra space after "if (unlikely("
>
>
>>    mutex_unlock(&inode->i_mutex);
>>
>>    if (ret > 0 && ((file->f_flags & O_SYNC) || IS_SYNC(inode))) {
>> @@ -2254,8 +2278,8 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct iovec
```

*iov,
>> EXPORT_SYMBOL(generic_file_aio_write);
>>
>> /*
>> - * Called under i_mutex for writes to S_ISREG files.   Returns -EIO if something
>> - * went wrong during pagecache shootdown.
>> + * May be called without i_mutex for writes to S_ISREG files.
>> + * Returns -EIO if something went wrong during pagecache shootdown.
>>   */
>
> These comments updates are for DIO_OWN_LOCKING, right? In that case, you
> should mention that.

Subject: Re: [PATCH 2/2] mm: incorrect direct io error handling (v6)
Posted by Nick Piggin on Mon, 12 Mar 2007 09:09:17 GMT
View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 11:55:30AM +0300, Dmitriy Monakhov wrote:
> Nick Piggin <npiggin@suse.de> writes:
>
> > On Mon, Mar 12, 2007 at 10:58:10AM +0300, Dmitriy Monakhov wrote:

> >> @@ -2240,6 +2241,29 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct
iovec *iov,
> >>   mutex_lock(&inode->i_mutex);
> >>   ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
> >>   &iocb->ki_pos);
> >> + /*
> >> + * If __generic_file_aio_write_nolock has failed.
> >> + * This may happen because of:
> >> + * 1) Bad segment found (failed before actual write attempt)
> >> + * 2) Segments are good, but actual write operation failed
> >> + *   and may have instantiated a few blocks outside i_size.
> >> + *   a) in case of buffered write these blocks was already
> >> + *   trimmed by generic_file_buffered_write()
> >> + *   b) in case of O_DIRECT these blocks weren't trimmed yet.
> >> + *
> >> + * In case of (2b) these blocks have to be trimmed off again.
> >> + */
> >> + if (unlikely(ret < 0 && file->f_flags & O_DIRECT)) {
> >> + unsigned long nr_segs_avail = nr_segs;
> >> + size_t count = 0;
> >> + if (!generic_segment_checks(iov, &nr_segs_avail, &count,
> >> +   VERIFY_READ)) {
> >> +   /*It is (2b) case, because segments are good*/
> >> +   loff_t isize = i_size_read(inode);
> >> +   if (pos + count > isize)

> >> +   vmtruncate(inode, isize);
> >> + }
> >> + }
> >
> > OK, but wouldn't this be better to be done in the actual direct IO
> > functions themselves? Thus you could be sure that you have the 2b case,
> > and the code would be less fragile to something changing?
> Ohh, We can't just call vmtruncate() after generic_file_direct_write()
> failure while __generic_file_aio_write_nolock() becase where is no guarantee
> what i_mutex held. In fact all existing fs always invoke
> __generic_file_aio_write_nolock() with i_mutex held in case of S_ISREG files,
> but this was't explicitly demanded and documented. I've proposed to do it in
> previous versions of this patch, because it this just document current state
> of affairs, but David Chinner wasn't agree with it.

It seemed like it was documented in the comments that you altered in this
patch...

How would such a filesystem that did not hold i_mutex propose to fix the
problem?

The burden should be on those filesystems that might not want to hold
i_mutex here, to solve the problem nicely, rather than generic code to take
this ugly code.

Subject: Re: [PATCH 2/2] mm: incorrect direct io error handling (v6)
Posted by Dmitriy Monakhov on Mon, 12 Mar 2007 09:22:50 GMT
View Forum Message <> Reply to Message

Nick Piggin <npiggin@suse.de> writes:

> On Mon, Mar 12, 2007 at 11:55:30AM +0300, Dmitriy Monakhov wrote:
>> Nick Piggin <npiggin@suse.de> writes:
>>
>> > On Mon, Mar 12, 2007 at 10:58:10AM +0300, Dmitriy Monakhov wrote:
>
>> >> @@ -2240,6 +2241,29 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct
iovec *iov,
>> >>   mutex_lock(&inode->i_mutex);
>> >>   ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
>> >>   &iocb->ki_pos);
>> >> + /*
>> >> + * If __generic_file_aio_write_nolock has failed.
>> >> + * This may happen because of:
>> >> + * 1) Bad segment found (failed before actual write attempt)
>> >> + * 2) Segments are good, but actual write operation failed
>> >> + *   and may have instantiated a few blocks outside i_size.

>> >> + *   a) in case of buffered write these blocks was already
>> >> + *   trimmed by generic_file_buffered_write()
>> >> + *   b) in case of O_DIRECT these blocks weren't trimmed yet.
>> >> + *
>> >> + * In case of (2b) these blocks have to be trimmed off again.
>> >> + */
>> >> + if (unlikely(ret < 0 && file->f_flags & O_DIRECT)) {
>> >> + unsigned long nr_segs_avail = nr_segs;
>> >> + size_t count = 0;
>> >> + if (!generic_segment_checks(iov, &nr_segs_avail, &count,
>> >> +   VERIFY_READ)) {
>> >> +   /*It is (2b) case, because segments are good*/
>> >> +   loff_t isize = i_size_read(inode);
>> >> +   if (pos + count > isize)
>> >> +   vmtruncate(inode, isize);
>> >> + }
>> >> + }
>> >>
>> > OK, but wouldn't this be better to be done in the actual direct IO
>> > functions themselves? Thus you could be sure that you have the 2b case,
>> > and the code would be less fragile to something changing?
>> Ohh, We can't just call vmtruncate() after generic_file_direct_write()
>> failure while __generic_file_aio_write_nolock() becase where is no guarantee
>> what i_mutex held. In fact all existing fs always invoke
>> __generic_file_aio_write_nolock() with i_mutex held in case of S_ISREG files,
>> but this was't explicitly demanded and documented. I've proposed to do it in
>> previous versions of this patch, because it this just document current state
>> of affairs, but David Chinner wasn't agree with it.
>
> It seemed like it was documented in the comments that you altered in this
> patch...
>
> How would such a filesystem that did not hold i_mutex propose to fix the
> problem?
>
> The burden should be on those filesystems that might not want to hold
> i_mutex here, to solve the problem nicely, rather than generic code to take
> this ugly code.
Ok then what do you think about this version http://lkml.org/lkml/2006/12/18/103
witch was posted almost month ago :)
>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

Subject: Re: [PATCH 2/2] mm: incorrect direct io error handling (v6)
Posted by Nick Piggin on Mon, 12 Mar 2007 12:14:47 GMT
View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 12:23:00PM +0300, Dmitriy Monakhov wrote:
> Nick Piggin <npiggin@suse.de> writes:
>
> > On Mon, Mar 12, 2007 at 11:55:30AM +0300, Dmitriy Monakhov wrote:
> >> Nick Piggin <npiggin@suse.de> writes:
> >>
> >> > On Mon, Mar 12, 2007 at 10:58:10AM +0300, Dmitriy Monakhov wrote:
> > >
> >> >> @@ -2240,6 +2241,29 @@ ssize_t generic_file_aio_write(struct kiocb *iocb, const struct
iovec *iov,
> >> >> mutex_lock(&inode->i_mutex);
> >> >> ret = __generic_file_aio_write_nolock(iocb, iov, nr_segs,
> >> >> &iocb->ki_pos);
> >> >> + /*
> >> >> + * If __generic_file_aio_write_nolock has failed.
> >> >> + * This may happen because of:
> >> >> + * 1) Bad segment found (failed before actual write attempt)
> >> >> + * 2) Segments are good, but actual write operation failed
> >> >> + * and may have instantiated a few blocks outside i_size.
> >> >> + * a) in case of buffered write these blocks was already
> >> >> + * trimmed by generic_file_buffered_write()
> >> >> + * b) in case of O_DIRECT these blocks weren't trimmed yet.
> >> >> + *
> >> >> + * In case of (2b) these blocks have to be trimmed off again.
> >> >> + */
> >> >> + if (unlikely(ret < 0 && file->f_flags & O_DIRECT)) {
> >> >> + unsigned long nr_segs_avail = nr_segs;
> >> >> + size_t count = 0;
> >> >> + if (!generic_segment_checks(iov, &nr_segs_avail, &count,
> >> >> + VERIFY_READ)) {
> >> >> + /*It is (2b) case, because segments are good*/
> >> >> + loff_t isize = i_size_read(inode);
> >> >> + if (pos + count > isize)
> >> >> + vmtruncate(inode, isize);
> >> >> + }
> >> >> + }
> >> >
> >> > OK, but wouldn't this be better to be done in the actual direct IO
> >> > functions themselves? Thus you could be sure that you have the 2b case,
> >> > and the code would be less fragile to something changing?
> >> Ohh, We can't just call vmtruncate() after generic_file_direct_write()
> >> failure while __generic_file_aio_write_nolock() becase where is no guarantee
> >> what i_mutex held. In fact all existing fs always invoke
> >> __generic_file_aio_write_nolock() with i_mutex held in case of S_ISREG files,
> >> but this was't explicitly demanded and documented. I've proposed to do it in

> >> previous versions of this patch, because it this just document current state
> >> of affairs, but David Chinner wasn't agree with it.
> >
> > It seemed like it was documented in the comments that you altered in this
> > patch...
> >
> > How would such a filesystem that did not hold i_mutex propose to fix the
> > problem?
> >
> > The burden should be on those filesystems that might not want to hold
> > i_mutex here, to solve the problem nicely, rather than generic code to take
> > this ugly code.
> Ok then what do you think about this version http://lkml.org/lkml/2006/12/18/103
> witch was posted almost month ago :)

That seems better, but people might take issue with the fact that it has
to make the check for S_ISREG files. I don't know... people with more
knowledge of the vfs+fs side of things might have better input.
