Subject: [Q] missing unused dentry in prune_dcache()?
Posted by vaverin on Wed, 25 Oct 2006 12:30:51 GMT
View Forum Message <> Reply to Message

Hello folks,

I would like to ask you clarify me one question in the the following patch:
 http://linux.bkbits.net:8080/linux-2.6/gnupatch@449b144ecSF1 rYskg3q-SeR2vf88zg
ChangeSet
2006/06/22 15:05:57-07:00 neilb@suse.de
[PATCH] Fix dcache race during umount

If prune_dcache finds a dentry that it cannot free, it leaves it where it
is (at the tail of the list) and exits, on the assumption that some other
thread will be removing that dentry soon.

However as far as I see this comment is not correct: when we cannot take
s_umount rw_semaphore (for example because it was taken in do_remount) this
dentry is already extracted from dentry_unused list and we do not add it into
the list again. Therefore dentry will not be found by prune_dcache() and
shrink_dcache_sb() and will leave in memory very long time until the partition
will be unmounted.

Am I probably err?

The patch adds this dentry into tail of the dentry_unused list.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- linux-2.6.19-rc3/fs/dcache.c.prdch 2006-10-25 16:09:19.000000000 +0400
+++ linux-2.6.19-rc3/fs/dcache.c 2006-10-25 16:08:20.000000000 +0400
@@ -477,6 +477,8 @@ static void prune_dcache(int count, stru
    }
    up_read(s_umount);
   }
+   list_add_tail(&dentry->d_lru, &dentry_unused);
+  dentry_stat.nr_unused++;
   spin_unlock(&dentry->d_lock);
   /* Cannot remove the first dentry, and it isn't appropriate
    * to move it to the head of the list, so give up, and try
```

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Wed, 25 Oct 2006 13:51:04 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> # If prune_dcache finds a dentry that it cannot free, it leaves it where it
> # is (at the tail of the list) and exits, on the assumption that some other
> # thread will be removing that dentry soon.
>
> However as far as I see this comment is not correct: when we cannot take
> s_umount rw_semaphore (for example because it was taken in do_remount) this
> dentry is already extracted from dentry_unused list and we do not add it into
> the list again.

You would seem to be correct.

> Therefore dentry will not be found by prune_dcache() and shrink_dcache_sb()
> and will leave in memory very long time until the partition will be
> unmounted.

And here too:-/

> Am I probably err?

Unfortunately not. I wonder if remount should be getting a writelock on the
s_umount sem, but I don't see why not. grab_super() also gets a writelock on
it, and so that could cause problems too.

shrink_dcache_for_umount_subtree() doesn't care because it doesn't scan the
dcache_unused list, but as you say, other things are affected.

> The patch adds this dentry into tail of the dentry_unused list.

I think that's reasonable. I wonder if we can avoid removing it from the list
in the first place, but I suspect it's less optimal.

Acked-By: David Howells <dhowells@redhat.com>

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by vaverin on Wed, 25 Oct 2006 13:58:40 GMT
View Forum Message <> Reply to Message

David Howells wrote:
> Vasily Averin <vvs@sw.ru> wrote:
>> The patch adds this dentry into tail of the dentry_unused list.
>
> I think that's reasonable. I wonder if we can avoid removing it from the list
> in the first place, but I suspect it's less optimal.

Could you please explain this place in details, I do not understand why tail of
the list is better than head.
Also I do not understand why we should go to out in this case. Why we cannot use

next dentry in the list instead?

> Acked-By: David Howells <dhowells@redhat.com>

Thank you,
Vasily Averin

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Wed, 25 Oct 2006 14:23:29 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> >> The patch adds this dentry into tail of the dentry_unused list.
> >
> > I think that's reasonable. I wonder if we can avoid removing it from the
> > list in the first place, but I suspect it's less optimal.
>
> Could you please explain this place in details, I do not understand why tail
> of the list is better than head. Also I do not understand why we should go
> to out in this case. Why we cannot use next dentry in the list instead?

I meant adding it back into the list is reasonable; I didn't actually consider where you were adding it back.

However, you've just raised a good point: moving a dentry to the head of the list is what happens when we see it's been referenced recently. The most recently used entry is at the head and the least at the tail.

If you look at the list scan algorithm, you can see it works from the tail towards the head.

We could re-insert the dentry at the position from which we removed it, but that would mean retaining a pointer to one of its neighbouring dentries. We can do that - it's just stack space after all...

I think the main point though, is generally that the superblock that owns the dentry is very busy in some critical way; either it's being unmounted (in which case we may as well put the dentry at the MRU end), or it's being remounted (in which case, it's probably best putting it back where we found it).

You also have to consider how often superblocks are going to be unmounted or remounted vs the dcache being pruned...

There is also grab_super(), but that's only used by sget() when a superblock is being mounted again - also unlikely.

So, given that the three ops that affect this are very unlikely to be happening
at any one time, it's probably worth just slapping it at the head and ignoring
it. The main thing is that it's reinserted somewhere.

Hope that helps...

David

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by vaverin on Thu, 26 Oct 2006 11:36:53 GMT
View Forum Message <> Reply to Message

Hello David

David Howells wrote:
> Vasily Averin <vvs@sw.ru> wrote:
>>>> The patch adds this dentry into tail of the dentry_unused list.
>>> I think that's reasonable. I wonder if we can avoid removing it from the
>>> list in the first place, but I suspect it's less optimal.
>> Could you please explain this place in details, I do not understand why tail
>> of the list is better than head. Also I do not understand why we should go
>> to out in this case. Why we cannot use next dentry in the list instead?
>
> I meant adding it back into the list is reasonable; I didn't actually consider
> where you were adding it back.
>
> So, given that the three ops that affect this are very unlikely to be happening
> at any one time, it's probably worth just slapping it at the head and ignoring
> it. The main thing is that it's reinserted somewhere.

I don't like to insert this dentry into tail of list because of it prevents
shrink_dcache_memory. It finds "skipped" dentry at the tail of the list, does
not free it and goes to out without any progress.

Therefore I've removed break of cycle and insert this dentry to head of the
list. Theoretically it can lead to the second using of the same dentry, however
I do not think that it is a big problem.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- linux-2.6.19-rc3/fs/dcache.c.prdch 2006-10-25 16:09:19.000000000 +0400
+++ linux-2.6.19-rc3/fs/dcache.c 2006-10-26 15:14:51.000000000 +0400
@@ -478,11 +478,9 @@ static void prune_dcache(int count, stru
    up_read(s_umount);
   }
   spin_unlock(&dentry->d_lock);
```

```
-  /* Cannot remove the first dentry, and it isn't appropriate
-   * to move it to the head of the list, so give up, and try
-   * later
-   */
-  break;
+  /* Inserting dentry to tail of the list leads to cycle */
+   list_add(&dentry->d_lru, &dentry_unused);
+  dentry_stat.nr_unused++;
  }
  spin_unlock(&dcache_lock);
 }
```

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by vaverin on Thu, 26 Oct 2006 11:49:25 GMT
View Forum Message <> Reply to Message

Hello David,

I've noticed one more minor issue in your patch: in
shrink_dcache_for_umount_subtree() function you decrement dentry_stat.nr_dentry
without dcache_lock.

Thank you,
 Vasily Averin

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Thu, 26 Oct 2006 12:33:58 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> I've noticed one more minor issue in your patch: in
> shrink_dcache_for_umount_subtree() function you decrement
> dentry_stat.nr_dentry without dcache_lock.

Hmmm... that's a very good point:-/ I wonder if I can batch them.

David

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Thu, 26 Oct 2006 13:23:35 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> I've noticed one more minor issue in your patch: in
> shrink_dcache_for_umount_subtree() function you decrement
> dentry_stat.nr_dentry without dcache_lock.

How about the attached patch?

David

VFS: Fix an error in unused dentry counting

From: David Howells <dhowells@redhat.com>

Fix an error in unused dentry counting in shrink_dcache_for_umount_subtree() in
which the count is modified without the dcache_lock held.

Signed-Off-By: David Howells <dhowells@redhat.com>
---

 fs/dcache.c | 22 ++++++++++++++++++-------
 1 files changed, 15 insertions(+), 7 deletions(-)

```
diff --git a/fs/dcache.c b/fs/dcache.c
index a1ff91e..eab1bf4 100644
--- a/fs/dcache.c
+++ b/fs/dcache.c
@@ -553,16 +553,17 @@ repeat:
  * - see the comments on shrink_dcache_for_umount() for a description of the
  *   locking
  */
-static void shrink_dcache_for_umount_subtree(struct dentry *dentry)
+static unsigned shrink_dcache_for_umount_subtree(struct dentry *dentry)
 {
  struct dentry *parent;
+ unsigned detached = 0;

  BUG_ON(!IS_ROOT(dentry));

  /* detach this root from the system */
  spin_lock(&dcache_lock);
  if (!list_empty(&dentry->d_lru)) {
-  dentry_stat.nr_unused--;
+  detached++;
   list_del_init(&dentry->d_lru);
  }
  __d_drop(dentry);
@@ -579,7 +580,7 @@ static void shrink_dcache_for_umount_sub
    list_for_each_entry(loop, &dentry->d_subdirs,
```

```
          d_u.d_child) {
      if (!list_empty(&loop->d_lru)) {
-      dentry_stat.nr_unused--;
+      detached++;
       list_del_init(&loop->d_lru);
      }

@@ -620,7 +621,7 @@ static void shrink_dcache_for_umount_sub
      atomic_dec(&parent->d_count);

     list_del(&dentry->d_u.d_child);
-    dentry_stat.nr_dentry--; /* For d_free, below */
+    detached++; /* For d_free, below */

     inode = dentry->d_inode;
     if (inode) {
@@ -638,7 +639,7 @@ static void shrink_dcache_for_umount_sub
      * otherwise we ascend to the parent and move to the
      * next sibling if there is one */
     if (!parent)
-     return;
+     return detached;

     dentry = parent;

@@ -663,6 +664,7 @@ static void shrink_dcache_for_umount_sub
 void shrink_dcache_for_umount(struct super_block *sb)
 {
   struct dentry *dentry;
+ unsigned detached = 0;

   if (down_read_trylock(&sb->s_umount))
    BUG();
@@ -670,11 +672,17 @@ void shrink_dcache_for_umount(struct sup
   dentry = sb->s_root;
   sb->s_root = NULL;
   atomic_dec(&dentry->d_count);
- shrink_dcache_for_umount_subtree(dentry);
+ detached = shrink_dcache_for_umount_subtree(dentry);

   while (!hlist_empty(&sb->s_anon)) {
    dentry = hlist_entry(sb->s_anon.first, struct dentry, d_hash);
-   shrink_dcache_for_umount_subtree(dentry);
+   detached += shrink_dcache_for_umount_subtree(dentry);
+ }
+
+ if (detached) {
+   spin_lock(&dcache_lock);
```

```
+  dentry_stat.nr_unused -= detached;
+  spin_unlock(&dcache_lock);
  }
 }
```

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Thu, 26 Oct 2006 13:25:29 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> Therefore I've removed break of cycle and insert this dentry to head of the
> list. Theoretically it can lead to the second using of the same dentry,
> however I do not think that it is a big problem.

Hmmm... Or maybe it could be a problem. If whenever we find the dentry we stick it back on the head of the list, this could be a problem as we're traversing the list from tail to head.

David

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by vaverin on Thu, 26 Oct 2006 13:58:20 GMT
View Forum Message <> Reply to Message

David Howells wrote:
> Vasily Averin <vvs@sw.ru> wrote:
>
>> I've noticed one more minor issue in your patch: in
>> shrink_dcache_for_umount_subtree() function you decrement
>> dentry_stat.nr_dentry without dcache_lock.
>
> How about the attached patch?

I would note that in first two hunks you have decremented dentry_stat.nr_unused correctly, under dcache_lock. Probably it's better to count freed dentries only in third case and corrects dentry_stat.nr_unused value inside shrink_dcache_for_umount_subtree() function before return.

Thank you,
 Vasily Averin
> David
> ---
> VFS: Fix an error in unused dentry counting
>

> From: David Howells <dhowells@redhat.com>
>
> Fix an error in unused dentry counting in shrink_dcache_for_umount_subtree() in
> which the count is modified without the dcache_lock held.
>
> Signed-Off-By: David Howells <dhowells@redhat.com>
> ---
>
> fs/dcache.c | 22 +++++++++++++++++-------
> 1 files changed, 15 insertions(+), 7 deletions(-)
>
> diff --git a/fs/dcache.c b/fs/dcache.c
> index a1ff91e..eab1bf4 100644
> --- a/fs/dcache.c
> +++ b/fs/dcache.c
> @@ -553,16 +553,17 @@ repeat:
>   * - see the comments on shrink_dcache_for_umount() for a description of the
>   *   locking
>   */
> -static void shrink_dcache_for_umount_subtree(struct dentry *dentry)
> +static unsigned shrink_dcache_for_umount_subtree(struct dentry *dentry)
> {
>   struct dentry *parent;
> + unsigned detached = 0;
>
>   BUG_ON(!IS_ROOT(dentry));
>
>   /* detach this root from the system */
>   spin_lock(&dcache_lock);
>   if (!list_empty(&dentry->d_lru)) {
> - dentry_stat.nr_unused--;
> + detached++;
>   list_del_init(&dentry->d_lru);
>   }
>   __d_drop(dentry);
> @@ -579,7 +580,7 @@ static void shrink_dcache_for_umount_sub
>   list_for_each_entry(loop, &dentry->d_subdirs,
>   d_u.d_child) {
>   if (!list_empty(&loop->d_lru)) {
> -   dentry_stat.nr_unused--;
> +   detached++;
>   list_del_init(&loop->d_lru);
>   }
>
> @@ -620,7 +621,7 @@ static void shrink_dcache_for_umount_sub
>   atomic_dec(&parent->d_count);
>
>   list_del(&dentry->d_u.d_child);

```
> -   dentry_stat.nr_dentry--; /* For d_free, below */
> +   detached++; /* For d_free, below */
>
>     inode = dentry->d_inode;
>     if (inode) {
> @@ -638,7 +639,7 @@ static void shrink_dcache_for_umount_sub
>      * otherwise we ascend to the parent and move to the
>      * next sibling if there is one */
>     if (!parent)
> -    return;
> +    return detached;
>
>     dentry = parent;
>
> @@ -663,6 +664,7 @@ static void shrink_dcache_for_umount_sub
>  void shrink_dcache_for_umount(struct super_block *sb)
>  {
>   struct dentry *dentry;
> + unsigned detached = 0;
>
>   if (down_read_trylock(&sb->s_umount))
>    BUG();
> @@ -670,11 +672,17 @@ void shrink_dcache_for_umount(struct sup
>   dentry = sb->s_root;
>   sb->s_root = NULL;
>   atomic_dec(&dentry->d_count);
> - shrink_dcache_for_umount_subtree(dentry);
> + detached = shrink_dcache_for_umount_subtree(dentry);
>
>   while (!hlist_empty(&sb->s_anon)) {
>    dentry = hlist_entry(sb->s_anon.first, struct dentry, d_hash);
> -  shrink_dcache_for_umount_subtree(dentry);
> +  detached += shrink_dcache_for_umount_subtree(dentry);
> + }
> +
> + if (detached) {
> +  spin_lock(&dcache_lock);
> +  dentry_stat.nr_unused -= detached;
> +  spin_unlock(&dcache_lock);
>   }
>  }
>
>
```

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Thu, 26 Oct 2006 14:07:32 GMT

Vasily Averin <vvs@sw.ru> wrote:

> I would note that in first two hunks you have decremented
> dentry_stat.nr_unused correctly, under dcache_lock. Probably it's better to
> count freed dentries only in third case and corrects dentry_stat.nr_unused
> value inside shrink_dcache_for_umount_subtree() function before return.

Maybe, maybe not. This way we only touch the dentry_stat cacheline once, if
at all. I'm not sure it'd make much difference though.

David

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by vaverin on Fri, 27 Oct 2006 06:32:35 GMT

David Howells wrote:
> Vasily Averin <vvs@sw.ru> wrote:
>
>> I've noticed one more minor issue in your patch: in
>> shrink_dcache_for_umount_subtree() function you decrement
>> dentry_stat.nr_dentry without dcache_lock.
>
> How about the attached patch?
I'm sorry, but your patch is wrong:
you have mixed calculation of 2 variables:
dentry_stat.nr_unused -- were correct, it was decremented under dcache_lock.
dentry_stat.nr_dentry -- were incorrect, it was decremented without dcache_lock.

You should correct dentry_stat.nr_dentry, but instead you broke calculation of
dentry_stat.nr_unused.

I've fixed this issue by following patch.

Thank you,
Vasily Averin

VFS: Fix an error in dentry_stat.nr_dentry counting

From: Vasily Averin <vvs@sw.ru>

Fix an error in dentry_stat.nr_dentry counting in
shrink_dcache_for_umount_subtree() in which the count is modified without the
dcache_lock held.

Signed-Off-By: Vasily Averin <vvs@sw.ru>

```
--- linux-2.6.19-rc3/fs/dcache.c.nrdntr 2006-10-26 15:14:51.000000000 +0400
+++ linux-2.6.19-rc3/fs/dcache.c 2006-10-27 10:24:07.000000000 +0400
@@ -554,6 +554,7 @@ repeat:
 static void shrink_dcache_for_umount_subtree(struct dentry *dentry)
 {
  struct dentry *parent;
+ unsigned detached = 0;

  BUG_ON(!IS_ROOT(dentry));

@@ -618,7 +619,7 @@ static void shrink_dcache_for_umount_sub
     atomic_dec(&parent->d_count);

    list_del(&dentry->d_u.d_child);
-   dentry_stat.nr_dentry--; /* For d_free, below */
+   detached++;

    inode = dentry->d_inode;
    if (inode) {
@@ -635,8 +636,8 @@ static void shrink_dcache_for_umount_sub
    /* finished when we fall off the top of the tree,
     * otherwise we ascend to the parent and move to the
     * next sibling if there is one */
-   if (!parent)
-    return;
+   if (!parent)
+    goto out;

    dentry = parent;

@@ -645,6 +646,11 @@ static void shrink_dcache_for_umount_sub
   dentry = list_entry(dentry->d_subdirs.next,
         struct dentry, d_u.d_child);
  }
+out:
+ /* several dentries were freed, need to correct nr_dentry */
+ spin_lock(&dcache_lock);
+ dentry_stat.nr_dentry -= detached;
+ spin_unlock(&dcache_lock);
 }

 /*
```

Subject: Re: [Q] missing unused dentry in prune_dcache()?

Vasily Averin wrote:
> David Howells wrote:
>> Vasily Averin <vvs@sw.ru> wrote:
>>
>>> I've noticed one more minor issue in your patch: in
>>> shrink_dcache_for_umount_subtree() function you decrement
>>> dentry_stat.nr_dentry without dcache_lock.
>> How about the attached patch?
> I'm sorry, but your patch is wrong:
> you have mixed calculation of 2 variables:
> dentry_stat.nr_unused -- were correct, it was decremented under dcache_lock.
> dentry_stat.nr_dentry -- were incorrect, it was decremented without dcache_lock.
>
> You should correct dentry_stat.nr_dentry, but instead you broke calculation of
> dentry_stat.nr_unused.
>
> I've fixed this issue by following patch.
corrected version, extra space were removed

Thank you,
 Vasily Averin

VFS: Fix an error in dentry_stat.nr_dentry counting

From: Vasily Averin <vvs@sw.ru>

Fix an error in dentry_stat.nr_dentry counting in
shrink_dcache_for_umount_subtree() in which the count is modified without the
dcache_lock held.

Signed-Off-By: Vasily Averin <vvs@sw.ru>

```
--- linux-2.6.19-rc3/fs/dcache.c.nrdntr 2006-10-26 15:14:51.000000000 +0400
+++ linux-2.6.19-rc3/fs/dcache.c 2006-10-27 10:45:11.000000000 +0400
@@ -554,6 +554,7 @@ repeat:
 static void shrink_dcache_for_umount_subtree(struct dentry *dentry)
 {
  struct dentry *parent;
+ unsigned detached = 0;

  BUG_ON(!IS_ROOT(dentry));

@@ -618,7 +619,7 @@ static void shrink_dcache_for_umount_sub
     atomic_dec(&parent->d_count);
```

```
    list_del(&dentry->d_u.d_child);
-   dentry_stat.nr_dentry--; /* For d_free, below */
+   detached++;

    inode = dentry->d_inode;
    if (inode) {
@@ -636,7 +637,7 @@ static void shrink_dcache_for_umount_sub
     * otherwise we ascend to the parent and move to the
     * next sibling if there is one */
    if (!parent)
-    return;
+    goto out;

    dentry = parent;

@@ -645,6 +646,11 @@ static void shrink_dcache_for_umount_sub
   dentry = list_entry(dentry->d_subdirs.next,
         struct dentry, d_u.d_child);
  }
+out:
+ /* several dentries were freed, need to correct nr_dentry */
+ spin_lock(&dcache_lock);
+ dentry_stat.nr_dentry -= detached;
+ spin_unlock(&dcache_lock);
 }

 /*
```

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by vaverin on Fri, 27 Oct 2006 08:05:54 GMT
View Forum Message <> Reply to Message

David Howells wrote:
> Vasily Averin <vvs@sw.ru> wrote:
>
>> Therefore I've removed break of cycle and insert this dentry to head of the
>> list. Theoretically it can lead to the second using of the same dentry,
>> however I do not think that it is a big problem.
>
> Hmmm... Or maybe it could be a problem. If whenever we find the dentry we
> stick it back on the head of the list, this could be a problem as we're
> traversing the list from tail to head.

I still do not think that it could be a problem:
count argument of prune_dcache is 128 if prune_dcache is called from
shrink_dcache_memory() function and even lesser if prune_dcache is called from
shrink_dcache_parent() function. I think usually the size of unused_dentry list

(nr_usnused) is much greater and may be compared with these values only in case of very hard memory shortage. From my point of view it is very rare situation and I even not sure that it can really happen at all.

However even if this situation will happen I do not see here any seriously troubles: Yes, we will try to free the same dentries, much probably without success again, but why it is bad in case of hard memory shortage?

If my arguments are not convincing, I can protect second use of the same dentry: we can compare counter of the skipped dentries with nr_unused value.

And what the alternatives we have?
1) We can move this dentry to end of list? But it is bad because it will prevent shrink_dcache_memory().
2) we can move these dentries into some temporal list, and insert it back to unused_list later? But it is bad because of we can be rescheduled and drop dcache_lock and may confuse someone who will assume that these dentries are in unused_list.

Therefore I believe that my patch is optimal solution.

Thank you,
Vasily Averin

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Fri, 27 Oct 2006 09:36:37 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> > You should correct dentry_stat.nr_dentry, but instead you broke
> > calculation of dentry_stat.nr_unused.
> >
> > I've fixed this issue by following patch.
> corrected version, extra space were removed

Oops. You're right.

Acked-By: David Howells <dhowells@redhat.com>

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Fri, 27 Oct 2006 10:42:49 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> Therefore I believe that my patch is optimal solution.

I'm not sure that prune_dcache() is particularly optimal. If we're looking to prune for a specific superblock, it may scan most of the dentry_unused list several times, once for each dentry it eliminates.

Imagine the list with a million dentries on it. Imagine further that all the dentries you're trying to eliminate are up near the head end: you're going to have to scan most of the list several times unnecessarily; if you're asked to kill 128 dentries, you might wind up examining on the order of 100,000,000 dentries, 99% of which you scan 128 times.

I wonder if this could be improved by making the assumption that there won't be any entries inserted tailwards of where we've just looked. The problem is that if dcache_lock is dropped, we've no way of keeping track of the current position without inserting a marker into the list.

Now we could do the marker thing quite easily. We'd have to insert a dummy dcache entry, probably with d_sb pointing to some special location that is recognised as saying "that dentry is a marker".

We could do something like the attached patch, for example. Note that the patch compiles, but I haven't tested it. It also uses a big chunk of stack space for the marker. It ought to be safe enough with respect to the other functions that touch that list - all of those deal with specific dentries or look for dentries by superblock.

David

```
diff --git a/fs/dcache.c b/fs/dcache.c
index eab1bf4..a1cae74 100644
--- a/fs/dcache.c
+++ b/fs/dcache.c
@@ -395,18 +395,27 @@ static void prune_one_dentry(struct dent
  * This function may fail to free any resources if
  * all the dentries are in use.
  */
-
+
 static void prune_dcache(int count, struct super_block *sb)
 {
+ struct dentry marker = {
+  .d_sb = (struct super_block *) &prune_dcache,
+ };
+
+ struct list_head *tmp;
+
```

```
  spin_lock(&dcache_lock);
+ list_add_tail(&marker.d_lru, &dentry_unused);
+
  for (; count ; count--) {
   struct dentry *dentry;
-  struct list_head *tmp;
   struct rw_semaphore *s_umount;

   cond_resched_lock(&dcache_lock);

-  tmp = dentry_unused.prev;
+  tmp = marker.d_lru.prev;
+  list_del_init(&marker.d_lru);
+
   if (sb) {
    /* Try to find a dentry for this sb, but don't try
     * too hard, if they aren't near the tail they will
@@ -418,9 +427,18 @@ static void prune_dcache(int count, stru
     skip--;
     tmp = tmp->prev;
    }
+  } else {
+   /* We may not be the only pruner */
+   while (tmp != &dentry_unused) {
+    dentry = list_entry(tmp, struct dentry, d_lru);
+    if (dentry->d_sb !=
+        (struct super_block *) &prune_dcache)
+     break;
+   }
   }
   if (tmp == &dentry_unused)
    break;
+  list_add(&marker.d_lru, tmp);
   list_del_init(tmp);
   prefetch(dentry_unused.prev);
    dentry_stat.nr_unused--;
@@ -439,7 +457,7 @@ static void prune_dcache(int count, stru
   if (dentry->d_flags & DCACHE_REFERENCED) {
    dentry->d_flags &= ~DCACHE_REFERENCED;
-    list_add(&dentry->d_lru, &dentry_unused);
+    list_add(&dentry->d_lru, &marker.d_lru);
     dentry_stat.nr_unused++;
     spin_unlock(&dentry->d_lock);
    continue;
@@ -478,12 +496,10 @@ static void prune_dcache(int count, stru
    up_read(s_umount);
   }
```

```
   spin_unlock(&dentry->d_lock);
-  /* Cannot remove the first dentry, and it isn't appropriate
-   * to move it to the head of the list, so give up, and try
-   * later
-   */
-  break;
+  list_add(&dentry->d_lru, &marker.d_lru);
+  dentry_stat.nr_unused++;
  }
+ list_del(&marker.d_lru);
  spin_unlock(&dcache_lock);
 }
```

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by vaverin on Fri, 27 Oct 2006 11:50:59 GMT
View Forum Message <> Reply to Message

David,

David Howells wrote:
> Vasily Averin <vvs@sw.ru> wrote:
>> Therefore I believe that my patch is optimal solution.
> I'm not sure that prune_dcache() is particularly optimal.

I means that my patch is optimal for problem in subject. I would like to ask you
to approve it and we will go to next issue.

> If we're looking to
> prune for a specific superblock, it may scan most of the dentry_unused list
> several times, once for each dentry it eliminates.
>
> Imagine the list with a million dentries on it. Imagine further that all the
> dentries you're trying to eliminate are up near the head end: you're going to
> have to scan most of the list several times unnecessarily; if you're asked to
> kill 128 dentries, you might wind up examining on the order of 100,000,000
> dentries, 99% of which you scan 128 times.

I would note that we (Virtuozzo/OpenVZ team) have seen similar issue on praxis.
We have kernel that handles a few dozens Virtual servers, and each of them have
the several isolated filesystems. We have seen that umount (and remount) can
work very slowly, it was cycled inside shrink_dcache_sb() up to several hours
with taken s_umount semaphore.

We are trying to resolve this issue by using per-sb lru list. I'm preparing the
patch for 2.6.19-rc3 right now and going to send it soon.

thank you,

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Fri, 27 Oct 2006 12:11:21 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> > Vasily Averin <vvs@sw.ru> wrote:
> >> Therefore I believe that my patch is optimal solution.
> > I'm not sure that prune_dcache() is particularly optimal.
>
> I means that my patch is optimal for problem in subject.

I didn't say it wasn't.

> I would like to ask you to approve it and we will go to next issue.

I did ack it didn't I? I must fix my mail client so that it doesn't automatically remove my email address from the To/Cc fields when I'm replying to a message:-/

> We have seen that umount (and remount) can work very slowly, it was cycled
> inside shrink_dcache_sb() up to several hours with taken s_umount semaphore.

umount at least should be fixed as that should no longer use shrink_dcache_sb().

> We are trying to resolve this issue by using per-sb lru list. I'm preparing
> the patch for 2.6.19-rc3 right now and going to send it soon.

That sounds tricky; you have to check all your LRU lists to find the LRU dentry.

David

Subject: [PATCH 2.6.19-rc3] VFS: missing unused dentry in prune_dcache()
Posted by vaverin on Fri, 27 Oct 2006 13:42:03 GMT
View Forum Message <> Reply to Message

Andrew,

could you please use this patch instead of missing-unused-dentry-in-prune_dcache.patch

As far as I understand David Howells have not any objections

Thank you,
Vasily Averin

VFS: missing unused dentry in prune_dcache()

From: Vasily Averin <vvs@sw.ru>

If we cannot delete dentry we should insert it back to the dentry_unused list.
To prevent cycle and do not blocks prune_dcache() call from
shrink_dcache_memory() we adding this dentry to head of list.

Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- linux-2.6.19-rc3/fs/dcache.c.prdch 2006-10-25 16:09:19.000000000 +0400
+++ linux-2.6.19-rc3/fs/dcache.c 2006-10-26 15:14:51.000000000 +0400
@@ -478,11 +478,9 @@ static void prune_dcache(int count, stru
    up_read(s_umount);
   }
   spin_unlock(&dentry->d_lock);
-  /* Cannot remove the first dentry, and it isn't appropriate
-   * to move it to the head of the list, so give up, and try
-   * later
-   */
-  break;
+  /* Inserting dentry to tail of the list leads to cycle */
+   list_add(&dentry->d_lru, &dentry_unused);
+  dentry_stat.nr_unused++;
  }
  spin_unlock(&dcache_lock);
 }
```

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by vaverin on Fri, 27 Oct 2006 13:47:00 GMT
View Forum Message <> Reply to Message

David Howells wrote:
> Vasily Averin <vvs@sw.ru> wrote:
>> I would like to ask you to approve it and we will go to next issue.
>
> I did ack it didn't I? I must fix my mail client so that it doesn't
> automatically remove my email address from the To/Cc fields when I'm replying
> to a message:-/

To prevent any mistake I've resend this patch again to you and akpm@.

>> We have seen that umount (and remount) can work very slowly, it was cycled
>> inside shrink_dcache_sb() up to several hours with taken s_umount semaphore.
>
> umount at least should be fixed as that should no longer use
> shrink_dcache_sb().

Umount calls shrink_dcache_sb in "Special case for "unmounting" root".
Usually it happen only once, but in case OpenVZ it happens every time when any
Virtual server is stopped, each of them have own isolated root partition.

>> We are trying to resolve this issue by using per-sb lru list. I'm preparing
>> the patch for 2.6.19-rc3 right now and going to send it soon.

Please wait until my following email, it will be ready soon

thank you,
Vasily Averin

Subject: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by vaverin on Fri, 27 Oct 2006 14:05:50 GMT
View Forum Message <> Reply to Message

From: Vasily Averin <vvs@sw.ru>

Virtuozzo/OpenVZ linux kernel team has discovered that umount/remount can last
for hours looping in shrink_dcache_sb() without much successes. Since during
shrinking s_umount semaphore is taken lots of other unrelated operations like
sync can stop working until shrink finished.

It happens due to very long unused dentries list (>1 million of dentries),
so that it takes shrink_dcache_sb() longer then 1/HZ seconds to get to the
required dentry and after freeing this single dentry it reschedules and restarts.

The proposed fix prevents this issue by using per-sb dentry LRU list. It
provides very quickly search for the unused dentries for given super block thus
forcing shrinking always making good progress.

It was well tested on 2.6.9-based Virtuozzo/OpenVZ kernels, but the port to the
latest mainstream kernel is not tested.

Signed-off-by: Kirill Korotaev <dev@openvz.org>
Signed-off-by: Vasily Averin <vvs@sw.ru>

--- linux-2.6.19-rc3/fs/dcache.c.shrsb 2006-10-27 10:45:11.000000000 +0400
+++ linux-2.6.19-rc3/fs/dcache.c 2006-10-27 15:49:54.000000000 +0400
@@ -173,6 +173,7 @@ repeat:

```
    if (list_empty(&dentry->d_lru)) {
     dentry->d_flags |= DCACHE_REFERENCED;
     list_add(&dentry->d_lru, &dentry_unused);
+  list_add(&dentry->d_sb_lru, &dentry->d_sb->s_dentry_unused);
     dentry_stat.nr_unused++;
    }
   spin_unlock(&dentry->d_lock);
@@ -190,6 +191,7 @@ kill_it: {
    */
     if (!list_empty(&dentry->d_lru)) {
      list_del(&dentry->d_lru);
+   list_del(&dentry->d_sb_lru);
      dentry_stat.nr_unused--;
     }
     list_del(&dentry->d_u.d_child);
@@ -270,6 +272,7 @@ static inline struct dentry * __dget_loc
  if (!list_empty(&dentry->d_lru)) {
   dentry_stat.nr_unused--;
   list_del_init(&dentry->d_lru);
+  list_del_init(&dentry->d_sb_lru);
  }
  return dentry;
 }
@@ -398,6 +401,10 @@ static void prune_one_dentry(struct dent

 static void prune_dcache(int count, struct super_block *sb)
 {
+ struct list_head *lru_head;
+
+ lru_head = sb ? &sb->s_dentry_unused : &dentry_unused;
+
  spin_lock(&dcache_lock);
  for (; count ; count--) {
   struct dentry *dentry;
@@ -406,25 +413,16 @@ static void prune_dcache(int count, stru

   cond_resched_lock(&dcache_lock);

-  tmp = dentry_unused.prev;
-  if (sb) {
-   /* Try to find a dentry for this sb, but don't try
-    * too hard, if they aren't near the tail they will
-    * be moved down again soon
-    */
-   int skip = count;
-   while (skip && tmp != &dentry_unused &&
-       list_entry(tmp, struct dentry, d_lru)->d_sb != sb) {
-    skip--;
```

```
-    tmp = tmp->prev;
-   }
-  }
-  if (tmp == &dentry_unused)
+  tmp = lru_head->prev;
+  if (tmp == lru_head)
     break;
-  list_del_init(tmp);
-  prefetch(dentry_unused.prev);
+
+  prefetch(lru_head->prev);
     dentry_stat.nr_unused--;
-  dentry = list_entry(tmp, struct dentry, d_lru);
+  dentry = sb ? list_entry(tmp, struct dentry, d_sb_lru) :
+    list_entry(tmp, struct dentry, d_lru);
+  list_del_init(&dentry->d_lru);
+  list_del_init(&dentry->d_sb_lru);

    spin_lock(&dentry->d_lock);
   /*
@@ -440,6 +438,8 @@ static void prune_dcache(int count, stru
   if (dentry->d_flags & DCACHE_REFERENCED) {
     dentry->d_flags &= ~DCACHE_REFERENCED;
     list_add(&dentry->d_lru, &dentry_unused);
+   list_add(&dentry->d_sb_lru,
+     &dentry->d_sb->s_dentry_unused);
     dentry_stat.nr_unused++;
     spin_unlock(&dentry->d_lock);
    continue;
@@ -455,7 +455,8 @@ static void prune_dcache(int count, stru
    * If this dentry is for "my" filesystem, then I can prune it
    * without taking the s_umount lock (I already hold it).
    */
-  if (sb && dentry->d_sb == sb) {
+  if (sb) {
+   BUG_ON(dentry->d_sb != sb);
    prune_one_dentry(dentry);
    continue;
   }
@@ -480,6 +481,8 @@ static void prune_dcache(int count, stru
   spin_unlock(&dentry->d_lock);
   /* Inserting dentry to tail of the list leads to cycle */
    list_add(&dentry->d_lru, &dentry_unused);
+  list_add(&dentry->d_sb_lru,
+    &dentry->d_sb->s_dentry_unused);
   dentry_stat.nr_unused++;
  }
  spin_unlock(&dcache_lock);
```

```
@@ -509,31 +512,15 @@ static void prune_dcache(int count, stru

 void shrink_dcache_sb(struct super_block * sb)
 {
- struct list_head *tmp, *next;
- struct dentry *dentry;
-
- /*
-  * Pass one ... move the dentries for the specified
-  * superblock to the most recent end of the unused list.
-  */
  spin_lock(&dcache_lock);
- list_for_each_safe(tmp, next, &dentry_unused) {
-  dentry = list_entry(tmp, struct dentry, d_lru);
-  if (dentry->d_sb != sb)
-   continue;
-  list_move(tmp, &dentry_unused);
- }
+ while (!list_empty(&sb->s_dentry_unused)) {
+  struct dentry *dentry;

- /*
-  * Pass two ... free the dentries for this superblock.
-  */
-repeat:
- list_for_each_safe(tmp, next, &dentry_unused) {
-  dentry = list_entry(tmp, struct dentry, d_lru);
-  if (dentry->d_sb != sb)
-   continue;
+  dentry = list_entry((&sb->s_dentry_unused)->next,
+      struct dentry, d_sb_lru);
   dentry_stat.nr_unused--;
-  list_del_init(tmp);
+  list_del_init(&dentry->d_lru);
+  list_del_init(&dentry->d_sb_lru);
   spin_lock(&dentry->d_lock);
   if (atomic_read(&dentry->d_count)) {
    spin_unlock(&dentry->d_lock);
@@ -541,7 +528,6 @@ repeat:
   }
   prune_one_dentry(dentry);
   cond_resched_lock(&dcache_lock);
-  goto repeat;
  }
  spin_unlock(&dcache_lock);
 }
@@ -563,6 +549,7 @@ static void shrink_dcache_for_umount_sub
  if (!list_empty(&dentry->d_lru)) {
```

```
   dentry_stat.nr_unused--;
   list_del_init(&dentry->d_lru);
+  list_del_init(&dentry->d_sb_lru);
  }
  __d_drop(dentry);
  spin_unlock(&dcache_lock);
@@ -580,6 +567,7 @@ static void shrink_dcache_for_umount_sub
     if (!list_empty(&loop->d_lru)) {
      dentry_stat.nr_unused--;
      list_del_init(&loop->d_lru);
+     list_del_init(&loop->d_sb_lru);
     }

     __d_drop(loop);
@@ -766,6 +754,7 @@ resume:
   if (!list_empty(&dentry->d_lru)) {
    dentry_stat.nr_unused--;
    list_del_init(&dentry->d_lru);
+   list_del_init(&dentry->d_sb_lru);
   }
   /*
    * move only zero ref count dentries to the end
@@ -773,6 +762,8 @@ resume:
    */
   if (!atomic_read(&dentry->d_count)) {
    list_add_tail(&dentry->d_lru, &dentry_unused);
+   list_add_tail(&dentry->d_sb_lru,
+     &dentry->d_sb->s_dentry_unused);
    dentry_stat.nr_unused++;
    found++;
   }
@@ -892,6 +883,7 @@ struct dentry *d_alloc(struct dentry * p
 #endif
  INIT_HLIST_NODE(&dentry->d_hash);
  INIT_LIST_HEAD(&dentry->d_lru);
+ INIT_LIST_HEAD(&dentry->d_sb_lru);
  INIT_LIST_HEAD(&dentry->d_subdirs);
  INIT_LIST_HEAD(&dentry->d_alias);

--- linux-2.6.19-rc3/fs/super.c.shrsb 2006-10-24 10:29:13.000000000 +0400
+++ linux-2.6.19-rc3/fs/super.c 2006-10-27 15:36:33.000000000 +0400
@@ -69,6 +69,7 @@ static struct super_block *alloc_super(s
   INIT_LIST_HEAD(&s->s_io);
   INIT_LIST_HEAD(&s->s_files);
   INIT_LIST_HEAD(&s->s_instances);
+  INIT_LIST_HEAD(&s->s_dentry_unused);
   INIT_HLIST_HEAD(&s->s_anon);
   INIT_LIST_HEAD(&s->s_inodes);
```

```
   init_rwsem(&s->s_umount);
--- linux-2.6.19-rc3/include/linux/dcache.h.shrsb 2006-10-24 10:29:14.000000000
+0400
+++ linux-2.6.19-rc3/include/linux/dcache.h 2006-10-27 15:36:33.000000000 +0400
@@ -94,6 +94,7 @@ struct dentry {
  struct qstr d_name;

  struct list_head d_lru;  /* LRU list */
+ struct list_head d_sb_lru; /* per-sb LRU list */
  /*
   * d_child and d_rcu can share memory
   */
--- linux-2.6.19-rc3/include/linux/fs.h.shrsb 2006-10-24 10:29:14.000000000 +0400
+++ linux-2.6.19-rc3/include/linux/fs.h 2006-10-27 15:36:33.000000000 +0400
@@ -939,6 +939,7 @@ struct super_block {
  struct list_head s_dirty; /* dirty inodes */
  struct list_head s_io;  /* parked for writeback */
  struct hlist_head s_anon;  /* anonymous dentries for (nfs) exporting */
+ struct list_head s_dentry_unused;
  struct list_head s_files;

  struct block_device *s_bdev;
```

Subject: Re: [PATCH 2.6.19-rc3] VFS: missing unused dentry in prune_dcache()
Posted by David Howells on Fri, 27 Oct 2006 14:24:33 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> + /* Inserting dentry to tail of the list leads to cycle */
> +   list_add(&dentry->d_lru, &dentry_unused);
> + dentry_stat.nr_unused++;

I'd phrase that comment differently: "Insert dentry at the head of the list as
inserting at the tail leads to a cycle".

But other than that:

Acked-By: David Howells <dhowells@redhat.com>

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by David Howells on Fri, 27 Oct 2006 14:29:05 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> Umount calls shrink_dcache_sb in "Special case for "unmounting" root".
> Usually it happen only once, but in case OpenVZ it happens every time when any
> Virtual server is stopped, each of them have own isolated root partition.

Where is that? Are you talking about do_remount_sb()?

David

Subject: Re: [Q] missing unused dentry in prune_dcache()?
Posted by dev on Fri, 27 Oct 2006 14:34:12 GMT
View Forum Message <> Reply to Message

> Vasily Averin <vvs@sw.ru> wrote:
>
>
>>Umount calls shrink_dcache_sb in "Special case for "unmounting" root".
>>Usually it happen only once, but in case OpenVZ it happens every time when any
>>Virtual server is stopped, each of them have own isolated root partition.
>
>
> Where is that? Are you talking about do_remount_sb()?
AFAIR yes.

Kirill

Subject: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by Andrew Morton on Fri, 27 Oct 2006 18:06:45 GMT
View Forum Message <> Reply to Message

On Fri, 27 Oct 2006 18:05:50 +0400
Vasily Averin <vvs@sw.ru> wrote:

> Virtuozzo/OpenVZ linux kernel team has discovered that umount/remount can last
> for hours looping in shrink_dcache_sb() without much successes. Since during
> shrinking s_umount semaphore is taken lots of other unrelated operations like
> sync can stop working until shrink finished.

Did you consider altering shrink_dcache_sb() so that it holds onto
dcache_lock and moves all the to-be-pruned dentries onto a private list in
a single pass, then prunes them all outside the lock?

Subject: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list

Posted by David Chinner on Mon, 30 Oct 2006 04:24:19 GMT
View Forum Message <> Reply to Message

On Fri, Oct 27, 2006 at 06:05:50PM +0400, Vasily Averin wrote:
>
> The proposed fix prevents this issue by using per-sb dentry LRU list. It
> provides very quickly search for the unused dentries for given super block thus
> forcing shrinking always making good progress.

We've been down this path before:

http://marc.theaimsgroup.com/?l=linux-kernel&m=114861109 717260&w=2

A lot of comments on per-sb unused dentry lists were made in
the threads associated with the above. other solutions were
found to the problem that the above patch addressed, but I don't
think any of them have made it to mainline yet. You might want
to have a bit of a read of these threads first...

Cheers,

Dave.
--
Dave Chinner
Principal Engineer
SGI Australian Software Group

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by dev on Mon, 30 Oct 2006 06:22:14 GMT
View Forum Message <> Reply to Message

David,

>>The proposed fix prevents this issue by using per-sb dentry LRU list. It
>>provides very quickly search for the unused dentries for given super block thus
>>forcing shrinking always making good progress.
>
>
> We've been down this path before:
>
> http://marc.theaimsgroup.com/?l=linux-kernel&m=114861109 717260&w=2
>
> A lot of comments on per-sb unused dentry lists were made in
> the threads associated with the above. other solutions were
> found to the problem that the above patch addressed, but I don't
> think any of them have made it to mainline yet. You might want
> to have a bit of a read of these threads first...

The major difference between our patch and the one discussed in the link
it that we keep both global and per-sb dentry LRU lists.
Thus, when needed normal LRU is used and prune logic is unchanged,
while umount/remount use per-sb list and do its job faster.

Thanks,
Kirill

Subject: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by vaverin on Mon, 30 Oct 2006 14:24:22 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Fri, 27 Oct 2006 18:05:50 +0400
> Vasily Averin <vvs@sw.ru> wrote:
>
>> Virtuozzo/OpenVZ linux kernel team has discovered that umount/remount can last
>> for hours looping in shrink_dcache_sb() without much successes. Since during
>> shrinking s_umount semaphore is taken lots of other unrelated operations like
>> sync can stop working until shrink finished.
>
> Did you consider altering shrink_dcache_sb() so that it holds onto
> dcache_lock and moves all the to-be-pruned dentries onto a private list in
> a single pass, then prunes them all outside the lock?

At the first glance it is wrong because of 2 reasons:
1) it continues to check the whole global LRU list (we propose to use per-sb
LRU, it will provide very quick search)
2) we have not any guarantee that someone will add new unused dentries to the
list when we prune it outside the lock. And to the contrary, some of unused
dentries can be used again. As far as I understand we should hold dcache_lock
beginning at the removing dentry from unused_list until dentry_iput() call.

David did it inside shrink_dcache_for_umount() just because it have guarantee
that all the filesystem operations are finished and new ones cannot be started.

Thank you,
Vasily Averin

Subject: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by dev on Mon, 30 Oct 2006 15:06:48 GMT
View Forum Message <> Reply to Message

Andrew,

>>Virtuozzo/OpenVZ linux kernel team has discovered that umount/remount can last
>>for hours looping in shrink_dcache_sb() without much successes. Since during
>>shrinking s_umount semaphore is taken lots of other unrelated operations like
>>sync can stop working until shrink finished.
>
>
> Did you consider altering shrink_dcache_sb() so that it holds onto
> dcache_lock and moves all the to-be-pruned dentries onto a private list in
> a single pass, then prunes them all outside the lock?

moving dentries from global list to the local one can take arbitrary number
of milliseconds (with huge amount of memory), so nothing good here from latency
view point.

Thanks,
Kirill

Subject: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by Eric Dumazet on Mon, 30 Oct 2006 15:08:25 GMT
View Forum Message <> Reply to Message

On Monday 30 October 2006 15:24, Vasily Averin wrote:
> Andrew Morton wrote:
> > On Fri, 27 Oct 2006 18:05:50 +0400
> >
> > Vasily Averin <vvs@sw.ru> wrote:
> >> Virtuozzo/OpenVZ linux kernel team has discovered that umount/remount
> >> can last for hours looping in shrink_dcache_sb() without much successes.
> >> Since during shrinking s_umount semaphore is taken lots of other
> >> unrelated operations like sync can stop working until shrink finished.
> >
> > Did you consider altering shrink_dcache_sb() so that it holds onto
> > dcache_lock and moves all the to-be-pruned dentries onto a private list
> > in a single pass, then prunes them all outside the lock?
>
> At the first glance it is wrong because of 2 reasons:
> 1) it continues to check the whole global LRU list (we propose to use
> per-sb LRU, it will provide very quick search)

Quick search maybe, but your patch adds 2 pointers to each dentry in the
system... That's pretty expensive, as dentries are already using a *lot* of
ram.

Maybe an alternative would be to not have anymore a global dentry_unused, but
only per-sb unused dentries lists ?

> 2) we have not any guarantee that someone will add new unused dentries to
> the list when we prune it outside the lock. And to the contrary, some of
> unused dentries can be used again. As far as I understand we should hold
> dcache_lock beginning at the removing dentry from unused_list until
> dentry_iput() call.
>
> David did it inside shrink_dcache_for_umount() just because it have
> guarantee that all the filesystem operations are finished and new ones
> cannot be started.

Subject: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by dev on Mon, 30 Oct 2006 15:27:43 GMT
View Forum Message <> Reply to Message

> Quick search maybe, but your patch adds 2 pointers to each dentry in the
> system... That's pretty expensive, as dentries are already using a *lot* of
> ram.
I don't see much problems with it... it is cache and it can be pruned if needed.
Some time ago, for example, my patch introducing the same list for inodes
was commited.

> Maybe an alternative would be to not have anymore a global dentry_unused, but
> only per-sb unused dentries lists ?
I don't know global LRU implementation based on per-sb lists, do you?
If someone suggest the algorithm for more or less fair global LRU
based on non-global list we will implement it. However, so far,
AFAICS there were problems with it.

Thanks,
Kirill

Subject: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by Eric Dumazet on Mon, 30 Oct 2006 16:09:31 GMT
View Forum Message <> Reply to Message

On Monday 30 October 2006 16:34, Kirill Korotaev wrote:
> > Quick search maybe, but your patch adds 2 pointers to each dentry in the
> > system... That's pretty expensive, as dentries are already using a *lot*
> > of ram.
>
> I don't see much problems with it... it is cache and it can be pruned if
> needed. Some time ago, for example, my patch introducing the same list for
> inodes was commited.

The ratio of dentries/PAGE is higher than ration of inodes/PAGE.

(On a x86_64 machine, 19 dentries per PAGE, 5 ext3 inodes per PAGE=

Therefore I suspect that the number of pages locked in dentry_cache because of
one inuse dentry is higher.

>
> > Maybe an alternative would be to not have anymore a global dentry_unused,
> > but only per-sb unused dentries lists ?
>
> I don't know global LRU implementation based on per-sb lists, do you?
> If someone suggest the algorithm for more or less fair global LRU
> based on non-global list we will implement it. However, so far,
> AFAICS there were problems with it.

Using 32 bytes per dentry for unused LRUs sounds too much, maybe we should
revert LRUS handling to timestamps or things like that.

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by Neil Brown on Tue, 31 Oct 2006 04:38:58 GMT
View Forum Message <> Reply to Message

On Monday October 30, dev@sw.ru wrote:
> David,
>
> >>The proposed fix prevents this issue by using per-sb dentry LRU list. It
> >>provides very quickly search for the unused dentries for given super block thus
> >>forcing shrinking always making good progress.
> >
> >
> > We've been down this path before:
> >
> > http://marc.theaimsgroup.com/?l=linux-kernel&m=114861109 717260&w=2
> >
> > A lot of comments on per-sb unused dentry lists were made in
> > the threads associated with the above. other solutions were
> > found to the problem that the above patch addressed, but I don't
> > think any of them have made it to mainline yet. You might want
> > to have a bit of a read of these threads first...
> The major difference between our patch and the one discussed in the link
> it that we keep both global and per-sb dentry LRU lists.
> Thus, when needed normal LRU is used and prune logic is unchanged,
> while umount/remount use per-sb list and do its job faster.

Yes, we have been down this path before - several times I think.
Below is the patch that I like (not tested recently - just rediffed
and reviewed).

NeilBrown

Subject: Reduce contention in dentry_unused when unmounting.

When we unmount a filesystem we need to release all dentries.
We currently
  - move a collection of dentries to the end of the dentry_unused list
  - call prune_dcache to prune that number of dentries.

If lots of other dentries are added to the end of the list while
the prune_dcache proceeds (e.g. another filesystem is unmounted),
this can involve a lot of wasted time wandering through the
list looking for dentries that we had previously found.

This patch allows the dentry_unused list to temporarily be multiple
lists.
When unmounting, dentries that are found to require pruning are moved
to a temporary list, but accounted as though they were on dentry_unused.
Then this list is passed to prune_dcache for freeing. Any entries
that are not pruned for whatever reason are added to the end of
dentry_unused.

Also change shrink_dcache_sb to simply call shrink_dcache_parent.
This avoids a long walk of the LRU.

Signed-off-by: Neil Brown <neilb@suse.de>

Diffstat output
 ./fs/dcache.c | 104 ++++++

```
  * all the dentries are in use.
+ *
+ * Any dentries that were not removed due to the @count
+ * limit will be splice on to the end of dentry_unused,
+ * so they should already be founded in dentry_stat.nr_unused.
  */

-static void prune_dcache(int count, struct super_block *sb)
+static void prune_dcache(int count, struct list_head *list)
 {
+ int have_list = list != NULL;
+
  spin_lock(&dcache_lock);
+ if (!have_list)
+  /* use the dentry_unused list */
+  list = &dentry_unused;
+
  for (; count ; count--) {
   struct dentry *dentry;
   struct list_head *tmp;
@@ -406,23 +416,11 @@ static void prune_dcache(int count, stru

   cond_resched_lock(&dcache_lock);

-  tmp = dentry_unused.prev;
-  if (sb) {
-   /* Try to find a dentry for this sb, but don't try
-    * too hard, if they aren't near the tail they will
-    * be moved down again soon
-    */
-   int skip = count;
-   while (skip && tmp != &dentry_unused &&
-       list_entry(tmp, struct dentry, d_lru)->d_sb != sb) {
-    skip--;
-    tmp = tmp->prev;
-   }
-  }
-  if (tmp == &dentry_unused)
+  tmp = list->prev;
+  if (tmp == list)
    break;
   list_del_init(tmp);
-  prefetch(dentry_unused.prev);
+  prefetch(list->prev);
    dentry_stat.nr_unused--;
   dentry = list_entry(tmp, struct dentry, d_lru);

@@ -455,7 +453,7 @@ static void prune_dcache(int count, stru
```

```
     * If this dentry is for "my" filesystem, then I can prune it
     * without taking the s_umount lock (I already hold it).
     */
-  if (sb && dentry->d_sb == sb) {
+  if (have_list) {
     prune_one_dentry(dentry);
     continue;
   }
@@ -485,68 +483,25 @@ static void prune_dcache(int count, stru
     list_add(&dentry->d_lru, &dentry_unused);
   dentry_stat.nr_unused++;
  }
+ /* split any remaining entries back onto dentry_unused */
+ if (have_list)
+  list_splice(list, dentry_unused.prev);
  spin_unlock(&dcache_lock);
 }

-/*
- * Shrink the dcache for the specified super block.
- * This allows us to unmount a device without disturbing
- * the dcache for the other devices.
- *
- * This implementation makes just two traversals of the
- * unused list.  On the first pass we move the selected
- * dentries to the most recent end, and on the second
- * pass we free them.  The second pass must restart after
- * each dput(), but since the target dentries are all at
- * the end, it's really just a single traversal.
- */
-
 /**
  * shrink_dcache_sb - shrink dcache for a superblock
  * @sb: superblock
  *
  * Shrink the dcache for the specified super block. This
- * is used to free the dcache before unmounting a file
- * system
+ * is used to reduce the dcache presence of a file system
+ * before re-mounting, and when invalidating the device
+ * holding a file system.
  */

 void shrink_dcache_sb(struct super_block * sb)
 {
- struct list_head *tmp, *next;
- struct dentry *dentry;
-
```

```
- /*
-  * Pass one ... move the dentries for the specified
-  * superblock to the most recent end of the unused list.
-  */
- spin_lock(&dcache_lock);
- list_for_each_safe(tmp, next, &dentry_unused) {
-  dentry = list_entry(tmp, struct dentry, d_lru);
-  if (dentry->d_sb != sb)
-   continue;
-  list_move(tmp, &dentry_unused);
- }
-
- /*
-  * Pass two ... free the dentries for this superblock.
-  */
-repeat:
- list_for_each_safe(tmp, next, &dentry_unused) {
-  dentry = list_entry(tmp, struct dentry, d_lru);
-  if (dentry->d_sb != sb)
-   continue;
-  dentry_stat.nr_unused--;
-  list_del_init(tmp);
-  spin_lock(&dentry->d_lock);
-  if (atomic_read(&dentry->d_count)) {
-   spin_unlock(&dentry->d_lock);
-   continue;
-  }
-  prune_one_dentry(dentry);
-  cond_resched_lock(&dcache_lock);
-  goto repeat;
- }
- spin_unlock(&dcache_lock);
+ shrink_dcache_parent(sb->s_root);
 }

 /*
@@ -739,7 +694,7 @@ positive:

 /*
  * Search the dentry child list for the specified parent,
- * and move any unused dentries to the end of the unused
+ * and move any unused dentries to the end of a new unused
  * list for prune_dcache(). We descend to the next level
  * whenever the d_subdirs list is non-empty and continue
  * searching.
@@ -751,7 +706,7 @@ positive:
  * drop the lock and return early due to latency
  * constraints.
```

```
 */
-static int select_parent(struct dentry * parent)
+static int select_parent(struct dentry * parent, struct list_head *new)
 {
  struct dentry *this_parent = parent;
  struct list_head *next;
@@ -775,7 +730,7 @@ resume:
     * of the unused list for prune_dcache
     */
    if (!atomic_read(&dentry->d_count)) {
-    list_add_tail(&dentry->d_lru, &dentry_unused);
+    list_add_tail(&dentry->d_lru, new);
     dentry_stat.nr_unused++;
     found++;
    }
@@ -819,9 +774,10 @@ out:
 void shrink_dcache_parent(struct dentry * parent)
 {
  int found;
+ LIST_HEAD(list);

- while ((found = select_parent(parent)) != 0)
-  prune_dcache(found, parent->d_sb);
+ while ((found = select_parent(parent, &list)) != 0)
+  prune_dcache(found, &list);
 }

 /*
```

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by David Howells on Tue, 31 Oct 2006 10:40:14 GMT
View Forum Message <> Reply to Message

Neil Brown <neilb@suse.de> wrote:

> When we unmount a filesystem we need to release all dentries.
> We currently
> - move a collection of dentries to the end of the dentry_unused list
> - call prune_dcache to prune that number of dentries.

This is not true anymore.

David

Subject: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list

Hello Neil,

Neil Brown wrote:
>>> We've been down this path before:
>>>
>>> http://marc.theaimsgroup.com/?l=linux-kernel&m=114861109 717260&w=2
>>>
>>> A lot of comments on per-sb unused dentry lists were made in
>>> the threads associated with the above. other solutions were
>>> found to the problem that the above patch addressed, but I don't
>>> think any of them have made it to mainline yet. You might want
>>> to have a bit of a read of these threads first...
>> The major difference between our patch and the one discussed in the link
>> it that we keep both global and per-sb dentry LRU lists.
>> Thus, when needed normal LRU is used and prune logic is unchanged,
>> while umount/remount use per-sb list and do its job faster.
>
> Yes, we have been down this path before - several times I think.
> Below is the patch that I like (not tested recently - just rediffed
> and reviewed).

I've reviewed your patch, in general it looks great and I'm very like an idea to
have only one place to free the unused dentries.
I have several minor notes and one big question:

> @@ -485,68 +483,25 @@ static void prune_dcache(int count, stru
>   list_add(&dentry->d_lru, &dentry_unused);
>   dentry_stat.nr_unused++;
>   }
> + /* split any remaining entries back onto dentry_unused */
> + if (have_list)
> + list_splice(list, dentry_unused.prev);
>   spin_unlock(&dcache_lock);
> }

- It seems for me this hunk is not required here. You are knows how many entries
are present in the private list and nobody can add new dentries to it. Therefore
I think your list should be empty on exit.

- I do not like idea to split dentry_unused list, IMHO it broke LRU slightly (if
somebody on the other CPU's shrinks dcache). However on the other hand this
split is temporal and therefore it should not lead to any problems.

Unfortunately I'm not sure that it can help us against long remount. As far as I
understand the work time of shrink_dcache_sb() function depends on the subtree
size, and you need to check whole tree several (at least 3) times:

1) to remove all the unused dentries from dentry_unused to the private list
2) to collect all the non-freed dentries (prune_dcache will insert
DCACHE_REFERENCED dentries back to the global LRU list)
3) to check that there is not any unused dentries.

If the tree is large we will handle it too long and can be rescheduled. In these cases steps 1) and 2) may be restarted several times. In the worst case we will prune only one dentry per cycle. And it is without any filesystem activity.

And now let's assume that we do the same on the live filesystem. I'm afraid in this case shrink_dcache_sb() may be executed unpredictable long time with taken s_umount semaphore. :(I would note that it is real issue, and some of our users have complained about this behaviour.

In case of per-sb LRU we will not have any additional overheads, we will be able to traverse over the unused dentries list without any additional overheads. Heavy activity on the filesystem will delay us too, but we will not depends on filesystem size.

As far as I understand nobody is against per-sb LRU, but nobody wants to add the new fields to the dentry struct.

Let's see on the possible alternatives:

1) per-sb LRU + global LRU (OpenVZ patch): it is fastest and (IMHO) performance-optimal but requires 2 additional pointers on dentry struct.

2) per-sb LRU + per-dentry time stamp (Eric Dumazet idea):
shirnk_dcache_sb() and per-sb prune_dcache(,sb) are optimal (like in case 1), but shrink_dcache(,NULL) called from shrink_dcache_memory() will be executed slower than in current version(). It requires only one additional field on dentry struct for time stamp, but probably we can use some of current fields on the dentry struct (d_count? d_time?) for this purpose?

3) we can remove both LRU from struct dentry to the some dynamic-allocated structure. It will be allocated for each new unused dentry and it can contain both LRU fields and probably some other fields. Also by this way we can decrease size of dentry. However ratio used/unused dentries is too low and probably it will not have any advantages.

4) think how we can modify the other proposed patches (Neil Brown, Dave Chinner, David Howells)

5) any new ideas?

Thank you,
Vasily Averin

Subject: [Q] missing ->d_delete() in shrink_dcache_for_umount()?
Posted by vaverin on Tue, 31 Oct 2006 13:24:36 GMT
View Forum Message <> Reply to Message

David,

Vasily Averin wrote:
> Hello folks,
>
> I would like to ask you clarify me one question in the the following patch:
> http://linux.bkbits.net:8080/linux-2.6/gnupatch@449b144ecSF1 rYskg3q-SeR2vf88zg
> # ChangeSet
> # 2006/06/22 15:05:57-07:00 neilb@suse.de
> # [PATCH] Fix dcache race during umount
>
> # If prune_dcache finds a dentry that it cannot free, it leaves it where it
> # is (at the tail of the list) and exits, on the assumption that some other
> # thread will be removing that dentry soon.

It looks like I've noticed yet one suspicious place in your patch. As far as I
see you have removed dput(root) call in shrink_dcache_for_umount() function.
However I would note that dput contains ->d_delete() call that is missing in
your function:

```
if (dentry->d_op && dentry->d_op->d_delete) {
 if (dentry->d_op->d_delete(dentry))
  goto unhash_it;
```

I'm not sure but it seems to me some (probably out-of-tree) filesystems can do
something useful in this place.

Thank you,
 Vasily Averin

Subject: Re: [Q] missing ->d_delete() in shrink_dcache_for_umount()?
Posted by David Howells on Tue, 31 Oct 2006 15:06:57 GMT
View Forum Message <> Reply to Message

Vasily Averin <vvs@sw.ru> wrote:

> It looks like I've noticed yet one suspicious place in your patch. As far as I
> see you have removed dput(root) call in shrink_dcache_for_umount() function.
> However I would note that dput contains ->d_delete() call that is missing in
> your function:
>
> if (dentry->d_op && dentry->d_op->d_delete) {
> if (dentry->d_op->d_delete(dentry))

> goto unhash_it;
>
> I'm not sure but it seems to me some (probably out-of-tree) filesystems can do
> something useful in this place.

Can they though? I'm not so sure. I've added Al to the To list to get his
opinion.

It seems to me that d_op->d_delete() is asking a question of whether the dentry
should be discarded immediately upon dput() reducing the usage count to 0. The
documentation in vfs.txt isn't entirely clear on this point, but what it does
say is that that op isn't allowed to sleep, so there's a limit as to what it
can do.

Furthermore, d_op->d_delete() is probably the wrong hook to call. It returns
an indication of whether the dentry should be retained or not, but we aren't
interested in that at this point: we have to get rid of the dentry anyway, so
the fs's opinion is irrelevant.

Finally, we do still call d_op->d_release() by way of d_free(), so it's not as
if the fs doesn't get a chance to clean up the dentry.

David

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by Neil Brown on Wed, 01 Nov 2006 06:32:54 GMT
View Forum Message <> Reply to Message

On Tuesday October 31, dhowells@redhat.com wrote:
> Neil Brown <neilb@suse.de> wrote:
>
> > When we unmount a filesystem we need to release all dentries.
> > We currently
> > - move a collection of dentries to the end of the dentry_unused list
> > - call prune_dcache to prune that number of dentries.
>
> This is not true anymore.

True. That should read:

 When we remount a filesystem or invalidate a block device which has a
 mounted filesystem we call shrink dcache_sb which currently:
    - moves a collection of dentries to the end of the dentry_unused list
    - calls prune_dcache to prune that number of dentries.

but the patch is still valid.

Any objections to it going in to -mm and maybe .20 ??

Thanks,
NeilBrown

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by dev on Wed, 01 Nov 2006 10:48:06 GMT
View Forum Message <> Reply to Message

Neil Brown wrote:
> On Monday October 30, dev@sw.ru wrote:
>
>>David,
>>
>>
>>>>>The proposed fix prevents this issue by using per-sb dentry LRU list. It
>>>>provides very quickly search for the unused dentries for given super block thus
>>>>forcing shrinking always making good progress.
>>>
>>>
>>>We've been down this path before:
>>>
>>> http://marc.theaimsgroup.com/?l=linux-kernel&m=114861109 717260&w=2
>>>
>>>A lot of comments on per-sb unused dentry lists were made in
>>>the threads associated with the above. other solutions were
>>>found to the problem that the above patch addressed, but I don't
>>>think any of them have made it to mainline yet. You might want
>>>to have a bit of a read of these threads first...
>>
>>The major difference between our patch and the one discussed in the link
>>it that we keep both global and per-sb dentry LRU lists.
>>Thus, when needed normal LRU is used and prune logic is unchanged,
>>while umount/remount use per-sb list and do its job faster.
>
>
> Yes, we have been down this path before - several times I think.
> Below is the patch that I like (not tested recently - just rediffed
> and reviewed).
>
> NeilBrown
>
> Subject: Reduce contention in dentry_unused when unmounting.
>
> When we unmount a filesystem we need to release all dentries.
> We currently

>   - move a collection of dentries to the end of the dentry_unused list
>   - call prune_dcache to prune that number of dentries.
>
> If lots of other dentries are added to the end of the list while
> the prune_dcache proceeds (e.g. another filesystem is unmounted),
> this can involve a lot of wasted time wandering through the
> list looking for dentries that we had previously found.
>
> This patch allows the dentry_unused list to temporarily be multiple
> lists.
> When unmounting, dentries that are found to require pruning are moved
> to a temporary list, but accounted as though they were on dentry_unused.
> Then this list is passed to prune_dcache for freeing. Any entries
> that are not pruned for whatever reason are added to the end of
> dentry_unused.
>
> Also change shrink_dcache_sb to simply call shrink_dcache_parent.
> This avoids a long walk of the LRU.
>
> Signed-off-by: Neil Brown <neilb@suse.de>
>
> ### Diffstat output
> ./fs/dcache.c | 104 +++++++++++++++++++----------------------------------
> 1 file changed, 30 insertions(+), 74 deletions(-)
>
> diff .prev/fs/dcache.c ./fs/dcache.c
> --- .prev/fs/dcache.c 2006-10-31 15:22:10.000000000 +1100
> +++ ./fs/dcache.c 2006-10-31 15:37:22.000000000 +1100
> @@ -384,8 +384,8 @@ static void prune_one_dentry(struct dent
> /**
>   * prune_dcache - shrink the dcache
>   * @count: number of entries to try and free
> - * @sb: if given, ignore dentries for other superblocks
> - *   which are being unmounted.
> + * @list: If given, remove from this list instead of
> + *   from dentry_unused.
>   *
>   * Shrink the dcache. This is done when we need
>   * more memory, or simply when we need to unmount
> @@ -394,11 +394,21 @@ static void prune_one_dentry(struct dent
>   *
>   * This function may fail to free any resources if
>   * all the dentries are in use.
> + *
> + * Any dentries that were not removed due to the @count
> + * limit will be splice on to the end of dentry_unused,
> + * so they should already be founded in dentry_stat.nr_unused.
>   */

```
>
> -static void prune_dcache(int count, struct super_block *sb)
> +static void prune_dcache(int count, struct list_head *list)
>  {
> + int have_list = list != NULL;
> +
>   spin_lock(&dcache_lock);
> + if (!have_list)
> +  /* use the dentry_unused list */
> +  list = &dentry_unused;
> +
>   for (; count ; count--) {
>    struct dentry *dentry;
>    struct list_head *tmp;
> @@ -406,23 +416,11 @@ static void prune_dcache(int count, stru
>
>    cond_resched_lock(&dcache_lock);
>
> -  tmp = dentry_unused.prev;
> -  if (sb) {
> -   /* Try to find a dentry for this sb, but don't try
> -    * too hard, if they aren't near the tail they will
> -    * be moved down again soon
> -    */
> -   int skip = count;
> -   while (skip && tmp != &dentry_unused &&
> -       list_entry(tmp, struct dentry, d_lru)->d_sb != sb) {
> -    skip--;
> -    tmp = tmp->prev;
> -   }
> -  }
> -  if (tmp == &dentry_unused)
> +  tmp = list->prev;
> +  if (tmp == list)
>     break;
>    list_del_init(tmp);
> -  prefetch(dentry_unused.prev);
> +  prefetch(list->prev);
>     dentry_stat.nr_unused--;
>    dentry = list_entry(tmp, struct dentry, d_lru);
>
> @@ -455,7 +453,7 @@ static void prune_dcache(int count, stru
>     * If this dentry is for "my" filesystem, then I can prune it
>     * without taking the s_umount lock (I already hold it).
>     */
> -  if (sb && dentry->d_sb == sb) {
> +  if (have_list) {
>     prune_one_dentry(dentry);
```

>   continue;
>   }
> @@ -485,68 +483,25 @@ static void prune_dcache(int count, stru
>   list_add(&dentry->d_lru, &dentry_unused);
>   dentry_stat.nr_unused++;
>   }
> + /* split any remaining entries back onto dentry_unused */
> + if (have_list)
> + list_splice(list, dentry_unused.prev);
>   spin_unlock(&dcache_lock);
> }
>
> -/*
> - * Shrink the dcache for the specified super block.
> - * This allows us to unmount a device without disturbing
> - * the dcache for the other devices.
> - *
> - * This implementation makes just two traversals of the
> - * unused list. On the first pass we move the selected
> - * dentries to the most recent end, and on the second
> - * pass we free them. The second pass must restart after
> - * each dput(), but since the target dentries are all at
> - * the end, it's really just a single traversal.
> - */
> -
> /**
>   * shrink_dcache_sb - shrink dcache for a superblock
>   * @sb: superblock
>   *
>   * Shrink the dcache for the specified super block. This
> - * is used to free the dcache before unmounting a file
> - * system
> + * is used to reduce the dcache presence of a file system
> + * before re-mounting, and when invalidating the device
> + * holding a file system.
>   */
>
> void shrink_dcache_sb(struct super_block * sb)
> {
> - struct list_head *tmp, *next;
> - struct dentry *dentry;
> -
> - /*
> - * Pass one ... move the dentries for the specified
> - * superblock to the most recent end of the unused list.
> - */
> - spin_lock(&dcache_lock);
> - list_for_each_safe(tmp, next, &dentry_unused) {

> - dentry = list_entry(tmp, struct dentry, d_lru);
> - if (dentry->d_sb != sb)
> - continue;
> - list_move(tmp, &dentry_unused);
> - }
> -
> - /*
> - * Pass two ... free the dentries for this superblock.
> - */
> -repeat:
> - list_for_each_safe(tmp, next, &dentry_unused) {
> - dentry = list_entry(tmp, struct dentry, d_lru);
> - if (dentry->d_sb != sb)
> - continue;
> - dentry_stat.nr_unused--;
> - list_del_init(tmp);
> - spin_lock(&dentry->d_lock);
> - if (atomic_read(&dentry->d_count)) {
> - spin_unlock(&dentry->d_lock);
> - continue;
> - }
> - prune_one_dentry(dentry);
> - cond_resched_lock(&dcache_lock);
> - goto repeat;
> - }
> - spin_unlock(&dcache_lock);
> + shrink_dcache_parent(sb->s_root);
<<<< AFAICS, doing so you introduced a leak of anonymous dentries.

d_alloc_anon() calls d_alloc() with parent == NULL, i.e. dentries have no parent
and are not linked to the sb->s_root...
BTW, looking at it, I found that s_anon field on super block is not used any more.
we can add BUG_ON(!hlist_empty(&sb->s_anon)) in generic_shutdown_super to avoid such
issues like this.

maybe we can fix it adding something like:
while (!list_empty(&sb->s_anon)))
    prune_dcache(MAX_INT, &sb->s_anon);

> }
>
> /*
> @@ -739,7 +694,7 @@ positive:
>
> /*
> * Search the dentry child list for the specified parent,
> - * and move any unused dentries to the end of the unused
> + * and move any unused dentries to the end of a new unused

```
>   * list for prune_dcache(). We descend to the next level
>   * whenever the d_subdirs list is non-empty and continue
>   * searching.
> @@ -751,7 +706,7 @@ positive:
>   * drop the lock and return early due to latency
>   * constraints.
>   */
> -static int select_parent(struct dentry * parent)
> +static int select_parent(struct dentry * parent, struct list_head *new)
>  {
>   struct dentry *this_parent = parent;
>   struct list_head *next;
> @@ -775,7 +730,7 @@ resume:
>     * of the unused list for prune_dcache
>     */
>    if (!atomic_read(&dentry->d_count)) {
> -   list_add_tail(&dentry->d_lru, &dentry_unused);
> +   list_add_tail(&dentry->d_lru, new);
>     dentry_stat.nr_unused++;
>     found++;
>    }
> @@ -819,9 +774,10 @@ out:
>  void shrink_dcache_parent(struct dentry * parent)
>  {
>   int found;
> + LIST_HEAD(list);
>
> - while ((found = select_parent(parent)) != 0)
> -  prune_dcache(found, parent->d_sb);
> + while ((found = select_parent(parent, &list)) != 0)
> +  prune_dcache(found, &list);
>  }
>
>  /*
>
```

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by vaverin on Wed, 01 Nov 2006 13:32:03 GMT

View Forum Message <> Reply to Message

Hello Neil,

Neil Brown wrote:
> On Tuesday October 31, dhowells@redhat.com wrote:
>> Neil Brown <neilb@suse.de> wrote:
>>
>>> When we unmount a filesystem we need to release all dentries.

>>> We currently
>>>   - move a collection of dentries to the end of the dentry_unused list
>>>   - call prune_dcache to prune that number of dentries.
>> This is not true anymore.
>
> True. That should read:
>
> When we remount a filesystem or invalidate a block device which has a
> mounted filesystem we call shrink dcache_sb which currently:
>   - moves a collection of dentries to the end of the dentry_unused list
>   - calls prune_dcache to prune that number of dentries.
>
> but the patch is still valid.
>
> Any objections to it going in to -mm and maybe .20 ??

Currently we have 3 type of functions that works with dentry_unused list:

1) prune_dcache(NULL) -- called from shrink_dcache_memory, frees the memory and requires global LRU. works well in current implementation.
2) prune_dcache(sb) -- called from shrink_dcache_parent(), frees subtree, LRU is not need here. Current implementation uses global LRU for these purposes, it is ineffective, and patch from Neil Brown fixes this issue.
3) shrink_dcache_sb() -- called when we need to free the unused dentries for given super block. Current implementation is not effective too, and per-sb LRU would be the best solution here. On the other hand patch from Neil Brown is much better than current implementation.

In general I think that we should approve Neil Brown's patch. We (I and Kirill Korotaev) are ready to acknowledge it when the following remarks fill be fixed:

- it seems for me list_splice() is not required inside prune_dcache(),
- DCACHE_REFERENCED dentries should not be removed from private list to dentry_unused list, this flag should be ignored if the private list is used,
- count argument should be ignored in this case too, we want to free all the dentries in private list,
- when we shrink the whole super block we should free per-sb anonymous dentries too (please see Kirill Korotaev's letter)

Then I'm going to prepare new patch that will enhance the shrink_dcache_sb() performance:
- we can add new list head into struct superblock and use it in shrink_dcache_sb() instead of temporal private list. We will check is it empty in dput() and add the new unused dentries to per-sb list instead of dentry_unused list.

thank you,
 Vasily Averin

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by Neil Brown on Tue, 14 Nov 2006 04:51:08 GMT
View Forum Message <> Reply to Message

On Wednesday November 1, dev@sw.ru wrote:
> <<<< AFAICS, doing so you introduced a leak of anonymous dentries.
>
> d_alloc_anon() calls d_alloc() with parent == NULL, i.e. dentries have no parent
> and are not linked to the sb->s_root...

Yep, thanks.

> BTW, looking at it, I found that s_anon field on super block is not
> used any more.

I don't know what you mean by that. It is still used...

> we can add BUG_ON(!hlist_empty(&sb->s_anon)) in generic_shutdown_super to avoid such issues like this.
>
> maybe we can fix it adding something like:
> while (!list_empty(&sb->s_anon)))
> prune_dcache(MAX_INT, &sb->s_anon);

It seems that anon dentries can now have children (I think someone explained that too me - shrink_dcache_for_umount certainly suggests it).
Also, in this context we cannot be sure that all dentries can be freed. This is being called a remount time remember, and some stuff might still be in use.

We probably need to move the s_anon list to a temporary list and repeatedly:
move the top entry back to s_anon and call shrink_dcache_parent on it.

Needs more thought.

NeilBrown

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by Neil Brown on Tue, 14 Nov 2006 05:44:46 GMT

On Wednesday November 1, vvs@sw.ru wrote:
>
> Currently we have 3 type of functions that works with dentry_unused list:
>
> 1) prune_dcache(NULL) -- called from shrink_dcache_memory, frees the memory and
> requires global LRU. works well in current implementation.
> 2) prune_dcache(sb) -- called from shrink_dcache_parent(), frees subtree, LRU
> is not need here. Current implementation uses global LRU for these purposes, it
> is ineffective, and patch from Neil Brown fixes this issue.
> 3) shrink_dcache_sb() -- called when we need to free the unused dentries for
> given super block. Current implementation is not effective too, and per-sb LRU
> would be the best solution here. On the other hand patch from Neil Brown is much
> better than current implementation.
>
> In general I think that we should approve Neil Brown's patch. We (I and Kirill
> Korotaev) are ready to acknowledge it when the following remarks fill be fixed:

>
> - it seems for me list_splice() is not required inside
>   prune_dcache(),

Yes, the list should be empty when we finish so you are right.

> - DCACHE_REFERENCED dentries should not be removed from private list to
> dentry_unused list, this flag should be ignored if the private list is used,

Agreed.

> - count argument should be ignored in this case too, we want to free all the
> dentries in private list,

Agreed.

> - when we shrink the whole super block we should free per-sb anonymous dentries
> too (please see Kirill Korotaev's letter)
>

Yes. Unfortunately I don't think it is as easy as it sounds.
I'll have a closer look.

> Then I'm going to prepare new patch that will enhance the shrink_dcache_sb()
> performance:
> - we can add new list head into struct superblock and use it in
> shrink_dcache_sb() instead of temporal private list. We will check is it empty
> in dput() and add the new unused dentries to per-sb list instead of

> dentry_unused list.

I think that makes sense. It means that you end up doing less work in
select_parent, because the work has already been done in dput.

How is the patch going?

NeilBrown

Subject: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by vaverin on Tue, 14 Nov 2006 06:12:45 GMT
View Forum Message <> Reply to Message

Hello Neil

Neil Brown wrote:
> On Wednesday November 1, vvs@sw.ru wrote:
>> Currently we have 3 type of functions that works with dentry_unused list:
>>
>> 1) prune_dcache(NULL) -- called from shrink_dcache_memory, frees the memory and
>> requires global LRU. works well in current implementation.
>> 2) prune_dcache(sb) -- called from shrink_dcache_parent(), frees subtree, LRU
>> is not need here. Current implementation uses global LRU for these purposes, it
>> is ineffective, and patch from Neil Brown fixes this issue.
>> 3) shrink_dcache_sb() -- called when we need to free the unused dentries for
>> given super block. Current implementation is not effective too, and per-sb LRU
>> would be the best solution here. On the other hand patch from Neil Brown is much
>> better than current implementation.
>>
>> In general I think that we should approve Neil Brown's patch. We (I and Kirill
>> Korotaev) are ready to acknowledge it when the following remarks fill be fixed:
>
>
>> - it seems for me list_splice() is not required inside
>>   prune_dcache(),
>
> Yes, the list should be empty when we finish so you are right.
>
>> - DCACHE_REFERENCED dentries should not be removed from private list to
>> dentry_unused list, this flag should be ignored if the private list is used,
>
> Agreed.
>
>> - count argument should be ignored in this case too, we want to free all the
>> dentries in private list,
>
> Agreed.

>
>> - when we shrink the whole super block we should free per-sb anonymous dentries
>> too (please see Kirill Korotaev's letter)
>>
>
> Yes. Unfortunately I don't think it is as easy as it sounds.
> I'll have a closer look.
>
>
>> Then I'm going to prepare new patch that will enhance the shrink_dcache_sb()
>> performance:
>> - we can add new list head into struct superblock and use it in
>> shrink_dcache_sb() instead of temporal private list. We will check is it empty
>> in dput() and add the new unused dentries to per-sb list instead of
>> dentry_unused list.
>
> I think that makes sense. It means that you end up doing less work in
> select_parent, because the work has already been done in dput.
>
> How is the patch going?

Please see patches in attach files:
first one is incremental for your patch,
second one is merged version.

It would be very interesting to know your opinion about these changes.

thank you,
 Vasily Averin

```
--- linux-2.6.19-rc4/fs/dcache.c.vshud1 2006-11-02 14:23:03.000000000 +0300
+++ linux-2.6.19-rc4/fs/dcache.c 2006-11-03 14:57:58.000000000 +0300
@@ -171,8 +171,14 @@ repeat:
   if (d_unhashed(dentry))
   goto kill_it;
    if (list_empty(&dentry->d_lru)) {
-    dentry->d_flags |= DCACHE_REFERENCED;
-    list_add(&dentry->d_lru, &dentry_unused);
+  struct list_head *list;
+
+  list = &dentry->d_sb->s_shrink_list;
+  if (list_empty(list)) {
+   list = &dentry_unused;
+     dentry->d_flags |= DCACHE_REFERENCED;
+  }
+    list_add(&dentry->d_lru, list);
     dentry_stat.nr_unused++;
    }
```

```
   spin_unlock(&dentry->d_lock);
@@ -394,10 +400,6 @@ static void prune_one_dentry(struct dent
  *
  * This function may fail to free any resources if
  * all the dentries are in use.
- *
- * Any dentries that were not removed due to the @count
- * limit will be splice on to the end of dentry_unused,
- * so they should already be founded in dentry_stat.nr_unused.
  */

 static void prune_dcache(int count, struct list_head *list)
@@ -409,7 +411,7 @@ static void prune_dcache(int count, stru
   /* use the dentry_unused list */
   list = &dentry_unused;

- for (; count ; count--) {
+ for (; have_list || count-- ;) {
   struct dentry *dentry;
   struct list_head *tmp;
   struct rw_semaphore *s_umount;
@@ -434,6 +436,17 @@ static void prune_dcache(int count, stru
     spin_unlock(&dentry->d_lock);
    continue;
   }
+  /*
+   * If this dentry is for "my" filesystem, then I can prune it
+   * without taking the s_umount lock: either I already hold it
+   * (called from shrink_dcache_sb) or are called from some
+   * filesystem opernations and therefore cannot race with
+   * generic_shutdown_super().
+   */
+  if (have_list) {
+   prune_one_dentry(dentry);
+   continue;
+  }
   /* If the dentry was recently referenced, don't free it. */
   if (dentry->d_flags & DCACHE_REFERENCED) {
    dentry->d_flags &= ~DCACHE_REFERENCED;
@@ -443,21 +456,6 @@ static void prune_dcache(int count, stru
    continue;
   }
   /*
-   * If the dentry is not DCACHED_REFERENCED, it is time
-   * to remove it from the dcache, provided the super block is
-   * NULL (which means we are trying to reclaim memory)
-   * or this dentry belongs to the same super block that
-   * we want to shrink.
```

```
-   */
-  /*
-   * If this dentry is for "my" filesystem, then I can prune it
-   * without taking the s_umount lock (I already hold it).
-   */
-  if (have_list) {
-   prune_one_dentry(dentry);
-   continue;
-  }
-  /*
    * ...otherwise we need to be sure this filesystem isn't being
    * unmounted, otherwise we could race with
    * generic_shutdown_super(), and end up holding a reference to
@@ -483,27 +481,9 @@ static void prune_dcache(int count, stru
    list_add(&dentry->d_lru, &dentry_unused);
   dentry_stat.nr_unused++;
  }
- /* split any remaining entries back onto dentry_unused */
- if (have_list)
-  list_splice(list, dentry_unused.prev);
  spin_unlock(&dcache_lock);
 }

-/**
- * shrink_dcache_sb - shrink dcache for a superblock
- * @sb: superblock
- *
- * Shrink the dcache for the specified super block. This
- * is used to reduce the dcache presence of a file system
- * before re-mounting, and when invalidating the device
- * holding a file system.
- */
-
-void shrink_dcache_sb(struct super_block * sb)
-{
- shrink_dcache_parent(sb->s_root);
-}
-
 /*
  * destroy a single subtree of dentries for unmount
  * - see the comments on shrink_dcache_for_umount() for a description of the
@@ -706,11 +686,10 @@ positive:
  * drop the lock and return early due to latency
  * constraints.
  */
-static int select_parent(struct dentry * parent, struct list_head *new)
+static void select_parent(struct dentry * parent, struct list_head *new)
 {
```

```
  struct dentry *this_parent = parent;
  struct list_head *next;
- int found = 0;

  spin_lock(&dcache_lock);
 repeat:
@@ -732,7 +711,6 @@ resume:
    if (!atomic_read(&dentry->d_count)) {
     list_add_tail(&dentry->d_lru, new);
     dentry_stat.nr_unused++;
-    found++;
    }

    /*
@@ -740,7 +718,7 @@ resume:
     * ensures forward progress). We'll be coming back to find
     * the rest.
     */
-  if (found && need_resched())
+  if (!list_empty(new) && need_resched())
     goto out;

    /*
@@ -761,7 +739,37 @@ resume:
  }
 out:
  spin_unlock(&dcache_lock);
- return found;
+}
+
+/**
+ * select_anon - further prune the cache
+ * @sb: superblock
+ *
+ * Prune the dentries that are anonymous
+ */
+
+static void select_anon(struct super_block *sb)
+{
+ struct hlist_node *lp;
+ struct hlist_head *head = &sb->s_anon;
+
+ spin_lock(&dcache_lock);
+ hlist_for_each(lp, head) {
+  struct dentry *this = hlist_entry(lp, struct dentry, d_hash);
+  if (!list_empty(&this->d_lru)) {
+   dentry_stat.nr_unused--;
+   list_del_init(&this->d_lru);
```

```
+  }
+  /*
+   * move only zero ref count dentries to the end
+   * of list for prune_dcache
+   */
+  if (!atomic_read(&this->d_count)) {
+   list_add(&this->d_lru, &sb->s_shrink_list);
+   dentry_stat.nr_unused++;
+  }
+ }
+ spin_unlock(&dcache_lock);
 }

 /**
@@ -773,11 +781,38 @@ out:

 void shrink_dcache_parent(struct dentry * parent)
 {
- int found;
  LIST_HEAD(list);

- while ((found = select_parent(parent, &list)) != 0)
-  prune_dcache(found, &list);
+ for (;;) {
+  select_parent(parent, &list);
+  if (list_empty(&list))
+   break;
+  prune_dcache(0, &list);
+ }
+}
+
+/**
+ * shrink_dcache_sb - shrink dcache for a superblock
+ * @sb: superblock
+ *
+ * Shrink the dcache for the specified super block. This
+ * is used to reduce the dcache presence of a file system
+ * before re-mounting, and when invalidating the device
+ * holding a file system.
+ */
+
+void shrink_dcache_sb(struct super_block * sb)
+{
+ struct list_head *list;
+
+ list = &sb->s_shrink_list;
+ for (;;) {
+  select_parent(sb->s_root, list);
```

```
+  select_anon(sb);
+  if (list_empty(list))
+   break;
+  prune_dcache(0, list);
+ }
 }

 /*
--- linux-2.6.19-rc4/include/linux/fs.h.vshud1 2006-11-03 15:28:04.000000000 +0300
+++ linux-2.6.19-rc4/include/linux/fs.h 2006-11-03 14:52:16.000000000 +0300
@@ -941,6 +941,7 @@ struct super_block {
  struct hlist_head s_anon;  /* anonymous dentries for (nfs) exporting */
  struct list_head s_files;

+ struct list_head s_shrink_list;
  struct block_device *s_bdev;
  struct list_head s_instances;
  struct quota_info s_dquot; /* Diskquota specific options */

--- linux-2.6.19-rc4/fs/dcache.c.vdch 2006-11-03 14:22:54.000000000 +0300
+++ linux-2.6.19-rc4/fs/dcache.c 2006-11-03 14:57:58.000000000 +0300
@@ -171,8 +171,14 @@ repeat:
   if (d_unhashed(dentry))
   goto kill_it;
    if (list_empty(&dentry->d_lru)) {
-    dentry->d_flags |= DCACHE_REFERENCED;
-    list_add(&dentry->d_lru, &dentry_unused);
+  struct list_head *list;
+
+  list = &dentry->d_sb->s_shrink_list;
+  if (list_empty(list)) {
+   list = &dentry_unused;
+     dentry->d_flags |= DCACHE_REFERENCED;
+  }
+    list_add(&dentry->d_lru, list);
     dentry_stat.nr_unused++;
    }
   spin_unlock(&dentry->d_lock);
@@ -384,8 +390,8 @@ static void prune_one_dentry(struct dent
 /**
  * prune_dcache - shrink the dcache
  * @count: number of entries to try and free
- * @sb: if given, ignore dentries for other superblocks
- *        which are being unmounted.
+ * @list: If given, remove from this list instead of
+ *        from dentry_unused.
  *
  * Shrink the dcache. This is done when we need
```

```
  * more memory, or simply when we need to unmount
@@ -396,33 +402,27 @@ static void prune_one_dentry(struct dent
  * all the dentries are in use.
  */

-static void prune_dcache(int count, struct super_block *sb)
+static void prune_dcache(int count, struct list_head *list)
 {
+ int have_list = list != NULL;
+
  spin_lock(&dcache_lock);
- for (; count ; count--) {
+ if (!have_list)
+  /* use the dentry_unused list */
+  list = &dentry_unused;
+
+ for (; have_list || count-- ;) {
   struct dentry *dentry;
   struct list_head *tmp;
   struct rw_semaphore *s_umount;

   cond_resched_lock(&dcache_lock);

-  tmp = dentry_unused.prev;
-  if (sb) {
-   /* Try to find a dentry for this sb, but don't try
-    * too hard, if they aren't near the tail they will
-    * be moved down again soon
-    */
-   int skip = count;
-   while (skip && tmp != &dentry_unused &&
-       list_entry(tmp, struct dentry, d_lru)->d_sb != sb) {
-    skip--;
-    tmp = tmp->prev;
-   }
-  }
-  if (tmp == &dentry_unused)
+  tmp = list->prev;
+  if (tmp == list)
    break;
   list_del_init(tmp);
-  prefetch(dentry_unused.prev);
+  prefetch(list->prev);
   dentry_stat.nr_unused--;
   dentry = list_entry(tmp, struct dentry, d_lru);

@@ -436,6 +436,17 @@ static void prune_dcache(int count, stru
    spin_unlock(&dentry->d_lock);
```

```
    continue;
   }
+  /*
+   * If this dentry is for "my" filesystem, then I can prune it
+   * without taking the s_umount lock: either I already hold it
+   * (called from shrink_dcache_sb) or are called from some
+   * filesystem opernations and therefore cannot race with
+   * generic_shutdown_super().
+   */
+  if (have_list) {
+   prune_one_dentry(dentry);
+   continue;
+  }
   /* If the dentry was recently referenced, don't free it. */
   if (dentry->d_flags & DCACHE_REFERENCED) {
    dentry->d_flags &= ~DCACHE_REFERENCED;
@@ -445,21 +456,6 @@ static void prune_dcache(int count, stru
    continue;
   }
   /*
-   * If the dentry is not DCACHED_REFERENCED, it is time
-   * to remove it from the dcache, provided the super block is
-   * NULL (which means we are trying to reclaim memory)
-   * or this dentry belongs to the same super block that
-   * we want to shrink.
-   */
-  /*
-   * If this dentry is for "my" filesystem, then I can prune it
-   * without taking the s_umount lock (I already hold it).
-   */
-  if (sb && dentry->d_sb == sb) {
-   prune_one_dentry(dentry);
-   continue;
-  }
-  /*
    * ...otherwise we need to be sure this filesystem isn't being
    * unmounted, otherwise we could race with
    * generic_shutdown_super(), and end up holding a reference to
@@ -489,67 +485,6 @@ static void prune_dcache(int count, stru
 }

 /*
- * Shrink the dcache for the specified super block.
- * This allows us to unmount a device without disturbing
- * the dcache for the other devices.
- *
- * This implementation makes just two traversals of the
- * unused list.  On the first pass we move the selected
```

```
- * dentries to the most recent end, and on the second
- * pass we free them.  The second pass must restart after
- * each dput(), but since the target dentries are all at
- * the end, it's really just a single traversal.
- */
-
-/**
- * shrink_dcache_sb - shrink dcache for a superblock
- * @sb: superblock
- *
- * Shrink the dcache for the specified super block. This
- * is used to free the dcache before unmounting a file
- * system
- */
-
-void shrink_dcache_sb(struct super_block * sb)
-{
- struct list_head *tmp, *next;
- struct dentry *dentry;
-
- /*
-  * Pass one ... move the dentries for the specified
-  * superblock to the most recent end of the unused list.
-  */
- spin_lock(&dcache_lock);
- list_for_each_safe(tmp, next, &dentry_unused) {
-  dentry = list_entry(tmp, struct dentry, d_lru);
-  if (dentry->d_sb != sb)
-   continue;
-  list_move(tmp, &dentry_unused);
- }
-
- /*
-  * Pass two ... free the dentries for this superblock.
-  */
-repeat:
- list_for_each_safe(tmp, next, &dentry_unused) {
-  dentry = list_entry(tmp, struct dentry, d_lru);
-  if (dentry->d_sb != sb)
-   continue;
-  dentry_stat.nr_unused--;
-  list_del_init(tmp);
-  spin_lock(&dentry->d_lock);
-  if (atomic_read(&dentry->d_count)) {
-   spin_unlock(&dentry->d_lock);
-   continue;
-  }
-  prune_one_dentry(dentry);
```

```
-  cond_resched_lock(&dcache_lock);
-  goto repeat;
- }
- spin_unlock(&dcache_lock);
-}
-
-/*
  * destroy a single subtree of dentries for unmount
  * - see the comments on shrink_dcache_for_umount() for a description of the
  *   locking
@@ -739,7 +674,7 @@ positive:

 /*
  * Search the dentry child list for the specified parent,
- * and move any unused dentries to the end of the unused
+ * and move any unused dentries to the end of a new unused
  * list for prune_dcache(). We descend to the next level
  * whenever the d_subdirs list is non-empty and continue
  * searching.
@@ -751,11 +686,10 @@ positive:
  * drop the lock and return early due to latency
  * constraints.
  */
-static int select_parent(struct dentry * parent)
+static void select_parent(struct dentry * parent, struct list_head *new)
 {
  struct dentry *this_parent = parent;
  struct list_head *next;
- int found = 0;

  spin_lock(&dcache_lock);
 repeat:
@@ -775,9 +709,8 @@ resume:
    */
   if (!atomic_read(&dentry->d_count)) {
-   list_add_tail(&dentry->d_lru, &dentry_unused);
+   list_add_tail(&dentry->d_lru, new);
    dentry_stat.nr_unused++;
-   found++;
   }

   /*
@@ -785,7 +718,7 @@ resume:
    * ensures forward progress). We'll be coming back to find
    * the rest.
    */
-  if (found && need_resched())
```

```
+  if (!list_empty(new) && need_resched())
    goto out;

   /*
@@ -806,7 +739,37 @@ resume:
  }
 out:
  spin_unlock(&dcache_lock);
- return found;
+}
+
+/**
+ * select_anon - further prune the cache
+ * @sb: superblock
+ *
+ * Prune the dentries that are anonymous
+ */
+
+static void select_anon(struct super_block *sb)
+{
+ struct hlist_node *lp;
+ struct hlist_head *head = &sb->s_anon;
+
+ spin_lock(&dcache_lock);
+ hlist_for_each(lp, head) {
+  struct dentry *this = hlist_entry(lp, struct dentry, d_hash);
+  if (!list_empty(&this->d_lru)) {
+   dentry_stat.nr_unused--;
+   list_del_init(&this->d_lru);
+  }
+  /*
+   * move only zero ref count dentries to the end
+   * of list for prune_dcache
+   */
+  if (!atomic_read(&this->d_count)) {
+   list_add(&this->d_lru, &sb->s_shrink_list);
+   dentry_stat.nr_unused++;
+  }
+ }
+ spin_unlock(&dcache_lock);
 }

 /**
@@ -818,10 +781,38 @@ out:

 void shrink_dcache_parent(struct dentry * parent)
 {
- int found;
```

```
+ LIST_HEAD(list);

- while ((found = select_parent(parent)) != 0)
-  prune_dcache(found, parent->d_sb);
+ for (;;) {
+  select_parent(parent, &list);
+  if (list_empty(&list))
+   break;
+  prune_dcache(0, &list);
+ }
+}
+
+/**
+ * shrink_dcache_sb - shrink dcache for a superblock
+ * @sb: superblock
+ *
+ * Shrink the dcache for the specified super block. This
+ * is used to reduce the dcache presence of a file system
+ * before re-mounting, and when invalidating the device
+ * holding a file system.
+ */
+
+void shrink_dcache_sb(struct super_block * sb)
+{
+ struct list_head *list;
+
+ list = &sb->s_shrink_list;
+ for (;;) {
+  select_parent(sb->s_root, list);
+  select_anon(sb);
+  if (list_empty(list))
+   break;
+  prune_dcache(0, list);
+ }
 }

 /*
--- linux-2.6.19-rc4/include/linux/fs.h.vdch 2006-11-03 14:22:12.000000000 +0300
+++ linux-2.6.19-rc4/include/linux/fs.h 2006-11-03 14:52:16.000000000 +0300
@@ -941,6 +941,7 @@ struct super_block {
  struct hlist_head s_anon;  /* anonymous dentries for (nfs) exporting */
  struct list_head s_files;

+ struct list_head s_shrink_list;
  struct block_device *s_bdev;
  struct list_head s_instances;
  struct quota_info s_dquot; /* Diskquota specific options */
```

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by David Howells on Tue, 14 Nov 2006 09:29:42 GMT
View Forum Message <> Reply to Message

Neil Brown <neilb@suse.de> wrote:

> It seems that anon dentries can now have children (I think someone
> explained that too me - shrink_dcache_for_umount certainly suggests
> it).

That is correct. NFS can have multiple roots in its superblocks, and these take the form of anon dentries with children.

David