Subject: Re: [PATCH 1/6] fuse: general infrastructure for pages[] of variable size
Posted by Miklos Szeredi on Wed, 12 Sep 2012 16:09:01 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The patch removes inline array of FUSE_MAX_PAGES_PER_REQ page pointers from
> fuse_req. Instead of that, req->pages may now point either to small inline
> array or to an array allocated dynamically.
>
> This essentially means that all callers of fuse_request_alloc[_nofs] should
> pass the number of pages needed explicitly.
>
> The patch doesn't make any logic changes.

See comments inline.

> ---
> fs/fuse/dev.c | 40 ++++++++++++++++++++++++++++++++++++++----------
> fs/fuse/file.c | 4 ++--
> fs/fuse/fuse_i.h | 9 ++++++---
> fs/fuse/inode.c | 4 ++--
> 4 files changed, 40 insertions(+), 17 deletions(-)
>
> diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
> index 7df2b5e..c0283a1 100644
> --- a/fs/fuse/dev.c
> +++ b/fs/fuse/dev.c
> @@ -36,32 +36,52 @@ static struct fuse_conn *fuse_get_conn(struct file *file)
>
> static void fuse_request_init(struct fuse_req *req)
> {
> + struct page **pages = req->pages;
> +

Rather than this, make fuse_request_init() take 'pages' as a second
argument.

>   memset(req, 0, sizeof(*req));
>   INIT_LIST_HEAD(&req->list);
>   INIT_LIST_HEAD(&req->intr_entry);
>   init_waitqueue_head(&req->waitq);
>   atomic_set(&req->count, 1);
> +
> + req->pages = pages;
> }
>

```
> -struct fuse_req *fuse_request_alloc(void)
> +static struct fuse_req *__fuse_request_alloc(int npages, gfp_t flags)
>  {
> - struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, GFP_KERNEL);
> - if (req)
> + struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, flags);
> + if (req) {
> +  if (npages <= 1)
```

Instead of '1' use a constant (e.g. FUSE_REQ_INLINE_PAGES)

```
> +   req->pages = req->inline_pages;
> +  else
> +   req->pages = kmalloc(sizeof(struct page *) * npages,
> +          flags);
> +
> +  if (!req->pages) {
> +   kmem_cache_free(fuse_req_cachep, req);
> +   return NULL;
> +  }
> +
>    fuse_request_init(req);
> + }
>   return req;
>  }
> +
> +struct fuse_req *fuse_request_alloc(int npages)
> +{
> + return __fuse_request_alloc(npages, GFP_KERNEL);
> +}
>  EXPORT_SYMBOL_GPL(fuse_request_alloc);
>
> -struct fuse_req *fuse_request_alloc_nofs(void)
> +struct fuse_req *fuse_request_alloc_nofs(int npages)
>  {
> - struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, GFP_NOFS);
> - if (req)
> -  fuse_request_init(req);
> - return req;
> + return __fuse_request_alloc(npages, GFP_NOFS);
>  }
>
>  void fuse_request_free(struct fuse_req *req)
>  {
> + if (req->pages != req->inline_pages)
> +  kfree(req->pages);
>   kmem_cache_free(fuse_req_cachep, req);
>  }
```

>
> @@ -116,7 +136,7 @@ struct fuse_req *fuse_get_req(struct fuse_conn *fc)
> if (!fc->connected)
> goto out;
>
> - req = fuse_request_alloc();
> + req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
> err = -ENOMEM;
> if (!req)
> goto out;
> @@ -193,7 +213,7 @@ struct fuse_req *fuse_get_req_nofail(struct fuse_conn *fc, struct file
*file)
>
> atomic_inc(&fc->num_waiting);
> wait_event(fc->blocked_waitq, !fc->blocked);
> - req = fuse_request_alloc();
> + req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
> if (!req)
> req = get_reserved_req(fc, file);
>
> diff --git a/fs/fuse/file.c b/fs/fuse/file.c
> index aba15f1..7423ea4 100644
> --- a/fs/fuse/file.c
> +++ b/fs/fuse/file.c
> @@ -57,7 +57,7 @@ struct fuse_file *fuse_file_alloc(struct fuse_conn *fc)
> return NULL;
>
> ff->fc = fc;
> - ff->reserved_req = fuse_request_alloc();
> + ff->reserved_req = fuse_request_alloc(0);
> if (unlikely(!ff->reserved_req)) {
> kfree(ff);
> return NULL;
> @@ -1272,7 +1272,7 @@ static int fuse_writepage_locked(struct page *page)
>
> set_page_writeback(page);
>
> - req = fuse_request_alloc_nofs();
> + req = fuse_request_alloc_nofs(1);
> if (!req)
> goto err;
>
> diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
> index e24dd74..5e78840 100644
> --- a/fs/fuse/fuse_i.h
> +++ b/fs/fuse/fuse_i.h
> @@ -291,7 +291,10 @@ struct fuse_req {
> } misc;

>
>   /** page vector */
> - struct page *pages[FUSE_MAX_PAGES_PER_REQ];
> + struct page **pages;
> +
> + /** inline page vector */
> + struct page *inline_pages[1];
>

To defend against programming errors, I think it would be wise to add

 /** size of the 'pages' array */
 unsigned max_pages;

so that when storing pages in the array we can check whether we are
overflowing the array.

>   /** number of pages in vector */
>   unsigned num_pages;
> @@ -658,9 +661,9 @@ void fuse_ctl_cleanup(void);
> /**
>   * Allocate a request
>   */
> -struct fuse_req *fuse_request_alloc(void);
> +struct fuse_req *fuse_request_alloc(int npages);
>
> -struct fuse_req *fuse_request_alloc_nofs(void);
> +struct fuse_req *fuse_request_alloc_nofs(int npages);
>
> /**
>   * Free a request
> diff --git a/fs/fuse/inode.c b/fs/fuse/inode.c
> index ce0a283..3f399ba 100644
> --- a/fs/fuse/inode.c
> +++ b/fs/fuse/inode.c
> @@ -1027,12 +1027,12 @@ static int fuse_fill_super(struct super_block *sb, void *data, int
silent)
>   /* only now - we want root dentry with NULL ->d_op */
>   sb->s_d_op = &fuse_dentry_operations;
>
> - init_req = fuse_request_alloc();
> + init_req = fuse_request_alloc(0);
>   if (!init_req)
>   goto err_put_root;
>
>   if (is_bdev) {
> - fc->destroy_req = fuse_request_alloc();
> + fc->destroy_req = fuse_request_alloc(0);

> if (!fc->destroy_req)
> goto err_free_init_req;
> }