Subject: [PATCH 01/12] fuse: general infrastructure for pages[] of variable size
Posted by Maxim Patlasov on Fri, 26 Oct 2012 15:48:07 GMT
View Forum Message <> Reply to Message

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.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
---
 fs/fuse/dev.c | 47 ++---------
 fs/fuse/file.c | 4 ++--
 fs/fuse/fuse_i.h | 15 ++++++++++++---
 fs/fuse/inode.c | 4 ++--
 4 files changed, 50 insertions(+), 20 deletions(-)

```
diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index 8c23fa7..54c2d84 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -34,34 +34,55 @@ static struct fuse_conn *fuse_get_conn(struct file *file)
  return file->private_data;
 }

-static void fuse_request_init(struct fuse_req *req)
+static void fuse_request_init(struct fuse_req *req, struct page **pages,
+         unsigned npages)
 {
  memset(req, 0, sizeof(*req));
+ memset(pages, 0, sizeof(*pages) * npages);
  INIT_LIST_HEAD(&req->list);
  INIT_LIST_HEAD(&req->intr_entry);
  init_waitqueue_head(&req->waitq);
  atomic_set(&req->count, 1);
+ req->pages = pages;
+ req->max_pages = npages;
 }

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

```
-  fuse_request_init(req);
+ struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, flags);
+ if (req) {
+  struct page **pages;
+
+  if (npages <= FUSE_REQ_INLINE_PAGES)
+   pages = req->inline_pages;
+  else
+   pages = kmalloc(sizeof(struct page *) * npages, flags);
+
+  if (!pages) {
+   kmem_cache_free(fuse_req_cachep, req);
+   return NULL;
+  }
+
+  fuse_request_init(req, pages, npages);
+ }
  return req;
 }
+
+struct fuse_req *fuse_request_alloc(unsigned 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(unsigned 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 +137,7 @@ 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;
@@ -165,7 +186,7 @@ static void put_reserved_req(struct fuse_conn *fc, struct fuse_req *req)
  struct fuse_file *ff = file->private_data;

  spin_lock(&fc->lock);
- fuse_request_init(req);
+ fuse_request_init(req, req->pages, req->max_pages);
  BUG_ON(ff->reserved_req);
  ff->reserved_req = req;
  wake_up_all(&fc->reserved_req_waitq);
@@ -192,7 +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 78d2837..967e3e6 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..d52a2fa 100644
--- a/fs/fuse/fuse_i.h
+++ b/fs/fuse/fuse_i.h
```

```
@@ -44,6 +44,9 @@
     doing the mount will be allowed to access the filesystem */
 #define FUSE_ALLOW_OTHER         (1 << 1)

+/** Number of page pointers embedded in fuse_req */
+#define FUSE_REQ_INLINE_PAGES 1
+
 /** List of active connections */
 extern struct list_head fuse_conn_list;

@@ -291,7 +294,13 @@ struct fuse_req {
  } misc;

  /** page vector */
- struct page *pages[FUSE_MAX_PAGES_PER_REQ];
+ struct page **pages;
+
+ /** size of the 'pages' array */
+ unsigned max_pages;
+
+ /** inline page vector */
+ struct page *inline_pages[FUSE_REQ_INLINE_PAGES];

  /** number of pages in vector */
  unsigned num_pages;
@@ -658,9 +667,9 @@ void fuse_ctl_cleanup(void);
 /**
  * Allocate a request
  */
-struct fuse_req *fuse_request_alloc(void);
+struct fuse_req *fuse_request_alloc(unsigned npages);

-struct fuse_req *fuse_request_alloc_nofs(void);
+struct fuse_req *fuse_request_alloc_nofs(unsigned npages);

 /**
  * Free a request
diff --git a/fs/fuse/inode.c b/fs/fuse/inode.c
index f0eda12..0fda955 100644
--- a/fs/fuse/inode.c
+++ b/fs/fuse/inode.c
@@ -1029,12 +1029,12 @@ 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;
  }
```