Subject: [PATCH v2 00/11] fuse: optimize scatter-gather direct IO
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:31:10 GMT
View Forum Message <> Reply to Message

Hi,

Existing fuse implementation processes scatter-gather direct IO in suboptimal way: fuse_direct_IO passes iovec[] to fuse_loop_dio and the latter calls fuse_direct_read/write for each iovec from iovec[] array. Thus we have as many submitted fuse-requests as the number of elements in iovec[] array. This is pure waste of resources and affects performance negatively especially for the case of many small chunks (e.g. page-size) packed in one iovec[] array.

The patch-set amends situation in a natural way: let's simply pack as many iovec[] segments to every fuse-request as possible.

To estimate performance improvement I used slightly modified fusexmp over tmpfs (clearing O_DIRECT bit from fi->flags in xmp_open). The test opened a file with O_DIRECT, then called readv/writev in a loop. An iovec[] for readv/writev consisted of 32 segments of 4K each. The throughput on some commodity (rather feeble) server was (in MB/sec):

	original / patched
writev:	~107 / ~480
readv:	~114 / ~569

We're exploring possibility to use fuse for our own distributed storage implementation and big iovec[] arrays of many page-size chunks is typical use-case for device virtualization thread performing i/o on behalf of virtual-machine it serves.

Changed in v2:
- inline array of page pointers req->pages[] is replaced with dynamically allocated one; the number of elements is calculated a bit more intelligently than being equal to FUSE_MAX_PAGES_PER_REQ; this is done for the sake of memory economy.
- a dynamically allocated array of so-called 'page descriptors' - an offset in page plus the length of fragment - is added to fuse_req; this is done to simplify processing fuse requests covering several iov-s.

Thanks,
Maxim

Maxim Patlasov (11):
fuse: general infrastructure for pages[] of variable size
fuse: categorize fuse_get_req()

fuse: rework fuse_retrieve()
fuse: rework fuse_readpages()
fuse: rework fuse_perform_write()
fuse: rework fuse_do_ioctl()
fuse: add per-page descriptor <offset, length> to fuse_req
fuse: use req->page_descs[] for argpages cases
fuse: pass iov[] to fuse_get_user_pages()
fuse: optimize fuse_get_user_pages()
fuse: optimize __fuse_direct_io()

```
 fs/fuse/cuse.c   |    3 -
 fs/fuse/dev.c    |   96 ++++++++++++++-------
 fs/fuse/dir.c    |   39 ++++----
 fs/fuse/file.c   |  250 +++++++++++++++++++++++++++++++++++-----------------------------
 fs/fuse/fuse_i.h |   47 +++++++++--
 fs/fuse/inode.c  |    6 +
 6 files changed, 296 insertions(+), 145 deletions(-)
```

--
Signature

Subject: [PATCH 01/11] fuse: general infrastructure for pages[] of variable size
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:31:45 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    |   46 +++++++++++++++++++++++++++++++++-------------
 fs/fuse/file.c   |    4 ++--
 fs/fuse/fuse_i.h |   15 ++++++++++++---
 fs/fuse/inode.c  |    4 ++--
 4 files changed, 49 insertions(+), 20 deletions(-)

diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index 7df2b5e..1b5fc73 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
```

```
@@ -34,34 +34,54 @@ 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));
  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(int 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(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;
@@ -166,7 +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);
@@ -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..e686bf1 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(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;
  }
```

Subject: [PATCH 02/11] fuse: categorize fuse_get_req()
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:31:54 GMT
View Forum Message <> Reply to Message

The patch categorizes all fuse_get_req() invocations into two categories:
 - fuse_get_req_nopages(fc) - when caller doesn't care about req->pages
 - fuse_get_req(fc, n) - when caller need n page pointers (n > 0)

Adding fuse_get_req_nopages() helps to avoid numerous fuse_get_req(fc, 0)
scattered over code. Now it's clear from the first glance when a caller need
fuse_req with page pointers.

The patch doesn't make any logic changes. In multi-page case, it silly
allocates array of FUSE_MAX_PAGES_PER_REQ page pointers. This will be amended
by future patches.

```
Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
---
 fs/fuse/cuse.c   |    2 +-
 fs/fuse/dev.c    |   11 ++++++-----
 fs/fuse/dir.c    |   38 ++++++++++++++++++++++----------------
 fs/fuse/file.c   |   30 ++++++++++++++++++++----------
 fs/fuse/fuse_i.h |   17 ++++++++++++++---
 fs/fuse/inode.c  |    2 +-
 6 files changed, 56 insertions(+), 44 deletions(-)

diff --git a/fs/fuse/cuse.c b/fs/fuse/cuse.c
index 3426521..2f1d5dd 100644
--- a/fs/fuse/cuse.c
+++ b/fs/fuse/cuse.c
@@ -411,7 +411,7 @@ static int cuse_send_init(struct cuse_conn *cc)

  BUILD_BUG_ON(CUSE_INIT_INFO_MAX > PAGE_SIZE);

- req = fuse_get_req(fc);
+ req = fuse_get_req(fc, 1);
  if (IS_ERR(req)) {
   rc = PTR_ERR(req);
   goto err;
diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index 1b5fc73..3ad5570 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -117,7 +117,7 @@ static void fuse_req_init_context(struct fuse_req *req)
  req->in.h.pid = current->pid;
 }

-struct fuse_req *fuse_get_req(struct fuse_conn *fc)
+struct fuse_req *fuse_get_req(struct fuse_conn *fc, int npages)
 {
  struct fuse_req *req;
  sigset_t oldset;
@@ -136,7 +136,7 @@ struct fuse_req *fuse_get_req(struct fuse_conn *fc)
  if (!fc->connected)
   goto out;

- req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
+ req = fuse_request_alloc(npages);
  err = -ENOMEM;
  if (!req)
   goto out;
@@ -207,13 +207,14 @@ static void put_reserved_req(struct fuse_conn *fc, struct fuse_req
*req)
  * filesystem should not have it's own file open.  If deadlock is
```

```
  * intentional, it can still be broken by "aborting" the filesystem.
  */
-struct fuse_req *fuse_get_req_nofail(struct fuse_conn *fc, struct file *file)
+struct fuse_req *fuse_get_req_nofail_nopages(struct fuse_conn *fc,
+           struct file *file)
 {
  struct fuse_req *req;

  atomic_inc(&fc->num_waiting);
  wait_event(fc->blocked_waitq, !fc->blocked);
- req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
+ req = fuse_request_alloc(0);
  if (!req)
   req = get_reserved_req(fc, file);

@@ -1563,7 +1564,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
  unsigned int offset;
  size_t total_len = 0;

- req = fuse_get_req(fc);
+ req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
  if (IS_ERR(req))
   return PTR_ERR(req);

diff --git a/fs/fuse/dir.c b/fs/fuse/dir.c
index 324bc08..1929bfb 100644
--- a/fs/fuse/dir.c
+++ b/fs/fuse/dir.c
@@ -178,7 +178,7 @@ static int fuse_dentry_revalidate(struct dentry *entry, unsigned int flags)
    return -ECHILD;

   fc = get_fuse_conn(inode);
-  req = fuse_get_req(fc);
+  req = fuse_get_req_nopages(fc);
   if (IS_ERR(req))
    return 0;

@@ -271,7 +271,7 @@ int fuse_lookup_name(struct super_block *sb, u64 nodeid, struct qstr
*name,
  if (name->len > FUSE_NAME_MAX)
   goto out;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  err = PTR_ERR(req);
  if (IS_ERR(req))
   goto out;
@@ -391,7 +391,7 @@ static int fuse_create_open(struct inode *dir, struct dentry *entry,
```

```
  if (!forget)
   goto out_err;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  err = PTR_ERR(req);
  if (IS_ERR(req))
   goto out_put_forget_req;
@@ -592,7 +592,7 @@ static int fuse_mknod(struct inode *dir, struct dentry *entry, umode_t
mode,
 {
  struct fuse_mknod_in inarg;
  struct fuse_conn *fc = get_fuse_conn(dir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -623,7 +623,7 @@ static int fuse_mkdir(struct inode *dir, struct dentry *entry, umode_t
mode)
 {
  struct fuse_mkdir_in inarg;
  struct fuse_conn *fc = get_fuse_conn(dir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -647,7 +647,7 @@ static int fuse_symlink(struct inode *dir, struct dentry *entry,
 {
  struct fuse_conn *fc = get_fuse_conn(dir);
  unsigned len = strlen(link) + 1;
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -664,7 +664,7 @@ static int fuse_unlink(struct inode *dir, struct dentry *entry)
 {
  int err;
  struct fuse_conn *fc = get_fuse_conn(dir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -696,7 +696,7 @@ static int fuse_rmdir(struct inode *dir, struct dentry *entry)
 {
```

```
  int err;
  struct fuse_conn *fc = get_fuse_conn(dir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -723,7 +723,7 @@ static int fuse_rename(struct inode *olddir, struct dentry *oldent,
  int err;
  struct fuse_rename_in inarg;
  struct fuse_conn *fc = get_fuse_conn(olddir);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);

  if (IS_ERR(req))
   return PTR_ERR(req);
@@ -776,7 +776,7 @@ static int fuse_link(struct dentry *entry, struct inode *newdir,
  struct fuse_link_in inarg;
  struct inode *inode = entry->d_inode;
  struct fuse_conn *fc = get_fuse_conn(inode);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -848,7 +848,7 @@ static int fuse_do_getattr(struct inode *inode, struct kstat *stat,
  struct fuse_req *req;
  u64 attr_version;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1029,7 +1029,7 @@ static int fuse_access(struct inode *inode, int mask)
  if (fc->no_access)
   return 0;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1167,7 +1167,7 @@ static int fuse_readdir(struct file *file, void *dstbuf, filldir_t filldir)
  if (is_bad_inode(inode))
   return -EIO;

- req = fuse_get_req(fc);
```

```
+ req = fuse_get_req(fc, 1);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1197,7 +1197,7 @@ static char *read_link(struct dentry *dentry)
 {
  struct inode *inode = dentry->d_inode;
  struct fuse_conn *fc = get_fuse_conn(inode);
- struct fuse_req *req = fuse_get_req(fc);
+ struct fuse_req *req = fuse_get_req_nopages(fc);
  char *link;

  if (IS_ERR(req))
@@ -1410,7 +1410,7 @@ static int fuse_do_setattr(struct dentry *entry, struct iattr *attr,
  if (attr->ia_valid & ATTR_SIZE)
   is_truncate = true;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1518,7 +1518,7 @@ static int fuse_setxattr(struct dentry *entry, const char *name,
  if (fc->no_setxattr)
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1557,7 +1557,7 @@ static ssize_t fuse_getxattr(struct dentry *entry, const char *name,
  if (fc->no_getxattr)
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1609,7 +1609,7 @@ static ssize_t fuse_listxattr(struct dentry *entry, char *list, size_t size)
  if (fc->no_listxattr)
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);
```

```
@@ -1654,7 +1654,7 @@ static int fuse_removexattr(struct dentry *entry, const char *name)
  if (fc->no_removexattr)
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 7423ea4..214e13e 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -25,7 +25,7 @@ static int fuse_send_open(struct fuse_conn *fc, u64 nodeid, struct file *file,
  struct fuse_req *req;
  int err;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -368,7 +368,7 @@ static int fuse_flush(struct file *file, fl_owner_t id)
  if (fc->no_flush)
   return 0;

- req = fuse_get_req_nofail(fc, file);
+ req = fuse_get_req_nofail_nopages(fc, file);
  memset(&inarg, 0, sizeof(inarg));
  inarg.fh = ff->fh;
  inarg.lock_owner = fuse_lock_owner_id(fc, id);
@@ -436,7 +436,7 @@ int fuse_fsync_common(struct file *file, loff_t start, loff_t end,

  fuse_sync_writes(inode);

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req)) {
   err = PTR_ERR(req);
   goto out;
@@ -544,7 +544,7 @@ static int fuse_readpage(struct file *file, struct page *page)
   */
  fuse_wait_on_page_writeback(inode, page->index);

- req = fuse_get_req(fc);
+ req = fuse_get_req(fc, 1);
  err = PTR_ERR(req);
```

```
  if (IS_ERR(req))
   goto out;
@@ -657,7 +657,7 @@ static int fuse_readpages_fill(void *_data, struct page *page)
       (req->num_pages + 1) * PAGE_CACHE_SIZE > fc->max_read ||
       req->pages[req->num_pages - 1]->index + 1 != page->index)) {
   fuse_send_readpages(req, data->file);
-  data->req = req = fuse_get_req(fc);
+  data->req = req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
   if (IS_ERR(req)) {
    unlock_page(page);
    return PTR_ERR(req);
@@ -683,7 +683,7 @@ static int fuse_readpages(struct file *file, struct address_space *mapping,

  data.file = file;
  data.inode = inode;
- data.req = fuse_get_req(fc);
+ data.req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
  err = PTR_ERR(data.req);
  if (IS_ERR(data.req))
   goto out;
@@ -890,7 +890,7 @@ static ssize_t fuse_perform_write(struct file *file,
   struct fuse_req *req;
   ssize_t count;

-  req = fuse_get_req(fc);
+  req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
   if (IS_ERR(req)) {
    err = PTR_ERR(req);
    break;
@@ -1072,7 +1072,7 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
  ssize_t res = 0;
  struct fuse_req *req;

- req = fuse_get_req(fc);
+ req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1108,7 +1108,7 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
    break;
   if (count) {
    fuse_put_request(fc, req);
-   req = fuse_get_req(fc);
+   req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
    if (IS_ERR(req))
     break;
   }
@@ -1470,7 +1470,7 @@ static int fuse_getlk(struct file *file, struct file_lock *fl)
```

```
  struct fuse_lk_out outarg;
  int err;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1505,7 +1505,7 @@ static int fuse_setlk(struct file *file, struct file_lock *fl, int flock)
  if (fl->fl_flags & FL_CLOSE)
   return 0;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1574,7 +1574,7 @@ static sector_t fuse_bmap(struct address_space *mapping, sector_t
block)
  if (!inode->i_sb->s_bdev || fc->no_bmap)
   return 0;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return 0;

@@ -1872,7 +1872,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
   num_pages++;
  }

- req = fuse_get_req(fc);
+ req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
  if (IS_ERR(req)) {
   err = PTR_ERR(req);
   req = NULL;
@@ -2075,7 +2075,7 @@ unsigned fuse_file_poll(struct file *file, poll_table *wait)
   fuse_register_polled_file(fc, ff);
  }

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return POLLERR;

@@ -2193,7 +2193,7 @@ long fuse_file_fallocate(struct file *file, int mode, loff_t offset,
  if (fc->no_fallocate)
```

```
   return -EOPNOTSUPP;

- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);

diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
index e686bf1..fd69d62 100644
--- a/fs/fuse/fuse_i.h
+++ b/fs/fuse/fuse_i.h
@@ -677,14 +677,25 @@ struct fuse_req *fuse_request_alloc_nofs(int npages);
 void fuse_request_free(struct fuse_req *req);

 /**
- * Get a request, may fail with -ENOMEM
+ * Get a request, may fail with -ENOMEM,
+ * caller should specify # elements in req->pages[] explicitly
  */
-struct fuse_req *fuse_get_req(struct fuse_conn *fc);
+struct fuse_req *fuse_get_req(struct fuse_conn *fc, int npages);
+
+/**
+ * Get a request, may fail with -ENOMEM,
+ * useful for callers who doesn't use req->pages[]
+ */
+static inline struct fuse_req *fuse_get_req_nopages(struct fuse_conn *fc)
+{
+ return fuse_get_req(fc, 0);
+}

 /**
  * Gets a requests for a file operation, always succeeds
  */
-struct fuse_req *fuse_get_req_nofail(struct fuse_conn *fc, struct file *file);
+struct fuse_req *fuse_get_req_nofail_nopages(struct fuse_conn *fc,
+          struct file *file);

 /**
  * Decrement reference count of a request.  If count goes to zero free
diff --git a/fs/fuse/inode.c b/fs/fuse/inode.c
index 3f399ba..efb6144 100644
--- a/fs/fuse/inode.c
+++ b/fs/fuse/inode.c
@@ -411,7 +411,7 @@ static int fuse_statfs(struct dentry *dentry, struct kstatfs *buf)
   return 0;
  }
```

```
- req = fuse_get_req(fc);
+ req = fuse_get_req_nopages(fc);
  if (IS_ERR(req))
   return PTR_ERR(req);
```

Subject: [PATCH 03/11] fuse: rework fuse_retrieve()
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:32:03 GMT
View Forum Message <> Reply to Message

The patch reworks fuse_retrieve() to allocate only so many page pointers
as needed. The core part of the patch is the following calculation:

```
 num_pages = (num + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
```

(thanks Miklos for formula). All other changes are mostly shuffling lines.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>

```
---
 fs/fuse/dev.c |   25 +++++++++++++++++++----------
 1 files changed, 15 insertions(+), 10 deletions(-)

diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index 3ad5570..b241a7d 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -1563,13 +1563,24 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
  unsigned int num;
  unsigned int offset;
  size_t total_len = 0;
+ int num_pages;

- req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
+ offset = outarg->offset & ~PAGE_CACHE_MASK;
+ file_size = i_size_read(inode);
+
+ num = outarg->size;
+ if (outarg->offset > file_size)
+  num = 0;
+ else if (outarg->offset + num > file_size)
+  num = file_size - outarg->offset;
+
+ num_pages = (num + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
+ num_pages = min(num_pages, FUSE_MAX_PAGES_PER_REQ);
+
+ req = fuse_get_req(fc, num_pages);
  if (IS_ERR(req))
   return PTR_ERR(req);
```

```
- offset = outarg->offset & ~PAGE_CACHE_MASK;
-
  req->in.h.opcode = FUSE_NOTIFY_REPLY;
  req->in.h.nodeid = outarg->nodeid;
  req->in.numargs = 2;
@@ -1578,14 +1589,8 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
  req->end = fuse_retrieve_end;

  index = outarg->offset >> PAGE_CACHE_SHIFT;
- file_size = i_size_read(inode);
- num = outarg->size;
- if (outarg->offset > file_size)
-  num = 0;
- else if (outarg->offset + num > file_size)
-  num = file_size - outarg->offset;

- while (num && req->num_pages < FUSE_MAX_PAGES_PER_REQ) {
+ while (num && req->num_pages < num_pages) {
   struct page *page;
   unsigned int this_num;
```

Subject: [PATCH 04/11] fuse: rework fuse_readpages()
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:32:16 GMT
View Forum Message <> Reply to Message

The patch uses 'nr_pages' argument of fuse_readpages() as heuristics for the number of page pointers to allocate.

This can be improved further by taking in consideration fc->max_read and gaps between page indices, but it's not clear whether it's worthy or not.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>

```
---
 fs/fuse/file.c |   16 ++++++++++++++--
 1 files changed, 14 insertions(+), 2 deletions(-)

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 214e13e..a618371 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -641,6 +641,7 @@ struct fuse_fill_data {
  struct fuse_req *req;
  struct file *file;
  struct inode *inode;
+ unsigned nr_pages;
 };
```

```
 static int fuse_readpages_fill(void *_data, struct page *page)
@@ -656,16 +657,25 @@ static int fuse_readpages_fill(void *_data, struct page *page)
       (req->num_pages == FUSE_MAX_PAGES_PER_REQ ||
        (req->num_pages + 1) * PAGE_CACHE_SIZE > fc->max_read ||
        req->pages[req->num_pages - 1]->index + 1 != page->index)) {
+  int nr_alloc = min_t(unsigned, data->nr_pages,
+          FUSE_MAX_PAGES_PER_REQ);
    fuse_send_readpages(req, data->file);
-  data->req = req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
+  data->req = req = fuse_get_req(fc, nr_alloc);
    if (IS_ERR(req)) {
     unlock_page(page);
     return PTR_ERR(req);
    }
   }
+
+ if (WARN_ON(req->num_pages >= req->max_pages)) {
+  fuse_put_request(fc, req);
+  return -EIO;
+ }
+
   page_cache_get(page);
   req->pages[req->num_pages] = page;
   req->num_pages++;
+ data->nr_pages--;
   return 0;
  }

@@ -676,6 +686,7 @@ static int fuse_readpages(struct file *file, struct address_space *mapping,
   struct fuse_conn *fc = get_fuse_conn(inode);
   struct fuse_fill_data data;
   int err;
+ int nr_alloc = min_t(unsigned, nr_pages, FUSE_MAX_PAGES_PER_REQ);

   err = -EIO;
   if (is_bad_inode(inode))
@@ -683,7 +694,8 @@ static int fuse_readpages(struct file *file, struct address_space *mapping,

   data.file = file;
   data.inode = inode;
-  data.req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
+  data.req = fuse_get_req(fc, nr_alloc);
+  data.nr_pages = nr_pages;
   err = PTR_ERR(data.req);
   if (IS_ERR(data.req))
    goto out;
```

Subject: [PATCH 05/11] fuse: rework fuse_perform_write()
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:32:25 GMT
View Forum Message <> Reply to Message

The patch allocates as many page pointers in fuse_req as needed to cover
interval [pos .. pos+len-1]. Inline helper fuse_wr_pages() is introduced
to hide this cumbersome arithmetic.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
---
 fs/fuse/file.c | 13 ++++++++++--
 1 files changed, 11 insertions(+), 2 deletions(-)

```
diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index a618371..d72d638 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -881,11 +881,19 @@ static ssize_t fuse_fill_write_pages(struct fuse_req *req,
   if (!fc->big_writes)
    break;
  } while (iov_iter_count(ii) && count < fc->max_write &&
-   req->num_pages < FUSE_MAX_PAGES_PER_REQ && offset == 0);
+   req->num_pages < req->max_pages && offset == 0);

  return count > 0 ? count : err;
 }

+static inline unsigned fuse_wr_pages(loff_t pos, size_t len)
+{
+ return min_t(unsigned,
+       ((pos + len - 1) >> PAGE_CACHE_SHIFT) -
+       (pos >> PAGE_CACHE_SHIFT) + 1,
+       FUSE_MAX_PAGES_PER_REQ);
+}
+
 static ssize_t fuse_perform_write(struct file *file,
       struct address_space *mapping,
       struct iov_iter *ii, loff_t pos)
@@ -901,8 +909,9 @@ static ssize_t fuse_perform_write(struct file *file,
  do {
   struct fuse_req *req;
   ssize_t count;
+  unsigned nr_pages = fuse_wr_pages(pos, iov_iter_count(ii));

-  req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
+  req = fuse_get_req(fc, nr_pages);
   if (IS_ERR(req)) {
    err = PTR_ERR(req);
    break;
```

Subject: [PATCH 06/11] fuse: rework fuse_do_ioctl()
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:32:37 GMT
View Forum Message <> Reply to Message

fuse_do_ioctl() already calculates the number of pages it's going to use. It is
stored in 'num_pages' variable. So the patch simply uses it for allocating
fuse_req.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
---
fs/fuse/file.c | 2 +-
1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index d72d638..08899a6 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -1893,7 +1893,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
num_pages++;
}

- req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
+ req = fuse_get_req(fc, num_pages);
if (IS_ERR(req)) {
err = PTR_ERR(req);
req = NULL;

Subject: [PATCH 07/11] fuse: add per-page descriptor <offset, length> to fuse_req
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:32:55 GMT
View Forum Message <> Reply to Message

The ability to save page pointers along with lengths and offsets in fuse_req
will be useful to cover several iovec-s with a single fuse_req.

Per-request page_offset is removed because anybody who need it can use
req->page_descs[0].offset instead.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
---
fs/fuse/dev.c | 27 ++++++++++++++++++++--------
fs/fuse/file.c | 10 +++++-----
fs/fuse/fuse_i.h | 15 +++++++++++---
3 files changed, 36 insertions(+), 16 deletions(-)

diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c

```
index b241a7d..ea63d39 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -35,14 +35,16 @@ static struct fuse_conn *fuse_get_conn(struct file *file)
 }

 static void fuse_request_init(struct fuse_req *req, struct page **pages,
-         unsigned npages)
+         struct page_desc *page_descs, unsigned npages)
 {
  memset(req, 0, sizeof(*req));
+ memset(page_descs, 0, sizeof(*page_descs) * 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->page_descs = page_descs;
  req->max_pages = npages;
 }

@@ -51,18 +53,25 @@ static struct fuse_req *__fuse_request_alloc(int npages, gfp_t flags)
  struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, flags);
  if (req) {
   struct page **pages;
+  struct page_desc *page_descs;

-  if (npages <= FUSE_REQ_INLINE_PAGES)
+  if (npages <= FUSE_REQ_INLINE_PAGES) {
    pages = req->inline_pages;
-  else
+   page_descs = req->inline_page_descs;
+  } else {
    pages = kmalloc(sizeof(struct page *) * npages, flags);
+   page_descs = kmalloc(sizeof(struct page_desc) * npages,
+         flags);
+  }

-  if (!pages) {
+  if (!pages || !page_descs) {
+   kfree(pages);
+   kfree(page_descs);
    kmem_cache_free(fuse_req_cachep, req);
    return NULL;
   }

-  fuse_request_init(req, pages, npages);
+  fuse_request_init(req, pages, page_descs, npages);
```

```
 }
 return req;
}
@@ -82,6 +91,8 @@ void fuse_request_free(struct fuse_req *req)
{
 if (req->pages != req->inline_pages)
  kfree(req->pages);
+ if (req->page_descs != req->inline_page_descs)
+  kfree(req->page_descs);
 kmem_cache_free(fuse_req_cachep, req);
}

@@ -186,7 +197,7 @@ 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, req->pages, req->max_pages);
+ fuse_request_init(req, req->pages, req->page_descs, req->max_pages);
 BUG_ON(ff->reserved_req);
 ff->reserved_req = req;
 wake_up_all(&fc->reserved_req_waitq);
@@ -877,7 +888,7 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned nbytes,
{
 unsigned i;
 struct fuse_req *req = cs->req;
- unsigned offset = req->page_offset;
+ unsigned offset = req->page_descs[0].offset;
 unsigned count = min(nbytes, (unsigned) PAGE_SIZE - offset);

 for (i = 0; i < req->num_pages && (nbytes || zeroing); i++) {
@@ -1585,7 +1596,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
 req->in.h.nodeid = outarg->nodeid;
 req->in.numargs = 2;
 req->in.argpages = 1;
- req->page_offset = offset;
+ req->page_descs[0].offset = offset;
 req->end = fuse_retrieve_end;

 index = outarg->offset >> PAGE_CACHE_SHIFT;
diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 08899a6..b68cf3b 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -798,7 +798,7 @@ static size_t fuse_send_write_pages(struct fuse_req *req, struct file *file,

 res = fuse_send_write(req, file, pos, count, NULL);

- offset = req->page_offset;
```

```
+ offset = req->page_descs[0].offset;
  count = res;
  for (i = 0; i < req->num_pages; i++) {
   struct page *page = req->pages[i];
@@ -829,7 +829,7 @@ static ssize_t fuse_fill_write_pages(struct fuse_req *req,
  int err;

  req->in.argpages = 1;
- req->page_offset = offset;
+ req->page_descs[0].offset = offset;

  do {
   size_t tmp;
@@ -1070,14 +1070,14 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,
   return npages;

  req->num_pages = npages;
- req->page_offset = offset;
+ req->page_descs[0].offset = offset;

  if (write)
   req->in.argpages = 1;
  else
   req->out.argpages = 1;

- nbytes = (req->num_pages << PAGE_SHIFT) - req->page_offset;
+ nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;
  *nbytesp = min(*nbytesp, nbytes);

  return 0;
@@ -1314,7 +1314,7 @@ static int fuse_writepage_locked(struct page *page)
  req->in.argpages = 1;
  req->num_pages = 1;
  req->pages[0] = tmp_page;
- req->page_offset = 0;
+ req->page_descs[0].offset = 0;
  req->end = fuse_writepage_end;
  req->inode = inode;

diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
index fd69d62..fbf9379 100644
--- a/fs/fuse/fuse_i.h
+++ b/fs/fuse/fuse_i.h
@@ -203,6 +203,12 @@ struct fuse_out {
  struct fuse_arg args[3];
 };
```

```
+/** FUSE page descriptor */
+struct page_desc {
+ unsigned int length;
+ unsigned int offset;
+};
+
 /** The request state */
 enum fuse_req_state {
  FUSE_REQ_INIT = 0,
@@ -296,18 +302,21 @@ struct fuse_req {
  /** page vector */
  struct page **pages;

+ /** page-descriptor vector */
+ struct page_desc *page_descs;
+
  /** size of the 'pages' array */
  unsigned max_pages;

  /** inline page vector */
  struct page *inline_pages[FUSE_REQ_INLINE_PAGES];

+ /** inline page-descriptor vector */
+ struct page_desc inline_page_descs[FUSE_REQ_INLINE_PAGES];
+
  /** number of pages in vector */
  unsigned num_pages;

- /** offset of data on first page */
- unsigned page_offset;
-
  /** File used in the request (or NULL) */
  struct fuse_file *ff;
```

Subject: [PATCH 08/11] fuse: use req->page_descs[] for argpages cases
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:33:03 GMT
View Forum Message <> Reply to Message

Previously, anyone who set flag 'argpages' only filled req->pages[] and set
per-request page_offset. This patch re-works all cases where argpages=1 to
fill req->page_descs[] properly.

Having req->page_descs[] filled properly allows to re-work fuse_copy_pages()
to copy page fragments described by req->page_descs[]. This will be useful
for next patches optimizing direct_IO.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>

```
---
 fs/fuse/cuse.c |    1 +
 fs/fuse/dev.c  |    7 +++----
 fs/fuse/dir.c  |    1 +
 fs/fuse/file.c |   20 ++++++++++++++++++++
 4 files changed, 25 insertions(+), 4 deletions(-)

diff --git a/fs/fuse/cuse.c b/fs/fuse/cuse.c
index 2f1d5dd..38efe69 100644
--- a/fs/fuse/cuse.c
+++ b/fs/fuse/cuse.c
@@ -441,6 +441,7 @@ static int cuse_send_init(struct cuse_conn *cc)
  req->out.argvar = 1;
  req->out.argpages = 1;
  req->pages[0] = page;
+ req->page_descs[0].length = req->out.args[1].size;
  req->num_pages = 1;
  req->end = cuse_process_init_reply;
  fuse_request_send_background(fc, req);
diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index ea63d39..21b6272 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -888,11 +888,11 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned
nbytes,
 {
  unsigned i;
  struct fuse_req *req = cs->req;
- unsigned offset = req->page_descs[0].offset;
- unsigned count = min(nbytes, (unsigned) PAGE_SIZE - offset);

  for (i = 0; i < req->num_pages && (nbytes || zeroing); i++) {
   int err;
+  unsigned offset = req->page_descs[i].offset;
+  unsigned count = min(nbytes, req->page_descs[i].length);

   err = fuse_copy_page(cs, &req->pages[i], offset, count,
          zeroing);
@@ -900,8 +900,6 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned nbytes,
    return err;

   nbytes -= count;
-  count = min(nbytes, (unsigned) PAGE_SIZE);
-  offset = 0;
  }
  return 0;
 }
@@ -1611,6 +1609,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
```

```
   this_num = min_t(unsigned, num, PAGE_CACHE_SIZE - offset);
   req->pages[req->num_pages] = page;
+  req->page_descs[req->num_pages].length = this_num;
   req->num_pages++;

   num -= this_num;
diff --git a/fs/fuse/dir.c b/fs/fuse/dir.c
index 1929bfb..95e4016 100644
--- a/fs/fuse/dir.c
+++ b/fs/fuse/dir.c
@@ -1179,6 +1179,7 @@ static int fuse_readdir(struct file *file, void *dstbuf, filldir_t filldir)
  req->out.argpages = 1;
  req->num_pages = 1;
  req->pages[0] = page;
+ req->page_descs[0].length = PAGE_SIZE;
  fuse_read_fill(req, file, file->f_pos, PAGE_SIZE, FUSE_READDIR);
  fuse_request_send(fc, req);
  nbytes = req->out.args[0].size;
diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index b68cf3b..8a1f833 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -555,6 +555,7 @@ static int fuse_readpage(struct file *file, struct page *page)
  req->out.argpages = 1;
  req->num_pages = 1;
  req->pages[0] = page;
+ req->page_descs[0].length = count;
  num_read = fuse_send_read(req, file, pos, count, NULL);
  err = req->out.h.error;
  fuse_put_request(fc, req);
@@ -674,6 +675,7 @@ static int fuse_readpages_fill(void *_data, struct page *page)

  page_cache_get(page);
  req->pages[req->num_pages] = page;
+ req->page_descs[req->num_pages].length = PAGE_SIZE;
  req->num_pages++;
  data->nr_pages--;
  return 0;
@@ -869,6 +871,7 @@ static ssize_t fuse_fill_write_pages(struct fuse_req *req,

   err = 0;
   req->pages[req->num_pages] = page;
+  req->page_descs[req->num_pages].length = tmp;
   req->num_pages++;

   iov_iter_advance(ii, tmp);
@@ -1044,6 +1047,15 @@ static void fuse_release_user_pages(struct fuse_req *req, int write)
```

```
 }
}

+static inline void fuse_page_descs_length_init(struct fuse_req *req)
+{
+ int i;
+
+ for (i = 0; i < req->num_pages; i++)
+  req->page_descs[i].length = PAGE_SIZE -
+   req->page_descs[i].offset;
+}
+
 static int fuse_get_user_pages(struct fuse_req *req, const char __user *buf,
           size_t *nbytesp, int write)
 {
@@ -1071,6 +1083,7 @@ static int fuse_get_user_pages(struct fuse_req *req, const char __user
*buf,

  req->num_pages = npages;
  req->page_descs[0].offset = offset;
+ fuse_page_descs_length_init(req);

  if (write)
   req->in.argpages = 1;
@@ -1078,6 +1091,11 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,
   req->out.argpages = 1;

  nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;
+
+ if (*nbytesp < nbytes)
+  req->page_descs[req->num_pages - 1].length -=
+   nbytes - *nbytesp;
+
  *nbytesp = min(*nbytesp, nbytes);

  return 0;
@@ -1315,6 +1333,7 @@ static int fuse_writepage_locked(struct page *page)
  req->num_pages = 1;
  req->pages[0] = tmp_page;
  req->page_descs[0].offset = 0;
+ req->page_descs[0].length = PAGE_SIZE;
  req->end = fuse_writepage_end;
  req->inode = inode;

@@ -1901,6 +1920,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
  }
```

```
  memcpy(req->pages, pages, sizeof(req->pages[0]) * num_pages);
  req->num_pages = num_pages;
+ fuse_page_descs_length_init(req);

  /* okay, let's send it to the client */
  req->in.h.opcode = FUSE_IOCTL;
```

Subject: [PATCH 09/11] fuse: pass iov[] to fuse_get_user_pages()
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:33:12 GMT
View Forum Message <> Reply to Message

The patch makes preliminary work for the next patch optimizing scatter-gather
direct IO. The idea is to allow fuse_get_user_pages() to pack as many iov-s
to each fuse request as possible. So, here we only rework all related
call-paths to carry iov[] from fuse_direct_IO() to fuse_get_user_pages().

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>

```
---
 fs/file.c |  108 +++++++++++++++++++++++++++++-----------------------
 1 files changed, 55 insertions(+), 53 deletions(-)

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 8a1f833..db9efb5 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -1056,11 +1056,15 @@ static inline void fuse_page_descs_length_init(struct fuse_req *req)
    req->page_descs[i].offset;
 }

-static int fuse_get_user_pages(struct fuse_req *req, const char __user *buf,
+static int fuse_get_user_pages(struct fuse_req *req,
+          const struct iovec **iov_pp,
+          unsigned long *nr_segs_p,
+          size_t *iov_offset_p,
          size_t *nbytesp, int write)
 {
  size_t nbytes = *nbytesp;
- unsigned long user_addr = (unsigned long) buf;
+ size_t frag_size = min_t(size_t, nbytes, (*iov_pp)->iov_len - *iov_offset_p);
+ unsigned long user_addr = (unsigned long)(*iov_pp)->iov_base + *iov_offset_p;
  unsigned offset = user_addr & ~PAGE_MASK;
  int npages;

@@ -1071,10 +1075,13 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,
   else
    req->out.args[0].value = (void *) user_addr;
```

```
+  (*iov_pp)++;
+  (*nr_segs_p)--;
+  *nbytesp = frag_size;
   return 0;
  }

- nbytes = min_t(size_t, nbytes, FUSE_MAX_PAGES_PER_REQ << PAGE_SHIFT);
+ nbytes = min_t(size_t, frag_size, FUSE_MAX_PAGES_PER_REQ << PAGE_SHIFT);
  npages = (nbytes + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
  npages = clamp(npages, 1, FUSE_MAX_PAGES_PER_REQ);
  npages = get_user_pages_fast(user_addr, npages, !write, req->pages);
@@ -1092,17 +1099,26 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,

  nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;

- if (*nbytesp < nbytes)
+ if (frag_size < nbytes)
   req->page_descs[req->num_pages - 1].length -=
-   nbytes - *nbytesp;
+   nbytes - frag_size;

- *nbytesp = min(*nbytesp, nbytes);
+ *nbytesp = min(frag_size, nbytes);
+
+ if (*nbytesp < (*iov_pp)->iov_len - *iov_offset_p) {
+  *iov_offset_p += *nbytesp;
+ } else {
+  (*iov_pp)++;
+  (*nr_segs_p)--;
+  *iov_offset_p = 0;
+ }

  return 0;
 }

-ssize_t fuse_direct_io(struct file *file, const char __user *buf,
-         size_t count, loff_t *ppos, int write)
+static ssize_t __fuse_direct_io(struct file *file, const struct iovec *iov,
+    unsigned long nr_segs, size_t count,
+    loff_t *ppos, int write)
 {
  struct fuse_file *ff = file->private_data;
  struct fuse_conn *fc = ff->fc;
@@ -1110,6 +1126,7 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
  loff_t pos = *ppos;
  ssize_t res = 0;
```

```
  struct fuse_req *req;
+ size_t iov_offset = 0;

  req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
  if (IS_ERR(req))
@@ -1119,7 +1136,8 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
   size_t nres;
   fl_owner_t owner = current->files;
   size_t nbytes = min(count, nmax);
-  int err = fuse_get_user_pages(req, buf, &nbytes, write);
+  int err = fuse_get_user_pages(req, &iov, &nr_segs, &iov_offset,
+            &nbytes, write);
   if (err) {
    res = err;
    break;
@@ -1142,7 +1160,6 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
   count -= nres;
   res += nres;
   pos += nres;
-  buf += nres;
   if (nres != nbytes)
    break;
   if (count) {
@@ -1159,10 +1176,17 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,

  return res;
 }
+
+ssize_t fuse_direct_io(struct file *file, const char __user *buf,
+         size_t count, loff_t *ppos, int write)
+{
+ struct iovec iov = { .iov_base = (void *)buf, .iov_len = count };
+ return __fuse_direct_io(file, &iov, 1, count, ppos, write);
+}
 EXPORT_SYMBOL_GPL(fuse_direct_io);

-static ssize_t fuse_direct_read(struct file *file, char __user *buf,
-         size_t count, loff_t *ppos)
+static ssize_t __fuse_direct_read(struct file *file, const struct iovec *iov,
+       unsigned long nr_segs, loff_t *ppos)
 {
  ssize_t res;
  struct inode *inode = file->f_path.dentry->d_inode;
@@ -1170,22 +1194,31 @@ static ssize_t fuse_direct_read(struct file *file, char __user *buf,
  if (is_bad_inode(inode))
   return -EIO;

- res = fuse_direct_io(file, buf, count, ppos, 0);
```

```
+ res = __fuse_direct_io(file, iov, nr_segs, iov_length(iov, nr_segs),
+          ppos, 0);

  fuse_invalidate_attr(inode);

  return res;
 }

-static ssize_t __fuse_direct_write(struct file *file, const char __user *buf,
-       size_t count, loff_t *ppos)
+static ssize_t fuse_direct_read(struct file *file, char __user *buf,
+         size_t count, loff_t *ppos)
+{
+ struct iovec iov = { .iov_base = (void *)buf, .iov_len = count };
+ return __fuse_direct_read(file, &iov, 1, ppos);
+}
+
+static ssize_t __fuse_direct_write(struct file *file, const struct iovec *iov,
+       unsigned long nr_segs, loff_t *ppos)
 {
  struct inode *inode = file->f_path.dentry->d_inode;
+ size_t count = iov_length(iov, nr_segs);
  ssize_t res;

  res = generic_write_checks(file, ppos, &count, 0);
  if (!res) {
-  res = fuse_direct_io(file, buf, count, ppos, 1);
+  res = __fuse_direct_io(file, iov, nr_segs, count, ppos, 1);
    if (res > 0)
     fuse_write_update_size(inode, *ppos);
  }
@@ -1198,6 +1231,7 @@ static ssize_t __fuse_direct_write(struct file *file, const char __user
*buf,
 static ssize_t fuse_direct_write(struct file *file, const char __user *buf,
      size_t count, loff_t *ppos)
 {
+ struct iovec iov = { .iov_base = (void *)buf, .iov_len = count };
  struct inode *inode = file->f_path.dentry->d_inode;
  ssize_t res;

@@ -1206,7 +1240,7 @@ static ssize_t fuse_direct_write(struct file *file, const char __user *buf,

  /* Don't allow parallel writes to the same file */
  mutex_lock(&inode->i_mutex);
- res = __fuse_direct_write(file, buf, count, ppos);
+ res = __fuse_direct_write(file, &iov, 1, ppos);
  mutex_unlock(&inode->i_mutex);
```

```
  return res;
@@ -2166,41 +2200,6 @@ int fuse_notify_poll_wakeup(struct fuse_conn *fc,
  return 0;
 }

-static ssize_t fuse_loop_dio(struct file *filp, const struct iovec *iov,
-        unsigned long nr_segs, loff_t *ppos, int rw)
-{
- const struct iovec *vector = iov;
- ssize_t ret = 0;
-
- while (nr_segs > 0) {
-  void __user *base;
-  size_t len;
-  ssize_t nr;
-
-  base = vector->iov_base;
-  len = vector->iov_len;
-  vector++;
-  nr_segs--;
-
-  if (rw == WRITE)
-   nr = __fuse_direct_write(filp, base, len, ppos);
-  else
-   nr = fuse_direct_read(filp, base, len, ppos);
-
-  if (nr < 0) {
-   if (!ret)
-    ret = nr;
-   break;
-  }
-  ret += nr;
-  if (nr != len)
-   break;
- }
-
- return ret;
-}
-
-
 static ssize_t
 fuse_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
    loff_t offset, unsigned long nr_segs)
@@ -2212,7 +2211,10 @@ fuse_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
  file = iocb->ki_filp;
  pos = offset;

- ret = fuse_loop_dio(file, iov, nr_segs, &pos, rw);
```

```
+ if (rw == WRITE)
+  ret = __fuse_direct_write(file, iov, nr_segs, &pos);
+ else
+  ret = __fuse_direct_read(file, iov, nr_segs, &pos);

  return ret;
 }
```

Subject: [PATCH 10/11] fuse: optimize fuse_get_user_pages()
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:33:22 GMT
View Forum Message <> Reply to Message

Let fuse_get_user_pages() pack as many iov-s to a single fuse_req as
possible. This is very beneficial in case of iov[] consisting of many
iov-s of relatively small sizes (e.g. PAGE_SIZE).

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
---
 fs/fuse/file.c | 94 ++++++++++++++++++++++++++++++++++++++----------------
 1 files changed, 58 insertions(+), 36 deletions(-)

```
diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index db9efb5..5b0fa5d 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -1047,13 +1047,24 @@ static void fuse_release_user_pages(struct fuse_req *req, int write)
  }
 }

-static inline void fuse_page_descs_length_init(struct fuse_req *req)
+static inline void fuse_page_descs_length_init(struct fuse_req *req,
+            unsigned i, int n)
 {
- int i;
+ while (n-- > 0)
+  req->page_descs[i + n].length = PAGE_SIZE -
+    req->page_descs[i + n].offset;
+}

- for (i = 0; i < req->num_pages; i++)
-  req->page_descs[i].length = PAGE_SIZE -
-    req->page_descs[i].offset;
+static inline unsigned long fuse_get_ua(const struct iovec *iov,
+     size_t iov_offset)
+{
+ return (unsigned long)iov->iov_base + iov_offset;
+}
```

```
+
+static inline size_t fuse_get_fr_sz(const struct iovec *iov, size_t iov_offset,
+        size_t max_size)
+{
+ return min_t(size_t, iov->iov_len - iov_offset, max_size);
 }

 static int fuse_get_user_pages(struct fuse_req *req,
@@ -1062,14 +1073,12 @@ static int fuse_get_user_pages(struct fuse_req *req,
          size_t *iov_offset_p,
          size_t *nbytesp, int write)
 {
- size_t nbytes = *nbytesp;
- size_t frag_size = min_t(size_t, nbytes, (*iov_pp)->iov_len - *iov_offset_p);
- unsigned long user_addr = (unsigned long)(*iov_pp)->iov_base + *iov_offset_p;
- unsigned offset = user_addr & ~PAGE_MASK;
- int npages;
+ size_t nbytes = 0;  /* # bytes already packed in req */

  /* Special case for kernel I/O: can copy directly into the buffer */
  if (segment_eq(get_fs(), KERNEL_DS)) {
+  unsigned long user_addr = fuse_get_ua(*iov_pp, *iov_offset_p);
+
   if (write)
    req->in.args[1].value = (void *) user_addr;
   else
@@ -1077,42 +1086,55 @@ static int fuse_get_user_pages(struct fuse_req *req,

   (*iov_pp)++;
   (*nr_segs_p)--;
-  *nbytesp = frag_size;
+  *nbytesp = fuse_get_fr_sz(*iov_pp, *iov_offset_p, *nbytesp);
   return 0;
  }

- nbytes = min_t(size_t, frag_size, FUSE_MAX_PAGES_PER_REQ << PAGE_SHIFT);
- npages = (nbytes + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
- npages = clamp(npages, 1, FUSE_MAX_PAGES_PER_REQ);
- npages = get_user_pages_fast(user_addr, npages, !write, req->pages);
- if (npages < 0)
-  return npages;
+ while (nbytes < *nbytesp && req->num_pages < FUSE_MAX_PAGES_PER_REQ) {
+  int npages;
+  unsigned long user_addr = fuse_get_ua(*iov_pp, *iov_offset_p);
+  unsigned offset = user_addr & ~PAGE_MASK;
+  size_t frag_size = fuse_get_fr_sz(*iov_pp, *iov_offset_p,
+         *nbytesp - nbytes);
```

```
- req->num_pages = npages;
- req->page_descs[0].offset = offset;
- fuse_page_descs_length_init(req);
+  int n = FUSE_MAX_PAGES_PER_REQ - req->num_pages;
+  frag_size = min_t(size_t, frag_size, n << PAGE_SHIFT);

- if (write)
-  req->in.argpages = 1;
- else
-  req->out.argpages = 1;
+  npages = (frag_size + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
+  npages = clamp(npages, 1, n);

- nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;
+  npages = get_user_pages_fast(user_addr, npages, !write,
+         &req->pages[req->num_pages]);
+  if (npages < 0)
+   return npages;

- if (frag_size < nbytes)
-  req->page_descs[req->num_pages - 1].length -=
-   nbytes - frag_size;
+  frag_size = min_t(size_t, frag_size,
+      (npages << PAGE_SHIFT) - offset);
+  nbytes += frag_size;

- *nbytesp = min(frag_size, nbytes);
+  if (frag_size < (*iov_pp)->iov_len - *iov_offset_p) {
+   *iov_offset_p += frag_size;
+  } else {
+   (*iov_pp)++;
+   (*nr_segs_p)--;
+   *iov_offset_p = 0;
+  }

- if (*nbytesp < (*iov_pp)->iov_len - *iov_offset_p) {
-  *iov_offset_p += *nbytesp;
- } else {
-  (*iov_pp)++;
-  (*nr_segs_p)--;
-  *iov_offset_p = 0;
+  req->page_descs[req->num_pages].offset = offset;
+  fuse_page_descs_length_init(req, req->num_pages, npages);
+
+  req->num_pages += npages;
+  req->page_descs[req->num_pages - 1].length -=
+   (npages << PAGE_SHIFT) - offset - frag_size;
  }
```

```
+ if (write)
+  req->in.argpages = 1;
+ else
+  req->out.argpages = 1;
+
+ *nbytesp = nbytes;
+
  return 0;
 }

@@ -1954,7 +1976,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
  }
  memcpy(req->pages, pages, sizeof(req->pages[0]) * num_pages);
  req->num_pages = num_pages;
- fuse_page_descs_length_init(req);
+ fuse_page_descs_length_init(req, 0, req->num_pages);

  /* okay, let's send it to the client */
  req->in.h.opcode = FUSE_IOCTL;
```

Subject: [PATCH 11/11] fuse: optimize __fuse_direct_io()
Posted by Maxim Patlasov on Wed, 19 Sep 2012 16:33:31 GMT
View Forum Message <> Reply to Message

__fuse_direct_io() allocates fuse-requests by calling fuse_get_req(fc, n). The
patch calculates 'n' based on iov[] array. This is useful because allocating
FUSE_MAX_PAGES_PER_REQ page pointers and descriptors for each fuse request
would be waste of memory in case of iov-s of smaller size.

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>

```
---
 fs/fuse/file.c |   27 +++++++++++++++++++++----
 1 files changed, 23 insertions(+), 4 deletions(-)

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 5b0fa5d..5ecab19 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -1090,14 +1090,14 @@ static int fuse_get_user_pages(struct fuse_req *req,
   return 0;
  }

- while (nbytes < *nbytesp && req->num_pages < FUSE_MAX_PAGES_PER_REQ) {
+ while (nbytes < *nbytesp && req->num_pages < req->max_pages) {
   int npages;
```

```
   unsigned long user_addr = fuse_get_ua(*iov_pp, *iov_offset_p);
   unsigned offset = user_addr & ~PAGE_MASK;
   size_t frag_size = fuse_get_fr_sz(*iov_pp, *iov_offset_p,
         *nbytesp - nbytes);

-  int n = FUSE_MAX_PAGES_PER_REQ - req->num_pages;
+  int n = req->max_pages - req->num_pages;
   frag_size = min_t(size_t, frag_size, n << PAGE_SHIFT);

   npages = (frag_size + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
@@ -1138,6 +1138,24 @@ static int fuse_get_user_pages(struct fuse_req *req,
  return 0;
 }

+static inline int fuse_iov_npages(const struct iovec *iov,
+       unsigned long nr_segs, size_t iov_offset)
+{
+ int npages = 0;
+
+ while (nr_segs-- > 0 && npages < FUSE_MAX_PAGES_PER_REQ) {
+  unsigned long user_addr = fuse_get_ua(iov, iov_offset);
+  unsigned offset = user_addr & ~PAGE_MASK;
+  size_t frag_size = iov->iov_len - iov_offset;
+
+  npages += (frag_size + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
+  iov++;
+  iov_offset = 0;
+ }
+
+ return min(npages, FUSE_MAX_PAGES_PER_REQ);
+}
+
 static ssize_t __fuse_direct_io(struct file *file, const struct iovec *iov,
      unsigned long nr_segs, size_t count,
      loff_t *ppos, int write)
@@ -1150,7 +1168,7 @@ static ssize_t __fuse_direct_io(struct file *file, const struct iovec *iov,
  struct fuse_req *req;
  size_t iov_offset = 0;

- req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
+ req = fuse_get_req(fc, fuse_iov_npages(iov, nr_segs, iov_offset));
  if (IS_ERR(req))
   return PTR_ERR(req);

@@ -1186,7 +1204,8 @@ static ssize_t __fuse_direct_io(struct file *file, const struct iovec *iov,
    break;
   if (count) {
    fuse_put_request(fc, req);
```

```
-   req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
+   req = fuse_get_req(fc,
+    fuse_iov_npages(iov, nr_segs, iov_offset));
    if (IS_ERR(req))
     break;
   }
```

Subject: [PATCH 07/11] fuse: add per-page descriptor <offset, length> to fuse_req (v2)
Posted by Maxim Patlasov on Mon, 01 Oct 2012 11:20:00 GMT
View Forum Message <> Reply to Message

The ability to save page pointers along with lengths and offsets in fuse_req
will be useful to cover several iovec-s with a single fuse_req.

Per-request page_offset is removed because anybody who need it can use
req->page_descs[0].offset instead.

Changed in v2:
 - replaced structure page_desc with fuse_page_desc

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
---
 fs/fuse/dev.c | 26 ++++++++++++++++++++-------
 fs/fuse/file.c | 10 +++++-----
 fs/fuse/fuse_i.h | 15 ++++++++++++---
 3 files changed, 36 insertions(+), 15 deletions(-)

```
diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
index b241a7d..72ad962 100644
--- a/fs/fuse/dev.c
+++ b/fs/fuse/dev.c
@@ -35,14 +35,17 @@ static struct fuse_conn *fuse_get_conn(struct file *file)
 }

 static void fuse_request_init(struct fuse_req *req, struct page **pages,
+         struct fuse_page_desc *page_descs,
         unsigned npages)
 {
  memset(req, 0, sizeof(*req));
+ memset(page_descs, 0, sizeof(*page_descs) * 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->page_descs = page_descs;
```

```
  req->max_pages = npages;
 }

@@ -51,18 +54,25 @@ static struct fuse_req *__fuse_request_alloc(int npages, gfp_t flags)
   struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, flags);
   if (req) {
    struct page **pages;
+   struct fuse_page_desc *page_descs;

-   if (npages <= FUSE_REQ_INLINE_PAGES)
+   if (npages <= FUSE_REQ_INLINE_PAGES) {
     pages = req->inline_pages;
-   else
+    page_descs = req->inline_page_descs;
+   } else {
     pages = kmalloc(sizeof(struct page *) * npages, flags);
+    page_descs = kmalloc(sizeof(struct fuse_page_desc) *
+           npages, flags);
+   }

-   if (!pages) {
+   if (!pages || !page_descs) {
+    kfree(pages);
+    kfree(page_descs);
     kmem_cache_free(fuse_req_cachep, req);
     return NULL;
    }

-   fuse_request_init(req, pages, npages);
+   fuse_request_init(req, pages, page_descs, npages);
   }
   return req;
 }
@@ -82,6 +92,8 @@ void fuse_request_free(struct fuse_req *req)
 {
   if (req->pages != req->inline_pages)
    kfree(req->pages);
+ if (req->page_descs != req->inline_page_descs)
+   kfree(req->page_descs);
   kmem_cache_free(fuse_req_cachep, req);
 }

@@ -186,7 +198,7 @@ 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, req->pages, req->max_pages);
+ fuse_request_init(req, req->pages, req->page_descs, req->max_pages);
```

```
  BUG_ON(ff->reserved_req);
  ff->reserved_req = req;
  wake_up_all(&fc->reserved_req_waitq);
@@ -877,7 +889,7 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned nbytes,
 {
  unsigned i;
  struct fuse_req *req = cs->req;
- unsigned offset = req->page_offset;
+ unsigned offset = req->page_descs[0].offset;
  unsigned count = min(nbytes, (unsigned) PAGE_SIZE - offset);

  for (i = 0; i < req->num_pages && (nbytes || zeroing); i++) {
@@ -1585,7 +1597,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
  req->in.h.nodeid = outarg->nodeid;
  req->in.numargs = 2;
  req->in.argpages = 1;
- req->page_offset = offset;
+ req->page_descs[0].offset = offset;
  req->end = fuse_retrieve_end;

  index = outarg->offset >> PAGE_CACHE_SHIFT;
diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index 08899a6..b68cf3b 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -798,7 +798,7 @@ static size_t fuse_send_write_pages(struct fuse_req *req, struct file *file,

  res = fuse_send_write(req, file, pos, count, NULL);

- offset = req->page_offset;
+ offset = req->page_descs[0].offset;
  count = res;
  for (i = 0; i < req->num_pages; i++) {
   struct page *page = req->pages[i];
@@ -829,7 +829,7 @@ static ssize_t fuse_fill_write_pages(struct fuse_req *req,
  int err;

  req->in.argpages = 1;
- req->page_offset = offset;
+ req->page_descs[0].offset = offset;

  do {
   size_t tmp;
@@ -1070,14 +1070,14 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,
   return npages;

  req->num_pages = npages;
```

```
- req->page_offset = offset;
+ req->page_descs[0].offset = offset;

  if (write)
   req->in.argpages = 1;
  else
   req->out.argpages = 1;

- nbytes = (req->num_pages << PAGE_SHIFT) - req->page_offset;
+ nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;
  *nbytesp = min(*nbytesp, nbytes);

  return 0;
@@ -1314,7 +1314,7 @@ static int fuse_writepage_locked(struct page *page)
  req->in.argpages = 1;
  req->num_pages = 1;
  req->pages[0] = tmp_page;
- req->page_offset = 0;
+ req->page_descs[0].offset = 0;
  req->end = fuse_writepage_end;
  req->inode = inode;

diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
index fd69d62..c07e454 100644
--- a/fs/fuse/fuse_i.h
+++ b/fs/fuse/fuse_i.h
@@ -203,6 +203,12 @@ struct fuse_out {
  struct fuse_arg args[3];
 };

+/** FUSE page descriptor */
+struct fuse_page_desc {
+ unsigned int length;
+ unsigned int offset;
+};
+
 /** The request state */
 enum fuse_req_state {
  FUSE_REQ_INIT = 0,
@@ -296,18 +302,21 @@ struct fuse_req {
  /** page vector */
  struct page **pages;

+ /** page-descriptor vector */
+ struct fuse_page_desc *page_descs;
+
  /** size of the 'pages' array */
  unsigned max_pages;
```

```
 /** inline page vector */
 struct page *inline_pages[FUSE_REQ_INLINE_PAGES];

+ /** inline page-descriptor vector */
+ struct fuse_page_desc inline_page_descs[FUSE_REQ_INLINE_PAGES];
+
 /** number of pages in vector */
 unsigned num_pages;

- /** offset of data on first page */
- unsigned page_offset;
-
 /** File used in the request (or NULL) */
 struct fuse_file *ff;
```

Subject: [PATCH 10/11] fuse: optimize fuse_get_user_pages() - v2
Posted by Maxim Patlasov on Mon, 15 Oct 2012 12:03:51 GMT

View Forum Message <> Reply to Message

Let fuse_get_user_pages() pack as many iov-s to a single fuse_req as
possible. This is very beneficial in case of iov[] consisting of many
iov-s of relatively small sizes (e.g. PAGE_SIZE).

Changed in v2:
 - renamed local vars in fuse_page_descs_length_init() to be more readable

Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>

```
---
 fs/fuse/file.c |   94 +++++++++++++++++++++++++++++++++++---------------------
 1 files changed, 58 insertions(+), 36 deletions(-)

diff --git a/fs/fuse/file.c b/fs/fuse/file.c
index db9efb5..4d30697 100644
--- a/fs/fuse/file.c
+++ b/fs/fuse/file.c
@@ -1047,13 +1047,24 @@ static void fuse_release_user_pages(struct fuse_req *req, int write)
  }
 }

-static inline void fuse_page_descs_length_init(struct fuse_req *req)
+static inline void fuse_page_descs_length_init(struct fuse_req *req,
+		unsigned index, int nr_pages)
 {
- int i;
+ while (nr_pages-- > 0)
+  req->page_descs[index + nr_pages].length = PAGE_SIZE -
```

```
+   req->page_descs[index + nr_pages].offset;
+}

- for (i = 0; i < req->num_pages; i++)
-  req->page_descs[i].length = PAGE_SIZE -
-   req->page_descs[i].offset;
+static inline unsigned long fuse_get_ua(const struct iovec *iov,
+     size_t iov_offset)
+{
+ return (unsigned long)iov->iov_base + iov_offset;
+}
+
+static inline size_t fuse_get_fr_sz(const struct iovec *iov, size_t iov_offset,
+        size_t max_size)
+{
+ return min_t(size_t, iov->iov_len - iov_offset, max_size);
 }

 static int fuse_get_user_pages(struct fuse_req *req,
@@ -1062,14 +1073,12 @@ static int fuse_get_user_pages(struct fuse_req *req,
          size_t *iov_offset_p,
          size_t *nbytesp, int write)
 {
- size_t nbytes = *nbytesp;
- size_t frag_size = min_t(size_t, nbytes, (*iov_pp)->iov_len - *iov_offset_p);
- unsigned long user_addr = (unsigned long)(*iov_pp)->iov_base + *iov_offset_p;
- unsigned offset = user_addr & ~PAGE_MASK;
- int npages;
+ size_t nbytes = 0;  /* # bytes already packed in req */

  /* Special case for kernel I/O: can copy directly into the buffer */
  if (segment_eq(get_fs(), KERNEL_DS)) {
+  unsigned long user_addr = fuse_get_ua(*iov_pp, *iov_offset_p);
+
   if (write)
    req->in.args[1].value = (void *) user_addr;
   else
@@ -1077,42 +1086,55 @@ static int fuse_get_user_pages(struct fuse_req *req,

   (*iov_pp)++;
   (*nr_segs_p)--;
-  *nbytesp = frag_size;
+  *nbytesp = fuse_get_fr_sz(*iov_pp, *iov_offset_p, *nbytesp);
   return 0;
  }

- nbytes = min_t(size_t, frag_size, FUSE_MAX_PAGES_PER_REQ << PAGE_SHIFT);
- npages = (nbytes + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
```

```
- npages = clamp(npages, 1, FUSE_MAX_PAGES_PER_REQ);
- npages = get_user_pages_fast(user_addr, npages, !write, req->pages);
- if (npages < 0)
-  return npages;
+ while (nbytes < *nbytesp && req->num_pages < FUSE_MAX_PAGES_PER_REQ) {
+  int npages;
+  unsigned long user_addr = fuse_get_ua(*iov_pp, *iov_offset_p);
+  unsigned offset = user_addr & ~PAGE_MASK;
+  size_t frag_size = fuse_get_fr_sz(*iov_pp, *iov_offset_p,
+         *nbytesp - nbytes);

- req->num_pages = npages;
- req->page_descs[0].offset = offset;
- fuse_page_descs_length_init(req);
+  int n = FUSE_MAX_PAGES_PER_REQ - req->num_pages;
+  frag_size = min_t(size_t, frag_size, n << PAGE_SHIFT);

- if (write)
-  req->in.argpages = 1;
- else
-  req->out.argpages = 1;
+  npages = (frag_size + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
+  npages = clamp(npages, 1, n);

- nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;
+  npages = get_user_pages_fast(user_addr, npages, !write,
+          &req->pages[req->num_pages]);
+  if (npages < 0)
+   return npages;

- if (frag_size < nbytes)
-  req->page_descs[req->num_pages - 1].length -=
-   nbytes - frag_size;
+  frag_size = min_t(size_t, frag_size,
+      (npages << PAGE_SHIFT) - offset);
+  nbytes += frag_size;

- *nbytesp = min(frag_size, nbytes);
+  if (frag_size < (*iov_pp)->iov_len - *iov_offset_p) {
+   *iov_offset_p += frag_size;
+  } else {
+   (*iov_pp)++;
+   (*nr_segs_p)--;
+   *iov_offset_p = 0;
+  }

- if (*nbytesp < (*iov_pp)->iov_len - *iov_offset_p) {
-  *iov_offset_p += *nbytesp;
```

```
- } else {
-  (*iov_pp)++;
-  (*nr_segs_p)--;
-  *iov_offset_p = 0;
+  req->page_descs[req->num_pages].offset = offset;
+  fuse_page_descs_length_init(req, req->num_pages, npages);
+
+  req->num_pages += npages;
+  req->page_descs[req->num_pages - 1].length -=
+   (npages << PAGE_SHIFT) - offset - frag_size;
  }

+ if (write)
+  req->in.argpages = 1;
+ else
+  req->out.argpages = 1;
+
+ *nbytesp = nbytes;
+
  return 0;
 }

@@ -1954,7 +1976,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
  }
  memcpy(req->pages, pages, sizeof(req->pages[0]) * num_pages);
  req->num_pages = num_pages;
- fuse_page_descs_length_init(req);
+ fuse_page_descs_length_init(req, 0, req->num_pages);

  /* okay, let's send it to the client */
  req->in.h.opcode = FUSE_IOCTL;
```

Subject: Re: [PATCH v2 00/11] fuse: optimize scatter-gather direct IO
Posted by Maxim Patlasov on Tue, 16 Oct 2012 17:32:17 GMT
View Forum Message <> Reply to Message

Hi Miklos,

> Hi,
>
> Existing fuse implementation processes scatter-gather direct IO in suboptimal
> way: fuse_direct_IO passes iovec[] to fuse_loop_dio and the latter calls
> fuse_direct_read/write for each iovec from iovec[] array. Thus we have as many
> submitted fuse-requests as the number of elements in iovec[] array. This is
> pure waste of resources and affects performance negatively especially for the

> case of many small chunks (e.g. page-size) packed in one iovec[] array.
>
> The patch-set amends situation in a natural way: let's simply pack as
> many iovec[] segments to every fuse-request as possible.
>
> To estimate performance improvement I used slightly modified fusexmp over
> tmpfs (clearing O_DIRECT bit from fi->flags in xmp_open). The test opened
> a file with O_DIRECT, then called readv/writev in a loop. An iovec[] for
> readv/writev consisted of 32 segments of 4K each. The throughput on some
> commodity (rather feeble) server was (in MB/sec):
>
>   original / patched
> writev: ~107   / ~480
> readv: ~114   / ~569
>
> We're exploring possibility to use fuse for our own distributed storage
> implementation and big iovec[] arrays of many page-size chunks is typical
> use-case for device virtualization thread performing i/o on behalf of
> virtual-machine it serves.
>
> Changed in v2:
>   - inline array of page pointers req->pages[] is replaced with dynamically
>   allocated one; the number of elements is calculated a bit more
>   intelligently than being equal to FUSE_MAX_PAGES_PER_REQ; this is done
>   for the sake of memory economy.
>   - a dynamically allocated array of so-called 'page descriptors' - an offset
>   in page plus the length of fragment - is added to fuse_req; this is done
>   to simplify processing fuse requests covering several iov-s.
>
> Thanks,
> Maxim
>
> ---
>
> Maxim Patlasov (11):
>   fuse: general infrastructure for pages[] of variable size
>   fuse: categorize fuse_get_req()
>   fuse: rework fuse_retrieve()
>   fuse: rework fuse_readpages()
>   fuse: rework fuse_perform_write()
>   fuse: rework fuse_do_ioctl()
>   fuse: add per-page descriptor <offset, length> to fuse_req
>   fuse: use req->page_descs[] for argpages cases
>   fuse: pass iov[] to fuse_get_user_pages()
>   fuse: optimize fuse_get_user_pages()
>   fuse: optimize __fuse_direct_io()
>
>

> fs/fuse/cuse.c | 3 -
> fs/fuse/dev.c | 96 +++++++++++++++-------
> fs/fuse/dir.c | 39 ++++----
> fs/fuse/file.c | 250 +++------------------
> fs/fuse/fuse_i.h | 47 +++++++++--
> fs/fuse/inode.c | 6 +
> 6 files changed, 296 insertions(+), 145 deletions(-)
>

Any feedback on this patch-set (v2) would be highly appreciated.

Thanks,
Maxim

Subject: Re: [PATCH 01/11] fuse: general infrastructure for pages[] of variable size
Posted by Miklos Szeredi on Wed, 24 Oct 2012 16:09:22 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.
>
> Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
> ---
> fs/fuse/dev.c | 46 +++++++++++++++++++++++++++++++++++-------------
> fs/fuse/file.c | 4 ++--
> fs/fuse/fuse_i.h | 15 ++++++++++++---
> fs/fuse/inode.c | 4 ++--
> 4 files changed, 49 insertions(+), 20 deletions(-)
>
> diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
> index 7df2b5e..1b5fc73 100644
> --- a/fs/fuse/dev.c
> +++ b/fs/fuse/dev.c
> @@ -34,34 +34,54 @@ 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));
>   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(int npages, gfp_t flags)

I don't see any reason for npages to be a signed integer.

> {
> - 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(int npages)

Ditto.

> +{
> + 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)

Ditto.

> {
> - 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;
> @@ -166,7 +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);
> @@ -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..e686bf1 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(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;
> }

Subject: Re: [PATCH 02/11] fuse: categorize fuse_get_req()
Posted by Miklos Szeredi on Wed, 24 Oct 2012 16:26:45 GMT
View Forum Message <> Reply to Message

Hi,

Sorry about the long delay in responding.

See comments below.

Thanks,

Miklos

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The patch categorizes all fuse_get_req() invocations into two categories:
> - fuse_get_req_nopages(fc) - when caller doesn't care about req->pages
> - fuse_get_req(fc, n) - when caller need n page pointers (n > 0)
>
> Adding fuse_get_req_nopages() helps to avoid numerous fuse_get_req(fc, 0)
> scattered over code. Now it's clear from the first glance when a caller need
> fuse_req with page pointers.
>
> The patch doesn't make any logic changes. In multi-page case, it silly
> allocates array of FUSE_MAX_PAGES_PER_REQ page pointers. This will be amended
> by future patches.
>
> Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
> ---
> fs/fuse/cuse.c | 2 +-
> fs/fuse/dev.c | 11 ++++++-----
> fs/fuse/dir.c | 38 ++++++++++++++++++++----------------
> fs/fuse/file.c | 30 ++++++++++++++++++++----------
> fs/fuse/fuse_i.h | 17 ++++++++++++++---
> fs/fuse/inode.c | 2 +-
> 6 files changed, 56 insertions(+), 44 deletions(-)
>
> diff --git a/fs/fuse/cuse.c b/fs/fuse/cuse.c
> index 3426521..2f1d5dd 100644
> --- a/fs/fuse/cuse.c
> +++ b/fs/fuse/cuse.c
> @@ -411,7 +411,7 @@ static int cuse_send_init(struct cuse_conn *cc)
>
>   BUILD_BUG_ON(CUSE_INIT_INFO_MAX > PAGE_SIZE);
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req(fc, 1);
>   if (IS_ERR(req)) {
>   rc = PTR_ERR(req);
>   goto err;
> diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
> index 1b5fc73..3ad5570 100644
> --- a/fs/fuse/dev.c
> +++ b/fs/fuse/dev.c
> @@ -117,7 +117,7 @@ static void fuse_req_init_context(struct fuse_req *req)
>   req->in.h.pid = current->pid;
> }
>

> -struct fuse_req *fuse_get_req(struct fuse_conn *fc)
> +struct fuse_req *fuse_get_req(struct fuse_conn *fc, int npages)

Again, npages should be unsigned, AFAICS.

> {
>   struct fuse_req *req;
>   sigset_t oldset;
> @@ -136,7 +136,7 @@ struct fuse_req *fuse_get_req(struct fuse_conn *fc)
>   if (!fc->connected)
>   goto out;
>
> - req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
> + req = fuse_request_alloc(npages);
>   err = -ENOMEM;
>   if (!req)
>   goto out;
> @@ -207,13 +207,14 @@ static void put_reserved_req(struct fuse_conn *fc, struct fuse_req
*req)
>   * filesystem should not have it's own file open. If deadlock is
>   * intentional, it can still be broken by "aborting" the filesystem.
>   */
> -struct fuse_req *fuse_get_req_nofail(struct fuse_conn *fc, struct file *file)
> +struct fuse_req *fuse_get_req_nofail_nopages(struct fuse_conn *fc,
> +   struct file *file)
> {
>   struct fuse_req *req;
>
>   atomic_inc(&fc->num_waiting);
>   wait_event(fc->blocked_waitq, !fc->blocked);
> - req = fuse_request_alloc(FUSE_MAX_PAGES_PER_REQ);
> + req = fuse_request_alloc(0);
>   if (!req)
>   req = get_reserved_req(fc, file);
>
> @@ -1563,7 +1564,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
>   unsigned int offset;
>   size_t total_len = 0;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> diff --git a/fs/fuse/dir.c b/fs/fuse/dir.c
> index 324bc08..1929bfb 100644
> --- a/fs/fuse/dir.c
> +++ b/fs/fuse/dir.c

> @@ -178,7 +178,7 @@ static int fuse_dentry_revalidate(struct dentry *entry, unsigned int flags)
> return -ECHILD;
>
> fc = get_fuse_conn(inode);
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
> if (IS_ERR(req))
> return 0;
>
> @@ -271,7 +271,7 @@ int fuse_lookup_name(struct super_block *sb, u64 nodeid, struct qstr
*name,
> if (name->len > FUSE_NAME_MAX)
> goto out;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
> err = PTR_ERR(req);
> if (IS_ERR(req))
> goto out;
> @@ -391,7 +391,7 @@ static int fuse_create_open(struct inode *dir, struct dentry *entry,
> if (!forget)
> goto out_err;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
> err = PTR_ERR(req);
> if (IS_ERR(req))
> goto out_put_forget_req;
> @@ -592,7 +592,7 @@ static int fuse_mknod(struct inode *dir, struct dentry *entry, umode_t
mode,
> {
> struct fuse_mknod_in inarg;
> struct fuse_conn *fc = get_fuse_conn(dir);
> - struct fuse_req *req = fuse_get_req(fc);
> + struct fuse_req *req = fuse_get_req_nopages(fc);
> if (IS_ERR(req))
> return PTR_ERR(req);
>
> @@ -623,7 +623,7 @@ static int fuse_mkdir(struct inode *dir, struct dentry *entry, umode_t
mode)
> {
> struct fuse_mkdir_in inarg;
> struct fuse_conn *fc = get_fuse_conn(dir);
> - struct fuse_req *req = fuse_get_req(fc);
> + struct fuse_req *req = fuse_get_req_nopages(fc);
> if (IS_ERR(req))
> return PTR_ERR(req);
>

```
> @@ -647,7 +647,7 @@ static int fuse_symlink(struct inode *dir, struct dentry *entry,
>  {
>   struct fuse_conn *fc = get_fuse_conn(dir);
>   unsigned len = strlen(link) + 1;
> - struct fuse_req *req = fuse_get_req(fc);
> + struct fuse_req *req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>    return PTR_ERR(req);
>
> @@ -664,7 +664,7 @@ static int fuse_unlink(struct inode *dir, struct dentry *entry)
>  {
>   int err;
>   struct fuse_conn *fc = get_fuse_conn(dir);
> - struct fuse_req *req = fuse_get_req(fc);
> + struct fuse_req *req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>    return PTR_ERR(req);
>
> @@ -696,7 +696,7 @@ static int fuse_rmdir(struct inode *dir, struct dentry *entry)
>  {
>   int err;
>   struct fuse_conn *fc = get_fuse_conn(dir);
> - struct fuse_req *req = fuse_get_req(fc);
> + struct fuse_req *req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>    return PTR_ERR(req);
>
> @@ -723,7 +723,7 @@ static int fuse_rename(struct inode *olddir, struct dentry *oldent,
>   int err;
>   struct fuse_rename_in inarg;
>   struct fuse_conn *fc = get_fuse_conn(olddir);
> - struct fuse_req *req = fuse_get_req(fc);
> + struct fuse_req *req = fuse_get_req_nopages(fc);
>
>   if (IS_ERR(req))
>    return PTR_ERR(req);
> @@ -776,7 +776,7 @@ static int fuse_link(struct dentry *entry, struct inode *newdir,
>   struct fuse_link_in inarg;
>   struct inode *inode = entry->d_inode;
>   struct fuse_conn *fc = get_fuse_conn(inode);
> - struct fuse_req *req = fuse_get_req(fc);
> + struct fuse_req *req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>    return PTR_ERR(req);
>
> @@ -848,7 +848,7 @@ static int fuse_do_getattr(struct inode *inode, struct kstat *stat,
>   struct fuse_req *req;
>   u64 attr_version;
```

```
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>    return PTR_ERR(req);
>
> @@ -1029,7 +1029,7 @@ static int fuse_access(struct inode *inode, int mask)
>   if (fc->no_access)
>    return 0;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>    return PTR_ERR(req);
>
> @@ -1167,7 +1167,7 @@ static int fuse_readdir(struct file *file, void *dstbuf, filldir_t filldir)
>   if (is_bad_inode(inode))
>    return -EIO;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req(fc, 1);
>   if (IS_ERR(req))
>    return PTR_ERR(req);
>
> @@ -1197,7 +1197,7 @@ static char *read_link(struct dentry *dentry)
>  {
>   struct inode *inode = dentry->d_inode;
>   struct fuse_conn *fc = get_fuse_conn(inode);
> - struct fuse_req *req = fuse_get_req(fc);
> + struct fuse_req *req = fuse_get_req_nopages(fc);
>   char *link;
>
>   if (IS_ERR(req))
> @@ -1410,7 +1410,7 @@ static int fuse_do_setattr(struct dentry *entry, struct iattr *attr,
>   if (attr->ia_valid & ATTR_SIZE)
>    is_truncate = true;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>    return PTR_ERR(req);
>
> @@ -1518,7 +1518,7 @@ static int fuse_setxattr(struct dentry *entry, const char *name,
>   if (fc->no_setxattr)
>    return -EOPNOTSUPP;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
```

>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> @@ -1557,7 +1557,7 @@ static ssize_t fuse_getxattr(struct dentry *entry, const char *name,
>   if (fc->no_getxattr)
>   return -EOPNOTSUPP;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> @@ -1609,7 +1609,7 @@ static ssize_t fuse_listxattr(struct dentry *entry, char *list, size_t size)
>   if (fc->no_listxattr)
>   return -EOPNOTSUPP;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> @@ -1654,7 +1654,7 @@ static int fuse_removexattr(struct dentry *entry, const char *name)
>   if (fc->no_removexattr)
>   return -EOPNOTSUPP;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> diff --git a/fs/fuse/file.c b/fs/fuse/file.c
> index 7423ea4..214e13e 100644
> --- a/fs/fuse/file.c
> +++ b/fs/fuse/file.c
> @@ -25,7 +25,7 @@ static int fuse_send_open(struct fuse_conn *fc, u64 nodeid, struct file
*file,
>   struct fuse_req *req;
>   int err;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> @@ -368,7 +368,7 @@ static int fuse_flush(struct file *file, fl_owner_t id)
>   if (fc->no_flush)
>   return 0;
>

> - req = fuse_get_req_nofail(fc, file);
> + req = fuse_get_req_nofail_nopages(fc, file);
>   memset(&inarg, 0, sizeof(inarg));
>   inarg.fh = ff->fh;
>   inarg.lock_owner = fuse_lock_owner_id(fc, id);
> @@ -436,7 +436,7 @@ int fuse_fsync_common(struct file *file, loff_t start, loff_t end,
>
>   fuse_sync_writes(inode);
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req)) {
>   err = PTR_ERR(req);
>   goto out;
> @@ -544,7 +544,7 @@ static int fuse_readpage(struct file *file, struct page *page)
>   */
>   fuse_wait_on_page_writeback(inode, page->index);
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req(fc, 1);
>   err = PTR_ERR(req);
>   if (IS_ERR(req))
>   goto out;
> @@ -657,7 +657,7 @@ static int fuse_readpages_fill(void *_data, struct page *page)
>   (req->num_pages + 1) * PAGE_CACHE_SIZE > fc->max_read ||
>   req->pages[req->num_pages - 1]->index + 1 != page->index)) {
>   fuse_send_readpages(req, data->file);
> - data->req = req = fuse_get_req(fc);
> + data->req = req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
>   if (IS_ERR(req)) {
>   unlock_page(page);
>   return PTR_ERR(req);
> @@ -683,7 +683,7 @@ static int fuse_readpages(struct file *file, struct address_space
*mapping,
>
>   data.file = file;
>   data.inode = inode;
> - data.req = fuse_get_req(fc);
> + data.req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
>   err = PTR_ERR(data.req);
>   if (IS_ERR(data.req))
>   goto out;
> @@ -890,7 +890,7 @@ static ssize_t fuse_perform_write(struct file *file,
>   struct fuse_req *req;
>   ssize_t count;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);

>   if (IS_ERR(req)) {
>   err = PTR_ERR(req);
>   break;
> @@ -1072,7 +1072,7 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
>   ssize_t res = 0;
>   struct fuse_req *req;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> @@ -1108,7 +1108,7 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
>   break;
>   if (count) {
>   fuse_put_request(fc, req);
> -   req = fuse_get_req(fc);
> +   req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
>   if (IS_ERR(req))
>   break;
>   }
> @@ -1470,7 +1470,7 @@ static int fuse_getlk(struct file *file, struct file_lock *fl)
>   struct fuse_lk_out outarg;
>   int err;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> @@ -1505,7 +1505,7 @@ static int fuse_setlk(struct file *file, struct file_lock *fl, int flock)
>   if (fl->fl_flags & FL_CLOSE)
>   return 0;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> @@ -1574,7 +1574,7 @@ static sector_t fuse_bmap(struct address_space *mapping, sector_t
block)
>   if (!inode->i_sb->s_bdev || fc->no_bmap)
>   return 0;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return 0;

>
> @@ -1872,7 +1872,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
>   num_pages++;
>   }
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
>   if (IS_ERR(req)) {
>   err = PTR_ERR(req);
>   req = NULL;
> @@ -2075,7 +2075,7 @@ unsigned fuse_file_poll(struct file *file, poll_table *wait)
>   fuse_register_polled_file(fc, ff);
>   }
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return POLLERR;
>
> @@ -2193,7 +2193,7 @@ long fuse_file_fallocate(struct file *file, int mode, loff_t offset,
>   if (fc->no_fallocate)
>   return -EOPNOTSUPP;
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>
> diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
> index e686bf1..fd69d62 100644
> --- a/fs/fuse/fuse_i.h
> +++ b/fs/fuse/fuse_i.h
> @@ -677,14 +677,25 @@ struct fuse_req *fuse_request_alloc_nofs(int npages);
> void fuse_request_free(struct fuse_req *req);
>
> /**
> - * Get a request, may fail with -ENOMEM
> + * Get a request, may fail with -ENOMEM,
> + * caller should specify # elements in req->pages[] explicitly
>   */
> -struct fuse_req *fuse_get_req(struct fuse_conn *fc);
> +struct fuse_req *fuse_get_req(struct fuse_conn *fc, int npages);

Ditto.

> +

> +/**
> + * Get a request, may fail with -ENOMEM,
> + * useful for callers who doesn't use req->pages[]
> + */
> +static inline struct fuse_req *fuse_get_req_nopages(struct fuse_conn *fc)
> +{
> + return fuse_get_req(fc, 0);
> +}
>
> /**
>   * Gets a requests for a file operation, always succeeds
>   */
> -struct fuse_req *fuse_get_req_nofail(struct fuse_conn *fc, struct file *file);
> +struct fuse_req *fuse_get_req_nofail_nopages(struct fuse_conn *fc,
> +   struct file *file);
>
> /**
>   * Decrement reference count of a request. If count goes to zero free
> diff --git a/fs/fuse/inode.c b/fs/fuse/inode.c
> index 3f399ba..efb6144 100644
> --- a/fs/fuse/inode.c
> +++ b/fs/fuse/inode.c
> @@ -411,7 +411,7 @@ static int fuse_statfs(struct dentry *dentry, struct kstatfs *buf)
>   return 0;
>   }
>
> - req = fuse_get_req(fc);
> + req = fuse_get_req_nopages(fc);
>   if (IS_ERR(req))
>   return PTR_ERR(req);
>

Subject: Re: [PATCH 07/11] fuse: add per-page descriptor <offset, length> to fuse_req (v2)
Posted by Miklos Szeredi on Thu, 25 Oct 2012 13:24:17 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The ability to save page pointers along with lengths and offsets in fuse_req
> will be useful to cover several iovec-s with a single fuse_req.
>
> Per-request page_offset is removed because anybody who need it can use
> req->page_descs[0].offset instead.
>
> Changed in v2:
> - replaced structure page_desc with fuse_page_desc

>
> Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
> ---
> fs/fuse/dev.c | 26 ++++++++++++++++++-------
> fs/fuse/file.c | 10 +++++-----
> fs/fuse/fuse_i.h | 15 +++++++++++---
> 3 files changed, 36 insertions(+), 15 deletions(-)
>
> diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
> index b241a7d..72ad962 100644
> --- a/fs/fuse/dev.c
> +++ b/fs/fuse/dev.c
> @@ -35,14 +35,17 @@ static struct fuse_conn *fuse_get_conn(struct file *file)
> }
>
> static void fuse_request_init(struct fuse_req *req, struct page **pages,
> +   struct fuse_page_desc *page_descs,
>   unsigned npages)
> {
>   memset(req, 0, sizeof(*req));
> + memset(page_descs, 0, sizeof(*page_descs) * npages);

Makes me wonder: why aren't we zeroing out the page array too?

>   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->page_descs = page_descs;
>   req->max_pages = npages;
> }
>
> @@ -51,18 +54,25 @@ static struct fuse_req *__fuse_request_alloc(int npages, gfp_t flags)
>   struct fuse_req *req = kmem_cache_alloc(fuse_req_cachep, flags);
>   if (req) {
>   struct page **pages;
> + struct fuse_page_desc *page_descs;
>
> - if (npages <= FUSE_REQ_INLINE_PAGES)
> + if (npages <= FUSE_REQ_INLINE_PAGES) {
>   pages = req->inline_pages;
> - else
> +   page_descs = req->inline_page_descs;
> + } else {
>   pages = kmalloc(sizeof(struct page *) * npages, flags);
> +   page_descs = kmalloc(sizeof(struct fuse_page_desc) *
> +   npages, flags);

> + }
>
> - if (!pages) {
> + if (!pages || !page_descs) {
> +   kfree(pages);
> +   kfree(page_descs);
>   kmem_cache_free(fuse_req_cachep, req);
>   return NULL;
>   }
>
> - fuse_request_init(req, pages, npages);
> + fuse_request_init(req, pages, page_descs, npages);
>   }
>   return req;
> }
> @@ -82,6 +92,8 @@ void fuse_request_free(struct fuse_req *req)
> {
>   if (req->pages != req->inline_pages)
>   kfree(req->pages);
> + if (req->page_descs != req->inline_page_descs)
> + kfree(req->page_descs);

You allocate them together, you can free them together. Just add a
BUG_ON() if you feel paranoid.

>   kmem_cache_free(fuse_req_cachep, req);
> }
>
> @@ -186,7 +198,7 @@ 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, req->pages, req->max_pages);
> + fuse_request_init(req, req->pages, req->page_descs, req->max_pages);
>   BUG_ON(ff->reserved_req);
>   ff->reserved_req = req;
>   wake_up_all(&fc->reserved_req_waitq);
> @@ -877,7 +889,7 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned
nbytes,
> {
>   unsigned i;
>   struct fuse_req *req = cs->req;
> - unsigned offset = req->page_offset;
> + unsigned offset = req->page_descs[0].offset;
>   unsigned count = min(nbytes, (unsigned) PAGE_SIZE - offset);
>
>   for (i = 0; i < req->num_pages && (nbytes || zeroing); i++) {
> @@ -1585,7 +1597,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,

>   req->in.h.nodeid = outarg->nodeid;
>   req->in.numargs = 2;
>   req->in.argpages = 1;
> - req->page_offset = offset;
> + req->page_descs[0].offset = offset;
>   req->end = fuse_retrieve_end;
>
>   index = outarg->offset >> PAGE_CACHE_SHIFT;
> diff --git a/fs/fuse/file.c b/fs/fuse/file.c
> index 08899a6..b68cf3b 100644
> --- a/fs/fuse/file.c
> +++ b/fs/fuse/file.c
> @@ -798,7 +798,7 @@ static size_t fuse_send_write_pages(struct fuse_req *req, struct file
*file,
>
>   res = fuse_send_write(req, file, pos, count, NULL);
>
> - offset = req->page_offset;
> + offset = req->page_descs[0].offset;
>   count = res;
>   for (i = 0; i < req->num_pages; i++) {
>   struct page *page = req->pages[i];
> @@ -829,7 +829,7 @@ static ssize_t fuse_fill_write_pages(struct fuse_req *req,
>   int err;
>
>   req->in.argpages = 1;
> - req->page_offset = offset;
> + req->page_descs[0].offset = offset;
>
>   do {
>   size_t tmp;
> @@ -1070,14 +1070,14 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,
>   return npages;
>
>   req->num_pages = npages;
> - req->page_offset = offset;
> + req->page_descs[0].offset = offset;
>
>   if (write)
>   req->in.argpages = 1;
>   else
>   req->out.argpages = 1;
>
> - nbytes = (req->num_pages << PAGE_SHIFT) - req->page_offset;
> + nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;
>   *nbytesp = min(*nbytesp, nbytes);
>

>   return 0;
> @@ -1314,7 +1314,7 @@ static int fuse_writepage_locked(struct page *page)
>   req->in.argpages = 1;
>   req->num_pages = 1;
>   req->pages[0] = tmp_page;
> - req->page_offset = 0;
> + req->page_descs[0].offset = 0;
>   req->end = fuse_writepage_end;
>   req->inode = inode;
>
> diff --git a/fs/fuse/fuse_i.h b/fs/fuse/fuse_i.h
> index fd69d62..c07e454 100644
> --- a/fs/fuse/fuse_i.h
> +++ b/fs/fuse/fuse_i.h
> @@ -203,6 +203,12 @@ struct fuse_out {
>   struct fuse_arg args[3];
> };
>
> +/** FUSE page descriptor */
> +struct fuse_page_desc {
> + unsigned int length;
> + unsigned int offset;
> +};
> +
> /** The request state */
> enum fuse_req_state {
>   FUSE_REQ_INIT = 0,
> @@ -296,18 +302,21 @@ struct fuse_req {
>   /** page vector */
>   struct page **pages;
>
> + /** page-descriptor vector */
> + struct fuse_page_desc *page_descs;
> +
>   /** size of the 'pages' array */
>   unsigned max_pages;
>
>   /** inline page vector */
>   struct page *inline_pages[FUSE_REQ_INLINE_PAGES];
>
> + /** inline page-descriptor vector */
> + struct fuse_page_desc inline_page_descs[FUSE_REQ_INLINE_PAGES];
> +
>   /** number of pages in vector */
>   unsigned num_pages;
>
> - /** offset of data on first page */
> - unsigned page_offset;

> -
> /** File used in the request (or NULL) */
> struct fuse_file *ff;
>

Subject: Re: [PATCH 07/11] fuse: add per-page descriptor <offset, length> to fuse_req (v2)
Posted by Maxim Patlasov on Thu, 25 Oct 2012 13:39:21 GMT
View Forum Message <> Reply to Message

Hi,

> Maxim Patlasov <mpatlasov@parallels.com> writes:
>
>> The ability to save page pointers along with lengths and offsets in fuse_req
>> will be useful to cover several iovec-s with a single fuse_req.
>>
>> Per-request page_offset is removed because anybody who need it can use
>> req->page_descs[0].offset instead.
>>
>> Changed in v2:
>> - replaced structure page_desc with fuse_page_desc
>>
>> Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
>> ---
>> fs/fuse/dev.c | 26 ++++++++++++++++++++-------
>> fs/fuse/file.c | 10 +++++-----
>> fs/fuse/fuse_i.h | 15 +++++++++++---
>> 3 files changed, 36 insertions(+), 15 deletions(-)
>>
>> diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
>> index b241a7d..72ad962 100644
>> --- a/fs/fuse/dev.c
>> +++ b/fs/fuse/dev.c
>> @@ -35,14 +35,17 @@ static struct fuse_conn *fuse_get_conn(struct file *file)
>> }
>>
>> static void fuse_request_init(struct fuse_req *req, struct page **pages,
>> +   struct fuse_page_desc *page_descs,
>>   unsigned npages)
>> {
>>   memset(req, 0, sizeof(*req));
>> + memset(page_descs, 0, sizeof(*page_descs) * npages);
> Makes me wonder: why aren't we zeroing out the page array too?

Good catch, thnx! This is a legacy since time when I attempted to use

<page, length, offset> as fuse_page_desc. Now, when we have both
page_descs[] and pages[], it's natural to zero both.

>
> @@ -82,6 +92,8 @@ void fuse_request_free(struct fuse_req *req)
> {
>   if (req->pages != req->inline_pages)
>   kfree(req->pages);
> + if (req->page_descs != req->inline_page_descs)
> + kfree(req->page_descs);
> You allocate them together, you can free them together. Just add a
> BUG_ON() if you feel paranoid.

OK.

Thanks,
Maxim

Subject: Re: [PATCH 08/11] fuse: use req->page_descs[] for argpages cases
Posted by Miklos Szeredi on Thu, 25 Oct 2012 14:05:22 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> Previously, anyone who set flag 'argpages' only filled req->pages[] and set
> per-request page_offset. This patch re-works all cases where argpages=1 to
> fill req->page_descs[] properly.
>
> Having req->page_descs[] filled properly allows to re-work fuse_copy_pages()
> to copy page fragments described by req->page_descs[]. This will be useful
> for next patches optimizing direct_IO.
>
> Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
> ---
> fs/fuse/cuse.c | 1 +
> fs/fuse/dev.c | 7 ++++----
> fs/fuse/dir.c | 1 +
> fs/fuse/file.c | 20 ++++++++++++++++++++
> 4 files changed, 25 insertions(+), 4 deletions(-)
>
> diff --git a/fs/fuse/cuse.c b/fs/fuse/cuse.c
> index 2f1d5dd..38efe69 100644
> --- a/fs/fuse/cuse.c
> +++ b/fs/fuse/cuse.c
> @@ -441,6 +441,7 @@ static int cuse_send_init(struct cuse_conn *cc)
>   req->out.argvar = 1;
>   req->out.argpages = 1;

>   req->pages[0] = page;
> + req->page_descs[0].length = req->out.args[1].size;
>   req->num_pages = 1;
>   req->end = cuse_process_init_reply;
>   fuse_request_send_background(fc, req);
> diff --git a/fs/fuse/dev.c b/fs/fuse/dev.c
> index ea63d39..21b6272 100644
> --- a/fs/fuse/dev.c
> +++ b/fs/fuse/dev.c
> @@ -888,11 +888,11 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned
nbytes,
> {
>   unsigned i;
>   struct fuse_req *req = cs->req;
> - unsigned offset = req->page_descs[0].offset;
> - unsigned count = min(nbytes, (unsigned) PAGE_SIZE - offset);
>
>   for (i = 0; i < req->num_pages && (nbytes || zeroing); i++) {
>   int err;
> + unsigned offset = req->page_descs[i].offset;
> + unsigned count = min(nbytes, req->page_descs[i].length);

Wouldn't it be cleaner if callers calculated the last page's .length
value from the total number of bytes? So this would just be

  unsigned count = req->page_descs[i].length;

And at the end of the function we can assert that nbytes went to exactly
zero with a WARN_ON().

But this is a change that needs careful testing, so maybe we're better
off having that as a separate incremental patch later...

>
>   err = fuse_copy_page(cs, &req->pages[i], offset, count,
>   zeroing);
> @@ -900,8 +900,6 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned
nbytes,
>   return err;
>
>   nbytes -= count;
> - count = min(nbytes, (unsigned) PAGE_SIZE);
> - offset = 0;
>   }
>   return 0;
> }
> @@ -1611,6 +1609,7 @@ static int fuse_retrieve(struct fuse_conn *fc, struct inode *inode,
>

>   this_num = min_t(unsigned, num, PAGE_CACHE_SIZE - offset);
>   req->pages[req->num_pages] = page;
> + req->page_descs[req->num_pages].length = this_num;
>   req->num_pages++;
>
>   num -= this_num;
> diff --git a/fs/fuse/dir.c b/fs/fuse/dir.c
> index 1929bfb..95e4016 100644
> --- a/fs/fuse/dir.c
> +++ b/fs/fuse/dir.c
> @@ -1179,6 +1179,7 @@ static int fuse_readdir(struct file *file, void *dstbuf, filldir_t filldir)
>   req->out.argpages = 1;
>   req->num_pages = 1;
>   req->pages[0] = page;
> + req->page_descs[0].length = PAGE_SIZE;
>   fuse_read_fill(req, file, file->f_pos, PAGE_SIZE, FUSE_READDIR);
>   fuse_request_send(fc, req);
>   nbytes = req->out.args[0].size;
> diff --git a/fs/fuse/file.c b/fs/fuse/file.c
> index b68cf3b..8a1f833 100644
> --- a/fs/fuse/file.c
> +++ b/fs/fuse/file.c
> @@ -555,6 +555,7 @@ static int fuse_readpage(struct file *file, struct page *page)
>   req->out.argpages = 1;
>   req->num_pages = 1;
>   req->pages[0] = page;
> + req->page_descs[0].length = count;
>   num_read = fuse_send_read(req, file, pos, count, NULL);
>   err = req->out.h.error;
>   fuse_put_request(fc, req);
> @@ -674,6 +675,7 @@ static int fuse_readpages_fill(void *_data, struct page *page)
>
>   page_cache_get(page);
>   req->pages[req->num_pages] = page;
> + req->page_descs[req->num_pages].length = PAGE_SIZE;
>   req->num_pages++;
>   data->nr_pages--;
>   return 0;
> @@ -869,6 +871,7 @@ static ssize_t fuse_fill_write_pages(struct fuse_req *req,
>
>   err = 0;
>   req->pages[req->num_pages] = page;
> + req->page_descs[req->num_pages].length = tmp;
>   req->num_pages++;
>
>   iov_iter_advance(ii, tmp);
> @@ -1044,6 +1047,15 @@ static void fuse_release_user_pages(struct fuse_req *req, int write)
>   }

```
>  }
>
> +static inline void fuse_page_descs_length_init(struct fuse_req *req)
> +{
> + int i;
> +
> + for (i = 0; i < req->num_pages; i++)
> +  req->page_descs[i].length = PAGE_SIZE -
> +   req->page_descs[i].offset;
> +}
> +
>  static int fuse_get_user_pages(struct fuse_req *req, const char __user *buf,
>            size_t *nbytesp, int write)
>  {
> @@ -1071,6 +1083,7 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,
>
>   req->num_pages = npages;
>   req->page_descs[0].offset = offset;
> + fuse_page_descs_length_init(req);
>
>   if (write)
>    req->in.argpages = 1;
> @@ -1078,6 +1091,11 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,
>    req->out.argpages = 1;
>
>   nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;
> +
> + if (*nbytesp < nbytes)
> +  req->page_descs[req->num_pages - 1].length -=
> +   nbytes - *nbytesp;
> +
>   *nbytesp = min(*nbytesp, nbytes);
>
>   return 0;
> @@ -1315,6 +1333,7 @@ static int fuse_writepage_locked(struct page *page)
>   req->num_pages = 1;
>   req->pages[0] = tmp_page;
>   req->page_descs[0].offset = 0;
> + req->page_descs[0].length = PAGE_SIZE;
>   req->end = fuse_writepage_end;
>   req->inode = inode;
>
> @@ -1901,6 +1920,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
>   }
>   memcpy(req->pages, pages, sizeof(req->pages[0]) * num_pages);
```

> req->num_pages = num_pages;
> + fuse_page_descs_length_init(req);
>
> /* okay, let's send it to the client */
> req->in.h.opcode = FUSE_IOCTL;

Subject: Re: [PATCH 09/11] fuse: pass iov[] to fuse_get_user_pages()
Posted by Miklos Szeredi on Thu, 25 Oct 2012 14:29:27 GMT
View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

> The patch makes preliminary work for the next patch optimizing scatter-gather
> direct IO. The idea is to allow fuse_get_user_pages() to pack as many iov-s
> to each fuse request as possible. So, here we only rework all related
> call-paths to carry iov[] from fuse_direct_IO() to fuse_get_user_pages().
>
> Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
> ---
> fs/fuse/file.c | 108 ++++++++++++++++++++++++++++++++++++++----------------
> 1 files changed, 55 insertions(+), 53 deletions(-)
>
> diff --git a/fs/fuse/file.c b/fs/fuse/file.c
> index 8a1f833..db9efb5 100644
> --- a/fs/fuse/file.c
> +++ b/fs/fuse/file.c
> @@ -1056,11 +1056,15 @@ static inline void fuse_page_descs_length_init(struct fuse_req
*req)
>   req->page_descs[i].offset;
> }
>
> -static int fuse_get_user_pages(struct fuse_req *req, const char __user *buf,
> +static int fuse_get_user_pages(struct fuse_req *req,
> +   const struct iovec **iov_pp,
> +   unsigned long *nr_segs_p,
> +   size_t *iov_offset_p,

Aren't these arguments exactly what 'struct iov_iter' is supposed to
represent? Would be a nice cleanup, and in addition we'd get well
tested helper functions that iterate over the iovec.

>   size_t *nbytesp, int write)
> {
>   size_t nbytes = *nbytesp;
> - unsigned long user_addr = (unsigned long) buf;
> + size_t frag_size = min_t(size_t, nbytes, (*iov_pp)->iov_len - *iov_offset_p);
> + unsigned long user_addr = (unsigned long)(*iov_pp)->iov_base + *iov_offset_p;

>   unsigned offset = user_addr & ~PAGE_MASK;
>   int npages;
>
> @@ -1071,10 +1075,13 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,
>   else
>   req->out.args[0].value = (void *) user_addr;
>
> + (*iov_pp)++;
> + (*nr_segs_p)--;
> + *nbytesp = frag_size;
>   return 0;
>   }
>
> - nbytes = min_t(size_t, nbytes, FUSE_MAX_PAGES_PER_REQ << PAGE_SHIFT);
> + nbytes = min_t(size_t, frag_size, FUSE_MAX_PAGES_PER_REQ << PAGE_SHIFT);
>   npages = (nbytes + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
>   npages = clamp(npages, 1, FUSE_MAX_PAGES_PER_REQ);
>   npages = get_user_pages_fast(user_addr, npages, !write, req->pages);
> @@ -1092,17 +1099,26 @@ static int fuse_get_user_pages(struct fuse_req *req, const char
__user *buf,
>
>   nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;
>
> - if (*nbytesp < nbytes)
> + if (frag_size < nbytes)
>   req->page_descs[req->num_pages - 1].length -=
> -   nbytes - *nbytesp;
> +   nbytes - frag_size;
>
> - *nbytesp = min(*nbytesp, nbytes);
> + *nbytesp = min(frag_size, nbytes);
> +
> + if (*nbytesp < (*iov_pp)->iov_len - *iov_offset_p) {
> + *iov_offset_p += *nbytesp;
> + } else {
> + (*iov_pp)++;
> + (*nr_segs_p)--;
> + *iov_offset_p = 0;
> + }
>
>   return 0;
> }
>
> -ssize_t fuse_direct_io(struct file *file, const char __user *buf,
> -   size_t count, loff_t *ppos, int write)
> +static ssize_t __fuse_direct_io(struct file *file, const struct iovec *iov,
> +   unsigned long nr_segs, size_t count,

```
> +    loff_t *ppos, int write)
>  {
>   struct fuse_file *ff = file->private_data;
>   struct fuse_conn *fc = ff->fc;
> @@ -1110,6 +1126,7 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
>   loff_t pos = *ppos;
>   ssize_t res = 0;
>   struct fuse_req *req;
> + size_t iov_offset = 0;
>
>   req = fuse_get_req(fc, FUSE_MAX_PAGES_PER_REQ);
>   if (IS_ERR(req))
> @@ -1119,7 +1136,8 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
>    size_t nres;
>    fl_owner_t owner = current->files;
>    size_t nbytes = min(count, nmax);
> -  int err = fuse_get_user_pages(req, buf, &nbytes, write);
> +  int err = fuse_get_user_pages(req, &iov, &nr_segs, &iov_offset,
> +            &nbytes, write);
>    if (err) {
>     res = err;
>     break;
> @@ -1142,7 +1160,6 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
>    count -= nres;
>    res += nres;
>    pos += nres;
> -  buf += nres;
>    if (nres != nbytes)
>     break;
>    if (count) {
> @@ -1159,10 +1176,17 @@ ssize_t fuse_direct_io(struct file *file, const char __user *buf,
>
>   return res;
>  }
> +
> +ssize_t fuse_direct_io(struct file *file, const char __user *buf,
> +         size_t count, loff_t *ppos, int write)
> +{
> + struct iovec iov = { .iov_base = (void *)buf, .iov_len = count };
> + return __fuse_direct_io(file, &iov, 1, count, ppos, write);
> +}
>  EXPORT_SYMBOL_GPL(fuse_direct_io);
>
> -static ssize_t fuse_direct_read(struct file *file, char __user *buf,
> -         size_t count, loff_t *ppos)
> +static ssize_t __fuse_direct_read(struct file *file, const struct iovec *iov,
> +       unsigned long nr_segs, loff_t *ppos)
>  {
```

```
>   ssize_t res;
>   struct inode *inode = file->f_path.dentry->d_inode;
> @@ -1170,22 +1194,31 @@ static ssize_t fuse_direct_read(struct file *file, char __user *buf,
>   if (is_bad_inode(inode))
>    return -EIO;
>
> - res = fuse_direct_io(file, buf, count, ppos, 0);
> + res = __fuse_direct_io(file, iov, nr_segs, iov_length(iov, nr_segs),
> +          ppos, 0);
>
>   fuse_invalidate_attr(inode);
>
>   return res;
>  }
>
> -static ssize_t __fuse_direct_write(struct file *file, const char __user *buf,
> -       size_t count, loff_t *ppos)
> +static ssize_t fuse_direct_read(struct file *file, char __user *buf,
> +         size_t count, loff_t *ppos)
> +{
> + struct iovec iov = { .iov_base = (void *)buf, .iov_len = count };
> + return __fuse_direct_read(file, &iov, 1, ppos);
> +}
> +
> +static ssize_t __fuse_direct_write(struct file *file, const struct iovec *iov,
> +       unsigned long nr_segs, loff_t *ppos)
>  {
>   struct inode *inode = file->f_path.dentry->d_inode;
> + size_t count = iov_length(iov, nr_segs);
>   ssize_t res;
>
>   res = generic_write_checks(file, ppos, &count, 0);
>   if (!res) {
> -  res = fuse_direct_io(file, buf, count, ppos, 1);
> +  res = __fuse_direct_io(file, iov, nr_segs, count, ppos, 1);
>    if (res > 0)
>     fuse_write_update_size(inode, *ppos);
>   }
> @@ -1198,6 +1231,7 @@ static ssize_t __fuse_direct_write(struct file *file, const char __user
*buf,
>  static ssize_t fuse_direct_write(struct file *file, const char __user *buf,
>       size_t count, loff_t *ppos)
>  {
> + struct iovec iov = { .iov_base = (void *)buf, .iov_len = count };
>   struct inode *inode = file->f_path.dentry->d_inode;
>   ssize_t res;
>
> @@ -1206,7 +1240,7 @@ static ssize_t fuse_direct_write(struct file *file, const char __user
```

```
*buf,
>
>   /* Don't allow parallel writes to the same file */
>   mutex_lock(&inode->i_mutex);
> - res = __fuse_direct_write(file, buf, count, ppos);
> + res = __fuse_direct_write(file, &iov, 1, ppos);
>   mutex_unlock(&inode->i_mutex);
>
>   return res;
> @@ -2166,41 +2200,6 @@ int fuse_notify_poll_wakeup(struct fuse_conn *fc,
>   return 0;
>  }
>
> -static ssize_t fuse_loop_dio(struct file *filp, const struct iovec *iov,
> -        unsigned long nr_segs, loff_t *ppos, int rw)
> -{
> - const struct iovec *vector = iov;
> - ssize_t ret = 0;
> -
> - while (nr_segs > 0) {
> -  void __user *base;
> -  size_t len;
> -  ssize_t nr;
> -
> -  base = vector->iov_base;
> -  len = vector->iov_len;
> -  vector++;
> -  nr_segs--;
> -
> -  if (rw == WRITE)
> -   nr = __fuse_direct_write(filp, base, len, ppos);
> -  else
> -   nr = fuse_direct_read(filp, base, len, ppos);
> -
> -  if (nr < 0) {
> -   if (!ret)
> -    ret = nr;
> -   break;
> -  }
> -  ret += nr;
> -  if (nr != len)
> -   break;
> - }
> -
> - return ret;
> -}
> -
> -
```

```
>  static ssize_t
>  fuse_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
>     loff_t offset, unsigned long nr_segs)
> @@ -2212,7 +2211,10 @@ fuse_direct_IO(int rw, struct kiocb *iocb, const struct iovec *iov,
>   file = iocb->ki_filp;
>   pos = offset;
>
> - ret = fuse_loop_dio(file, iov, nr_segs, &pos, rw);
> + if (rw == WRITE)
> +  ret = __fuse_direct_write(file, iov, nr_segs, &pos);
> + else
> +  ret = __fuse_direct_read(file, iov, nr_segs, &pos);
>
>   return ret;
>  }
```

Subject: Re: [PATCH 10/11] fuse: optimize fuse_get_user_pages() - v2
Posted by Miklos Szeredi on Thu, 25 Oct 2012 14:50:45 GMT

View Forum Message <> Reply to Message

Maxim Patlasov <mpatlasov@parallels.com> writes:

```
> Let fuse_get_user_pages() pack as many iov-s to a single fuse_req as
> possible. This is very beneficial in case of iov[] consisting of many
> iov-s of relatively small sizes (e.g. PAGE_SIZE).
>
> Changed in v2:
>  - renamed local vars in fuse_page_descs_length_init() to be more readable
>
> Signed-off-by: Maxim Patlasov <mpatlasov@parallels.com>
> ---
>  fs/fuse/file.c |   94 +++++++++++++++++++++++++++++++++++++++++++++++++--------------------
>  1 files changed, 58 insertions(+), 36 deletions(-)
>
> diff --git a/fs/fuse/file.c b/fs/fuse/file.c
> index db9efb5..4d30697 100644
> --- a/fs/fuse/file.c
> +++ b/fs/fuse/file.c
> @@ -1047,13 +1047,24 @@ static void fuse_release_user_pages(struct fuse_req *req, int
write)
>   }
>  }
>
> -static inline void fuse_page_descs_length_init(struct fuse_req *req)
> +static inline void fuse_page_descs_length_init(struct fuse_req *req,
> +           unsigned index, int nr_pages)
>  {
```

> - int i;
> + while (nr_pages-- > 0)

Please avoid such constructs. They are hard to read and easy to mess
up.

Use an auxiliary local variable, just like you used to.

> + req->page_descs[index + nr_pages].length = PAGE_SIZE -
> +   req->page_descs[index + nr_pages].offset;
> +}
>
> - for (i = 0; i < req->num_pages; i++)
> - req->page_descs[i].length = PAGE_SIZE -
> -   req->page_descs[i].offset;
> +static inline unsigned long fuse_get_ua(const struct iovec *iov,

fuse_get_user_addr()

> +   size_t iov_offset)
> +{
> + return (unsigned long)iov->iov_base + iov_offset;
> +}
> +
> +static inline size_t fuse_get_fr_sz(const struct iovec *iov, size_t
> iov_offset,

iov_iter_single_seg_count() is the function you are reimplementing here,
I guess.

I haven't reviewed the as I think it will automatically get cleaner if
you switch to iov_iter.

> +   size_t max_size)
> +{
> + return min_t(size_t, iov->iov_len - iov_offset, max_size);
> }
>
> static int fuse_get_user_pages(struct fuse_req *req,
> @@ -1062,14 +1073,12 @@ static int fuse_get_user_pages(struct fuse_req *req,
>   size_t *iov_offset_p,
>   size_t *nbytesp, int write)
> {
> - size_t nbytes = *nbytesp;
> - size_t frag_size = min_t(size_t, nbytes, (*iov_pp)->iov_len - *iov_offset_p);
> - unsigned long user_addr = (unsigned long)(*iov_pp)->iov_base + *iov_offset_p;
> - unsigned offset = user_addr & ~PAGE_MASK;

```
> - int npages;
> + size_t nbytes = 0;  /* # bytes already packed in req */
>
>   /* Special case for kernel I/O: can copy directly into the buffer */
>   if (segment_eq(get_fs(), KERNEL_DS)) {
> +  unsigned long user_addr = fuse_get_ua(*iov_pp, *iov_offset_p);
> +
>    if (write)
>     req->in.args[1].value = (void *) user_addr;
>    else
> @@ -1077,42 +1086,55 @@ static int fuse_get_user_pages(struct fuse_req *req,
>
>    (*iov_pp)++;
>    (*nr_segs_p)--;
> -  *nbytesp = frag_size;
> +  *nbytesp = fuse_get_fr_sz(*iov_pp, *iov_offset_p, *nbytesp);
>    return 0;
>   }
>
> - nbytes = min_t(size_t, frag_size, FUSE_MAX_PAGES_PER_REQ << PAGE_SHIFT);
> - npages = (nbytes + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
> - npages = clamp(npages, 1, FUSE_MAX_PAGES_PER_REQ);
> - npages = get_user_pages_fast(user_addr, npages, !write, req->pages);
> - if (npages < 0)
> -  return npages;
> + while (nbytes < *nbytesp && req->num_pages < FUSE_MAX_PAGES_PER_REQ) {
> +  int npages;
> +  unsigned long user_addr = fuse_get_ua(*iov_pp, *iov_offset_p);
> +  unsigned offset = user_addr & ~PAGE_MASK;
> +  size_t frag_size = fuse_get_fr_sz(*iov_pp, *iov_offset_p,
> +         *nbytesp - nbytes);
>
> - req->num_pages = npages;
> - req->page_descs[0].offset = offset;
> - fuse_page_descs_length_init(req);
> +  int n = FUSE_MAX_PAGES_PER_REQ - req->num_pages;
> +  frag_size = min_t(size_t, frag_size, n << PAGE_SHIFT);
>
> - if (write)
> -  req->in.argpages = 1;
> - else
> -  req->out.argpages = 1;
> +  npages = (frag_size + offset + PAGE_SIZE - 1) >> PAGE_SHIFT;
> +  npages = clamp(npages, 1, n);
>
> - nbytes = (req->num_pages << PAGE_SHIFT) - req->page_descs[0].offset;
> +  npages = get_user_pages_fast(user_addr, npages, !write,
> +           &req->pages[req->num_pages]);
```

```
> +  if (npages < 0)
> +   return npages;
>
> - if (frag_size < nbytes)
> -  req->page_descs[req->num_pages - 1].length -=
> -   nbytes - frag_size;
> +  frag_size = min_t(size_t, frag_size,
> +      (npages << PAGE_SHIFT) - offset);
> +  nbytes += frag_size;
>
> - *nbytesp = min(frag_size, nbytes);
> +  if (frag_size < (*iov_pp)->iov_len - *iov_offset_p) {
> +   *iov_offset_p += frag_size;
> +  } else {
> +   (*iov_pp)++;
> +   (*nr_segs_p)--;
> +   *iov_offset_p = 0;
> +  }
>
> - if (*nbytesp < (*iov_pp)->iov_len - *iov_offset_p) {
> -  *iov_offset_p += *nbytesp;
> - } else {
> -  (*iov_pp)++;
> -  (*nr_segs_p)--;
> -  *iov_offset_p = 0;
> +  req->page_descs[req->num_pages].offset = offset;
> +  fuse_page_descs_length_init(req, req->num_pages, npages);
> +
> +  req->num_pages += npages;
> +  req->page_descs[req->num_pages - 1].length -=
> +   (npages << PAGE_SHIFT) - offset - frag_size;
>   }
>
> + if (write)
> +  req->in.argpages = 1;
> + else
> +  req->out.argpages = 1;
> +
> + *nbytesp = nbytes;
> +
>   return 0;
>  }
>
> @@ -1954,7 +1976,7 @@ long fuse_do_ioctl(struct file *file, unsigned int cmd, unsigned long
arg,
>   }
>   memcpy(req->pages, pages, sizeof(req->pages[0]) * num_pages);
>   req->num_pages = num_pages;
```

> - fuse_page_descs_length_init(req);
> + fuse_page_descs_length_init(req, 0, req->num_pages);
>
> /* okay, let's send it to the client */
> req->in.h.opcode = FUSE_IOCTL;

Subject: Re: [PATCH 08/11] fuse: use req->page_descs[] for argpages cases
Posted by Maxim Patlasov on Thu, 25 Oct 2012 15:38:30 GMT
View Forum Message <> Reply to Message

Miklos,

> Maxim Patlasov <mpatlasov@parallels.com> writes:
>
>> @@ -888,11 +888,11 @@ static int fuse_copy_pages(struct fuse_copy_state *cs, unsigned nbytes,
>> {
>>   unsigned i;
>>   struct fuse_req *req = cs->req;
>> - unsigned offset = req->page_descs[0].offset;
>> - unsigned count = min(nbytes, (unsigned) PAGE_SIZE - offset);
>>
>>   for (i = 0; i < req->num_pages && (nbytes || zeroing); i++) {
>>   int err;
>> + unsigned offset = req->page_descs[i].offset;
>> + unsigned count = min(nbytes, req->page_descs[i].length);
> Wouldn't it be cleaner if callers calculated the last page's .length
> value from the total number of bytes? So this would just be
>
> unsigned count = req->page_descs[i].length;
>
> And at the end of the function we can assert that nbytes went to exactly
> zero with a WARN_ON().
>
> But this is a change that needs careful testing, so maybe we're better
> off having that as a separate incremental patch later...

It cannot be as simple as 'unsigned count = req->page_descs[i].length' because in case of short reads 'nbytes' (coming from userspace) can be unpredictably small. Modulo you share my opinion that a caller of fuse_copy_pages() shouldn't modify req->page_descs[i].length.

As for WARN_ON(), we could probably guarantee that 'nbytes' <= capacity(req->pages[]) in WRITEs, but in READs, 'nbytes' comes from userspace and I'm not sure it's OK to clutter logs due to misbehaved userspace fuse (if we get 'nbytes' unexpectedly large).

Thanks,
Maxim