

[PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2003-09/1563.html>

From: Jamie Lokier (jamie_at_shareable.org)

Date: 09/05/03

Date: Fri, 5 Sep 2003 05:57:38 +0100

To: Linus Torvalds <torvalds@osdl.org>

Folks listed in To & Cc, please take a look. Barring bugs and fundamental disagreements, this should be the final version.

(Linus may have something to say about the occurrence of VM_MAYSHARE. Otherwise there's nothing contentious).

Thanks,
— Jamie

Patch: `futex-fixes-2.6.0-test4-11jl`

This patch is against `test-2.6.0-test4`. It changes the way futexes are indexed, so that they don't pin pages. It also fixes some bugs with private mappings and COW pages.

Currently, all futexes look up the page at the userspace address and pin it, using the pair (page,offset) as an index into a table of waiting futexes. Any page with a futex waiting on it remains pinned in RAM, which is a problem when many futexes are used, especially with FUTEX_FD.

Another problem is that the page is not always the correct one, if it can be changed later by a COW (copy on write) operation. This can happen when waiting on a futex without writing to it after `fork()`, `exec()` or `mmap()`, if the page is then written to before attempting to wake a futex at the same address.

There are two symptoms of the COW problem: 1. The wrong process can receive wakeups; 2. A process can fail to receive required wakeups.

This patch fixes both by changing the indexing so that VM_SHARED mappings use the triple (inode,offset,index), and private mappings use the pair (mm,virtual_address).

The former correctly handles all shared mappings, including tmpfs and

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

therefore all kinds of shared memory (IPC shm, /dev/shm and MAP_ANON|MAP_SHARED). This works because every mapping which is VM_SHARED has an associated non-zero vma->vm_file, and hence inode. (This is ensured in do_mmap_pgoff, where it calls shmem_zero_setup).

The latter handles all private mappings, both files and anonymous. It isn't affected by COW, because it doesn't care about the actual pages, just the virtual address.

The only obvious problem is that mremap() can move a private mapping without informing futexes waiting on that mapping. However, mremap() was already broken with futexes, because it doesn't update the vcache, so this just changes an existing bug.

(The next patch in this series fixes mremap()).

The patch has a few bonuses:

1. It removes the vcache implementation, as only futexes were using it, and they don't any more.
2. Removing the vcache should make COW page faults a bit faster.
3. Futex operations no longer take the page table lock, walk the page table, fault in pages that aren't mapped in the page table, or do a vcache hash lookup – they are mostly a simple offset calculation with one hash for the futex table. So they should be noticeably faster.

Special thanks to Hugh Dickins, Andrew Morton and Rusty Russell for insightful feedback. All suggestions are included.

Enjoy,
-- Jamie

```
diff -urN --exclude-from=dontdiff orig-2.6.0-test4/include/linux/mm.h
newfut-2.6.0-test4/include/linux/mm.h
--- orig-2.6.0-test4/include/linux/mm.h 2003-09-02 23:06:10.000000000 +0100
+++ newfut-2.6.0-test4/include/linux/mm.h 2003-09-05 00:50:09.000000000 +0100
@@ -110,6 +110,7 @@
#define VM_RESERVED 0x00080000 /* Don't unmap it from swap_out */
#define VM_ACCOUNT 0x00100000 /* Is a VM accounted object */
#define VM_HUGETLB 0x00400000 /* Huge TLB Page VM */
+#define VM_NONLINEAR 0x00800000 /* Is non-linear (remap_file_pages) */

#ifdef VM_STACK_DEFAULT_FLAGS /* arch can override this */
#define VM_STACK_DEFAULT_FLAGS VM_DATA_DEFAULT_FLAGS
diff -urN --exclude-from=dontdiff orig-2.6.0-test4/include/linux/vcache.h
newfut-2.6.0-test4/include/linux/vcache.h
--- orig-2.6.0-test4/include/linux/vcache.h 2003-07-08 21:44:12.000000000 +0100
+++ newfut-2.6.0-test4/include/linux/vcache.h 1970-01-01 01:00:00.000000000 +0100
```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```
@@ -1,26 +0,0 @@
-/*
- * virtual => physical mapping cache support.
- */
-#ifndef _LINUX_VCACHE_H
-#define _LINUX_VCACHE_H
-
-#typedef struct vcache_s {
- unsigned long address;
- struct mm_struct *mm;
- struct list_head hash_entry;
- void (*callback)(struct vcache_s *data, struct page *new_page);
-} vcache_t;
-
-extern spinlock_t vcache_lock;
-
-extern void __attach_vcache(vcache_t *vcache,
- unsigned long address,
- struct mm_struct *mm,
- void (*callback)(struct vcache_s *data, struct page *new_page));
-
-extern void __detach_vcache(vcache_t *vcache);
-
-extern void invalidate_vcache(unsigned long address, struct mm_struct *mm,
- struct page *new_page);
-
-#endif
diff -urN --exclude-from=dontdiff orig-2.6.0-test4/kernel/futex.c newfut-2.6.0-test4/kernel/futex.c
--- orig-2.6.0-test4/kernel/futex.c 2003-07-08 21:44:25.000000000 +0100
+++ newfut-2.6.0-test4/kernel/futex.c 2003-09-05 03:23:17.000000000 +0100
@@ -5,6 +5,9 @@
 * Generalized futexes, futex requeueing, misc fixes by Ingo Molnar
 * (C) Copyright 2003 Red Hat Inc, All Rights Reserved
 *
+ * Removed page pinning, fix privately mapped COW pages and other cleanups
+ * (C) Copyright 2003 Jamie Lokier
+ *
 * Thanks to Ben LaHaise for yelling "hashed waitqueues" loudly
 * enough at me, Linus for the original (flawed) idea, Matthew
 * Kirkwood for proof-of-concept implementation.
@@ -32,12 +35,32 @@
#include <linux/hash.h>
#include <linux/init.h>
#include <linux/futex.h>
-#include <linux/vcache.h>
#include <linux/mount.h>
+#include <linux/pagemap.h>

#define FUTEX_HASHBITS 8

/*
```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

+ * Futexes are matched on equal values of this key.
+ * The key type depends on whether it's a shared or private mapping.
+ */
+union futex_key {
+ struct {
+ unsigned long pgoff;
+ struct inode *inode;
+ } shared;
+ struct {
+ unsigned long uaddr;
+ struct mm_struct *mm;
+ } private;
+ struct {
+ unsigned long word;
+ void *ptr;
+ } both;
+ int offset;
+};
+
+/*
+ * We use this hashed waitqueue instead of a normal wait_queue_t, so
+ * we can wake only the relevant ones (hashed queues may be shared):
+ */
@@ -45,12 +68,8 @@
     struct list_head list;
     wait_queue_head_t waiters;

- /* Page struct and offset within it. */
- struct page *page;
- int offset;
-
- /* the virtual => physical COW-safe cache */
- vcache_t vcache;
+ /* Key which the futex is hashed on. */
+ union futex_key key;

     /* For fd, sigio sent using these. */
     int fd;
@@ -67,111 +86,149 @@
 static struct vfsmount *futex_mnt;

/*
- * These are all locks that are necessary to look up a physical
- * mapping safely, and modify/search the futex hash, atomically:
+ * We hash on the keys returned from get_futex_key (see below).
+ */
- static inline void lock_futex_mm(void)
+ static inline struct list_head *hash_futex(union futex_key *key)
     {
- spin_lock(&current->mm->page_table_lock);
- spin_lock(&vcache_lock);

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

- spin_lock(&futex_lock);
-}
-
-static inline void unlock_futex_mm(void)
-{
- spin_unlock(&futex_lock);
- spin_unlock(&vcache_lock);
- spin_unlock(&current->mm->page_table_lock);
+ return &futex_queues[hash_long(key->both.word
+ + (unsigned long) key->both.ptr
+ + key->offset, FUTEX_HASHBITS)];
}

/*
- * The physical page is shared, so we can hash on its address:
+ * Return 1 if two futex_keys are equal, 0 otherwise.
*/
-static inline struct list_head *hash_futex(struct page *page, int offset)
+static inline int match_futex(union futex_key *key1, union futex_key *key2)
{
- return &futex_queues[hash_long((unsigned long)page + offset,
- FUTEX_HASHBITS)];
+ return (key1->both.word == key2->both.word
+ && key1->both.ptr == key2->both.ptr
+ && key1->offset == key2->offset);
}

/*
- * Get kernel address of the user page and pin it.
+ * Get parameters which are the keys for a futex.
+ *
+ * For shared mappings, it's (page->index, vma->vm_file->f_dentry->d_inode,
+ * offset_within_page). For private mappings, it's (uaddr, current->mm).
+ * We can usually work out the index without swapping in the page.
+ *
- * Must be called with (and returns with) all futex-MM locks held.
+ * Returns: 0, or negative error code.
+ * The key words are stored in *key on success.
+ *
+ * Should be called with &current->mm->mmap_sem,
+ * but NOT &futex_lock or &current->mm->page_table_lock.
*/
-static inline struct page *__pin_page_atomic (struct page *page)
-{
- if (!PageReserved(page))
- get_page(page);
- return page;
-}
-
-static struct page *__pin_page(unsigned long addr)
+static int get_futex_key(unsigned long uaddr, union futex_key *key)

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

{
    struct mm_struct *mm = current->mm;
- struct page *page, *tmp;
+ struct vm_area_struct *vma;
+ struct page *page;
    int err;

    /*
- * Do a quick atomic lookup first – this is the fastpath.
+ * The futex address must be "naturally" aligned.
+ */
+ key->offset = uaddr % PAGE_SIZE;
+ if (unlikely((key->offset % sizeof(u32)) != 0))
+ return -EINVAL;
+ uaddr -= key->offset;
+
+ /*
+ * The futex is hashed differently depending on whether
+ * it's in a shared or private mapping. So check vma first.
+ */
+ vma = find_extend_vma(mm, uaddr);
+ if (unlikely(!vma))
+ return -EFAULT;
+
+ /*
+ * Permissions.
+ */
- page = follow_page(mm, addr, 0);
- if (likely(page != NULL))
- return __pin_page_atomic(page);
+ if (unlikely((vma->vm_flags & (VM_IO|VM_READ)) != VM_READ))
+ return (vma->vm_flags & VM_IO) ? -EPERM : -EACCES;

    /*
- * No luck – need to fault in the page:
+ * Private mappings are handled in a simple way.
+ *
+ * NOTE: When userspace waits on a MAP_SHARED mapping, even if
+ * it's a read-only handle, it's expected that futexes attach to
+ * the object not the particular process. Therefore we use
+ * VM_MAYSHARE here, not VM_SHARED which is restricted to shared
+ * mappings of _writable_ handles.
+ */
-repeat_lookup:
+ if (likely(!(vma->vm_flags & VM_MAYSHARE))) {
+ key->private.mm = mm;
+ key->private.uaddr = uaddr;
+ return 0;
+ }

- unlock_futex_mm();

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

+ /*
+ * Linear mappings are also simple.
+ */
+ key->shared.inode = vma->vm_file->f_dentry->d_inode;
+ if (likely(!(vma->vm_flags & VM_NONLINEAR))) {
+ key->shared.pgoff = (((uaddr - vma->vm_start) >> PAGE_SHIFT)
+ + vma->vm_pgoff);
+ return 0;
+ }

- down_read(&mm->mmap_sem);
- err = get_user_pages(current, mm, addr, 1, 0, 0, &page, NULL);
- up_read(&mm->mmap_sem);
+ /*
+ * We could walk the page table to read the non-linear
+ * pte, and get the page index without fetching the page
+ * from swap. But that's a lot of code to duplicate here
+ * for a rare case, so we simply fetch the page.
+ */

- lock_futex_mm();
+ /*
+ * Do a quick atomic lookup first – this is the fastpath.
+ */
+ spin_lock(&current->mm->page_table_lock);
+ page = follow_page(mm, uaddr, 0);
+ if (likely(page != NULL)) {
+ key->shared.pgoff =
+ page->index << (PAGE_CACHE_SHIFT - PAGE_SHIFT);
+ spin_unlock(&current->mm->page_table_lock);
+ return 0;
+ }
+ spin_unlock(&current->mm->page_table_lock);

- if (err < 0)
- return NULL;
+ /*
+ * Since the faulting happened with locks released, we have to
+ * check for races:
+ * Do it the general way.
+ */
- tmp = follow_page(mm, addr, 0);
- if (tmp != page) {
+ err = get_user_pages(current, mm, uaddr, 1, 0, 0, &page, NULL);
+ if (err >= 0) {
+ key->shared.pgoff =
+ page->index << (PAGE_CACHE_SHIFT - PAGE_SHIFT);
+ put_page(page);
- goto repeat_lookup;
+ }
-

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

- return page;
+ return err;
}

+
/*
 * Wake up all waiters hashed on the physical page that is mapped
 * to this virtual address:
 */
-static inline int futex_wake(unsigned long uaddr, int offset, int num)
+static inline int futex_wake(unsigned long uaddr, int num)
{
    struct list_head *i, *next, *head;
- struct page *page;
- int ret = 0;
+ union futex_key key;
+ int ret;

- lock_futex_mm();
+ down_read(&current->mm->mmap_sem);

- page = __pin_page(uaddr - offset);
- if (!page) {
- unlock_futex_mm();
- return -EFAULT;
- }
+ ret = get_futex_key(uaddr, &key);
+ if (unlikely(ret != 0))
+ goto out;

- head = hash_futex(page, offset);
+ head = hash_futex(&key);

+ spin_lock(&futex_lock);
    list_for_each_safe(i, next, head) {
        struct futex_q *this = list_entry(i, struct futex_q, list);

- if (this->page == page && this->offset == offset) {
+ if (match_futex (&this->key, &key)) {
            list_del_init(i);
- __detach_vcache(&this->vcache);
            wake_up_all(&this->waiters);
            if (this->filp)
                send_sigio(&this->filp->f_owner, this->fd, POLL_IN);
@@ -180,113 +237,74 @@
                break;
        }
    }
+ spin_unlock(&futex_lock);

- unlock_futex_mm();

```

Linux–Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

– put_page(page);
–
+out:
+ up_read(&current->mm->mmap_sem);
    return ret;
}

/*
– * This gets called by the COW code, we have to rehash any
– * futexes that were pending on the old physical page, and
– * rehash it to the new physical page. The pagetable_lock
– * and vcache_lock is already held:
– */
–static void futex_vcache_callback(vcache_t *vcache, struct page *new_page)
–{
– struct futex_q *q = container_of(vcache, struct futex_q, vcache);
– struct list_head *head = hash_futex(new_page, q->offset);
–
– spin_lock(&futex_lock);
–
– if (!list_empty(&q->list)) {
– put_page(q->page);
– q->page = new_page;
– __pin_page_atomic(new_page);
– list_del(&q->list);
– list_add_tail(&q->list, head);
– }
–
– spin_unlock(&futex_lock);
–}
–
–/*
– * Requeue all waiters hashed on one physical page to another
– * physical page.
– */
–static inline int futex_requeue(unsigned long uaddr1, int offset1,
– unsigned long uaddr2, int offset2, int nr_wake, int nr_requeue)
+static inline int futex_requeue(unsigned long uaddr1, unsigned long uaddr2,
+ int nr_wake, int nr_requeue)
{
    struct list_head *i, *next, *head1, *head2;
– struct page *page1 = NULL, *page2 = NULL;
– int ret = 0;
+ union futex_key key1, key2;
+ int ret;

– lock_futex_mm();
+ down_read(&current->mm->mmap_sem);

– page1 = __pin_page(uaddr1 – offset1);
– if (!page1)

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

+ ret = get_futex_key(uaddr1, &key1);
+ if (unlikely(ret != 0))
    goto out;
- page2 = __pin_page(uaddr2 - offset2);
- if (!page2)
+ ret = get_futex_key(uaddr2, &key2);
+ if (unlikely(ret != 0))
    goto out;

- head1 = hash_futex(page1, offset1);
- head2 = hash_futex(page2, offset2);
+ head1 = hash_futex(&key1);
+ head2 = hash_futex(&key2);

+ spin_lock(&futex_lock);
    list_for_each_safe(i, next, head1) {
        struct futex_q *this = list_entry(i, struct futex_q, list);

- if (this->page == page1 && this->offset == offset1) {
+ if (match_futex (&this->key, &key1)) {
        list_del_init(i);
- __detach_vcache(&this->vcache);
        if (++ret <= nr_wake) {
            wake_up_all(&this->waiters);
            if (this->filp)
                send_sigio(&this->filp->f_owner,
                    this->fd, POLL_IN);
        } else {
- put_page(this->page);
- __pin_page_atomic (page2);
            list_add_tail(i, head2);
- __attach_vcache(&this->vcache, uaddr2,
- current->mm, futex_vcache_callback);
- this->offset = offset2;
- this->page = page2;
+ this->key = key2;
            if (ret - nr_wake >= nr_requeue)
                break;
        }
    }
}
+ spin_unlock(&futex_lock);

out:
- unlock_futex_mm();
-
- if (page1)
- put_page(page1);
- if (page2)
- put_page(page2);
-

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

+ up_read(&current->mm->mmap_sem);
    return ret;
}

-static inline void __queue_me(struct futex_q *q, struct page *page,
- unsigned long uaddr, int offset,
- int fd, struct file *filp)
+static inline void queue_me(struct futex_q *q, union futex_key *key,
+ int fd, struct file *filp)
{
- struct list_head *head = hash_futex(page, offset);
+ struct list_head *head = hash_futex(key);

- q->offset = offset;
+ q->key = *key;
    q->fd = fd;
    q->filp = filp;
- q->page = page;

+ spin_lock(&futex_lock);
    list_add_tail(&q->list, head);
- /*
- * We register a futex callback to this virtual address,
- * to make sure a COW properly rehashes the futex-queue.
- */
- __attach_vcache(&q->vcache, uaddr, current->mm, futex_vcache_callback);
+ spin_unlock(&futex_lock);
}

/* Return 1 if we were still queued (ie. 0 means we were woken) */
@@ -294,83 +312,107 @@
{
    int ret = 0;

- spin_lock(&vcache_lock);
    spin_lock(&futex_lock);
    if (!list_empty(&q->list)) {
        list_del(&q->list);
- __detach_vcache(&q->vcache);
        ret = 1;
    }
    spin_unlock(&futex_lock);
- spin_unlock(&vcache_lock);
    return ret;
}

-static inline int futex_wait(unsigned long uaddr,
- int offset,
- int val,
- unsigned long time)
+static inline int futex_wait(unsigned long uaddr, int val, unsigned long time)

```

```

{
    DECLARE_WAITQUEUE(wait, current);
- int ret = 0, curval;
- struct page *page;
+ int ret, curval;
+ union futex_key key;
    struct futex_q q;

+ try_again:
    init_waitqueue_head(&q.waiters);

- lock_futex_mm();
+ down_read(&current->mm->mmap_sem);

- page = __pin_page(uaddr - offset);
- if (!page) {
- unlock_futex_mm();
- return -EFAULT;
- }
- __queue_me(&q, page, uaddr, offset, -1, NULL);
+ ret = get_futex_key(uaddr, &key);
+ if (unlikely(ret != 0))
+ goto out_release_sem;
+
+ queue_me(&q, &key, -1, NULL);

    /*
- * Page is pinned, but may no longer be in this address space.
- * It cannot schedule, so we access it with the spinlock held.
+ * Access the page after the futex is queued.
+ * We hold the mmap semaphore, so the mapping cannot have changed
+ * since we looked it up.
    */
    if (get_user(curval, (int *)uaddr) != 0) {
- unlock_futex_mm();
        ret = -EFAULT;
- goto out;
+ goto out_unqueue;
    }
    if (curval != val) {
- unlock_futex_mm();
        ret = -EWOULDBLOCK;
- goto out;
+ goto out_unqueue;
    }
+
+ /*
+ * Now the futex is queued and we have checked the data, we
+ * don't want to hold mmap_sem while we sleep.
+ */
+ up_read(&current->mm->mmap_sem);

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

+
+   /*
+   * The get_user() above might fault and schedule so we
+   * cannot just set TASK_INTERRUPTIBLE state when queueing
+   * ourselves into the futex hash. This code thus has to
+   * There might have been scheduling since the queue_me(), as we
+   * cannot hold a spinlock across the get_user() in case it
+   * faults. So we cannot just set TASK_INTERRUPTIBLE state when
+   * queueing ourselves into the futex hash. This code thus has to
+   *   * rely on the futex_wake() code doing a wakeup after removing
+   *   * the waiter from the list.
+   */
+   add_wait_queue(&q.waiters, &wait);
+ spin_lock(&futex_lock);
+   set_current_state(TASK_INTERRUPTIBLE);
+   if (!list_empty(&q.list)) {
+   unlock_futex_mm();
+   time = schedule_timeout(time);
+
+   + if (unlikely(list_empty(&q.list))) {
+   + /*
+   + * We were woken already.
+   + */
+   + spin_unlock(&futex_lock);
+   + set_current_state(TASK_RUNNING);
+   + return 0;
+   }
+
+   + spin_unlock(&futex_lock);
+   + time = schedule_timeout(time);
+   + set_current_state(TASK_RUNNING);
+
+   /*
+   * NOTE: we don't remove ourselves from the waitqueue because
+   * we are the only user of it.
+   */
+
+   - if (time == 0) {
+   - ret = -ETIMEDOUT;
+   - goto out;
+   - }
+
+   + /*
+   + * Were we woken or interrupted for a valid reason?
+   + */
+   + ret = unqueue_me(&q);
+   + if (ret == 0)
+   + return 0;
+   + if (time == 0)
+   + return -ETIMEDOUT;
+   + if (signal_pending(current))
+   - ret = -EINTR;

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

-out:
- /* Were we woken up anyway? */
+ return -EINTR;
+
+ /*
+ * No, it was a spurious wakeup. Try again. Should never happen. :)
+ */
+ goto try_again;
+
+ out_unqueue:
+ /*
+ * Were we unqueued anyway?
+ */
+     if (!unqueue_me(&q))
+         ret = 0;
- put_page(q.page);
-
+ out_release_sem:
+ up_read(&current->mm->mmap_sem);
+     return ret;
+ }

@@ -379,7 +421,6 @@
+     struct futex_q *q = filp->private_data;

+     unqueue_me(q);
- put_page(q->page);
+     kfree(filp->private_data);
+     return 0;
+ }

@@ -407,12 +448,12 @@

+ /* Signal allows caller to avoid the race which would occur if they
+     set the sigio stuff up afterwards. */
- static int futex_fd(unsigned long uaddr, int offset, int signal)
+ static int futex_fd(unsigned long uaddr, int signal)
+ {
- struct page *page = NULL;
+     struct futex_q *q;
+ union futex_key key;
+     struct file *filp;
- int ret;
+ int ret, err;

+     ret = -EINVAL;
+     if (signal < 0 || signal > _NSIG)
@@ -451,69 +492,47 @@
+         goto out;
+     }

- lock_futex_mm();

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

-
- page = __pin_page(uaddr - offset);
- if (!page) {
- unlock_futex_mm();
+ down_read(&current->mm->mmap_sem);
+ err = get_futex_key(uaddr, &key);
+ up_read(&current->mm->mmap_sem);

+ if (unlikely(err != 0)) {
        put_unused_fd(ret);
        put_filp(filp);
        kfree(q);
- return -EFAULT;
+ return err;
    }

        init_waitqueue_head(&q->waiters);
        filp->private_data = q;

- __queue_me(q, page, uaddr, offset, ret, filp);
-
- unlock_futex_mm();
+ queue_me(q, &key, ret, filp);

        /* Now we map fd to filp, so userspace can access it */
        fd_install(ret, filp);
- page = NULL;
out:
- if (page)
- put_page(page);
    return ret;
}

long do_futex(unsigned long uaddr, int op, int val, unsigned long timeout,
              unsigned long uaddr2, int val2)
{
- unsigned long pos_in_page;
    int ret;

- pos_in_page = uaddr % PAGE_SIZE;
-
- /* Must be "naturally" aligned */
- if (pos_in_page % sizeof(u32))
- return -EINVAL;
-
        switch (op) {
        case FUTEX_WAIT:
- ret = futex_wait(uaddr, pos_in_page, val, timeout);
+ ret = futex_wait(uaddr, val, timeout);
            break;
        case FUTEX_WAKE:

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

- ret = futex_wake(uaddr, pos_in_page, val);
+ ret = futex_wake(uaddr, val);
    break;
    case FUTEX_FD:
        /* non-zero val means F_SETOWN(getpid()) & F_SETSIG(val) */
- ret = futex_fd(uaddr, pos_in_page, val);
+ ret = futex_fd(uaddr, val);
        break;
    case FUTEX_REQUEUE:
- {
- unsigned long pos_in_page2 = uaddr2 % PAGE_SIZE;
-
- /* Must be "naturally" aligned */
- if (pos_in_page2 % sizeof(u32))
- return -EINVAL;
-
- ret = futex_requeue(uaddr, pos_in_page, uaddr2, pos_in_page2,
- val, val2);
+ ret = futex_requeue(uaddr, uaddr2, val, val2);
        break;
- }
    default:
        ret = -ENOSYS;
    }
diff -urN --exclude-from=dontdiff orig-2.6.0-test4/mm/fremap.c newfut-2.6.0-test4/mm/fremap.c
--- orig-2.6.0-test4/mm/fremap.c 2003-07-08 21:44:29.000000000 +0100
+++ newfut-2.6.0-test4/mm/fremap.c 2003-09-05 01:16:17.000000000 +0100
@@ -140,9 +140,10 @@
        return err;
#endif

- down_read(&mm->mmap_sem);
-
+ /* We need down_write() to change vma->vm_flags. */
+ down_write(&mm->mmap_sem);
    vma = find_vma(mm, start);
+
    /*
     * Make sure the vma is shared, that it supports prefaulting,
     * and that the remapped range is valid and fully within
@@ -151,11 +152,27 @@
    if (vma && (vma->vm_flags & VM_SHARED) &&
        vma->vm_ops && vma->vm_ops->populate &&
            end > start && start >= vma->vm_start &&
- end <= vma->vm_end)
- err = vma->vm_ops->populate(vma, start, size, vma->vm_page_prot,
- pgoff, flags & MAP_NONBLOCK);
+ end <= vma->vm_end) {
+
+ /* Must set VM_NONLINEAR before any pages are populated. */
+ if (pgoff != ((start - vma->vm_start) >> PAGE_SHIFT) + vma->vm_pgoff)

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

+ vma->vm_flags |= VM_NONLINEAR;
+
+ /* ->populate can take a long time, so downgrade the lock. */
+ downgrade_write(&mm->mmap_sem);
+ err = vma->vm_ops->populate(vma, start, size,
+ vma->vm_page_prot,
+ pgoff, flags & MAP_NONBLOCK);

- up_read(&mm->mmap_sem);
+ /*
+ * We can't clear VM_NONLINEAR because we'd have to do
+ * it after ->populate completes, and that would prevent
+ * downgrading the lock. (Locks can't be upgraded).
+ */
+ up_read(&mm->mmap_sem);
+ } else {
+ up_write(&mm->mmap_sem);
+ }

    return err;
}
diff -urN --exclude-from=dontdiff orig-2.6.0-test4/mm/Makefile newfut-2.6.0-test4/mm/Makefile
--- orig-2.6.0-test4/mm/Makefile 2003-07-08 21:44:29.000000000 +0100
+++ newfut-2.6.0-test4/mm/Makefile 2003-09-05 00:50:09.000000000 +0100
@@ -9,6 +9,6 @@

obj-y := bootmem.o filemap.o mempool.o oom_kill.o fadvise.o \
        page_alloc.o page-writeback.o pdflush.o readahead.o \
- slab.o swap.o truncate.o vcache.o vmscan.o $(mmu-y)
+ slab.o swap.o truncate.o vmscan.o $(mmu-y)

obj-$(CONFIG_SWAP) += page_io.o swap_state.o swapfile.o
diff -urN --exclude-from=dontdiff orig-2.6.0-test4/mm/memory.c newfut-2.6.0-test4/mm/memory.c
--- orig-2.6.0-test4/mm/memory.c 2003-09-02 23:06:13.000000000 +0100
+++ newfut-2.6.0-test4/mm/memory.c 2003-09-05 00:50:09.000000000 +0100
@@ -43,7 +43,6 @@
#include <linux/swap.h>
#include <linux/highmem.h>
#include <linux/pagemap.h>
-#include <linux/vcache.h>
#include <linux/rmap-locking.h>

#include <asm/pgalloc.h>
@@ -960,7 +959,6 @@
static inline void break_cow(struct vm_area_struct * vma, struct page * new_page, unsigned long address,
        pte_t *page_table)
{
- invalidate_vcache(address, vma->vm_mm, new_page);
    flush_cache_page(vma, address);
    establish_pte(vma, address, page_table, pte_mkwrite(pte_mkdirty(mk_pte(new_page,
vma->vm_page_prot))));

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```

}
diff -urN --exclude-from=dontdiff orig-2.6.0-test4/mm/vcache.c newfut-2.6.0-test4/mm/vcache.c
--- orig-2.6.0-test4/mm/vcache.c 2003-07-08 21:44:31.000000000 +0100
+++ newfut-2.6.0-test4/mm/vcache.c 1970-01-01 01:00:00.000000000 +0100
@@ -1,90 +0,0 @@
-/*
- * linux/mm/vcache.c
- *
- * virtual => physical page mapping cache. Users of this mechanism
- * register callbacks for a given (virt,mm,phys) page mapping, and
- * the kernel guarantees to call back when this mapping is invalidated.
- * (ie. upon COW or unmap.)
- *
- * Started by Ingo Molnar, Copyright (C) 2002
- */
-
-#include <linux/mm.h>
-#include <linux/init.h>
-#include <linux/hash.h>
-#include <linux/vcache.h>
-
-#define VCACHE_HASHBITS 8
-#define VCACHE_HASHSIZE (1 << VCACHE_HASHBITS)
-
-#spinlock_t vcache_lock = SPIN_LOCK_UNLOCKED;
-
-#static struct list_head hash[VCACHE_HASHSIZE];
-
-#static struct list_head *hash_vcache(unsigned long address,
-# struct mm_struct *mm)
-#{
-# return &hash[hash_long(address + (unsigned long)mm, VCACHE_HASHBITS)];
-#}
-
-#void __attach_vcache(vcache_t *vcache,
-# unsigned long address,
-# struct mm_struct *mm,
-# void (*callback)(struct vcache_s *data, struct page *new))
-#{
-# struct list_head *hash_head;
-#
-# address &= PAGE_MASK;
-# vcache->address = address;
-# vcache->mm = mm;
-# vcache->callback = callback;
-#
-# hash_head = hash_vcache(address, mm);
-#
-# list_add_tail(&vcache->hash_entry, hash_head);
-#}
-

```

Linux-Kernel: [PATCH 1/2] Unpinned futexes v2 – part 1: indexing changes

```
-void __detach_vcache(vcache_t *vcache)
-{
- list_del_init(&vcache->hash_entry);
-}
-
-void invalidate_vcache(unsigned long address, struct mm_struct *mm,
- struct page *new_page)
-{
- struct list_head *l, *hash_head;
- vcache_t *vcache;
-
- address &= PAGE_MASK;
-
- hash_head = hash_vcache(address, mm);
- /*
- * This is safe, because this path is called with the pagetable
- * lock held. So while other mm's might add new entries in
- * parallel, *this* mm is locked out, so if the list is empty
- * now then we do not have to take the vcache lock to see it's
- * really empty.
- */
- if (likely(list_empty(hash_head)))
- return;
-
- spin_lock(&vcache_lock);
- list_for_each(l, hash_head) {
- vcache = list_entry(l, vcache_t, hash_entry);
- if (vcache->address != address || vcache->mm != mm)
- continue;
- vcache->callback(vcache, new_page);
- }
- spin_unlock(&vcache_lock);
-}
-
-static int __init vcache_init(void)
-{
- unsigned int i;
-
- for (i = 0; i < VCACHE_HASHSIZE; i++)
- INIT_LIST_HEAD(hash + i);
- return 0;
-}
-__initcall(vcache_init);
-
-
```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>