
Subject: Re: [IPC]: Logical refcount loop in ipc ns -> massive leakage
Posted by [ebiederm](#) on Mon, 05 Feb 2007 18:23:06 GMT
[View Forum Message](#) <> [Reply to Message](#)

Alexey Kuznetsov <kuznet@ms2.inr.ac.ru> writes:

> Yes, this would save the day.
>
> Indeed, shm_file_ns() is required only when the segment is already mapped,
> except for shm_mmap() and even there shm_nattch is incremented before
> do_mmap() is used. It will work.
>
> Possibility to use this file directly will be lost. It is a little unpleasant;
> openvz checkpointing used it to restore sysv shm mappings like another file
> mappings, it was nice, but this code can be a little uglified to treat
> those mapping specially. No harm either.
>
> Alexey

So I don't know if this will solve the checkpoint problem. However I find a slightly cleaner way to handle this that should be a little more maintainable. I'm still testing this patch so there might be a stupid bug but the general idea remains.

Subject: [PATCH] shm: Make sysv ipc shared memory use stacked files.

The current ipc shared memory code runs into several problems because it does not quite use files like the rest of the kernel. With the option of backing ipc shared memory with either hugetlbfs or ordinary shared memory the problems got worse. With the added support for ipc namespaces things behaved so unexpected that we now have several bad namespace reference counting bugs when using what appears at first glance to be a reasonable idiom.

So to attack these problems and hopefully make the code more maintainable this patch simply uses the files provided but other parts of the kernel and builds it's own files out of them. The shm files are allocated in do_shmat and freed with their last unmap. The file and vm operations that we don't want to implement or we don't implement completely we just delegate to the operations of our backing file.

This means that we now get an accurate shm_nattch count for we have a hugetlbfs inode for backing store, and the shm accounting of last attach and last detach time work as well.

This means that getting a reference to the ipc namespace when we create the file and dropping the reference in the release method

is now safe and correct.

This means we no longer need a special case for clearing VM_MAYWRITE as our file descriptor now only has write permissions when we have requested write access when calling shmat. Although VM_SHARED is now cleared as well which I believe is harmless and is mostly likely a minor bug fix.

By using the same set of operations for both the hugetlb case and regular shared memory case shmdt is not simplified and made slightly more correct as now the test "vma->vm_ops == &shm_vm_ops" is 100% accurate in spotting all shared memory regions generated from sysvipc shared memory.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>

```
ipc/shm.c | 204 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-----
1 files changed, 133 insertions(+), 71 deletions(-)
```

```
diff --git a/ipc/shm.c b/ipc/shm.c
index f8e10a2..e0b6544 100644
```

```
--- a/ipc/shm.c
```

```
+++ b/ipc/shm.c
```

```
@@ -37,11 +37,21 @@
```

```
#include <linux/seq_file.h>
```

```
#include <linux/mutex.h>
```

```
#include <linux/nsproxy.h>
```

```
+#include <linux/mount.h>
```

```
#include <asm/uaccess.h>
```

```
#include "util.h"
```

```
+struct shm_file_data {
```

```
+ int id;
```

```
+ struct ipc_namespace *ns;
```

```
+ struct file *file;
```

```
+ const struct vm_operations_struct *vm_ops;
```

```
+};
```

```
+
```

```
+#define shm_file_data(file) *((struct shm_file_data **)&(file)->private_data)
```

```
+
```

```
static struct file_operations shm_file_operations;
```

```
static struct vm_operations_struct shm_vm_ops;
```

```
@@ -60,8 +70,8 @@ static struct ipc_ids init_shm_ids;
```

```
static int newseg (struct ipc_namespace *ns, key_t key,
```

```

    int shmflg, size_t size);
-static void shm_open (struct vm_area_struct *shmd);
-static void shm_close (struct vm_area_struct *shmd);
+static void shm_open (struct vm_area_struct *vma);
+static void shm_close (struct vm_area_struct *vma);
    static void shm_destroy (struct ipc_namespace *ns, struct shmid_kernel *shp);
#ifdef CONFIG_PROC_FS
    static int sysvipc_shm_proc_show(struct seq_file *s, void *it);
@@ -150,11 +160,14 @@ static inline int shm_addid(struct ipc_namespace *ns, struct
shmid_kernel *shp)

```

```

-static inline void shm_inc(struct ipc_namespace *ns, int id)
+/* This is called by fork, once for every shm attach. */
+static void shm_open(struct vm_area_struct *vma)
{
+ struct file *file = vma->vm_file;
+ struct shm_file_data *sfd = shm_file_data(file);
+ struct shmid_kernel *shp;

- shp = shm_lock(ns, id);
+ shp = shm_lock(sfd->ns, sfd->id);
+ BUG_ON(!shp);
+ shp->shm_atim = get_seconds();
+ shp->shm_lprid = current->tgid;
@@ -162,15 +175,6 @@ static inline void shm_inc(struct ipc_namespace *ns, int id)
shm_unlock(shp);
}

```

```

-#define shm_file_ns(file) (*((struct ipc_namespace **)&(file)->private_data))
-
-/* This is called by fork, once for every shm attach. */
-static void shm_open(struct vm_area_struct *shmd)
-{
- shm_inc(shm_file_ns(shmd->vm_file),
- shmd->vm_file->f_path.dentry->d_inode->i_ino);
-}
-
/*
* shm_destroy - free the struct shmid_kernel
*
@@ -195,23 +199,21 @@ static void shm_destroy(struct ipc_namespace *ns, struct
shmid_kernel *shp)
}

/*
- * remove the attach descriptor shmd.

```

```
+ * remove the attach descriptor vma.  
+ * free memory for segment if it is marked destroyed.  
+ * The descriptor has already been removed from the current->mm->mmap list  
+ * and will later be kfree()d.  
+ */  
  
-static void shm_close (struct vm_area_struct *shmd)  
+static void shm_close (struct vm_area_struct *vma)  
{  
- struct file * file = shmd->vm_file;  
- int id = file->f_path.dentry->d_inode->i_ino;  
+ struct file * file = vma->vm_file;  
+ struct shm_file_data *sfd = shm_file_data(file);  
    struct shmid_kernel *shp;  
- struct ipc_namespace *ns;  
-  
- ns = shm_file_ns(file);  
+ struct ipc_namespace *ns = sfd->ns;  
  
    mutex_lock(&shm_ids(ns).mutex);  
    /* remove from the list of attaches of the shm segment */  
- shp = shm_lock(ns, id);  
+ shp = shm_lock(ns, sfd->id);  
    BUG_ON(!shp);  
    shp->shm_lprid = current->tgid;  
    shp->shm_dtim = get_seconds();  
@@ -224,46 +226,90 @@ static void shm_close (struct vm_area_struct *shmd)  
    mutex_unlock(&shm_ids(ns).mutex);  
}  
  
+struct page *shm_nopage(struct vm_area_struct *vma, unsigned long address, int *type)  
+{  
+ struct file *file = vma->vm_file;  
+ struct shm_file_data *sfd = shm_file_data(file);  
+  
+ return sfd->vm_ops->nopage(vma, address, type);  
+}  
+  
+  
+#ifdef CONFIG_NUMA  
+int shm_set_policy(struct vm_area_struct *vma, struct mempolicy *new)  
+{  
+ struct file *file = vma->vm_file;  
+ struct shm_file_data *sfd = shm_file_data(file);  
+ int err = 0;  
+ if (sfd->vm_ops->set_policy)  
+   err = sfd->vm_ops->set_policy(vma, new);  
+ return err;  
+}  
+
```

```

+struct mempolicy *shm_get_policy(struct vm_area_struct *vma, unsigned long addr)
+{
+ struct file *file = vma->vm_file;
+ struct shm_file_data *sfd = shm_file_data(file);
+ struct mempolicy *pol = NULL;
+
+ if (sfd->vm_ops->get_policy)
+ pol = sfd->vm_ops->get_policy(vma, addr);
+ else
+ pol = vma->vm_policy;
+ return pol;
+}
+
+static int shm_mmap(struct file * file, struct vm_area_struct * vma)
+{
+ struct shm_file_data *sfd = shm_file_data(file);
+ int ret;
+
+ ret = shmem_mmap(file, vma);
+ if (ret == 0) {
+ vma->vm_ops = &shm_vm_ops;
+ if (!(vma->vm_flags & VM_WRITE))
+ vma->vm_flags &= ~VM_MAYWRITE;
+ shm_inc(shm_file_ns(file), file->f_path.dentry->d_inode->i_ino);
+ }
+ ret = sfd->file->f_op->mmap(sfd->file, vma);
+ if (ret != 0)
+ return ret;
+ sfd->vm_ops = vma->vm_ops;
+ vma->vm_ops = &shm_vm_ops;
+ shm_open(vma);
+
+ return ret;
+}
+
+static int shm_release(struct inode *ino, struct file *file)
+{
+ struct ipc_namespace *ns;
+ struct shm_file_data *sfd = shm_file_data(file);
+
+ ns = shm_file_ns(file);
+ put_ipc_ns(ns);
+ shm_file_ns(file) = NULL;
+ put_ipc_ns(sfd->ns);
+ shm_file_data(file) = NULL;
+ kfree(sfd);
+ return 0;
+}

```

```

}

+#ifndef CONFIG_MMU
+static unsigned long shm_get_unmapped_area(struct file *file,
+ unsigned long addr, unsigned long len, unsigned long pgoff,
+ unsigned long flags)
+{
+ struct shm_file_data *sfd = shm_file_data(file);
+ return sfd->file->f_op->get_unmapped_area(sfd->file, addr, len, pgoff,
+     flags);
+}
+#else
+#define shm_get_unmapped_area NULL
+#endif
+
+static struct file_operations shm_file_operations = {
+ .mmap = shm_mmap,
+ .release = shm_release,
-#ifndef CONFIG_MMU
- .get_unmapped_area = shmem_get_unmapped_area,
-#endif
+ .get_unmapped_area = shm_get_unmapped_area,
+};

static struct vm_operations_struct shm_vm_ops = {
+ .open = shm_open, /* callback for a new vm-area open */
+ .close = shm_close, /* callback for when the vm-area is released */
- .npage = shmem_npage,
-#if defined(CONFIG_NUMA) && defined(CONFIG_SHMEM)
- .set_policy = shmem_set_policy,
- .get_policy = shmem_get_policy,
+ .npage = shm_npage,
+#if defined(CONFIG_NUMA)
+ .set_policy = shm_set_policy,
+ .get_policy = shm_get_policy,
+ #endif
+};

@@ -330,13 +376,6 @@ static int newseg (struct ipc_namespace *ns, key_t key, int shmflg,
size_t size)
+ shp->shm_nattch = 0;
+ shp->id = shm_buildid(ns, id, shp->shm_perm.seq);
+ shp->shm_file = file;
- file->f_path.dentry->d_inode->i_ino = shp->id;
-
- shm_file_ns(file) = get_ipc_ns(ns);
-
- /* Hugetlb ops would have already been assigned. */

```

```

- if (!(shmflg & SHM_HUGETLB))
- file->f_op = &shm_file_operations;

ns->shm_tot += numpages;
shm_unlock(shp);
@@ -607,10 +646,7 @@ asmlinkage long sys_shmctl (int shmid, int cmd, struct shmid_ds __user
*buf)
    tbuf.shm_ctime = shp->shm_ctim;
    tbuf.shm_cpid = shp->shm_cpid;
    tbuf.shm_lpid = shp->shm_lpid;
- if (!is_file_hugepages(shp->shm_file))
- tbuf.shm_nattch = shp->shm_nattch;
- else
- tbuf.shm_nattch = file_count(shp->shm_file) - 1;
+ tbuf.shm_nattch = shp->shm_nattch;
    shm_unlock(shp);
    if(copy_shmid_to_user (buf, &tbuf, version))
        err = -EFAULT;
@@ -781,6 +817,8 @@ long do_shmat(int shmid, char __user *shmaddr, int shmflg, ulong
*raddr)
    int acc_mode;
    void *user_addr;
    struct ipc_namespace *ns;
+ struct shm_file_data *sfd;
+ mode_t f_mode;

    if (shmid < 0) {
        err = -EINVAL;
@@ -806,9 +844,11 @@ long do_shmat(int shmid, char __user *shmaddr, int shmflg, ulong
*raddr)
    if (shmflg & SHM_RDONLY) {
        prot = PROT_READ;
        acc_mode = S_IRUGO;
+ f_mode = FMODE_READ;
    } else {
        prot = PROT_READ | PROT_WRITE;
        acc_mode = S_IRUGO | S_IWUGO;
+ f_mode = FMODE_READ | FMODE_WRITE;
    }
    if (shmflg & SHM_EXEC) {
        prot |= PROT_EXEC;
@@ -820,29 +860,43 @@ long do_shmat(int shmid, char __user *shmaddr, int shmflg, ulong
*raddr)
    * additional creator id...
    */
    ns = current->nsproxy->ipc_ns;
+ err = -EINVAL;
    shp = shm_lock(ns, shmid);

```

```

- if(shp == NULL) {
- err = -EINVAL;
+ if(shp == NULL)
    goto out;
- }
+
    err = shm_checkid(ns, shp, shmid);
- if (err) {
- shm_unlock(shp);
- goto out;
- }
- if (ipcperms(&shp->shm_perm, acc_mode)) {
- shm_unlock(shp);
- err = -EACCES;
- goto out;
- }
+ if (err)
+ goto out_unlock;
+
+ err = -EACCES;
+ if (ipcperms(&shp->shm_perm, acc_mode))
+ goto out_unlock;

    err = security_shm_shmat(shp, shmaddr, shmflg);
- if (err) {
- shm_unlock(shp);
- return err;
- }
-
- file = shp->shm_file;
+ if (err)
+ goto out_unlock;
+
+ err = -ENOMEM;
+ sfd = kzalloc(sizeof(*sfd), GFP_KERNEL);
+ if (!sfd)
+ goto out_unlock;
+
+ file = get_empty_filp();
+ if (!file)
+ goto out_free;
+
+ file->f_op = &shm_file_operations;
+ file->private_data = sfd;
+ file->f_path.dentry = dget(shp->shm_file->f_path.dentry);
+ file->f_path.mnt = mntget(shp->shm_file->f_path.mnt);
+ file->f_mapping = shp->shm_file->f_mapping;
+ file->f_mode = f_mode;

```



```

+ sfd->id = shp->id;
+ sfd->ns = get_ipc_ns(ns);
+ sfd->file = shp->shm_file;
+ sfd->vm_ops = NULL;
+
+ size = i_size_read(file->f_path.dentry->d_inode);
+ shp->shm_nattch++;
+ shm_unlock(shp);
@@ -866,6 +920,8 @@ long do_shmat(int shmid, char __user *shmaddr, int shmflg, ulong
*raddr)
invalid:
+ up_write(&current->mm->mmap_sem);

+ fput(file);
+
+ mutex_lock(&shm_ids(ns).mutex);
+ shp = shm_lock(ns, shmid);
+ BUG_ON(!shp);
@@ -883,6 +939,12 @@ invalid:
+ err = PTR_ERR(user_addr);
out:
+ return err;
+out_free:
+ kfree(sfd);
+out_unlock:
+ shm_unlock(shp);
+ goto out;
+
+ }

asmlinkage long sys_shmat(int shmid, char __user *shmaddr, int shmflg)
@@ -944,7 +1006,7 @@ asmlinkage long sys_shmdt(char __user *shmaddr)
+ * a fragment created by mprotect() and/or munmap(), or it
+ * otherwise it starts at this address with no hassles.
+ */
- if ((vma->vm_ops == &shm_vm_ops || is_vm_hugetlb_page(vma)) &&
+ if ((vma->vm_ops == &shm_vm_ops) &&
+ (vma->vm_start - addr)/PAGE_SIZE == vma->vm_pgoff) {

@@ -973,7 +1035,7 @@ asmlinkage long sys_shmdt(char __user *shmaddr)
+ next = vma->vm_next;

+ /* finding a matching vma now does not alter retval */
- if ((vma->vm_ops == &shm_vm_ops || is_vm_hugetlb_page(vma)) &&
+ if ((vma->vm_ops == &shm_vm_ops) &&
+ (vma->vm_start - addr)/PAGE_SIZE == vma->vm_pgoff)

```

```
do_munmap(mm, vma->vm_start, vma->vm_end - vma->vm_start);
@@ -1004,7 +1066,7 @@ static int sysvipc_shm_proc_show(struct seq_file *s, void *it)
    shp->shm_segsz,
    shp->shm_cprid,
    shp->shm_lprid,
-   is_file_hugepages(shp->shm_file) ? (file_count(shp->shm_file) - 1) : shp->shm_nattch,
+   shp->shm_nattch,
    shp->shm_perm.uid,
    shp->shm_perm.gid,
    shp->shm_perm.cuid,
--
1.4.4.1.g278f
```

Containers mailing list
Containers@lists.osdl.org
<https://lists.osdl.org/mailman/listinfo/containers>
