Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by Dave Hansen on Mon, 27 Feb 2006 17:42:12 GMT
View Forum Message <> Reply to Message

On Fri, 2006-02-24 at 14:44 -0700, Eric W. Biederman wrote:
> We can start on a broad front, looking at several different things.
> But I suggest the first thing we all look at is SYSVIPC. It is
> currently a clearly recognized namespace in the kernel so the scope is
> well defined. SYSVIPC is just complicated enough to have a
> non-trivial implementation while at the same time being simple enough
> that we can go through the code in exhausting detail. Getting the
> group dynamics working properly.

Here's a quick stab at the ipc/msg.c portion of this work. The basic approach was to move msg_ids, msg_bytes, and msg_hdrs into a structure, put a pointer to that structure in the task_struct and then dynamically allocate it.

There is still only one system-wide one of these for now. It can obviously be extended, though. :)

This is a very simple, brute-force, hack-until-it-compiles-and-boots approach. (I just realized that I didn't check the return of the alloc properly.)

Is this the form that we'd like these patches to take? Any comments about the naming? Do we want to keep the _namespace nomenclature, or does the "context" that I used here make more sense

-- Dave

```
 work-dave/include/linux/ipc.h   |   12 +++
 work-dave/include/linux/sched.h |    1
 work-dave/ipc/msg.c             |  152 ++++++++++++++++++++++++++++++----------
 work-dave/ipc/util.c            |    7 +
 work-dave/ipc/util.h            |    2
 work-dave/kernel/fork.c         |    5 +
 6 files changed, 108 insertions(+), 71 deletions(-)

--- work/ipc/msg.c~sysv-container 2006-02-27 09:30:23.000000000 -0800
+++ work-dave/ipc/msg.c 2006-02-27 09:32:18.000000000 -0800
@@ -60,35 +60,44 @@ struct msg_sender {
 #define SEARCH_NOTEQUAL  3
 #define SEARCH_LESSEQUAL 4

-static atomic_t msg_bytes = ATOMIC_INIT(0);
-static atomic_t msg_hdrs = ATOMIC_INIT(0);
+#define msg_lock(ctx, id) ((struct msg_queue*)ipc_lock(&ctx->msg_ids,id))
```

```
+#define msg_unlock(ctx, msq) ipc_unlock(&(msq)->q_perm)
+#define msg_rmid(ctx, id) ((struct msg_queue*)ipc_rmid(&ctx->msg_ids,id))
+#define msg_checkid(ctx, msq, msgid) \
+ ipc_checkid(&ctx->msg_ids,&msq->q_perm,msgid)
+#define msg_buildid(ctx, id, seq) \
+ ipc_buildid(&ctx->msg_ids, id, seq)

-static struct ipc_ids msg_ids;
-
-#define msg_lock(id) ((struct msg_queue*)ipc_lock(&msg_ids,id))
-#define msg_unlock(msq) ipc_unlock(&(msq)->q_perm)
-#define msg_rmid(id) ((struct msg_queue*)ipc_rmid(&msg_ids,id))
-#define msg_checkid(msq, msgid) \
- ipc_checkid(&msg_ids,&msq->q_perm,msgid)
-#define msg_buildid(id, seq) \
- ipc_buildid(&msg_ids, id, seq)
-
-static void freeque (struct msg_queue *msq, int id);
-static int newque (key_t key, int msgflg);
+static void freeque (struct ipc_msg_context *, struct msg_queue *msq, int id);
+static int newque (struct ipc_msg_context *context, key_t key, int id);
 #ifdef CONFIG_PROC_FS
 static int sysvipc_msg_proc_show(struct seq_file *s, void *it);
 #endif

-void __init msg_init (void)
+struct ipc_msg_context *alloc_ipc_msg_context(gfp_t flags)
+{
+ struct ipc_msg_context *msg_context;
+
+ msg_context = kzalloc(sizeof(*msg_context), flags);
+ if (!msg_context)
+  return NULL;
+
+ atomic_set(&msg_context->msg_bytes, 0);
+ atomic_set(&msg_context->msg_hdrs, 0);
+
+ return msg_context;
+}
+
+void __init msg_init (struct ipc_msg_context *context)
 {
- ipc_init_ids(&msg_ids,msg_ctlmni);
+ ipc_init_ids(&context->msg_ids,msg_ctlmni);
  ipc_init_proc_interface("sysvipc/msg",
     "       key      msqid perms      cbytes       qnum lspid lrpid   uid   gid  cuid  cgid      stime      rtime
     ctime\n",
-    &msg_ids,
```

```
+    &context->msg_ids,
     sysvipc_msg_proc_show);
 }

-static int newque (key_t key, int msgflg)
+static int newque (struct ipc_msg_context *context, key_t key, int msgflg)
 {
  int id;
  int retval;
@@ -108,14 +117,14 @@ static int newque (key_t key, int msgflg
   return retval;
  }

- id = ipc_addid(&msg_ids, &msq->q_perm, msg_ctlmni);
+ id = ipc_addid(&context->msg_ids, &msq->q_perm, msg_ctlmni);
  if(id == -1) {
   security_msg_queue_free(msq);
   ipc_rcu_putref(msq);
   return -ENOSPC;
  }

- msq->q_id = msg_buildid(id,msq->q_perm.seq);
+ msq->q_id = msg_buildid(context,id,msq->q_perm.seq);
  msq->q_stime = msq->q_rtime = 0;
  msq->q_ctime = get_seconds();
  msq->q_cbytes = msq->q_qnum = 0;
@@ -124,7 +133,7 @@ static int newque (key_t key, int msgflg
  INIT_LIST_HEAD(&msq->q_messages);
  INIT_LIST_HEAD(&msq->q_receivers);
  INIT_LIST_HEAD(&msq->q_senders);
- msg_unlock(msq);
+ msg_unlock(context, msq);

  return msq->q_id;
 }
@@ -182,23 +191,24 @@ static void expunge_all(struct msg_queue
  * msg_ids.sem and the spinlock for this message queue is hold
  * before freeque() is called. msg_ids.sem remains locked on exit.
  */
-static void freeque (struct msg_queue *msq, int id)
+static void freeque (struct ipc_msg_context *context,
+       struct msg_queue *msq, int id)
 {
  struct list_head *tmp;

  expunge_all(msq,-EIDRM);
  ss_wakeup(&msq->q_senders,1);
- msq = msg_rmid(id);
```

```
- msg_unlock(msq);
+ msq = msg_rmid(context, id);
+ msg_unlock(context, msq);

  tmp = msq->q_messages.next;
  while(tmp != &msq->q_messages) {
   struct msg_msg* msg = list_entry(tmp,struct msg_msg,m_list);
   tmp = tmp->next;
-  atomic_dec(&msg_hdrs);
+  atomic_dec(&context->msg_hdrs);
   free_msg(msg);
  }
- atomic_sub(msq->q_cbytes, &msg_bytes);
+ atomic_sub(msq->q_cbytes, &context->msg_bytes);
  security_msg_queue_free(msq);
  ipc_rcu_putref(msq);
 }
@@ -207,32 +217,34 @@ asmlinkage long sys_msgget (key_t key, i
 {
  int id, ret = -EPERM;
  struct msg_queue *msq;
-
- down(&msg_ids.sem);
+ struct ipc_msg_context *context = current->ipc_msg_context;
+
+ down(&context->msg_ids.sem);
  if (key == IPC_PRIVATE)
-  ret = newque(key, msgflg);
- else if ((id = ipc_findkey(&msg_ids, key)) == -1) { /* key not used */
+  ret = newque(context, key, msgflg);
+ else if ((id = ipc_findkey(&context->msg_ids, key)) == -1) {
+  /* key not used */
   if (!(msgflg & IPC_CREAT))
    ret = -ENOENT;
   else
-   ret = newque(key, msgflg);
+   ret = newque(context, key, msgflg);
  } else if (msgflg & IPC_CREAT && msgflg & IPC_EXCL) {
   ret = -EEXIST;
  } else {
-  msq = msg_lock(id);
+  msq = msg_lock(context, id);
   if(msq==NULL)
    BUG();
   if (ipcperms(&msq->q_perm, msgflg))
    ret = -EACCES;
   else {
-   int qid = msg_buildid(id, msq->q_perm.seq);
```

```
+   int qid = msg_buildid(context, id, msq->q_perm.seq);
        ret = security_msg_queue_associate(msq, msgflg);
    if (!ret)
     ret = qid;
   }
-  msg_unlock(msq);
+  msg_unlock(context, msq);
  }
- up(&msg_ids.sem);
+ up(&context->msg_ids.sem);
  return ret;
 }

@@ -333,6 +345,7 @@ asmlinkage long sys_msgctl (int msqid, i
  struct msg_queue *msq;
  struct msq_setbuf setbuf;
  struct kern_ipc_perm *ipcp;
+ struct ipc_msg_context *context = current->ipc_msg_context;

  if (msqid < 0 || cmd < 0)
   return -EINVAL;
@@ -362,18 +375,18 @@ asmlinkage long sys_msgctl (int msqid, i
   msginfo.msgmnb = msg_ctlmnb;
   msginfo.msgssz = MSGSSZ;
   msginfo.msgseg = MSGSEG;
-  down(&msg_ids.sem);
+  down(&context->msg_ids.sem);
   if (cmd == MSG_INFO) {
-   msginfo.msgpool = msg_ids.in_use;
-   msginfo.msgmap = atomic_read(&msg_hdrs);
-   msginfo.msgtql = atomic_read(&msg_bytes);
+   msginfo.msgpool = context->msg_ids.in_use;
+   msginfo.msgmap = atomic_read(&context->msg_hdrs);
+   msginfo.msgtql = atomic_read(&context->msg_bytes);
   } else {
    msginfo.msgmap = MSGMAP;
    msginfo.msgpool = MSGPOOL;
    msginfo.msgtql = MSGTQL;
   }
-  max_id = msg_ids.max_id;
-  up(&msg_ids.sem);
+  max_id = context->msg_ids.max_id;
+  up(&context->msg_ids.sem);
   if (copy_to_user (buf, &msginfo, sizeof(struct msginfo)))
    return -EFAULT;
   return (max_id < 0) ? 0: max_id;
@@ -385,20 +398,21 @@ asmlinkage long sys_msgctl (int msqid, i
   int success_return;
```

```
   if (!buf)
    return -EFAULT;
-  if(cmd == MSG_STAT && msqid >= msg_ids.entries->size)
+  if(cmd == MSG_STAT && msqid >= context->msg_ids.entries->size)
    return -EINVAL;

   memset(&tbuf,0,sizeof(tbuf));

-  msq = msg_lock(msqid);
+  msq = msg_lock(context, msqid);
   if (msq == NULL)
    return -EINVAL;

   if(cmd == MSG_STAT) {
-   success_return = msg_buildid(msqid, msq->q_perm.seq);
+   success_return =
+    msg_buildid(context, msqid, msq->q_perm.seq);
   } else {
    err = -EIDRM;
-   if (msg_checkid(msq,msqid))
+   if (msg_checkid(context,msq,msqid))
     goto out_unlock;
    success_return = 0;
   }
@@ -419,7 +433,7 @@ asmlinkage long sys_msgctl (int msqid, i
   tbuf.msg_qbytes = msq->q_qbytes;
   tbuf.msg_lspid  = msq->q_lspid;
   tbuf.msg_lrpid  = msq->q_lrpid;
-  msg_unlock(msq);
+  msg_unlock(context, msq);
   if (copy_msqid_to_user(buf, &tbuf, version))
    return -EFAULT;
   return success_return;
@@ -438,14 +452,14 @@ asmlinkage long sys_msgctl (int msqid, i
   return  -EINVAL;
  }

- down(&msg_ids.sem);
- msq = msg_lock(msqid);
+ down(&context->msg_ids.sem);
+ msq = msg_lock(context, msqid);
  err=-EINVAL;
  if (msq == NULL)
   goto out_up;

  err = -EIDRM;
- if (msg_checkid(msq,msqid))
+ if (msg_checkid(context,msq,msqid))
```

```
   goto out_unlock_up;
  ipcp = &msq->q_perm;
  err = -EPERM;
@@ -480,22 +494,22 @@ asmlinkage long sys_msgctl (int msqid, i
     * due to a larger queue size.
     */
   ss_wakeup(&msq->q_senders,0);
-  msg_unlock(msq);
+  msg_unlock(context, msq);
   break;
  }
  case IPC_RMID:
-  freeque (msq, msqid);
+  freeque (context, msq, msqid);
   break;
  }
  err = 0;
 out_up:
- up(&msg_ids.sem);
+ up(&context->msg_ids.sem);
  return err;
 out_unlock_up:
- msg_unlock(msq);
+ msg_unlock(context, msq);
  goto out_up;
 out_unlock:
- msg_unlock(msq);
+ msg_unlock(context, msq);
  return err;
 }

@@ -558,7 +572,8 @@ asmlinkage long sys_msgsnd (int msqid, s
  struct msg_msg *msg;
  long mtype;
  int err;
-
+ struct ipc_msg_context *context = current->ipc_msg_context;
+
  if (msgsz > msg_ctlmax || (long) msgsz < 0 || msqid < 0)
   return -EINVAL;
  if (get_user(mtype, &msgp->mtype))
@@ -573,13 +588,13 @@ asmlinkage long sys_msgsnd (int msqid, s
  msg->m_type = mtype;
  msg->m_ts = msgsz;

- msq = msg_lock(msqid);
+ msq = msg_lock(context, msqid);
  err=-EINVAL;
```

```
  if(msq==NULL)
   goto out_free;

  err= -EIDRM;
- if (msg_checkid(msq,msqid))
+ if (msg_checkid(context,msq,msqid))
   goto out_unlock_free;

  for (;;) {
@@ -605,7 +620,7 @@ asmlinkage long sys_msgsnd (int msqid, s
   }
   ss_add(msq, &s);
   ipc_rcu_getref(msq);
-  msg_unlock(msq);
+  msg_unlock(context, msq);
   schedule();

   ipc_lock_by_ptr(&msq->q_perm);
@@ -630,15 +645,15 @@ asmlinkage long sys_msgsnd (int msqid, s
   list_add_tail(&msg->m_list,&msq->q_messages);
   msq->q_cbytes += msgsz;
   msq->q_qnum++;
-  atomic_add(msgsz,&msg_bytes);
-  atomic_inc(&msg_hdrs);
+  atomic_add(msgsz,&context->msg_bytes);
+  atomic_inc(&context->msg_hdrs);
  }

  err = 0;
  msg = NULL;

 out_unlock_free:
- msg_unlock(msq);
+ msg_unlock(context, msq);
 out_free:
  if(msg!=NULL)
   free_msg(msg);
@@ -670,17 +685,18 @@ asmlinkage long sys_msgrcv (int msqid, s
  struct msg_queue *msq;
  struct msg_msg *msg;
  int mode;
+ struct ipc_msg_context *context = current->ipc_msg_context;

  if (msqid < 0 || (long) msgsz < 0)
   return -EINVAL;
  mode = convert_mode(&msgtyp,msgflg);

- msq = msg_lock(msqid);
```

```
+ msq = msg_lock(context, msqid);
  if(msq==NULL)
   return -EINVAL;

  msg = ERR_PTR(-EIDRM);
- if (msg_checkid(msq,msqid))
+ if (msg_checkid(context,msq,msqid))
   goto out_unlock;

  for (;;) {
@@ -720,10 +736,10 @@ asmlinkage long sys_msgrcv (int msqid, s
    msq->q_rtime = get_seconds();
    msq->q_lrpid = current->tgid;
    msq->q_cbytes -= msg->m_ts;
-   atomic_sub(msg->m_ts,&msg_bytes);
-   atomic_dec(&msg_hdrs);
+   atomic_sub(msg->m_ts,&context->msg_bytes);
+   atomic_dec(&context->msg_hdrs);
    ss_wakeup(&msq->q_senders,0);
-   msg_unlock(msq);
+   msg_unlock(context, msq);
    break;
   }
   /* No message waiting. Wait for a message */
@@ -741,7 +757,7 @@ asmlinkage long sys_msgrcv (int msqid, s
    msr_d.r_maxsize = msgsz;
   msr_d.r_msg = ERR_PTR(-EAGAIN);
   current->state = TASK_INTERRUPTIBLE;
-  msg_unlock(msq);
+  msg_unlock(context, msq);

   schedule();

@@ -794,7 +810,7 @@ asmlinkage long sys_msgrcv (int msqid, s
   if (signal_pending(current)) {
    msg = ERR_PTR(-ERESTARTNOHAND);
 out_unlock:
-   msg_unlock(msq);
+   msg_unlock(context, msq);
    break;
   }
  }
diff -puN ipc/msgutil.c~sysv-container ipc/msgutil.c
diff -puN ipc/sem.c~sysv-container ipc/sem.c
diff -puN ipc/shm.c~sysv-container ipc/shm.c
diff -puN ipc/util.c~sysv-container ipc/util.c
--- work/ipc/util.c~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/ipc/util.c 2006-02-27 09:31:43.000000000 -0800
```

```
@@ -45,11 +45,14 @@ struct ipc_proc_iface {
  * The various system5 IPC resources (semaphores, messages and shared
  * memory are initialised
  */
-
+
+struct ipc_msg_context *ipc_msg_context;
 static int __init ipc_init(void)
 {
+ ipc_msg_context = alloc_ipc_msg_context(GFP_KERNEL);
+
  sem_init();
- msg_init();
+ msg_init(ipc_msg_context);
  shm_init();
  return 0;
 }
diff -puN ipc/util.h~sysv-container ipc/util.h
--- work/ipc/util.h~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/ipc/util.h 2006-02-27 09:30:24.000000000 -0800
@@ -12,7 +12,7 @@
 #define SEQ_MULTIPLIER (IPCMNI)

 void sem_init (void);
-void msg_init (void);
+void msg_init (struct ipc_msg_context *context);
 void shm_init (void);

 struct seq_file;
diff -puN include/linux/sched.h~sysv-container include/linux/sched.h
--- work/include/linux/sched.h~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/include/linux/sched.h 2006-02-27 09:30:24.000000000 -0800
@@ -793,6 +793,7 @@ struct task_struct {
  int link_count, total_link_count;
 /* ipc stuff */
  struct sysv_sem sysvsem;
+ struct ipc_msg_context *ipc_msg_context;
 /* CPU-specific state of this task */
  struct thread_struct thread;
 /* filesystem information */
diff -puN include/linux/ipc.h~sysv-container include/linux/ipc.h
--- work/include/linux/ipc.h~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/include/linux/ipc.h 2006-02-27 09:30:24.000000000 -0800
@@ -2,6 +2,9 @@
 #define _LINUX_IPC_H

 #include <linux/types.h>
+#include <linux/kref.h>
```

```
+#include <asm/atomic.h>
+#include <asm/semaphore.h>

 #define IPC_PRIVATE ((__kernel_key_t) 0)

@@ -83,6 +86,15 @@ struct ipc_ids {
  struct ipc_id_ary* entries;
 };

+struct ipc_msg_context {
+ atomic_t msg_bytes;
+ atomic_t msg_hdrs;
+
+ struct ipc_ids msg_ids;
+ struct kref count;
+};
+
+extern struct ipc_msg_context *alloc_ipc_msg_context(gfp_t flags);
 #endif /* __KERNEL__ */

 #endif /* _LINUX_IPC_H */
--- work/kernel/fork.c~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/kernel/fork.c 2006-02-27 09:30:24.000000000 -0800
@@ -1184,6 +1184,11 @@ static task_t *copy_process(unsigned lon
  }
  attach_pid(p, PIDTYPE_TGID, p->tgid);
  attach_pid(p, PIDTYPE_PID, p->pid);
+ { // this extern will go away when we start to dynamically
+   // allocate these, nothing to see here
+  extern struct ipc_msg_context ipc_msg_context;
+  p->ipc_msg_context = current->ipc_msg_context;
+ }

  nr_threads++;
  total_forks++;
_
```