Subject: [PATCH v3 00/10] IPC: checkpoint/restore in userspace enhancements
Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:25:15 GMT
View Forum Message <> Reply to Message

v3:
1) Copy messages to user-space under spinlock was replaced by allocation of
dummy message before queue lock and then copy of desired message to the dummy
one instead of unlinking it from queue list.
I.e. the message queue copy logic was changed: messages can be retrived one by
one (instead of receiving of the whole list at once).

This patch set is aimed to provide additional functionality for all IPC objects,
which is required for migration of these objects by user-space
checkpoint/restore utils (CRIU).

The main problem here was impossibility to set up object id. This patch set
solves the problem in two steps:
1) Makes it possible to create new object (shared memory, semaphores set or
messages queue) with ID, equal to passed key.
2) Makes it possible to change existent object key.

Another problem was to peek messages from queues without deleting them.
This was achived by introducing of new MSG_COPY flag for sys_msgrcv() and
MSG_SET_COPY command for sys_msgctl() (which can be used to select message
number to copy).

The following series implements...

Stanislav Kinsbursky (10):
ipc: remove forced assignment of selected message
ipc: "use key as id" functionality for resource get system call introduced
ipc: segment key change helper introduced
ipc: add new SHM_SET command for sys_shmctl() call
ipc: add new MSG_SET command for sys_msgctl() call
ipc: add new SEM_SET command for sys_semctl() call
IPC: message queue receive cleanup
IPC: message queue copy feature introduced
ipc: add new MSG_SET_COPY command for sys_msgctl() call
test: IPC message queue migration test

arch/tile/kernel/compat.c | 29 +++++++
include/linux/compat.h | 2
include/linux/ipc.h | 1
include/linux/msg.h | 11 ++
include/linux/sem.h | 1

```
 include/linux/shm.h                 |    1
 ipc/compat.c                        |   45 +++++++---
 ipc/msg.c                           |  129 +++++++++++++++++++++++------
 ipc/msgutil.c                       |   38 +++++++++
 ipc/sem.c                           |   11 ++
 ipc/shm.c                           |   14 +++
 ipc/util.c                          |   59 ++++++++++++-
 ipc/util.h                          |    4 +
 security/selinux/hooks.c            |    3 +
 security/smack/smack_lsm.c          |    3 +
 tools/testing/selftests/ipc/msgque.c |  151 ++++++++++++++++++++++++++++++
 16 files changed, 450 insertions(+), 52 deletions(-)
 create mode 100644 tools/testing/selftests/ipc/msgque.c
```

Subject: [PATCH v3 01/10] ipc: remove forced assignment of selected message

Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:25:25 GMT

View Forum Message <> Reply to Message

This is a cleanup patch. The assignment is redundant.

```
---
 ipc/msg.c |    1 -
 1 files changed, 0 insertions(+), 1 deletions(-)

diff --git a/ipc/msg.c b/ipc/msg.c
index 7385de2..f3bfbb8 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -787,7 +787,6 @@ long do_msgrcv(int msqid, long *pmtype, void __user *mtext,
 		    !security_msg_queue_msgrcv(msq, walk_msg, current,
 					       msgtyp, mode)) {
 
-			msg = walk_msg;
 			if (mode == SEARCH_LESSEQUAL &&
 			    walk_msg->m_type != 1) {
 				msg = walk_msg;
```

Subject: [PATCH v3 02/10] ipc: "use key as id" functionality for resource get system call i

Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:25:30 GMT

View Forum Message <> Reply to Message

This patch introduces new IPC resource get request flag IPC_PRESET, which
should be interpreted as a request to try to allocate IPC slot with number,
starting from value resented by key. IOW, kernel will try
allocate new segment in specified slot. If slot is not emply, them -EEXIST

returned.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
Signed-off-by: Cyrill Gorcunov <gorcunov@openvz.org>
---
 include/linux/ipc.h | 1 +
 ipc/msg.c | 1 +
 ipc/sem.c | 1 +
 ipc/shm.c | 1 +
 ipc/util.c | 8 +++++++-
 ipc/util.h | 1 +
 6 files changed, 12 insertions(+), 1 deletions(-)

```
diff --git a/include/linux/ipc.h b/include/linux/ipc.h
index 30e8161..d7e5632 100644
--- a/include/linux/ipc.h
+++ b/include/linux/ipc.h
@@ -24,6 +24,7 @@ struct ipc_perm
 #define IPC_CREAT  00001000   /* create if key is nonexistent */
 #define IPC_EXCL   00002000   /* fail if key exists */
 #define IPC_NOWAIT 00004000   /* return error on wait */
+#define IPC_PRESET 00040000   /* use key as id */

 /* these fields are used by the DIPC package so the kernel as standard
    should avoid using them if possible */
diff --git a/ipc/msg.c b/ipc/msg.c
index f3bfbb8..70939a0 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -190,6 +190,7 @@ static int newque(struct ipc_namespace *ns, struct ipc_params *params)

  msq->q_perm.mode = msgflg & S_IRWXUGO;
  msq->q_perm.key = key;
+ msq->q_perm.id = (msgflg & IPC_PRESET) ? key : 0;

  msq->q_perm.security = NULL;
  retval = security_msg_queue_alloc(msq);
diff --git a/ipc/sem.c b/ipc/sem.c
index 5215a81..845c912 100644
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -306,6 +306,7 @@ static int newary(struct ipc_namespace *ns, struct ipc_params *params)

  sma->sem_perm.mode = (semflg & S_IRWXUGO);
  sma->sem_perm.key = key;
+ sma->sem_perm.id = (semflg & IPC_PRESET) ? key : 0;

  sma->sem_perm.security = NULL;
```

```
  retval = security_sem_alloc(sma);
diff --git a/ipc/shm.c b/ipc/shm.c
index 41c1285..d4c0a9e 100644
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -480,6 +480,7 @@ static int newseg(struct ipc_namespace *ns, struct ipc_params *params)

  shp->shm_perm.key = key;
  shp->shm_perm.mode = (shmflg & S_IRWXUGO);
+ shp->shm_perm.id = (shmflg & IPC_PRESET) ? key : 0;
  shp->mlock_user = NULL;

  shp->shm_perm.security = NULL;
diff --git a/ipc/util.c b/ipc/util.c
index 75261a3..ac9c6a9 100644
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -264,7 +264,8 @@ int ipc_addid(struct ipc_ids* ids, struct kern_ipc_perm* new, int size)
  rcu_read_lock();
  spin_lock(&new->lock);

- err = idr_get_new(&ids->ipcs_idr, new, &id);
+ err = idr_get_new_above(&ids->ipcs_idr, new,
+     ipcid_to_idx(new->id), &id);
  if (err) {
   spin_unlock(&new->lock);
   rcu_read_unlock();
@@ -277,6 +278,11 @@ int ipc_addid(struct ipc_ids* ids, struct kern_ipc_perm* new, int size)
  new->cuid = new->uid = euid;
  new->gid = new->cgid = egid;

+ if (new->id && ipcid_to_idx(new->id) == id) {
+  new->seq = ipcid_to_seq(new->id);
+  return id;
+ }
+
  new->seq = ids->seq++;
  if(ids->seq > ids->seq_max)
   ids->seq = 0;
diff --git a/ipc/util.h b/ipc/util.h
index 6f5c20b..5f04b02 100644
--- a/ipc/util.h
+++ b/ipc/util.h
@@ -92,6 +92,7 @@ void __init ipc_init_proc_interface(const char *path, const char *header,
 #define IPC_SHM_IDS 2

 #define ipcid_to_idx(id) ((id) % SEQ_MULTIPLIER)
+#define ipcid_to_seq(id) ((id) / SEQ_MULTIPLIER)
```

```
/* must be called with ids->rw_mutex acquired for writing */
int ipc_addid(struct ipc_ids *, struct kern_ipc_perm *, int);
```

Subject: [PATCH v3 03/10] ipc: segment key change helper introduced
Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:25:35 GMT
View Forum Message <> Reply to Message

This patch introduces existent segment key changing infrastructure.
New function ipc_update_key() can be used change segment key, cuid, cgid
values. It checks for that new key is not used (except IPC_PRIVATE) prior to
set it on existent.
To make this possible, added copying of this fields from user-space in
__get_compat_ipc_perm() and __get_compat_ipc64_perm() functions. Also segment
search by key and lock were splitted into different functions, because
ipc_update_key() doesn't need to lock the segment during check that new key is
not used.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
Signed-off-by: Cyrill Gorcunov <gorcunov@openvz.org>

```
 ipc/compat.c |    6 ++++++
 ipc/util.c   |   51 ++++++++++++++++++++++++++++++++++++++++++++++++---
 ipc/util.h   |    2 ++
 3 files changed, 56 insertions(+), 3 deletions(-)

diff --git a/ipc/compat.c b/ipc/compat.c
index a6df704..aa88515 100644
--- a/ipc/compat.c
+++ b/ipc/compat.c
@@ -144,6 +144,9 @@ static inline int __get_compat_ipc64_perm(struct ipc64_perm *p64,
  err  = __get_user(p64->uid, &up64->uid);
  err |= __get_user(p64->gid, &up64->gid);
  err |= __get_user(p64->mode, &up64->mode);
+ err |= __get_user(p64->cuid, &up64->cuid);
+ err |= __get_user(p64->cgid, &up64->cgid);
+ err |= __get_user(p64->key, &up64->key);
  return err;
 }

@@ -155,6 +158,9 @@ static inline int __get_compat_ipc_perm(struct ipc64_perm *p,
  err  = __get_user(p->uid, &up->uid);
  err |= __get_user(p->gid, &up->gid);
  err |= __get_user(p->mode, &up->mode);
+ err |= __get_user(p->cuid, &up->cuid);
+ err |= __get_user(p->cgid, &up->cgid);
+ err |= __get_user(p->key, &up->key);
```

```
  return err;
 }

diff --git a/ipc/util.c b/ipc/util.c
index ac9c6a9..d1e2f25 100644
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -173,7 +173,7 @@ void __init ipc_init_proc_interface(const char *path, const char *header,
  * @key: The key to find
  *
  * Requires ipc_ids.rw_mutex locked.
- * Returns the LOCKED pointer to the ipc structure if found or NULL
+ * Returns the UNLOCKED pointer to the ipc structure if found or NULL
  * if not.
  * If key is found ipc points to the owning ipc structure
  */
@@ -195,7 +195,6 @@ static struct kern_ipc_perm *ipc_findkey(struct ipc_ids *ids, key_t key)
    continue;
   }

-  ipc_lock_by_ptr(ipc);
   return ipc;
  }

@@ -203,6 +202,27 @@ static struct kern_ipc_perm *ipc_findkey(struct ipc_ids *ids, key_t key)
 }

 /**
+ * ipc_findkey_locked - find and lock a key in an ipc identifier set
+ * @ids: Identifier set
+ * @key: The key to find
+ *
+ * Requires ipc_ids.rw_mutex locked.
+ * Returns the LOCKED pointer to the ipc structure if found or NULL
+ * if not.
+ * If key is found ipc points to the owning ipc structure
+ */
+
+static struct kern_ipc_perm *ipc_findkey_locked(struct ipc_ids *ids, key_t key)
+{
+ struct kern_ipc_perm *ipc;
+
+ ipc = ipc_findkey(ids, key);
+ if (ipc)
+  ipc_lock_by_ptr(ipc);
+ return ipc;
+}
+
```

```
+/**
  * ipc_get_maxid  - get the last assigned id
  * @ids: IPC identifier set
  *
@@ -382,7 +402,7 @@ retry:
   * a new entry + read locks are not "upgradable"
   */
  down_write(&ids->rw_mutex);
- ipcp = ipc_findkey(ids, params->key);
+ ipcp = ipc_findkey_locked(ids, params->key);
  if (ipcp == NULL) {
   /* key not used */
   if (!(flg & IPC_CREAT))
@@ -749,6 +769,31 @@ int ipcget(struct ipc_namespace *ns, struct ipc_ids *ids,
 }

 /**
+ * ipc_update_key - update the key of an IPC.
+ * @in:  the permission given as input.
+ * @out: the permission of the ipc to set.
+ *
+ * Common routine called by sys_shmctl(), sys_semctl(). sys_msgctl().
+ */
+int ipc_update_key(struct ipc_ids *ids, struct ipc64_perm *in,
+       struct kern_ipc_perm *out)
+{
+
+ if (in->key && out->key != in->key) {
+  /*
+   * Check for existent segment with the same key.
+   * Note: ipc_ids.rw_mutex is taken for write already.
+   */
+  if (ipc_findkey(ids, in->key))
+   return -EEXIST;
+ }
+ out->cuid = in->cuid;
+ out->cgid = in->cgid;
+ out->key = in->key;
+ return 0;
+}
+
+/**
  * ipc_update_perm - update the permissions of an IPC.
  * @in:  the permission given as input.
  * @out: the permission of the ipc to set.
diff --git a/ipc/util.h b/ipc/util.h
index 5f04b02..2bc6a9a 100644
--- a/ipc/util.h
```

```
+++ b/ipc/util.h
@@ -126,6 +126,8 @@ struct kern_ipc_perm *ipc_lock(struct ipc_ids *, int);

 void kernel_to_ipc64_perm(struct kern_ipc_perm *in, struct ipc64_perm *out);
 void ipc64_perm_to_ipc_perm(struct ipc64_perm *in, struct ipc_perm *out);
+int ipc_update_key(struct ipc_ids *ids, struct ipc64_perm *in,
+     struct kern_ipc_perm *out);
 void ipc_update_perm(struct ipc64_perm *in, struct kern_ipc_perm *out);
 struct kern_ipc_perm *ipcctl_pre_down(struct ipc_namespace *ns,
          struct ipc_ids *ids, int id, int cmd,
```

Subject: [PATCH v3 04/10] ipc: add new SHM_SET command for sys_shmctl() call

Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:25:41 GMT

View Forum Message <> Reply to Message

New SHM_SET command will be interpreted exactly as IPC_SET, but also will update key, cuid and cgid values. IOW, it allows to change existent key value. The fact, that key is not used is checked before update. Otherwise -EEXIST is returned.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
Signed-off-by: Cyrill Gorcunov <gorcunov@openvz.org>

```
---
 include/linux/shm.h       |    1 +
 ipc/compat.c              |    1 +
 ipc/shm.c                 |   13 ++++++++++--
 security/selinux/hooks.c  |    1 +
 security/smack/smack_lsm.c |    1 +
 5 files changed, 15 insertions(+), 2 deletions(-)

diff --git a/include/linux/shm.h b/include/linux/shm.h
index 92808b8..6fc5447 100644
--- a/include/linux/shm.h
+++ b/include/linux/shm.h
@@ -63,6 +63,7 @@ struct shmid_ds {
 /* ipcs ctl commands */
 #define SHM_STAT  13
 #define SHM_INFO  14
+#define SHM_SET  15

 /* Obsolete, used only for backwards compatibility */
 struct shminfo {
diff --git a/ipc/compat.c b/ipc/compat.c
index aa88515..e7d4af7 100644
--- a/ipc/compat.c
+++ b/ipc/compat.c
@@ -688,6 +688,7 @@ long compat_sys_shmctl(int first, int second, void __user *uptr)
```

```
  case IPC_SET:
+ case SHM_SET:
   if (version == IPC_64) {
    err = get_compat_shmid64_ds(&s64, uptr);
   } else {
diff --git a/ipc/shm.c b/ipc/shm.c
index d4c0a9e..677b9f6 100644
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -635,6 +635,9 @@ copy_shmid_from_user(struct shmid64_ds *out, void __user *buf, int
version)
   out->shm_perm.uid = tbuf_old.shm_perm.uid;
   out->shm_perm.gid = tbuf_old.shm_perm.gid;
   out->shm_perm.mode = tbuf_old.shm_perm.mode;
+  out->shm_perm.cuid = tbuf_old.shm_perm.cuid;
+  out->shm_perm.cgid = tbuf_old.shm_perm.cgid;
+  out->shm_perm.key = tbuf_old.shm_perm.key;

   return 0;
      }
@@ -739,12 +742,13 @@ static int shmctl_down(struct ipc_namespace *ns, int shmid, int cmd,
  struct shmid_kernel *shp;
  int err;

- if (cmd == IPC_SET) {
+ if (cmd == IPC_SET || cmd == SHM_SET) {
   if (copy_shmid_from_user(&shmid64, buf, version))
    return -EFAULT;
  }

- ipcp = ipcctl_pre_down(ns, &shm_ids(ns), shmid, cmd,
+ ipcp = ipcctl_pre_down(ns, &shm_ids(ns), shmid,
+          (cmd != SHM_SET) ? : IPC_SET,
           &shmid64.shm_perm, 0);
  if (IS_ERR(ipcp))
   return PTR_ERR(ipcp);
@@ -758,6 +762,10 @@ static int shmctl_down(struct ipc_namespace *ns, int shmid, int cmd,
  case IPC_RMID:
   do_shm_rmid(ns, ipcp);
   goto out_up;
+ case SHM_SET:
+  err = ipc_update_key(&shm_ids(ns), &shmid64.shm_perm, ipcp);
+  if (err)
+   break;
  case IPC_SET:
   ipc_update_perm(&shmid64.shm_perm, ipcp);
```

```
   shp->shm_ctim = get_seconds();
@@ -935,6 +943,7 @@ SYSCALL_DEFINE3(shmctl, int, shmid, int, cmd, struct shmid_ds __user
*, buf)
  }
  case IPC_RMID:
  case IPC_SET:
+ case SHM_SET:
   err = shmctl_down(ns, shmid, cmd, buf, version);
   return err;
  default:
diff --git a/security/selinux/hooks.c b/security/selinux/hooks.c
index 6c56ebf..9e07e22 100644
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
@@ -5054,6 +5054,7 @@ static int selinux_shm_shmctl(struct shmid_kernel *shp, int cmd)
   perms = SHM__GETATTR | SHM__ASSOCIATE;
   break;
  case IPC_SET:
+ case SHM_SET:
   perms = SHM__SETATTR;
   break;
  case SHM_LOCK:
diff --git a/security/smack/smack_lsm.c b/security/smack/smack_lsm.c
index ee0bb57..a25dfd1 100644
--- a/security/smack/smack_lsm.c
+++ b/security/smack/smack_lsm.c
@@ -2145,6 +2145,7 @@ static int smack_shm_shmctl(struct shmid_kernel *shp, int cmd)
   may = MAY_READ;
   break;
  case IPC_SET:
+ case SHM_SET:
  case SHM_LOCK:
  case SHM_UNLOCK:
  case IPC_RMID:
```

Subject: [PATCH v3 05/10] ipc: add new MSG_SET command for sys_msgctl() call
Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:25:46 GMT
View Forum Message <> Reply to Message

New MSG_SET command will be interpreted exactly as IPC_SET, but also will
update key, cuid and cgid values. IOW, it allows to change existent key value.
The fact, that key is not used is checked before update. Otherwise -EEXIST is
returned.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
Signed-off-by: Cyrill Gorcunov <gorcunov@openvz.org>

```
 include/linux/msg.h        |    1 +
 ipc/compat.c               |    1 +
 ipc/msg.c                  |   13 ++++++++++--
 security/selinux/hooks.c   |    1 +
 security/smack/smack_lsm.c |    1 +
 5 files changed, 15 insertions(+), 2 deletions(-)

diff --git a/include/linux/msg.h b/include/linux/msg.h
index 56abf15..6689e73 100644
--- a/include/linux/msg.h
+++ b/include/linux/msg.h
@@ -6,6 +6,7 @@
 /* ipcs ctl commands */
 #define MSG_STAT 11
 #define MSG_INFO 12
+#define MSG_SET  13

 /* msgrcv options */
 #define MSG_NOERROR     010000  /* no error if message is too big */
diff --git a/ipc/compat.c b/ipc/compat.c
index e7d4af7..da4f5c6 100644
--- a/ipc/compat.c
+++ b/ipc/compat.c
@@ -483,6 +483,7 @@ long compat_sys_msgctl(int first, int second, void __user *uptr)
   break;

  case IPC_SET:
+ case MSG_SET:
   if (version == IPC_64) {
    err = get_compat_msqid64(&m64, uptr);
   } else {
diff --git a/ipc/msg.c b/ipc/msg.c
index 70939a0..6bd66d3 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -391,6 +391,9 @@ copy_msqid_from_user(struct msqid64_ds *out, void __user *buf, int
version)
   out->msg_perm.uid      = tbuf_old.msg_perm.uid;
   out->msg_perm.gid      = tbuf_old.msg_perm.gid;
   out->msg_perm.mode     = tbuf_old.msg_perm.mode;
+  out->msg_perm.cuid = tbuf_old.msg_perm.cuid;
+  out->msg_perm.cgid = tbuf_old.msg_perm.cgid;
+  out->msg_perm.key = tbuf_old.msg_perm.key;

   if (tbuf_old.msg_qbytes == 0)
    out->msg_qbytes = tbuf_old.msg_lqbytes;
@@ -417,12 +420,13 @@ static int msgctl_down(struct ipc_namespace *ns, int msqid, int cmd,
  struct msg_queue *msq;
```

```
  int err;

- if (cmd == IPC_SET) {
+ if (cmd == IPC_SET || cmd == MSG_SET) {
   if (copy_msqid_from_user(&msqid64, buf, version))
    return -EFAULT;
  }

- ipcp = ipcctl_pre_down(ns, &msg_ids(ns), msqid, cmd,
+ ipcp = ipcctl_pre_down(ns, &msg_ids(ns), msqid,
+          (cmd != MSG_SET) ? cmd : IPC_SET,
           &msqid64.msg_perm, msqid64.msg_qbytes);
  if (IS_ERR(ipcp))
   return PTR_ERR(ipcp);
@@ -438,6 +442,7 @@ static int msgctl_down(struct ipc_namespace *ns, int msqid, int cmd,
   freeque(ns, ipcp);
   goto out_up;
  case IPC_SET:
+ case MSG_SET:
   if (msqid64.msg_qbytes > ns->msg_ctlmnb &&
       !capable(CAP_SYS_RESOURCE)) {
    err = -EPERM;
@@ -446,6 +451,9 @@ static int msgctl_down(struct ipc_namespace *ns, int msqid, int cmd,

   msq->q_qbytes = msqid64.msg_qbytes;

+  if (cmd == MSG_SET)
+   ipc_update_key(&msg_ids(ns), &msqid64.msg_perm, ipcp);
+
   ipc_update_perm(&msqid64.msg_perm, ipcp);
   msq->q_ctime = get_seconds();
   /* sleeping receivers might be excluded by
@@ -565,6 +573,7 @@ SYSCALL_DEFINE3(msgctl, int, msqid, int, cmd, struct msqid_ds __user
*, buf)
  }
  case IPC_SET:
  case IPC_RMID:
+ case MSG_SET:
   err = msgctl_down(ns, msqid, cmd, buf, version);
   return err;
  default:
diff --git a/security/selinux/hooks.c b/security/selinux/hooks.c
index 9e07e22..622e020 100644
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
@@ -4912,6 +4912,7 @@ static int selinux_msg_queue_msgctl(struct msg_queue *msq, int cmd)
   perms = MSGQ__GETATTR | MSGQ__ASSOCIATE;
   break;
```

```
  case IPC_SET:
+ case MSG_SET:
    perms = MSGQ__SETATTR;
    break;
  case IPC_RMID:
diff --git a/security/smack/smack_lsm.c b/security/smack/smack_lsm.c
index a25dfd1..62826b1 100644
--- a/security/smack/smack_lsm.c
+++ b/security/smack/smack_lsm.c
@@ -2398,6 +2398,7 @@ static int smack_msg_queue_msgctl(struct msg_queue *msq, int cmd)
    may = MAY_READ;
    break;
  case IPC_SET:
+ case MSG_SET:
  case IPC_RMID:
    may = MAY_READWRITE;
    break;
```

Subject: [PATCH v3 06/10] ipc: add new SEM_SET command for sys_semctl() call
Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:25:51 GMT
View Forum Message <> Reply to Message

New SEM_SET command will be interpreted exactly as IPC_SET, but also will
update key, cuid and cgid values. IOW, it allows to change existent key value.
The fact, that key is not used is checked before update. Otherwise -EEXIST is
returned.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
Signed-off-by: Cyrill Gorcunov <gorcunov@openvz.org>

```
---
 include/linux/sem.h        |    1 +
 ipc/compat.c               |    1 +
 ipc/sem.c                  |   10 ++++++++--
 security/selinux/hooks.c   |    1 +
 security/smack/smack_lsm.c |    1 +
 5 files changed, 12 insertions(+), 2 deletions(-)

diff --git a/include/linux/sem.h b/include/linux/sem.h
index 10d6b22..c74b9b5 100644
--- a/include/linux/sem.h
+++ b/include/linux/sem.h
@@ -18,6 +18,7 @@
 /* ipcs ctl cmds */
 #define SEM_STAT 18
 #define SEM_INFO 19
+#define SEM_SET  20
```

```
 /* Obsolete, used only for backwards compatibility and libc5 compiles */
 struct semid_ds {
diff --git a/ipc/compat.c b/ipc/compat.c
index da4f5c6..7cfdb5b 100644
--- a/ipc/compat.c
+++ b/ipc/compat.c
@@ -290,6 +290,7 @@ static long do_compat_semctl(int first, int second, int third, u32 pad)
   break;

  case IPC_SET:
+ case SEM_SET:
   if (version == IPC_64) {
    err = get_compat_semid64_ds(&s64, compat_ptr(pad));
   } else {
diff --git a/ipc/sem.c b/ipc/sem.c
index 845c912..d9024d5 100644
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -1084,12 +1084,13 @@ static int semctl_down(struct ipc_namespace *ns, int semid,
  struct semid64_ds semid64;
  struct kern_ipc_perm *ipcp;

- if(cmd == IPC_SET) {
+ if (cmd == IPC_SET || cmd == SEM_SET) {
   if (copy_semid_from_user(&semid64, arg.buf, version))
    return -EFAULT;
  }

- ipcp = ipcctl_pre_down(ns, &sem_ids(ns), semid, cmd,
+ ipcp = ipcctl_pre_down(ns, &sem_ids(ns), semid,
+          (cmd != SEM_SET) ? : IPC_SET,
          &semid64.sem_perm, 0);
  if (IS_ERR(ipcp))
   return PTR_ERR(ipcp);
@@ -1104,6 +1105,10 @@ static int semctl_down(struct ipc_namespace *ns, int semid,
  case IPC_RMID:
   freeary(ns, ipcp);
   goto out_up;
+ case SEM_SET:
+  err = ipc_update_key(&sem_ids(ns), &semid64.sem_perm, ipcp);
+  if (err)
+   break;
  case IPC_SET:
   ipc_update_perm(&semid64.sem_perm, ipcp);
   sma->sem_ctime = get_seconds();
@@ -1149,6 +1154,7 @@ SYSCALL_DEFINE(semctl)(int semid, int semnum, int cmd, union
semun arg)
   return err;
```

```
  case IPC_RMID:
  case IPC_SET:
+ case SEM_SET:
   err = semctl_down(ns, semid, cmd, version, arg);
   return err;
  default:
diff --git a/security/selinux/hooks.c b/security/selinux/hooks.c
index 622e020..704391b 100644
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
@@ -5160,6 +5160,7 @@ static int selinux_sem_semctl(struct sem_array *sma, int cmd)
   perms = SEM__DESTROY;
   break;
  case IPC_SET:
+ case SEM_SET:
   perms = SEM__SETATTR;
   break;
  case IPC_STAT:
diff --git a/security/smack/smack_lsm.c b/security/smack/smack_lsm.c
index 62826b1..1116076 100644
--- a/security/smack/smack_lsm.c
+++ b/security/smack/smack_lsm.c
@@ -2277,6 +2277,7 @@ static int smack_sem_semctl(struct sem_array *sma, int cmd)
  case SETALL:
  case IPC_RMID:
  case IPC_SET:
+ case SEM_SET:
   may = MAY_READWRITE;
   break;
  case IPC_INFO:
```

Subject: [PATCH v3 07/10] IPC: message queue receive cleanup
Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:25:56 GMT
View Forum Message <> Reply to Message

This patch moves all message related manipulation into one function msg_fill().
Actually, two functions because of the compat one.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
Signed-off-by: Cyrill Gorcunov <gorcunov@openvz.org>

Conflicts:

```
 arch/tile/kernel/compat.c
 include/linux/compat.h
---
 arch/tile/kernel/compat.c |   29 +++++++++++++++++++++++++++++
```

```
 include/linux/compat.h    |    2 ++
 include/linux/msg.h       |    5 +++--
 ipc/compat.c              |   33 +++++++++++++++++++++----------
 ipc/msg.c                 |   44 ++++++++++++++++++++++++++++++++++++++++++++
 5 files changed, 74 insertions(+), 39 deletions(-)

diff --git a/arch/tile/kernel/compat.c b/arch/tile/kernel/compat.c
index d67459b..e74d61b 100644
--- a/arch/tile/kernel/compat.c
+++ b/arch/tile/kernel/compat.c
@@ -94,6 +94,35 @@ long compat_sys_sched_rr_get_interval(compat_pid_t pid,
  return ret;
 }

+/*
+ * The usual compat_sys_msgsnd() and _msgrcv() seem to be assuming
+ * some different calling convention than our normal 32-bit tile code.
+ */
+
+/* Already defined in ipc/compat.c, but we need it here. */
+struct compat_msgbuf {
+ compat_long_t mtype;
+ char mtext[1];
+};
+
+long tile_compat_sys_msgsnd(int msqid,
+       struct compat_msgbuf __user *msgp,
+       size_t msgsz, int msgflg)
+{
+ compat_long_t mtype;
+
+ if (get_user(mtype, &msgp->mtype))
+  return -EFAULT;
+ return do_msgsnd(msqid, mtype, msgp->mtext, msgsz, msgflg);
+}
+
+long tile_compat_sys_msgrcv(int msqid,
+       struct compat_msgbuf __user *msgp,
+       size_t msgsz, long msgtyp, int msgflg)
+{
+ return do_msgrcv(msqid, msgp, msgsz, msgtyp, msgflg, compat_do_msg_fill);
+}
+
 /* Provide the compat syscall number to call mapping. */
 #undef __SYSCALL
 #define __SYSCALL(nr, call) [nr] = (call),
diff --git a/include/linux/compat.h b/include/linux/compat.h
index 4e89039..6e035dc 100644
```

```
--- a/include/linux/compat.h
+++ b/include/linux/compat.h
@@ -245,6 +245,7 @@ struct compat_sysctl_args;
 struct compat_kexec_segment;
 struct compat_mq_attr;
 struct compat_msgbuf;
+struct msg_msg;

 extern void compat_exit_robust_list(struct task_struct *curr);

@@ -258,6 +259,7 @@ compat_sys_get_robust_list(int pid, compat_uptr_t __user *head_ptr,
 #ifdef CONFIG_ARCH_WANT_OLD_COMPAT_IPC
 long compat_sys_semctl(int first, int second, int third, void __user *uptr);
 long compat_sys_msgsnd(int first, int second, int third, void __user *uptr);
+long compat_do_msg_fill(void __user *dest, struct msg_msg *msg, size_t bufsz);
 long compat_sys_msgrcv(int first, int second, int msgtyp, int third,
   int version, void __user *uptr);
 long compat_sys_shmat(int first, int second, compat_uptr_t third, int version,
diff --git a/include/linux/msg.h b/include/linux/msg.h
index 6689e73..9411b76 100644
--- a/include/linux/msg.h
+++ b/include/linux/msg.h
@@ -105,8 +105,9 @@ struct msg_queue {
 /* Helper routines for sys_msgsnd and sys_msgrcv */
 extern long do_msgsnd(int msqid, long mtype, void __user *mtext,
    size_t msgsz, int msgflg);
-extern long do_msgrcv(int msqid, long *pmtype, void __user *mtext,
-   size_t msgsz, long msgtyp, int msgflg);
+extern long do_msgrcv(int msqid, void __user *buf, size_t bufsz, long msgtyp,
+       int msgflg,
+       long (*msg_fill)(void __user *, struct msg_msg *, size_t ));

 #endif /* __KERNEL__ */

diff --git a/ipc/compat.c b/ipc/compat.c
index 7cfdb5b..6b07f5c 100644
--- a/ipc/compat.c
+++ b/ipc/compat.c
@@ -341,13 +341,23 @@ long compat_sys_msgsnd(int first, int second, int third, void __user
*uptr)
  return do_msgsnd(first, type, up->mtext, second, third);
 }

+long compat_do_msg_fill(void __user *dest, struct msg_msg *msg, size_t bufsz)
+{
+ struct compat_msgbuf __user *msgp;
+ size_t msgsz;
+
```

```
+ if (put_user(msg->m_type, &msgp->mtype))
+  return -EFAULT;
+
+ msgsz = (bufsz > msg->m_ts) ? msg->m_ts : bufsz;
+ if (store_msg(msgp->mtext, msg, msgsz))
+  return -EFAULT;
+ return msgsz;
+}
+
 long compat_sys_msgrcv(int first, int second, int msgtyp, int third,
       int version, void __user *uptr)
 {
- struct compat_msgbuf __user *up;
- long type;
- int err;
-
  if (first < 0)
   return -EINVAL;
  if (second < 0)
@@ -355,23 +365,14 @@ long compat_sys_msgrcv(int first, int second, int msgtyp, int third,

  if (!version) {
   struct compat_ipc_kludge ipck;
-  err = -EINVAL;
   if (!uptr)
-   goto out;
-  err = -EFAULT;
+   return -EINVAL;
   if (copy_from_user (&ipck, uptr, sizeof(ipck)))
-   goto out;
+   return -EFAULT;
   uptr = compat_ptr(ipck.msgp);
   msgtyp = ipck.msgtyp;
  }
- up = uptr;
- err = do_msgrcv(first, &type, up->mtext, second, msgtyp, third);
- if (err < 0)
-  goto out;
- if (put_user(type, &up->mtype))
-  err = -EFAULT;
-out:
- return err;
+ return do_msgrcv(first, uptr, second, msgtyp, third, compat_do_msg_fill);
 }
 #else
 long compat_sys_semctl(int semid, int semnum, int cmd, int arg)
diff --git a/ipc/msg.c b/ipc/msg.c
index 6bd66d3..08009f5 100644
```

```
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -762,15 +762,30 @@ static inline int convert_mode(long *msgtyp, int msgflg)
  return SEARCH_EQUAL;
 }

-long do_msgrcv(int msqid, long *pmtype, void __user *mtext,
-  size_t msgsz, long msgtyp, int msgflg)
+static long do_msg_fill(void __user *dest, struct msg_msg *msg, size_t bufsz)
+{
+ struct msgbuf __user *msgp = dest;
+ size_t msgsz;
+
+ if (put_user(msg->m_type, &msgp->mtype))
+  return -EFAULT;
+
+ msgsz = (bufsz > msg->m_ts) ? msg->m_ts : bufsz;
+ if (store_msg(msgp->mtext, msg, msgsz))
+  return -EFAULT;
+ return msgsz;
+}
+
+long do_msgrcv(int msqid, void __user *buf, size_t bufsz, long msgtyp,
+       int msgflg,
+       long (*msg_handler)(void __user *, struct msg_msg *, size_t ))
 {
  struct msg_queue *msq;
  struct msg_msg *msg;
  int mode;
  struct ipc_namespace *ns;

- if (msqid < 0 || (long) msgsz < 0)
+ if (msqid < 0 || (long) bufsz < 0)
   return -EINVAL;
  mode = convert_mode(&msgtyp, msgflg);
  ns = current->nsproxy->ipc_ns;
@@ -813,7 +828,7 @@ long do_msgrcv(int msqid, long *pmtype, void __user *mtext,
     * Found a suitable message.
     * Unlink it from the queue.
     */
-   if ((msgsz < msg->m_ts) && !(msgflg & MSG_NOERROR)) {
+   if ((bufsz < msg->m_ts) && !(msgflg & MSG_NOERROR)) {
     msg = ERR_PTR(-E2BIG);
     goto out_unlock;
    }
@@ -840,7 +855,7 @@ long do_msgrcv(int msqid, long *pmtype, void __user *mtext,
   if (msgflg & MSG_NOERROR)
    msr_d.r_maxsize = INT_MAX;
```

```
   else
-   msr_d.r_maxsize = msgsz;
+   msr_d.r_maxsize = bufsz;
   msr_d.r_msg = ERR_PTR(-EAGAIN);
   current->state = TASK_INTERRUPTIBLE;
   msg_unlock(msq);
@@ -903,29 +918,16 @@ out_unlock:
  if (IS_ERR(msg))
   return PTR_ERR(msg);

- msgsz = (msgsz > msg->m_ts) ? msg->m_ts : msgsz;
- *pmtype = msg->m_type;
- if (store_msg(mtext, msg, msgsz))
-  msgsz = -EFAULT;
-
+ bufsz = msg_handler(buf, msg, bufsz);
  free_msg(msg);

- return msgsz;
+ return bufsz;
 }

 SYSCALL_DEFINE5(msgrcv, int, msqid, struct msgbuf __user *, msgp, size_t, msgsz,
   long, msgtyp, int, msgflg)
 {
- long err, mtype;
-
- err =  do_msgrcv(msqid, &mtype, msgp->mtext, msgsz, msgtyp, msgflg);
- if (err < 0)
-  goto out;
-
- if (put_user(mtype, &msgp->mtype))
-  err = -EFAULT;
-out:
- return err;
+ return do_msgrcv(msqid, msgp, msgsz, msgtyp, msgflg, do_msg_fill);
 }

 #ifdef CONFIG_PROC_FS
```

Subject: [PATCH v3 08/10] IPC: message queue copy feature introduced
Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:26:02 GMT
View Forum Message <> Reply to Message

This patch is required for checkpoint/restore in userspace.
IOW, c/r requires some way to get all pending IPC messages without deleting
them from the queue (checkpoint can fail and in this case tasks will be resumed,

so queue have to be valid).
To achive this, new operation flag MSG_COPY for sys_msgrcv() system call was
introduced. Also, copy counter was added to msg_queue structure. It's set to
zero by default and increases by one on each copy operation and decreased by
one on each receive operation until reaches zero.
If MSG_COPY is set, then kernel will allocate dummy message with passed size,
and then use new copy_msg() helper function to copy desired message (instead of
unlinking it from the queue).

Notes:
1) syscall type-based logic remains the same. It means, that passed type is
not zero and MSG_COPY is specified, then n-th message of that type will be
copied.
2) Return -ENOSYS if MSG_COPY is specified, but CONFIG_CHECKPOINT_RESTORE is
not set.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
---
 include/linux/msg.h | 4 ++++
 ipc/msg.c | 52 ++--
 ipc/msgutil.c | 38 ++++++++++++++++++++++++++++++++++++++
 ipc/util.h | 1 +
 4 files changed, 93 insertions(+), 2 deletions(-)

diff --git a/include/linux/msg.h b/include/linux/msg.h
index 9411b76..07a5e1e 100644
--- a/include/linux/msg.h
+++ b/include/linux/msg.h
@@ -11,6 +11,7 @@
 /* msgrcv options */
 #define MSG_NOERROR   010000 /* no error if message is too big */
 #define MSG_EXCEPT   020000 /* recv any msg except of specified type.*/
+#define MSG_COPY   040000 /* copy (not remove) all queue messages */

 /* Obsolete, used only for backwards compatibility and libc5 compiles */
 struct msqid_ds {
@@ -96,6 +97,9 @@ struct msg_queue {
  unsigned long q_qbytes; /* max number of bytes on queue */
  pid_t q_lspid;   /* pid of last msgsnd */
  pid_t q_lrpid;   /* last receive pid */
+#ifdef CONFIG_CHECKPOINT_RESTORE
+ unsigned int q_copy_cnt; /* message number for copy operations */
+#endif

  struct list_head q_messages;
  struct list_head q_receivers;
diff --git a/ipc/msg.c b/ipc/msg.c
index 08009f5..44ba0ce 100644

```
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -214,6 +214,9 @@ static int newque(struct ipc_namespace *ns, struct ipc_params *params)
  msq->q_cbytes = msq->q_qnum = 0;
  msq->q_qbytes = ns->msg_ctlmnb;
  msq->q_lspid = msq->q_lrpid = 0;
+#ifdef CONFIG_CHECKPOINT_RESTORE
+ msq->q_copy_cnt = 0;
+#endif
  INIT_LIST_HEAD(&msq->q_messages);
  INIT_LIST_HEAD(&msq->q_receivers);
  INIT_LIST_HEAD(&msq->q_senders);
@@ -784,19 +787,41 @@ long do_msgrcv(int msqid, void __user *buf, size_t bufsz, long msgtyp,
  struct msg_msg *msg;
  int mode;
  struct ipc_namespace *ns;
+#ifdef CONFIG_CHECKPOINT_RESTORE
+ struct msg_msg *copy = NULL;
+#endif

  if (msqid < 0 || (long) bufsz < 0)
   return -EINVAL;
  mode = convert_mode(&msgtyp, msgflg);
  ns = current->nsproxy->ipc_ns;

+ if (msgflg & MSG_COPY) {
+#ifdef CONFIG_CHECKPOINT_RESTORE
+  /*
+   * Create dummy message to copy real message to.
+   */
+  copy = load_msg(buf, bufsz);
+  if (IS_ERR(copy))
+   return PTR_ERR(copy);
+  copy->m_ts = bufsz;
+#else
+  return -ENOSYS;
+#endif
+ }
  msq = msg_lock_check(ns, msqid);
- if (IS_ERR(msq))
+ if (IS_ERR(msq)) {
+#ifdef CONFIG_CHECKPOINT_RESTORE
+  if (msgflg & MSG_COPY)
+   free_msg(copy);
+#endif
   return PTR_ERR(msq);
+ }
```

```
  for (;;) {
   struct msg_receiver msr_d;
   struct list_head *tmp;
+  int msg_counter = 0;

   msg = ERR_PTR(-EACCES);
   if (ipcperms(ns, &msq->q_perm, S_IRUGO))
@@ -816,10 +841,18 @@ long do_msgrcv(int msqid, void __user *buf, size_t bufsz, long msgtyp,
       walk_msg->m_type != 1) {
      msg = walk_msg;
      msgtyp = walk_msg->m_type - 1;
+#ifdef CONFIG_CHECKPOINT_RESTORE
+    } else if (msgflg & MSG_COPY) {
+     if (msq->q_copy_cnt == msg_counter) {
+      msg = copy_msg(walk_msg, copy);
+      break;
+     }
+#endif
     } else {
      msg = walk_msg;
      break;
     }
+    msg_counter++;
    }
    tmp = tmp->next;
   }
@@ -832,11 +865,21 @@ long do_msgrcv(int msqid, void __user *buf, size_t bufsz, long msgtyp,
     msg = ERR_PTR(-E2BIG);
     goto out_unlock;
    }
+#ifdef CONFIG_CHECKPOINT_RESTORE
+   if (msgflg & MSG_COPY) {
+    msq->q_copy_cnt++;
+    goto out_unlock;
+   }
+#endif
    list_del(&msg->m_list);
    msq->q_qnum--;
    msq->q_rtime = get_seconds();
    msq->q_lrpid = task_tgid_vnr(current);
    msq->q_cbytes -= msg->m_ts;
+#ifdef CONFIG_CHECKPOINT_RESTORE
+   if (msq->q_copy_cnt)
+    msq->q_copy_cnt--;
+#endif
    atomic_sub(msg->m_ts, &ns->msg_bytes);
    atomic_dec(&ns->msg_hdrs);
    ss_wakeup(&msq->q_senders, 0);
```

```
@@ -915,8 +958,13 @@ out_unlock:
    break;
   }
  }
- if (IS_ERR(msg))
+ if (IS_ERR(msg)) {
+#ifdef CONFIG_CHECKPOINT_RESTORE
+  if (msgflg & MSG_COPY)
+   free_msg(copy);
+#endif
   return PTR_ERR(msg);
+ }

  bufsz = msg_handler(buf, msg, bufsz);
  free_msg(msg);
diff --git a/ipc/msgutil.c b/ipc/msgutil.c
index 26143d3..b281f5c 100644
--- a/ipc/msgutil.c
+++ b/ipc/msgutil.c
@@ -100,7 +100,45 @@ out_err:
  free_msg(msg);
  return ERR_PTR(err);
 }
+#ifdef CONFIG_CHECKPOINT_RESTORE
+struct msg_msg *copy_msg(struct msg_msg *src, struct msg_msg *dst)
+{
+ struct msg_msgseg *dst_pseg, *src_pseg;
+ int len = src->m_ts;
+ int alen;
+
+ BUG_ON(dst == NULL);
+ if (src->m_ts > dst->m_ts)
+  return ERR_PTR(-EINVAL);
+
+ alen = len;
+ if (alen > DATALEN_MSG)
+  alen = DATALEN_MSG;
+
+ dst->next = NULL;
+ dst->security = NULL;

+ memcpy(dst + 1, src + 1, alen);
+
+ len -= alen;
+ dst_pseg = dst->next;
+ src_pseg = src->next;
+ while (len > 0) {
+  alen = len;
```

```
+  if (alen > DATALEN_SEG)
+   alen = DATALEN_SEG;
+  memcpy(dst_pseg + 1, src_pseg + 1, alen);
+  dst_pseg = dst_pseg->next;
+  len -= alen;
+  src_pseg = src_pseg->next;
+ }
+
+ dst->m_type = src->m_type;
+ dst->m_ts = src->m_ts;
+
+ return dst;
+}
+#endif
 int store_msg(void __user *dest, struct msg_msg *msg, int len)
 {
  int alen;
diff --git a/ipc/util.h b/ipc/util.h
index 2bc6a9a..c1e1d5c 100644
--- a/ipc/util.h
+++ b/ipc/util.h
@@ -142,6 +142,7 @@ int ipc_parse_version (int *cmd);

 extern void free_msg(struct msg_msg *msg);
 extern struct msg_msg *load_msg(const void __user *src, int len);
+extern struct msg_msg *copy_msg(struct msg_msg *src, struct msg_msg *dst);
 extern int store_msg(void __user *dest, struct msg_msg *msg, int len);

 extern void recompute_msgmni(struct ipc_namespace *);
```

Subject: [PATCH v3 09/10] ipc: add new MSG_SET_COPY command for sys_msgctl() call

Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:26:07 GMT

View Forum Message <> Reply to Message

New MSG_SET_COPY allows to set specified queue copy counter to passed value.
Passed "struct msqid_ds *buf" interpreted as pointer to unsigned int in this
case.

```
---
 include/linux/msg.h |    1 +
 ipc/compat.c        |    3 +++
 ipc/msg.c           |   18 ++++++++++++++++++
 3 files changed, 22 insertions(+), 0 deletions(-)

diff --git a/include/linux/msg.h b/include/linux/msg.h
index 07a5e1e..4e9ddf4 100644
--- a/include/linux/msg.h
```

```
+++ b/include/linux/msg.h
@@ -7,6 +7,7 @@
 #define MSG_STAT 11
 #define MSG_INFO 12
 #define MSG_SET  13
+#define MSG_SET_COPY  14

 /* msgrcv options */
 #define MSG_NOERROR     010000  /* no error if message is too big */
diff --git a/ipc/compat.c b/ipc/compat.c
index 6b07f5c..9e5acc7 100644
--- a/ipc/compat.c
+++ b/ipc/compat.c
@@ -481,6 +481,9 @@ long compat_sys_msgctl(int first, int second, void __user *uptr)
  case IPC_INFO:
  case IPC_RMID:
  case MSG_INFO:
+#ifdef CONFIG_CHECKPOINT_RESTORE
+ case MSG_SET_COPY:
+#endif
   err = sys_msgctl(first, second, uptr);
   break;

diff --git a/ipc/msg.c b/ipc/msg.c
index 44ba0ce..c63b22e 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -579,6 +579,24 @@ SYSCALL_DEFINE3(msgctl, int, msqid, int, cmd, struct msqid_ds
__user *, buf)
  case MSG_SET:
   err = msgctl_down(ns, msqid, cmd, buf, version);
   return err;
+#ifdef CONFIG_CHECKPOINT_RESTORE
+ case MSG_SET_COPY:
+ {
+  int copy_cnt;
+
+  if (!buf)
+   return -EFAULT;
+  if (get_user(copy_cnt, (unsigned int *)buf))
+   return -EFAULT;
+
+  msq = msg_lock(ns, msqid);
+  if (IS_ERR(msq))
+   return PTR_ERR(msq);
+  msq->q_copy_cnt = copy_cnt;
+  err = 0;
+  break;
```

```
+ }
+#endif
  default:
   return  -EINVAL;
  }
```

Subject: [PATCH v3 10/10] test: IPC message queue migration test
Posted by Stanislav Kinsbursky on Fri, 10 Aug 2012 14:26:12 GMT
View Forum Message <> Reply to Message

This test is a part of CRIU development test suit.

```
---
 tools/testing/selftests/ipc/msgque.c |  151 ++++++++++++++++++++++++++++++++++++
 1 files changed, 151 insertions(+), 0 deletions(-)
 create mode 100644 tools/testing/selftests/ipc/msgque.c

diff --git a/tools/testing/selftests/ipc/msgque.c b/tools/testing/selftests/ipc/msgque.c
new file mode 100644
index 0000000..c101e25
--- /dev/null
+++ b/tools/testing/selftests/ipc/msgque.c
@@ -0,0 +1,151 @@
+#define _GNU_SOURCE
+#include <sched.h>
+
+#include <stdio.h>
+#include <string.h>
+#include <stdlib.h>
+#include <unistd.h>
+#include <sys/types.h>
+#include <sys/wait.h>
+#include <sys/sem.h>
+#include <sys/ipc.h>
+#include <sys/msg.h>
+#include <signal.h>
+#include <errno.h>
+
+#include "zdtmtst.h"
+
+const char *test_doc="Tests sysv5 msg queues support by user space checkpointing";
+const char *test_author="Stanislav Kinsbursky <skinsbursky@openvz.org>";
+
+struct msg1 {
+ long mtype;
+ char mtext[20];
+};
+#define TEST_STRING "Test sysv5 msg"
```

```
+#define MSG_TYPE 1
+
+#define ANOTHER_TEST_STRING "Yet another test sysv5 msg"
+#define ANOTHER_MSG_TYPE 26538
+
+static int test_fn(int argc, char **argv)
+{
+ key_t key;
+ int msg, pid;
+ struct msg1 msgbuf;
+ int chret;
+
+ key = ftok(argv[0], 822155650);
+ if (key == -1) {
+  err("Can't make key");
+  exit(1);
+ }
+
+ pid = test_fork();
+ if (pid < 0) {
+  err("Can't fork");
+  exit(1);
+ }
+
+ msg = msgget(key, IPC_CREAT | IPC_EXCL | 0666);
+ if (msg == -1) {
+  msg = msgget(key, 0666);
+  if (msg == -1) {
+   err("Can't get queue");
+   goto err_kill;
+  }
+ }
+
+ if (pid == 0) {
+  /*
+   * Here is the place where test sleeps and waits for signal.
+   * This place is used for suspend/restore test.
+   */
+  test_waitsig();
+
+  if (msgrcv(msg, &msgbuf, sizeof(TEST_STRING), MSG_TYPE, IPC_NOWAIT) == -1) {
+   fail("Child: msgrcv failed (%m)");
+   return -errno;
+  }
+
+  if (strncmp(TEST_STRING, msgbuf.mtext, sizeof(TEST_STRING))) {
+   fail("Child: the source and received strings aren't equal");
+   return -errno;
```

```
+  }
+  test_msg("Child: received %s\n", msgbuf.mtext);
+
+  msgbuf.mtype = ANOTHER_MSG_TYPE;
+  memcpy(msgbuf.mtext, ANOTHER_TEST_STRING, sizeof(ANOTHER_TEST_STRING));
+  if (msgsnd(msg, &msgbuf, sizeof(ANOTHER_TEST_STRING), IPC_NOWAIT) != 0) {
+   fail("Child: msgsnd failed (%m)");
+   return -errno;
+  };
+  pass();
+  return 0;
+ } else {
+  msgbuf.mtype = MSG_TYPE;
+  memcpy(msgbuf.mtext, TEST_STRING, sizeof(TEST_STRING));
+  if (msgsnd(msg, &msgbuf, sizeof(TEST_STRING), IPC_NOWAIT) != 0) {
+   fail("Parent: msgsnd failed (%m)");
+   goto err_kill;
+  };
+
+  msgbuf.mtype = ANOTHER_MSG_TYPE;
+  memcpy(msgbuf.mtext, ANOTHER_TEST_STRING, sizeof(ANOTHER_TEST_STRING));
+  if (msgsnd(msg, &msgbuf, sizeof(ANOTHER_TEST_STRING), IPC_NOWAIT) != 0) {
+   fail("child: msgsnd (2) failed (%m)");
+   return -errno;
+  };
+
+  test_daemon();
+  test_waitsig();
+
+  kill(pid, SIGTERM);
+
+  wait(&chret);
+  chret = WEXITSTATUS(chret);
+  if (chret) {
+   fail("Parent: child exited with non-zero code %d (%s)\n",
+        chret, strerror(chret));
+   goto out;
+  }
+
+  if (msgrcv(msg, &msgbuf, sizeof(ANOTHER_TEST_STRING), ANOTHER_MSG_TYPE,
IPC_NOWAIT) == -1) {
+   fail("Parent: msgrcv failed (%m)");
+   goto err;
+  }
+
+  if (strncmp(ANOTHER_TEST_STRING, msgbuf.mtext, sizeof(ANOTHER_TEST_STRING))) {
+   fail("Parent: the source and received strings aren't equal");
+   goto err;
```

```
+  }
+  test_msg("Parent: received %s\n", msgbuf.mtext);
+
+  pass();
+ }
+
+out:
+ if (msgctl(msg, IPC_RMID, 0)) {
+  fail("Failed to destroy message queue: %d\n", -errno);
+  return -errno;
+ }
+ return chret;
+
+err_kill:
+ kill(pid, SIGKILL);
+ wait(NULL);
+err:
+ chret = -errno;
+ goto out;
+}
+
+int main(int argc, char **argv)
+{
+#ifdef NEW_IPC_NS
+ test_init_ns(argc, argv, CLONE_NEWIPC, test_fn);
+#else
+ test_init(argc, argv);
+ test_fn();
+#endif
+ return 0;
+}
```

Subject: Re: [PATCH v3 08/10] IPC: message queue copy feature introduced
Posted by Manfred Spraul on Sat, 11 Aug 2012 11:20:58 GMT
View Forum Message <> Reply to Message

Hi Stanislav,

2012/8/10 Stanislav Kinsbursky <skinsbursky@parallels.com>:
> This patch is required for checkpoint/restore in userspace.
> IOW, c/r requires some way to get all pending IPC messages without deleting
> them from the queue (checkpoint can fail and in this case tasks will be resumed,
> so queue have to be valid).
> To achive this, new operation flag MSG_COPY for sys_msgrcv() system call was
> introduced. Also, copy counter was added to msg_queue structure. It's set to
> zero by default and increases by one on each copy operation and decreased by
> one on each receive operation until reaches zero.

Is msq->q_copy_cnt really necessary?
As far as I can see user space needs the ability to read the n-th message.

The implementation adds a state variable to the kernel, adds two
automatic updates of the state into msgrcv() (an increase during
MSG_COPY, a decrease during normal receives) and adds a msgctl() to
set the state to a certain value.

a) What about the simpler approach:
- if MSG_COPY is set, then @mtype is interpreted as the number of the
message that should be copied.
  If there are less than @mtype messages, then -ENOMSG is returned.

b) I do not understand the purpose of the decrease of msq->q_copy_cnt:
Do you want to handle normal msgrcv() calls in parallel with
msgrcv(|MSG_COPY) calls?
I don't think that this will work:
What if msq->q_copy_cnt is 1 and and msgrcv() call receives the 20th
message in the queue?

--
  Manfred

Subject: Re: [PATCH v3 08/10] IPC: message queue copy feature introduced
Posted by Stanislav Kinsbursky on Sat, 11 Aug 2012 13:31:53 GMT
View Forum Message <> Reply to Message

> Hi Stanislav,
>
> 2012/8/10 Stanislav Kinsbursky <skinsbursky@parallels.com>:
>> This patch is required for checkpoint/restore in userspace.
>> IOW, c/r requires some way to get all pending IPC messages without deleting
>> them from the queue (checkpoint can fail and in this case tasks will be resumed,
>> so queue have to be valid).
>> To achive this, new operation flag MSG_COPY for sys_msgrcv() system call was
>> introduced. Also, copy counter was added to msg_queue structure. It's set to
>> zero by default and increases by one on each copy operation and decreased by
>> one on each receive operation until reaches zero.
> Is msq->q_copy_cnt really necessary?
> As far as I can see user space needs the ability to read the n-th message.
>
> The implementation adds a state variable to the kernel, adds two
> automatic updates of the state into msgrcv() (an increase during
> MSG_COPY, a decrease during normal receives) and adds a msgctl() to
> set the state to a certain value.

>
> a) What about the simpler approach:
> - if MSG_COPY is set, then @mtype is interpreted as the number of the
> message that should be copied.
> If there are less than @mtype messages, then -ENOMSG is returned.

Hi, Manfred.
Your approach is simplier, but makes the call less generic and adds limitations.
I.e. sys_msgrcv() allows you to receive message by type. And from my pow this logic have to be preserved - you can specify type and then copy all the messages of specified type.

> b) I do not understand the purpose of the decrease of msq->q_copy_cnt:
> Do you want to handle normal msgrcv() calls in parallel with
> msgrcv(|MSG_COPY) calls?

Actually, I'm not going to copy a message from a queue, when somebody is reading from it. But better to handle this case by decreasing msq->q_copy_cnt, because otherwise this counter becomes invalid in case of somebody is reading from queue. And this logic is similar to new "peek" logic for sockets (introduced in 3.4 or 3.5).
But I understand, that in case of queue with messages with different types this approach works only if mtype is not specified for copy operation. Otherwise result is unpredictable.

> I don't think that this will work:
> What if msq->q_copy_cnt is 1 and and msgrcv() call receives the 20th
> message in the queue?

By "receives" you mean "copied"? If so, then it can happen only if mtype was specified. And this logic is a part of current implementation.

>
> --
> Manfred

Subject: Re: [PATCH v3 08/10] IPC: message queue copy feature introduced
Posted by Manfred Spraul on Sun, 12 Aug 2012 09:48:18 GMT
View Forum Message <> Reply to Message

Hi Stanislav,

2012/8/11 Stanislav Kinsbursky <skinsbursky@parallels.com>:

>> a) What about the simpler approach:
>> - if MSG_COPY is set, then @mtype is interpreted as the number of the

>> message that should be copied.
>>   If there are less than @mtype messages, then -ENOMSG is returned.
>
>
> Hi, Manfred.
> Your approach is simplier, but makes the call less generic and adds
> limitations.
> I.e. sys_msgrcv() allows you to receive message by type. And from my pow
> this logic have to be preserved - you can specify type and then copy all the
> messages of specified type.
>

Your implementation adds the ability to select a message for MSG_COPY by id.
But I think the price is way too high:
a) added complexity
b) I didn't notice it immediately:
The implementation means that MSG_COPY cannot be used at all by
multiple processes:

task 1: msgrcv(id,buf,len,0,MSG_COPY).
task 2: msgrcv(id,buf,len,0,MSG_COPY).

It is unpredictable if a task receives the first or the 2nd message
from the queue.

task 1: int msgnr=0;
         msgctl(id,MSG_SET_COPY,&msgnr)
         msgrcv(id,buf,len,0,MSG_COPY).
task 2: int msgnr=0;
         msgctl(id,MSG_SET_COPY,&msgnr)
         msgrcv(id,buf,len,0,MSG_COPY).

Doesn't work either, it's a race condition.

>
>> b) I do not understand the purpose of the decrease of msq->q_copy_cnt:
>> Do you want to handle normal msgrcv() calls in parallel with
>> msgrcv(|MSG_COPY) calls?
>
>
> Actually, I'm not going to copy a message from a queue, when somebody is
> reading from it. But better to handle this case by decreasing
> msq->q_copy_cnt, because otherwise this counter becomes invalid in case of
> somebody is reading from queue. And this logic is similar to new "peek"
> logic for sockets (introduced in 3.4 or 3.5).
> But I understand, that in case of queue with messages with different types
> this approach works only if mtype is not specified for copy operation.
> Otherwise result is unpredictable.

>
a) If the result is unpredictable when mtype is used, does it make
   sense to implement msgcrl(id,buf,len,id=<x>,MSG_COPY)?

b) The result is also unpredictable if mtype is used for the "normal"
   msgrcv() (i.e. without MSG_COPY) call.
>
>> I don't think that this will work:
>> What if msq->q_copy_cnt is 1 and and msgrcv() call receives the 20th
>> message in the queue?
>
>
> By "receives" you mean "copied"? If so, then it can happen only if mtype was
> specified. And this logic is a part of current implementation.
>

I was thinking about the following case:

```
task 1: /* copy all entries */
        errno=0;
        for (i=0;errno<>0;i++)
              msgrcv(msgid,buf[i],buflen,0,MSG_COPY);

task 2: /* receive a message with a given ID */
        msgrcv(msqid,buf,buflen,123,0);
```

Now suppose that task 1 has read 5 messages from the queue and then
task 2 tries to receive the message with the id=123. Suppose this is
the 20th message in the queue.

Result: task 1 will copy the message 5 twice.

I would keep it simple - unless there is a clear use case where "peek
by id" is useful.

Or - since MSG_COPY is linux specific anyway:
What about storing the number of the message that should be returned in *msgp?
Store it as "int64", just to avoid any 32/64 bit issues.

--
  Manfred

Subject: Re: [PATCH v3 08/10] IPC: message queue copy feature introduced
Posted by Stanislav Kinsbursky on Mon, 13 Aug 2012 10:35:26 GMT
View Forum Message <> Reply to Message

> I would keep it simple - unless there is a clear use case where "peek
> by id" is useful.
>
> Or - since MSG_COPY is linux specific anyway:
> What about storing the number of the message that should be returned in *msgp?
> Store it as "int64", just to avoid any 32/64 bit issues.

Sounds reasonable. But, probably, mtype suit better for message number.
I'll update.

--
Best regards,
Stanislav Kinsbursky