Subject: [PATCH v5 00/10] IPC: checkpoint/restore in userspace enhancements
Posted by Stanislav Kinsbursky on Wed, 19 Sep 2012 16:05:21 GMT
View Forum Message <> Reply to Message

v5:
1) Several define-dependent compile bugs fixed
2) IPC message copy test updated
3) A couple of minor fixes.
4) Qlogic driver update: rename of its internal SEM_SET define into SEM_INIT
(compile error).

v4:
1) If MSG_COPY flag is specified, then "mtype" is not a type, but message
number to copy.
2) MSG_SET_COPY logic for sys_msgctl() was removed.

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(). If
MSG_COPY flag is set, then msgtyp is interpreted as message number.

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
    glge driver: rename internal SEM_SET macro to SEM_INIT

ipc: add new SEM_SET command for sys_semctl() call
IPC: message queue receive cleanup
IPC: message queue copy feature introduced
test: IPC message queue copy feture test

```
 drivers/net/ethernet/qlogic/qlge/qlge.h      |    4
 drivers/net/ethernet/qlogic/qlge/qlge_main.c |   16 +-
 include/linux/ipc.h                          |    1
 include/linux/msg.h                          |    7 +
 include/linux/sem.h                          |    1
 include/linux/shm.h                          |    1
 ipc/compat.c                                 |   45 +++--
 ipc/msg.c                                    |  112 ++++++++++---
 ipc/msgutil.c                                |   38 ++++
 ipc/sem.c                                    |   14 +
 ipc/shm.c                                    |   17 +-
 ipc/util.c                                   |   69 +++++++
 ipc/util.h                                   |    6 +
 security/selinux/hooks.c                     |    3
 security/smack/smack_lsm.c                   |    3
 tools/testing/selftests/ipc/Makefile         |   28 +++
 tools/testing/selftests/ipc/msgque.c         |  251 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 17 files changed, 547 insertions(+), 69 deletions(-)
 create mode 100644 tools/testing/selftests/ipc/Makefile
 create mode 100644 tools/testing/selftests/ipc/msgque.c
```

--
Signature

Subject: [PATCH v5 01/10] ipc: remove forced assignment of selected message
Posted by Stanislav Kinsbursky on Wed, 19 Sep 2012 16:05:31 GMT
View Forum Message <> Reply to Message

This is a cleanup patch. The assignment is redundant.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>

```
 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 v5 02/10] ipc: "use key as id" functionality for resource get system call i
Posted by Stanislav Kinsbursky on Wed, 19 Sep 2012 16:05:37 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.

Note: if desired slot is not emply, then next free slot will be used.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>

```
---
 include/linux/ipc.h |    1 +
 ipc/msg.c           |    4 +++-
 ipc/sem.c           |    4 +++-
 ipc/shm.c           |    4 +++-
 ipc/util.c          |   18 ++++++++++++++---
 ipc/util.h          |    3 ++-
 6 files changed, 27 insertions(+), 7 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..1cecaf2 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);
@@ -201,7 +202,8 @@ static int newque(struct ipc_namespace *ns, struct ipc_params *params)
  /*
   * ipc_addid() locks msq
   */
- id = ipc_addid(&msg_ids(ns), &msq->q_perm, ns->msg_ctlmni);
+ id = ipc_addid(&msg_ids(ns), &msq->q_perm, ns->msg_ctlmni,
+         msgflg & IPC_PRESET);
  if (id < 0) {
   security_msg_queue_free(msq);
   ipc_rcu_putref(msq);
diff --git a/ipc/sem.c b/ipc/sem.c
index 5215a81..e89b90c 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);
@@ -314,7 +315,8 @@ static int newary(struct ipc_namespace *ns, struct ipc_params *params)
   return retval;
  }

- id = ipc_addid(&sem_ids(ns), &sma->sem_perm, ns->sc_semmni);
+ id = ipc_addid(&sem_ids(ns), &sma->sem_perm, ns->sc_semmni,
+         semflg & IPC_PRESET);
  if (id < 0) {
   security_sem_free(sma);
   ipc_rcu_putref(sma);
diff --git a/ipc/shm.c b/ipc/shm.c
index 00faa05..0088418 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;
@@ -510,7 +511,8 @@ static int newseg(struct ipc_namespace *ns, struct ipc_params *params)
  if (IS_ERR(file))
   goto no_file;

- id = ipc_addid(&shm_ids(ns), &shp->shm_perm, ns->shm_ctlmni);
+ id = ipc_addid(&shm_ids(ns), &shp->shm_perm, ns->shm_ctlmni,
+         shmflg & IPC_PRESET);
  if (id < 0) {
   error = id;
   goto no_id;
diff --git a/ipc/util.c b/ipc/util.c
index eb07fd3..328abd1 100644
--- a/ipc/util.c
+++ b/ipc/util.c
@@ -238,16 +238,22 @@ int ipc_get_maxid(struct ipc_ids *ids)
  * @ids: IPC identifier set
  * @new: new IPC permission set
  * @size: limit for the number of used ids
+ * @preset: use passed new->id value as desired id
  *
  * Add an entry 'new' to the IPC ids idr. The permissions object is
  * initialised and the first free entry is set up and the id assigned
  * is returned. The 'new' entry is returned in a locked state on success.
  * On failure the entry is not locked and a negative err-code is returned.
  *
+ * If 'preset' is set, then passed new->id is desired to be set for new
+ * segment. And allocated id is equal to passed value, then ipc ids will
+ * left unchanged and new->seq will be updated to correspond specified id value.
+ *
  * Called with ipc_ids.rw_mutex held as a writer.
  */

-int ipc_addid(struct ipc_ids* ids, struct kern_ipc_perm* new, int size)
+int ipc_addid(struct ipc_ids* ids, struct kern_ipc_perm* new, int size,
+       int preset)
 {
  uid_t euid;
  gid_t egid;
@@ -264,7 +270,8 @@ 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 +284,11 @@ 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 (preset && 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;
@@ -736,7 +748,7 @@ struct kern_ipc_perm *ipc_lock_check(struct ipc_ids *ids, int id)
 int ipcget(struct ipc_namespace *ns, struct ipc_ids *ids,
    struct ipc_ops *ops, struct ipc_params *params)
 {
- if (params->key == IPC_PRIVATE)
+ if (params->key == IPC_PRIVATE && ((params->flg & IPC_PRESET) == 0))
   return ipcget_new(ns, ids, ops, params);
  else
   return ipcget_public(ns, ids, ops, params);
diff --git a/ipc/util.h b/ipc/util.h
index 850ef3e..878df18 100644
--- a/ipc/util.h
+++ b/ipc/util.h
@@ -92,9 +92,10 @@ 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);
+int ipc_addid(struct ipc_ids *, struct kern_ipc_perm *, int, int);

 /* must be called with ids->rw_mutex acquired for reading */
 int ipc_get_maxid(struct ipc_ids *);
```

Subject: [PATCH v5 03/10] ipc: segment key change helper introduced
Posted by Stanislav Kinsbursky on Wed, 19 Sep 2012 16:05:42 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>
---
 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 ad9518e..af30d13 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 328abd1..1154245 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
  *
@@ -388,7 +408,7 @@ 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))
```

```
@@ -755,6 +775,31 @@ 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 878df18..b48016d 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 v5 04/10] ipc: add new SHM_SET command for sys_shmctl() call
Posted by Stanislav Kinsbursky on Wed, 19 Sep 2012 16:05:48 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>
---
 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 edd0868..9a3e423 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 af30d13..35c750d 100644
--- a/ipc/compat.c
+++ b/ipc/compat.c
@@ -692,6 +692,7 @@ 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 0088418..65c0c5c 100644
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -636,6 +636,9 @@ 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;
     }
@@ -740,12 +743,13 @@ 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) ? cmd : IPC_SET,
           &shmid64.shm_perm, 0);
  if (IS_ERR(ipcp))
   return PTR_ERR(ipcp);
@@ -759,6 +763,10 @@ 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();
@@ -936,6 +944,7 @@ 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 6c77f63..928ffc2 100644
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
```

```
@@ -5058,6 +5058,7 @@ 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 8221514..0f2c481 100644
--- a/security/smack/smack_lsm.c
+++ b/security/smack/smack_lsm.c
@@ -2142,6 +2142,7 @@ 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 v5 05/10] ipc: add new MSG_SET command for sys_msgctl() call
Posted by Stanislav Kinsbursky on Wed, 19 Sep 2012 16:05:54 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>

```
---
 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 35c750d..9c70f9a 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 1cecaf2..ef4f118 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -392,6 +392,9 @@ 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;
@@ -418,12 +421,13 @@ 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);
@@ -439,6 +443,7 @@ 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;
@@ -447,6 +452,9 @@ 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
@@ -566,6 +574,7 @@ 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 928ffc2..8269286 100644
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
@@ -4916,6 +4916,7 @@ 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 0f2c481..193e147 100644
--- a/security/smack/smack_lsm.c
+++ b/security/smack/smack_lsm.c
@@ -2395,6 +2395,7 @@ 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 v5 06/10] glge driver: rename internal SEM_SET macro to SEM_INIT
Posted by Stanislav Kinsbursky on Wed, 19 Sep 2012 16:05:59 GMT
View Forum Message <> Reply to Message

The reason for shit patch is that SET_SET is desired to be a part of new part
of API of IPC sys_semctl() system call.
The name itself for IPC is quite natural, because all linux-specific commands
names for IPC system calls are originally created by replacing "IPC_" part by
"SEM_"("MSG_", "SHM_") part.
So, I'm hoping, that this change doesn't really matters for "QLogic qlge NIC
HBA Driver" developers, since it's just an internal define.
---
 drivers/net/ethernet/qlogic/qlge/qlge.h | 4 ++--
 drivers/net/ethernet/qlogic/qlge/qlge_main.c | 16 ++++++++--------
 2 files changed, 10 insertions(+), 10 deletions(-)

```
diff --git a/drivers/net/ethernet/qlogic/qlge/qlge.h b/drivers/net/ethernet/qlogic/qlge/qlge.h
index a131d7b..6f46ea5 100644
--- a/drivers/net/ethernet/qlogic/qlge/qlge.h
+++ b/drivers/net/ethernet/qlogic/qlge/qlge.h
@@ -347,10 +347,10 @@ enum {
 enum {
  /*
   * Example:
-  * reg = SEM_XGMAC0_MASK | (SEM_SET << SEM_XGMAC0_SHIFT)
+  * reg = SEM_XGMAC0_MASK | (SEM_INIT << SEM_XGMAC0_SHIFT)
   */
  SEM_CLEAR = 0,
- SEM_SET = 1,
+ SEM_INIT = 1,
  SEM_FORCE = 3,
  SEM_XGMAC0_SHIFT = 0,
  SEM_XGMAC1_SHIFT = 2,
diff --git a/drivers/net/ethernet/qlogic/qlge/qlge_main.c
b/drivers/net/ethernet/qlogic/qlge/qlge_main.c
index b53a3b6..18e0a0c 100644
--- a/drivers/net/ethernet/qlogic/qlge/qlge_main.c
+++ b/drivers/net/ethernet/qlogic/qlge/qlge_main.c
@@ -109,28 +109,28 @@ static int ql_sem_trylock(struct ql_adapter *qdev, u32 sem_mask)

  switch (sem_mask) {
  case SEM_XGMAC0_MASK:
-  sem_bits = SEM_SET << SEM_XGMAC0_SHIFT;
```

```
+  sem_bits = SEM_INIT << SEM_XGMAC0_SHIFT;
   break;
  case SEM_XGMAC1_MASK:
-  sem_bits = SEM_SET << SEM_XGMAC1_SHIFT;
+  sem_bits = SEM_INIT << SEM_XGMAC1_SHIFT;
   break;
  case SEM_ICB_MASK:
-  sem_bits = SEM_SET << SEM_ICB_SHIFT;
+  sem_bits = SEM_INIT << SEM_ICB_SHIFT;
   break;
  case SEM_MAC_ADDR_MASK:
-  sem_bits = SEM_SET << SEM_MAC_ADDR_SHIFT;
+  sem_bits = SEM_INIT << SEM_MAC_ADDR_SHIFT;
   break;
  case SEM_FLASH_MASK:
-  sem_bits = SEM_SET << SEM_FLASH_SHIFT;
+  sem_bits = SEM_INIT << SEM_FLASH_SHIFT;
   break;
  case SEM_PROBE_MASK:
-  sem_bits = SEM_SET << SEM_PROBE_SHIFT;
+  sem_bits = SEM_INIT << SEM_PROBE_SHIFT;
   break;
  case SEM_RT_IDX_MASK:
-  sem_bits = SEM_SET << SEM_RT_IDX_SHIFT;
+  sem_bits = SEM_INIT << SEM_RT_IDX_SHIFT;
   break;
  case SEM_PROC_REG_MASK:
-  sem_bits = SEM_SET << SEM_PROC_REG_SHIFT;
+  sem_bits = SEM_INIT << SEM_PROC_REG_SHIFT;
   break;
  default:
   netif_alert(qdev, probe, qdev->ndev, "bad Semaphore mask!.\n");
```

Subject: [PATCH v5 07/10] ipc: add new SEM_SET command for sys_semctl() call

Posted by Stanislav Kinsbursky on Wed, 19 Sep 2012 16:06:05 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>

```
---
 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 9c70f9a..84d8efd 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 e89b90c..b4f80082 100644
--- a/ipc/sem.c
+++ b/ipc/sem.c
@@ -1085,12 +1085,13 @@ 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) ? cmd : IPC_SET,
           &semid64.sem_perm, 0);
  if (IS_ERR(ipcp))
   return PTR_ERR(ipcp);
```

```
@@ -1105,6 +1106,10 @@ 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();
@@ -1150,6 +1155,7 @@ 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 8269286..a23a7d6 100644
--- a/security/selinux/hooks.c
+++ b/security/selinux/hooks.c
@@ -5164,6 +5164,7 @@ 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 193e147..fea40cf 100644
--- a/security/smack/smack_lsm.c
+++ b/security/smack/smack_lsm.c
@@ -2274,6 +2274,7 @@ 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 v5 08/10] IPC: message queue receive cleanup

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>
---
 include/linux/msg.h | 5 +++--
 ipc/compat.c | 36 ++++++++++++++++++++++++++----------
 ipc/msg.c | 44 ++
 3 files changed, 45 insertions(+), 40 deletions(-)

```
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 84d8efd..b879d50 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 = 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 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)
@@ -394,7 +395,8 @@ long compat_sys_msgrcv(int msqid, struct compat_msgbuf __user
*msgp,
 {
  long err, mtype;

- err =  do_msgrcv(msqid, &mtype, msgp->mtext, (ssize_t)msgsz, msgtyp, msgflg);
+ err =  do_msgrcv(msqid, &mtype, msgp->mtext, (ssize_t)msgsz, msgtyp,
+     msgflg, compat_do_msg_fill);
```

```
  if (err < 0)
   goto out;

diff --git a/ipc/msg.c b/ipc/msg.c
index ef4f118..d8168a7 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -763,15 +763,30 @@ 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;
@@ -814,7 +829,7 @@ 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;
    }
@@ -841,7 +856,7 @@ 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);
@@ -904,29 +919,16 @@ 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 v5 09/10] IPC: message queue copy feature introduced

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. If this flag was specified, then mtype is interpreted as number of
the message to copy.
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) 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 | 1 +
 ipc/msg.c | 50 ++--
 ipc/msgutil.c | 38 ++++++++++++++++++++++++++++++++++++++
 ipc/util.h | 1 +
 4 files changed, 88 insertions(+), 2 deletions(-)

```
diff --git a/include/linux/msg.h b/include/linux/msg.h
index 9411b76..4ca337f 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 {
diff --git a/ipc/msg.c b/ipc/msg.c
index d8168a7..0984f07 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -785,19 +785,48 @@ 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;
+ unsigned long copy_number = 0;
```

```
+#endif

  if (msqid < 0 || (long) bufsz < 0)
   return -EINVAL;
+ if (msgflg & MSG_COPY) {
+#ifdef CONFIG_CHECKPOINT_RESTORE
+
+  if (msgflg & MSG_COPY) {
+   copy_number = msgtyp;
+   msgtyp = 0;
+  }
+
+  /*
+   * 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
+ }
  mode = convert_mode(&msgtyp, msgflg);
  ns = current->nsproxy->ipc_ns;

  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;
+  long msg_counter = 0;

   msg = ERR_PTR(-EACCES);
   if (ipcperms(ns, &msq->q_perm, S_IRUGO))
@@ -817,10 +846,18 @@ 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 (copy_number == msg_counter) {
+      msg = copy_msg(walk_msg, copy);
+      break;
+     }
+#endif
     } else {
      msg = walk_msg;
      break;
     }
+    msg_counter++;
    }
    tmp = tmp->next;
   }
@@ -833,6 +870,10 @@ 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)
+    goto out_unlock;
+#endif
    list_del(&msg->m_list);
    msq->q_qnum--;
    msq->q_rtime = get_seconds();
@@ -916,8 +957,13 @@ 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 b48016d..953339f 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 v5 10/10] test: IPC message queue copy feture test
Posted by Stanislav Kinsbursky on Wed, 19 Sep 2012 16:06:20 GMT
View Forum Message <> Reply to Message

This test can be used to check wheither kernel supports IPC message queue copy
and restore features (required by CRIU project).

```
---
 tools/testing/selftests/ipc/Makefile |   28 ++++
 tools/testing/selftests/ipc/msgque.c |  251 ++++++++++++++++++++++++++++++++++
 2 files changed, 279 insertions(+), 0 deletions(-)
 create mode 100644 tools/testing/selftests/ipc/Makefile
 create mode 100644 tools/testing/selftests/ipc/msgque.c

diff --git a/tools/testing/selftests/ipc/Makefile b/tools/testing/selftests/ipc/Makefile
new file mode 100644
index 0000000..6c547bf
--- /dev/null
+++ b/tools/testing/selftests/ipc/Makefile
@@ -0,0 +1,28 @@
+uname_M := $(shell uname -m 2>/dev/null || echo not)
+ARCH ?= $(shell echo $(uname_M) | sed -e s/i.86/i386/)
+ifeq ($(ARCH),i386)
+        ARCH := X86
+ CFLAGS := -DCONFIG_X86_32 -D__i386__
+endif
+ifeq ($(ARCH),x86_64)
+ ARCH := X86
+ CFLAGS := -DCONFIG_X86_64 -D__x86_64__
+endif
+
+CFLAGS += -I../../../../arch/x86/include/generated/
+CFLAGS += -I../../../../include/
+CFLAGS += -I../../../../usr/include/
+CFLAGS += -I../../../../arch/x86/include/
+
+all:
+ifeq ($(ARCH),X86)
+ gcc $(CFLAGS) msgque.c -o msgque_test
+else
+ echo "Not an x86 target, can't build msgque selftest"
```

```
+endif
+
+run_tests: all
+ ./msgque_test
+
+clean:
+ rm -fr ./msgque_test
diff --git a/tools/testing/selftests/ipc/msgque.c b/tools/testing/selftests/ipc/msgque.c
new file mode 100644
index 0000000..ffcc8b7
--- /dev/null
+++ b/tools/testing/selftests/ipc/msgque.c
@@ -0,0 +1,251 @@
+#include <stdio.h>
+#include <sys/types.h>
+#include <sys/ipc.h>
+#include <sys/msg.h>
+#include <errno.h>
+#include <string.h>
+#include <stdlib.h>
+
+#define MAX_MSG_SIZE  32
+
+struct msg1 {
+ int msize;
+ long mtype;
+ char mtext[MAX_MSG_SIZE];
+};
+
+#define TEST_STRING "Test sysv5 msg"
+#define MSG_TYPE 1
+
+#define ANOTHER_TEST_STRING "Yet another test sysv5 msg"
+#define ANOTHER_MSG_TYPE 26538
+
+#ifndef IPC_PRESET
+#define IPC_PRESET  00040000
+#endif
+
+#ifndef MSG_COPY
+#define MSG_COPY  040000
+#endif
+
+#ifndef MSG_SET
+#define MSG_SET   13
+#endif
+
+#if defined (__GLIBC__) && __GLIBC__ >= 2
```

```
+#define KEY __key
+#else
+#define KEY key
+#endif
+
+struct msgque_data {
+ int msq_id;
+ int qbytes;
+ int kern_id;
+ int qnum;
+ int mode;
+ struct msg1 *messages;
+};
+
+int restore_queue(struct msgque_data *msgque)
+{
+ struct msqid_ds ds;
+ int id, i;
+
+ id = msgget(msgque->msq_id,
+       msgque->mode | IPC_CREAT | IPC_EXCL | IPC_PRESET);
+ if (id == -1) {
+  printf("Failed to create queue\n");
+  return -errno;
+ }
+
+ if (id != msgque->msq_id) {
+  printf("Failed to preset id (%d instead of %d)\n",
+       id, msgque->msq_id);
+  return -EFAULT;
+ }
+
+ if (msgctl(id, MSG_STAT, &ds) < 0) {
+  printf("Failed to stat queue\n");
+  return -errno;
+ }
+
+ ds.msg_perm.KEY = msgque->msq_id;
+ ds.msg_qbytes = msgque->qbytes;
+ if (msgctl(id, MSG_SET, &ds) < 0) {
+  printf("Failed to update message key\n");
+  return -errno;
+ }
+
+ for (i = 0; i < msgque->qnum; i++) {
+  if (msgsnd(msgque->msq_id, &msgque->messages[i].mtype, msgque->messages[i].msize,
IPC_NOWAIT) != 0) {
+   printf("msgsnd failed (%m)\n");
```

```
+   return -errno;
+  };
+ }
+ return 0;
+}
+
+int check_and_destroy_queue(struct msgque_data *msgque)
+{
+ struct msg1 message;
+ int cnt = 0, ret;
+
+ while (1) {
+  ret = msgrcv(msgque->msq_id, &message.mtype, MAX_MSG_SIZE, 0, IPC_NOWAIT);
+  if (ret < 0) {
+   if (errno == ENOMSG)
+    break;
+   printf("Failed to read IPC message: %m\n");
+   ret = -errno;
+   goto err;
+  }
+  if (ret != msgque->messages[cnt].msize) {
+   printf("Wrong message size: %d (expected %d)\n", ret, msgque->messages[cnt].msize);
+   ret = -EINVAL;
+   goto err;
+  }
+  if (message.mtype != msgque->messages[cnt].mtype) {
+   printf("Wrong message type\n");
+   ret = -EINVAL;
+   goto err;
+  }
+  if (memcmp(message.mtext, msgque->messages[cnt].mtext, ret)) {
+   printf("Wrong message content\n");
+   ret = -EINVAL;
+   goto err;
+  }
+  cnt++;
+ }
+
+ if (cnt != msgque->qnum) {
+  printf("Wrong message number\n");
+  ret = -EINVAL;
+  goto err;
+ }
+
+ ret = 0;
+err:
+ if (msgctl(msgque->msq_id, IPC_RMID, 0)) {
+  printf("Failed to destroy queue: %d\n", -errno);
```

```
+  return -errno;
+ }
+ return ret;
+}
+
+int dump_queue(struct msgque_data *msgque)
+{
+ struct msqid_ds ds;
+ int i, ret;
+
+ for (msgque->kern_id = 0; msgque->kern_id < 256; msgque->kern_id++) {
+  ret = msgctl(msgque->kern_id, MSG_STAT, &ds);
+  if (ret < 0) {
+   if (errno == -EINVAL)
+    continue;
+   printf("Failed to get stats for IPC queue with id %d\n", msgque->kern_id);
+   return -errno;
+  }
+
+  if (ret == msgque->msq_id)
+   break;
+ }
+
+ msgque->messages = malloc(sizeof(struct msg1) * ds.msg_qnum);
+ if (msgque->messages == NULL) {
+  printf("Failed to get stats for IPC queue\n");
+  return -ENOMEM;
+ }
+
+ msgque->qnum = ds.msg_qnum;
+ msgque->mode = ds.msg_perm.mode;
+ msgque->qbytes = ds.msg_qbytes;
+
+ for (i = 0; i < msgque->qnum; i++) {
+  ret = msgrcv(msgque->msq_id, &msgque->messages[i].mtype, MAX_MSG_SIZE, i,
IPC_NOWAIT | MSG_COPY);
+  if (ret < 0) {
+   printf("Failed to copy IPC message: %m (%d)\n", errno);
+   return -errno;
+  }
+  msgque->messages[i].msize = ret;
+ }
+ return 0;
+}
+
+int fill_msgque(struct msgque_data *msgque)
+{
+ struct msg1 msgbuf;
```

```
+
+ msgbuf.mtype = MSG_TYPE;
+ memcpy(msgbuf.mtext, TEST_STRING, sizeof(TEST_STRING));
+ if (msgsnd(msgque->msq_id, &msgbuf.mtype, sizeof(TEST_STRING), IPC_NOWAIT) != 0) {
+  printf("First message send failed (%m)\n");
+  return -errno;
+ };
+
+ msgbuf.mtype = ANOTHER_MSG_TYPE;
+ memcpy(msgbuf.mtext, ANOTHER_TEST_STRING, sizeof(ANOTHER_TEST_STRING));
+ if (msgsnd(msgque->msq_id, &msgbuf.mtype, sizeof(ANOTHER_TEST_STRING),
IPC_NOWAIT) != 0) {
+  printf("Second message send failed (%m)\n");
+  return -errno;
+ };
+ return 0;
+}
+
+int main (int argc, char **argv)
+{
+ key_t key;
+ int msg, pid, err;
+ struct msgque_data msgque;
+
+ key = ftok(argv[0], 822155650);
+ if (key == -1) {
+  printf("Can't make key\n");
+  return -errno;
+ }
+
+ msgque.msq_id = msgget(key, IPC_CREAT | IPC_EXCL | 0666);
+ if (msgque.msq_id == -1) {
+  printf("Can't create queue\n");
+  goto err_out;
+ }
+
+ err = fill_msgque(&msgque);
+ if (err) {
+  printf("Failed to fill queue\n");
+  goto err_destroy;
+ }
+
+ err = dump_queue(&msgque);
+ if (err) {
+  printf("Failed to dump queue\n");
+  goto err_destroy;
+ }
+
```

```
+ err = check_and_destroy_queue(&msgque);
+ if (err) {
+  printf("Failed to check and destroy queue\n");
+  goto err_out;
+ }
+
+ err = restore_queue(&msgque);
+ if (err) {
+  printf("Failed to restore queue\n");
+  goto err_destroy;
+ }
+
+ err = check_and_destroy_queue(&msgque);
+ if (err) {
+  printf("Failed to test queue\n");
+  goto err_out;
+ }
+ return 0;
+
+err_destroy:
+ if (msgctl(msgque.msq_id, IPC_RMID, 0)) {
+  printf("Failed to destroy queue: %d\n", -errno);
+  return -errno;
+ }
+err_out:
+ return err;
+}
```

Subject: Re: [PATCH v5 01/10] ipc: remove forced assignment of selected message
Posted by Serge E. Hallyn on Wed, 26 Sep 2012 17:37:18 GMT
View Forum Message <> Reply to Message

Quoting Stanislav Kinsbursky (skinsbursky@parallels.com):
> This is a cleanup patch. The assignment is redundant.
>
> Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
> ---
> 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;

Perhaps your tree is different from mine, but it looks to me like it would
be simpler to remove the 'msg = walk_msg' from both the 'if' and 'else',
and keep them above the if/else?

Subject: Re: [PATCH v5 01/10] ipc: remove forced assignment of selected message
Posted by Stanislav Kinsbursky on Thu, 27 Sep 2012 06:53:21 GMT
View Forum Message <> Reply to Message

> Quoting Stanislav Kinsbursky (skinsbursky@parallels.com):
>> This is a cleanup patch. The assignment is redundant.
>>
>> Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
>> ---
>>   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;
>
> Perhaps your tree is different from mine, but it looks to me like it would
> be simpler to remove the 'msg = walk_msg' from both the 'if' and 'else',
> and keep them above the if/else?
>

Yep, thanks. Will update.

--
Best regards,
Stanislav Kinsbursky
