Subject: [PATCH v6 02/10] ipc: "use key as id" functionality for resource get system call i
Posted by Stanislav Kinsbursky on Mon, 15 Oct 2012 15:59:46 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/uapi/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/uapi/linux/ipc.h b/include/uapi/linux/ipc.h
index de08dd4..f5f52b6 100644
--- a/include/uapi/linux/ipc.h
+++ b/include/uapi/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 2f272fa..2f44946 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 58d31f1..10e9085 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 dff40c9..80b0046 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 72fd078..503946e 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)
 {
  kuid_t euid;
  kgid_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 c8fe2f7..3a9e558 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: Re: [PATCH v6 02/10] ipc: "use key as id" functionality for resource get system ca
Posted by ebiederm on Mon, 15 Oct 2012 19:39:31 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> 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.

This way of handling things is pretty nasty.

- You don't fail if the requested id is not available.
- You don't allow assigning the key (which leads to the need to change
  the key in later patches). Changing the creator uid and creator
  gid and key is semantically ugly.

It would be much cleaner if you could instead add IPC_PRESET and then
extend the definition of the creation functions all by one argument.

aka
int msgget(key_t key, int msgflg, int id);
int semget(key_t key, int nsems, int semflg, int id);
int shmget(key_t key, size_t size, int shmflg, int id);

Where the extra id argument is ignored unless IPC_PRESET is specified.

Also msgget, semget, and shmget should fail if unregconized flags are
passed in. That ipcget doesn't do that today is bizarre.

Eric

Subject: Re: [PATCH v6 02/10] ipc: "use key as id" functionality for resource get system ca
Posted by Stanislav Kinsbursky on Tue, 16 Oct 2012 07:52:01 GMT
View Forum Message <> Reply to Message

> Stanislav Kinsbursky <skinsbursky@parallels.com> writes:
>
>> 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.
>
> This way of handling things is pretty nasty.
>
> - You don't fail if the requested id is not available.

It's a part of design. Consider IPC_PRESET as an advice.
It's up to user to check, does returned id suits it's needs.

> - You don't allow assigning the key (which leads to the need to change
>   the key in later patches). Changing the creator uid and creator
>   gid and key is semantically ugly.
>

Key assigning ability is implemented in 4-6'th patches of the series.
Changing creator uid and creator gid with key can be dropped. The reason why I added this was to give more abilities to syscall caller.

> It would be much cleaner if you could instead add IPC_PRESET and then
> extend the definition of the creation functions all by one argument.
>
> aka
> int msgget(key_t key, int msgflg, int id);
> int semget(key_t key, int nsems, int semflg, int id);
> int shmget(key_t key, size_t size, int shmflg, int id);
>
> Where the extra id argument is ignored unless IPC_PRESET is specified.
>

This way suits my needs. The reason, why I didn't this that way was trying to affect user-space API as less, as possible.
So, if there will be more votes for adding one more argument to existent systems calls - I'll do it.

> Also msgget, semget, and shmget should fail if unregconized flags are
> passed in. That ipcget doesn't do that today is bizarre.
>

This looks like another issue, which can be fixed separately.

> Eric
>

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH v6 02/10] ipc: "use key as id" functionality for resource get system ca
Posted by Stanislav Kinsbursky on Tue, 16 Oct 2012 08:34:52 GMT
View Forum Message <> Reply to Message

> Changing the creator uid and creator
>   gid and key is semantically ugly.

BTW, changing the creator uid and creator gid is done already by IPC_SET call.
So doing it in ipc_update_key() is redundant. I'll simplify.

--
Best regards,
Stanislav Kinsbursky