Subject: [PATCH v6 00/10] IPC: checkpoint/restore in userspace enhancements
Posted by Stanislav Kinsbursky on Mon, 15 Oct 2012 15:59:27 GMT
View Forum Message <> Reply to Message

v6:
1) rebased on 3.7-rc1

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/msg.h                          |    5 -
include/uapi/linux/ipc.h                     |    1
include/uapi/linux/msg.h                     |    2
include/uapi/linux/sem.h                     |    1
include/uapi/linux/shm.h                     |    1
ipc/compat.c                                 |   45 +++--
ipc/msg.c                                    |  116 +++++++++---
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 ++++++++++++++++++++++++++
18 files changed, 548 insertions(+), 72 deletions(-)
create mode 100644 tools/testing/selftests/ipc/Makefile
create mode 100644 tools/testing/selftests/ipc/msgque.c
```

Subject: [PATCH v6 08/10] IPC: message queue receive cleanup
Posted by Stanislav Kinsbursky on Mon, 15 Oct 2012 16:00:17 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>

```
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 7a4b9e9..f38edba 100644
--- a/include/linux/msg.h
+++ b/include/linux/msg.h
@@ -34,7 +34,8 @@ 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 /* _LINUX_MSG_H */
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 68515dc..9808da8 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -766,15 +766,30 @@ 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;
@@ -815,7 +830,7 @@ 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;
    }
@@ -842,7 +857,7 @@ 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);
@@ -905,29 +920,16 @@ 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 v6 09/10] IPC: message queue copy feature introduced
Posted by Stanislav Kinsbursky on Mon, 15 Oct 2012 16:00:22 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. 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/uapi/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/uapi/linux/msg.h b/include/uapi/linux/msg.h
index 76999c9..c1af84a 100644
--- a/include/uapi/linux/msg.h
+++ b/include/uapi/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 9808da8..6f52c6b 100644
--- a/ipc/msg.c
+++ b/ipc/msg.c
@@ -788,19 +788,48 @@ 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))
@@ -820,8 +849,16 @@ long do_msgrcv(int msqid, void __user *buf, size_t bufsz, long msgtyp,
     if (mode == SEARCH_LESSEQUAL &&
       walk_msg->m_type != 1) {
      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
      break;
+    msg_counter++;
    }
    tmp = tmp->next;
   }
@@ -834,6 +871,10 @@ 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();
@@ -917,8 +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 271bded..027f507 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 v6 10/10] test: IPC message queue copy feture test
Posted by Stanislav Kinsbursky on Mon, 15 Oct 2012 16:00:27 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 v6 10/10] test: IPC message queue copy feture test
Posted by David Howells on Mon, 15 Oct 2012 19:23:26 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> wrote:

> +CFLAGS += -I../../../../arch/x86/include/generated/
> +CFLAGS += -I../../../../include/
> +CFLAGS += -I../../../../usr/include/
> +CFLAGS += -I../../../../arch/x86/include/

Do you need to add uapi/ into some of these?

David

Subject: Re: [PATCH v6 10/10] test: IPC message queue copy feture test
Posted by Stanislav Kinsbursky on Tue, 16 Oct 2012 07:57:31 GMT
View Forum Message <> Reply to Message

> Stanislav Kinsbursky <skinsbursky@parallels.com> wrote:
>
>> +CFLAGS += -I../../../../arch/x86/include/generated/
>> +CFLAGS += -I../../../../include/
>> +CFLAGS += -I../../../../usr/include/
>> +CFLAGS += -I../../../../arch/x86/include/
>
> Do you need to add uapi/ into some of these?
>

Yes, most probably.
Thanks for notice.

> David
>

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH v6 09/10] IPC: message queue copy feature introduced
Posted by Serge E. Hallyn on Tue, 23 Oct 2012 16:39:48 GMT
View Forum Message <> Reply to Message

Quoting 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. 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.

How much could it clean things up if a new ipc/cr.c contained

```
#ifdef CONFIG_CHECKPOINT_RESTORE
struct msg_msg *copy_msg(struct msg_msg *src, struct msg_msg *dst)
{
...
}
#else
struct msg_msg *copy_msg(struct msg_msg *src, struct msg_msg *dst)
{
 return -ENOSYS;
}
#endif
```

and you went on from there to try to remove all the new #ifdefs from the
existing files?

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

```
> ---
>  include/uapi/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/uapi/linux/msg.h b/include/uapi/linux/msg.h
> index 76999c9..c1af84a 100644
> --- a/include/uapi/linux/msg.h
> +++ b/include/uapi/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 9808da8..6f52c6b 100644
> --- a/ipc/msg.c
> +++ b/ipc/msg.c
> @@ -788,19 +788,48 @@ 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))
> @@ -820,8 +849,16 @@ long do_msgrcv(int msqid, void __user *buf, size_t bufsz, long
msgtyp,
>      if (mode == SEARCH_LESSEQUAL &&
>        walk_msg->m_type != 1) {
>       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
>       break;
> +    msg_counter++;
>     }
>     tmp = tmp->next;
>    }
> @@ -834,6 +871,10 @@ 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();
> @@ -917,8 +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 271bded..027f507 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 *);
>
> --
> To unsubscribe from this list: send the line "unsubscribe linux-security-module" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at  http://vger.kernel.org/majordomo-info.html
```