
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 whether 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;
+}
```
