Subject: [patch 0/5][RFC - ipv4/udp checkpoint/restart] dumping/restoring the IPV4/UDP sockets
Posted by Daniel Lezcano on Wed, 06 Jun 2007 12:18:06 GMT
View Forum Message <> Reply to Message

Hi,

I would like to resurect the discussion we had concerning the socket checkpoint/restart.

I began to look how to checkpoint them. I thought the following:

The socket can be checkpointed one by one from userspace.
That will allow to provide a mechanism to application which wants to checkpoint himself (I think for example at the tcpcp project) and increase the scope of usability.

If we are inside a container, only the sockets related to the container are viewable and by the way, checkpointable, so the socket checkpoint/restart will bring mobility to the container.

The socket relies to the socket fs. We can use the inode number to identify the socket. Using that, we can checkpoint/restart a socket associated with a fd
because inode number can be easily retrieved from a fstat call and we can identify orphan sockets. Inode number of orphan sockets are viewable in the /proc/net/tcp file.

The checkpoint/restart data can be transfered between kernel and userspace via the generic netlink. It is a clean and secure way to define a message format with ascendant compatibily, ie : move the socket to an os with a superior kernel version. The generic netlink message can be either used as raw data to be directly dumped to disk or can be modified from userspace for some specific purpose (I dont have examples)

The way the socket are checkpointed/restored is to stick as much as possible to user/kernel frontier in order to catch errors and bad values the sooner.
I think for example, all the kernel_setsockopt, kernel_connect, etc ...
function family, it is more reliable and secure.

The following patchset is a RFC for C/R the UDP sockets. It applies to 2.6.20.

One question is pending. Should we dump/restore send and receive queue knowing the protocol is not reliable ?

Regards.

-- Daniel
--

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [patch 1/5][RFC - ipv4/udp checkpoint/restart] : add lookup for unhashed inode
Posted by Daniel Lezcano on Wed, 06 Jun 2007 12:18:07 GMT
View Forum Message <> Reply to Message

The socket relies on the sockfs. In some cases, the socket are orphans and
it is not possible to access them via a file descriptor, this is the case for
example for timewait sockets. Hopefully, an inode is still usable to specify
a socket. This one can be retrieved from /proc/net/tcp for orphan sockets or
from a fstat.

When a socket is created the socket inode is added to the sockfs.
Unfortunatly, this one is not stored into the hashed inode list, so
I need a helper to browse the inode list contained in the superblock
of the sockfs.

This is one solution, another solution is to stored the inode into
the hashed list when socket is created.

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 fs/inode.c | 29 ++++++++++++++++++++++++++++
 include/linux/fs.h | 1 +
 2 files changed, 30 insertions(+)

Index: 2.6.20-cr/fs/inode.c
===
--- 2.6.20-cr.orig/fs/inode.c
+++ 2.6.20-cr/fs/inode.c
@@ -877,6 +877,35 @@

 EXPORT_SYMBOL(ilookup);

+
+/**
+ * ilookup_unhased - search for an inode in the superblock
+ * @sb: super block of file system to search
+ * @ino: inode number to search for
+ *
+ * The ilookup_unhashed browse the superblock inode list to find the inode.
+ *
+ * If the inode is found in the inode list stored in the superblock, the inode is

```
+ * with an incremented reference count.
+ *
+ * Otherwise NULL is returned.
+ */
+struct inode *ilookup_unhashed(struct super_block *sb, unsigned long ino)
+{
+ struct inode *inode = NULL;
+
+ spin_lock(&inode_lock);
+ list_for_each_entry(inode, &sb->s_inodes, i_sb_list)
+  if (inode->i_ino == ino) {
+   __iget(inode);
+   break;
+  }
+ spin_unlock(&inode_lock);
+ return inode;
+
+}
+EXPORT_SYMBOL(ilookup_unhashed);
+
 /**
  * iget5_locked - obtain an inode from a mounted file system
  * @sb:  super block of file system
Index: 2.6.20-cr/include/linux/fs.h
===================================================================
--- 2.6.20-cr.orig/include/linux/fs.h
+++ 2.6.20-cr/include/linux/fs.h
@@ -1657,6 +1657,7 @@
 extern struct inode *ilookup5(struct super_block *sb, unsigned long hashval,
   int (*test)(struct inode *, void *), void *data);
 extern struct inode *ilookup(struct super_block *sb, unsigned long ino);
+extern struct inode *ilookup_unhashed(struct super_block *sb, unsigned long ino);

 extern struct inode * iget5_locked(struct super_block *, unsigned long, int (*test)(struct inode *,
void *), int (*set)(struct inode *, void *), void *);
 extern struct inode * iget_locked(struct super_block *, unsigned long);

--
```


_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [patch 2/5][RFC - ipv4/udp checkpoint/restart] : provide compilation option and genetlink framework
Posted by Daniel Lezcano on Wed, 06 Jun 2007 12:18:08 GMT

From: Daniel Lezcano <dlezcano@fr.ibm.com>

This patchset provide the AF_INET C/R option in the makefile and a generic
netlink framework for passing the socket messages.

It seems that we are encouraged to use netlink instead of the /proc, /sysfs, ioctls:
http://kerneltrap.org/node/6637

I found that there is a lot of advantages to use the netlink:
 * the protocol is secure
 * the protocol will describe the socket attributes and obviously brings a little
   abstraction layer with the internal kernel structure/data type. This is a good
   way to implement ascendant compatibility (eg. move a socket to a kernel with an upper
   version)
 * the amount of socket informations is variable. With netlink, we don't need to take
   care of the size of the data (eg. just read data until there is something)
 * the netlink attributes can be direclty dumped to disk and reused to restore the socket
   or examined for specific processing (I don't have example).

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 include/linux/af_inet_cr.h | 15 +++++
 net/ipv4/Kconfig | 8 +++
 net/ipv4/Makefile | 1
 net/ipv4/af_inet_cr.c | 119 ++
 4 files changed, 143 insertions(+)

Index: 2.6.20-cr/net/ipv4/Kconfig
===
--- 2.6.20-cr.orig/net/ipv4/Kconfig
+++ 2.6.20-cr/net/ipv4/Kconfig
@@ -1,6 +1,14 @@
 #
 # IP configuration
 #
+config IP_CR
+	tristate "IP: checkpoint/restart"
+	help
+	 The checkpoint/restart allows to dump the sockets states and
+	 the associated protocols internals to the userspace land.
+	 The data can be reused to recreate the socket in the same state.
+	 It's safe to say N.
+
 config IP_MULTICAST
  bool "IP: multicasting"
  help
Index: 2.6.20-cr/net/ipv4/Makefile

```
===================================================================
--- 2.6.20-cr.orig/net/ipv4/Makefile
+++ 2.6.20-cr/net/ipv4/Makefile
@@ -53,3 +53,4 @@

 obj-$(CONFIG_XFRM) += xfrm4_policy.o xfrm4_state.o xfrm4_input.o \
        xfrm4_output.o
+obj-$(CONFIG_IP_CR) += af_inet_cr.o
Index: 2.6.20-cr/net/ipv4/af_inet_cr.c
===================================================================
--- /dev/null
+++ 2.6.20-cr/net/ipv4/af_inet_cr.c
@@ -0,0 +1,119 @@
+/*
+ *  Copyright (C) 2007 IBM Corporation
+ *
+ *  Author: Daniel Lezcano <dlezcano@fr.ibm.com>
+ *
+ *  This program is free software; you can redistribute it and/or
+ *  modify it under the terms of the GNU General Public License as
+ *  published by the Free Software Foundation, version 2 of the
+ *  License.
+ */
+
+#include <net/genetlink.h>
+#include <net/sock.h>
+#include <linux/fs.h>
+#include <linux/af_inet_cr.h>
+
+/*
+ * af_inet_cr_nldump : this function is called when a netlink message is received
+ * with AF_INET_CR_CMD_DUMP command.
+ * @skb  : the netlink packet giving the restore command
+ * @info : the generic netlink message
+ */
+static int af_inet_cr_nldump(struct sk_buff *skb, struct genl_info *info)
+{
+ return 0;
+}
+
+/*
+ * af_inet_cr_nldump : this function is called when a netlink message is received
+ * with AF_INET_CR_CMD_RESTORE command.
+ * @skb  : the netlink packet giving the restore command
+ * @info : the generic netlink message
+ */
+static int af_inet_cr_nlrestore(struct sk_buff *skb, struct genl_info *info)
+{
```

```
+ return 0;
+}
+
+/*
+ * Netlink message policy definition
+ */
+static struct nla_policy af_inet_cr_policy[AF_INET_CR_ATTR_MAX] = {
+ [AF_INET_CR_ATTR_INODE] = { .type = NLA_U32 },
+};
+
+/*
+ * Netlink dumping command configuration
+ */
+static struct genl_ops af_inet_cr_nldump_ops = {
+ .cmd = AF_INET_CR_CMD_DUMP,
+ .doit = af_inet_cr_nldump,
+ .policy = af_inet_cr_policy,
+};
+
+/*
+ * Netlink restore command configuration
+ */
+static struct genl_ops af_inet_cr_nlrestore_ops = {
+ .cmd = AF_INET_CR_CMD_RESTORE,
+ .doit = af_inet_cr_nlrestore,
+ .policy = af_inet_cr_policy,
+};
+
+/*
+ * Generic netlink family definition
+ */
+static struct genl_family af_inet_cr_family = {
+ .id             = GENL_ID_GENERATE,
+ .name           = "af_inet_cr",
+ .version        = 0x1,
+ .maxattr        = AF_INET_CR_ATTR_MAX - 1,
+};
+
+/*
+ * af_inet_cr_init : this function is called at initialization
+ * time. It register the generic netlink family associated with
+ * this module and hang different ops with it.
+ */
+static __init int af_inet_cr_init(void)
+{
+ int err;
+
+ err = genl_register_family(&af_inet_cr_family);
```

```
+ if (err < 0)
+  goto out;
+
+ err = genl_register_ops(&af_inet_cr_family,
+    &af_inet_cr_nldump_ops);
+ if (err < 0)
+  goto out_unregister_fam;
+
+ err = genl_register_ops(&af_inet_cr_family,
+    &af_inet_cr_nlrestore_ops);
+ if (err < 0)
+  goto out_unregister_dump;
+
+ return 0;
+
+out_unregister_dump:
+ genl_unregister_ops(&af_inet_cr_family, &af_inet_cr_nldump_ops);
+out_unregister_fam:
+ genl_unregister_family(&af_inet_cr_family);
+out:
+ return err;
+}
+
+/*
+ * af_inet_cr_exit : this function is called at exit time. It unregister
+ * the generic netlink family.
+ */
+static __exit void af_inet_cr_exit(void)
+{
+ genl_unregister_ops(&af_inet_cr_family, &af_inet_cr_nlrestore_ops);
+ genl_unregister_ops(&af_inet_cr_family, &af_inet_cr_nldump_ops);
+ genl_unregister_family(&af_inet_cr_family);
+}
+
+module_init(af_inet_cr_init);
+module_exit(af_inet_cr_exit);
Index: 2.6.20-cr/include/linux/af_inet_cr.h
===================================================================
--- /dev/null
+++ 2.6.20-cr/include/linux/af_inet_cr.h
@@ -0,0 +1,15 @@
+#ifndef _AF_INET_CR_H
+#define _AF_INET_CR_H
+enum {
+ AF_INET_CR_CMD_UNSPEC,
+ AF_INET_CR_CMD_DUMP,
+ AF_INET_CR_CMD_RESTORE,
+ AF_INET_CR_CMD_MAX
```

```
+};
+
+enum {
+ AF_INET_CR_ATTR_UNSPEC,
+ AF_INET_CR_ATTR_INODE,
+ AF_INET_CR_ATTR_MAX
+};
+#endif

--
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [patch 3/5][RFC - ipv4/udp checkpoint/restart] : c/r the socket information and options

Posted by Daniel Lezcano on Wed, 06 Jun 2007 12:18:09 GMT

View Forum Message <> Reply to Message

From: Daniel Lezcano <dlezcano@fr.ibm.com>

This patch defines a set of netlink attributes to store/retrieve socket options.

* At dump time, a netlink message specify the inode of the socket to be checkpointed. The socket is retrieved with the inode number. A new netlink message is built in order to store the socket information. The type, state and socket options are stored into it and the netlink message is transmitted to the requestor.

* At restore time, the netlink message contains the type of the socket. A new socket is created, using this type and the attributes are browsed in order to use the values to restore the differents options.

The choice of the C/R is to stick as much as possible to the user/kernel frontier. For this reason, the kernel_{set,get}sockopt are used. That allows to reduce code and delegate the different checks to the corresponding function. Unfortunatly, some get/set are not symetric, so some options can be retrieved but not set and vice-versa. For this reason, there are a few helpers, and the option definitions contains a GET|SET|BOTH flag.

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>

```
---
 include/linux/af_inet_cr.h |   61 ++++
 net/ipv4/af_inet_cr.c      |  640 ++++++++++++++++++++++++++++++++++++++++++++--
```

2 files changed, 680 insertions(+), 21 deletions(-)

Index: 2.6.20-cr/net/ipv4/af_inet_cr.c
===
--- 2.6.20-cr.orig/net/ipv4/af_inet_cr.c
+++ 2.6.20-cr/net/ipv4/af_inet_cr.c
@@ -12,36 +12,644 @@
 #include <net/genetlink.h>
 #include <net/sock.h>
 #include <linux/fs.h>
+#include <linux/syscalls.h>
 #include <linux/af_inet_cr.h>

 /*
- * af_inet_cr_nldump : this function is called when a netlink message is received
- * with AF_INET_CR_CMD_DUMP command.
+ * Netlink message policy definition
+ */
+static struct nla_policy af_inet_cr_policy[AF_INET_CR_ATTR_MAX] = {
+ [AF_INET_CR_ATTR_INODE]   = { .type = NLA_U32 },
+
+ [AF_INET_CR_ATTR_SOCK_STATE]   = { .type = NLA_U32 },
+ [AF_INET_CR_ATTR_SOCK_TYPE]   = { .type = NLA_U32 },
+
+ [AF_INET_CR_ATTR_SOCKOPT_BROADCAST]   = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_SOCKOPT_DEBUG]   = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_SOCKOPT_DONTROUTE]   = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_SOCKOPT_KEEPALIVE]   = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_SOCKOPT_OOBINLINE]   = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_SOCKOPT_PASSCRED]   = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_SOCKOPT_REUSEADDR]   = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_SOCKOPT_TIMESTAMP]   = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_SOCKOPT_SNDBUF_ULOCK] = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_SOCKOPT_RCVBUF_ULOCK] = { .type = NLA_FLAG },
+
+ [AF_INET_CR_ATTR_SOCKOPT_RCVBUF]   = { .type = NLA_U32 },
+ [AF_INET_CR_ATTR_SOCKOPT_SNDBUF]   = { .type = NLA_U32 },
+ [AF_INET_CR_ATTR_SOCKOPT_PRIORITY]   = { .type = NLA_U32 },
+ [AF_INET_CR_ATTR_SOCKOPT_RCVLOWAT]   = { .type = NLA_U32 },
+
+ [AF_INET_CR_ATTR_SOCKOPT_RCVTIMEO] = { .len = sizeof(struct timeval) },
+ [AF_INET_CR_ATTR_SOCKOPT_SNDTIMEO] = { .len = sizeof(struct timeval) },
+ [AF_INET_CR_ATTR_SOCKOPT_LINGER]   = { .len = sizeof(struct linger) },
+ [AF_INET_CR_ATTR_SOCKOPT_BINDTODEVICE] = { .len = IFNAMSIZ },
+};
+
+/*
+ * Generic netlink family definition

```
+ */
+static struct genl_family af_inet_cr_family = {
+ .id             = GENL_ID_GENERATE,
+ .name           = "af_inet_cr",
+ .version        = 0x1,
+ .maxattr        = AF_INET_CR_ATTR_MAX - 1,
+};
+
+/*
+ * socket options association with netlink attribute
+ */
+struct af_inet_cr_optattr socket_options[] = {
+ { SO_BROADCAST,    AF_INET_CR_ATTR_SOCKOPT_BROADCAST,    0, BOTH },
+ { SO_DEBUG,        AF_INET_CR_ATTR_SOCKOPT_DEBUG,        0, BOTH },
+ { SO_DONTROUTE,    AF_INET_CR_ATTR_SOCKOPT_DONTROUTE,    0, BOTH },
+ { SO_KEEPALIVE,    AF_INET_CR_ATTR_SOCKOPT_KEEPALIVE,    0, BOTH },
+ { SO_OOBINLINE,    AF_INET_CR_ATTR_SOCKOPT_OOBINLINE,    0, BOTH },
+ { SO_PRIORITY,     AF_INET_CR_ATTR_SOCKOPT_PRIORITY,     0, BOTH },
+ { SO_RCVLOWAT,     AF_INET_CR_ATTR_SOCKOPT_RCVLOWAT,     0, BOTH },
+ { SO_RCVBUF,       AF_INET_CR_ATTR_SOCKOPT_RCVBUF,       0, GET  },
+ { SO_SNDBUF,       AF_INET_CR_ATTR_SOCKOPT_SNDBUF,       0, GET  },
+ { SO_REUSEADDR,    AF_INET_CR_ATTR_SOCKOPT_REUSEADDR,    0, BOTH },
+ { SO_TIMESTAMP,    AF_INET_CR_ATTR_SOCKOPT_TIMESTAMP,    0, BOTH },
+ { SO_LINGER,       AF_INET_CR_ATTR_SOCKOPT_LINGER,       0, BOTH },
+ { SO_RCVTIMEO,     AF_INET_CR_ATTR_SOCKOPT_RCVTIMEO,     0, BOTH },
+ { SO_SNDTIMEO,     AF_INET_CR_ATTR_SOCKOPT_SNDTIMEO,     0, BOTH },
+ { SO_BINDTODEVICE, AF_INET_CR_ATTR_SOCKOPT_BINDTODEVICE, 0, SET  },
+};
+
+
+/*
+ * socket_lookup : search for socket using the inode number
+ *
+ * @sb : superblock associated to the sockfs
+ * @ino : the inode number associated with the socket
+ * @sock : the socket resulting from the lookup
+ *
+ * Returns 0 on succes or if the call fails:
+ *  -ENOENT : inode is not found
+ *  -ENOTSOCK: the inode found is not associated with a socket
+ *  -EINVAL: unexpected error
+ */
+static inline int socket_lookup(struct super_block *sb, unsigned long ino,
+     struct socket **sock)
+{
+ int ret;
+  struct inode *inode;
+
```

```
+  inode = ilookup_unhashed(sb, ino);
+  if (!inode)
+  return -ENOENT;
+
+ ret = -ENOTSOCK;
+  if (!S_ISSOCK(inode->i_mode))
+  goto out;
+
+ ret = -EINVAL;
+  *sock = SOCKET_I(inode);
+  if (!*sock)
+  goto out;
+
+ ret = 0;
+out:
+ iput(inode);
+  return ret;
+}
+
+/*
+ *  af_inet_cr_opt2attr: convert a socket option to a netlink attribute and push it
+ *  to the skbuff
+ *
+ * @skb : the skbuff to be filled
+ * @sock : the socket to retrieve options
+ * @optlevel : the level of the option
+ * @optattr : the correpondance between the option and the attribute
+ *
+ * Return 0 on sucess, < 0 otherwise
+ */
+int af_inet_cr_opt2attr(struct sk_buff *skb, const struct socket *sock, int optlevel,
+   const struct af_inet_cr_optattr *optattr)
+{
+ int attr = optattr->attr;
+ int optname = optattr->optname;
+ int type = af_inet_cr_policy[attr].type;
+ int optlen;
+ char *optbuf = NULL;
+ int optval;
+ int ret;
+
+ if (!(optattr->get_and_set & GET))
+                return 0;
+
+ switch (type) {
+ case NLA_UNSPEC:
+  optlen = af_inet_cr_policy[attr].len;
+  optbuf = kmalloc(optlen, GFP_KERNEL);
```

```
+  if (!optbuf)
+   return -ENOMEM;
+  ret = kernel_getsockopt((struct socket *)sock, optlevel,
+      optname, optbuf, &optlen);
+  if (ret)
+   goto out;
+  ret = nla_put(skb, attr, af_inet_cr_policy[attr].len, optbuf);
+  goto out;
+ case NLA_U32:
+  optlen = sizeof(optval);
+  ret = kernel_getsockopt((struct socket *)sock, optlevel,
+      optname, (void *)&optval, &optlen);
+  if (ret)
+   goto out;
+  ret = nla_put_u32(skb, attr, optval);
+  goto out;
+ case NLA_U8:
+  optlen = sizeof(optval);
+  ret = kernel_getsockopt((struct socket *)sock, optlevel,
+      optname, (void *)&optval, &optlen);
+  if (ret)
+   goto out;
+  ret = nla_put_u8(skb, attr, optval);
+  goto out;
+ case NLA_FLAG:
+  optlen = sizeof(optval);
+  ret = kernel_getsockopt((struct socket *)sock, optlevel,
+      optname, (void *)&optval, &optlen);
+  if (ret)
+   goto out;
+  if (optval)
+   ret = nla_put_flag(skb, attr);
+  goto out;
+ default:
+  ret = -EINVAL;
+               goto out;
+ };
+ out:
+ if (optbuf)
+  kfree(optbuf);
+        return ret;
+}
+
+/*
+ * af_inet_cr_opt2attr : convert a netlink attribute to a socket option
+ *  and set the option to the socket
+ *
+ * @info : the generic netlink message
```

```
+ * @sock : the socket to set options
+ * @optlevel : the level of the option
+ * @optattr : the correpondance between the option and the attribute
+ *
+ * Return 0 on sucess, < 0 otherwise
+ */
+int af_inet_cr_attr2opt(const struct genl_info *info, struct socket *sock,
+   int optlevel, const struct af_inet_cr_optattr *optattr)
+{
+       int optname = optattr->optname;
+       int attr = optattr->attr;
+ int ret, type = af_inet_cr_policy[attr].type;
+ struct nlattr *nla = info->attrs[attr];
+ char *optbuf;
+ int optlen;
+ int optval;
+
+ if (!(optattr->get_and_set & SET))
+  return 0;
+ if (!nla)
+               return 0;
+
+ switch (type) {
+       case NLA_FLAG:
+  optval = nla_get_flag(nla);
+  break;
+ case NLA_U8:
+  optval = nla_get_u8(nla);
+  break;
+       case NLA_U32:
+  optval = nla_get_u32(nla);
+  break;
+ case NLA_UNSPEC:
+  optlen = af_inet_cr_policy[attr].len;
+  optbuf = kmalloc(optlen, GFP_KERNEL);
+  if (!optbuf)
+   return -ENOMEM;
+  nla_memcpy(optbuf, nla, optlen);
+  ret = kernel_setsockopt(sock, optlevel, optname,
+     optbuf, optlen);
+  kfree(optbuf);
+  goto out;
+ default:
+  ret = -EINVAL;
+  goto out;
+       }
+
+ optlen = sizeof(optval);
```

```
+ ret = kernel_setsockopt(sock, optlevel, optname,
+     (void *)&optval, optlen);
+out:
+ return ret;
+}
+
+/*
+ * dump_sockopt_sndbuf : retrieve the userlock flag on the sndbuf option and
+ *  add it to the netlink message
+ *
+ * @sock : the socket to retrieve the userlock
+ * @skb : the skbuff containing the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static inline int dump_sockopt_sndbuf(struct socket *sock,
+          struct sk_buff *skb)
+{
+ int ret;
+ struct sock *sk = sock->sk;
+
+ if (sk->sk_userlocks & SOCK_SNDBUF_LOCK) {
+  ret = nla_put_flag(skb, AF_INET_CR_ATTR_SOCKOPT_SNDBUF_ULOCK);
+  if (ret)
+   return ret;
+ }
+
+ return 0;
+}
+
+/*
+ * dump_sockopt_rcvbuf : retrieve the userlock flag on the rcv option and
+ *  add it to the netlink message
+ *
+ * @sock : the socket to retrieve the userlock
+ * @skb : the skbuff containing the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static inline int dump_sockopt_rcvbuf(struct socket *sock,
+          struct sk_buff *skb)
+{
+ int ret;
+ struct sock *sk = sock->sk;
+
+ if (sk->sk_userlocks & SOCK_RCVBUF_LOCK) {
+  ret = nla_put_flag(skb, AF_INET_CR_ATTR_SOCKOPT_RCVBUF_ULOCK);
+  if (ret)
```

```
+   return ret;
+ }
+
+ return 0;
+}
+
+/*
+ * dump_sockopt_bindtodevice : retrieve the bind device option and add it
+ *  to the netlink message. Note that is the name of the device which is
+ *  dumped, not the index
+ *
+ * @sock : the socket to retrieve the bindtodevice option
+ * @skb : the skbuff containing the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static inline int dump_sockopt_bindtodevice(struct socket *sock,
+         struct sk_buff *skb)
+{
+ int ret;
+ struct sock *sk = sock->sk;
+ struct net_device *dev = dev_get_by_index(sk->sk_bound_dev_if);
+
+ if (!sk->sk_bound_dev_if)
+  return 0;
+
+ dev = dev_get_by_index(sk->sk_bound_dev_if);
+ if (!dev)
+  return -EINVAL;
+
+ ret = nla_put(skb, AF_INET_CR_ATTR_SOCKOPT_BINDTODEVICE,
+        IFNAMSIZ, dev->name);
+ dev_put(dev);
+
+ return ret;
+}
+
+/*
+ * dump_sockopt_sockopt : retrieve the socket options and store them
+ *  to the netlink message
+ *
+ * @sock : the socket to retrieve the bindtodevice option
+ * @skb : the skbuff containing the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static int dump_sockopt(struct socket *sock, struct sk_buff *skb)
+{
```

```
+ size_t i, len;
+ int ret;
+
+ len = sizeof(socket_options)/sizeof(struct af_inet_cr_optattr);
+
+ for (i = 0; i < len; i++) {
+  ret = af_inet_cr_opt2attr(skb, sock, SOL_SOCKET, &socket_options[i]);
+  if (ret)
+   return ret;
+ }
+
+ ret = dump_sockopt_bindtodevice(sock, skb);
+ if (ret)
+  return ret;
+
+ ret = dump_sockopt_sndbuf(sock, skb);
+ if (ret)
+  return ret;
+
+ ret = dump_sockopt_rcvbuf(sock, skb);
+ if (ret)
+  return ret;
+
+ return 0;
+}
+
+/*
+ * dump_socket : dump the socket state, type and options into a netlink message
+ *  and transmit the message to the sender of the dump command.
+ *
+ * @sock : socket to be dumped
+ * @pid : pid of the sender
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static int dump_socket(struct socket *sock, pid_t pid)
+{
+ struct sock *sk = sock->sk;
+ unsigned short family = sk->sk_family;
+ struct sk_buff *skb;
+ void *msg_head;
+ int ret;
+
+ if (family != AF_INET)
+  return -EINVAL;
+
+ skb = genlmsg_new(NLMSG_GOODSIZE, GFP_KERNEL);
+ if (!skb)
```

```
+  return -ENOMEM;
+
+ msg_head = genlmsg_put(skb, pid, 0, &af_inet_cr_family, 0,
+          AF_INET_CR_CMD_DUMP);
+ ret = -ENOMEM;
+ if (!msg_head)
+  goto out;
+
+ ret = nla_put_u32(skb, AF_INET_CR_ATTR_SOCK_STATE, sock->state);
+ if (ret)
+  goto out;
+
+ ret = nla_put_u32(skb, AF_INET_CR_ATTR_SOCK_TYPE, sock->type);
+ if (ret)
+  goto out;
+
+ ret = dump_sockopt(sock, skb);
+ if (ret)
+  goto out;
+
+ ret = genlmsg_end(skb, msg_head);
+ if (ret < 0)
+  goto out;
+
+ ret = genlmsg_unicast(skb, pid);
+out:
+ if (ret)
+  nlmsg_free(skb);
+ return ret;
+}
+
+/*
+ * restore_sockopt_rcvbuf : extract rcvbuf value from the netlink
+ *  message and restore the socket rcvbuf option. Only of the value
+ *  is specified because the kernel multiplicate value by two, the
+ *  userlock flag is overwritten
+ *
+ * @sock : the socket to be restored
+ * @info : the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static inline int restore_sockopt_rcvbuf(struct socket *sock,
+      struct genl_info *info)
+{
+ struct nlattr *nla;
+ int val, ret;
+
```

```
+ nla = info->attrs[AF_INET_CR_ATTR_SOCKOPT_RCVBUF];
+ if (!nla)
+  return -EINVAL;
+
+ val = nla_get_u32(nla) / 2;
+
+ ret = kernel_setsockopt(sock, SOL_SOCKET, SO_RCVBUFFORCE,
+     (char *)&val, sizeof(val));
+ if (ret)
+  return ret;
+
+ if (!info->attrs[AF_INET_CR_ATTR_SOCKOPT_RCVBUF_ULOCK])
+  sock->sk->sk_userlocks &= ~SOCK_RCVBUF_LOCK;
+
+ return 0;
+}
+
+/*
+ * restore_sockopt_sndbuf : extract sndbuf value from the netlink
+ *  message and restore the socket sndbuf option. Only of the value
+ *  is specified because the kernel multiplicate value by two, the
+ *  userlock flag is overwritten
+ *
+ * @sock : the socket to be restored
+ * @info : the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static inline int restore_sockopt_sndbuf(struct socket *sock,
+       struct genl_info *info)
+{
+ struct nlattr *nla;
+ int val, ret;
+
+ nla = info->attrs[AF_INET_CR_ATTR_SOCKOPT_SNDBUF];
+ if (!nla)
+  return -EINVAL;
+
+ val = nla_get_u32(nla) / 2;
+
+ ret = kernel_setsockopt(sock, SOL_SOCKET, SO_SNDBUFFORCE,
+      (char *)&val, sizeof(val));
+ if (ret)
+  return ret;
+
+ if (!info->attrs[AF_INET_CR_ATTR_SOCKOPT_SNDBUF_ULOCK])
+  sock->sk->sk_userlocks &= ~SOCK_SNDBUF_LOCK;
+
```

```
+ return ret;
+}
+
+/*
+ * restore_sockopt_sndbuf : restore the socket option using the correponding
+ *  netlink attribute <-> socket option array
+ *
+ * @sock : the socket to be restored
+ * @info : the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static inline int restore_sockopt(struct socket *sock, struct genl_info *info)
+{
+ size_t i, len;
+ int ret;
+
+ len = sizeof(socket_options)/sizeof(struct af_inet_cr_optattr);
+
+ for (i = 0; i < len; i++) {
+  ret = af_inet_cr_attr2opt(info, sock, SOL_SOCKET, &socket_options[i]);
+  if (ret)
+   return ret;
+ }
+
+ ret = restore_sockopt_sndbuf(sock, info);
+ if (ret)
+  return ret;
+
+ ret = restore_sockopt_rcvbuf(sock, info);
+ if (ret)
+  return ret;
+
+ return 0;
+}
+
+/*
+ * restore_socket : restore the socket from the netlink message content
+ *
+ * @sock : the socket to be restored
+ * @info : the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static int restore_socket(struct socket *sock, struct genl_info *info)
+{
+ int ret;
+
```

```
+ struct nlattr *nla;
+
+ ret = -EINVAL;
+
+ nla = info->attrs[AF_INET_CR_ATTR_SOCK_STATE];
+ if (!nla)
+  goto out;
+ sock->state = nla_get_u32(nla);
+
+ nla = info->attrs[AF_INET_CR_ATTR_SOCK_TYPE];
+ if (!nla)
+  goto out;
+ sock->type = nla_get_u32(nla);
+
+ ret = restore_sockopt(sock, info);
+ if (ret)
+  goto out;
+out:
+ return ret;
+}
+
+/*
+ * af_inet_cr_nldump : this function is called when a netlink message is
+ *  received with AF_INET_CR_CMD_DUMP command.
+ *
  * @skb  : the netlink packet giving the restore command
  * @info : the generic netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
  */
 static int af_inet_cr_nldump(struct sk_buff *skb, struct genl_info *info)
 {
- return 0;
+ struct sock *sk = skb->sk;
+ struct socket *sock = sk->sk_socket;
+ struct inode *inode = SOCK_INODE(sock);
+ struct super_block *sb = inode->i_sb;
+ int ret;
+ unsigned long ino;
+ struct nlattr *nla = info->attrs[AF_INET_CR_ATTR_INODE];
+
+ if (!nla)
+  return -EINVAL;
+
+ ino = nla_get_u32(nla);
+
+ ret = socket_lookup(sb, ino, &sock);
+ if (ret)
```

```
+  return ret;
+
+ return dump_socket(sock, info->snd_pid);
 }

 /*
  * af_inet_cr_nldump : this function is called when a netlink message is received
- * with AF_INET_CR_CMD_RESTORE command.
+ *  with AF_INET_CR_CMD_RESTORE command.
  * @skb  : the netlink packet giving the restore command
  * @info : the generic netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
  */
 static int af_inet_cr_nlrestore(struct sk_buff *skb, struct genl_info *info)
 {
+
+ int ret;
+ int type;
+ void *msg_head;
+ struct socket *sock;
+ struct nlattr *nla;
+ struct sk_buff *answer;
+
+ answer = genlmsg_new(NLMSG_GOODSIZE, GFP_KERNEL);
+ if (!answer)
+  return -ENOMEM;
+
+ ret = -ENOMEM;
+ msg_head = genlmsg_put(answer, info->snd_pid, 0, &af_inet_cr_family, 0,
+          AF_INET_CR_CMD_RESTORE);
+ if (!msg_head)
+  goto out;
+
+ ret = -EINVAL;
+ nla = info->attrs[AF_INET_CR_ATTR_SOCK_TYPE];
+ if (!nla)
+  goto out;
+
+ type = nla_get_u32(nla);
+
+ ret = sock_create_kern(AF_INET, type, 0, &sock);
+ if (ret)
+  goto out;
+
+ ret = restore_socket(sock, info);
+ if (ret)
+  goto out_release;
```

```
+
+ ret = sock_map_fd(sock);
+ if (ret < 0)
+  goto out_release;
+
+ ret = nla_put_u32(answer, AF_INET_CR_ATTR_SOCK_FD, ret);
+ if (ret)
+  goto out_close;
+
+ ret = genlmsg_end(answer, msg_head);
+ if (ret < 0)
+  goto out_close;
+
+ ret = genlmsg_unicast(answer, info->snd_pid);
+ if (ret)
+  goto out_close;
+
  return 0;
-}

-/*
- * Netlink message policy definition
- */
-static struct nla_policy af_inet_cr_policy[AF_INET_CR_ATTR_MAX] = {
- [AF_INET_CR_ATTR_INODE] = { .type = NLA_U32 },
-};
+out_close:
+ sys_close(ret);
+out_release:
+ sock_release(sock);
+out:
+ nlmsg_free(answer);
+
+ return ret;
+}

 /*
  * Netlink dumping command configuration
@@ -62,16 +670,6 @@
 };

 /*
- * Generic netlink family definition
- */
-static struct genl_family af_inet_cr_family = {
- .id            = GENL_ID_GENERATE,
- .name          = "af_inet_cr",
- .version       = 0x1,
```

```
- .maxattr       = AF_INET_CR_ATTR_MAX - 1,
-};
-
-/*
  * af_inet_cr_init : this function is called at initialization
  * time. It register the generic netlink family associated with
  * this module and hang different ops with it.
Index: 2.6.20-cr/include/linux/af_inet_cr.h
===================================================================
--- 2.6.20-cr.orig/include/linux/af_inet_cr.h
+++ 2.6.20-cr/include/linux/af_inet_cr.h
@@ -1,5 +1,39 @@
 #ifndef _AF_INET_CR_H
 #define _AF_INET_CR_H
+
+#ifdef __KERNEL__
+
+#include <linux/net.h>
+#include <linux/skbuff.h>
+
+/*
+ * kernel_{get,set}sockopt
+ */
+#define NONE 0
+#define SET  1
+#define GET  2
+#define BOTH (SET|GET)
+
+/*
+ * Structure to associate a socket option with the netlink attribute
+ */
+struct af_inet_cr_optattr {
+ int optname;  /* option name */
+ int attr;     /* netlink attribute */
+ int unconnected; /* option related to unconnected sockets only */
+ int get_and_set; /* has kernel_{set,get}sockopt */
+};
+
+extern int af_inet_cr_opt2attr(struct sk_buff *skb,
+         const struct socket *sock,
+         int optlevel,
+         const struct af_inet_cr_optattr *optattr);
+
+int af_inet_cr_attr2opt(const struct genl_info *info,
+   struct socket *sock,
+   int optlevel,
+   const struct af_inet_cr_optattr *optattr);
+#endif
```

```
 enum {
  AF_INET_CR_CMD_UNSPEC,
  AF_INET_CR_CMD_DUMP,
@@ -10,6 +44,33 @@
 enum {
  AF_INET_CR_ATTR_UNSPEC,
  AF_INET_CR_ATTR_INODE,
+
+       /* socket state */
+ AF_INET_CR_ATTR_SOCK_STATE,
+ AF_INET_CR_ATTR_SOCK_TYPE,
+ AF_INET_CR_ATTR_SOCK_FD,
+
+ /* socket options */
+ AF_INET_CR_ATTR_SOCKOPT_BINDTODEVICE,
+
+ AF_INET_CR_ATTR_SOCKOPT_BROADCAST,
+ AF_INET_CR_ATTR_SOCKOPT_DEBUG,
+ AF_INET_CR_ATTR_SOCKOPT_DONTROUTE,
+ AF_INET_CR_ATTR_SOCKOPT_KEEPALIVE,
+ AF_INET_CR_ATTR_SOCKOPT_LINGER,
+ AF_INET_CR_ATTR_SOCKOPT_OOBINLINE,
+ AF_INET_CR_ATTR_SOCKOPT_PASSCRED,
+ AF_INET_CR_ATTR_SOCKOPT_PRIORITY,
+ AF_INET_CR_ATTR_SOCKOPT_RCVLOWAT,
+ AF_INET_CR_ATTR_SOCKOPT_RCVTIMEO,
+ AF_INET_CR_ATTR_SOCKOPT_SNDTIMEO,
+ AF_INET_CR_ATTR_SOCKOPT_RCVBUF,
+ AF_INET_CR_ATTR_SOCKOPT_REUSEADDR,
+ AF_INET_CR_ATTR_SOCKOPT_SNDBUF,
+ AF_INET_CR_ATTR_SOCKOPT_TIMESTAMP,
+ AF_INET_CR_ATTR_SOCKOPT_SNDBUF_ULOCK,
+ AF_INET_CR_ATTR_SOCKOPT_RCVBUF_ULOCK,
+
  AF_INET_CR_ATTR_MAX
 };
 #endif

--
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [patch 4/5][RFC - ipv4/udp checkpoint/restart] : c/r the inet options of the socket

From: Daniel Lezcano <dlezcano@fr.ibm.com>

This patch defines a set of netlink attributes to store/retrieve inet
options. The logic is to extend the netlink message attribute to take into
account these new values.

The multicast list is browsed first and the netlink nested attribute is filled
in the reverse order. That allows, when restoring the socket, to keep the
initial order of the multicast list. Not really a big issue if the list are
inverted, but that facilitate the test because the attribute will stay exactly,
the same and comparison with initial socket and restored socket can be done with
a simple "memcmp".

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 include/linux/af_inet_cr.h | 19 ++++
 net/ipv4/af_inet_cr.c | 205 ++-
 2 files changed, 223 insertions(+), 1 deletion(-)

Index: 2.6.20-cr/include/linux/af_inet_cr.h
===
--- 2.6.20-cr.orig/include/linux/af_inet_cr.h
+++ 2.6.20-cr/include/linux/af_inet_cr.h
@@ -71,6 +71,25 @@
  AF_INET_CR_ATTR_SOCKOPT_SNDBUF_ULOCK,
  AF_INET_CR_ATTR_SOCKOPT_RCVBUF_ULOCK,

+ /* ip options */
+ AF_INET_CR_ATTR_IPOPT_OPTIONS,
+ AF_INET_CR_ATTR_IPOPT_PKTINFO,
+ AF_INET_CR_ATTR_IPOPT_RECVTOS,
+ AF_INET_CR_ATTR_IPOPT_RECVTTL,
+ AF_INET_CR_ATTR_IPOPT_RECVOPTS,
+ AF_INET_CR_ATTR_IPOPT_RETOPTS,
+ AF_INET_CR_ATTR_IPOPT_TOS,
+ AF_INET_CR_ATTR_IPOPT_TTL,
+ AF_INET_CR_ATTR_IPOPT_HDRINCL,
+ AF_INET_CR_ATTR_IPOPT_RECVERR,
+ AF_INET_CR_ATTR_IPOPT_MTU_DISCOVER,
+ AF_INET_CR_ATTR_IPOPT_ROUTER_ALERT,
+ AF_INET_CR_ATTR_IPOPT_MULTICAST_TTL,
+ AF_INET_CR_ATTR_IPOPT_MULTICAST_LOOP,
+ AF_INET_CR_ATTR_IPOPT_MEMBERSHIP,
+ AF_INET_CR_ATTR_IPOPT_MREQ,
+ AF_INET_CR_ATTR_IPOPT_MULTICAST_IF,

```
+
  AF_INET_CR_ATTR_MAX
 };
 #endif
Index: 2.6.20-cr/net/ipv4/af_inet_cr.c
===================================================================
--- 2.6.20-cr.orig/net/ipv4/af_inet_cr.c
+++ 2.6.20-cr/net/ipv4/af_inet_cr.c
@@ -13,8 +13,12 @@
 #include <net/sock.h>
 #include <linux/fs.h>
 #include <linux/syscalls.h>
+#include <linux/in.h>
+#include <linux/igmp.h>
 #include <linux/af_inet_cr.h>

+#include <net/ip.h>
+
 /*
  * Netlink message policy definition
  */
@@ -44,6 +48,30 @@
  [AF_INET_CR_ATTR_SOCKOPT_SNDTIMEO] = { .len = sizeof(struct timeval) },
  [AF_INET_CR_ATTR_SOCKOPT_LINGER]   = { .len = sizeof(struct linger)  },
  [AF_INET_CR_ATTR_SOCKOPT_BINDTODEVICE]  = { .len  = IFNAMSIZ },
+
+ /* ip options */
+ [AF_INET_CR_ATTR_IPOPT_PKTINFO]         = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_IPOPT_RECVTOS]         = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_IPOPT_RECVTTL]         = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_IPOPT_RECVOPTS]        = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_IPOPT_RETOPTS]         = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_IPOPT_HDRINCL]         = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_IPOPT_RECVERR]         = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_IPOPT_ROUTER_ALERT]    = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_IPOPT_MULTICAST_LOOP]  = { .type = NLA_FLAG },
+ [AF_INET_CR_ATTR_IPOPT_MULTICAST_IF]    = { .type = NLA_U32 },
+ [AF_INET_CR_ATTR_IPOPT_TOS]             = { .type = NLA_U8 },
+ [AF_INET_CR_ATTR_IPOPT_TTL]             = { .type = NLA_U8 },
+ [AF_INET_CR_ATTR_IPOPT_MTU_DISCOVER]    = { .type = NLA_U8 },
+ [AF_INET_CR_ATTR_IPOPT_MULTICAST_TTL]   = { .type = NLA_U8 },
+ [AF_INET_CR_ATTR_IPOPT_MEMBERSHIP]      = { .type = NLA_NESTED },
+
+ [AF_INET_CR_ATTR_IPOPT_MREQ] =
+ { .len =  sizeof(struct ip_mreqn) },
+
+ [AF_INET_CR_ATTR_IPOPT_OPTIONS] =
+ { .len = sizeof(struct ip_options) + 40 },
```

```
+
 };

 /*
@@ -77,6 +105,28 @@
  { SO_BINDTODEVICE, AF_INET_CR_ATTR_SOCKOPT_BINDTODEVICE, 0, SET  },
 };

+/*
+ * ip options association with netlink attribute
+ */
+struct af_inet_cr_optattr ip_options[] = {
+ { IP_PKTINFO,        AF_INET_CR_ATTR_IPOPT_PKTINFO,        1, BOTH },
+ { IP_RECVTOS,        AF_INET_CR_ATTR_IPOPT_RECVTOS,        1, BOTH },
+ { IP_RECVTTL,        AF_INET_CR_ATTR_IPOPT_RECVTTL,        0, BOTH },
+ { IP_RECVOPTS,       AF_INET_CR_ATTR_IPOPT_RECVOPTS,       1, BOTH },
+ { IP_RETOPTS,        AF_INET_CR_ATTR_IPOPT_RETOPTS,        1, BOTH },
+ { IP_TOS,            AF_INET_CR_ATTR_IPOPT_TOS,            0, BOTH },
+ { IP_TTL,            AF_INET_CR_ATTR_IPOPT_TTL,            0, BOTH },
+ { IP_HDRINCL,        AF_INET_CR_ATTR_IPOPT_HDRINCL,        1, BOTH },
+ { IP_RECVERR,        AF_INET_CR_ATTR_IPOPT_RECVERR,        1, BOTH },
+ { IP_MTU_DISCOVER,   AF_INET_CR_ATTR_IPOPT_MTU_DISCOVER,   0, BOTH },
+ { IP_MULTICAST_TTL,  AF_INET_CR_ATTR_IPOPT_MULTICAST_TTL,  1, BOTH },
+ { IP_MULTICAST_LOOP, AF_INET_CR_ATTR_IPOPT_MULTICAST_LOOP, 0, BOTH },
+ { IP_MULTICAST_IF,   AF_INET_CR_ATTR_IPOPT_MULTICAST_IF,   1, BOTH },
+ /* FIXME (for RAW sockets) */
+ /*{ IP_ROUTER_ALERT, AF_INET_CR_ATTR_IPOPT_ROUTER_ALERT, 0, BOTH  },*/
+ /* FIXME */
+        /* { IP_OPTIONS, AF_INET_CR_ATTR_IPOPT_OPTIONS, */
+};

 /*
  * socket_lookup : search for socket using the inode number
@@ -188,7 +238,7 @@
 }

 /*
- * af_inet_cr_opt2attr : convert a netlink attribute to a socket option
+ * af_inet_cr_attr2opt : convert a netlink attribute to a socket option
  *  and set the option to the socket
  *
  * @info : the generic netlink message
@@ -363,6 +413,90 @@
 }

 /*
+ * dump_inetopt_mc : retrieve the multicast list and store them
+ *  to the netlink message
```

```
+ *
+ * @sock : the socket to retrieve the multicast list
+ * @skb : the skbuff containing the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static int dump_inetopt_mc(struct socket *sock, struct sk_buff *skb)
+{
+ int i, ret = 0;
+ struct sock *sk = sock->sk;
+ struct inet_sock *inet = inet_sk(sk);
+ struct ip_mc_socklist *mc;
+ struct nlattr *mx;
+ struct ip_mreqn **mreq = NULL;
+
+ mreq = kmalloc(sizeof(struct ip_mreqn *)*sysctl_igmp_max_memberships, GFP_KERNEL);
+ if (!mreq)
+  return -ENOMEM;
+
+ i = sysctl_igmp_max_memberships - 1;
+
+ rtnl_lock();
+ for (mc = inet->mc_list; mc; mc = mc->next)
+  mreq[i--] = &mc->multi;
+ i++;
+
+ if (i == sysctl_igmp_max_memberships)
+  goto out;
+
+ ret = -ENOMEM;
+ mx = nla_nest_start(skb, AF_INET_CR_ATTR_IPOPT_MEMBERSHIP);
+ if (!mx)
+  goto out;
+
+ for (; i < sysctl_igmp_max_memberships; i++) {
+  ret = nla_put(skb, AF_INET_CR_ATTR_IPOPT_MREQ,
+         sizeof(mc->multi), mreq[i]);
+  if (ret) {
+   nla_nest_cancel(skb, mx);
+   goto out;
+  }
+ }
+ nla_nest_end(skb, mx);
+out:
+ rtnl_unlock();
+ kfree(mreq);
+
+ return ret;
```

```
+}
+
+/*
+ * dump_inetopt : retrieve the inet options and store them
+ *  to the netlink message
+ *
+ * @sock : the socket to retrieve the inet options
+ * @skb : the skbuff containing the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static int dump_inetopt(struct socket *sock, struct sk_buff *skb)
+{
+ int ret;
+ size_t i, len;
+
+ len = sizeof(ip_options)/sizeof(struct af_inet_cr_optattr);
+
+ for (i = 0; i < len; i++) {
+  if (sock->type == SOCK_STREAM && ip_options[i].unconnected)
+   continue;
+  ret = af_inet_cr_opt2attr(skb, sock, IPPROTO_IP, &ip_options[i]);
+  if (ret)
+   return ret;
+ }
+
+ ret = dump_inetopt_mc(sock, skb);
+ if (ret)
+  return ret;
+
+ return 0;
+}
+
+/*
  * dump_socket : dump the socket state, type and options into a netlink message
  *  and transmit the message to the sender of the dump command.
  *
@@ -404,6 +538,10 @@
  if (ret)
   goto out;

+  ret = dump_inetopt(sock, skb);
+ if (ret)
+  goto out;
+
  ret = genlmsg_end(skb, msg_head);
  if (ret < 0)
   goto out;
```

```
@@ -517,6 +655,67 @@
 }

 /*
+ * restore_inetopt : extract multicast list from netlink message and
+ *  set it to the socket
+ *
+ * @sock : the socket to be restored
+ * @info : the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static inline int restore_inetopt_mc(struct socket *sock,
+         struct genl_info *info)
+{
+ struct ip_mreqn mreq;
+ struct nlattr *nla, *nested;
+ int nla_rem;
+ int ret;
+
+ nested = info->attrs[AF_INET_CR_ATTR_IPOPT_MEMBERSHIP];
+ if (nested) {
+  nla_for_each_nested(nla, nested, nla_rem) {
+   nla_memcpy(&mreq, nla, sizeof(mreq));
+   ret = kernel_setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP,
+       (char *)&mreq, sizeof(mreq));
+   if (ret)
+    return ret;
+  }
+ }
+ return 0;
+}
+
+/*
+ * restore_inetopt : extract inet options from netlink message and
+ *  set them to the socket
+ *
+ * @sock : the socket to be restored
+ * @info : the netlink message
+ *
+ * Returns 0 on success, < 0 otherwise
+ */
+static inline int restore_inetopt(struct socket *sock, struct genl_info *info)
+{
+ int ret;
+ size_t i, len;
+
+ len = sizeof(ip_options)/sizeof(struct af_inet_cr_optattr);
```

```
+
+ for (i = 0; i < len; i++) {
+  if (sock->type == SOCK_STREAM && ip_options[i].unconnected)
+   continue;
+  ret = af_inet_cr_attr2opt(info, sock, IPPROTO_IP, &ip_options[i]);
+  if (ret)
+   return ret;
+ }
+
+ ret = restore_inetopt_mc(sock, info);
+ if (ret)
+  return ret;
+
+ return 0;
+}
+
+/*
  * restore_socket : restore the socket from the netlink message content
  *
  * @sock : the socket to be restored
@@ -545,6 +744,10 @@
  ret = restore_sockopt(sock, info);
  if (ret)
   goto out;
+
+  ret = restore_inetopt(sock, info);
+ if (ret)
+  goto out;
 out:
  return ret;
 }

--
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [patch 5/5][RFC - ipv4/udp checkpoint/restart] : c/r the udp part of the socket

Posted by Daniel Lezcano on Wed, 06 Jun 2007 12:18:11 GMT

View Forum Message <> Reply to Message

From: Daniel Lezcano <dlezcano@fr.ibm.com>

This patch defines a set of netlink attributes to store/retrieve udp
option and endpoints. The logic is to extend the netlink message attribute

to take into account these new values.
The ops of struct sock is extended with the dump/restore callbacks, so when
a socket is asked to be checkpointed, the call will fail if the dump/restore
is not implemented in the protocol. That allows to bring C/R functionnality
for each protocol step by step.

 * At dump time : the local binding is retrieve from kernel_getname and
   distant binding is retrieve with kernel_getpeername.

 * At restore time : the local binding is set by the kernel_bind call and
   distant binding is set by kernel_connect
   If the local binding was done with an autobind, the userlock flags, will
   not be set, so the flag are resetted if they were not set during the dump.

One point to be discussed is : should we C/R sendQ and recvQ knowing the
protocol is not reliable ?

Signed-off-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 include/linux/af_inet_cr.h | 9 +++
 include/linux/udp_cr.h | 26 +++++++++
 include/net/sock.h | 6 +-
 net/ipv4/Makefile | 2
 net/ipv4/af_inet_cr.c | 23 ++++++++
 net/ipv4/udp.c | 6 +-
 net/ipv4/udp_cr.c | 119 ++
 7 files changed, 187 insertions(+), 4 deletions(-)

Index: 2.6.20-cr/include/linux/af_inet_cr.h
===
--- 2.6.20-cr.orig/include/linux/af_inet_cr.h
+++ 2.6.20-cr/include/linux/af_inet_cr.h
@@ -90,6 +90,15 @@
  AF_INET_CR_ATTR_IPOPT_MREQ,
  AF_INET_CR_ATTR_IPOPT_MULTICAST_IF,

+ /* udp options */
+ AF_INET_CR_ATTR_UDPOPT_CORK,
+
+ /* udp protocol */
+ UDP_CR_ATTR_BIND,
+ UDP_CR_ATTR_BIND_ADDR_ULOCK,
+ UDP_CR_ATTR_BIND_PORT_ULOCK,
+ UDP_CR_ATTR_PEER,
+
  AF_INET_CR_ATTR_MAX
 };
 #endif

```
Index: 2.6.20-cr/include/linux/udp_cr.h
===================================================================
--- /dev/null
+++ 2.6.20-cr/include/linux/udp_cr.h
@@ -0,0 +1,26 @@
+/*
+ *
+ *
+ */
+#ifndef _UDP_CR_H
+#define _UDP_CR_H
+#include <linux/skbuff.h>
+#include <linux/errno.h>
+#include <net/genetlink.h>
+
+#ifdef CONFIG_IP_CR
+extern int udp_dump(struct socket *sock, struct sk_buff *skb);
+extern int udp_restore(struct socket *sock, const struct genl_info *info);
+#else
+static inline int udp_dump(struct socket *sock, struct sk_buff *skb)
+{
+       return -ENOSYS;
+}
+
+static inline int udp_restore(struct socket *sock, const struct genl_info *info)
+{
+       return -ENOSYS;
+}
+#endif /* CONFIG_IP_CR */
+
+#endif
Index: 2.6.20-cr/include/net/sock.h
===================================================================
--- 2.6.20-cr.orig/include/net/sock.h
+++ 2.6.20-cr/include/net/sock.h
@@ -55,6 +55,7 @@
 #include <asm/atomic.h>
 #include <net/dst.h>
 #include <net/checksum.h>
+#include <net/genetlink.h>

 /*
  * This structure really needs to be cleaned up.
@@ -554,7 +555,10 @@
   void   (*hash)(struct sock *sk);
   void   (*unhash)(struct sock *sk);
   int    (*get_port)(struct sock *sk, unsigned short snum);
-
```

```
+#ifdef CONFIG_IP_CR
+ int                    (*dump)(struct socket *sock, struct sk_buff *skb);
+ int                    (*restore)(struct socket *sock, const struct genl_info *info);
+#endif
  /* Memory pressure */
  void   (*enter_memory_pressure)(void);
  atomic_t  *memory_allocated; /* Current allocated memory. */
Index: 2.6.20-cr/net/ipv4/Makefile
===================================================================
--- 2.6.20-cr.orig/net/ipv4/Makefile
+++ 2.6.20-cr/net/ipv4/Makefile
@@ -53,4 +53,4 @@

 obj-$(CONFIG_XFRM) += xfrm4_policy.o xfrm4_state.o xfrm4_input.o \
        xfrm4_output.o
-obj-$(CONFIG_IP_CR) += af_inet_cr.o
+obj-$(CONFIG_IP_CR) += af_inet_cr.o udp_cr.o
Index: 2.6.20-cr/net/ipv4/af_inet_cr.c
===================================================================
--- 2.6.20-cr.orig/net/ipv4/af_inet_cr.c
+++ 2.6.20-cr/net/ipv4/af_inet_cr.c
@@ -15,6 +15,7 @@
 #include <linux/syscalls.h>
 #include <linux/in.h>
 #include <linux/igmp.h>
+#include <linux/udp.h>
 #include <linux/af_inet_cr.h>

 #include <net/ip.h>
@@ -72,6 +73,14 @@
  [AF_INET_CR_ATTR_IPOPT_OPTIONS] =
  { .len = sizeof(struct ip_options) + 40 },

+ /* udp options */
+ [AF_INET_CR_ATTR_UDPOPT_CORK] = { .type = NLA_FLAG },
+
+ /* udp endpoints */
+ [UDP_CR_ATTR_BIND] = { .len =  sizeof(struct sockaddr_in) },
+ [UDP_CR_ATTR_PEER] = { .len =  sizeof(struct sockaddr_in) },
+ [UDP_CR_ATTR_BIND_ADDR_ULOCK] = { .type = NLA_FLAG },
+ [UDP_CR_ATTR_BIND_PORT_ULOCK] = { .type = NLA_FLAG },
 };

 /*
@@ -513,6 +522,9 @@
  void *msg_head;
  int ret;
```

```
+ if (!sk->sk_prot->dump)
+  return -ENOSYS;
+
  if (family != AF_INET)
   return -EINVAL;

@@ -542,6 +554,10 @@
  if (ret)
   goto out;

+ ret = sk->sk_prot->dump(sock, skb);
+ if (ret)
+                goto out;
+
  ret = genlmsg_end(skb, msg_head);
  if (ret < 0)
   goto out;
@@ -726,9 +742,12 @@
 static int restore_socket(struct socket *sock, struct genl_info *info)
 {
  int ret;
-
+ struct sock *sk = sock->sk;
  struct nlattr *nla;

+ if (!sk->sk_prot->dump)
+  return -ENOSYS;
+
  ret = -EINVAL;

  nla = info->attrs[AF_INET_CR_ATTR_SOCK_STATE];
@@ -748,6 +767,8 @@
   ret = restore_inetopt(sock, info);
  if (ret)
   goto out;
+
+ ret = sk->sk_prot->restore(sock, info);
 out:
  return ret;
 }
Index: 2.6.20-cr/net/ipv4/udp.c
===================================================================
--- 2.6.20-cr.orig/net/ipv4/udp.c
+++ 2.6.20-cr/net/ipv4/udp.c
@@ -93,10 +93,12 @@
 #include <linux/mm.h>
 #include <linux/inet.h>
 #include <linux/netdevice.h>
```

```
-#include <net/tcp_states.h>
+#include <linux/udp_cr.h>
 #include <linux/skbuff.h>
 #include <linux/proc_fs.h>
 #include <linux/seq_file.h>
+
+#include <net/tcp_states.h>
 #include <net/icmp.h>
 #include <net/route.h>
 #include <net/checksum.h>
@@ -1523,6 +1525,8 @@
  .unhash     = udp_lib_unhash,
  .get_port    = udp_v4_get_port,
  .obj_size    = sizeof(struct udp_sock),
+ .dump             = udp_dump,
+ .restore          = udp_restore,
 #ifdef CONFIG_COMPAT
  .compat_setsockopt = compat_udp_setsockopt,
  .compat_getsockopt = compat_udp_getsockopt,
Index: 2.6.20-cr/net/ipv4/udp_cr.c
===================================================================
--- /dev/null
+++ 2.6.20-cr/net/ipv4/udp_cr.c
@@ -0,0 +1,119 @@
+/*
+ *  Copyright (C) 2007 IBM Corporation
+ *
+ *  Author: Daniel Lezcano <dlezcano@fr.ibm.com>
+ *
+ *  This program is free software; you can redistribute it and/or
+ *  modify it under the terms of the GNU General Public License as
+ *  published by the Free Software Foundation, version 2 of the
+ *  License.
+ */
+
+
+#include <linux/in.h>
+#include <linux/udp.h>
+#include <linux/udp_cr.h>
+#include <linux/af_inet_cr.h>
+#include <net/sock.h>
+
+
+/*
+ * udp options association with netlink attribute
+ */
+struct af_inet_cr_optattr udp_options[] = {
+ { UDP_CORK, AF_INET_CR_ATTR_UDPOPT_CORK, BOTH },
```

```
+};
+
+int udp_dump(struct socket *sock, struct sk_buff *skb)
+{
+ int i, ret = 0;
+ size_t len;
+ struct sock *sk = sock->sk;
+ struct inet_sock *inet = inet_sk(sk);
+ struct sockaddr_in addr;
+ int addrlen = sizeof(addr);
+
+ len = sizeof(udp_options)/sizeof(struct af_inet_cr_optattr);
+
+ for (i = 0; i < len; i++) {
+  ret = af_inet_cr_opt2attr(skb, sock, SOL_UDP, &udp_options[i]);
+  if (ret)
+   goto out;
+ }
+
+ ret = kernel_getsockname(sock, (struct sockaddr *)&addr, &addrlen);
+ if (ret)
+  goto out;
+
+ if (addr.sin_addr.s_addr || addr.sin_port) {
+  ret = nla_put(skb, UDP_CR_ATTR_BIND, sizeof(addr), &addr);
+  if (ret)
+   goto out;
+ }
+
+ if (sk->sk_userlocks & SOCK_BINDADDR_LOCK) {
+  ret = nla_put_flag(skb, UDP_CR_ATTR_BIND_ADDR_ULOCK);
+  if (ret)
+   goto out;
+ }
+
+ if (sk->sk_userlocks & SOCK_BINDPORT_LOCK) {
+  ret = nla_put_flag(skb, UDP_CR_ATTR_BIND_PORT_ULOCK);
+  if (ret)
+   goto out;
+ }
+
+ if (inet->dport) {
+  ret = kernel_getpeername(sock, (struct sockaddr *)&addr, &addrlen);
+  if (ret)
+   goto out;
+  ret = nla_put(skb, UDP_CR_ATTR_PEER, sizeof(addr), &addr);
+  if (ret)
+   goto out;
```

```
+ }
+out:
+ return ret;
+}
+
+int udp_restore(struct socket *sock, const struct genl_info *info)
+{
+ int i, ret = 0;
+ size_t len;
+ struct nlattr *nla;
+ struct sockaddr_in addr;
+ int addrlen = sizeof(addr);
+
+ len = sizeof(udp_options)/sizeof(struct af_inet_cr_optattr);
+
+ for (i = 0; i < len; i++) {
+  ret = af_inet_cr_attr2opt(info, sock, SOL_UDP, &udp_options[i]);
+  if (ret)
+   goto out;
+ }
+
+ nla = info->attrs[UDP_CR_ATTR_BIND];
+ if (!nla)
+  goto out;
+
+ nla_memcpy(&addr, nla, sizeof(addr));
+
+ ret = kernel_bind(sock, (struct sockaddr *)&addr, addrlen);
+ if (ret)
+  goto out;
+
+ if (!info->attrs[UDP_CR_ATTR_BIND_ADDR_ULOCK])
+  sock->sk->sk_userlocks &= ~SOCK_BINDADDR_LOCK;
+
+ if (!info->attrs[UDP_CR_ATTR_BIND_PORT_ULOCK])
+  sock->sk->sk_userlocks &= ~SOCK_BINDPORT_LOCK;
+
+ nla = info->attrs[UDP_CR_ATTR_PEER];
+ if (!nla)
+  goto out;
+
+ ret = kernel_connect(sock, (struct sockaddr *)&addr, addrlen, 0);
+ if (ret)
+  goto out;
+out:
+ return ret;
+}
```

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers