Kernel Connector

Kernel connector - new netlink based userspace <-> kernel space easy to use communication module.

The Connector driver makes it easy to connect various agents using a netlink based network. One must register a callback and an identifier. When the driver receives a special netlink message with the appropriate identifier, the appropriate callback will be called.

From the userspace point of view it's quite straightforward:

- socket();
- bind();
- send();
- recv();

But if kernelspace wants to use the full power of such connections, the driver writer must create special sockets, must know about struct sk_buff handling, etc... The Connector driver allows any kernelspace agents to use netlink based networking for inter-process communication in a significantly easier way:

```
int cn_add_callback(const struct cb_id *id, char *name, void (*callback) (struct cn_msg *, struct netlink_skb_pa
void cn_netlink_send_mult(struct cn_msg *msg, u16 len, u32 portid, u32 __group, int gfp_mask);
void cn_netlink_send(struct cn_msg *msg, u32 portid, u32 __group, int gfp_mask);

struct cb_id
{
      __u32                   idx;
      __u32                   val;
};
```

idx and val are unique identifiers which must be registered in the connector.h header for in-kernel usage. *void (*callback) (void *)* is a callback function which will be called when a message with above idx.val is received by the connector core. The argument for that function must be dereferenced to *struct cn_msg *:

```
struct cn_msg
{
      struct cb_id            id;

      __u32                   seq;
      __u32                   ack;

      __u16                   len;    /* Length of the following data */
      __u16                   flags;
      __u8                    data[0];
};
```

Connector interfaces

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\driver-api\(linux-master)(Documentation)(driver-api)connector.rst, line 59)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/connector.h
```

Note:

When registering new callback user, connector core assigns netlink group to the user which is equal to its id.idx.

Protocol description

The current framework offers a transport layer with fixed headers. The recommended protocol which uses such a header is as following:

msg->seq and msg->ack are used to determine message genealogy. When someone sends a message, they use a locally unique sequence and random acknowledge number. The sequence number may be copied into nlmsghdr->nlmsg_seq too.

The sequence number is incremented with each message sent.

If you expect a reply to the message, then the sequence number in the received message MUST be the same as in the original message, and the acknowledge number MUST be the same + 1.

If we receive a message and its sequence number is not equal to one we are expecting, then it is a new message. If we receive a message and its sequence number is the same as one we are expecting, but its acknowledge is not equal to the sequence number in the original message + 1, then it is a new message.

Obviously, the protocol header contains the above id.

The connector allows event notification in the following form: kernel driver or userspace process can ask connector to notify it when selected ids will be turned on or off (registered or unregistered its callback). It is done by sending a special command to the connector driver (it also registers itself with id={-1, -1}).

As example of this usage can be found in the cn_test.c module which uses the connector to request notification and to send messages.

Reliability

Netlink itself is not a reliable protocol. That means that messages can be lost due to memory pressure or process' receiving queue overflowed, so caller is warned that it must be prepared. That is why the struct cn_msg [main connector's message header] contains u32 seq and u32 ack fields.

Userspace usage

2.6.14 has a new netlink socket implementation, which by default does not allow people to send data to netlink groups other than 1. So, if you wish to use a netlink socket (for example using connector) with a different group number, the userspace application must subscribe to that group first. It can be achieved by the following pseudocode:

```
s = socket(PF_NETLINK, SOCK_DGRAM, NETLINK_CONNECTOR);

l_local.nl_family = AF_NETLINK;
l_local.nl_groups = 12345;
l_local.nl_pid = 0;

if (bind(s, (struct sockaddr *)&l_local, sizeof(struct sockaddr_nl)) == -1) {
      perror("bind");
      close(s);
      return -1;
}

{
      int on = l_local.nl_groups;
      setsockopt(s, 270, 1, &on, sizeof(on));
}
```

Where 270 above is SOL_NETLINK, and 1 is a NETLINK_ADD_MEMBERSHIP socket option. To drop a multicast subscription, one should call the above socket option with the NETLINK_DROP_MEMBERSHIP parameter which is defined as 0.

2.6.14 netlink code only allows to select a group which is less or equal to the maximum group number, which is used at netlink_kernel_create() time. In case of connector it is CN_NETLINK_USERS + 0xf, so if you want to use group number 12345, you must increment CN_NETLINK_USERS to that number. Additional 0xf numbers are allocated to be used by non-in-kernel users.

Due to this limitation, group 0xffffffff does not work now, so one can not use add/remove connector's group notifications, but as far as I know, only cn_test.c test module used it.

Some work in netlink area is still being done, so things can be changed in 2.6.15 timeframe, if it will happen, documentation will be updated for that kernel.

Code samples

Sample code for a connector test module and user space can be found in samples/connector/. To build this code, enable CONFIG_CONNECTOR and CONFIG_SAMPLES.