NAME

Netlink - Kernel network configuration protocol

SYNOPSIS

#include <netlink/netlink.h>
#include <netlink/netlink_route.h>

int
socket(*AF_NETLINK, SOCK_RAW, int family*);

DESCRIPTION

Netlink is a user-kernel message-based communication protocol primarily used for network stack configuration. Netlink is easily extendable and supports large dumps and event notifications, all via a single socket. The protocol is fully asynchronous, allowing one to issue and track multiple requests at once. Netlink consists of multiple families, which commonly group the commands belonging to the particular kernel subsystem. Currently, the supported families are:

NETLINK_ROUTE	network configuration,
NETLINK_GENERIC	"container" family

The NETLINK_ROUTE family handles all interfaces, addresses, neighbors, routes, and VNETs configuration. More details can be found in rtnetlink(4). The NETLINK_GENERIC family serves as a "container", allowing registering other families under the NETLINK_GENERIC umbrella. This approach allows using a single netlink socket to interact with multiple netlink families at once. More details can be found in genetlink(4).

Netlink has its own sockaddr structure:

```
struct sockaddr_nl {
        uint8_t     nl_len;     /* sizeof(sockaddr_nl) */
        sa_family_t nl_family;  /* netlink family */
        uint16_t    nl_pad;     /* reserved, set to 0 */
        uint32_t    nl_pid;     /* automatically selected, set to 0 */
        uint32_t    nl_groups;  /* multicast groups mask to bind to */
};
```

Typically, filling this structure is not required for socket operations. It is presented here for completeness.

PROTOCOL DESCRIPTION

The protocol is message-based. Each message starts with the mandatory *nlmsghdr* header, followed by the family-specific header and the list of type-length-value pairs (TLVs). TLVs can be nested. All headers and TLVS are padded to 4-byte boundaries. Each send(2) or recv(2) system call may contain multiple messages.

BASE HEADER

```
struct nlmsghdr {
        uint32_t nlmsg_len;   /* Length of message including header */
        uint16_t nlmsg_type;  /* Message type identifier */
        uint16_t nlmsg_flags; /* Flags (NLM_F_) */
        uint32_t nlmsg_seq;   /* Sequence number */
        uint32_t nlmsg_pid;   /* Sending process port ID */
};
```

The *nlmsg_len* field stores the whole message length, in bytes, including the header. This length has to be rounded up to the nearest 4-byte boundary when iterating over messages. The *nlmsg_type* field represents the command/request type. This value is family-specific. The list of supported commands can be found in the relevant family header file. *nlmsg_seq* is a user-provided request identifier. An application can track the operation result using the NLMSG_ERROR messages and matching the *nlmsg_seq* The *nlmsg_pid* field is the message sender id. This field is optional for userland. The kernel sender id is zero. The *nlmsg_flags* field contains the message-specific flags. The following generic flags are defined:

NLM_F_REQUEST Indicates that the message is an actual request to the kernel
NLM_F_ACK Request an explicit ACK message with an operation result

The following generic flags are defined for the "GET" request types:

NLM_F_ROOT Return the whole dataset
NLM_F_MATCH Return all entries matching the criteria

These two flags are typically used together, aliased to NLM_F_DUMP

The following generic flags are defined for the "NEW" request types:

NLM_F_CREATE Create an object if none exists
NLM_F_EXCL Don’t replace an object if it exists
NLM_F_REPLACE Replace an existing matching object
NLM_F_APPEND Append to an existing object

The following generic flags are defined for the replies:

NLM_F_MULTI Indicates that the message is part of the message group
NLM_F_DUMP_INTR Indicates that the state dump was not completed
NLM_F_DUMP_FILTERED Indicates that the dump was filtered per request
NLM_F_CAPPED Indicates the original message was capped to its header
NLM_F_ACK_TLVS Indicates that extended ACK TLVs were included

TLVs

Most messages encode their attributes as type-length-value pairs (TLVs). The base TLV header:

```
struct nlattr {
        uint16_t nla_len;    /* Total attribute length */
        uint16_t nla_type;   /* Attribute type */
};
```

The TLV type (*nla_type*) scope is typically the message type or group within a family. For example, the RTN_MULTICAST type value is only valid for RTM_NEWROUTE , RTM_DELROUTE and RTM_GETROUTE messages. TLVs can be nested; in that case internal TLVs may have their own sub-types. All TLVs are packed with 4-byte padding.

CONTROL MESSAGES

A number of generic control messages are reserved in each family.

NLMSG_ERROR reports the operation result if requested, optionally followed by the metadata TLVs. The value of *nlmsg_seq* is set to its value in the original messages, while *nlmsg_pid* is set to the socket pid of the original socket. The operation result is reported via *struct nlmsgerr*:

```
struct nlmsgerr {
        int         error;          /* Standard errno */
        struct      nlmsghdr msg;   /* Original message header */
};
```

If the NETLINK_CAP_ACK socket option is not set, the remainder of the original message will follow. If the NETLINK_EXT_ACK socket option is set, the kernel may add a NLMSGERR_ATTR_MSG string TLV with the textual error description, optionally followed by the NLMSGERR_ATTR_OFFS TLV, indicating the offset from the message start that triggered an error. Some operations may return additional metadata encapsulated in the NLMSGERR_ATTR_COOKIE TLV. The metadata format is specific to the operation. If the operation reply is a multipart message, then no NLMSG_ERROR reply is generated, only a NLMSG_DONE message, closing multipart sequence.

NLMSG_DONE indicates the end of the message group: typically, the end of the dump. It contains a single *int* field, describing the dump result as a standard errno value.

SOCKET OPTIONS

Netlink supports a number of custom socket options, which can be set with setsockopt(2) with the SOL_NETLINK *level*:

NETLINK_ADD_MEMBERSHIP
: Subscribes to the notifications for the specific group (int).

NETLINK_DROP_MEMBERSHIP
: Unsubscribes from the notifications for the specific group (int).

NETLINK_LIST_MEMBERSHIPS
: Lists the memberships as a bitmask.

NETLINK_CAP_ACK
: Instructs the kernel to send the original message header in the reply without the message body.

NETLINK_EXT_ACK
: Acknowledges ability to receive additional TLVs in the ACK message.

Additionally, netlink overrides the following socket options from the SOL_SOCKET *level*:

SO_RCVBUF
: Sets the maximum size of the socket receive buffer. If the caller has PRIV_NET_ROUTE permission, the value can exceed the currently-set *kern.ipc.maxsockbuf* value.

SYSCTL VARIABLES

A set of sysctl(8) variables is available to tweak run-time parameters:

net.netlink.sendspace
: Default send buffer for the netlink socket. Note that the socket sendspace has to be at least as long as the longest message that can be transmitted via this socket.

net.netlink.recvspace
: Default receive buffer for the netlink socket. Note that the socket recvspace has to be least as long as the longest message that can be received from this socket.

net.netlink.nl_maxsockbuf
: Maximum receive buffer for the netlink socket that can be set via SO_RCVBUF socket option.

DEBUGGING

Netlink implements per-functional-unit debugging, with different severities controllable via the *net.netlink.debug* branch. These messages are logged in the kernel message buffer and can be seen in dmesg(8) The following severity levels are defined:

LOG_DEBUG(7)
: Rare events or per-socket errors are reported here. This is the default level, not impacting production performance.

LOG_DEBUG2(8)
: Socket events such as groups memberships, privilege checks, commands and dumps are logged. This level does not incur significant performance overhead.

LOG_DEBUG3(9)
: All socket events, each dumped or modified entities are logged. Turning it on may result in significant performance overhead.

ERRORS

Netlink reports operation results, including errors and error metadata, by sending a NLMSG_ERROR message for each request message. The following errors can be returned:

[EPERM] when the current privileges are insufficient to perform the required operation;

[ENOBUFS] or [ENOMEM]
: when the system runs out of memory for an internal data structure;

[ENOTSUP] when the requested command is not supported by the family or the family is not supported;

[EINVAL] when some necessary TLVs are missing or invalid, detailed info may be provided in NLMSGERR_ATTR_MSG and NLMSGERR_ATTR_OFFS TLVs;

[ENOENT] when trying to delete a non-existent object.

Additionally, a socket operation itself may fail with one of the errors specified in socket(2) , recv(2) or send(2)

SEE ALSO

genetlink(4), rtnetlink(4)

J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov, *Linux Netlink as an IP Services Protocol*, RFC

3549.

HISTORY

The netlink protocol appeared in FreeBSD 13.2.

AUTHORS

The netlink was implemented by Alexander Chernikov <*melifaro@FreeBSD.org*>. It was derived from the Google Summer of Code 2021 project by Ng Peng Nam Sean.