NAME

netlink − Communication between kernel and userspace (AF_NETLINK)

SYNOPSIS

#include <asm/types.h>
#include <sys/socket.h>
#include <linux/netlink.h>

netlink_socket = socket(AF_NETLINK, *socket_type***,** *netlink_family***);**

DESCRIPTION

Netlink is used to transfer information between kernel and userspace processes. It consists of a standard sockets-based interface for userspace processes and an internal kernel API for kernel modules. The internal kernel interface is not documented in this manual page. There is also an obsolete netlink interface via netlink character devices; this interface is not documented here and is only provided for backwards compatibility.

Netlink is a datagram-oriented service. Both **SOCK_RAW** and **SOCK_DGRAM** are valid values for *socket_type*. However, the netlink protocol does not distinguish between datagram and raw sockets.

netlink_family selects the kernel module or netlink group to communicate with. The currently assigned netlink families are:

NETLINK_ROUTE
Receives routing and link updates and may be used to modify the routing tables (both IPv4 and IPv6), IP addresses, link parameters, neighbor setups, queueing disciplines, traffic classes and packet classifiers (see **rtnetlink**(7)).

NETLINK_W1
Messages from 1-wire subsystem.

NETLINK_USERSOCK
Reserved for user-mode socket protocols.

NETLINK_FIREWALL
Transport IPv4 packets from netfilter to userspace. Used by *ip_queue* kernel module.

NETLINK_INET_DIAG
INET socket monitoring.

NETLINK_NFLOG
Netfilter/iptables ULOG.

NETLINK_XFRM
IPsec.

NETLINK_SELINUX
SELinux event notifications.

NETLINK_ISCSI
Open-iSCSI.

NETLINK_AUDIT
Auditing.

NETLINK_FIB_LOOKUP
Access to FIB lookup from userspace.

NETLINK_CONNECTOR
Kernel connector. See *Documentation/connector/** in the kernel source for further information.

NETLINK_NETFILTER
Netfilter subsystem.

NETLINK_IP6_FW
Transport IPv6 packets from netfilter to userspace. Used by *ip6_queue* kernel module.

NETLINK_DNRTMSG
DECnet routing messages.

NETLINK_KOBJECT_UEVENT
Kernel messages to userspace.

NETLINK_GENERIC
Generic netlink family for simplified netlink usage.

Netlink messages consist of a byte stream with one or multiple *nlmsghdr* headers and associated payload. The byte stream should only be accessed with the standard **NLMSG_*** macros. See **netlink**(3) for further information.

In multipart messages (multiple *nlmsghdr* headers with associated payload in one byte stream) the first and all following headers have the **NLM_F_MULTI** flag set, except for the last header which has the type **NLMSG_DONE**.

After each *nlmsghdr* the payload follows.

```
struct nlmsghdr {
  __u32 nlmsg_len;    /* Length of message including header. */
  __u16 nlmsg_type;   /* Type of message content. */
  __u16 nlmsg_flags;  /* Additional flags. */
  __u32 nlmsg_seq;    /* Sequence number. */
  __u32 nlmsg_pid;    /* PID of the sending process. */
};
```

nlmsg_type can be one of the standard message types: **NLMSG_NOOP** message is to be ignored, **NLMSG_ERROR** message signals an error and the payload contains an *nlmsgerr* structure, **NLMSG_DONE** message terminates a multipart message.

```
struct nlmsgerr {
  int error;       /* Negative errno or 0 for acknowledgements */
  struct nlmsghdr msg;  /* Message header that caused the error */
};
```

A netlink family usually specifies more message types, see the appropriate manual pages for that, for example, **rtnetlink**(7) for **NETLINK_ROUTE**.

Standard flag bits in *nlmsg_flags*

NLM_F_REQUEST	Must be set on all request messages.
NLM_F_MULTI	The message is part of a multipart message terminated by **NLMSG_DONE**.
NLM_F_ACK	Request for an acknowledgment on success.
NLM_F_ECHO	Echo this request.

Additional flag bits for GET requests

NLM_F_ROOT	Return the complete table instead of a single entry.

NLM_F_MATCH	Return all entries matching criteria passed in message content. Not implemented yet.
NLM_F_ATOMIC	Return an atomic snapshot of the table.
NLM_F_DUMP	Convenience macro; equivalent to (NLM_F_ROOT|NLM_F_MATCH).

Note that **NLM_F_ATOMIC** requires the **CAP_NET_ADMIN** capability or an effective UID of 0.

Additional flag bits for NEW requests

NLM_F_REPLACE	Replace existing matching object.
NLM_F_EXCL	Don't replace if the object already exists.
NLM_F_CREATE	Create object if it doesn't already exist.
NLM_F_APPEND	Add to the end of the object list.

nlmsg_seq and *nlmsg_pid* are used to track messages. *nlmsg_pid* shows the origin of the message. Note that there isn't a 1:1 relationship between *nlmsg_pid* and the PID of the process if the message originated from a netlink socket. See the **ADDRESS FORMATS** section for further information.

Both *nlmsg_seq* and *nlmsg_pid* are opaque to netlink core.

Netlink is not a reliable protocol. It tries its best to deliver a message to its destination(s), but may drop messages when an out-of-memory condition or other error occurs. For reliable transfer the sender can request an acknowledgement from the receiver by setting the **NLM_F_ACK** flag. An acknowledgment is an **NLMSG_ERROR** packet with the error field set to 0. The application must generate acknowledgements for received messages itself. The kernel tries to send an **NLMSG_ERROR** message for every failed packet. A user process should follow this convention too.

However, reliable transmissions from kernel to user are impossible in any case. The kernel can't send a netlink message if the socket buffer is full: the message will be dropped and the kernel and the userspace process will no longer have the same view of kernel state. It is up to the application to detect when this happens (via the **ENOBUFS** error returned by **recvmsg**(2)) and resynchronize.

Address Formats

The *sockaddr_nl* structure describes a netlink client in user space or in the kernel. A *sockaddr_nl* can be either unicast (only sent to one peer) or sent to netlink multicast groups (*nl_groups* not equal 0).

```
struct sockaddr_nl {
    sa_family_t     nl_family;  /* AF_NETLINK */
    unsigned short  nl_pad;     /* Zero. */
    pid_t           nl_pid;     /* Process ID. */
    __u32           nl_groups;  /* Multicast groups mask. */
};
```

nl_pid is the unicast address of netlink socket. It's always 0 if the destination is in the kernel. For a userspace process, *nl_pid* is usually the PID of the process owning the destination socket. However, *nl_pid* identifies a netlink socket, not a process. If a process owns several netlink sockets, then *nl_pid* can only be equal to the process ID for at most one socket. There are two ways to assign *nl_pid* to a netlink socket. If the application sets *nl_pid* before calling **bind**(2), then it is up to the application to make sure that *nl_pid* is unique. If the application sets it to 0, the kernel takes care of assigning it. The kernel assigns the process ID to the first netlink socket the process opens and assigns a unique *nl_pid* to every netlink socket that the process subsequently creates.

nl_groups is a bit mask with every bit representing a netlink group number. Each netlink family has a set of 32 multicast groups. When **bind**(2) is called on the socket, the *nl_groups* field in the *sockaddr_nl* should be set to a bit mask of the groups which it wishes to listen to. The default value for this field is zero

which means that no multicasts will be received. A socket may multicast messages to any of the multicast groups by setting *nl_groups* to a bit mask of the groups it wishes to send to when it calls **sendmsg**(2) or does a **connect**(2). Only processes with an effective UID of 0 or the **CAP_NET_ADMIN** capability may send or listen to a netlink multicast group. Any replies to a message received for a multicast group should be sent back to the sending PID and the multicast group.

VERSIONS

The socket interface to netlink is a new feature of Linux 2.2.

Linux 2.0 supported a more primitive device based netlink interface (which is still available as a compatibility option). This obsolete interface is not described here.

NETLINK_SELINUX appeared in Linux 2.6.4.

NETLINK_AUDIT appeared in Linux 2.6.6.

NETLINK_KOBJECT_UEVENT appeared in Linux 2.6.10.

NETLINK_W1 and NETLINK_FIB_LOOKUP appeared in Linux 2.6.13.

NETLINK_INET_DIAG, NETLINK_CONNECTOR and NETLINK_NETFILTER appeared in Linux 2.6.14.

NETLINK_GENERIC and NETLINK_ISCSI appeared in Linux 2.6.15.

NOTES

It is often better to use netlink via *libnetlink* or *libnl* than via the low-level kernel interface.

BUGS

This manual page is not complete.

EXAMPLE

The following example creates a **NETLINK_ROUTE** netlink socket which will listen to the **RTMGRP_LINK** (network interface create/delete/up/down events) and **RTMGRP_IPV4_IFADDR** (IPv4 addresses add/delete events) multicast groups.

```
struct sockaddr_nl sa;

memset(&sa, 0, sizeof(sa));
sa.nl_family = AF_NETLINK;
sa.nl_groups = RTMGRP_LINK | RTMGRP_IPV4_IFADDR;

fd = socket(AF_NETLINK, SOCK_RAW, NETLINK_ROUTE);
bind(fd, (struct sockaddr *) &sa, sizeof(sa));
```

The next example demonstrates how to send a netlink message to the kernel (pid 0). Note that application must take care of message sequence numbers in order to reliably track acknowledgements.

```
struct nlmsghdr *nh;    /* The nlmsghdr with payload to send. */
struct sockaddr_nl sa;
struct iovec iov = { (void *) nh, nh−>nlmsg_len };
struct msghdr msg;

msg = { (void *)&sa, sizeof(sa), &iov, 1, NULL, 0, 0 };
memset(&sa, 0, sizeof(sa));
sa.nl_family = AF_NETLINK;
nh−>nlmsg_pid = 0;
```

```
nh−>nlmsg_seq = ++sequence_number;
/* Request an ack from kernel by setting NLM_F_ACK. */
nh−>nlmsg_flags |= NLM_F_ACK;

sendmsg(fd, &msg, 0);
```

And the last example is about reading netlink message.

```
int len;
char buf[4096];
struct iovec iov = { buf, sizeof(buf) };
struct sockaddr_nl sa;
struct msghdr msg;
struct nlmsghdr *nh;

msg = { (void *)&sa, sizeof(sa), &iov, 1, NULL, 0, 0 };
len = recvmsg(fd, &msg, 0);

for (nh = (struct nlmsghdr *) buf; NLMSG_OK (nh, len);
     nh = NLMSG_NEXT (nh, len)) {
   /* The end of multipart message. */
   if (nh−>nlmsg_type == NLMSG_DONE)
      return;

   if (nh−>nlmsg_type == NLMSG_ERROR)
      /* Do some error handling. */
   ...

   /* Continue with parsing payload. */
   ...
}
```

SEE ALSO

cmsg(3), **netlink**(3), **capabilities**(7), **rtnetlink**(7)

ftp://ftp.inr.ac.ru/ip-routing/iproute2* for information about libnetlink.

http://people.suug.ch/~tgr/libnl/ for information about libnl.

RFC 3549 "Linux Netlink as an IP Services Protocol"

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.