NAME

route - kernel packet forwarding database

SYNOPSIS

#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/route.h>

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

DESCRIPTION

FreeBSD provides some packet routing facilities. The kernel maintains a routing information database, which is used in selecting the appropriate network interface when transmitting packets.

A user process (or possibly multiple co-operating processes) maintains this database by sending messages over a special kind of socket. This supplants fixed size ioctl(2)'s used in earlier releases. Routing table changes may only be carried out by the super user.

The operating system may spontaneously emit routing messages in response to external events, such as receipt of a re-direct, or failure to locate a suitable route for a request. The message types are described in greater detail below.

Routing database entries come in two flavors: for a specific host, or for all hosts on a generic subnetwork (as specified by a bit mask and value under the mask. The effect of wildcard or default route may be achieved by using a mask of all zeros, and there may be hierarchical routes.

When the system is booted and addresses are assigned to the network interfaces, each protocol family installs a routing table entry for each interface when it is ready for traffic. Normally the protocol specifies the route through each interface as a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface is requested to address the packet to the gateway listed in the routing entry (i.e., the packet is forwarded).

When routing a packet, the kernel will attempt to find the most specific route matching the destination. (If there are two different mask and value-under-the-mask pairs that match, the more specific is the one with more bits in the mask. A route to a host is regarded as being supplied with a mask of as many ones as there are bits in the destination). If no entry is found, the destination is declared to be unreachable, and a routing-miss message is generated if there are any listeners on the routing control socket described below.

A wildcard routing entry is specified with a zero destination address value, and a mask of all zeroes. Wildcard routes will be used when the system fails to find other routes matching the destination. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

One opens the channel for passing routing control messages by using the socket call shown in the synopsis above:

The *family* parameter may be AF_UNSPEC which will provide routing information for all address families, or can be restricted to a specific address family by specifying which one is desired. There can be more than one routing socket open per system.

Messages are formed by a header followed by a small number of sockaddrs (now variable length particularly in the ISO case), interpreted by position, and delimited by the new length entry in the sockaddr. An example of a message with four addresses might be an ISO redirect: Destination, Netmask, Gateway, and Author of the redirect. The interpretation of which address are present is given by a bit mask within the header, and the sequence is least significant to most significant bit within the vector.

Any messages sent to the kernel are returned, and copies are sent to all interested listeners. The kernel will provide the process ID for the sender, and the sender may use an additional sequence field to distinguish between outstanding messages. However, message replies may be lost when kernel buffers are exhausted.

The kernel may reject certain messages, and will indicate this by filling in the *rtm_errno* field. The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a non-existent entry, or ENOBUFS if insufficient resources were available to install a new route. In the current implementation, all routing processes run locally, and the values for *rtm_errno* are available through the normal *errno* mechanism, even if the routing reply message is lost.

A process may avoid the expense of reading replies to its own messages by issuing a setsockopt(2) call indicating that the SO_USELOOPBACK option at the SOL_SOCKET level is to be turned off. A process may ignore all messages from the routing socket by doing a shutdown(2) system call for further input.

If a route is in use when it is deleted, the routing entry will be marked down and removed from the routing table, but the resources associated with it will not be reclaimed until all references to it are released. User processes can obtain information about the routing entry to a specific destination by using a RTM_GET message, or by calling sysctl(3).

Messages include:

```
#define   RTM_ADD            0x1   /* Add Route */
#define   RTM_DELETE         0x2   /* Delete Route */
#define   RTM_CHANGE         0x3   /* Change Metrics, Flags, or Gateway */
#define   RTM_GET            0x4   /* Report Information */
#define   RTM_LOSING         0x5   /* Kernel Suspects Partitioning */
#define   RTM_REDIRECT       0x6   /* Told to use different route */
#define   RTM_MISS           0x7   /* Lookup failed on this address */
#define   RTM_LOCK           0x8   /* fix specified metrics */
#define   RTM_RESOLVE        0xb   /* request to resolve dst to LL addr - unused */
#define   RTM_NEWADDR        0xc   /* address being added to iface */
#define   RTM_DELADDR        0xd   /* address being removed from iface */
#define   RTM_IFINFO         0xe   /* iface going up/down etc. */
#define   RTM_NEWMADDR       0xf   /* mcast group membership being added to if */
#define   RTM_DELMADDR       0x10  /* mcast group membership being deleted */
#define   RTM_IFANNOUNCE     0x11  /* iface arrival/departure */
#define   RTM_IEEE80211      0x12  /* IEEE80211 wireless event */
```

A message header consists of one of the following:

```
struct rt_msghdr {
    u_short rtm_msglen;     /* to skip over non-understood messages */
    u_char  rtm_version;    /* future binary compatibility */
    u_char  rtm_type;       /* message type */
    u_short rtm_index;      /* index for associated ifp */
    int     rtm_flags;      /* flags, incl. kern & message, e.g. DONE */
    int     rtm_addrs;      /* bitmask identifying sockaddrs in msg */
    pid_t   rtm_pid;        /* identify sender */
    int     rtm_seq;        /* for sender to identify action */
    int     rtm_errno;      /* why failed */
    int     rtm_fmask;      /* bitmask used in RTM_CHANGE message */
    u_long  rtm_inits;      /* which metrics we are initializing */
    struct  rt_metrics rtm_rmx; /* metrics themselves */
};

struct if_msghdr {
```

```
    u_short ifm_msglen;        /* to skip over non-understood messages */
    u_char  ifm_version;       /* future binary compatibility */
    u_char  ifm_type;          /* message type */
    int     ifm_addrs;         /* like rtm_addrs */
    int     ifm_flags;         /* value of if_flags */
    u_short ifm_index;         /* index for associated ifp */
    struct  if_data ifm_data;  /* statistics and other data about if */
};

struct ifa_msghdr {
    u_short ifam_msglen;       /* to skip over non-understood messages */
    u_char  ifam_version;      /* future binary compatibility */
    u_char  ifam_type;         /* message type */
    int     ifam_addrs;        /* like rtm_addrs */
    int     ifam_flags;        /* value of ifa_flags */
    u_short ifam_index;        /* index for associated ifp */
    int     ifam_metric;       /* value of ifa_metric */
};

struct ifma_msghdr {
    u_short ifmam_msglen;      /* to skip over non-understood messages */
    u_char  ifmam_version;     /* future binary compatibility */
    u_char  ifmam_type;        /* message type */
    int     ifmam_addrs;       /* like rtm_addrs */
    int     ifmam_flags;       /* value of ifa_flags */
    u_short ifmam_index;       /* index for associated ifp */
};

struct if_announcemsghdr {
        u_short  ifan_msglen;        /* to skip over non-understood messages */
        u_char   ifan_version;       /* future binary compatibility */
        u_char   ifan_type;          /* message type */
        u_short  ifan_index;         /* index for associated ifp */
        char     ifan_name[IFNAMSIZ]; /* if name, e.g. "en0" */
        u_short  ifan_what;          /* what type of announcement */
};
```

The RTM_IFINFO message uses a *if_msghdr* header, the RTM_NEWADDR and RTM_DELADDR messages use a *ifa_msghdr* header, the RTM_NEWMADDR and RTM_DELMADDR messages use a *ifma_msghdr* header, the RTM_IFANNOUNCE message uses a *if_announcemsghdr* header, and all other messages use the *rt_msghdr* header.

The "struct rt_metrics" and the flag bits are as defined in rtentry(9).

Specifiers for metric values in rmx_locks and rtm_inits are:

```
#define    RTV_MTU       0x1    /* init or lock _mtu */
#define    RTV_HOPCOUNT  0x2    /* init or lock _hopcount */
#define    RTV_EXPIRE    0x4    /* init or lock _expire */
#define    RTV_RPIPE     0x8    /* init or lock _recvpipe */
#define    RTV_SPIPE     0x10   /* init or lock _sendpipe */
#define    RTV_SSTHRESH  0x20   /* init or lock _ssthresh */
#define    RTV_RTT       0x40   /* init or lock _rtt */
#define    RTV_RTTVAR    0x80   /* init or lock _rttvar */
#define    RTV_WEIGHT    0x100  /* init or lock _weight */
```

Specifiers for which addresses are present in the messages are:

```
#define RTA_DST       0x1    /* destination sockaddr present */
#define RTA_GATEWAY   0x2    /* gateway sockaddr present */
#define RTA_NETMASK   0x4    /* netmask sockaddr present */
#define RTA_GENMASK   0x8    /* cloning mask sockaddr present - unused */
#define RTA_IFP       0x10   /* interface name sockaddr present */
#define RTA_IFA       0x20   /* interface addr sockaddr present */
#define RTA_AUTHOR    0x40   /* sockaddr for author of redirect */
#define RTA_BRD       0x80   /* for NEWADDR, broadcast or p-p dest addr */
```

SEE ALSO

sysctl(3), route(8), rtentry(9)

The constants for the *rtm_flags* field are documented in the manual page for the route(8) utility.

HISTORY

A PF_ROUTE protocol family first appeared in 4.3BSD-Reno.