NAME

networking - introduction to networking facilities

SYNOPSIS

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

DESCRIPTION

This section is a general introduction to the networking facilities available in the system. Documentation in this part of section 4 is broken up into three areas: *protocol families* (domains), *protocols*, and *network interfaces*.

All network protocols are associated with a specific *protocol family*. A protocol family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. A protocol family may support multiple methods of addressing, though the current protocol implementations do not. A protocol family is normally comprised of a number of protocols, one per socket(2) type. It is not required that a protocol family support all socket types. A protocol family may contain multiple protocols supporting the same socket abstraction.

A protocol supports one of the socket abstractions detailed in socket(2). A specific protocol may be accessed either by creating a socket of the appropriate type and protocol family, or by requesting the protocol explicitly when creating a socket. Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol family/network architecture. Certain semantics of the basic socket abstractions are protocol specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide non-standard facilities or extensions to a mechanism. For example, a protocol supporting the SOCK_STREAM abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families and/or address formats. The SYNOPSIS section of each network interface entry gives a sample specification of the related drivers for use in providing a system description to the config(8) program. The DIAGNOSTICS section lists messages which may appear on the console and/or in the system error log, */var/log/messages* (see syslogd(8)), due to errors in device operation.

PROTOCOLS

The system currently supports the Internet protocols, the Xerox Network Systems(tm) protocols, and some of the ISO OSI protocols. Raw socket interfaces are provided to the IP protocol layer of the Internet, and to the IDP protocol of Xerox NS. Consult the appropriate manual pages in this section for more information regarding the support for each protocol family.

ADDRESSING

Associated with each protocol family is an address format. All network addresses adhere to a general structure, called a sockaddr, described below. However, each protocol imposes finer and more specific structure, generally renaming the variant, which is discussed in the protocol family manual page alluded to above.

```
struct sockaddr {
    u_char sa_len;
    u_char sa_family;
    char     sa_data[14];
};
```

The field *sa_len* contains the total length of the structure, which may exceed 16 bytes. The following address values for *sa_family* are known to the system (and additional formats are defined for possible future implementation):

```
#define    AF_UNIX      1    /* local to host (pipes, portals) */
#define    AF_INET      2    /* internetwork: UDP, TCP, etc. */
#define    AF_NS        6    /* Xerox NS protocols */
#define    AF_CCITT     10   /* CCITT protocols, X.25 etc */
#define    AF_HYLINK    15   /* NSC Hyperchannel */
#define    AF_ISO       18   /* ISO protocols */
```

ROUTING

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) used in earlier releases.

This facility is described in route(4).

INTERFACES

Each network interface in a system corresponds to a path through which messages may be sent and

received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, lo(4), do not.

The following ioctl(2) calls may be used to manipulate network interfaces. The **ioctl**() is made on a socket (typically of type SOCK_DGRAM) in the desired domain. Most of the requests supported in earlier releases take an *ifreq* structure as its parameter. This structure has the form

```
struct      ifreq {
#define    IFNAMSIZ    16
   char    ifr_name[IFNAMSIZ];       /* if name, e.g. "en0" */
   union {
      struct    sockaddr ifru_addr;
      struct    sockaddr ifru_dstaddr;
      struct    sockaddr ifru_broadaddr;
      struct    ifreq_buffer ifru_buffer;
      short     ifru_flags[2];
      short     ifru_index;
      int       ifru_metric;
      int       ifru_mtu;
      int       ifru_phys;
      int       ifru_media;
      caddr_t   ifru_data;
      int       ifru_cap[2];
   } ifr_ifru;
#define ifr_addr      ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr   ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_broadaddr ifr_ifru.ifru_broadaddr /* broadcast address */
#define ifr_buffer    ifr_ifru.ifru_buffer    /* user supplied buffer with its length */
#define ifr_flags     ifr_ifru.ifru_flags[0]  /* flags (low 16 bits) */
#define ifr_flagshigh ifr_ifru.ifru_flags[1]  /* flags (high 16 bits) */
#define ifr_metric    ifr_ifru.ifru_metric    /* metric */
#define ifr_mtu       ifr_ifru.ifru_mtu       /* mtu */
#define ifr_phys      ifr_ifru.ifru_phys      /* physical wire */
#define ifr_media     ifr_ifru.ifru_media     /* physical media */
#define ifr_data      ifr_ifru.ifru_data      /* for use by interface */
#define ifr_reqcap    ifr_ifru.ifru_cap[0]    /* requested capabilities */
#define ifr_curcap    ifr_ifru.ifru_cap[1]    /* current capabilities */
#define ifr_index     ifr_ifru.ifru_index     /* interface index */
};
```

Ioctl() requests to obtain addresses and requests both to set and retrieve other data are still fully supported and use the *ifreq* structure:

SIOCGIFADDR Get interface address for protocol family.

SIOCGIFDSTADDR Get point to point address for protocol family and interface.

SIOCGIFBRDADDR Get broadcast address for protocol family and interface.

SIOCSIFCAP Attempt to set the enabled capabilities field for the interface to the value of the *ifr_reqcap* field of the *ifreq* structure. Note that, depending on the particular interface features, some capabilities may appear hard-coded to enabled, or toggling a capability may affect the status of other ones. The supported capabilities field is read-only, and the *ifr_curcap* field is unused by this call.

SIOCGIFCAP Get the interface capabilities fields. The values for supported and enabled capabilities will be returned in the *ifr_reqcap* and *ifr_curcap* fields of the *ifreq* structure, respectively.

SIOCGIFDESCR Get the interface description, returned in the *buffer* field of *ifru_buffer* struct. The user supplied buffer length should be defined in the *length* field of *ifru_buffer* struct passed in as parameter, and the length would include the terminating nul character. If there is not enough space to hold the interface length, no copy would be done and the *buffer* field of *ifru_buffer* would be set to NULL. The kernel will store the buffer length in the *length* field upon return, regardless whether the buffer itself is sufficient to hold the data.

SIOCSIFDESCR Set the interface description to the value of the *buffer* field of *ifru_buffer* struct, with *length* field specifying its length (counting the terminating nul).

SIOCSIFFLAGS Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified; some interfaces may be reset so that incoming packets are no longer received. When marked up again, the interface is reinitialized.

SIOCGIFFLAGS Get interface flags.

SIOCSIFMETRIC Set interface routing metric. The metric is used only by user-level routers.

SIOCGIFMETRIC Get interface metric.

SIOCIFCREATE Attempt to create the specified interface. If the interface name is given without a unit number the system will attempt to create a new interface with an arbitrary unit number. On successful return the *ifr_name* field will contain the new interface name.

SIOCIFDESTROY Attempt to destroy the specified interface.

There are two requests that make use of a new structure:

SIOCAIFADDR An interface may have more than one address associated with it in some protocols. This request provides a means to add additional addresses (or modify characteristics of the primary address if the default address for the address family is specified). Rather than making separate calls to set destination or broadcast addresses, or network masks (now an integral feature of multiple protocols) a separate structure is used to specify all three facets simultaneously (see below). One would use a slightly tailored version of this struct specific to each family (replacing each sockaddr by one of the family-specific type). Where the sockaddr itself is larger than the default size, one needs to modify the **ioctl**() identifier itself to include the total size, as described in **ioctl**().

SIOCDIFADDR This requests deletes the specified address from the list associated with an interface. It also uses the *ifaliasreq* structure to allow for the possibility of protocols allowing multiple masks or destination addresses, and also adopts the convention that specification of the default address means to delete the first address for the interface belonging to the address family in which the original socket was opened.

SIOCGIFALIAS This request provides means to get additional addresses together with netmask and broadcast/destination from an interface. It also uses the *ifaliasreq* structure.

SIOCGIFCONF Get interface configuration list. This request takes an *ifconf* structure (see below) as a value-result parameter. The *ifc_len* field should be initially set to the size of the buffer pointed to by *ifc_buf*. On return it will contain the length, in bytes, of the configuration list.

SIOCIFGCLONERS Get list of clonable interfaces. This request takes an *if_clonereq* structure (see below) as a value-result parameter. The *ifcr_count* field should be set to the number of IFNAMSIZ sized strings that can be fit in the buffer pointed to by *ifcr_buffer*. On return, *ifcr_total* will be set to the number of clonable interfaces and the buffer pointed to by *ifcr_buffer* will be filled with the names of clonable

interfaces aligned on IFNAMSIZ boundaries.

```
/*
* Structure used in SIOCAIFADDR request.
*/
struct ifaliasreq {
        char    ifra_name[IFNAMSIZ];   /* if name, e.g. "en0" */
        struct  sockaddr        ifra_addr;
        struct  sockaddr        ifra_broadaddr;
        struct  sockaddr        ifra_mask;
};

/*
* Structure used in SIOCGIFCONF request.
* Used to retrieve interface configuration
* for machine (useful for programs which
* must know all networks accessible).
*/
struct ifconf {
    int   ifc_len;                  /* size of associated buffer */
    union {
        caddr_t    ifcu_buf;
        struct     ifreq *ifcu_req;
    } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req /* array of structures returned */
};

/* Structure used in SIOCIFGCLONERS request. */
struct if_clonereq {
        int     ifcr_total;     /* total cloners (out) */
        int     ifcr_count;     /* room for this many in user buffer */
        char    *ifcr_buffer;   /* buffer for cloner names */
};

/* Structure used in SIOCGIFDESCR and SIOCSIFDESCR requests */
struct ifreq_buffer {
        size_t  length;         /* length of the buffer */
        void   *buffer;         /* pointer to userland space buffer */
};
```

SEE ALSO

ioctl(2), socket(2), intro(4), config(8), routed(8), ifnet(9)

HISTORY

The **netintro** manual appeared in 4.3BSD-Tahoe.