NAME

domain, **protosw** - programming interface for kernel socket implementation

SYNOPSIS

#include <sys/param.h>
#include <sys/kernel.h>
#include <sys/protosw.h>
#include <sys/domain.h>

void
domain_add(*struct domain *dom*);

void
domain_remove(*struct domain *dom*);

void
DOMAIN_SET(*domain*);

int
protosw_register(*struct domain *dom*, *struct protosw *pr*);

int
protosw_unregister(*struct protosw *pr*);

DESCRIPTION

The **domain** subsystem allows implementation of communication protocols that are exposed to the userland via the socket(2) API. When an application performs a **socket**(*domain*, *type*, *protocol*) syscall, the kernel searches for a **domain** matching the *domain* argument, then within this domain, searches for a protocol matching *type*. If the third argument, *protocol*, is not 0, that value must also match. The structure found must implement certain methods, so that socket(2) API works for this particular kind of a socket.

A minimal **domain** structure implementing a domain shall be initialized with sparse C99 initializer and has public fields as follows:

```
struct domain {
    /*
     * Mandatory fields.
     */
    int     dom_family;         /* PF_xxx, first argument of socket(2) */
```

```
    char    *dom_name;          /* text name of the domain */
    u_int   dom_nprotosw;       /* length of dom_protosw[] */
    /*
     * Following methods are optional.
     */
    int     (*dom_probe)(void);                         /* check for support */
    struct rib_head *(*dom_rtattach)(uint32_t);     /* init route table */
    void (*dom_rtdetach)(struct rib_head *);        /* clean up table */
    void *(*dom_ifattach)(struct ifnet *);/* interface attach */
    void (*dom_ifdetach)(struct ifnet *, void *);/* & detach callbacks */
    int     (*dom_ifmtu)(struct ifnet *);               /* mtu change */
    /*
     * Mandatory variable size array of pointers to protosw structs.
     */
    struct  protosw *dom_protosw[];
};
```

Each domain contains the *dom_protosw* array of protocol switch structures (*struct protosw* *), one for each socket type supported. The array may have NULL spacers for loadable protocols. Sparse C99 initializers shall be used to initialize **protosw** structures. The structure has mandatory field *pr_type* and mandatory *pr_attach* method. The rest of the methods are optional, but a meaningful protocol should implement some.

```
struct protosw {
    short   pr_type;  /* second argument of socket(2) */
    short   pr_protocol;        /* third argument of socket(2) or 0 */
    short   pr_flags;  /* see protosw.h */
    pr_soreceive_t  *pr_soreceive;  /* recv(2) */
    pr_rcvd_t       *pr_rcvd;       /* soreceive_generic() if PR_WANTRCV */
    pr_sosend_t     *pr_sosend;     /* send(2) */
    pr_send_t       *pr_send;       /* send(2) via sosend_generic() */
    pr_ready_t      *pr_ready;      /* sendfile/ktls readyness */
    pr_sopoll_t     *pr_sopoll;     /* poll(2) */
    pr_attach_t     *pr_attach;     /* creation: socreate(), sonewconn() */
    pr_detach_t     *pr_detach;     /* destruction: sofree() */
    pr_connect_t    *pr_connect;    /* connect(2) */
    pr_disconnect_t *pr_disconnect; /* sodisconnect() */
    pr_close_t      *pr_close;      /* close(2) */
    pr_shutdown_t   *pr_shutdown;   /* shutdown(2) */
    pr_abort_t      *pr_abort;      /* abrupt tear down: soabort() */
```

```
    pr_aio_queue_t  *pr_aio_queue;  /* aio(9) */
    pr_bind_t       *pr_bind;       /* bind(2) */
    pr_bindat_t     *pr_bindat;     /* bindat(2) */
    pr_listen_t     *pr_listen;     /* listen(2) */
    pr_accept_t     *pr_accept;     /* accept(2) */
    pr_connectat_t  *pr_connectat;  /* connectat(2) */
    pr_connect2_t   *pr_connect2;   /* socketpair(2) */
    pr_control_t    *pr_control;    /* ioctl(2) */
    pr_rcvoob_t     *pr_rcvoob;     /* soreceive_rcvoob() */
    pr_ctloutput_t  *pr_ctloutput;  /* control output (from above) */
    pr_peeraddr_t   *pr_peeraddr;   /* getpeername(2) */
    pr_sockaddr_t   *pr_sockaddr;   /* getsockname(2) */
    pr_sense_t      *pr_sense;      /* stat(2) */
};
```

The following functions handle the registration of new domains and protocols.

domain_add() adds a new protocol domain to the system. In most cases **domain_add**() is not called directly, instead **DOMAIN_SET**() is used, which is a wrapper around **SYSINIT**() macro. If the new domain has defined a *dom_probe* routine, it is called first in **domain_add**() to determine if the domain should be supported on the current system. If the probe routine returns a non-0 value, then the domain will not be added. Once a domain is added it cannot be completely unloaded. This is because there is no reference counting system in place to determine if there are any active references from sockets within that domain. However, the exprimental **domain_remove**() exists, and unloadable domains may be supported in the future.

protosw_register() dynamically adds a protocol to a domain, if the latter has an empty slot in its *dom_protosw*. Dynamically added protocol can later be unloaded with **protosw_unregister**().

RETURN VALUES

The **domain_add**() never fails, but it may not add a domain if its *dom_probe* fails.

The **protosw_register**() function may fail if:

[EEXIST] A protocol with the same value of *pr_type* and *pr_protocol* already exists in the domain.

[ENOMEM] The domain doesn't have any NULL slots in its *dom_protosw*.

SEE ALSO

socket(2), SYSINIT(9)

HISTORY

The **domain** subsystem first appeared in 4.3BSD as the part of the very first socket(2) API implementation.

The **domain** subsystem and this manual page were significantly rewritten in FreeBSD 14.

AUTHORS

This manual page was written by Chad David <*davidc@acns.ab.ca*> and
Gleb Smirnoff <*glebius@FreeBSD.org*>.