NAME

socket - kernel socket interface

SYNOPSIS

#include <sys/socket.h>
#include <sys/socketvar.h>

void
soabort(*struct socket *so*);

int
soaccept(*struct socket *so*, *struct sockaddr **nam*);

int
socheckuid(*struct socket *so*, *uid_t uid*);

int
sobind(*struct socket *so*, *struct sockaddr *nam*, *struct thread *td*);

void
soclose(*struct socket *so*);

int
soconnect(*struct socket *so*, *struct sockaddr *nam*, *struct thread *td*);

int
socreate(*int dom*, *struct socket **aso*, *int type*, *int proto*, *struct ucred *cred*, *struct thread *td*);

int
sodisconnect(*struct socket *so*);

void
sodtor_set(*struct socket *so*, *void (*func)(struct socket *)*);

*struct sockaddr **
sodupsockaddr(*const struct sockaddr *sa*, *int mflags*);

void
sofree(*struct socket *so*);

void
sohasoutofband(*struct socket *so*);

int
solisten(*struct socket *so, int backlog, struct thread *td*);

void
solisten_proto(*struct socket *so, int backlog*);

int
solisten_proto_check(*struct socket *so*);

*struct socket **
sonewconn(*struct socket *head, int connstatus*);

int
sopoll(*struct socket *so, int events, struct ucred *active_cred, struct thread *td*);

int
sopoll_generic(*struct socket *so, int events, struct ucred *active_cred, struct thread *td*);

int
soreceive(*struct socket *so, struct sockaddr **psa, struct uio *uio, struct mbuf **mp0, struct mbuf **controlp, int *flagsp*);

int
soreceive_stream(*struct socket *so, struct sockaddr **paddr, struct uio *uio, struct mbuf **mp0, struct mbuf **controlp, int *flagsp*);

int
soreceive_dgram(*struct socket *so, struct sockaddr **paddr, struct uio *uio, struct mbuf **mp0, struct mbuf **controlp, int *flagsp*);

int
soreceive_generic(*struct socket *so, struct sockaddr **paddr, struct uio *uio, struct mbuf **mp0, struct mbuf **controlp, int *flagsp*);

int
soreserve(*struct socket *so, u_long sndcc, u_long rcvcc*);

void
sorflush(*struct socket *so*);

int
sosend(*struct socket *so, struct sockaddr *addr, struct uio *uio, struct mbuf *top, struct mbuf *control, int flags, struct thread *td*);

int
sosend_dgram(*struct socket *so, struct sockaddr *addr, struct uio *uio, struct mbuf *top, struct mbuf *control, int flags, struct thread *td*);

int
sosend_generic(*struct socket *so, struct sockaddr *addr, struct uio *uio, struct mbuf *top, struct mbuf *control, int flags, struct thread *td*);

int
soshutdown(*struct socket *so, int how*);

void
sotoxsocket(*struct socket *so, struct xsocket *xso*);

void
soupcall_clear(*struct socket *so, int which*);

void
soupcall_set(*struct socket *so, int which, int (*func)(struct socket *, void *, int), void *arg*);

void
sowakeup(*struct socket *so, struct sockbuf *sb*);

#include <sys/sockopt.h>

int
sosetopt(*struct socket *so, struct sockopt *sopt*);

int
sogetopt(*struct socket *so, struct sockopt *sopt*);

int
sooptcopyin(*struct sockopt *sopt, void *buf, size_t len, size_t minlen*);

int
sooptcopyout(*struct sockopt *sopt*, *const void *buf*, *size_t len*);

DESCRIPTION

The kernel **socket** programming interface permits in-kernel consumers to interact with local and network socket objects in a manner similar to that permitted using the socket(2) user API. These interfaces are appropriate for use by distributed file systems and other network-aware kernel services. While the user API operates on file descriptors, the kernel interfaces operate directly on *struct socket* pointers. Some portions of the kernel API exist only to implement the user API, and are not expected to be used by kernel code. The portions of the socket API used by socket consumers and implementations of network protocols will differ; some routines are only useful for protocol implementors.

Except where otherwise indicated, **socket** functions may sleep, and are not appropriate for use in an interrupt thread context or while holding non-sleepable kernel locks.

Creating and Destroying Sockets

A new socket may be created using **socreate**(). As with socket(2), arguments specify the requested domain, type, and protocol via *dom*, *type*, and *proto*. The socket is returned via *aso* on success. In addition, the credential used to authorize operations associated with the socket will be passed via *cred* (and will be cached for the lifetime of the socket), and the thread performing the operation via *td*. *Warning*: authorization of the socket creation operation will be performed using the thread credential for some protocols (such as raw sockets).

Sockets may be closed and freed using **soclose**(), which has similar semantics to close(2).

In certain circumstances, it is appropriate to destroy a socket without waiting for it to disconnect, for which **soabort**() is used. This is only appropriate for incoming connections which are in a partially connected state. It must be called on an unreferenced socket, by the thread which removed the socket from its listen queue, to prevent races. It will call into protocol code, so no socket locks may be held over the call. The caller of **soabort**() is responsible for setting the VNET context. The normal path to freeing a socket is **sofree**(), which handles reference counting on the socket. It should be called whenever a reference is released, and also whenever reference flags are cleared in socket or protocol code. Calls to **sofree**() should not be made from outside the socket layer; outside callers should use **soclose**() instead.

Connections and Addresses

The **sobind**() function is equivalent to the bind(2) system call, and binds the socket *so* to the address *nam*. The operation would be authorized using the credential on thread *td*.

The **soconnect**() function is equivalent to the connect(2) system call, and initiates a connection on the

socket *so* to the address *nam*. The operation will be authorized using the credential on thread *td*. Unlike the user system call, **soconnect**() returns immediately; the caller may msleep(9) on *so->so_timeo* while holding the socket mutex and waiting for the SS_ISCONNECTING flag to clear or *so->so_error* to become non-zero. If **soconnect**() fails, the caller must manually clear the SS_ISCONNECTING flag.

A call to **sodisconnect**() disconnects the socket without closing it.

The **soshutdown**() function is equivalent to the shutdown(2) system call, and causes part or all of a connection on a socket to be closed down.

Sockets are transitioned from non-listening status to listening with **solisten**().

Socket Options

The **sogetopt**() function is equivalent to the getsockopt(2) system call, and retrieves a socket option on socket *so*. The **sosetopt**() function is equivalent to the setsockopt(2) system call, and sets a socket option on socket *so*.

The second argument in both **sogetopt**() and **sosetopt**() is the *sopt* pointer to a *struct sopt* describing the socket option operation. The caller-allocated structure must be zeroed, and then have its fields initialized to specify socket option operation arguments:

sopt_dir Set to SOPT_SET or SOPT_GET depending on whether this is a get or set operation.

sopt_level Specify the level in the network stack the operation is targeted at; for example, SOL_SOCKET.

sopt_name Specify the name of the socket option to set.

sopt_val Kernel space pointer to the argument value for the socket option.

sopt_valsize Size of the argument value in bytes.

Socket Upcalls

In order for the owner of a socket to be notified when the socket is ready to send or receive data, an upcall may be registered on the socket. The upcall is a function that will be called by the socket framework when a socket buffer associated with the given socket is ready for reading or writing. **soupcall_set**() is used to register a socket upcall. The function *func* is registered, and the pointer *arg* will be passed as its second argument when it is called by the framework. The possible values for *which* are SO_RCV and SO_SND, which register upcalls for receive and send events, respectively. The upcall function **func**() must return either SU_OK or SU_ISCONNECTED, depending on whether or not a call

to soisconnected should be made by the socket framework after the upcall returns. The upcall *func* cannot call soisconnected itself due to lock ordering with the socket buffer lock. Only SO_RCV upcalls should return SU_ISCONNECTED. When a SO_RCV upcall returns SU_ISCONNECTED, the upcall will be removed from the socket.

Upcalls are removed from their socket by **soupcall_clear**(). The *which* argument again specifies whether the sending or receiving upcall is to be cleared, with SO_RCV or SO_SND.

Socket Destructor Callback

A kernel system can use the **sodtor_set**() function to set a destructor for a socket. The destructor is called when the socket is about to be freed. The destructor is called before the protocol detach routine. The destructor can serve as a callback to initiate additional cleanup actions.

Socket I/O

The **soreceive**() function is equivalent to the recvmsg(2) system call, and attempts to receive bytes of data from the socket *so*, optionally blocking awaiting for data if none is ready to read. Data may be retrieved directly to kernel or user memory via the *uio* argument, or as an mbuf chain returned to the caller via *mp0*, avoiding a data copy. The *uio* must always be non-NULL. If *mp0* is non-NULL, only the *uio_resid* of *uio* is used. The caller may optionally retrieve a socket address on a protocol with the PR_ADDR capability by providing storage via non-NULL *psa* argument. The caller may optionally retrieve control data mbufs via a non-NULL *controlp* argument. Optional flags may be passed to **soreceive**() via a non-NULL *flagsp* argument, and use the same flag name space as the recvmsg(2) system call.

The **sosend**() function is equivalent to the sendmsg(2) system call, and attempts to send bytes of data via the socket *so*, optionally blocking if data cannot be immediately sent. Data may be sent directly from kernel or user memory via the *uio* argument, or as an mbuf chain via *top*, avoiding a data copy. Only one of the *uio* or *top* pointers may be non-NULL. An optional destination address may be specified via a non-NULL *addr* argument, which may result in an implicit connect if supported by the protocol. The caller may optionally send control data mbufs via a non-NULL *control* argument. Flags may be passed to **sosend**() using the *flags* argument, and use the same flag name space as the sendmsg(2) system call.

Kernel callers running in an interrupt thread context, or with a mutex held, will wish to use non-blocking sockets and pass the MSG_DONTWAIT flag in order to prevent these functions from sleeping.

A socket can be queried for readability, writability, out-of-band data, or end-of-file using **sopoll**(). The possible values for *events* are as for poll(2), with symbolic values POLLIN, POLLPRI, POLLOUT, POLLRDNORM, POLLWRNORM, POLLRDBAND, and POLLINGEOF taken from *<sys/poll.h>*.

Calls to **soaccept**() pass through to the protocol's accept routine to accept an incoming connection.

Socket Utility Functions

The uid of a socket's credential may be compared against a *uid* with **socheckuid**().

A copy of an existing *struct sockaddr* may be made using **sodupsockaddr**().

Protocol implementations notify the socket layer of the arrival of out-of-band data using **sohasoutofband**(), so that the socket layer can notify socket consumers of the available data.

An "external-format" version of a *struct socket* can be created using **sotoxsocket**(), suitable for isolating user code from changes in the kernel structure.

Protocol Implementations

Protocols must supply an implementation for **solisten**(); such protocol implementations can call back into the socket layer using **solisten_proto_check**() and **solisten_proto**() to check and set the socket-layer listen state. These callbacks are provided so that the protocol implementation can order the socket layer and protocol locks as necessary. Protocols must supply an implementation of **soreceive**(); the functions **soreceive_stream**(), **soreceive_dgram**(), and **soreceive_generic**() are supplied for use by such implementations.

Protocol implementations can use **sonewconn**() to create a socket and attach protocol state to that socket. This can be used to create new sockets available for **soaccept**() on a listen socket. The returned socket has a reference count of zero.

Protocols must supply an implementation for **sopoll**(); **sopoll_generic**() is provided for the use by protocol implementations.

The functions **sosend_dgram**() and **sosend_generic**() are supplied to assist in protocol implementations of **sosend**().

When a protocol creates a new socket structure, it is necessary to reserve socket buffer space for that socket, by calling **soreserve**(). The rough inverse of this reservation is performed by **sorflush**(), which is called automatically by the socket framework.

When a protocol needs to wake up threads waiting for the socket to become ready to read or write, variants of **sowakeup**() are used. The **sowakeup**() function should not be called directly by protocol code, instead use the wrappers **sorwakeup**(), **sorwakeup_locked**(), **sowwakeup**(), and **sowwakeup_locked**() for readers and writers, with the corresponding socket buffer lock not already locked, or already held, respectively.

The functions **sooptcopyin**() and **sooptcopyout**() are useful for transferring *struct sockopt* data between

user and kernel code.

SEE ALSO

bind(2), close(2), connect(2), getsockopt(2), recv(2), send(2), setsockopt(2), shutdown(2), socket(2), ng_ksocket(4), intr_event(9), msleep(9), ucred(9)

HISTORY

The socket(2) system call appeared in 4.2BSD. This manual page was introduced in FreeBSD 7.0.

AUTHORS

This manual page was written by Robert Watson and
Benjamin Kaduk.

BUGS

The use of explicitly passed credentials, credentials hung from explicitly passed threads, the credential on curthread, and the cached credential from socket creation time is inconsistent, and may lead to unexpected behaviour. It is possible that several of the *td* arguments should be *cred* arguments, or simply not be present at all.

The caller may need to manually clear SS_ISCONNECTING if **soconnect**() returns an error.

The MSG_DONTWAIT flag is not implemented for **sosend**(), and may not always work with **soreceive**() when zero copy sockets are enabled.

This manual page does not describe how to register socket upcalls or monitor a socket for readability/writability without using blocking I/O.

The **soref**() and **sorele**() functions are not described, and in most cases should not be used, due to confusing and potentially incorrect interactions when **sorele**() is last called after **soclose**().