Appendix 1

System Calls And Library Routines Used With Sockets

Introduction

In BSD UNIX, communication centers around the socket abstraction. Applications use a set of socket system calls to communicate with TCP/IP software in the operating system. A client application creates a socket, connects it to a server on a remote machine, and uses it to transfer data and to receive data from the remote machine. Finally, when the client application finishes using the socket, it closes it. A server creates a socket, binds it to a well-known protocol port on the local machine, and waits for clients to contact it.

Each page of this appendix describes one of the system calls or library functions that programmers use when writing client or server applications. The functions are arranged in alphabetic order, with one page devoted to a given function. The functions listed include: *accept, bind, close, connect, fork, gethostbyaddr, gethostbyname, gethostid, gethostname, getpeername, getprotobyname, getservbyname, getsockname, getsockopt, gettimeofday, listen, read, recv, recvfrom, recvmsg, select, send, sendmsg, sendto, sethostid, setsockopt, shutdown, socket,* and *write*.

The Accept System Call

Use

```
retcode = accept ( socket, addr, addrlen );
```

Description

Servers use the *accept* function to accept the next incoming connection on a passive socket after they have called *socket* to create a socket, *bind* to specify a local IP address and protocol port number, and *listen* to make the socket passive and to set the length of the connection request queue. *Accept* removes the next connection request from the queue (or waits until a connection request arrives), creates a new socket for the request, and returns the descriptor for the new socket. *Accept* only applies to stream sockets (e.g., those used with TCP).

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* function.
addr	&sockaddr	A pointer to an address structure. *Accept* fills in the structure with the IP address and protocol port number of the remote machine.
addrlen	&int	A pointer to an integer that initially specifies the size of the *sockaddr* argument and, when the call returns, specifies the number of bytes stored in argument *addr*.

Return Code

Accept returns a nonnegative socket descriptor if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
EOPNOTSUPP	The socket is not of type SOCK_STREAM.
EFAULT	The pointer in argument 2 is invalid.
EWOULDBLOCK	The socket is marked nonblocking and no connections are waiting to be accepted (i.e. the call would block).

The Bind System Call

Use

retcode = bind (socket, localaddr, addrlen);

Description

Bind specifies a local IP address and protocol port number for a socket. *Bind* is primarily used by servers, which need to specify a well-known protocol port.

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* call.
localaddr	&sockaddr	The address of a structure that specifies an IP address and protocol port number.
addrlen	int	The size of the address structure in bytes.

Chapter 5 contains a description of the *sockaddr* structure.

Return Code

Bind returns 0 if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains a code that specifies the cause of the error. The possible errors are:

Value in errno	Cause of the Error
EBADF	Argument *socket* does not specify a valid descriptor.
ENOTSOCK	Argument *socket* does not specify a socket descriptor.
EADDRNOTAVAIL	The specified address is unavailable (e.g., an IP address does not match a local interface).
EADDRINUSE	The specified address is in use (e.g., another process has allocated the protocol port).
EINVAL	The socket already has an address bound to it.
EACCES	The application program does not have permission to use the address specified.
EFAULT	The *localaddr* argument pointer is invalid.

The Close System Call

Use

```
retcode = close ( socket );
```

Description

An application calls *close* after it finishes using a socket. *Close* terminates communication gracefully and removes the socket. Any unread data waiting at the socket will be discarded.

In practice, UNIX implements a reference count mechanism to allow multiple processes to share a socket. If *n* processes share a socket, the reference count will be *n*. *Close* decrements the reference count each time a process calls it. Once the reference count reaches zero (i.e. all processes have called *close*), the socket will be deallocated.

Arguments

Arg	Type	Meaning
socket	int	The descriptor of a socket to be closed.

Return Code

Close returns 0 if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains the following value:

Value in errno	Cause of the Error
EBADF	The argument does not specify a valid descriptor.

The Connect System Call

Use

retcode = connect (socket, addr, addrlen);

Description

Connect allows the caller to specify the remote endpoint address for a previously created socket. If the socket uses TCP, *connect* uses the 3-way handshake to establish a connection; if the socket uses UDP, *connect* specifies the remote endpoint but does not transfer any datagrams to it.

Arguments

Arg	Type	Meaning
socket	int	The descriptor of a socket.
addr	&sockaddr_in	The remote machine endpoint.
addrlen	int	The length of the second argument.

Chapter 5 contains a description of the *sockaddr_in* structure.

Return Code

Connect returns zero if successful and -1 to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
EAFNOSUPPORT	The address family specified in the remote endpoint cannot be used with this type of socket.
EADDRNOTAVAIL	The specified endpoint address is not available.
EISCONN	The socket is already connected.
ETIMEDOUT	(TCP only) The protocol reached timeout without successfully establishing a connection.
ECONNREFUSED	(TCP only) Connection refused by remote machine.
ENETUNREACH	(TCP only) The network is not currently reachable.
EADDRINUSE	The specified address is already in use.
EINPROGRESS	(TCP only) The socket is nonblocking and a connection attempt would block.
EALREADY	(TCP only) The socket is nonblocking and the call would wait for a previous connection attempt to complete.

The Fork System Call

Use

```
retcode = fork();
```

Description

Although not directly related to communication sockets, *fork* is essential because servers use it to create concurrent processes. *Fork* creates a new process executing the same code as the original. The two processes share all socket and file descriptors that are open when the call to *fork* occurs. The two processes have different process identifiers and different parent process identifiers.

Arguments

Fork does not take any arguments.

Return Code

If successful, *fork* returns 0 to the child process and the (nonzero) process identifier of the newly created process to the original process. It returns *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in *errno*	Cause of the Error
EAGAIN	The system limit on total processes has been reached, or the per-user limit on processes has been reached.
ENOMEM	The system has insufficient memory for a new process.

The Gethostbyaddr Library Function

Use

retcode = gethostbyaddr (addr, alen, atype);

Description

Gethostbyaddr searches for information about a host given its IP address.

Arguments

Arg	Type	Meaning
addr	&char	A pointer to an array that contains a host address (e.g., an IP address).
alen	int	An integer that gives the address length (*4* for IP).
atype	int	An integer that gives the address type (AF_INET for an IP address).

Return Code

Gethostbyaddr returns a pointer to a *hostent* structure if successful and *0* to indicate that an error has occurred. The *hostent* structure is declared to be:

```
struct hostent {          /* entry for a host              */
      char  *h_name;      /* official host name            */
      char  *h_aliases[]; /* list of other aliases         */
      int    h_addrtype;  /* host address type             */
      int    h_length;    /* length of host address        */
      char **h_addr_list; /* list of addresses for host    */
};
```

When an error occurs, the global variable *h_errno* contains one of the following values:

Value in h_errno	Cause of the Error
HOST_NOT_FOUND	The name specified is unknown.
TRY_AGAIN	Temporary error: local server could not contact the authority at present.
NO_RECOVERY	Irrecoverable error occurred.
NO_ADDRESS	The specified name is valid, but it does not correspond to an IP address.

The Gethostbyname Library Call

Use

retcode = gethostbyname (name);

Description

Gethostbyname maps a host name to an IP address.

Arguments

Arg	Type	Meaning
name	&char	The address of a character string that contains a host name.

Return Code

Gethostbyname returns a pointer to a *hostent* structure if successful and *0* to indicate that an error has occurred. The *hostent* structure is declared to be:

```
struct hostent {            /* entry for a host             */
      char  *h_name;        /* official host name           */
      char  *h_aliases[];   /* list of other aliases        */
      int    h_addrtype;    /* host address type            */
      int    h_length;      /* length of host address       */
      char **h_addr_list;   /* list of addresses for host   */
};
```

When an error occurs, the global variable *h_errno* contains one of the following values:

Value in h_errno	Cause of the Error
HOST_NOT_FOUND	The name specified is unknown.
TRY_AGAIN	Temporary error: local server could not contact the authority at present.
NO_RECOVERY	Irrecoverable error occurred.
NO_ADDRESS	The specified name is valid, but it does not correspond to an IP address.

The Gethostid System Call

Use

```
hostid = gethostid();
```

Description

Applications call *gethostid* to determine the unique *32*-bit host identifier assigned to the local machine. Usually, the host identifier is the machine's primary IP address.

Arguments

Gethostid does not take any arguments.

Return Value

Gethostid returns a long integer containing the host identifier.

The Gethostname System Call

Use

```
retcode = gethostname ( name, namelen );
```

Description

Gethostname returns the primary name of the local machine in the form of a text string.

Arguments

Arg	Type	Meaning
name	&char	The address of the character array into which the name should be placed.
namelen	int	The length of the *name* array (should be at least *65*).

Return Code

Gethostname returns *0* if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains the following value:

Value in errno	Cause of the Error
EFAULT	The *name* or *namelen* arguments were incorrect.

The Getpeername System Call

Use

retcode = getpeername (socket, remaddr, addrlen);

Description

An application uses *getpeername* to obtain the remote endpoint address for a connected socket. Usually, a client knows the remote endpoint address because it calls *connect* to set it. However, a server that uses *accept* to obtain a connection may need to interrogate the socket to find out the remote address.

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* function.
remaddr	&sockaddr	A pointer to a *sockaddr* structure that will contain the endpoint address.
addrlen	&int	A pointer to an integer that contains the length of the second argument initially, and the actual length of the endpoint address upon return.

Chapter 5 contains a description of the *sockaddr* structure.

Return Code

Getpeername returns 0 if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
ENOTCONN	The socket is not a connected socket.
ENOBUFS	The system had insufficient resources to perform the operation.
EFAULT	The *remaddr* argument pointer is invalid.

The Getprotobyname Function Call

Use

retcode = getprotobyname (name);

Description

Applications call *getprotobyname* to find a protocol's official integer value from its *name* alone.

Arguments

Arg	Type	Meaning
name	&char	The address of a string that contains the protocol name.

Return Code

Getprotobyname returns a pointer to a structure of type *protoent* if successful and *0* to indicate that an error has occurred. Structure *protoent* is declared to be:

```
struct protoent {          /* entry that describes a protocol  */
      char  *p_name;       /* official name of protocol        */
      char **p_aliases;    /* list of aliases for the protocol */
      int    p_proto;      /* official protocol number         */
};
```

The Getservbyname Library Call

Use

retcode = getservbyname (name, proto);

Description

Getservbyname obtains an entry from the network services database given a service name. Clients and servers both call *getservbyname* to map a service name to a protocol port number.

Arguments

Arg	Type	Meaning
name	&char	A pointer to a string of characters that contains a service name.
proto	&char	A pointer to a string of characters that contains the name of the protocol to be used (e.g., *tcp*).

Return Code

Getservbyname returns a pointer to a *servent* structure if successful and a null pointer (0) to indicate that an error has occurred. The *servent* structure is declared to be:

```
struct servent {            /* one service entry          */
      char  *s_name ;       /* official service name      */
      char **s_aliases;     /* list of other aliases      */
      int    s_port;        /* port used for this service */
      char  *s_proto;       /* protocol used for service  */
};
```

The Getsockname System Call

Use

retcode = getsockname (socket, name, namelen);

Description

Getsockname obtains the local address of the specified socket.

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* function.
name	&sockaddr	The address of a structure that will contain the IP address and protocol port number of the socket.
namelen	&int	The number of positions in the *name* structure; returned as the size of the structure.

Return Code

Getsockname returns *0* if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
ENOBUFS	Insufficient buffer space was available in the system.
EFAULT	The address of *name* or *namelen* is incorrect.

The Getsockopt System Call

Use

retcode = getsockopt (socket, level, opt, optval, optlen);

Description

Getsockopt permits an application to obtain the value of a parameter (option) for a socket or a protocol the socket uses.

Arguments

Arg	Type	Meaning
socket	int	The descriptor of a socket.
level	int	An integer that identifies a protocol level.
opt	int	An integer that identifies an option.
optval	&char	The address of a buffer in which the value is returned.
optlen	&int	Size of buffer; returned as length of the value found.

The socket-level options that apply to all sockets include:

SO_DEBUG	Status of debugging information
SO_REUSEADDR	Allow local address reuse?
SO_KEEPALIVE	Status of connection keep-alive
SO_DONTROUTE	Bypass routing for outgoing messages?
SO_LINGER	Linger on close if data present?
SO_BROADCAST	Permission to transmit broadcast messages?
SO_OOBINLINE	Receive out-of-band data in band?
SO_SNDBUF	Buffer size for output
SO_RCVBUF	Buffer size for input
SO_TYPE	Type of the socket
SO_ERROR	Get and clear the last error for the socket

Return Code

Getsockopt returns *0* if successful and *−1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
ENOPROTOOPT	The *opt* is incorrect.
EFAULT	The address of *optval* or *optlen* is incorrect.

The Gettimeofday System Call

Use

retcode = gettimeofday (tm, tmzone);

Description

Gettimeofday extracts the current time and date from the system along with information about the local time zone.

Arguments

Arg	Type	Meaning
tm	&struct timeval	The address of a timeval structure.
tmzone	&struct timezone	The address of a timezone structure.

The structures that *gettimeofday* assigns are declared:

```
struct timeval {          /* structure for storing time   */
     long tv_sec;         /* seconds since epoch (1/1/70) */
     long tv_usec;        /* microseconds beyond tv_sec   */
};

struct timezone {         /* structure for timezone info  */
     int  tz_minuteswest;/* minutes west of Greenwich     */
     int  tz_dsttime;     /* type of correction to apply  */
};
```

Return Code

Gettimeofday returns *0* if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains the following value:

Value in errno	Cause of the Error
EFAULT	The *tm* or *tmzone* arguments contained an incorrect address.

The Listen System Call

Use

```
retcode = listen ( socket, queuelen );
```

Description

Servers use *listen* to make a socket passive (i.e., ready to accept incoming requests). *Listen* also sets the number of incoming connection requests that the protocol software should enqueue for a given socket while the server handles another request. *Listen* only applies to sockets used with TCP.

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* call.
queuelen	int	The size of the incoming connection request queue (usually up to a maximum of 5).

Return Code

Listen returns 0 if successful and -1 to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
EOPNOTSUPP	The socket type does not support *listen*.

The Read System Call

Use

retcode = read (socket, buff, buflen);

Description

Clients or servers use *read* to obtain input from a socket.

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* function.
buff	&char	A pointer to an array of characters to hold the input.
buflen	int	An integer that specifies the number of bytes in the the *buff* array.

Return Code

Read returns zero if it detects an end-of-file condition on the socket, the number of bytes read if it obtains input, and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
EFAULT	Address *buff* is illegal.
EIO	An I/O error occurred while reading data.
EINTR	A signal interrupted the operation.
EWOULDBLOCK	Nonblocking I/O is specified, but the socket has no data.

The Recv System Call

Use

```
retcode = recv ( socket, buffer, length, flags );
```

Description

Recv obtains the next incoming message from a socket.

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* function.
buffer	&char	The address of a buffer to hold the message.
length	int	The length of the buffer.
flags	int	Control bits that specify whether to receive out-of-band data and whether to look ahead for messages.

Return Code

Recv returns the number of bytes in the message if successful and -1 to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
EWOULDBLOCK	The socket has no data, but nonblocking I/O has been specified.
EINTR	A signal arrived before the read operation could deliver data.
EFAULT	Argument *buffer* is incorrect.

The Recvfrom System Call

Use

retcode = recvfrom (socket, buffer, buflen, flags, from, fromlen);

Description

Recvfrom extracts the next message that arrives at a socket and records the sender's address (enabling the caller to send a reply).

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* function.
buffer	&char	The address of a buffer to hold the message.
buflen	int	The length of the buffer.
flags	int	Control bits that specify out-of-band data or message look-ahead.
from	&sockaddr	The address of a structure to hold the sender's address.
fromlen	&int	The length of the *from* buffer, returned as the size of the sender's address.

Chapter 5 contains a description of the *sockaddr* structure.

Return Code

Recvfrom returns the number of bytes in the message if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
EWOULDBLOCK	The socket has no data, but nonblocking I/O has been specified.
EINTR	A signal arrived before the read operation could deliver data.
EFAULT	Argument *buffer* is incorrect.

The Recvmsg System Call

Use

retcode = recvmsg (socket, msg, flags);

Description

Recvmsg returns the next message that arrives on a socket. It places the message in a structure that includes a header along with the data.

Arguments

Arg	Type	Meaning
socket	int	Socket descriptor created by the *socket* function.
msg	&struct msghdr	Address of a message structure.
flags	int	Control bits that specify out-of-band data or message look-ahead.

The message is delivered in a *msghdr* structure with format:

```
struct msghdr  {
     caddr_t   msg_name;        /* optional address        */
     int       msg_namelen;     /* size of address         */
     struct iovec *msg_iov;     /* scatter/gather array    */
     int       msg_iovlen;      /* # elements in msg_iov */
     caddr_t   msg_accrights;   /* rights sent/received    */
     int       msg_accrghtslen;/* length of prev. field */
};
```

Return Code

Recvmsg returns the number of bytes in the message if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
EWOULDBLOCK	The socket has no data, but nonblocking I/O has been specified.
EINTR	A signal arrived before the read operation could deliver data.
EFAULT	Argument *buffer* is incorrect.

The Select System Call

Use

retcode = select (numfds, refds, wrfds, exfds, time);

Description

Select provides asynchronous I/O by permitting a single process to wait for the first of any file descriptors in a specified set to become ready. The caller can also specify a maximum timeout for the wait.

Arguments

Arg	Type	Meaning
numfds	int	Number of file descriptors in the set.
refds	&fd_set	Address of file descriptors for input.
wrfds	&fd_set	Address of file descriptors for output.
exfds	&fd_set	Address of file descriptors for exceptions.
time	&struct timeval	Maximum time to wait or zero.

Arguments that refer to descriptors consist of integers in which the i^{th} bit corresponds to descriptor *i*. Macros *FD_CLR* and *FD_SET* clear or set individual bits. The UNIX manual page that describes *gettimeofday* contains a description of the *timeval* structure.

Return Code

Select returns the number of ready file descriptors if successful, *0* if the time limit was reached, and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	One of the descriptor sets specifies an invalid descriptor.
EINTR	A signal arrived before the time limit or any of the selected descriptors became ready.
EINVAL	The time limit value is incorrect.

The Send System Call

Use

retcode = send (socket, msg, msglen, flags);

Description

Applications call *send* to transfer a message to another machine.

Arguments

Arg	Type	Meaning
socket	int	Socket descriptor created by the *socket* function.
msg	&char	A pointer to the message.
msglen	int	The length of the message in bytes.
flags	int	Control bits that specify out-of-band data or message look-ahead.

Return Code

Send returns the number of characters sent if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
EFAULT	Argument *buffer* is incorrect.
EMSGSIZE	The message is too large for the socket.
EWOULDBLOCK	The socket has no data, but nonblocking I/O has been specified.
ENOBUFS	The system had insufficient resources to perform the operation.

The Sendmsg System Call

Use

retcode = sendmsg (socket, msg, flags);

Description

Sendmsg sends a message, extracting it from a *msghdr* structure.

Arguments

Arg	Type	Meaning
socket	int	Socket descriptor created by the *socket* function.
msg	&struct msghdr	A pointer to the message structure.
flags	int	Control bits that specify out-of-band data or message look-ahead.

The page that describes *recvmsg* contains a definition of structure *msghdr*.

Return Code

Sendmsg returns the number of bytes sent if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
EFAULT	Argument *buffer* is incorrect.
EMSGSIZE	The message is too large for the socket.
EWOULDBLOCK	The socket has no data, but nonblocking I/O has been specified.
ENOBUFS	The system had insufficient resources to perform the operation.

The Sendto System Call

Use

```
retcode = sendto ( socket, msg, msglen, flags, to, tolen );
```

Description

Sendto sends a message by taking the destination address from a structure.

Arguments

Arg	Type	Meaning
socket	int	Socket descriptor created by the *socket* function.
msg	&char	A pointer to the message.
msglen	int	The length of the message in bytes.
flags	int	Control bits that specify out-of-band data or message look-ahead.
to	&sockaddr	A pointer to the address structure.
tolen	int	The length of the address in bytes.

Chapter 5 contains a description of the *sockaddr* structure.

Return Code

Sendto returns the number of bytes sent if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
EFAULT	Argument *buffer* is incorrect.
EMSGSIZE	The message is too large for the socket.
EWOULDBLOCK	The socket has no data, but nonblocking I/O has been specified.
ENOBUFS	The system had insufficient resources to perform the operation.

The Sethostid System Call

Use

(void) sethostid (hostid);

Description

The system manager runs a privileged program at system startup that calls *sethostid* to assign the local machine a unique *32*-bit host identifier. Usually, the host identifier is the machine's primary IP address.

Arguments

Arg	Type	Meaning
hostid	int	A value to be stored as the host's identifier.

Errors

An application must have root privilege, or *sethosid* will not change the host identifier.

The Setsockopt System Call

Use

retcode = setsockopt (socket, level, opt, optval, optlen);

Description

Setsockopt permits an application to change an option associated with a socket or the protocols it uses.

Arguments

Arg	Type	Meaning
socket	int	The descriptor of a socket.
level	int	An integer that identifies a protocol (e.g., TCP).
opt	int	An integer that identifies an option.
optval	&char	The address of a buffer that contains a value (usually *1* to enable an option or *0* to disable it).
optlen	int	The length of *optval*.

The socket-level options that apply to all sockets include:

SO_DEBUG	Toggle status of debugging information
SO_REUSEADDR	Toggle local address reuse
SO_KEEPALIVE	Toggle status of connection keep-alive
SO_DONTROUTE	Toggle bypass routing for outgoing messages
SO_LINGER	Linger on close if data present
SO_BROADCAST	Toggle permission to broadcast messages
SO_OOBINLINE	Toggle reception of out-of-band data in band
SO_SNDBUF	Set buffer size for output
SO_RCVBUF	Set buffer size for input

Return Code

Setsockopt returns *0* if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
ENOPROTOOPT	The option integer, *opt*, is incorrect.
EFAULT	The address of *optval* or *optlen* is incorrect.

The Shutdown System Call

Use

```
retcode = shutdown ( socket, direction );
```

Description

The *shutdown* function applies to full-duplex sockets (i.e., a connected TCP socket), and is used to partially close the connection.

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* call.
direction	int	The direction in which shutdown is desired: *0* means terminate further input, *1* means terminate further output, and *2* means terminate both input and output.

Return Code

The *shutdown* call returns *0* if the operation succeeds or *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains a code that specifies the cause of the error. The possible errors are:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
ENOTSOCK	The first argument does not specify a socket descriptor.
ENOTCONN	The specified socket is not currently connected.

The Socket System Call

Use

```
retcode = socket ( family, type, protocol );
```

Description

The *socket* function creates a socket used for network communication, and returns an integer descriptor for that socket.

Arguments

Arg	Type	Meaning
family	int	Protocol or address family (PF_INET for TCP/IP, AF_INET can also be used).
type	int	Type of service (SOCK_STREAM for TCP or SOCK_DGRAM for UDP).
protocol	int	Protocol number to use or 0 to request the default for a given family and type.

Return Code

The *socket* call either returns a descriptor or *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains a code that specifies the cause of the error. The possible errors are:

Value in errno	Cause of the Error
EPROTONOSUPPORT	Error in arguments: the requested service or the specified protocol is invalid.
EMFILE	The application's descriptor table is full.
ENFILE	The internal system file table is full.
EACCES	Permission to create the socket is denied.
ENOBUFS	The system has no buffer space available.

The Write System Call

Use

```
retcode = write ( socket, buf, buflen );
```

Description

Write permits an application to transfer data to a remote machine.

Arguments

Arg	Type	Meaning
socket	int	A socket descriptor created by the *socket* call.
buf	&char	The address of a buffer containing data.
buflen	int	The number of bytes in *buf*.

Return Code

Write returns the number of bytes transferred if successful and *-1* to indicate that an error has occurred. When an error occurs, the global variable *errno* contains one of the following values:

Value in errno	Cause of the Error
EBADF	The first argument does not specify a valid descriptor.
EPIPE	Attempt to *write* on an unconnected stream socket.
EFBIG	Data written exceeds system capacity.
EFAULT	Address in *buf* is incorrect.
EINVAL	Socket pointer is invalid.
EIO	An I/O error occurred.
EWOULDBLOCK	The socket cannot accept all data that was written without blocking, but nonblocking I/O has been specified.

Appendix 2

Manipulation Of UNIX File And Socket Descriptors

Introduction

In UNIX, all input and output operations use an abstraction known as the *file descriptor*. A program calls the *open* system call to obtain access to a file or the *socket* system call to obtain a descriptor used for network communication. Chapters *4* and *5* describe the socket interface; Chapter *23* describes UNIX descriptors and I/O in more detail. It points out that a newly created process inherits copies of all file descriptors that the parent process had open at the time of creation. Finally, Chapter *27* discusses how production servers close extra file descriptors and open standard I/O descriptors.

This appendix describes how programs can use standard I/O descriptors as arguments, and illustrates how a parent rearranges existing descriptors to make them correspond to the standard I/O descriptors before invoking a child. The technique is especially useful in multiservice servers that invoke separate programs to handle each service.

Descriptors As Implicit Arguments

When the UNIX *fork* function creates a new process, the newly created child inherits a copy of all file descriptors that the parent had open at the time of the call. Furthermore, the child's descriptor for a given file or socket appears in exactly the same position as the parent's. Thus, if descriptor *5* in the parent corresponds to a TCP socket, descriptor *5* in the newly created child will correspond to exactly the same socket.

Descriptors also remain open across a call to *execve*. To create a new process that executes the code in a file, *F*, a process calls *fork* and arranges for the child to call *execve* with file name *F* as an argument.

Conceptually, file descriptors form implicit arguments to newly created processes and to processes that overlay the running program with code from a file. The parent can choose which descriptors to leave open and which to close before a call to *fork*, and can choose exactly which descriptors will remain open across a call to *execve*.

UNIX programs often use descriptors to control processing instead of explicit arguments. In particular, a UNIX program expects three standard I/O descriptors to be open when the program begins: *standard input* (descriptor *0*), *standard output* (descriptor *1*), and *standard error* (descriptor *2*). The program reads input from descriptor *0*, writes output to descriptor *1*, and sends error messages to descriptor *2*.

Choosing To Use A Fixed Descriptor

A server can also use descriptors as implicit arguments if it invokes a separate program to handle a given request. For example, a slave program that is part of a connection-oriented server can be written to expect descriptor *0* to correspond to a TCP connection. The master server establishes a connection on descriptor *0*, and then uses *execve* to execute the slave. The master could also be programmed to use an arbitrary descriptor and to pass an argument to the slave that specifies which descriptor corresponds to the connection. However, using a fixed descriptor simplifies the code without sacrificing any functionality. In summary:

> *Processes use descriptors as implicit arguments in calls to execve. Master servers are often programmed to use a single descriptor as an implicit argument for the slave programs they invoke.*

The Need To Rearrange Descriptors

If a master server calls the system function *socket* to create a socket for connectionless communication or calls *accept* to obtain a socket for connection-oriented communication, the server cannot specify which descriptor the call will return because the operating system chooses a descriptor. If the slave expects descriptor *0* to correspond to the socket used for communication, the master cannot merely create a socket on a random descriptor and then execute the slave. Instead, it must rearrange its descriptors before making the call.

Rearranging Descriptors

A process uses the UNIX system calls *close* and *dup2* to rearrange its descriptors. Closing a descriptor detaches it from the file or socket to which it corresponds, deallocates any resources associated with it, and makes it available for reuse. For example, if a process needs to use descriptor *0*, but it is already in use, the process calls *close* to make it free. Appendix *1* describes how a server can close unnecessary file descriptors when it begins.

A process calls function *dup2* to create a duplicate copy of one file descriptor in another. The call has two arguments:

```
(void) dup2(olddesc, newdesc);
```

where *olddesc* is an integer that identifies an existing descriptor, and *newdesc* is an integer that identifies the descriptor where the copy should appear. If *newdesc* is currently in use, the system deallocates it as if the user had called *close* before duplicating *olddesc*. After *dup2* finishes, both *olddesc* and *newdesc* refer to the same object (e.g., the same TCP connection).

To "move" a socket from one descriptor to another, the program first calls *dup2* and then calls *close* to deallocate the original. For example, suppose a master server calls *accept* to obtain an incoming connection, and suppose *accept* chooses to use descriptor *5* for the new connection. The master server can issue the following two calls to move the socket to descriptor *0*:

```
(void) dup2(5, 0);
(void) close(5);
```

Close-On-Exec

A server can choose to close unneeded file descriptors before executing another program. One approach requires the server to explicitly call *close* for each unneeded file descriptor. Another approach uses a system facility to close descriptors automatically.

To use the automatic facility, a server must set the *close-on-exec* flag in each descriptor that it wants the system to close. When the server calls *execve*, the system checks each descriptor to see if the flag has been set and calls *close* automatically if it has. While it may seem that automatic closing does not make programming easier, one must remember that in a large, complex program the purpose of each descriptor may not be apparent at the point of the call to *execve*. Using *close-on-exec* permits the programmer to decide whether a descriptor should remain open at the point in the program where the descriptor is created. Thus, it can be quite useful.

Summary

Processes use descriptors as implicit arguments when creating a new process or when overlaying the running program with code from a file. Often, servers that execute separately compiled programs rely on a descriptor to pass a TCP connection or a UDP socket.

To make programs easier to write and maintain, programmers usually choose a fixed descriptor for each implicit argument. The caller must rearrange its descriptors before invoking *execve* so that the socket used for I/O corresponds to the descriptor chosen as an implicit argument.

A process uses the *dup2* and *close* system calls to rearrange its descriptors. *Dup2* copies an existing descriptor into a specified location, and *close* deallocates a specified descriptor. To move descriptor *A* to descriptor *B*, a process first calls *dup2(A, B)* and then calls *close(A)*.

FOR FURTHER STUDY

The *UNIX Programmer's Manual* contains further information on the *dup2* and *close* system calls.

EXERCISES

A2.1 Read about the UNIX command interpreter. Which descriptors does it assign before executing a command?

A2.2 Sketch the algorithm a command interpreter uses to handle file redirection. Show how it moves file descriptors before calling *execve*.

A2.3 Modify the multiservice server from Chapter *14* so it uses descriptor *0* as an implicit argument and calls *execve* to execute a slave program.

A2.4 Read about the BSD UNIX program *inetd*. How does it use descriptors as implicit arguments?

A2.5 What does the call *dup2(n, m)* do if *n* already equals *m*? Write a version of *dup2* that works for arbitrary values of *n* and *m*.