Overview of Socket Interface

- Client-server model
  - local system (a client)
  - remote system (a server)
- These two machines need to communicate with one another
- socket provide a standard interface, API, for programmer to write client-server applications
  - declarations
  - definitions
  - procedures
- There are several APIs
  - socket interface
  - TLI (transport level interface)
  - RPC (remote procedure call)

Data types and Data structures

- 3 data types commonly used
  - u_char unsigned 8-bit character
  - u_short unsigned 16-bit integer
  - u_long unsigned 32-bit integer

  - This are available in <sys/types.h>

Internet Socket Address Structure

- TCP/IP protocol suite need a structure called socket address to holds IP address, port number, protocol type and protocol family
- A socket is an abstract representation of a communication endpoint.
  - acts a an end-port
  - two processes need a socket at each end to communicate with each other
- Sockets work with Unix I/O services just like files, pipes & FIFOs.
- Socket structure
  - family
  - type
  - protocol
  - local socket address
  - remote socket address

Socket Addresses

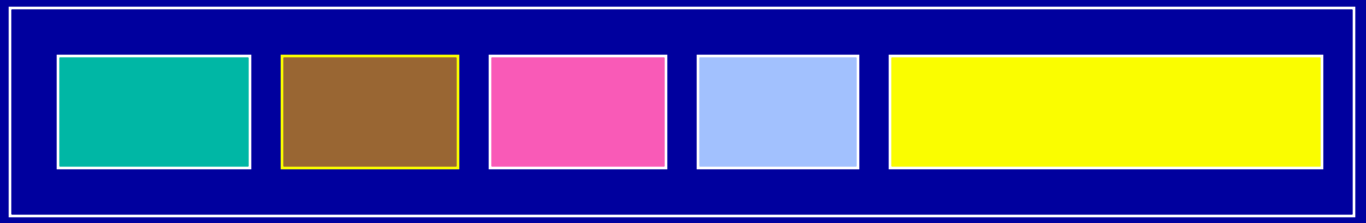
- Internet Address
  - IPv4 define a structure call **in_addr**
  - 32 bit binary number
  - struct in_addr

```
{
   u_long s_addr;
}
```

Internet socket address structure

- Struct sockaddr_in

```
{
    u_char          sin_len;
    u_short         sin_family;
    u_short         sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
}
```



sin_len sin_family sin_port sin_addr sin_zero

- For the Unix domain, the following structure is defined in <sys/un.h>

```
struct sockaddr_un{
short sun_family;          //AF_UNIX
char sun_path[108];     // Pathname
};
```

➢For the Xerox Ns family, the following structures are defined in <netns/ns.h>

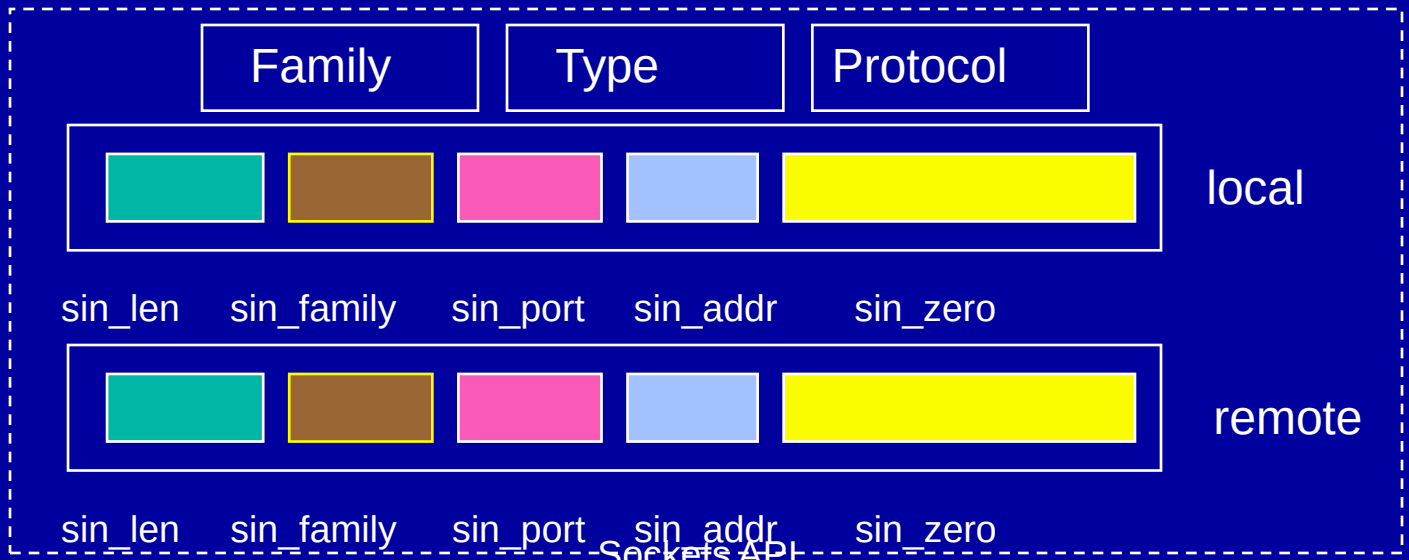
```
struct sockaddr_ns{
        u_short sns_family;
        struct ns_addr  sns_addr;
        char sns_zero[2];
    };

 struct ns_addr{
                        union ns_net  x_net;
                        union  x_host;
                        u_short  x_port;  };
```

```
union ns_host{
    u_char c_host[6];     //hostid addr as six bytes
    u_short s_host[3];   //hostid addr as three 16-bit shorts
    };
union ns_net{
    u_char c_net[4];  // netid as four bytes
    u_short s_net[2]; // netid as two 16-bit shorts
};
```

Socket structure

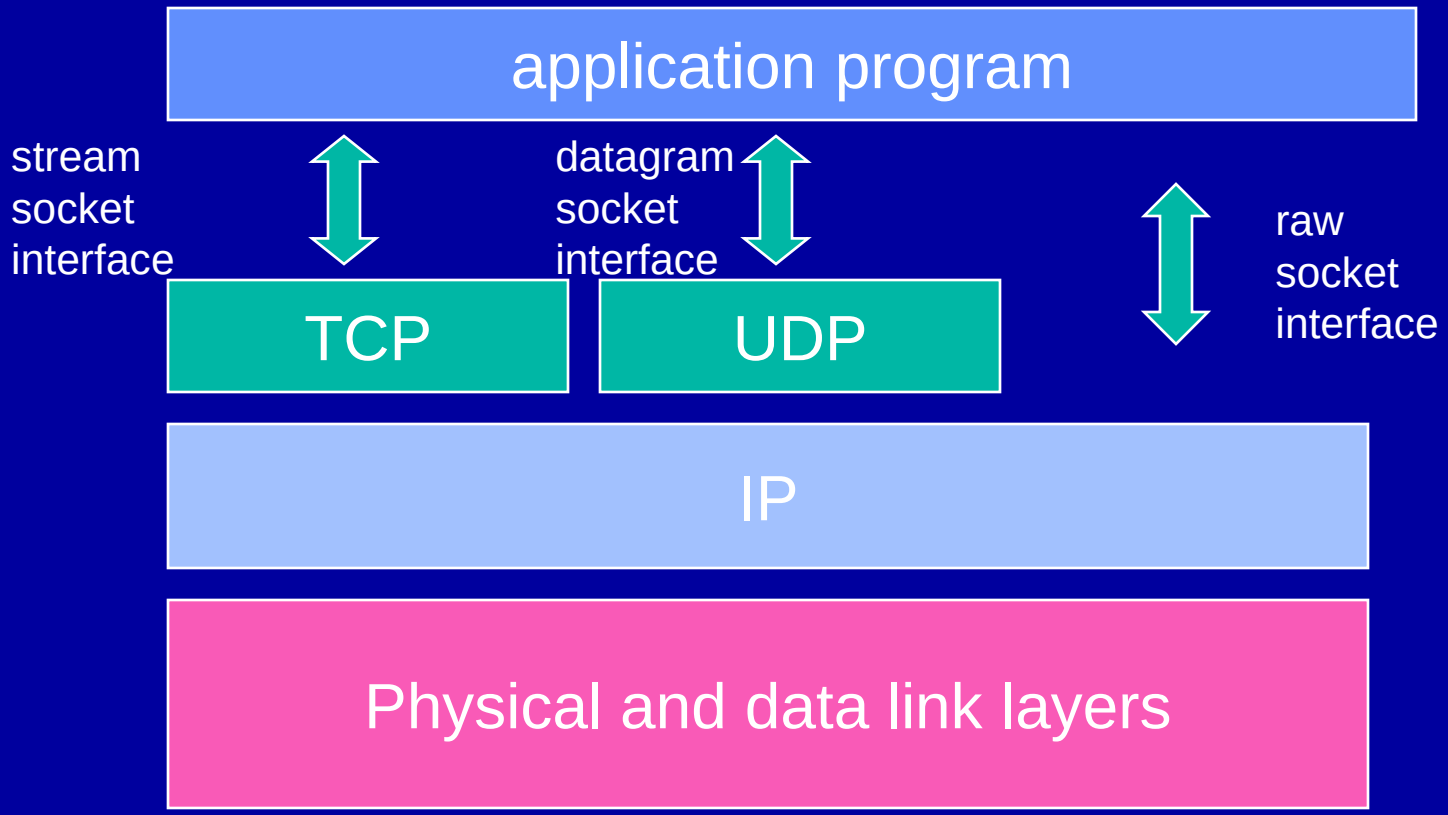
- Socket structure (it is a bigger picture!!)
  - family
  - type
  - protocol
  - local socket address
  - remote socket address



Socket types

- three types
  - stream socket
    - connection-oriented protcol such as TCP
    - TCP uses a pair of stream sockets to connect one application program to anther over the Internet
  - datagram socket
    - connectionless protocol such as UDP
    - UDP uses a pair of datagram sockets to send message from one applications program to another over the Internet
  - raw socket
    - designed for services and application such as ICMP or OSPF

Socket types



Socket family

- AF_UNIX Unix internal protocols
- AF_INET Internet protocols
- AF_NS Xerox NS protocols
- AF_IMPLINK IMP link layer

Combination of family, type and protocol

Family	Type	Protocol	Actual protocol
AF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
AF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
AF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
AF_INET	SOCK_RAW	IPPROTO_RAW	(raw)
AF_NS	SOCK_STREAM	NSPROTO_SPP	SPP
AF_NS	SOCK_SEQPACKET	NSPROTO_SPP	SPP
AF_NS	SOCK_RAW	NSPROTO_ERROR	Error protocol
AF_NS	SOCK_RAW	NSPROTO_RAW	(raw)

socket system calls: socket

- used by a process to create a socket
- allocates resources needed for a communication endpoint - but it does not deal with endpoint addressing.
- int socket (int family, int type, int protocol)
  - input
    - family: AF_INET
      - specifies the protocol family
    - type: SOCK_DGRAM or SOCK_STREAM
      - specifies the type of service
    - protocol: 0
      - specifies the specific protocol
  - output
    - returns a socket descriptor if successful
    - -1 if error

socket system calls: socketpair

- It is similar to the pipe systen call, but socketpair returns a pair of socket descriptors, not file descriptors.
- The two socket descriptors returned by socketpair are bidirectional, unlike pipes which are unidirectional.

int socketpair (int family, int type, int protocol, int sockvec[2])

--Returns two socket descriptors, sockvec[0] and sockvec[1].

socket system calls: bind

- bind function binds a socket to a local socket address by adding the local socket address to an already create socket.
- int bind (int sockfd, const struct sockaddr_in *localaddr, int localaddrlen);
- input
  - sockfd
  - localaddr
  - localaddrlen
- output
  - return 0 if successful; -1 if error

`bind()` Example

```
int mysock,err;
struct sockaddr_in myaddr;

  mysock = socket(PF_INET,SOCK_STREAM,0);
  myaddr.sin_family = AF_INET;
  myaddr.sin_port = htons( portnum );
  myaddr.sin_addr = htonl( ipaddress);

  err=bind(mysock, (sockaddr *) &myaddr,
                  sizeof(myaddr));
```

Uses for `bind()`

- There are a number of uses for **bind()**:
  - Server would like to bind to a well known address (port number).
  - Client can bind to a specific port.
  - Client can ask the O.S. to assign *any available* port number.

socket system calls: connect

- connect function is used by a process (usually a client) to establish an active connection to remote process (normally a server).
- int connect (int sockfd, const struct scokaddr_in *serveraddr, int serveraddrlen);
- Input
  - sockfd: socket descriptor
  - serveraddr: pointer to the remote socket address
  - serveraddrlen: length of that address
- output
  - returns 0 if successful
  - -1 if error

socket system calls: listen

- is called by the TCP server
- creates a passive socket from an unconnected socket
- informs the OS that the server is ready to accept connection thru' this socket
- int listen(int sockfd, int backlog);
  - sockfd: socket descriptor
  - backlog: number of requests that can be queued for this connection
- output
  - return 0 if successful
  - -1 if error

socket system calls: accept

- called by a TCP server to remove the first connection request from the corresponding queue
- if there is no request (I.e. the queue is empty), the accept function is put to sleep
- int accept(int sockfd, const struct sockaddr_in *clientaddr, int *clientaddrlen);
- input
  - sockfd: socket descriptor
  - clientaddr: pointer to the address of client
  - clientaddrlen: pointer to the client address length
- output
  - return a socket descriptor
  - -1 if error

socket system calls: sendto

- used by a process to send a UDP message to another process usually running on a remote machine
- int sendto(int sockfd, const void *buf, int buflen, int flags, const struct sockaddr_in *toaddr, int toaddrlen);
- input
  - sockfd: socket descriptor
  - buf: a pointer to the buffer holding the message to be sent
  - buflen: defines the length of the buffer
  - flags: specifies out-of-band data or lookahead messages
  - toaddr: a pointer to the socket address of the receiver
  - toaddrlen: length of the socket address
- output
  - number of characters sent if no error
  - -1 if error

socket system calls: recvfrom

- extracts the next message that arrives at a socket
- extracts the sender's socket address
- int recvfrom(int scokfd, const void *buf, int buflen, int flags, const struct sockaddr_in, *from addrlen);
- input
  - sockfd: socket descriptor
  - buf: a pointer to the buffer where the message will be stored
  - buflen: defines the length of the buffer
  - flags: specifies out-of-band data or look ahead messages
  - fromaddr: a pointer to the socket address of the sender
  - fromaddrlen: length of the socket address
- output
  - number of bytes received if no er2323ror
  - -1 if error

socket system calls: read

- used by a process to receive data from another process running on a remote machine
- assumes that there is already an open connection between two machines
- int read(int sockfd, const void *buf, int buflen);
- input
  - sockfd: socket descriptor
  - buf: a pointer to the buffer where the message will be stored
  - buflen: defines the length of the buffer
- output
  - returns number of bytes read if no error
  - -1 if error

socket system calls: write

- used by a process to send data to another process running on a remote machine
- assumes that there is already an open connection between two machines
- int write(int sockfd, const void *buf, int buflen);
- input
  - sockfd: socket descriptor
  - buf: a pointer to the buffer where the message will be stored
  - buflen: defines the length of the buffer
- output
  - returns number of bytes read if no error
  - -1 if error

socket system calls: close

- used by a process to close a socket and terminate a TCP connection
- int close(int sockfd);
- input
  - sockfd: socket descriptor
- output
  - returns 0 if no error
  - -1 if error

overview of API functions

- socket system calls (socket, bind, connect, listen, accept, sendto, recvfrom, read, write, close)
- byte order transformation (htons, htonl, ntohs and ntohl)
- address transformation (inet_aton, inet_ntoa)
- byte manipulation (memset, memcpy, memcmp)
- get information about remote host (gethostbyname)

Byte ordering

- two types
  - big-endian
    - MSB first
  - little-endian
    - LSB first
- IP address use big-endian
  - MSB first
- To create portability in applications program, API provides a set of functions that transforms integers from a host byte order (big endian or little endian) to network type order (big endian)
- Four functions are provided
  - htons, htonl, ntohs and ntohl

Byte order transformations

- The four functions handle the potential byte order differences between different computer architectures and different network protocols.

u_long htonl (u_long *hostlong*);

--- function converts the unsigned integer hostlong from host byte order to network byte order.

u_short htons (u_short *hostshort*);

---function converts the unsigned short integer hostshort from host byte order to network byte order.

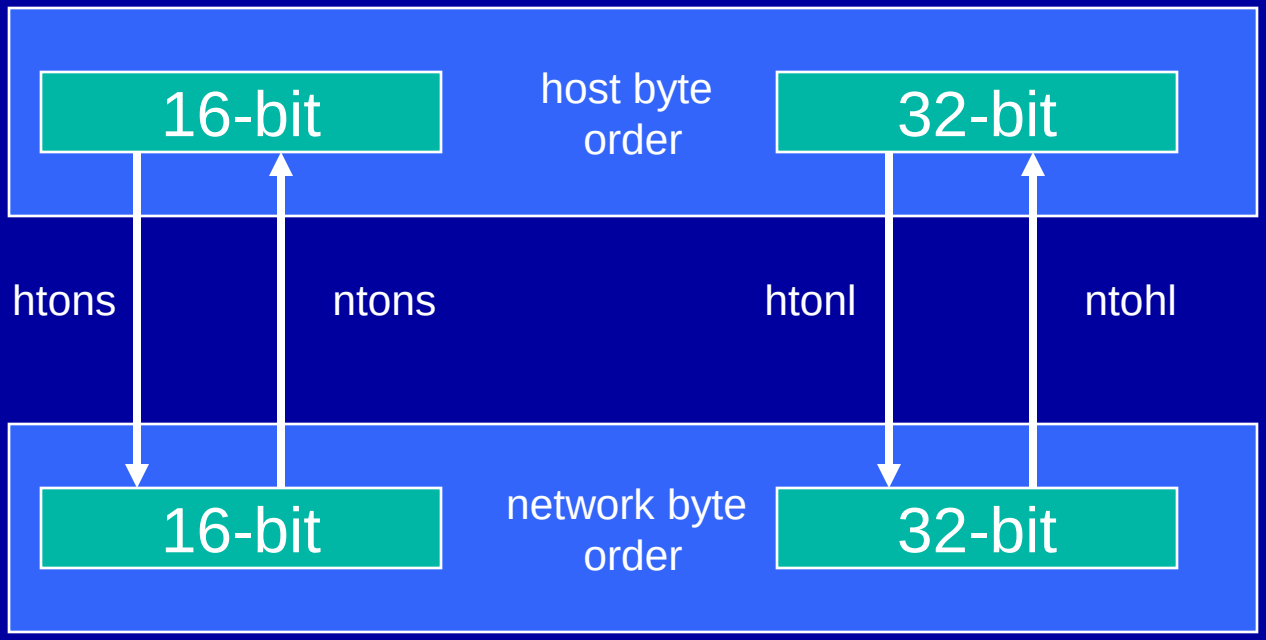
u_long ntohl (u_long *netlong*);

---function converts the unsigned integer netlong from network byte order to host byte order.

u_short ntohs (u_short *netshort*);

---function converts the unsigned short integer netshort from network byte order to host byte order.

Byte order transformation



Address Transformation

int inet_aton(const char *cp, struct in_addr *inp);

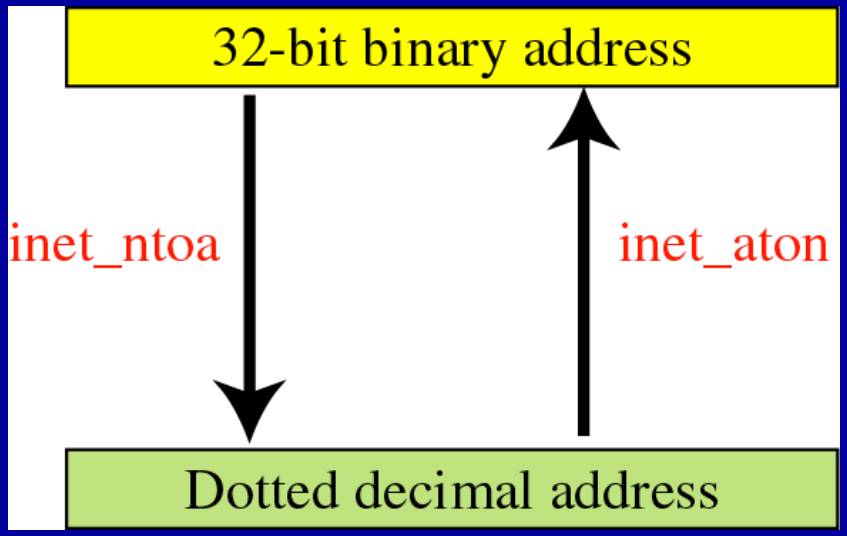
---inet_aton() converts the Internet host address cp from the standard numbers-and-dots notation into binary data and stores it in the struc-ture that inp points to. inet_aton returns nonzero if the address is valid, zero if not.

unsigned long inet_addr (char *ptr);

---The inet_addr() function converts the Internet host address cp from numbers-and-dots notation into binary data in **network byte order**. If the input is invalid, INADDR_NONE (usually -1) is returned. This is an obsolete interface to inet_aton, described immediately above; it is obsolete because -1 is a valid address (255.255.255.255), and inet_aton provides a cleaner way to indicate error return.

char *inet_ntoa(struct in_addr in);

---The inet_ntoa() function converts the Internet host address in given in network byte order to a string in standard numbers-and-dots notation. The string is returned in a statically allocated buffer, which subse-quent calls will overwrite.


32-bit binary address
inet_ntoa
inet_aton
Dotted decimal address

Advanced Socket System Calls

- readv and writev
- sendmsg and recvmsg
- getpeername
- getsockname
- getsockopt and setsockopt
- select

System Calls: readv and writev

- readv, writev - read or write a vector—Scatter read—Gather write

- SYNOPSIS
- #include <sys/uio.h>

```
ssize_t readv(int fd, const struct iovec *vector, int count);
ssize_t writev(int fd, const struct iovec *vector, int count);

struct iovec {
    void *iov_base;    /* Starting address */
    size_t iov_len;    /* Length in bytes  */
};
```

- readv reads data from file descriptor fd, and puts the result in the buffers described by vector. The number of buffers is specified by count. The buffers are filled in the order specified. Operates just like read except that data is put in vector instead of a contiguous buffer.

- writev writes data to file descriptor fd, and from the buffers described by vector. The number of buffers is specified by count. The buffers are used in the order specified. Operates just like write except that data is taken from vector instead of a contiguous buffer.

System Calls : sendmsg and recvmsg

- int sendmsg (int sockfd, struct msghdr msg[], int flags);
- int recvmsg (int sockfd, struct msghdr msg[], int flags);

- Struct msghdr{

```
        caddr_t msg_name;              //optional address
        int mag_namelen;               //size of address
        struct iovec *msg_iov;         //scatter and gather array
        int msg_iovlen;                //elements in msg_iov
        caddr_t msg_accrights;     // access rights send and recv
        int msg_accrightslen;
};
```

System Calls : sendmsg and recvmsg

- msg_name and msg_namelen members are used when the socket is not connected(Ex:UDP)
- msg_iov and msg_iovlen members specify the array of input or output buffers.
- msg_control and msg_controllen members specify the location and size of the optional data.

System Call: getpeername

- getpeername - get name of connected peer socket
- SYNOPSIS

  #include <sys/socket.h>

  int getpeername(int s, struct sockaddr *name, socklen_t *namelen);

  – Getpeername returns the name of the peer connected to socket s. The namelen parameter should be initialized to indicate the amount of space pointed to by name. On return it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

System Calls: getsockopt and setsockopt

getsockopt, setsockopt - get and set options on sockets

- int getsockopt(int s, int level, int optname, void *optval, socklen_t *optlen);
- int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);

- S—socket fd
- Level--- specifies whether the option is a general option or a protocol specific option
- optval & optlen --- The parameters optval and optlen are used to access option values for setsockopt. For getsockopt they identify a buffer in which the value for the requested option(s) are to be returned.
- optname – any specified option

Socket Options

- Various attributes that are used to determine the behavior of sockets.
- Setting options tells the OS/Protocol Stack the behavior we want.
- Support for generic options (apply to all sockets) and protocol specific options.

Option types

- Many socket options are Boolean flags indicating whether some feature is enabled (1) or disabled (0).

- Other options are associated with more complex types including **`int`**, **`timeval`**, **`in_addr`**, **`sockaddr`**, etc.

General Options

- Protocol independent options.
- Handled by the generic socket system code.
- Some general options are supported only by specific types of sockets (SOCK_DGRAM, SOCK_STREAM).

Some Generic Options

SO_BROADCAST

SO_DONTROUTE

SO_ERROR

SO_KEEPALIVE

SO_LINGER

SO_RCVBUF,SO_SNDBUF

SO_REUSEADDR

SO_BROADCAST

- Boolean option: enables/disables sending of broadcast messages.
- Underlying DL layer must support broadcasting!
- Applies only to SOCK_DGRAM sockets. And only on networks that support the concept of broadcast message.

SO_DONTROUTE

- Boolean option: enables bypassing of normal routing.
- Used by routing daemons.

SO_ERROR

- Integer value option.
- The value is an error indicator value (similar to **errno**).
- Holds standard unix error numbers

.

SO_KEEPALIVE

- Enables periodic transmissions on a connected socket, when no other data is exchanged
- If the other end does not respond the connection is broken and SO_ERROR variable is set.

SO_LINGER

Value is of type:

```
struct linger {
   int l_onoff;   /* 0 = off */
   int l_linger;  /* time in seconds */
};
```

- Used to control whether and how long a call to close will wait for pending ACKS.
- connection-oriented sockets only.

SO_LINGER usage

- By default, calling **`close()`** on a TCP socket will return immediately.
- The closing process has no way of knowing whether or not the peer received all data.
- Setting SO_LINGER means the closing process can determine that the peer machine has received the data (but not that the data has been **`read()`** !).

SO_RCVBUF
SO_SNDBUF

- Integer values options - change the receive and send buffer sizes.
- Can be used with STREAM and DGRAM sockets.
- With TCP, this option effects the window size used for flow control - must be established before connection is made.

SO_REUSEADDR

- Boolean option: enables binding to an address (port) that is already in use.

- Used by servers that are transient - allows binding a passive socket to a port currently in use (with active sockets) by other processes.

SO_REUSEADDR

- Can be used to establish separate servers for the same service on different interfaces (or different IP addresses on the same interface).

- Virtual Web Servers can work this way.

IP Options (IPv4)

- IP_HDRINCL: used on raw IP sockets when we want to build the IP header ourselves.
- IP_TOS: allows us to set the “Type-of-service” field in an IP header.
- IP_TTL: allows us to set the “Time-to-live” field in an IP header.

TCP socket options

- TCP_KEEPALIVE: set the idle time used when SO_KEEPALIVE is enabled.
- TCP_MAXSEG: set the maximum segment size sent by a TCP socket.

another TCP socket option

- TCP_NODELAY: that delays sending small packets if there is unACK'd data pending.

- TCP_NODELAY also disables delayed ACKS (TCP ACKs are cumulative).