# Socket Programming Function Reference

## Network Address Conversion Functions

### inet_pton()

```
int inet_pton(int af, const char *src, void *dst);
```

- **Parameters:**
  - `af`: Address family (AF_INET for IPv4, AF_INET6 for IPv6)
  - `src`: Source address string in presentation format (e.g., “192.168.1.1”)
  - `dst`: Destination buffer for network address in binary form
- **Returns:**
  - Success: 1 (address converted successfully)
  - Invalid format: 0
  - Error: -1 and sets errno
- **Common use:**

  ```
  struct sockaddr_in addr;
  inet_pton(AF_INET, "192.168.1.1", &(addr.sin_addr));
  ```

### inet_ntop()

```
const char *inet_ntop(int af, const void *src, char *dst, socklen_t
size);
```

- **Parameters:**
  - `af`: Address family (AF_INET for IPv4, AF_INET6 for IPv6)
  - `src`: Network address in binary form
  - `dst`: Destination buffer for string
  - `size`: Size of destination buffer (use INET_ADDRSTRLEN for IPv4, INET6_ADDRSTRLEN for IPv6)
- **Returns:**
  - Success: Pointer to destination string
  - Error: NULL and sets errno
- **Common use:** `c char ip_str[INET_ADDRSTRLEN]; inet_ntop(AF_INET, &(addr.sin_addr), ip_str, INET_ADDRSTRLEN);` printf(“received message : %s”, buffer);
- **Note:** These functions are preferred over older inet_addr() and inet_ntoa() as they support both IPv4 and IPv6

## Socket Address Structures

### struct sockaddr

```
struct sockaddr {
    sa_family_t sa_family;     // Address family (AF_INET, AF_INET6, 
etc.)
    char        sa_data[14];   // Protocol-specific address 
information
};
```

- **Purpose:** Generic socket address structure
- **Usage:** Used as a generic type for socket functions
- **Note:** Never used directly, cast from specific address types

### struct sockaddr_in (IPv4)

```
struct sockaddr_in {
    sa_family_t    sin_family; // Address family: AF_INET
    in_port_t      sin_port;   // Port number in network byte order
    struct in_addr sin_addr;   // IPv4 address
    char           sin_zero[8]; // Padding to match sockaddr size
};

struct in_addr {
    uint32_t       s_addr;     // IPv4 address in network byte order
};
```

- **Purpose:** IPv4 socket address structure
- **Usage:** Most common structure for IPv4 networking
- **Note:** All members except sin_family must be in network byte order

### struct sockaddr_in6 (IPv6)

```
struct sockaddr_in6 {
    sa_family_t     sin6_family;   // Address family: AF_INET6
    in_port_t       sin6_port;     // Port number in network byte 
order
    uint32_t        sin6_flowinfo; // IPv6 flow information
    struct in6_addr sin6_addr;     // IPv6 address
    uint32_t        sin6_scope_id; // Scope ID
};

struct in6_addr {
    unsigned char   s6_addr[16];   // IPv6 address
};
```

- **Purpose:** IPv6 socket address structure
- **Usage:** Used for IPv6 networking
- **Note:** All members except sin6_family must be in network byte order

## Socket Creation and Basic Setup Group

### socket()
```
int socket(int domain, int type, int protocol);
```

- **Parameters:**
  - `domain`: Protocol family (AF_INET for IPv4, AF_INET6 for IPv6, AF_UNIX for local)
  - `type`: Socket type (SOCK_STREAM for TCP, SOCK_DGRAM for UDP, SOCK_RAW for raw sockets)
  - `protocol`: Protocol (usually 0, or IPPROTO_TCP, IPPROTO_UDP, etc.)
- **Returns:**
  - Success: File descriptor (positive integer)
  - Error: -1 and sets errno
- **Common errno values:**
  - EACCES: Permission denied
  - EMFILE: Process file table overflow
  - ENFILE: System file table overflow
  - EPROTONOSUPPORT: Protocol not supported

### fcntl()
```
int fcntl(int fd, int cmd, ... /* arg */ );
```

- **Parameters:**
  - `fd`: File descriptor
  - `cmd`: Command (F_GETFL, F_SETFL most common for sockets)
  - `arg`: Argument depending on cmd
- **Returns:**
  - Success: Depends on cmd (for F_GETFL/F_SETFL, >= 0)
  - Error: -1 and sets errno
- **Common flags:**
  - O_NONBLOCK: Set non-blocking mode
  - O_ASYNC: Enable signal-driven I/O
- **Usage pattern:**

```
// Set non-blocking mode
int flags = fcntl(sockfd, F_GETFL, 0);
fcntl(sockfd, F_SETFL, flags | O_NONBLOCK);
```

## Server-Side Connection Group

### bind()
```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

- **Parameters:**
  - `sockfd`: Socket file descriptor

    - `addr`: Address structure (sockaddr_in for IPv4, sockaddr_in6 for IPv6)
    - `addrlen`: Size of address structure
- **Returns:**
    - Success: 0
    - Error: -1 and sets errno
- **Common errno values:**
    - EADDRINUSE: Address already in use
    - EACCES: Permission denied

#### listen()

```
int listen(int sockfd, int backlog);
```

- **Parameters:**
    - `sockfd`: Socket file descriptor
    - `backlog`: Maximum length of pending connections queue
- **Returns:**
    - Success: 0
    - Error: -1 and sets errno
- **Common backlog values:**
    - SOMAXCONN: System maximum (often 128 or higher)

#### accept()

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

- **Parameters:**
    - `sockfd`: Listening socket descriptor
    - `addr`: Will contain client address (can be NULL)
    - `addrlen`: Size of addr structure (input/output parameter)
- **Returns:**
    - Success: New socket descriptor for accepted connection
    - Error: -1 and sets errno
- **Common errno values:**
    - EAGAIN/EWOULDBLOCK: No pending connections (non-blocking mode)
    - ECONNABORTED: Connection aborted
    - EMFILE: Too many open files

### Client-Side Connection Group

#### connect()

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t
addrlen);
```

- **Parameters:**
    - `sockfd`: Socket file descriptor

    - `addr`: Server address structure
    - `addrlen`: Size of address structure
- **Returns:**
    - Success: 0
    - Error: -1 and sets errno
- **Common errno values:**
    - EINPROGRESS: Connection in progress (non-blocking mode)
    - ECONNREFUSED: Connection refused
    - ETIMEDOUT: Connection timed out

## Data Transfer Group

### send()

```
ssize_t send(int sockfd, const void *buf, size_t len, int flags);
```

- **Parameters:**
    - `sockfd`: Socket descriptor
    - `buf`: Data buffer
    - `len`: Buffer length
    - `flags`: Send flags
- **Returns:**
    - Success: Number of bytes sent (may be less than len)
    - Error: -1 and sets errno
- **Common flags:**
    - MSG_NOSIGNAL: Don't generate SIGPIPE
    - MSG_DONTWAIT: Non-blocking operation
    - MSG_MORE: More data coming (TCP_CORK)

### recv()

```
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

- **Parameters:**
    - `sockfd`: Socket descriptor
    - `buf`: Buffer for received data
    - `len`: Buffer size
    - `flags`: Receive flags
- **Returns:**
    - Success: Number of bytes received (0 means EOF)
    - Error: -1 and sets errno
- **Common flags:**
    - MSG_PEEK: Peek at incoming data
    - MSG_DONTWAIT: Non-blocking operation
    - MSG_WAITALL: Wait for full request

### sendto() (Primarily for UDP)

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

- **Parameters:**
  - All send() parameters plus:
  - `dest_addr`: Destination address
  - `addrlen`: Address structure size
- **Returns:** Same as send()

### recvfrom() (Primarily for UDP)

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```

- **Parameters:**
  - All recv() parameters plus:
  - `src_addr`: Source address will be stored here
  - `addrlen`: Address structure size (input/output)
- **Returns:** Same as recv()

## Multiplexing Group (select() and related)

### FD_ZERO()

```
void FD_ZERO(fd_set *set);
```

- **Parameters:**
  - `set`: fd_set to initialize
- **Purpose:** Clears all file descriptors from set

### FD_SET()

```
void FD_SET(int fd, fd_set *set);
```

- **Parameters:**
  - `fd`: File descriptor to add
  - `set`: fd_set to modify
- **Purpose:** Adds fd to set

### FD_ISSET()

```
int FD_ISSET(int fd, fd_set *set);
```

- **Parameters:**
  - `fd`: File descriptor to test
  - `set`: fd_set to check
- **Returns:**
  - Non-zero if fd is in set
  - 0 if fd is not in set

### select()

```c
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

- **Parameters:**
  - `nfds`: Highest fd plus 1
  - `readfds`: Set of fds to check for reading
  - `writefds`: Set of fds to check for writing
  - `exceptfds`: Set of fds to check for exceptions
  - `timeout`: Maximum time to wait
- **Returns:**
  - Success: Number of ready file descriptors
  - Timeout: 0
  - Error: -1 and sets errno
- **Common timeout values:**
  - NULL: Wait indefinitely
  - {0,0}: Immediate return (polling)
- **Usage pattern:**

```c
fd_set readfds;
struct timeval tv;

while(1) {
    FD_ZERO(&readfds);
    FD_SET(sockfd, &readfds);

    tv.tv_sec = 5;  // 5 second timeout
    tv.tv_usec = 0;

    int ready = select(sockfd + 1, &readfds, NULL, NULL, &tv);
    if (ready > 0 && FD_ISSET(sockfd, &readfds)) {
        // Socket is ready for reading
    }
}
```

## Common Error Values

- EAGAIN/EWOULDBLOCK: Operation would block (non-blocking mode)
- EINTR: Interrupted system call
- EINPROGRESS: Operation now in progress
- ECONNRESET: Connection reset by peer
- EPIPE: Broken pipe