Contents:

socket ()

Create an endpoint for communication.

#include <sys/socket.h>

int socket(int *domain*, **int** *type*, **int** *protocol***);**

RETURN VALUE

On success, a file descriptor for the new socket is returned. On error, -1 is returned, and *errno* is set appropriately.

ARGUMENTS:

domain = AF_INET
AF_INET stands for IPv4. Other options can be AF_INET6 for IPv6 etc.

type ∈ {SOCK_DGRAM, SOCK_STREAM, SOCK_RAW etc.}
SOCK_DGRAM Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK_STREAM Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.

SOCK_RAW Provides raw network protocol access.

protocol = 0 Always.

bind ()

bind a name to a socket.

#include <sys/socket.h>

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

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ARGUMENTS:

sockfd = socket file descriptor obtained from the socket ().

addr = An IPv4 in our case.

addrlen = Sizeof the *addr* argument in bytes.

listen ()

listen for connections on a socket.

#include <sys/socket.h>

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

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ARGUMENTS:

sockfd = File descriptor obtained from the socket ().
sockfd will be marked as a *passive* socket, that is, as a socket that will be used to accept incoming connection requests using accept(2)

backlog = Maximum length to which the queue of pending connections for *sockfd*.
Set to SOMAXCONN**.**

accept ()

accept a connection on a socket.

#include <sys/socket.h>

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

RETURN VALUE

On success, these system calls return a nonnegative integer that is a descriptor for the accepted socket. On error, -1 is returned, and errno is set appropriately.

ARGUMENTS:

sockfd = File descriptor obtained from the socket ().
It extracts the first connection request on the queue of pending connections for the listening socket, *sockfd*, creates a *new connected* socket, and returns a new file descriptor referring to that socket.

addr = This structure is filled in with the address of the peer socket, as known to the communications layer.

addrlen = Length of the address stored in *addr*.

close ()

Close a file descriptor.

#include <unistd.h>

int close(int *fd***);**

RETURN VALUE

close() returns zero on success. On error, -1 is returned, and *errno* is set appropriately.

ARGUMENTS:

fd = File descriptor to be closed.

connect ()

Initiate a connection on a socket.

#include <sys/socket.h>

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

RETURN VALUE

If the connection or binding succeeds, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

ARGUMENTS:

sockfd = connects the socket referred to by the file descriptor *sockfd* to the address specified by *addr*.

addr = This structure is filled in with the address of the peer socket, as known to the communications layer.

addrlen = Length of the address stored in *addr*.

send() and sendto ()

Send a message on a socket.

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

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

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

RETURN:

On success, these calls return the number of bytes sent. On error, -1 is returned, and *errno* is set appropriately.

ARGUMENTS:

sockfd:	File descriptor of the sending socket.
buf:	A buffer containing the message.
len:	Size of the buffer.
flags:	Set it as 0 initially. Check manual for details (man 2 send).
dest_addr:	Address of the destination peer.
addr_len:	Sizeof the address specified by *dest_addr.*

recv () and recvfrom ()

Receive a message from a socket.

```
#include <sys/types.h>
#include <sys/socket.h>

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

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

RETURN:

These calls return the number of bytes received, or -1 if an error occurred. In the event of an error, *errno* is set to indicate the error.

ARGUMENTS:

sockfd:	File descriptor of the socket to read from.
buf:	An empty buffer to store the read message.
len:	Size of the buffer.
flags:	Set it as 0 initially. Check manual for details (man 2 recv).
src_addr:	Address of the source peer.
addr_len:	Sizeof the address specified by *src_addr.*

inet_ntop ()

convert IPv4 and IPv6 addresses from binary to text form.

```
#include <arpa/inet.h>

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

RETURN:

On success, **inet_ntop**() returns a non-null pointer to *dst*. NULL is returned if there was an error, with *errno* set to indicate the error.

ARGUMENTS:

af: Address family ∈ {AF_INET, AF_INET6}.

src: An address family structure containing the address in network format to be converted to presentation format.

dst: An empty buffer to store the address in the string format.

size: Size of the buffer.
INET_ADDRSTRLEN for IPv4
INET6_ADDRSTRLEN for IPv6

inet_pton ()

Convert IPv4 and IPv6 addresses from text to binary form.

#include <arpa/inet.h>

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

RETURN:

inet_pton() returns 1 on success (network address was successfully converted). 0 is returned if *src* does not contain a character string representing a valid network address in the specified address family. If *af* does not contain a valid address family, -1 is returned and *errno* is set to **EAFNOSUPPORT**.

ARGUMENTS:

af: Address family ∈ {AF_INET, AF_INET6}.

src: An IPv4 or IPv6 address in dotted decimal (IPv4) or hexadecimal (IPv6) notation as a string.

dst: A pointer to an empty address family structure to store the address in network format.

Byte order conversion:

htonl, htons, ntohl, ntohs - convert values between host and network byte order

#include <arpa/inet.h>

uint32_t htonl(uint32_t *hostlong***);**

uint16_t htons(uint16_t *hostshort***);**

uint32_t ntohl(uint32_t *netlong***);**

uint16_t ntohs(uint16_t *netshort***);**

DESCRIPTION

The **htonl**() function converts the unsigned integer *host-long* from host byte order to network byte order.

The **htons**() function converts the unsigned short integer *host-short* from host byte order to network byte order.

The **ntohl**() function converts the unsigned integer *net-long* from network byte order to host byte order.

The **ntohs**() function converts the unsigned short integer *net-short* from network byte order to host byte order.

getnameinfo ()

address-to-name translation in protocol-independent manner.

```
#include <sys/socket.h>
#include <netdb.h>

int getnameinfo(const struct sockaddr *addr, socklen_t addrlen,
        char *host, socklen_t hostlen,
        char *serv, socklen_t servlen, int flags);
```

DESCRIPTION

addr: A pointer to the socket address structure that is to be converted.
addrlen: Size of the input address *addr.*

host, serv: contain the name of the host and service. They are outputs of this function.
hostlen, servlen: are the lengths of the *host* and the *serv* string respectively.

flags: Set as zero. See *man* for details.

RETURN VALUE

On success, 0 is returned, and node and service names, if requested, are filled with null-terminated strings, possibly truncated to fit the specified buffer lengths. On error, an error code is returned.

getaddrinfo ()

Network address and service translation.
Related functions are also covered.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
```

int getaddrinfo(const char ****host*, const char ****service***,**
const struct addrinfo ****hints*,**
struct addrinfo *****res*);**

void freeaddrinfo(struct addrinfo ****res*);**

const char *gai_strerror(int *errcode***);**

DESCRIPTION:

Given *host* and *service*, which identify an Internet host and a service, **getaddrinfo**() returns one or more *addrinfo* structures, each of which contains an Internet address that can be specified in a call to bind () or connect ().

hints: Specify as NULL.
res: Updated to contain a pointer to a linked list containing address structures corresponding to the given host and the service.

RETURN:
getaddrinfo() returns 0 if it succeeds, an error code otherwise.

freeaddrinfo (): Removes the linked list returned by getaddrinfo ().

gai_strerror (): Returns the message corresponding to given error code.

fork ()

Create a child process.

#include <unistd.h>

pid_t fork(void);

RETURN:

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created, and *errno* is set appropriately.

See manual entry for fork () (man 2 fork), a must read.

Address Structures IPv4:

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    uint32_t       s_addr;     /* address in network byte order */
};
```

Special addresses

- INADDR_LOOPBACK (127.0.0.1) always refers to the local host via the loopback device;
- INADDR_ANY (0.0.0.0) means any address for binding;
- INADDR_BROADCAST (255.255.255.255) means any host and has the same effect on bind as INADDR_ANY for historical reasons.

struct sockaddr_in* is type casted into **struct sockaddr*** and vice versa whenever used in a system call.

Xterm

The *xterm* program is a terminal emulator for the X Window System.
To invoke the *xterm* from C code use *execlp ()* system call.

Example

The example program displays a C program (*sample.c*) using the x-window. *more* is used to display the program within the xterminal.

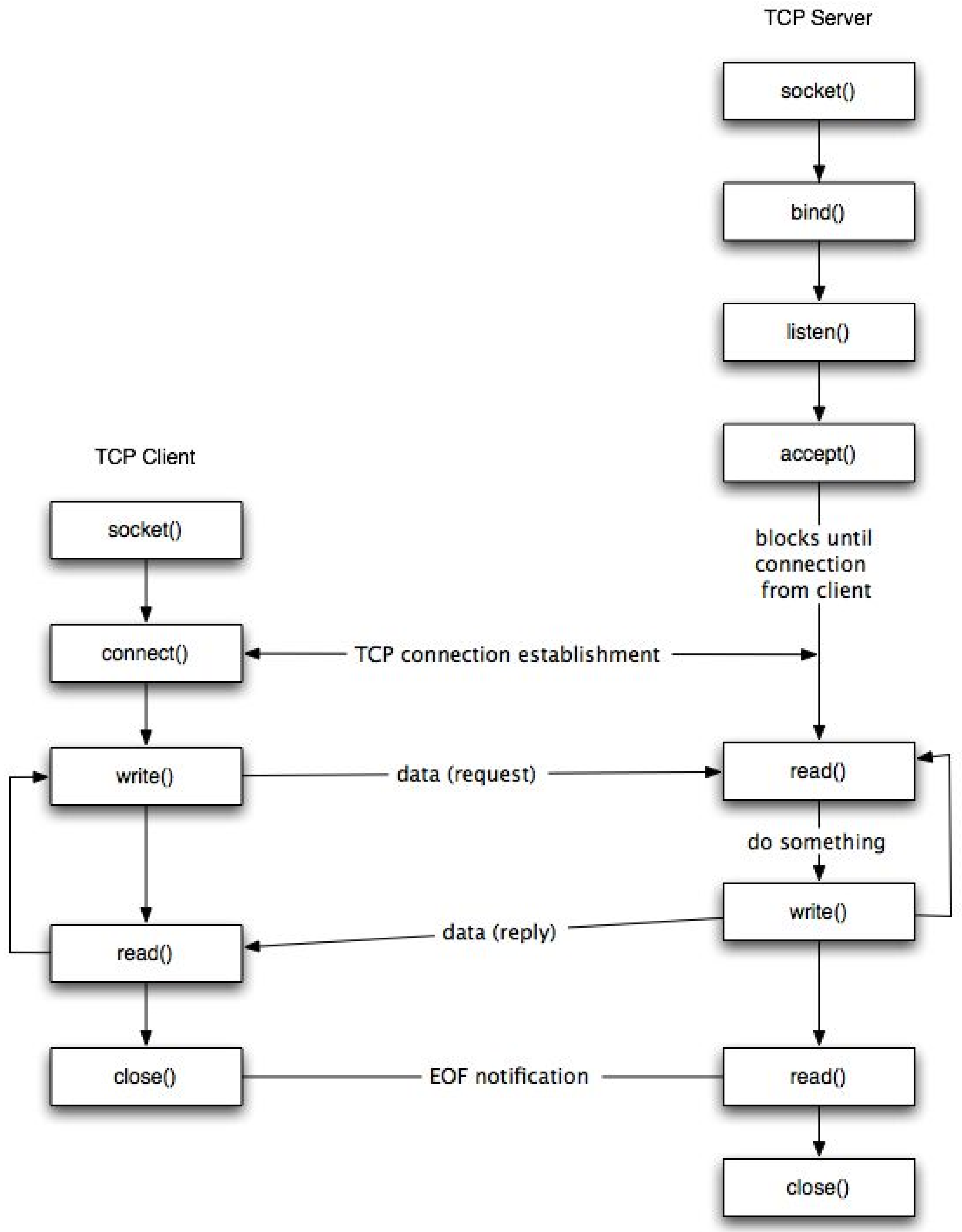
```
#include <stdio.h>
#include <unistd.h>

void main ()
{
      int rtv = execlp ("xterm", "xterm", "-e", "more", "sample.c",
                                            (const char *) NULL);
      if (rtv == -1)
             perror ("Error in exec");
      else
                   printf ("No error in exec.\n");  /* Note that on
successful execution this statement will not be executed as execlp ()
overwrites the program of a process with a new program. */
}

/* The typecasted NULL (last) argument of execlp () is to mark end of
arguments of the execlp (). */
```

Look at manual entry (`man 3 execlp`) of *execlp ()* for details about the function.

TCP Socket APIs for client-server communication:



Goto https://goo.gl/eVLjaz for sample codes.