Socket Programming Quick Reference

Data Structures

The `sockaddr_in` data structure contains an IP address and port number.

```
struct sockaddr_in {
    short int          sin_family;  // Address family
    unsigned short int sin_port;    // Port number
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};
```

- `sin_family`: Set this to `AF_INET`.
- `sin_port`: Port number of socket in network byte order.
- `sin_addr`: IP address of socket in network byte order
- `sin_zero`: Set this to all zeros to pad structure.

The `hostent` data structure contains information on the name of a host and its IP address.

```
struct hostent {
    char    *h_name;
    char    **h_aliases;
    int     h_addrtype;
    int     h_length;
    char    **h_addr_list;
};
#define h_addr h_addr_list[0]
```

- `h_name`: Offical name of host.
- `h_aliases`: A NULL-terminated array of alternate names for the host.
- `h_addrtype`: The type of address being returned; usually `AF_INET`.
- `h_length`: The length of the address in bytes.
- `h_addr_list`: A zero-terminated array of network addresses for the host in network byte order.
- `h_addr`: The first address in `h_addr_list`.

System Calls

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

- This function creates the socket and returns a unique file descriptor for the socket. The socket can either be a stream (TCP) socket, or a datagram (UDP) socket depending on the input arguments.
  - `domain`: Set this to `AF_INET` or `PF_INET`. It doesn't really matter which one.
  - `type`: Specifies what kind of socket. Set this to `SOCK_STREAM` for TCP streams (telnet, http, etc.). Set this to `SOCK_DGRAM` for UDP datagrams.
  - `protocol`: Takes one of the following values:
    - 0: Automatically selects correct protocol based on `type`.
    - `IPPROTO_TCP`: Selects TCP protocol.
    - `IPPROTO_UDP`: Selects UDP protocol.

`int bind(int sockfd, struct sockaddr *my_addr, int addrlen);`

- This function associates socket `sockfd` with a port on the local machine. This function only needs to be called for incoming connections on the server. Returns -1 if there is an error. The follwing are the input arguments:
  - `sockfd`: This is the value returned by the `socket()` function call.

  – `my_addr`: This is the `sockaddr_in` data structure containing the IP address and port number on the local machine.
  – `addrlen`: This is the size of the `sockaddr_in` data structure.

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

- This function listens for incoming connections. It only needs to be called by the server for connection-oriented (TCP) sockets. The function returns -1 on an error. The following are the input arguments:
  – `sockfd`: This is the value returned by the `socket()` function call.
  – `backlog`: The maximum number of connections allowed to wait in the incoming queue. Incoming connections remain in the queue until `accept()` is called.

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

- This function is called by the client to initiate a connection to a server. It is used for connection-oriented (TCP) sockets. The function returns -1 on an error. The following are the input arguments:
  – `sockfd`: This is the value returned by the `socket()` function call.
  – `serv_addr`: This is the `sockaddr_in` data structure containing the IP address and port number of the server.
  – `addrlen`: This is the size of the `sockaddr_in` data structure.

```
int accept(int sockfd, void *addr, int *addrlen);
```

- This function accepts a connection from the incoming queue associated with the socket `sockfd`. The function is used by the server for connection-oriented (TCP) sockets. The function returns a new socket file descriptor which can be used to send and receive information on the connection. The function returns -1 on an error. The following are the input arguments:
  – `sockfd`: This is the socket file descriptor for the socket that is listening for connections.
  – `addr`: This is a pointer to a local `sockaddr_in` data structure that can be used to hold the IP address and port number of the incoming connecting client. This data structure is different from the one that contains the IP address and port number of the server.
  – `addrlen`: This is the size of the above `sockaddr_in` data structure.

```
int send(int sockfd, const void *msg, int len, int flags);
```

- Once a connection has been established (the client has `connect()`ed and the server has `accept()`ed), this function is used to send information from client to server or from server to client. If one is using `send()`, the other should be using `recv()`. This function either returns the number of bytes sent out or returns -1 if there is an error.
  – `sockfd`: This is the socket file descriptor for the socket being used to send the data. On the client side, `sockfd` is the same socket used when calling `connect()`. On the server side, `sockfd` is the socket returned from `accept()`.
  – `msg`: This is a pointer to the data that is being sent.
  – `len`: This is the length of the data in bytes.
  – `flags`: Set this to 0.

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

- This function receives data that has been sent to a socket. The function returns the number of bytes actually received, returns -1 if there is an error, or returns 0 if the other end has closed the connection.
  – `sockfd`: This is the socket file descriptor for the socket from which the data is being read. On the client side, `sockfd` is the same socket used when calling `connect()`. On the server side, `sockfd` is the socket returned from `accept()`.
  – `buf`: This is a pointer to the buffer into which the data will be read.
  – `len`: This is the maximum length of the buffer in bytes.
  – `flags`: Set this to 0.

`int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *toaddr, int addrlen);`

- This function is similar to `send()`; however, it is used for connectionless datagrams rather than connection-oriented communications.
  - `sockfd`: This is the socket file descriptor for the socket being used to send the data.
  - `msg`: This is a pointer to the data that is being sent.
  - `len`: This is the length of the data in bytes.
  - `flags`: Set this to 0.
  - `toaddr`: This is the `sockaddr_in` data structure containing the IP address and port number to which the data is being sent.
  - `addrlen`: This is the size of the above `sockaddr_in` data structure.

`int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *fromaddr, int *addrlen);`

- This function is similar to `recv()`, but is used for connectionless datagrams. The additional arguments store information about the host that sent the data.
  - `sockfd`: This is the socket file descriptor for the socket being used to receive the data.
  - `buf`: This is a pointer to the buffer into which the data will be read.
  - `len`: This is the length of the data in bytes.
  - `flags`: Set this to 0.
  - `fromaddr`: This is a pointer to a local `sockaddr_in` data structure that will be used to store the IP address and port number of the host that is sending the data.
  - `addrlen`: Pointer to an `int` that will store the size of the above `sockaddr_in` data structure.

`void close(sockfd);`

- This function closes a socket.

`struct hostent *gethostbyname(const char *name);`

- This function returns a pointer to a structure of type `hostent` which contains the host name and host IP address. The argument `name` is the name of the host (e.g., `net49.utdallas.edu`).

Below is a table which summarizes the various functions that need to be called at the client and server for TCP and UDP sockets.

	client	server
connection-oriented TCP	socket()	socket()
		bind()
		listen()
	connect()	
		accept()
	send()	recv()
	recv()	send()
connectionless UDP	socket()	socket()
		bind()
	sendto()	recvfrom()
	recvfrom()	sendto()