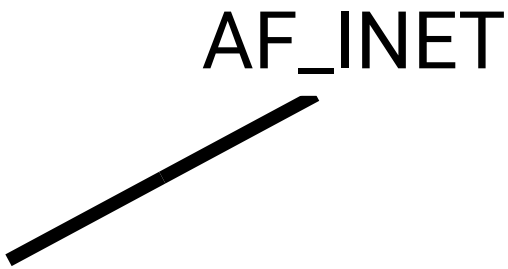
Socket Programming

socket() -- Get the file descriptor

- int socket(int domain, int type, int protocol);
  - domain should be set to AF_INET
  - type can be SOCK_STREAM or SOCK_DGRAM
  - set protocol to 0 to have socket choose the correct protocol based on type
  - socket() returns a socket descriptor for use in later system calls or -1 on error




```
int sockfd;
sockfd = socket (PF_INET, SOCK_STREAM, 0);
```

Socket Structures

- struct sockaddr: Holds socket address information for many types of sockets

```
struct sockaddr {
        unsigned short   sa_family;     //address family AF_xxx
        unsigned short   sa_data[14]; //14 bytes of protocol addr
}
```

- struct sockaddr_in: A parallel structure that makes it easy to reference elements of the socket address

```
struct sockaddr_in {
        short int          sin_family;    // set to AF_INET
        unsigned short int sin_port;      // Port number
        struct in_addr     sin_addr;      // Internet address
        unsigned char      sin_zero[8];  //set to all zeros
}
```

- sin_port and sin_addr must be in **Network Byte Order**

Byte Ordering

- Two types of “Byte ordering”
  - Network Byte Order: High-order byte of the number is stored in memory at the lowest address
  - Host Byte Order: Low-order byte of the number is stored in memory at the lowest address
  - Network stack (TCP/IP) expects Network Byte Order
- Conversions:
  - htons() - Host to Network Short
  - htonl() - Host to Network Long
  - ntohs() - Network to Host Short
  - ntohl() - Network to Host Long

Dealing with IP Addresses

- ```
  struct in_addr {
      unsigned long s_addr; // that's a 32-bit long, or 4 bytes
  };
  ```
- int inet_aton(const char *cp, struct in_addr *inp);

```
struct sockaddr_in     my_addr;
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(MYPORT);
inet_aton("10.0.0.5",&(my_addr.sin_addr));
memset(&(my_addr.sin_zero),'\0',8);
```

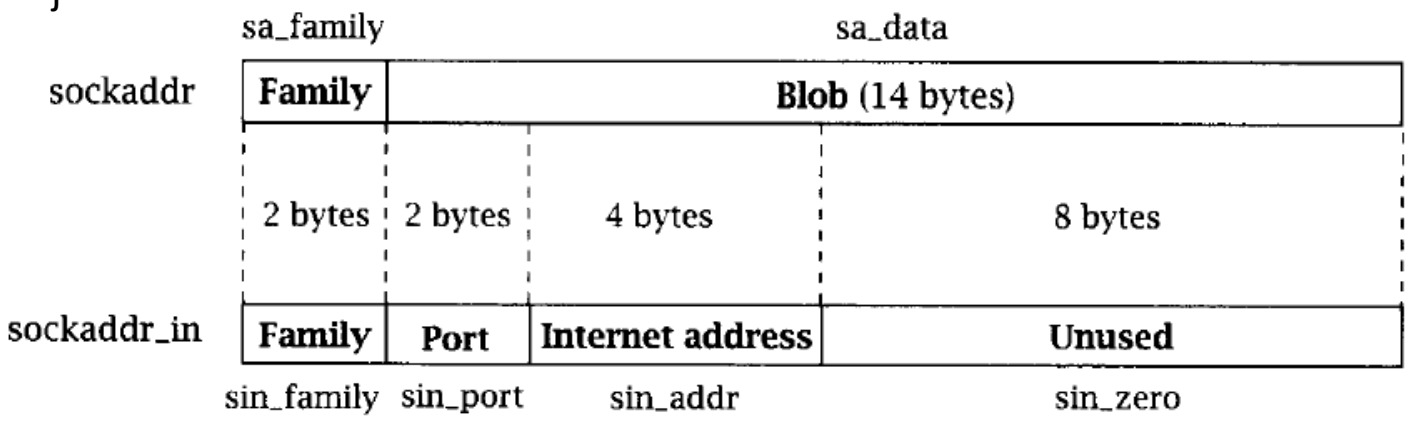
  - inet_aton() gives non-zero on success; zero on failure
- To convert binary IP to string: inet_ntoa( )

```
printf("%s",inet_ntoa(my_addr.sin_addr));
```
```

❖ Specifying Addresses

- Applications need to be able to specify Internet address and Port number. How?
- Use **Address Structure**
  1. Sockaddr: generic data type
  2. in_addr : internet address
  3. sockaddr_in: another view of Sockaddr

```
struct sockaddr_in{
unsigned short sin_family;    /* Internet protocol (AF_INET) */
unsigned short sin_port;     /* Address port (16 bits) */
struct in_addr sin_addr;    /* Internet address (32 bits) */
char sin_zero[8];              /* Not used */
}
```



bind() - what port am I on?

- Used to associate a socket with a port on the local machine
  - The port number is used by the kernel to match an incoming packet to a process
- int bind(int sockfd, struct sockaddr *my_addr, int addrlen)
  - sockfd is the socket descriptor returned by socket()
  - my_addr is pointer to struct sockaddr that contains information about your IP address and port
  - addrlen is set to sizeof(struct sockaddr)
  - returns -1 on error
  - my_addr.sin_port = 0; //choose an unused port at random
  - my_addr.sin_addr.s_addr = INADDR_ANY; //use my IP adr

Example

```
int sockfd;

struct sockaddr_in my_addr;

sockfd = socket(PF_INET, SOCK_STREAM, 0);

my_addr.sin_family = AF_INET;         // host byte order

my_addr.sin_port = htons(MYPORT);     // short, network byte
   order

my_addr.sin_addr.s_addr = inet_addr("172.28.44.57");

memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
   sockaddr));

/******  Code needs error checking. Don't forget to do that ****** /
```

connect() - Hello!

- Connects to a remote host
- int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)
  - sockfd is the socket descriptor returned by socket()
  - serv_addr is pointer to struct sockaddr that contains information on destination IP address and port
  - addrlen is set to sizeof(struct sockaddr)
  - returns -1 on error
- No need to bind(), kernel will choose a port

Example

```
#define DEST_IP   "172.28.44.57"
#define DEST_PORT 5000
main(){
   int sockfd;
   struct sockaddr_in dest_addr;   // will hold the destination addr
   sockfd = socket(PF_INET, SOCK_STREAM, 0);
    dest_addr.sin_family = AF_INET;          // host byte order
    dest_addr.sin_port = htons(DEST_PORT);   // network byte
   order
    dest_addr.sin_addr.s_addr = inet_addr(DEST_IP);
    memset(&(dest_addr.sin_zero), '\0', 8);  // zero the rest of the
   struct    connect(sockfd, (struct sockaddr *)&dest_addr,
   sizeof(struct sockaddr));
  /******* Don't forget error checking ********/
```

listen() - Call me please!

- Waits for incoming connections
- int listen(int sockfd, int backlog);
  - sockfd is the socket file descriptor returned by socket()
  - backlog is the number of connections allowed on the incoming queue
  - listen() returns -1 on error
  - Need to call bind() before you can listen()
    - socket()
    - bind()
    - listen()
    - accept()

accept() - Thank you for calling !

- accept() gets the pending connection on the port you are listen()ing on
- int accept(int sockfd, void *addr, int *addrlen);
  - sockfd is the listening socket descriptor
  - information about incoming connection is stored in addr which is a pointer to a local struct sockaddr_in
  - addrlen is set to sizeof(struct sockaddr_in)
  - accept returns *a new socket file descriptor* to use for this accepted connection and -1 on error

Example

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define MYPORT 3490    // the port users will be connecting to
#define BACKLOG 10     // pending connections queue will hold
main(){
    int sockfd, new_fd;  // listen on sock_fd, new connection on
    new_fd
    struct sockaddr_in my_addr;    // my address information
    struct sockaddr_in their_addr; // connector's address information
    int sin_size;
   sockfd = socket(PF_INET, SOCK_STREAM, 0);
```

Cont...

```
my_addr.sin_family = AF_INET;         // host byte order
my_addr.sin_port = htons(MYPORT);     // short, network byte order
my_addr.sin_addr.s_addr = INADDR_ANY; // auto-fill with my IP
memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct
// don't forget your error checking for these calls:
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
listen(sockfd, BACKLOG);
sin_size = sizeof(struct sockaddr_in);
new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
```

send() and recv() - Let's talk!

- The two functions are for communicating over stream sockets or connected datagram sockets.
- int send(int sockfd, const void *msg, int len, int flags);
  - sockfd is the socket descriptor you want to send data to (returned by socket() or got from accept())
  - msg is a pointer to the data you want to send
  - len is the length of that data in bytes
  - set flags to 0 for now
  - sent() returns the number of bytes actually sent (may be less than the number you told it to send) or -1 on error

send() and recv() - Let's talk!

- int recv(int sockfd, void *buf, int len, int flags);
  - sockfd is the socket descriptor to read from
  - buf is the buffer to read the information into
  - len is the maximum length of the buffer
  - set flags to 0 for now
  - recv() returns the number of bytes actually read into the buffer or -1 on error
  - If recv() returns 0, the remote side has closed connection on you

sendto() and recvfrom() - DGRAM style

- int sendto(int sockfd, const void *msg, int len, int flags, const struct sockaddr *to, int tolen);
  - *to* is a pointer to a struct sockaddr which contains the destination IP and port
  - *tolen* is sizeof(struct sockaddr)
- int recvfrom(int sockfd, void *buf, int len, int flags, struct sockaddr *from, int *fromlen);
  - *from* is a pointer to a local struct sockaddr that will be filled with IP address and port of the originating machine
  - *fromlen* will contain length of address stored in *from*

Sending message on socket

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

  The above call is used to send a message on the socket

- Arguments :
  - sockfd – File descriptor of socket
  - buf – Application buffer containing the data to be sent
  - len – Size of buf application buffer
  - flags – Bitwise OR of flags to modify socket behaviour
  - dest_addr – Structure containing address of destination
  - addrlen – Size of dest_addr structure

Receiving message from socket

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

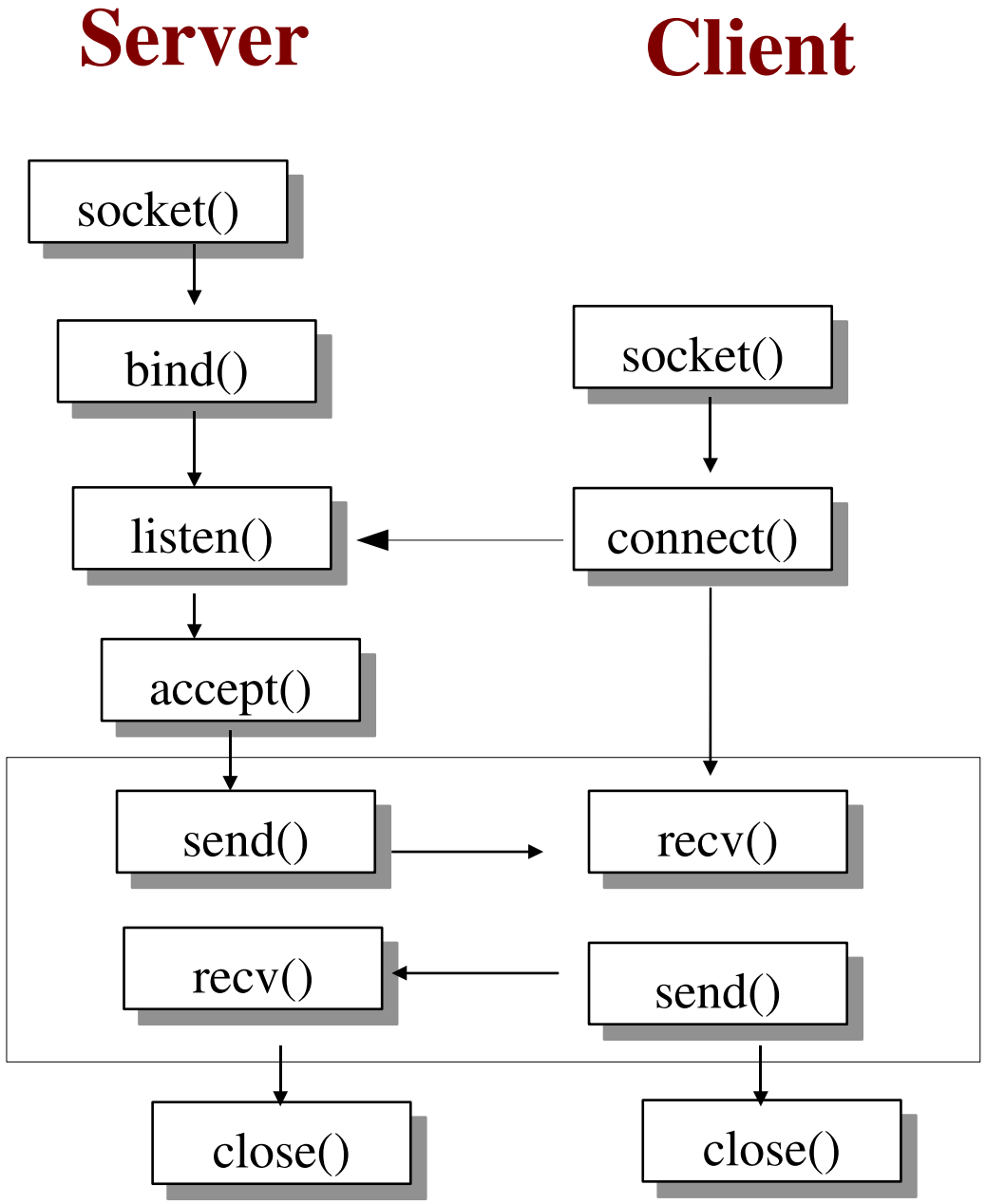
  Above call is used to receive message from the socket.

- Arguments :
  - sockfd – File descriptor of socket
  - buf – Application buffer in which to receive data
  - len – Size of buf application buffer
  - flags – Bitwise OR of flags to modify socket behaviour
  - src_addr – Structure containing source address is returned
  - addrlen – Variable in which size of src_addr structure is returned

close() - Bye Bye!

- int close(int sockfd);
  - Closes connection corresponding to the socket descriptor and frees the socket descriptor
  - Will prevent any more sends and recvs

Connection Oriented Protocol



UDP Client Server Implementation

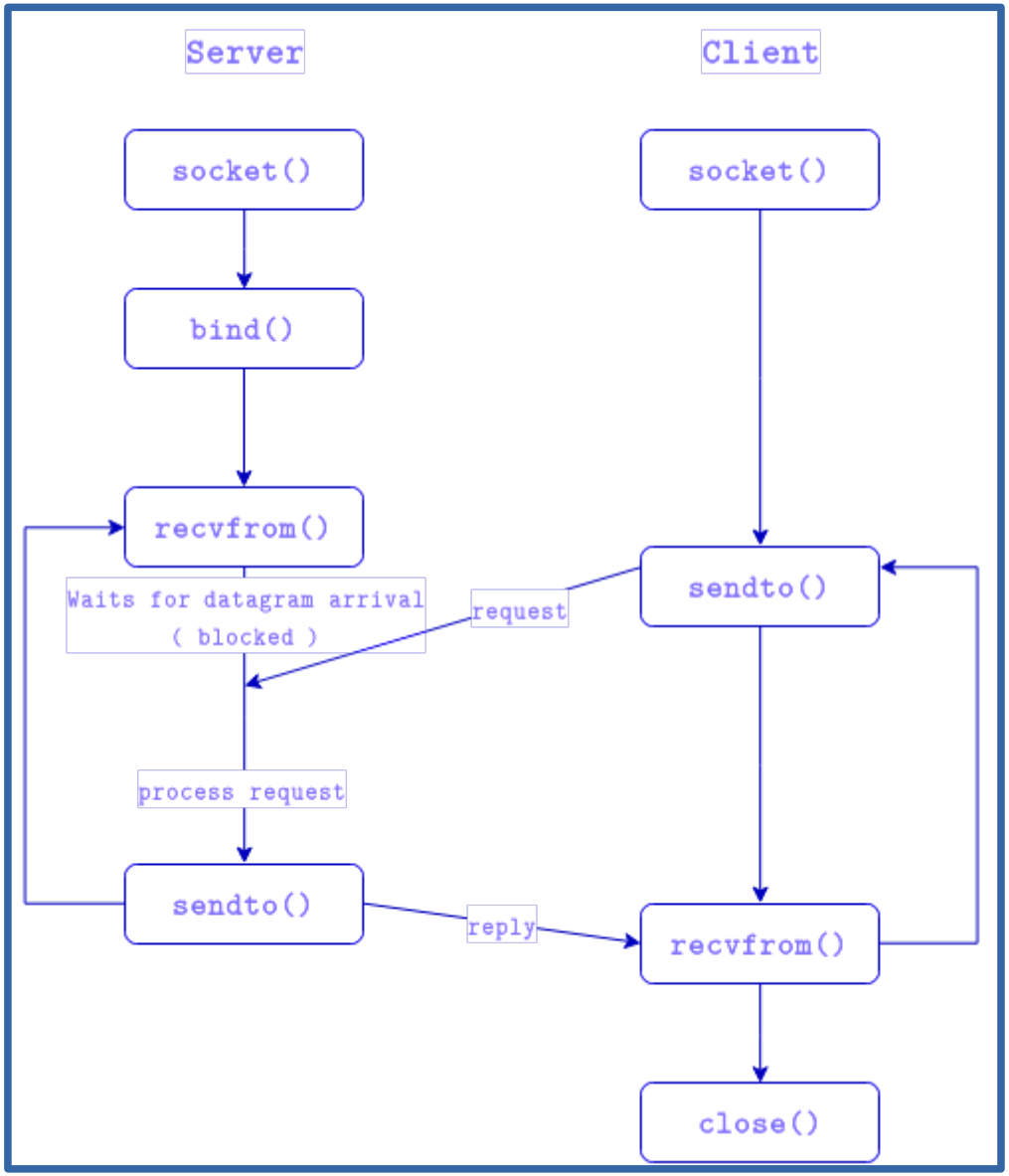
The entire process can be broken down into following steps :

UDP Server :
1)Create UDP socket.
2)Bind the socket to server address.
3)Wait until datagram packet arrives from client.
4)Process the datagram packet and send a reply to client.
5)Go back to Step 3.

UDP Client :
1)Create UDP socket.
2)Send message to server.
3)Wait until response from server is recieved.
4)Process reply and go back to step 2, if necessary.
5)Close socket descriptor and exit.

UDP Client-Server Implementation Flow

- In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram.
- Similarly, the server need not accept a connection and just waits for datagrams to arrive.
- Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.



Miscellaneous Routines

- int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
  - Will tell who is at the other end of a connected stream socket and store that info in *addr*
- int gethostname(char *hostname, size_t size);
  - Will get the name of the computer your program is running on and store that info in hostname

Miscellaneous Routines

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

```
struct hostent {
    char    *h_name;        //official name of host
    char    **h_aliases;    //alternate names for the host
    int     h_addrtype;     //usually AF_NET
    int     h_length;       //length of the address in bytes
    char    **h_addr_list; //array of network addresses for the host
}
```