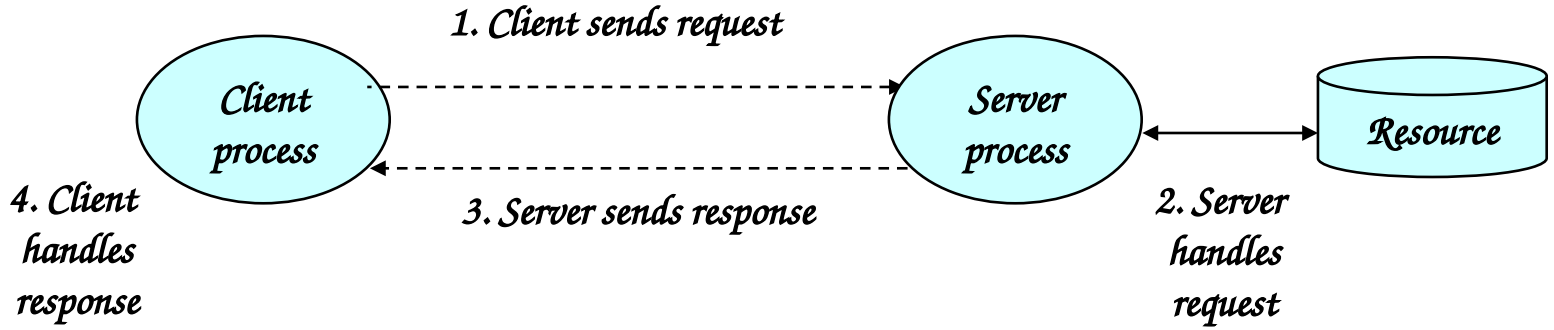
Topics

- *Client-server model*
- *Sockets interface: Address structure, byte ordering, port no*
- *Socket primitives: socket, bind,listen…*
- *Example code for echoclient and echoserver*
- *Programming Assignment*

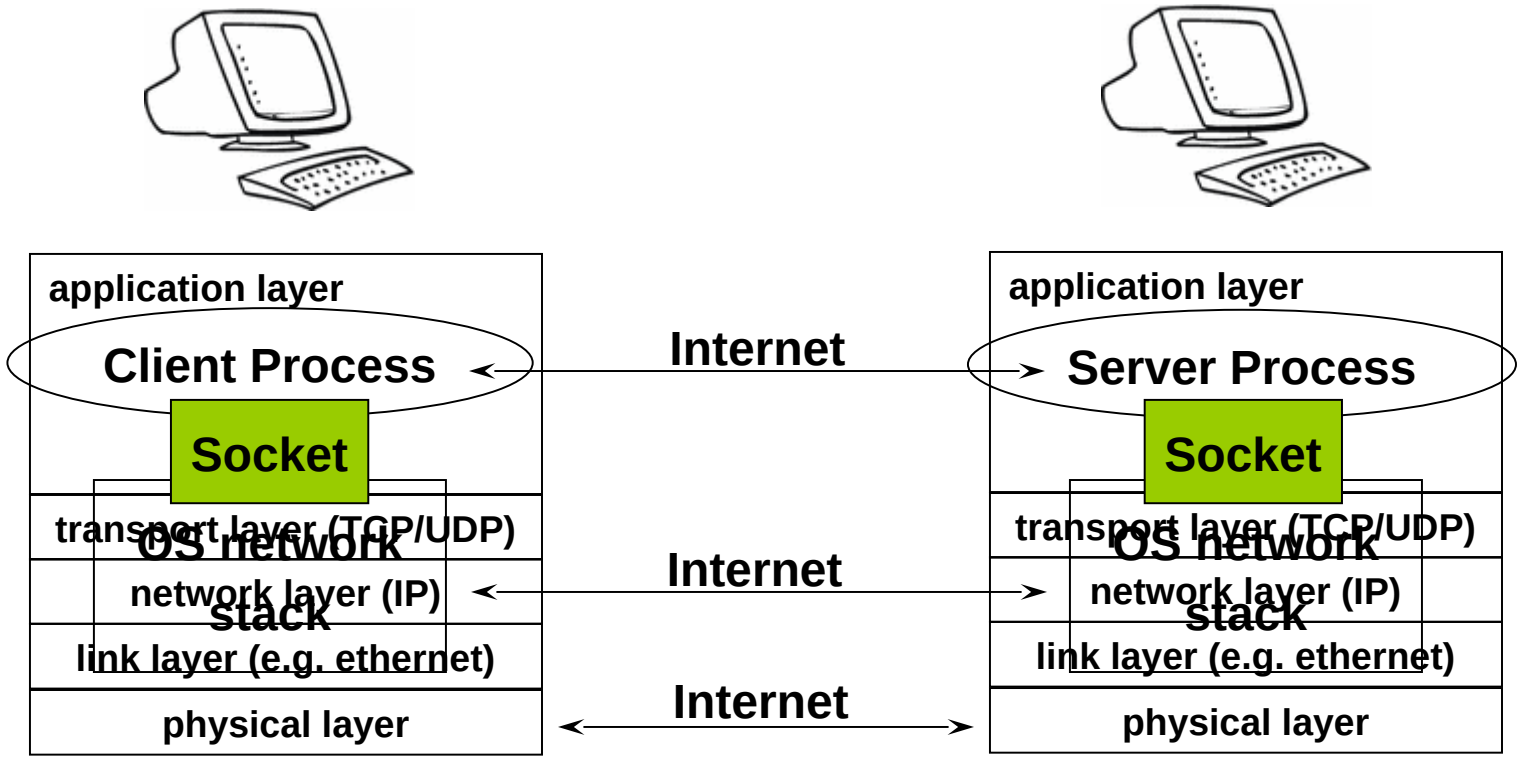
Client/sever model

- Client asks (request) – server provides (response)
- Typically: single server - multiple clients
- The server does not need to know anything about the client
  - even that it exists
- The client should always know something about the server
  - at least where it is located



Note: clients and servers are processes running on hosts (can be the same or different hosts).

Sockets as means for inter-process communication (IPC)



The interface that the OS provides to its networking subsystem

Sockets

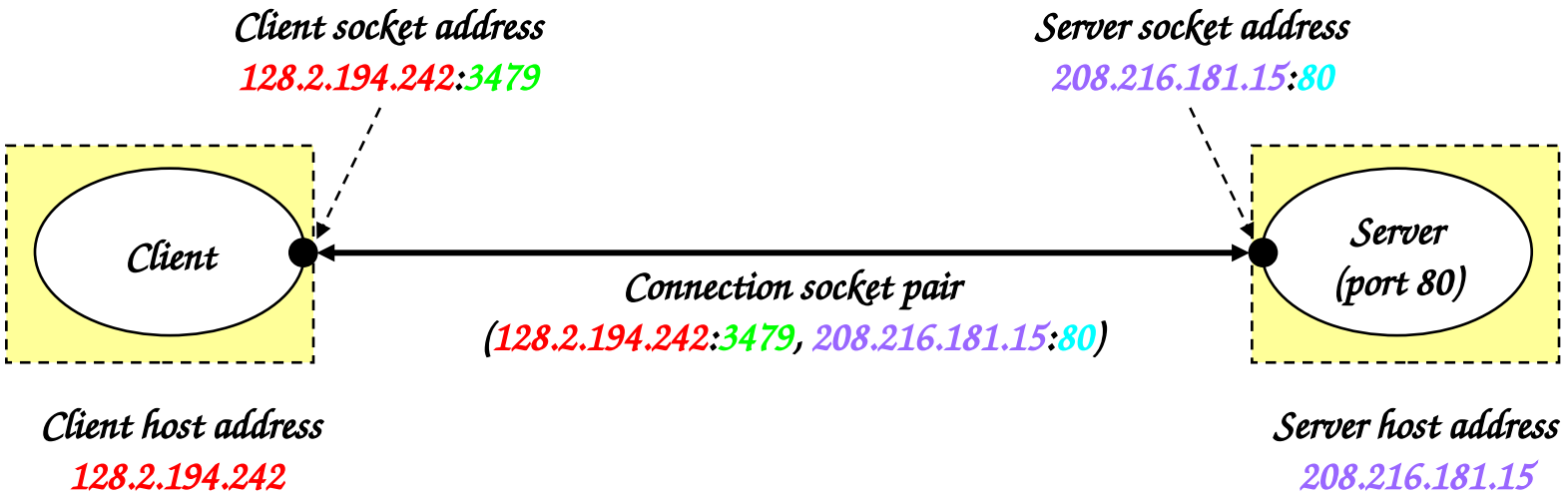
- What is a socket?
  - To the kernel, a socket is an endpoint of communication.
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network.
    - Remember: All Unix I/O devices, including networks, are modeled as files.

- Clients and servers communicate with each by reading from and writing to socket descriptors.

- The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors.

Internet Connections (TCP/IP)

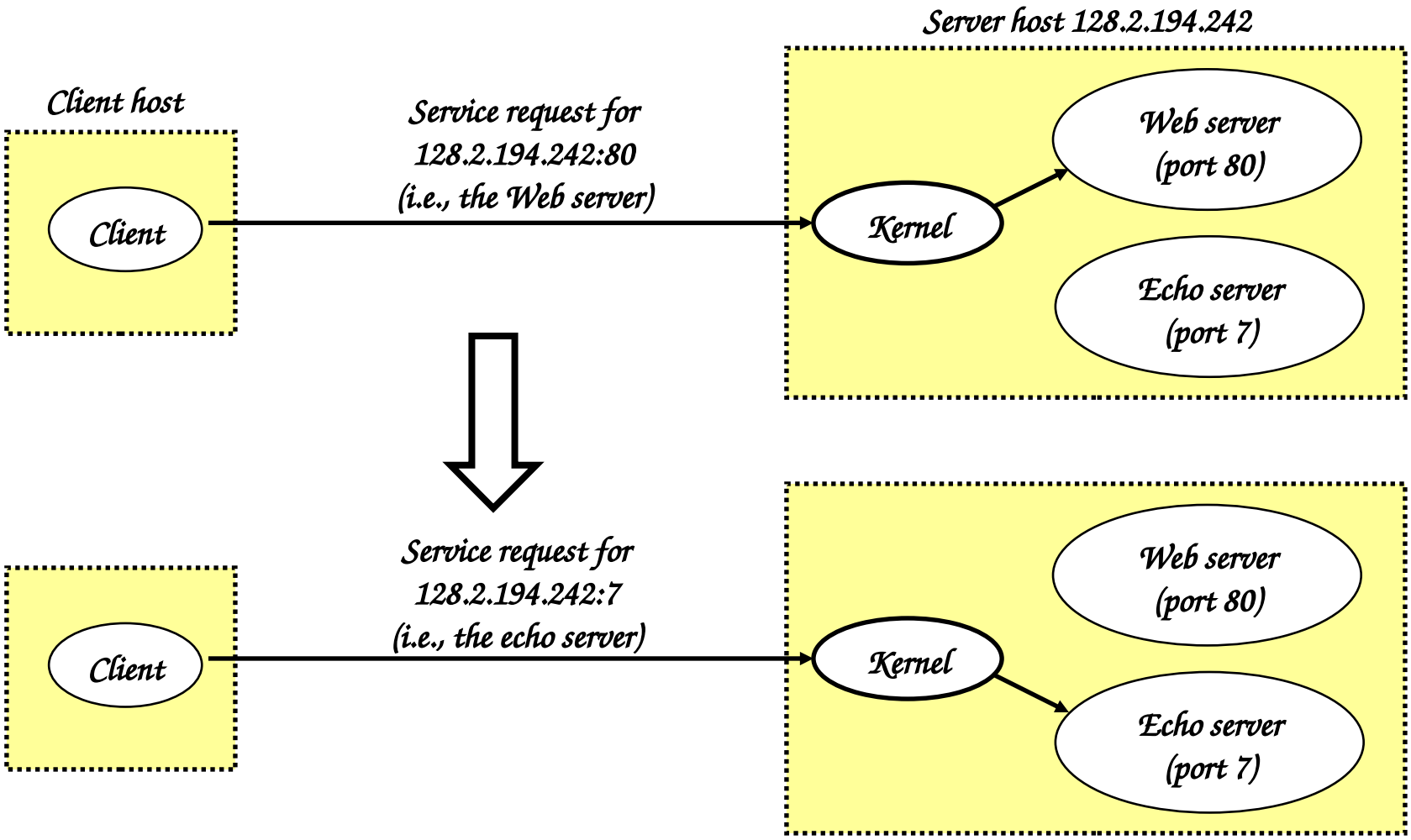
- Address the machine on the network
  - By IP address
- Address the process
  - By the "port"-number
- The pair of IP-address + port – makes up a "socket-address"



Clients

- *Examples of client programs*
  - *Web browsers,* `ftp`, `telnet`, `ssh`
- *How does a client find the server?*
  - *The IP address in the server socket address identifies the host*
  - *The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.*
  - *Examples of well known ports*
    - *Port 7: Echo server*
    - *Port 23: Telnet server*
    - *Port 25: Mail server*
    - *Port 80: Web server*

Using Ports to Identify Services



Data Structures

Let us now look at the data structures used to hold all the address Information:

- ```
  Struct sockaddr {
        unsigned short sa_family;
        char sa_data[14];
  }
  ```

- ```
  Struct sockaddr_in {
        short sin_family;
        unsigned short sin_port; // Port Number
        struct in_addr sin_addr; // IP Address
        char sin_zero[8];
  }
  ```

- ```
  Struct in_addr {
        unsigned long s_addr; // 4 bytes long
  }
  ```
  ```

Byte Ordering

- *Byte ordering is the attribute of a system which indicates whether integers are stored / represented left to right or right to left.*

- *Example 1: short int x = 0xAABB (hex)*

  This can be stored in memory as 2 adjacent bytes as either (0xaa , 0xbb) or as (0xbb, 0xaa).

  Big Endian:

  Byte Value : [0xAA] [0xBB]
  Memory : [0] [1]

  Little Endian:

  Byte Value : [0xBB] [0xAA]
  Memory : [0] [1]

Byte Ordering

- *Example 2: int x = 0xAABBCCDD*

  This 4 byte long integer can be represented in the same 2 orderings:

  Big Endian:

  Byte Value: [0xAA] [0xBB] [0xCC] [0xDD]
  Memory: [0] [1] [2] [3]

  Little Endian:

  Byte Value: [0xDD] [0xCC] [0xBB] [0xAA]
  Memory: [0] [1] [2] [3]

- *All Network data is sent in Big Endian format.*
- *In the networking world we call this representation as Network Byte Order and native representation on the host as Host Byte Order.*
- *We convert all data into Network Byte Order before transmission.*

Some utility functions :

- *Byte Ordering:*

  Host Byte Order to Network Byte Order:
  htons() , htonl()
  Network Byte Order to Host Byte Order:
  ntohs() , ntohl()

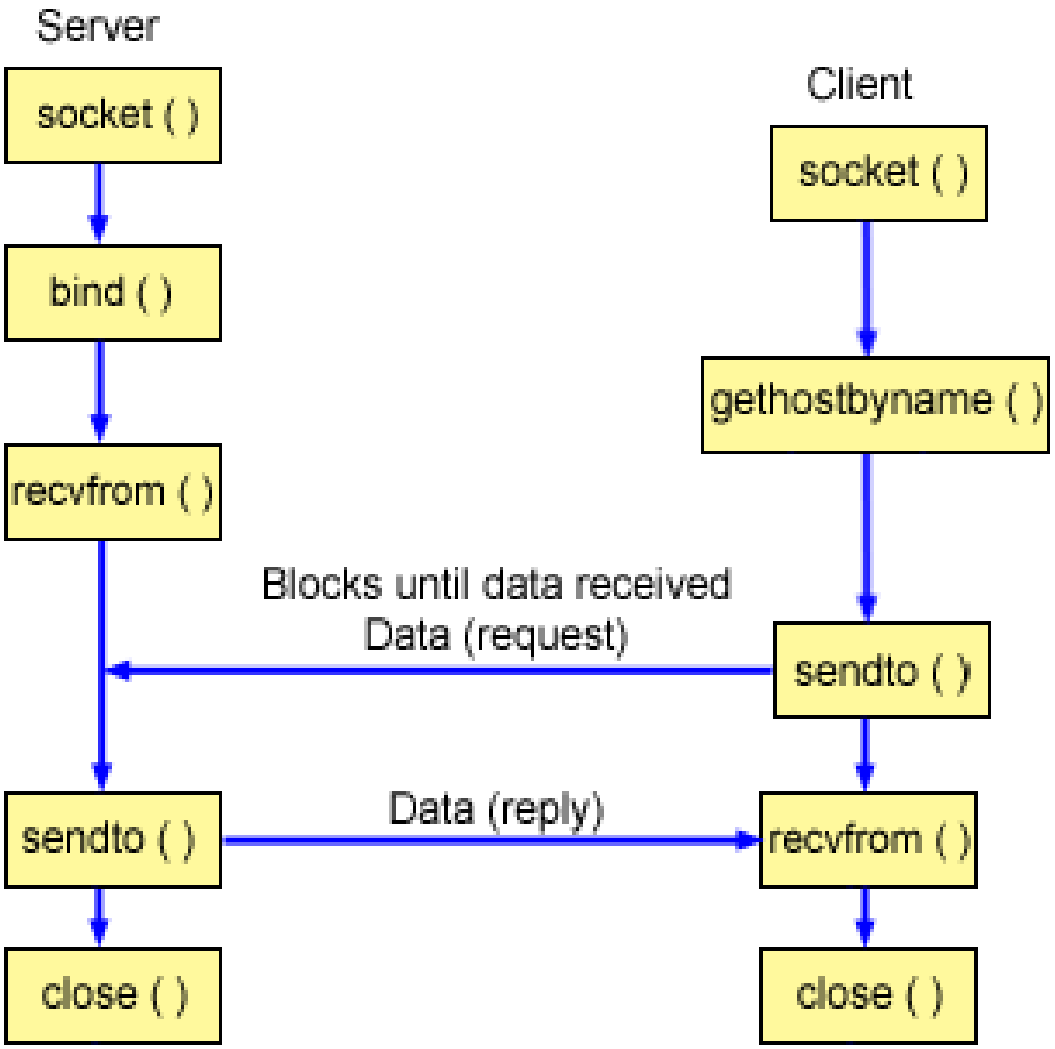
- *IP Address format:*

syscalls()

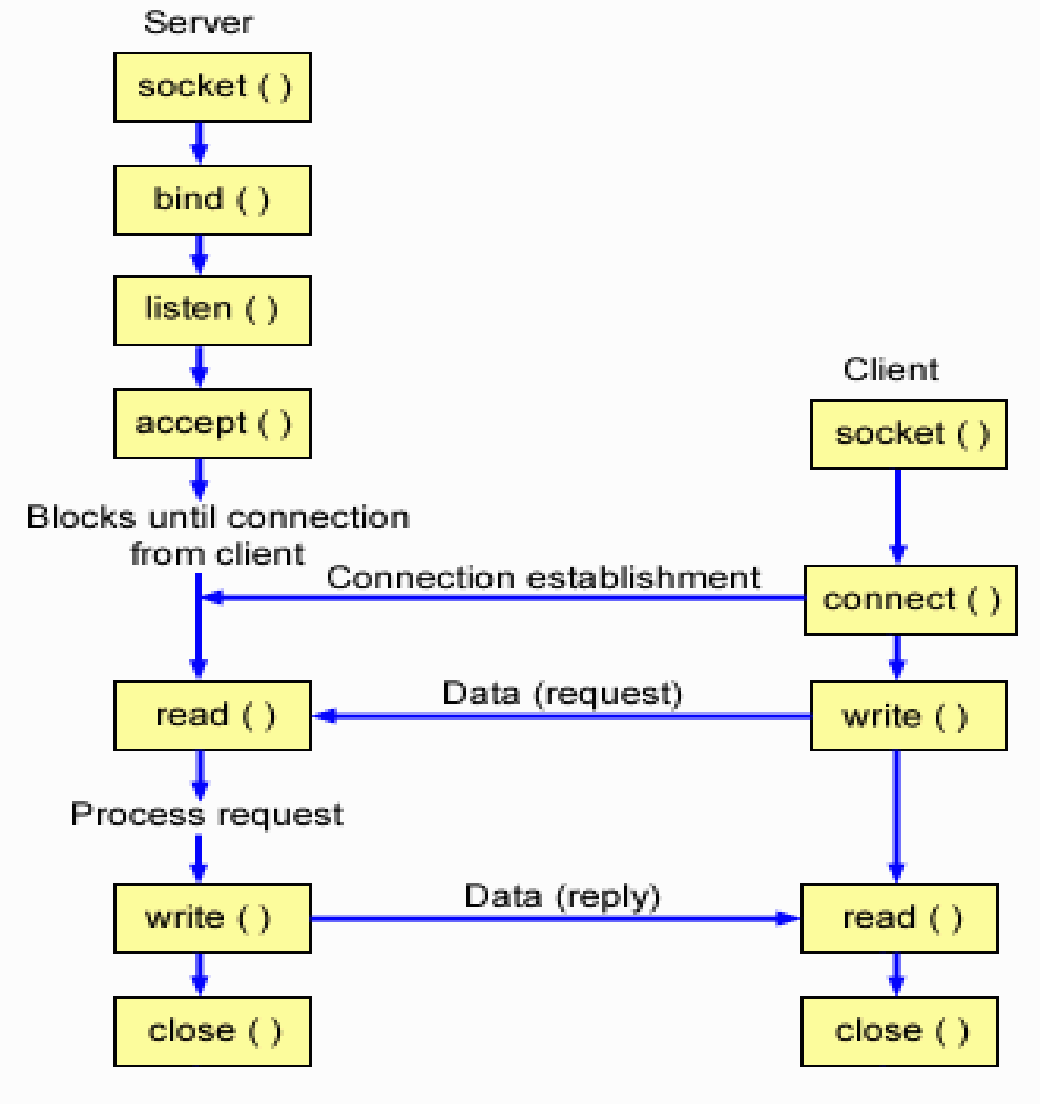
We will now describe the following calls in detail :

- *Socket()*
- *Bind()*
- *Listen()*
- *Accept()*
- *Connect()*
- *Read() / Send() / Sendto()*
- *Write() / Recv() / Recvfrom()*
- *Close()*

A UDP Server – Client Interaction



A TCP Server – Client Interaction



Socket() – A Connection Endpoint

- *This creates an endpoint for a network connection.*

  int socket(int doman, int type, int protocol)

  domain = AF_INET (IPv4 communication)
  type = SOCK_STREAM (TCP) , SOCK_DGRAM (UDP)
  protocol = 0 (for our discussion)

- *Example : socket(AF_INET, SOCK_STREAM, 0);*
  This will create a TCP socket.
- *The call returns a socket descriptor on success and -1 on an error.*

Bind() – Attaching to an IP and Port

- A server process calls bind to attach itself to a specific port and IP address.

  int bind(int sockfd, struct sockaddr *my_addr, sockaddr_in_addrlen)

  sockfd = socket descriptor returned by socket()

  my_addr = pointer to a valid sockaddr_in structure cast as a sockaddr * pointer

  addrlen = length of the sockaddr_in structure

- Example :

```
struct sockaddr_in my;
 bzero( (char *)&my, sizeof(my))
my.sin_family = AF_INET;
my.sin_port = htons(80);
my.sin_addr.s_addr =htonl( INADDR_ANY);


bind(sock, (struct sockaddr *)&my, sizeof(my));
```

Listen() – Wait for a connection

- *The server process calls listen to tell the kernel to initialize a wait queue of connections for this socket.*

  int listen(int sock, int backlog)

  sock = socket returned by socket()
  backlog = Maximum length of the pending connections queue

- *Example: Listen(sock, 10);*
  This will allow a maximum of 10 connections to be in pending state.

Accept() – A new connection !

- Accept is called by a Server process to accept new connections from new clients trying to connect to the server.

  int accept(int socket, (struct sockaddr *)&client, *client_len)

  socket = the socket in listen state
  client = will hold the new client's information when accept returns
  client_len = pointer to size of the client structure

- Example :

  struct sockaddr_in client;

  int len = sizeof(client);

  Accept(sock, (struct sockaddr *)&client, &len);

Connect() – connect to a service

- *Connect is called by a client to connect to a server port.*

  *int connect(int sock, (struct sockaddr *)&server_addr, socklen_len)*

  sock: a socket returned by socket()
  server_addr: a sockaddr_in struct pointer filled with all the remote server details and cast as a sockaddr struct pointer
  len: size of the server_addr struct

- *Example:*

  *connect(sock, (struct sockaddr *)server_addr, len);*

Send / Recv – Finally Data !!

- Send(), Recv() , Read() , Write() etc calls are used to send and receive data .

  Int send(int sock, void *mesg, size_t len, int flags)

  Int recv(int sock, void *mesg, size_t len, int flags)

  sock = A connected socket
  mesg = Pointer to a buffer to send/receive data from/in .
  len = Size of the message buffer
  flags = 0 (for our purpose)

  The return value is the number of bytes actually sent/received.

- Example:

  ```
  char send_buffer[1024];
  char recv_buffer[1024];
  int sent_bytes;
  int recvd_bytes;

  sent_bytes = send(sock, send_buffer, 1024, 0);
  recvd_bytes = recv(sock, recv_buffer, 1024, 0);
  ```

Close() – Bye ..Bye !

- Close signals the end of communication between a server-client pair. This effectively closes the socket.

  int close(int sock)

  sock = the socket to close

- Example :

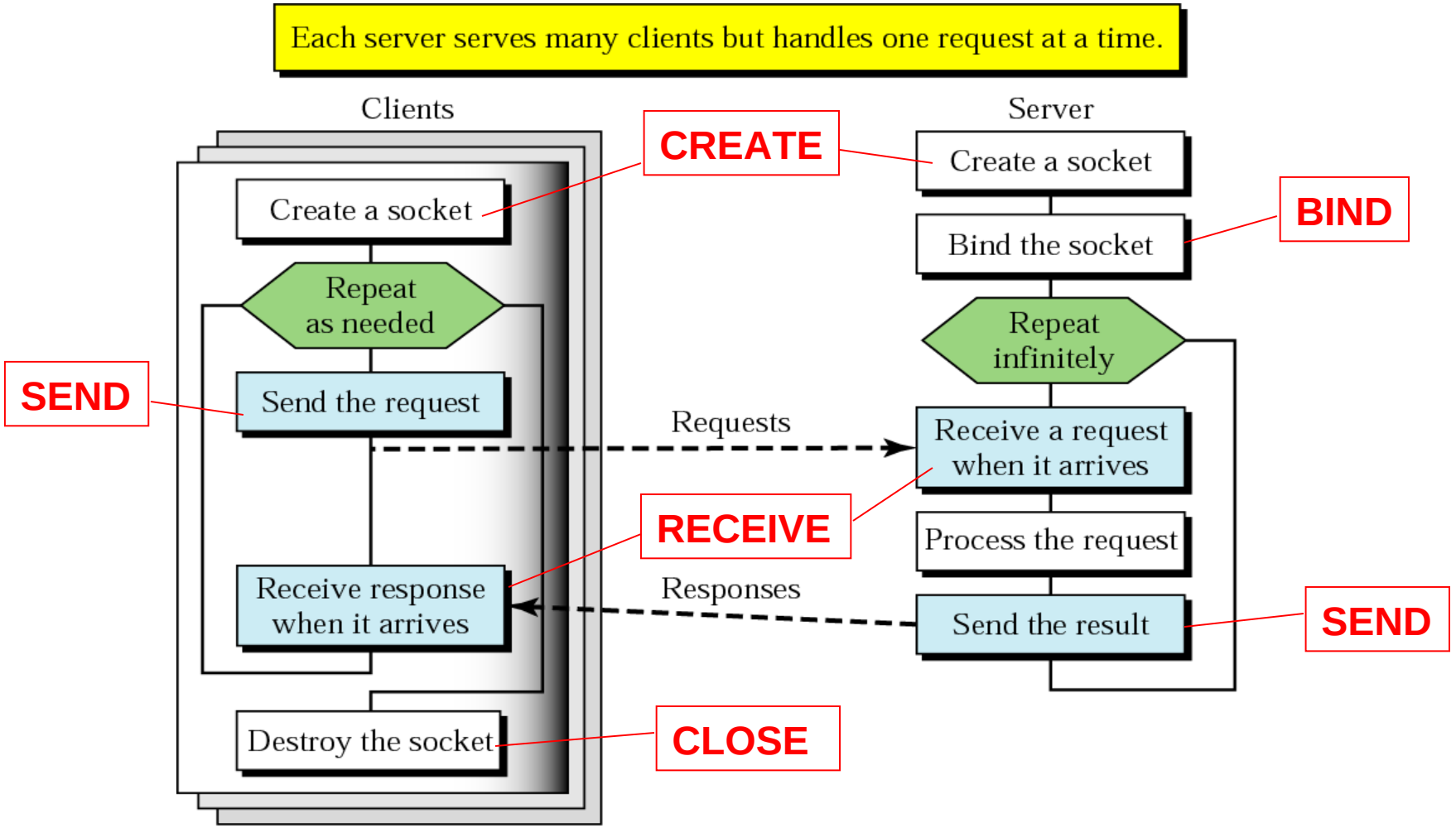
  close(sock);

- *SEND (DGRAM-style):* **int sendto(int sockfd, const void *msg, int len, int flags, const struct sockaddr *to, int tolen);**
  - *msg: message you want to send*
  - *len: length of the message*
  - *flags := 0*
  - *to: socket address of the remote process*
  - *tolen: = sizeof(struct sockaddr)*
  - *returned: the number of bytes actually sent*

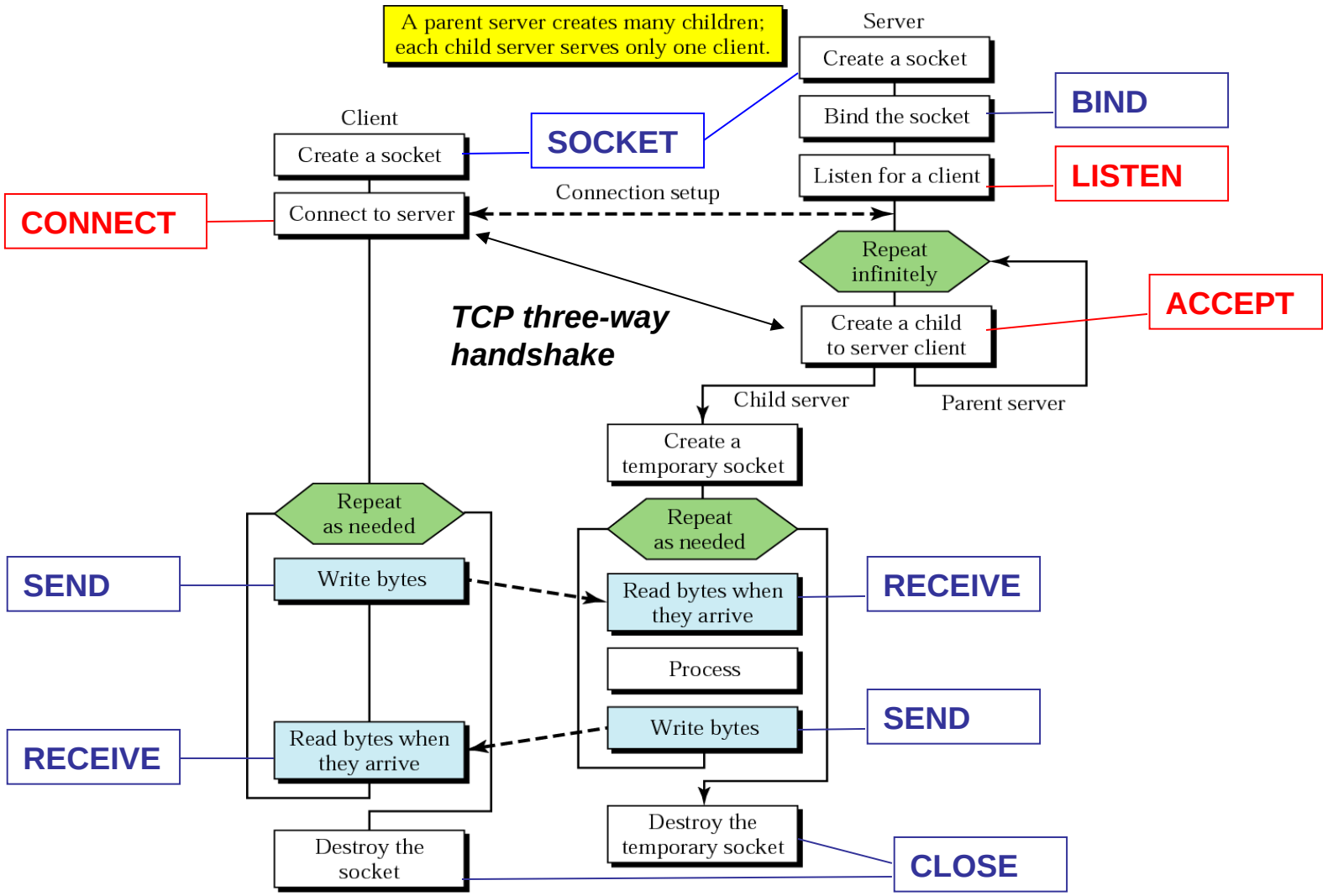
- *RECEIVE (DGRAM-style):* **int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);**
  - *buf: buffer to receive the message*
  - *len: length of the buffer ("don't give me more!")*
  - *from: socket address of the process that sent the data*
  - *fromlen:= sizeof(struct sockaddr)*
  - *flags := 0*
  - *returned: the number of bytes received*

- *CLOSE:* **close (socketfd);**

Client+server: connectionless



Client+server: connection-oriented



Concurrent server

EchoClient.c – #include's

```
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(),
                           sendto(), and recvfrom() */
#include <arpa/inet.h>  /* for sockaddr_in and
                                inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define ECHOMAX 255     /* Longest string to echo */
```

EchoClient.c -variable declarations

```
int main(int argc, char *argv[])
{
    int sock;                       /* Socket descriptor */
    struct sockaddr_in echoServAddr; /* Echo server address */
    struct sockaddr_in fromAddr;     /* Source address of echo */
    unsigned short echoServPort =10200;     /* Echo server port */
    unsigned int fromSize;          /* address size for recvfrom() */
    char *servIP="172.24.23.4";                 /* IP address of server */
    char *echoString="I hope this works";           /* String to send to echo server */
    char echoBuffer[ECHOMAX+1];      /* Buffer for receiving echoed string */
    int echoStringLen;              /* Length of string to echo */
    int respStringLen;              /* Length of received response */
```

EchoClient.c - creating the socket and sending

```
/* Create a datagram/UDP socket */
sock = socket(AF_INET, SOCK_DGRAM, 0);

/* Construct the server address structure */
memset(&echoServAddr, 0, sizeof(echoServAddr)); /* Zero out structure */
echoServAddr.sin_family = AF_INET; /* Internet addr family */
echoServAddr.sin_addr.s_addr = htonl(servIP); /* Server IP address */
echoServAddr.sin_port = htons(echoServPort); /* Server port */

/* Send the string to the server */
sendto(sock, echoString, echoStringLen, 0, (struct sockaddr *) &echoServAddr,
    sizeof(echoServAddr);
/* Recv a response */
```

EchoClient.c – receiving and printing

```
fromSize = sizeof(fromAddr);
recvfrom(sock, echoBuffer, ECHOMAX, 0, (struct sockaddr *) &fromAddr, &fromSize);
 /* Error checks like packet is received from the same server*/

/* null-terminate the received data */
echoBuffer[echoStringLen] = '\0';
printf("Received: %s\n", echoBuffer); /* Print the echoed arg */
close(sock);
exit(0);
} /* end of main () */
```

EchoServer.c

```
int main(int argc, char *argv[])
{
    int sock;                       /* Socket */
    struct sockaddr_in echoServAddr; /* Local address */
    struct sockaddr_in echoClntAddr; /* Client address */
    unsigned int cliAddrLen;        /* Length of incoming message */
    char echoBuffer[ECHOMAX];       /* Buffer for echo string */
    unsigned short echoServPort =10200;     /* Server port */
    int recvMsgSize;                /* Size of received message */
   /* Create socket for sending/receiving datagrams */
   sock = socket(AF_INET, SOCK_DGRAM, 0);
    /* Construct local address structure */
    memset(&echoServAddr, 0, sizeof(echoServAddr));   /* Zero out structure */
    echoServAddr.sin_family = AF_INET;                /* Internet address family */
    echoServAddr.sin_addr.s_addr = htonl("172.24.23.4");
    echoServAddr.sin_port = htons(echoServPort);      /* Local port */

    /* Bind to the local address */
    bind(sock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr);
```

```
for (;;) /* Run forever */
    {
        cliAddrLen = sizeof(echoClntAddr);

        /* Block until receive message from a client */
        recvMsgSize = recvfrom(sock, echoBuffer, ECHOMAX, 0,
            (struct sockaddr *) &echoClntAddr, &cliAddrLen);

        printf("Handling client %s\n", inet_ntoa(echoClntAddr.sin_addr));

        /* Send received datagram back to the client */
        sendto(sock, echoBuffer, recvMsgSize, 0,
            (struct sockaddr *) &echoClntAddr, sizeof(echoClntAddr);
    }

} /* end of main () */
```

Error handling is must

More information…

- *Socket programming*
  - *W. Richard Stevens, UNIX Network Programming*
  - *Infinite number of online resources*
  - *http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html*
- *GDB*
  - *Official GDB homepage: http://www.gnu.org/software/gdb/gdb.html*
  - *GDB primer: http://www.cs.pitt.edu/~mosse/gdb-note.html*

Example of Stream Server: echo

```
/* stream server: echo what is received from client */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
  int s, t, sinlen;
  struct sockaddr_in sin;
  char msg[80];
```

Example of Stream Server: echo (cont'd)

```
if (argc < 2) {
    printf ("%s port\n", argv[0] ); /* input error: need port no! */
   return -1;
}
if ( (s = socket(AF_INET, SOCK_STREAM, 0 ) ) < 0) { /* create
     socket*/
  perror("socket");  /* socket error */
  return -1;
}
sin.sin_family = AF_INET;            /*set protocol family to Internet */
sin.sin_port = htons(atoi(argv[1]));  /* set port no. */
sin.sin_addr.s_addr  = INADDR_ANY;   /* set IP addr to any interface */
if (bind(s, (struct sockaddr *)&sin, sizeof(sin) ) < 0 ){
    perror("bind"); return -1;  /* bind error */
}
```

Example of Stream Server: echo (cont'd)

```
/* server indicates it's ready, max. listen queue is 5 */
if (listen(s, 5)) {
    perror ("listen"); /* listen error*/
   return -1;
}
sinlen = sizeof(sin);
while (1) {
    /* accepting new connection request from client,
    socket id for the new connection is returned in t */
    if ( (t = accept(s, (struct sockaddr *) &sin, &sinlen) ) < 0 ){
            perror("accept ");  /* accpet error */
            return -1;
    }
```

Example of Stream Server: echo (cont'd)

```
    printf( "From %s:%d.\n",
            inet_ntoa(sin.sin_addr), ntohs(sin.sin_port) );
    if ( read(t, msg, sizeof(msg) ) <0) {  /* read message from client */
            perror("read");        /*  read error */
            return -1;
    }
    if ( write(t, msg, strlen(msg) ) < 0 ) {  /* echo message back */
            perror("write");    return -1; /*  write error */
    }
    /* close connection, clean up sockets */
    if (close(t) < 0) { perror("close"); return -1;}
} // not reach below
if (close(s) < 0) { perror("close"); return -1;}
return 0;
}
```

Example of Stream Client: echo

```
/* stream client: send a message to server */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#inlcude <stdio.h>
#include <netdb.h>
int main (int argc, char *argv[] )
{
  int s, n;
  struct sockaddr_in sin; struct hostent *hptr;
  char msg[80] = ”Hello World!”;
```

Example of Stream Client: echo (cont'd)

```
if ( argc < 3 ) {
      printf ( "%s host port\n", argv[0] );   /* input error: need host & port */
      return -1;
}
if ( (s = socket(AF_INET, SOCK_STREAM, 0 ) ) < 0) { /* create socket*/
  perror("socket");  /* socket error */
  return -1;
}
sin.sin_family = AF_INET;              /*set protocol family to Internet */
sin.sin_port = htons(atoi(argv[2]));  /* set port no. */
if ( (hptr =  gethostbyname(argv[1]) ) == NULL){
       fprintf(stderr, "gethostname error: %s", argv[1]);
       return = -1;
 }
memcpy( &sin.sin_addr, hptr->h_addr, hptr->h_length);
```

Example of Stream Client: echo (cont'd)

```
if (connect (s, (struct sockaddr *)&sin, sizeof(sin) ) < 0 ){
    perror(”connect”); return -1;   /* connect error */
}
if ( write(s, msg, strlen(msg) +1) < 0 ) {  /* send message to server */
      perror(”write”);    return -1; /*  write error */
}
if ( ( n = read(s, msg, sizeof(msg) ) ) <0) {  /* read message from server
    */
    perror(”read”); return -1; /*  read error */
}
printf (” %d bytes: %s\n”, n, msg);  /* print message to screen */
/* close connection, clean up socket */
if (close(s) < 0) {
   perror(”close”);   /* close error */
   return -1;}
return 0;
}
```

Compiling and Executing

```
garnet% g++ -o echo-server echo-server.c -lsocket -lnsl
garnet% g++ -o echo-client echo-client.c -lsocket -lnsl
garnet% echo-server 5700 &
garnet% echo-client opal 5700
From 128.226.123.110:32938.
  12 bytes: Hello World!
```