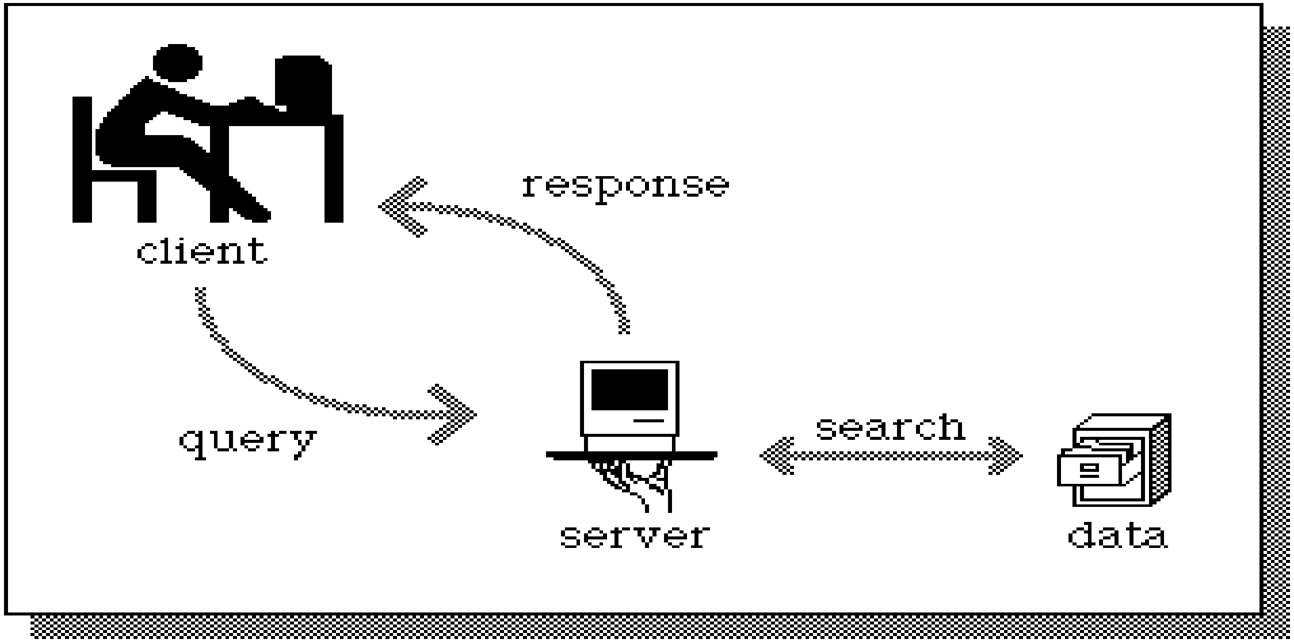
Socket Programming

- In client-server architecture, how does communication happen ? For file transfer, for chat,,
- The service provided by server are run on ports (application identifiers)
- A Single server with multiple ports support multiple applications
- Client needs to know the address of the machine and the appropriate port number
- Note: Server does not need to know about client particulars (address)
- Typically, when a client requests a service (first packet), the address of the client is known to the server
- Client is an ACTIVE device; requests for service
- Server is a PASSIVE device; simply waits for requests from client

A simple client server model



Note: In Operating system ~ interprocess communication happens through pipes
Important: communication happens within the same system

How do we establish communication between two applications on different machines ?
SOCKETs

Like PIPEs, socket acts as a communication channel

More about sockets

- Server and client exchange message over the network through a common socket-API

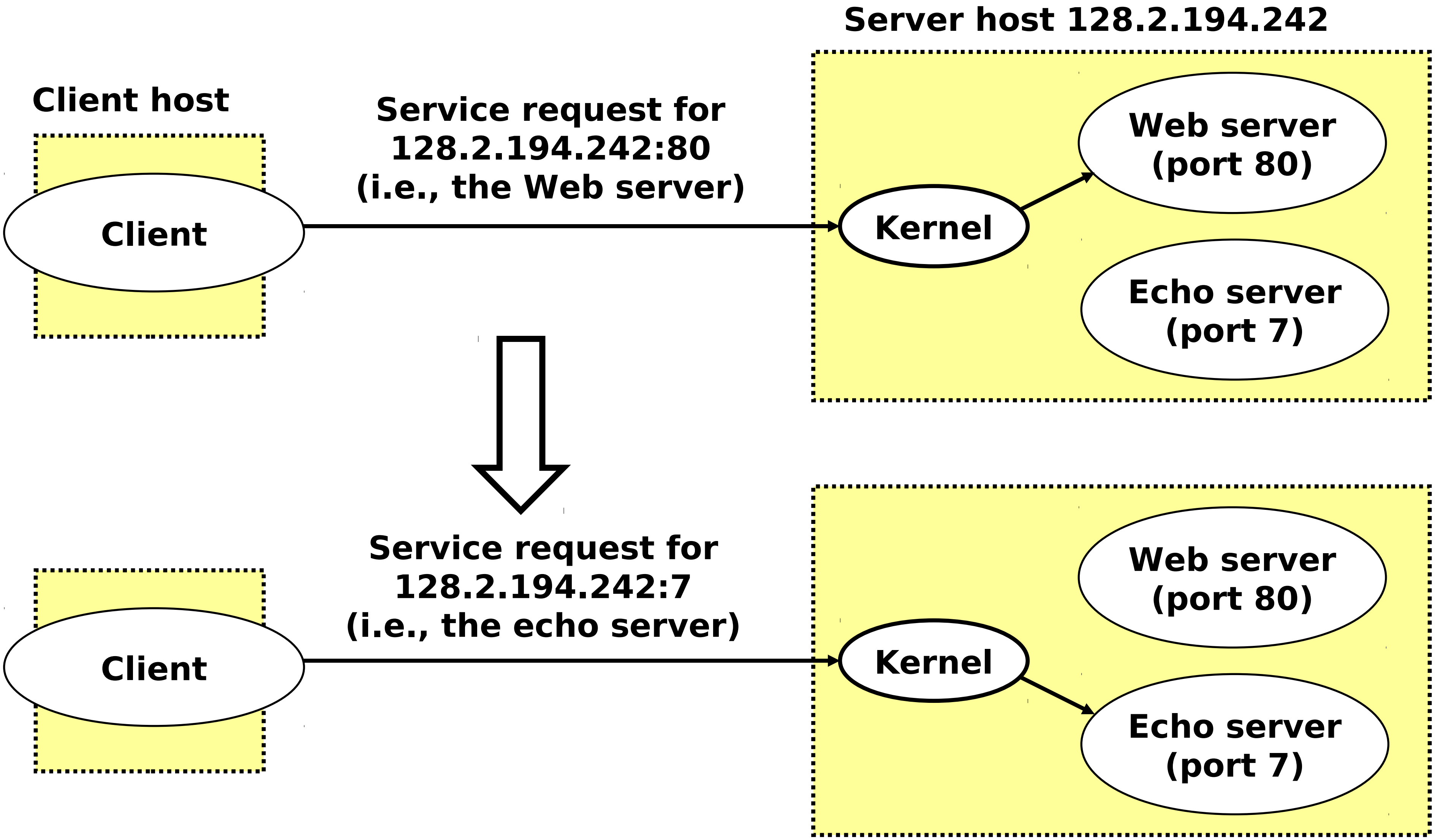
- ➔ API: Application Programming Interface ~ A set of routines that an application uses to carry out lower-level service (Reading/writing a file descriptor)

- ➔ Typical Server Examples: WebServer (port 80), FTP Server (port 20,21), Telnet (23), Mail Server (25), Echoserver (7)

- ➔ Client Examples: Web browser, telnet, ftp, ssh

Using Ports to Identify Services

See
/etc/services



Establishing a channel

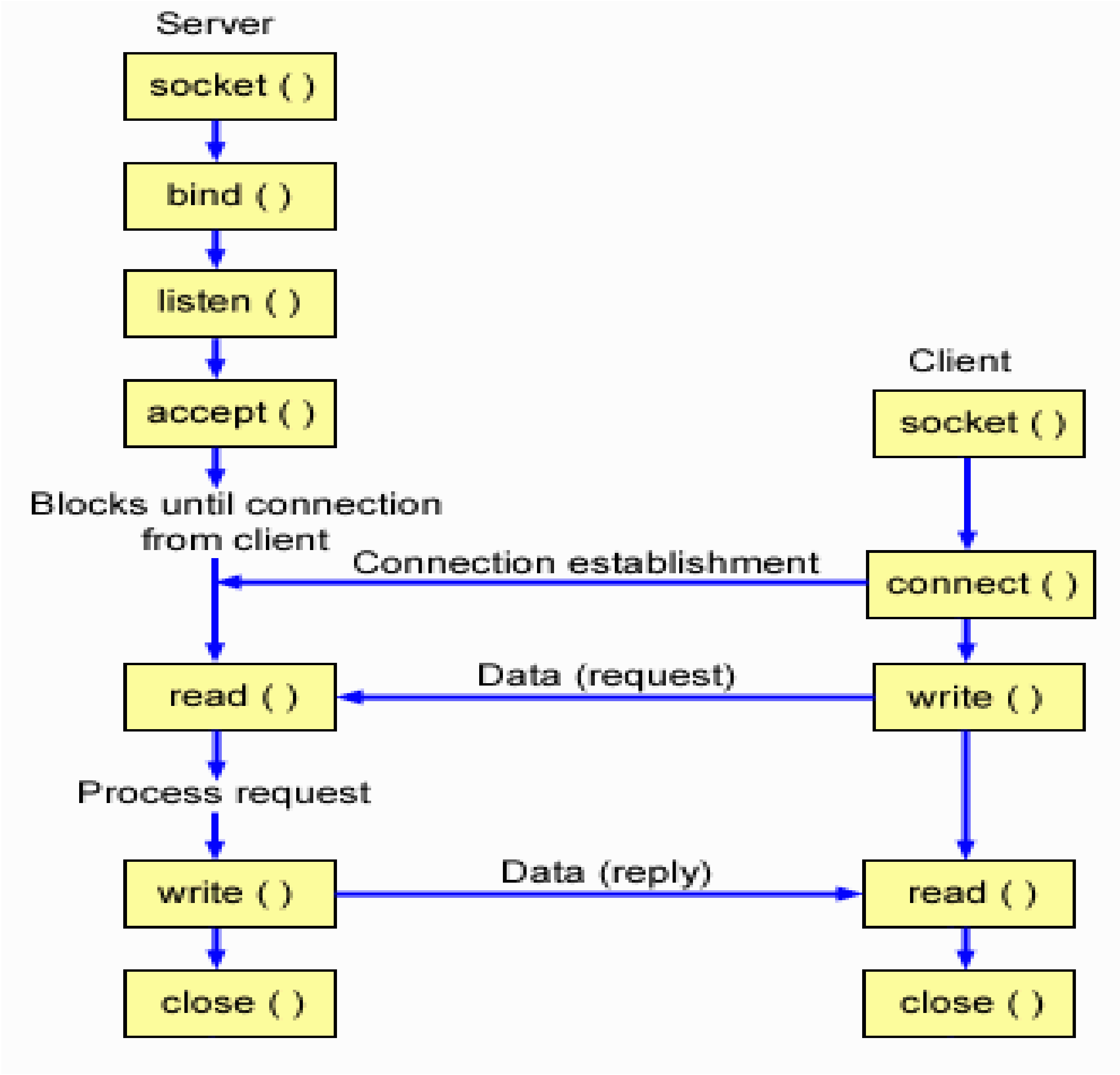
Steps followed by client to establish the connection:

- Create a socket
- Connect the socket to the address of the server
- Send/Receive data
- Close the socket

Steps followed by server to establish the connection:

- Create a socket
- Bind the socket to the port number known to all clients
- Listen for the connection request
- Accept connection request
- Send/Receive data

A view through system calls



Types of Sockets

SOCK_STREAM

- ✗ TCP
- ✗ Connected-Oriented
- ✗ Reliable Delivery
- ✗ In-order Guaranteed
- ✗ Bidirectional

SOCK_DGRAM

- ✓ UDP
- ✓ Connection less
- ✓ Unreliable delivery
- ✓ Out of order delivery
- ✓ Send or receive

Types of Sockets

SOCK_STREAM

- ✗ TCP
- ✗ Connected-Oriented
- ✗ Reliable Delivery
- ✗ In-order Guaranteed
- ✗ Bidirectional

SOCK_DGRAM

- ✓ UDP
- ✓ Connection less
- ✓ Unreliable delivery
- ✓ Out of order delivery
- ✓ Send or receive

Socket Creation

- *#include <sys/types.h>*
- *#include <sys/socket.h>*
- *int socket (int family, int type, int protocol);*

- *Family: protocol family ~ AF_NET for Internet and PF_NET for TCP/IP*
- *Type: type of service ~ SOCK_STREAM or SOCK_DGRAM*
- *Protocol: specific protocol, if value=0 then default protocol*

SOCKET() system call return a socket descriptor (like file descriptor in PIPEs) or -1 on error

How do we deal with end point addressing ? ~ IP address, Port number

Socket Address Structure

For IPv4 the socket structure is defined as:

```
struct sockaddr
{
u_short sa_family;   /* address family */
Char  sa_data[14];   /* up to 14 bytes of direct address */
};
```

This is the generic structure that most socket APIs accept, but the structure we will work with is

sockaddr_in (socket address internet).

```
struct sockaddr_in
{
Short sin_family;
u_short sin_port;
struct in_addr sin_addr;
Char sin_zero[8];
};
```

struct in_addr found inside sockaddr_in is a union defined as:

```
struct in_addr
{
union
{
struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
struct { u_short s_w1,s_w2; } S_un_w;
u_long S_addr;
} S_un;


#define s_addr S_un.S_addr
#define s_host S_un.S_un_b.s_b2
#define s_net S_un.S_un_b.s_b1
#define s_imp S_un.S_un_w.s_w2
#defines_impno S_un.S_un_b.s_b4
#define s_lh S_un.S_un_b.s_b3
};
```

This structure holds the IP address which can be accessed in many ways.

Binding a socket

#include <sys/types.h>

#include <sys/socket.h>

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

Sockfd ~ is the socket file descriptor returned by socket().

my_addr is a pointer to a struct sockaddr that contains information about your address, namely, port and IP address.

addrlen can be set to sizeof *my_addr or sizeof(struct sockaddr)

Other system calls (listen, connect)

```
#include<sys/socket.h>
int listen(int skfd, int backlog);
```

skfd is the socket descriptor of the socket on which the machine should start listening.
backlog is the maximum length of the queue for accepting requests.

Note: The connect system call signifies that the server is willing to accept connections and thereby start communicating.

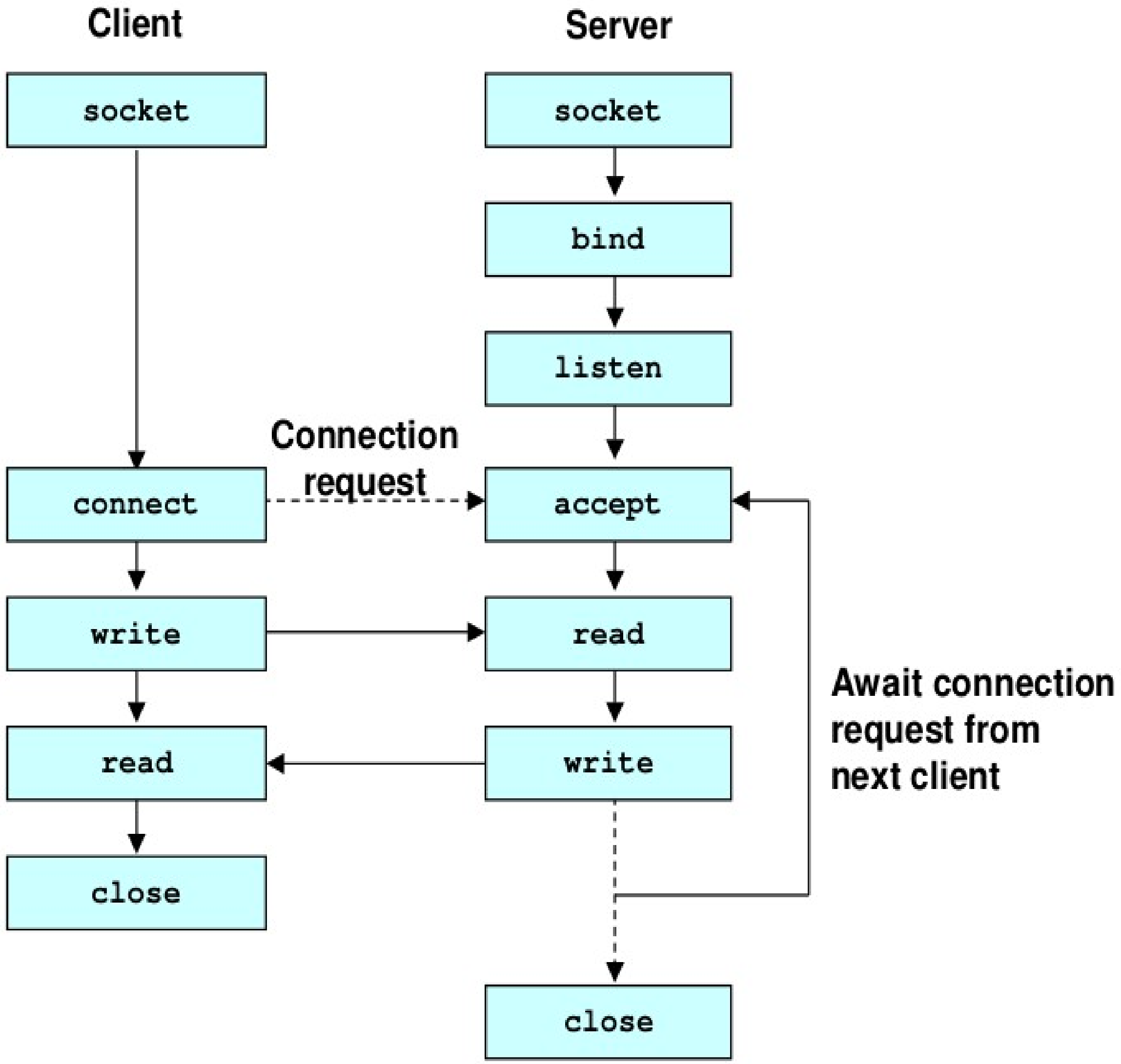
```
#include<sys/socket.h>
#include<netinet/in.h>
/* only for AF_INET , or the INET Domain */

int connect(int skfd, struct sockaddr* addr, int addrlen);
int accept(int skfd, struct sockaddr* addr, int addrlen);
int recv(int skfd, void *buf, int buflen, int flags);
int send(int skfd, void *buf, int buflen, int flags);
```

Food for thought

Byte ordering: client may follow little endian and server may be big endian ~ how do we handle compatability issues ?

Should we discover something new ~ say ~ network byte ordering ~ both client and server must follow network byte ordering

Socket programming *with TCP*


Client
Server
socket
socket
bind
listen
Connection request
connect
accept
write
read
read
write
Await connection request from next client
close
close