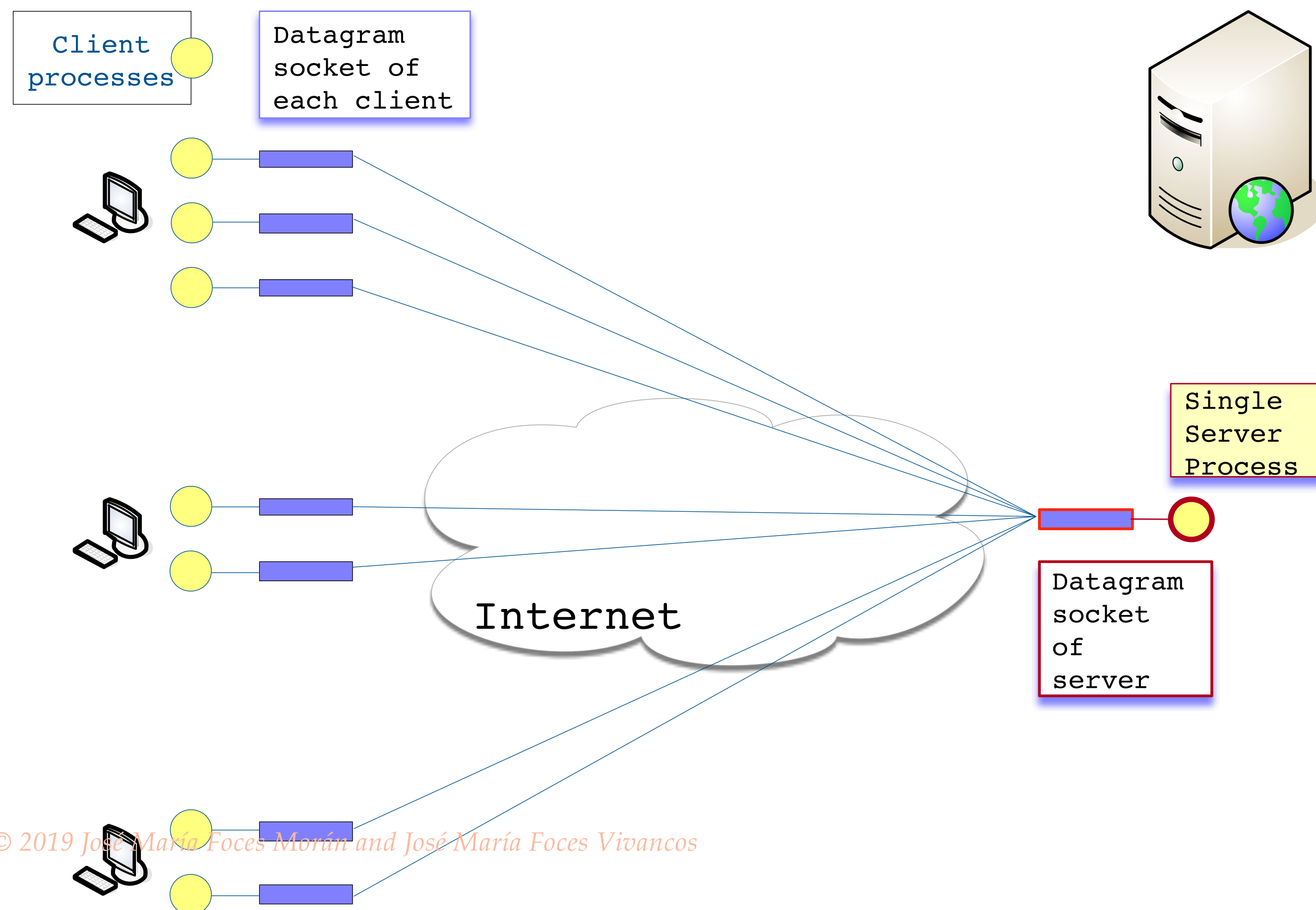
STREAM SOCKETS

THE INTERFACE TO THE TCP PROTOCOL

All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Datagram socket C/S model

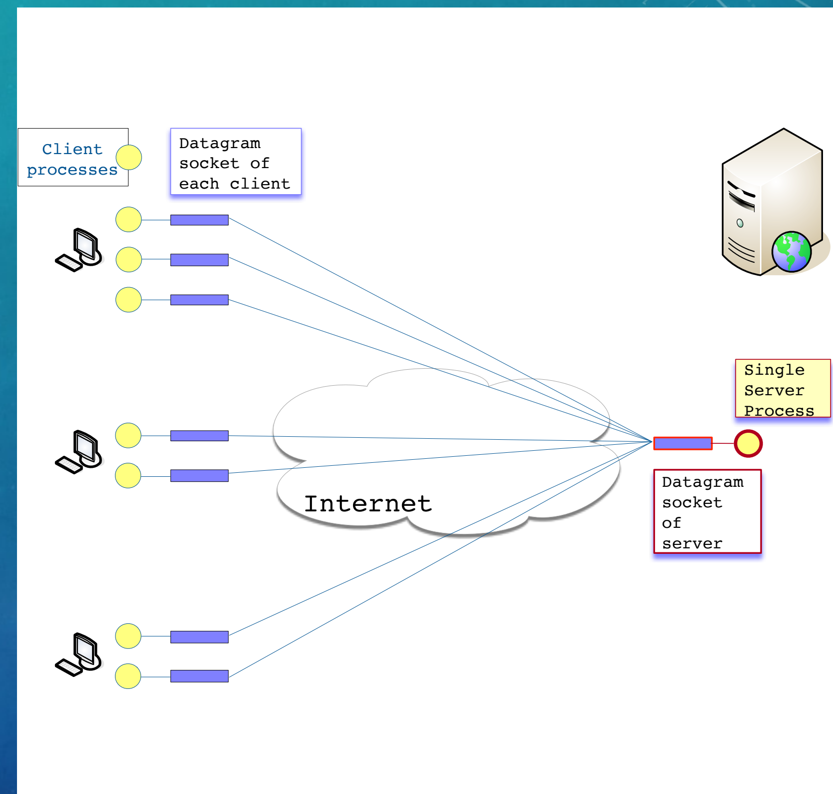
V 1.4 All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

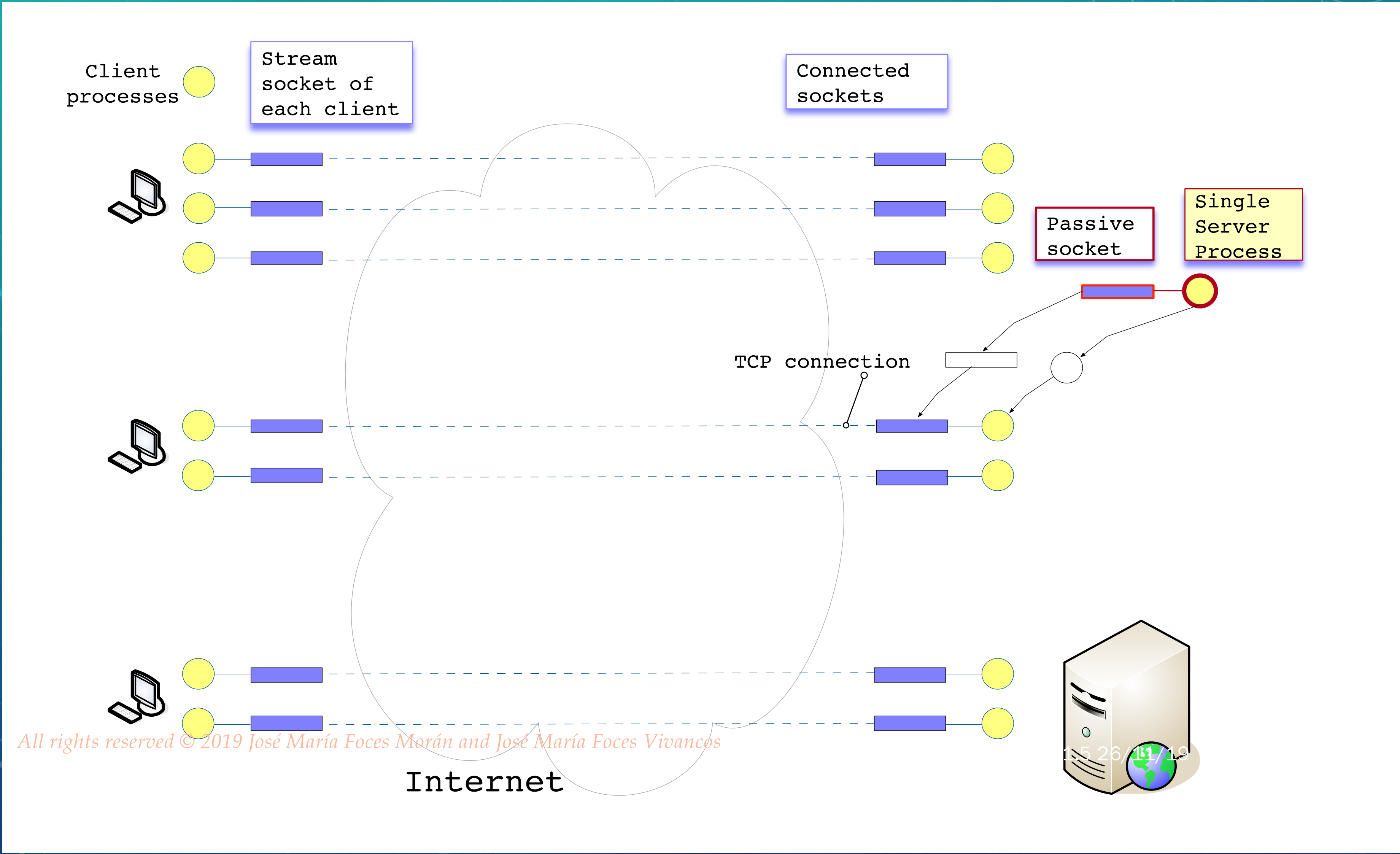
Datagram socket C/S model

- One server is started listening on a port
- Creates one Datagram Socket
- Many UDP clients Access it
- All traffic from all clients is received by this single socket
- This single socket will pass the traffic from all clients to the server process



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

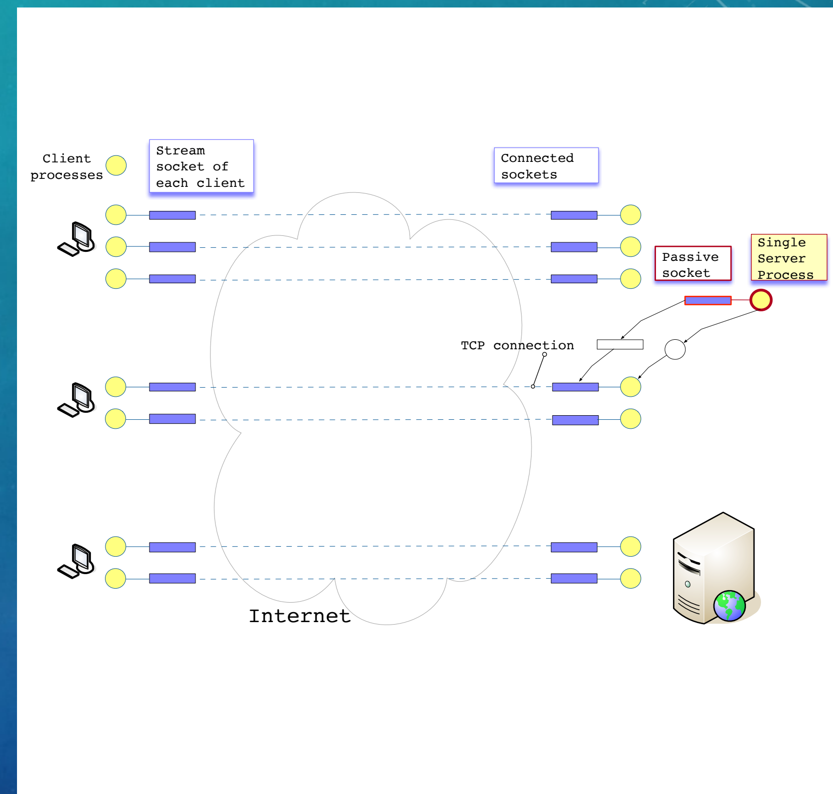
Stream socket C/S model



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Stream socket C/S model

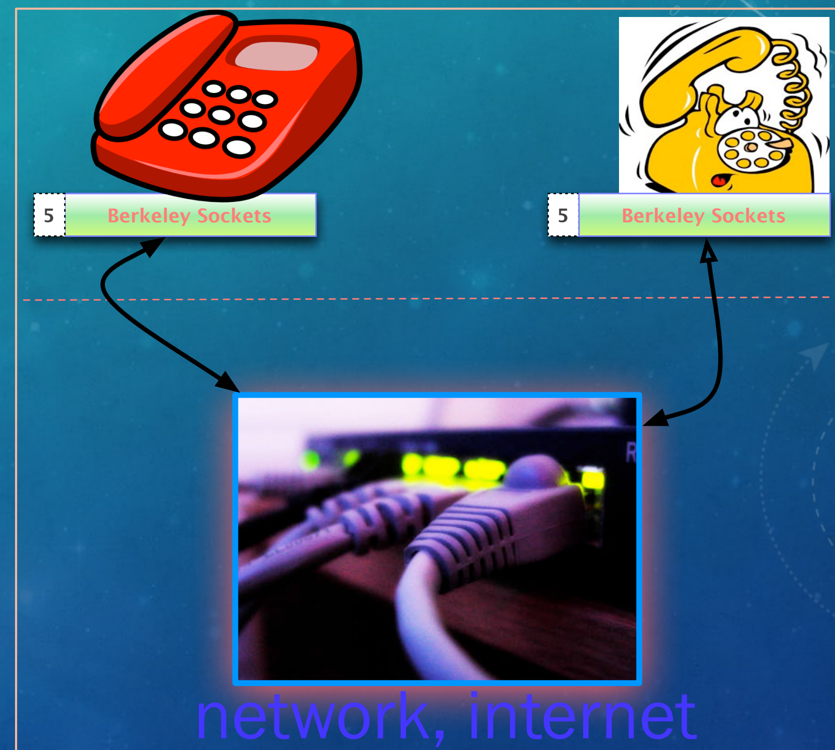
- Server process creates passive stream socket
  - **Welcome socket**
- Passive stream socket accepts new connection requests
  - Creates a new connected socket for each client
  - **Delegate socket**
  - Each connected socket is handled by a new server thread



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Stream Sockets: The interface to TCP

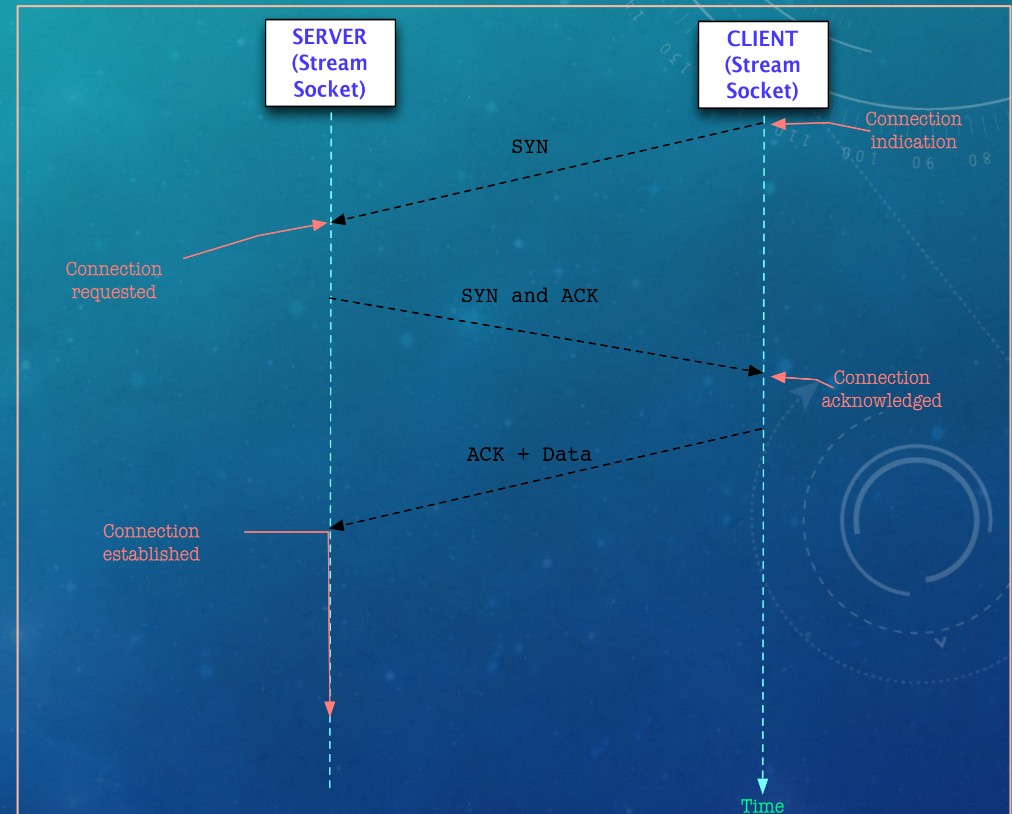
- TCP is connection oriented
- The client contacts the server and both establish the parameters of the communication
- This might resemble the dialing from a phone to another phone



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

TCP connection created by enacting **3-way handshake**

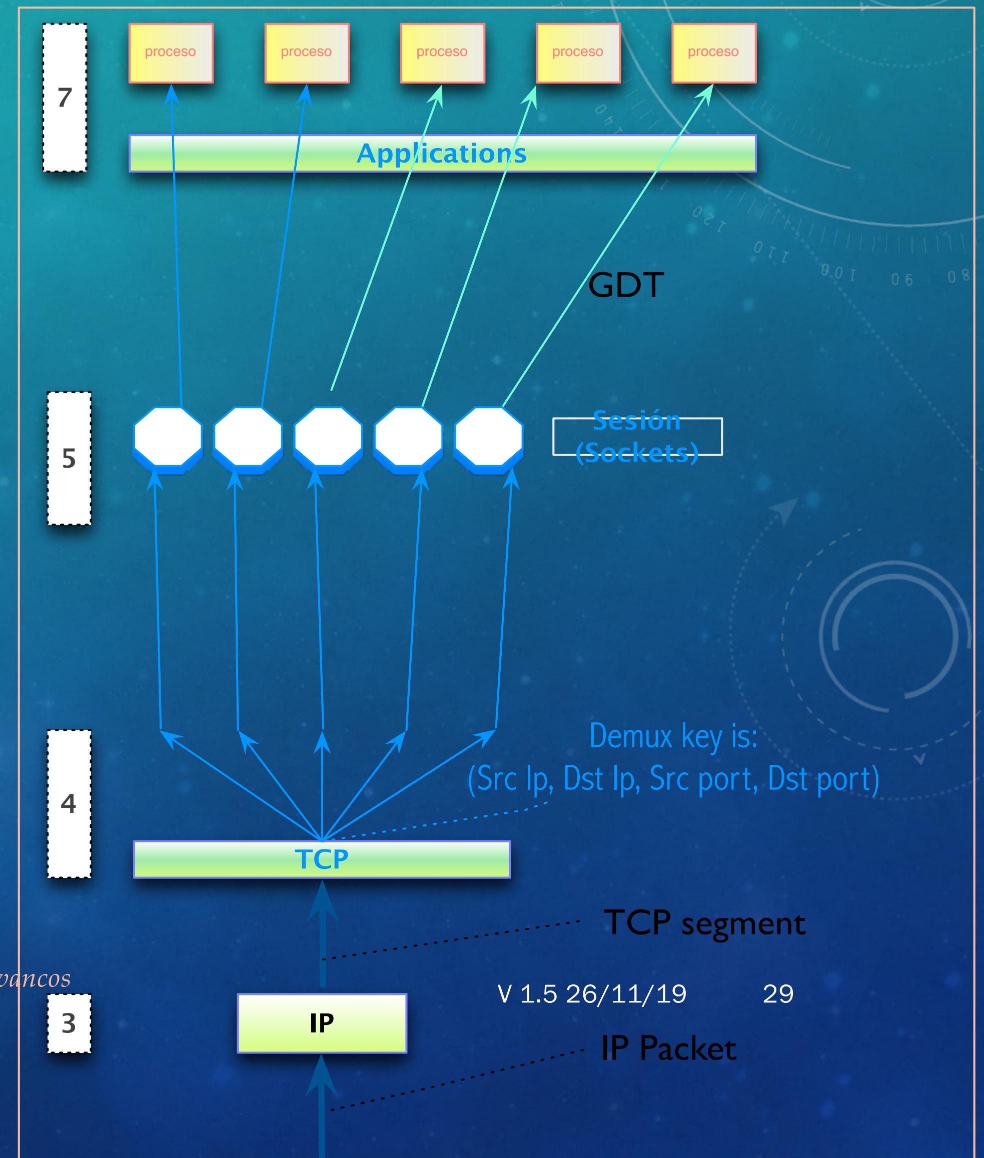
- A typical TCP connection is established by having the Client and the Server Exchange 3 messages:
  - C -> S: SYN
  - S -> C: ACK and SYN
  - C -> S: ACK



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

The multiplexing keys at work in TCP

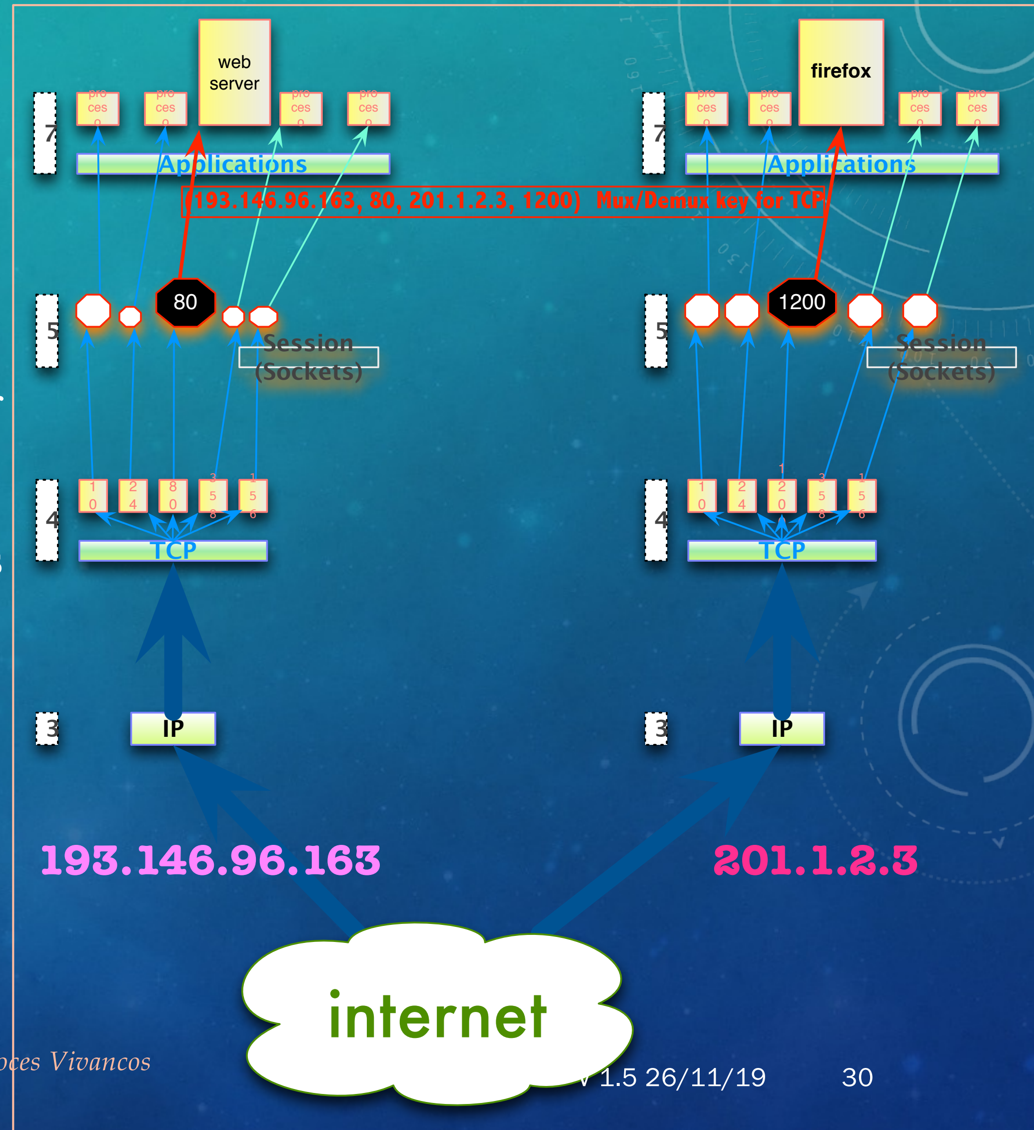
- Each connection represents a bidirectional flow between two processes
  - The Client (C)
  - The Server (S)
- Each process creates a socket that has a full sockaddr_in
  - IP
  - Port
- Therefore, the TCP mux key is comprised of four numbers
  - Client IP
  - Client Port
  - Server IP
  - Server Port



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

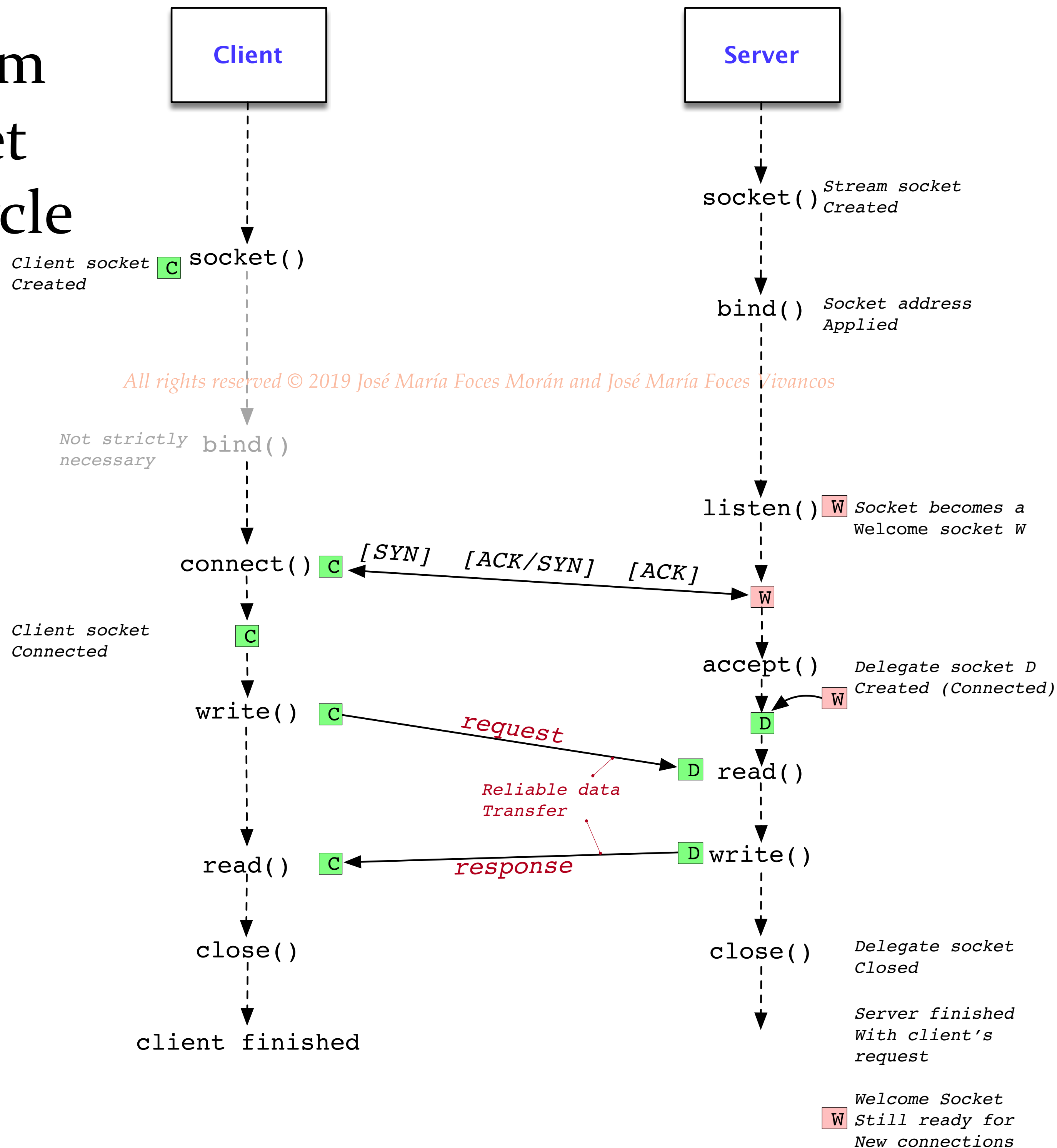
Example about mux keys in TCP

- Server is started at TCP port 80
- Client connects with server
  - Its local port is 1200
- The TCP multiplexing key is
  - 193.146.99.163
  - 80
  - 201.1.2.3
  - 1200
- It is used in the C stack and in the S stack for locating the C process and the S process respectively



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Stream socket lifecycle



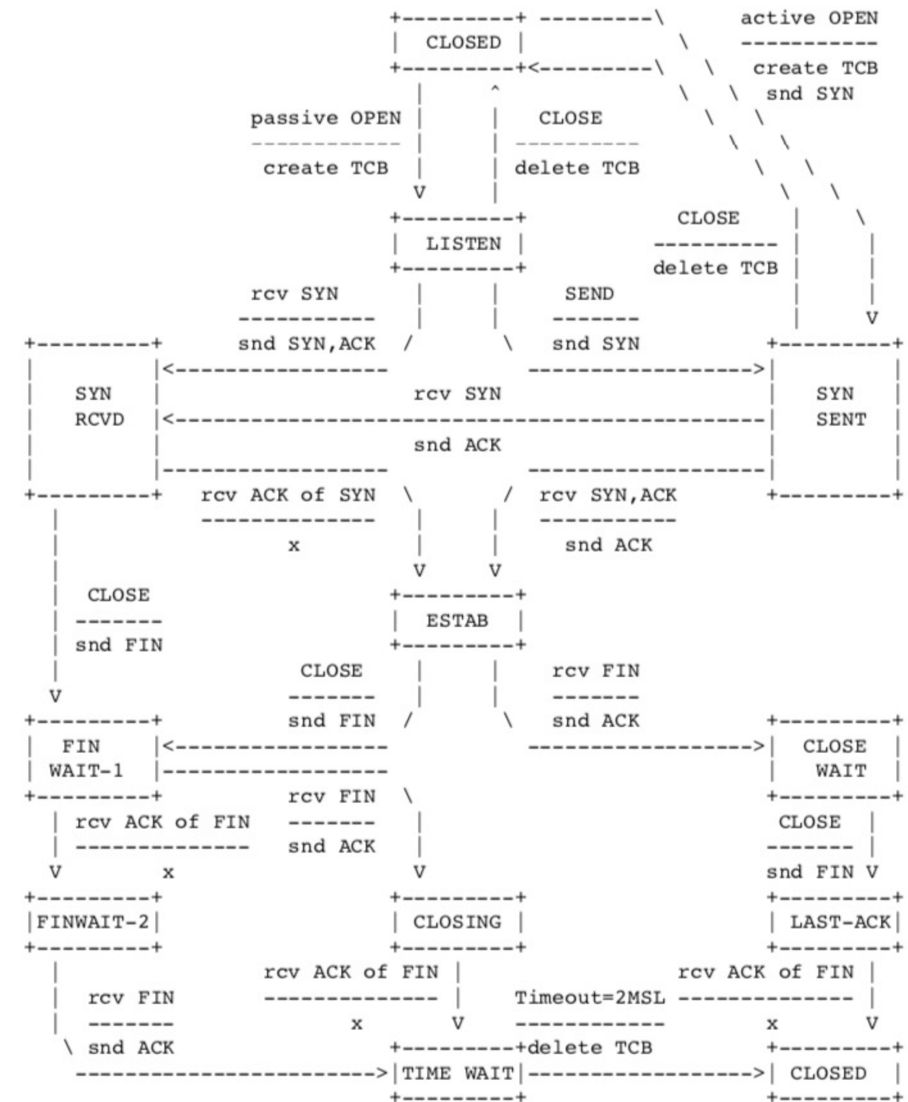
All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

TCP socket state diagram

- RFC 793 specifies TCP
- TCP state diagram represents the state changes of stream sockets
  - Citing RFC 793, **Closed** is a "fictional" state
  - **Listen** applies to the Welcome Socket only
  - **Estab** is the state the connected sockets are in when they Exchange Data. The Sliding Window algorithm governs *reliable* data transfer

All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

- State changes are caused by a socket receiving a legitimate and expected protocol message, e.g., SYN
  - The receiving socket, in general, also sends some response protocol message, e.g., ACK-SYN after receiving SYN

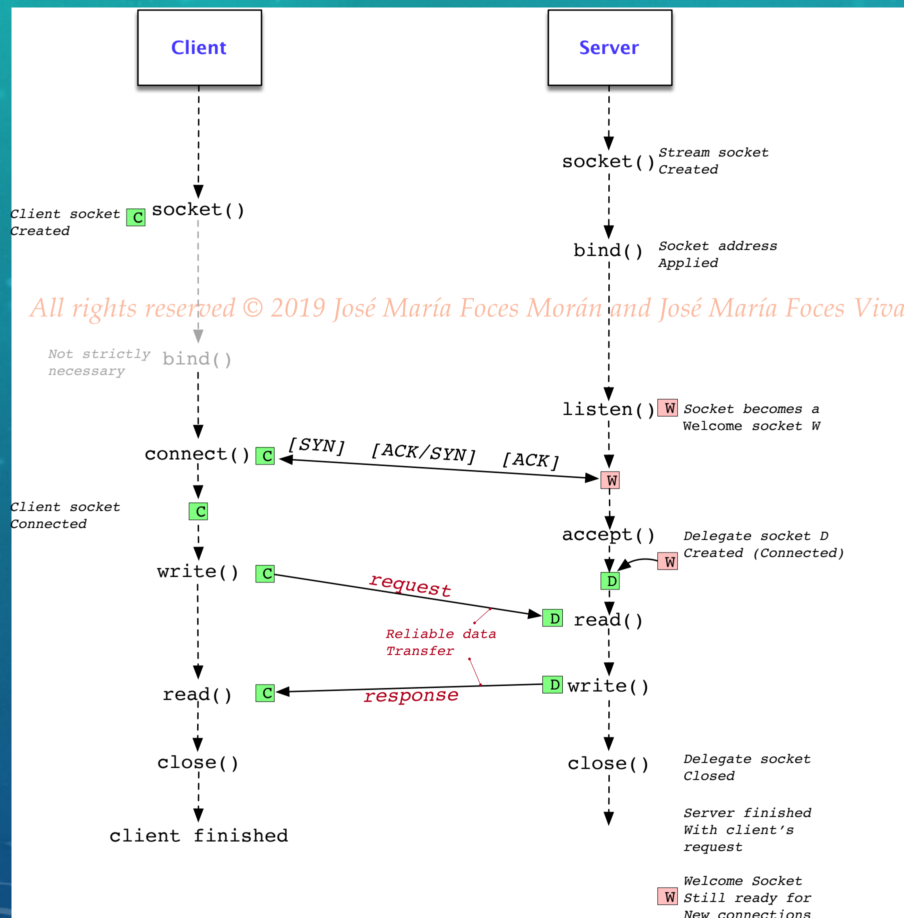


TCP Connection State Diagram
Figure 6.

Verbatim copy of RFC 793 TCP State diagram in pg. 23

[Page 23]

socket() call for a stream socket



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Create a Stream socket (TCP):

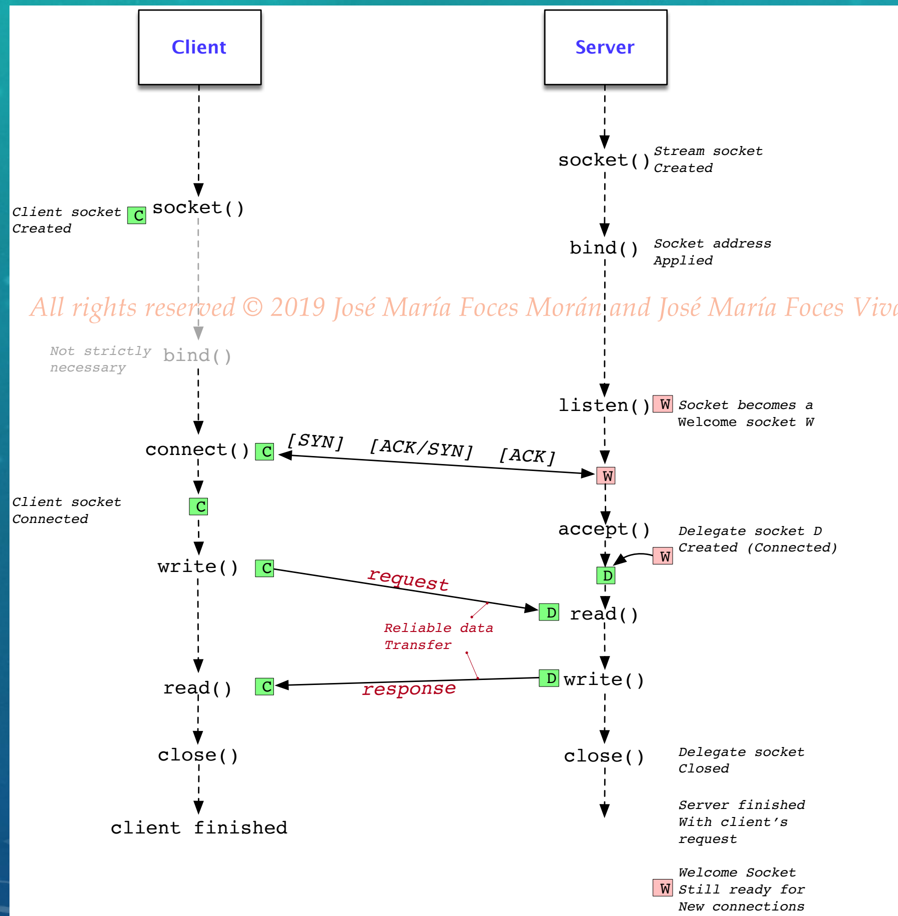
```
#include <sys/socket.h>

fd = socket(domain, type, protocol);
```

Domain: **AF_INET**; AF_INET6

Type: SOCK_DGRAM, **SOCK_STREAM**

Protocol: 0

bind() call for the Welcome socket

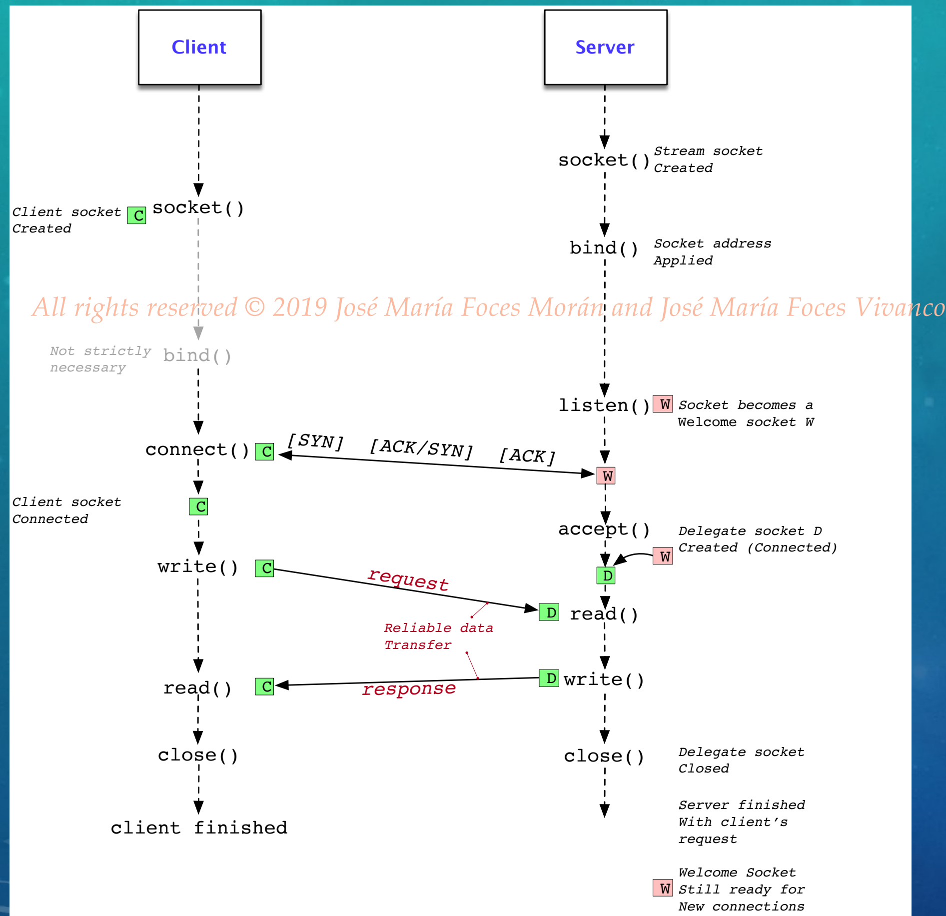


All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Server

```
socketAddress.sin_family = AF_INET;

int port = atoi(argv[1]);

socketAddress.sin_port = htons(port);

socketAddress.sin_addr.s_addr = INADDR_ANY;

…

bind(

welcomeSocket,

(struct sockaddr *) &socketAddress,

sizeof (socketAddress)

);


#include <sys/socket.h>


int bind(int fd,

const struct sockaddr *addr,

socklen_t addrlen);
```

listen() call for the Welcome socket



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Server

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

listen(welcomeSocket, 5);

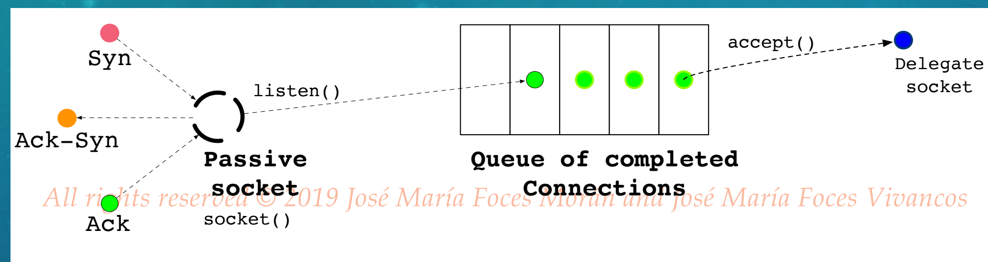
Sets welcomeSocket as a welcome socket and sets the length of the queue of **completed connections** (The **backlog**) to 5.

As the welcome socket receives each TCP connection request it stores each completed connection request in the backlog queue.

A later call to **accept()** on the welcomeSocket will **extract** the completed connection on the queue head and turn it into a fully functional delegate socket.

This doc. was obtained from $ man listen in a kernel which version is greater than **Linux 2.2**

listen() call for the Welcome socket



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Server

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

listen(welcomeSocket, 5);

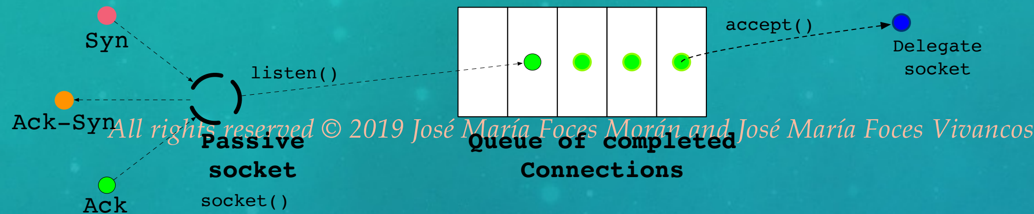
Sets welcomeSocket as a welcome socket and sets the length of the queue of **completed connections** (The **backlog**) to 5.

As the welcome socket receives each TCP connection request it stores each completed connection request in the backlog queue.

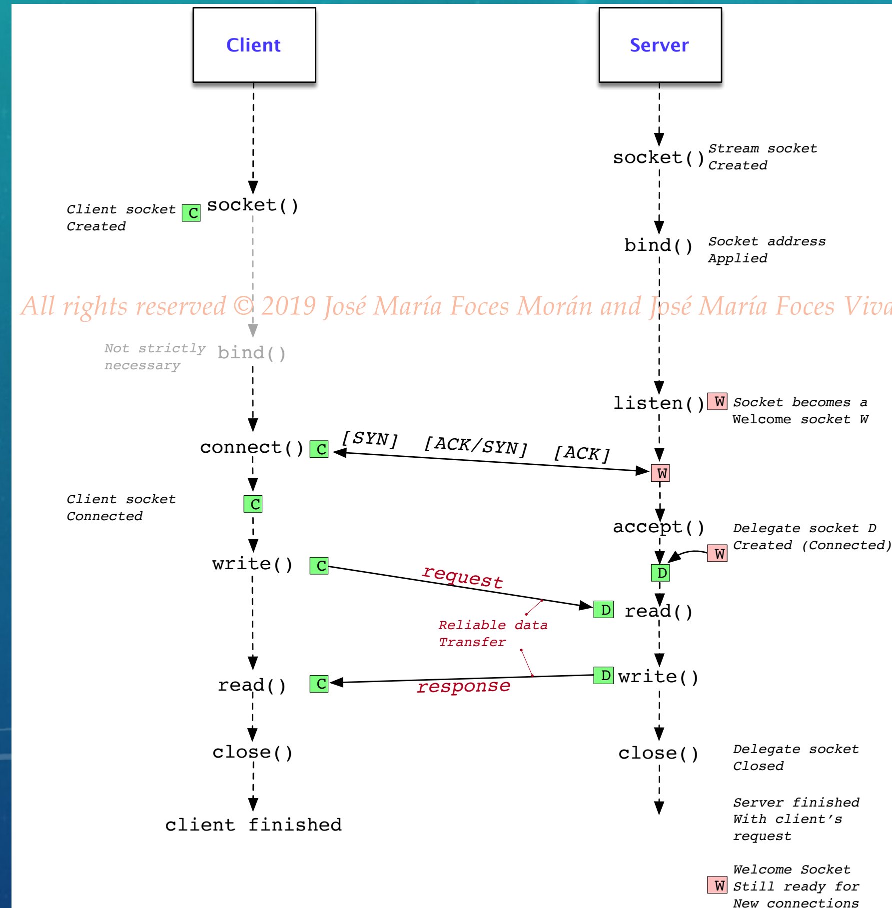
A later call to **accept()** on the welcomeSocket will **extract** the completed connection on the queue head and turn it into a fully functional delegate socket.

This doc. was obtained from $ man listen in a kernel which version is greater than **Linux 2.2**

accept() call for the Welcome socket



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

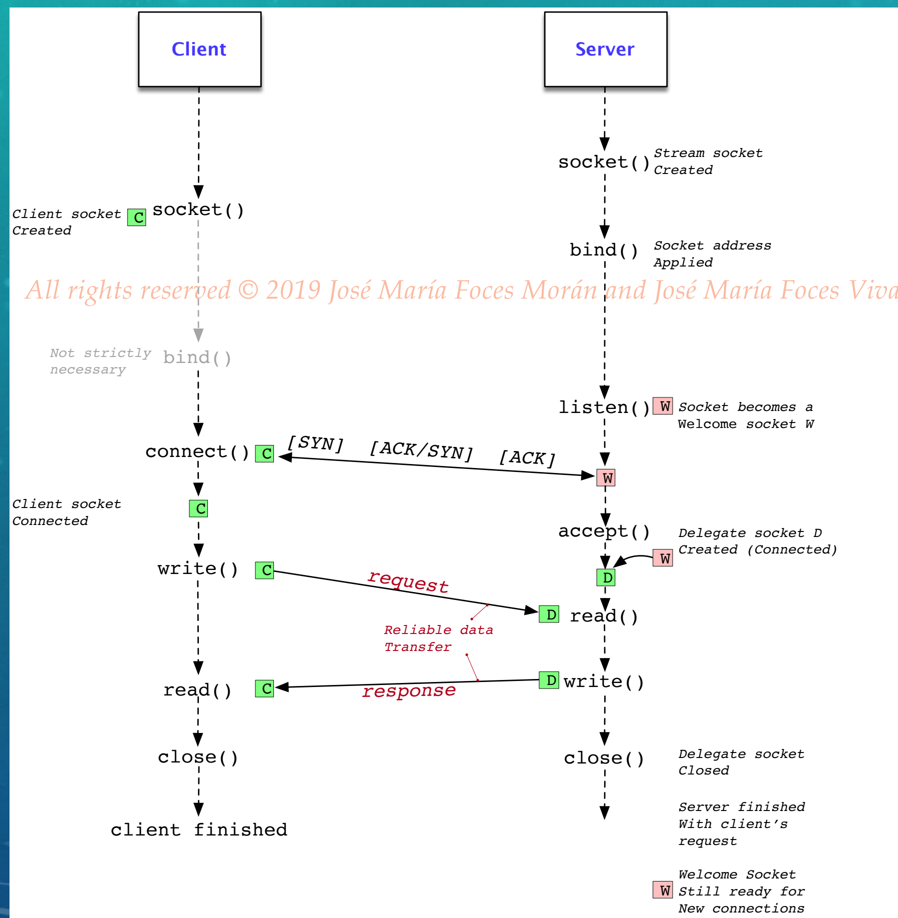
Server

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


int delegateSocket = accept(

welcomeSocket,

(struct sockaddr *)
&clientAddress,

&addressLength

);
```

- welcomeSocket must be in the listen state

- **accept()** extracts the connection on the queue head and turn it into a fully functional **delegate** socket. This socket allows **reliable bidirectional data transfer**

connect() call for the Client socket

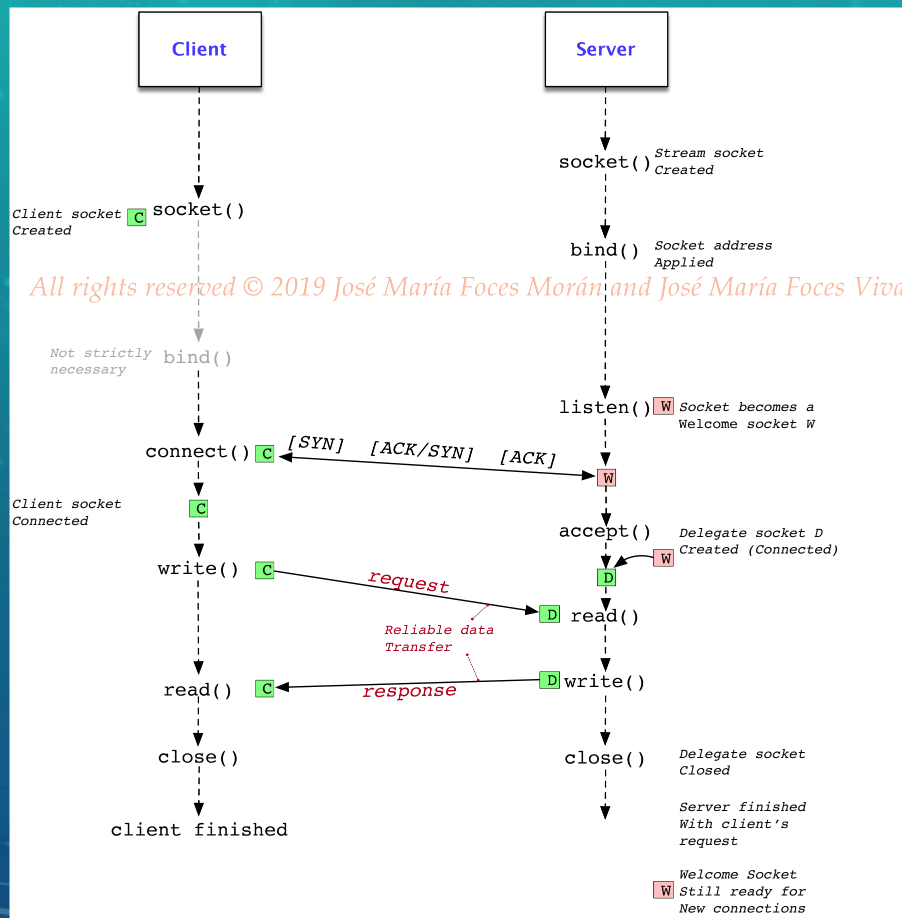


All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Client: connection to server

```
struct sockaddr_in server;
    server.sin_family =
AF_INET;
    server.sin_port =
htons(port);
    server.sin_addr.s_addr =
inet_addr(ipAddress);

int r = connect
(
sock,
(struct sockaddr *) &server,
sizeof (server)
);
```

listen() call for the Welcome socket



All rights reserved © 2019 José María Foces Morán and José María Foces Vivancos

Server and client

read() and write() system calls do result convenient with stream sockets

- The file descriptor represents the TCP connection

- Writing means sending from the side that calls write() to the other side

- Reading is exactly the opposite