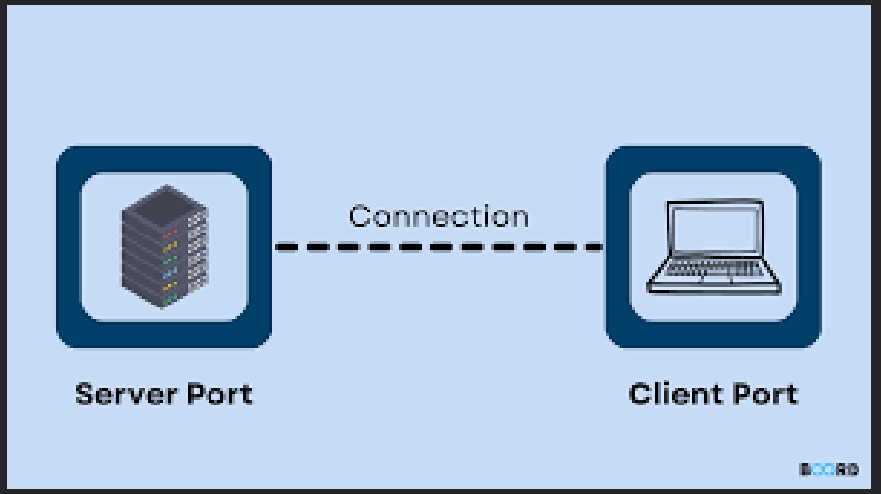
# Programación con Sockets

# Definición

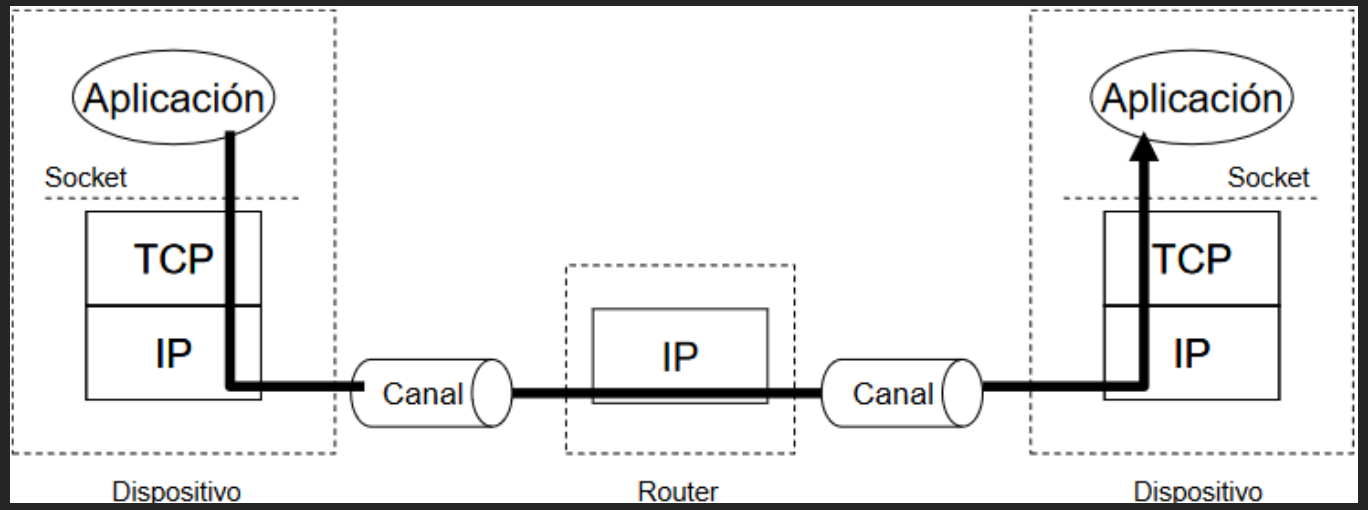
- Los sockets son una de las herramientas que ofrecen los Sistemas Operativos para la comunicación entre diferentes procesos.
- La particularidad que tienen frente a otros mecanismos de comunicación entre procesos (IPC – Inter-Process Communication) es que:

  *Posibilitan la comunicación aun cuando ambos procesos estén corriendo en distintos sistemas unidos mediante una red.*
- De hecho, el API de sockets es la base de cualquier aplicación que funcione en red

  *Ofrece una librería de funciones básicas que el programador puede usar para desarrollar aplicaciones en red*



# Sockets TCP/IP




- Permiten la comunicación de dos procesos que estén conectados a través de una red TCP/IP.

  *Cada máquina está identificada por medio de su dirección IP*

  *Cada máquina puede estar ejecutando múltiples procesos simultáneamente.*

  - Cada uno de estos procesos se asocia con un número de puerto

  *Un socket se identifica unívocamente por la dupla dirección IP + número de puerto.*

- Una comunicación entre dos procesos se identifica mediante:

  *La asociación de los sockets que estos emplean para enviar y recibir información hacia y desde la red*

  identificador de socket origen + identificador de socket destino.

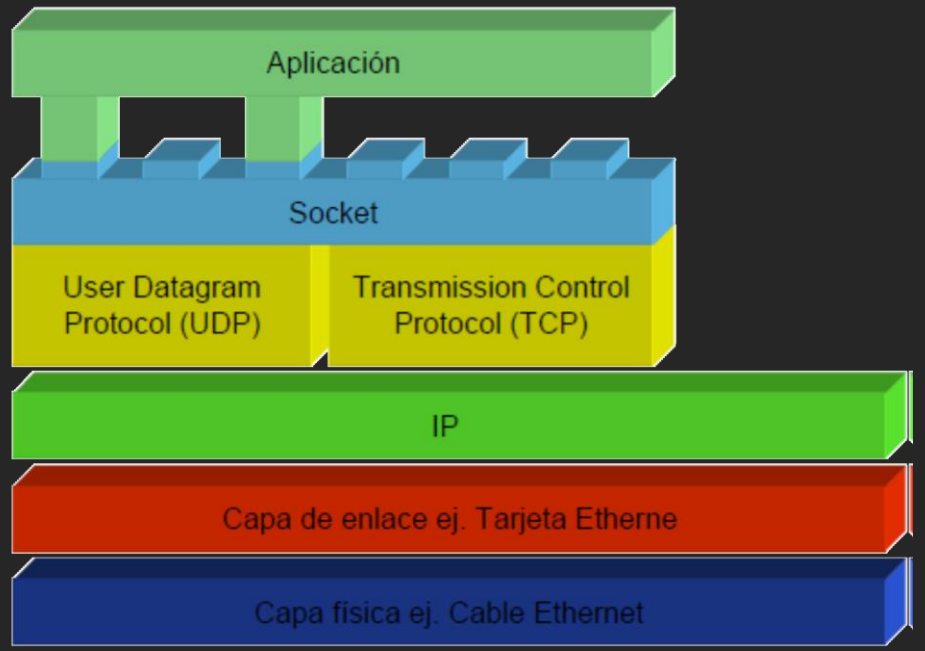
# Protocolos y sockets

- Un socket es una abstracción a través de la cual una aplicación puede enviar y recibir información

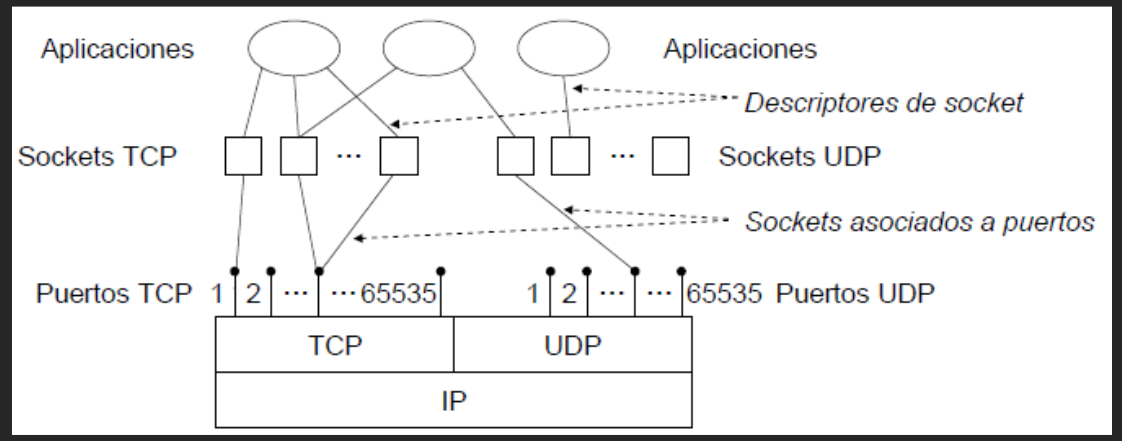
  *Muy similar a como se escribe y lee de un archivo.*

  *La información que una aplicación envía por su socket origen puede ser recibida por otra aplicación en el socket destino y viceversa.*

- Existen diferentes tipos de sockets dependiendo de la pila de protocolos sobre la que se cree dicho socket.



# Sockets, Protocolos y Puertos




- Relación lógica entre aplicaciones, sockets, protocolos y puertos en un dispositivo.

  *Aspectos por destacar en esta relación*

  - Primero, un programa puede usar más de un socket al mismo tiempo
  - Segundo, diferentes programas pueden usar el mismo socket al mismo tiempo,

    *Cada socket tiene asociado un puerto TCP o UDP, según sea el caso.*

    *Cuando se recibe un paquete dirigido a dicho puerto, este paquete se pasa a la aplicación correspondiente.*

# Dominios de comunicación

- Los sockets se crean dentro de lo que se denomina un dominio de comunicación que define cual será la pila de protocolos que se usará en la comunicación

| Dominio | Proposito |
|---|---|
| PF_UNIX, PF_LOCAL | Procesos que se comunican en un mismo sistema UNIX |
| PF_INET | Procesos que se comunican usando una red IPv4 |
| PF_INET6 | Procesos que se comunican usando una red IPv6 |

| | |
|---|---|
| PF_IPX | Procesos que se comunican usando una red Novell |
| PF_NETLINK | Comunicación con procesos del kernel |
| PF_X25 | ITU-T X.25 / ISO-8208 protocol |
| PF_AX25 | Amateur radio AX.25 |
| PF_ATMPVC | Acceso a PVCs ATM |
| PF_APPLETALK | Appletalk |
| PF_PACKET | Interfaz con paquetes de bajo nivel |

# Tipos de Sockets

- En el dominio PF_INET se definen los siguientes tipos de sockets:

  *Sockets Stream*

  - hace uso del protocolo TCP que provee un flujo de datos bidireccional, orientado a conexión, secuenciado, sin duplicación de paquetes y libre de errores

  *Sockets Datagram*

  - hacen uso del protocolo UDP, el cual provee un flujo de datos bidireccional, no orientado a conexión, en el cual los paquetes pueden llegar fuera de secuencia, puede haber pérdidas de paquetes o pueden llegar con errores

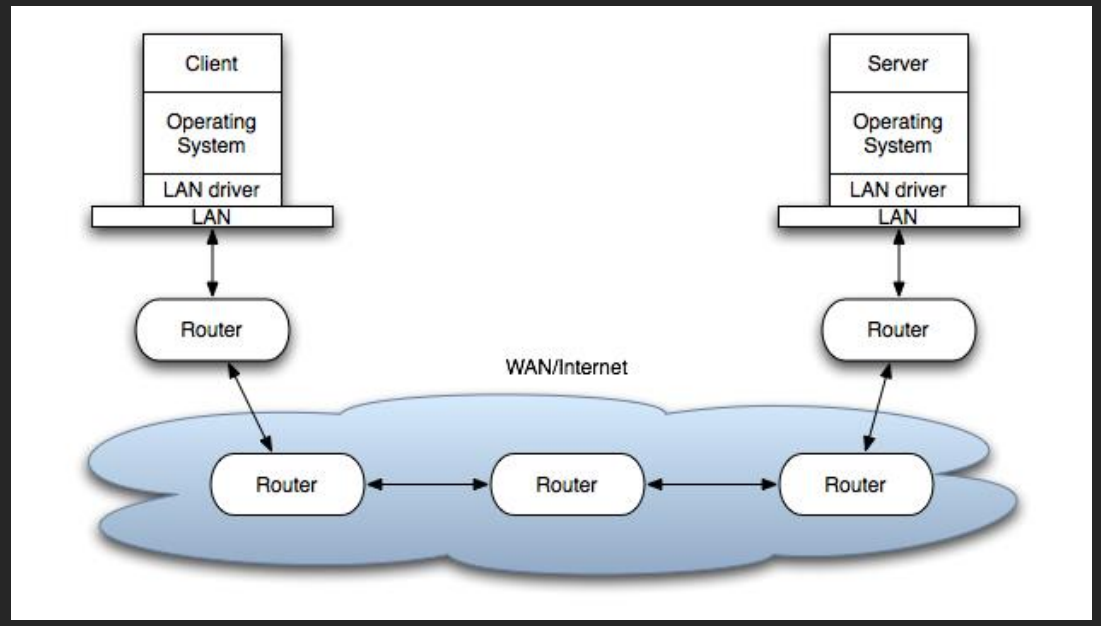
  *Sockets Raw*

  - Permiten un acceso a más bajo nivel, pudiendo acceder directamente al protocolo IP del nivel de Red. Su uso está mucho más limitado ya que está pensado principalmente para desarrollar nuevos protocolos de comunicación, o para obviar los protocolos del nivel de transporte.

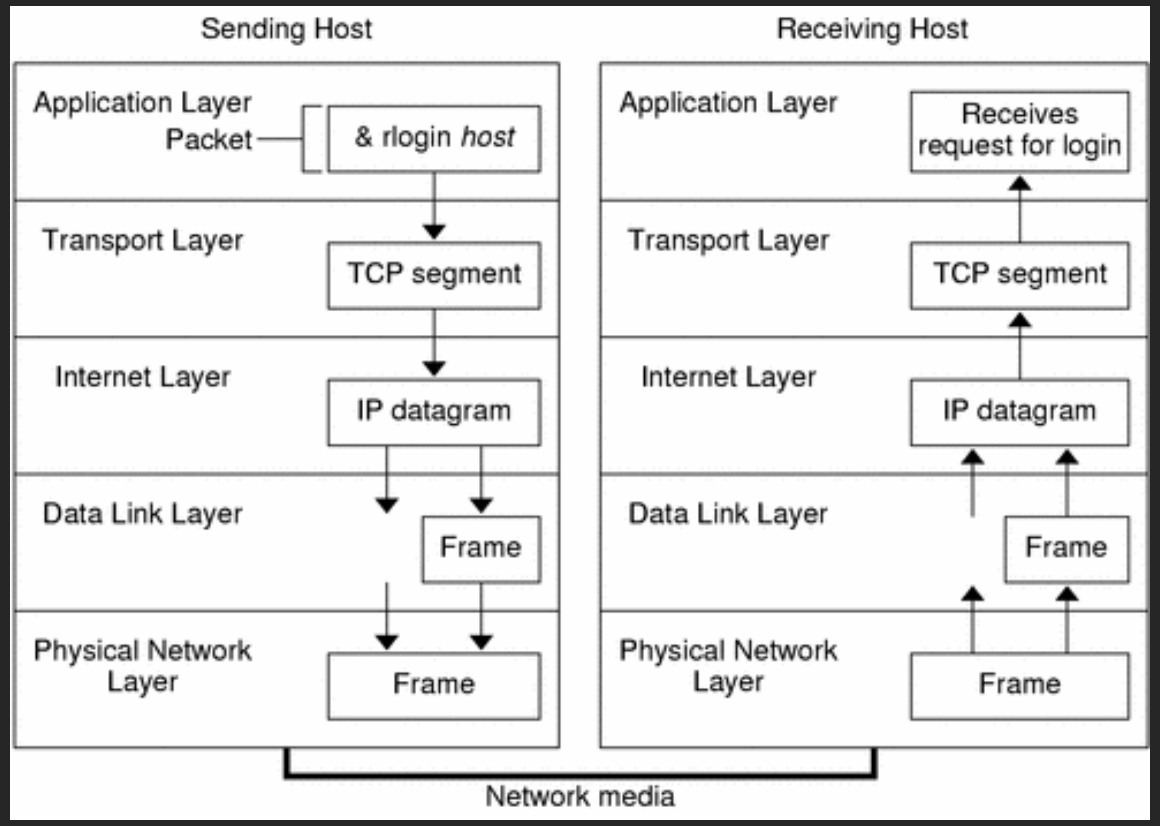
# Modelo Cliente Servidor

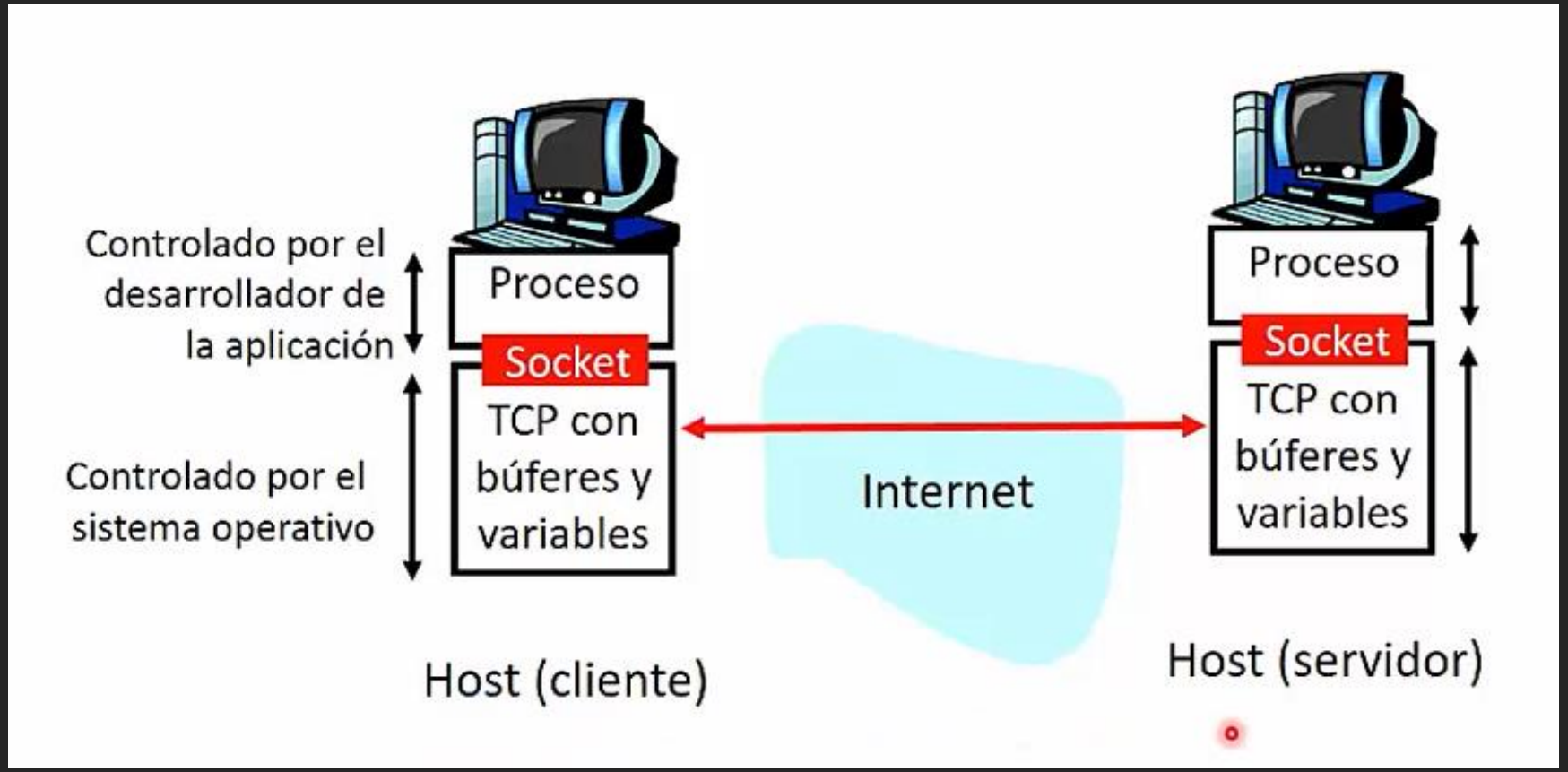
IPcte

IPser

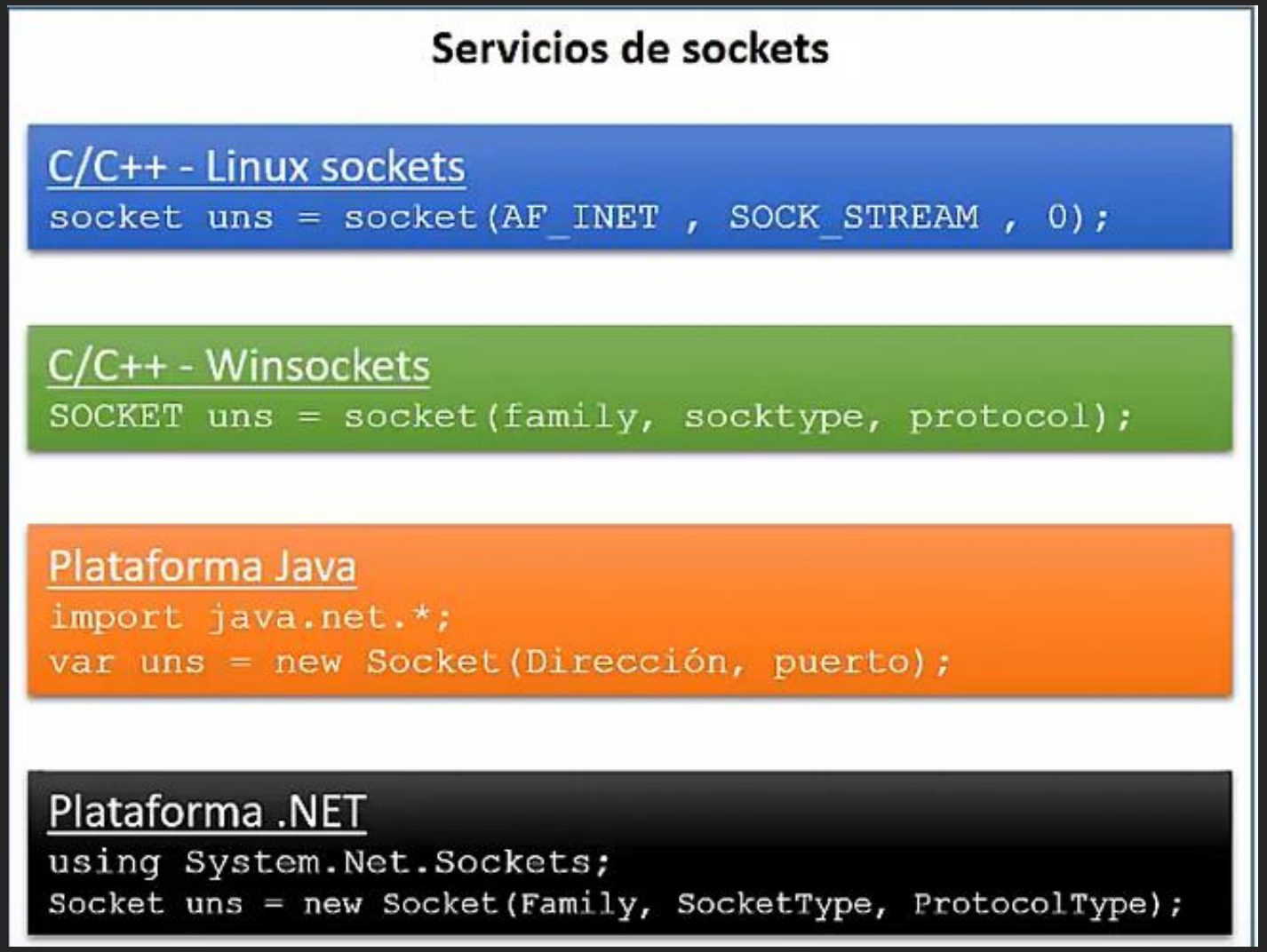



Cliente inicia comunicación






# Programación de Sockets TCP




# Sockets en lenguajes

Cliente

1. Crear un socket local TCP del cliente
2. Indicar la dirección IP y puerto del servidor
3. Activar la conexión con el servidor
4. Enviar y recibir información
5. Cerrar la conexión

Servidor

1. Crear un socket local TCP del servidor
2. Establecer un puerto de escucha
3. Quedar a la espera de peticiones
4. Atender peticiones entrantes
5. Volver al paso 3

---

# Programación de Sockets TCP

| Servicio orientado a la conexión (TCP) | |
|---|---|
| Cliente | Servidor |
| | `socket()` |
| | `bind()` |
| | `listen()` |
| | `accept()` |
| `socket()` | |
| *`[bind()]`* | |
| `connect()` | |
| `send()` | `recv()` |
| `recv()` | `send()` |

# Secuencia de ejecución

# Manejo de socket

- Creación de Socket

  *Los sockets se crean llamando a la función socket(), que devuelve el identificador de socket, de tipo entero*

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
int socket(int domain, int type, int protocol);


#include <stdio.h> /* para perror() */
#include <sys/types.h>
#include <sys/socket.h>
...
int sockfd;
sockfd = socket ( PF_INET, SOCK_STREAM, 0 );
if(sockfd < 0)
perror("Error creating the socket");
```

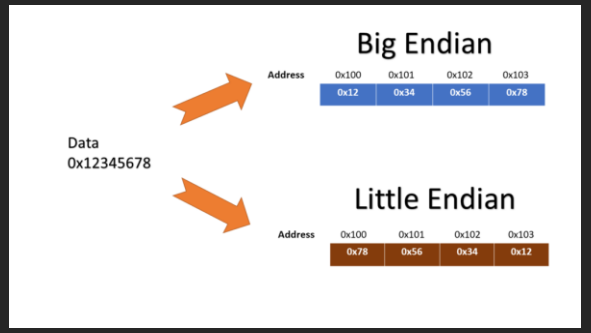
# Manejo de sockets

- Vinculación del socket a una dirección IP y un número de puerto específico

  *La función bind() se utiliza para asociar el socket a una dirección IP y número de puerto de la máquina local a través del que se enviarán y recibirán datos.*

  *El formato de la función es el siguiente:*

```
#include <sys/types.h>
#include <sys/socket.h>
...
int bind(int local_s, const struct sockaddr *addr, int addrlen);
```




```
struct sockaddr_in
{
short int sin_family; // PF_INET
unsigned short sin_port; // Numero de puerto.
struct in_addr sin_addr; // Dirección IP.
unsigned char sin_zero[8]; // Relleno.
};
```

```
...
struct sockaddr_in sin;
...
sin.sin_family = PF_INET;
sin.sin port = htons ( 1234 ); // Número de puerto donde
// recibirá paquetes el programa
sin.sin_addr.s_addr = inet_addr ("132.241.5.10");
// IP por la que recibirá paquetes el programa
```

# Manejo de sockets

- Vinculación

```
...
struct sockaddr_in sin;
...
sin.sin_family = PF_INET;
sin.sin port = htons ( 1234 ); // Número de puerto donde
// recibirá paquetes el programa
sin.sin_addr.s_addr = inet_addr ("132.241.5.10");
// IP por la que recibirá paquetes el programa
```

```
...
struct sockaddr_in sin;
...
ret = bind (sockfd, (struct sockaddr *)&sin, sizeof (sin));
if(ret < 0)
perror("Error binding the socket");
...
```

La llamada a la función `bind()` en el cliente es opcional, ya que en caso de no ser invocada el sistema la ejecutará automáticamente asignándole un puerto libre (al azar). En cambio, en el servidor es obligatorio ejecutar la llamada a la función `bind()` para reservar un puerto concreto y conocido.

# Manejo de sockets

- Escuchar conexiones

  *El servidor escuche las conexiones entrantes mediante la función listen():*

  *El servidor habilita su socket para poder recibir conexiones, llamando a la función listen().*

  - En el cliente este paso no es necesario, ya que no recibirá peticiones de conexión de otros procesos.

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

listen ( sockfd, 5);

# Manejo de Sockets

- Aceptar conexiones

  *Cuando un cliente intenta conectarse, el servidor utiliza la función accept() para aceptar la conexión entrante*

  - Función listen() prepara el socket y lo habilita para recibir peticiones de establecimiento de conexión
  - Función accept() la que realmente queda a la espera de estas peticiones.
  - Cuando una petición realizada desde un proceso remoto (cliente) es recibida, la conexión se completa en el servidor siempre y cuando éste esté esperando en la función accept().
  - La función accept() es utilizada en el servidor una vez que se ha invocado a la función listen().
  - Esta función espera hasta que algún cliente establezca una conexión con el servidor.

    *Es una llamada bloqueante, esto es, la función no finalizará hasta que se haya producido una conexión o sea interrumpida por una señal.*

# Manejo de Sockets

- Es conveniente destacar que una vez que se ha producido la conexión, la función accept() devuelve un nuevo identificador de socket que será utilizado para la comunicación con el cliente que se ha conectado.

```
#include <sys/types.h>
#include <sys/socket.h>
...
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

```
new_sockfd = accept (sockfd, &remote_addr, &addrlen);
```

# Manejo de sockets

```
...
int sockfd, new_sockfd;
struct sockaddr_in server_addr;
struct sockaddr_in remote_addr;
int addrlen;
// Creación del socket.
sockfd = socket (PF_INET, SOCK_STREAM, 0 );
// Definir valores en la estructura server_addr.
server_addr.sin_family = PF_INET;
server_addr.sin port = htons ( 1234 ); // Número de puerto donde
// recibirá paquetes el programa
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
// Asociar valores definidos al socket
bind(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr));
// Se habilita el socket para poder recibir conexiones.
listen ( sockfd, 5);
addrlen = sizeof (struct sockaddr );
// Se llama a accept() y el servidor queda en espera de conexiones.
new_sockfd = accept (sockfd, &remote_addr, &addrlen);
...
```

# Lanzar peticiones de conexión

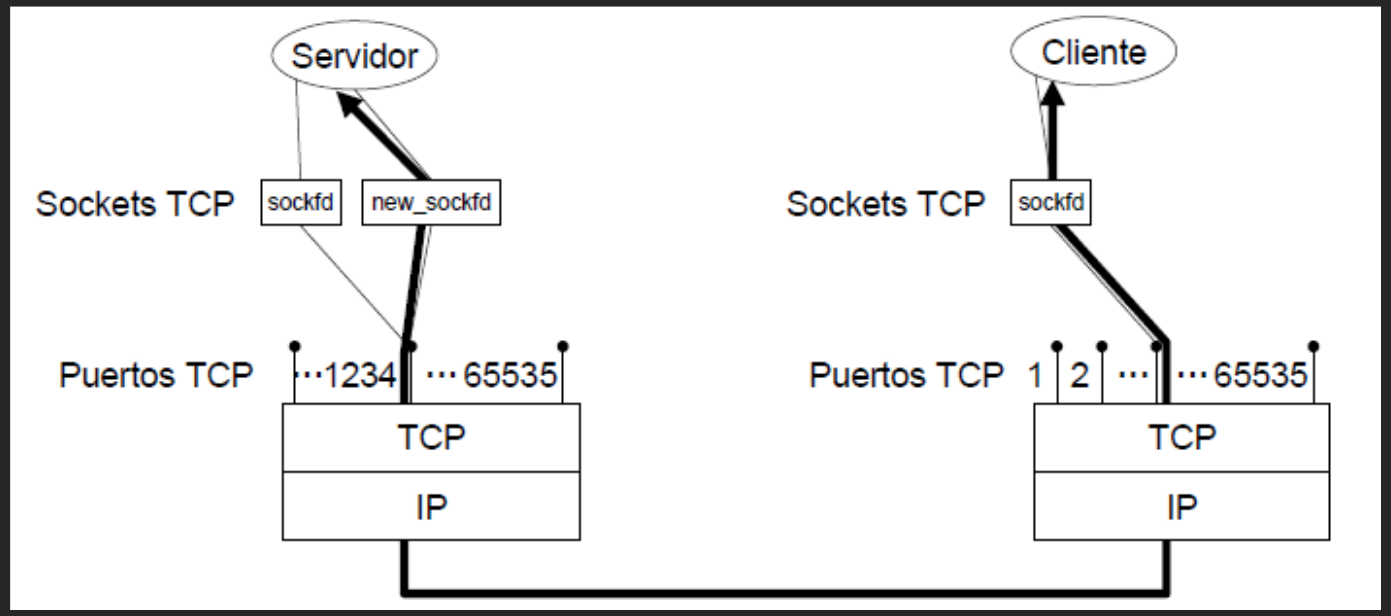
- Esta función es invocada desde el cliente para solicitar el establecimiento de una conexión TCP.

  *La función connect() inicia la conexión con el servidor remoto, por parte del cliente. El formato de la función es el siguiente:*

```
#include <sys/types.h>
#include <sys/socket.h>
...
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

# Lanzar peticiones de conexión

```
...
int sockfd;
struct sockaddr_in server_addr;
int addrlen;
// Creación del socket.
sockfd = socket (PF_INET, SOCK_STREAM, 0 );
// Definir valores en la estructura server_addr.
server_addr.sin_family = PF_INET;
server_addr.sin port = htons ( 1234 );
// Número de puerto donde
// está esperando el servidor
server_addr.sin_addr.s_addr = inet_addr("1.2.3.4");
// Dirección IP
// del servidor
addrlen = sizeof (struct sockaddr );
// Se llama a connect () y se hace la petición de conexión al servidor
connect (sockfd, &server_addr, addrlen);
...
```




# Conexión entre los sockets del cliente y servidor

- El cliente ha hecho la llamada a la función connect()
- Al concluir con éxito (lo cual significa que en el servidor se estaba esperando en la función accept() y se ha salido de ella y por tanto se ha creado el nuevo socket)

# Enviar y recibir datos a través del socket

- Una vez que la conexión ha sido establecida, se inicia el intercambio de datos, utilizando para ello las funciones send() y recv().

*La función send() devuelve el número de bytes enviados, que puede ser menor que la cantidad indicada en el parámetro len.*

```
#include <sys/types.h>
#include <sys/socket.h>
...
ssize_t send(int s, const void *buf, size_t len, int flags);
```

```
ssize_t send (int sockfd, const void *buf, size_t len, int flags );
```

*La función recv() se utiliza para recibir datos*

```
#include <sys/types.h>
#include <sys/socket.h>
...
ssize_t recv(int s, void *buf, size_t len, int flags);
```

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

# Enviar y recibir datos a través del socket

- La función recv() es bloqueante

  *No finaliza hasta que se ha recibido algún tipo de información.*

  *Resaltar que el campo len indica el número máximo de bytes a recibir*

  - No necesariamente se han de recibir exactamente ese número de bytes.

  *La función recv() devuelve el número de bytes que se han recibido.*

```
...
char mens_serv[100];
...
mens_clien = “Ejemplo”;
send(sid, mens_clien, strlen(mens_clien)+1, 0);
...
recv(sid, mens_serv, 100, 0);
...
```

# Cierre de un socket

- Simplemente hay que hacer la llamada a la función pasándole como argumento el descriptor del socket que se quiere cerrar.
- Es importante que cuando un socket no vaya a usarse más, se haga la llamada a close() para indicárselo al Sistema Operativo y que éste lo libere.

```
#include <unistd.h>
...
int close(int fd);
```