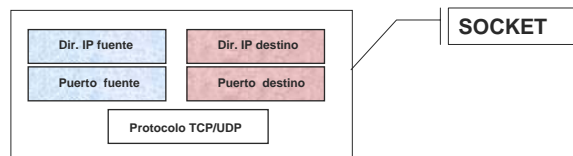


Interfaz de Sockets (2)

- › El **socket** es el mecanismo de comunicación básico que encapsula la información necesaria para establecer una comunicación entre dos procesos:



- › Presentan una interfaz igual a la de cualquier dispositivo UNIX (ficheros, consola,)

Interfaz de Sockets (3)

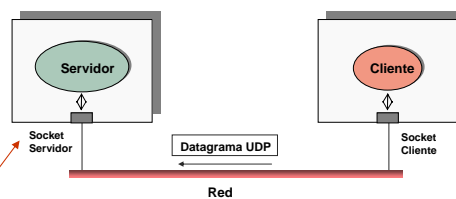
- › **Dominios (Protocol Family):**
 - › Familia de protocolos para conseguir el intercambio de datos entre sockets
 - › PF_UNIX: dominio UNIX
 - › Comunicación entre procesos en una misma máquina
 - › Utilizan manejadores de ficheros
 - › **PF_INET**: dominio INET
 - › Comunicación entre procesos que pueden estar ubicados en computadores diferentes
 - › Utiliza los protocolos TCP-UDP-IP
 - › PF_CCITT, PF_DECnet, PF_OSI, PF_X25
- › **Direcciones (Address Family):**
 - › AF_UNIX: formato UNIX
 - › Tiene el mismo formato de un fichero
 - › **AF_INET**: formato INET
 - › Dirección IP, Puerto y Protocolo (TCP/UDP)

Interfaz de Sockets (4)

- › Tipos de Comunicación:
 - **SOCK_STREAM**: comunicación orientada a la conexión (usa TCP)
 - **SOCK_DGRAM**: comunicación de tipo datagrama (usa UDP)
 - Otros tipos:
 - › **SOCK_SEQPAQUET** : comunicación secuencial similar al tipo STREAM pero garantizando que se mantienen los límites entre mensajes.
 - › **SOCK_RDM** : comunicación fiable sin conexión (similar al tipo DGRAM pero con control de errores)
 - › **SOCK_RAW** : privilegio de ROOT. Permite acceder a los protocolos de bajo nivel.

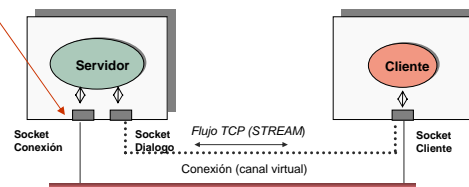
Tipos Comunicación:

› UDP



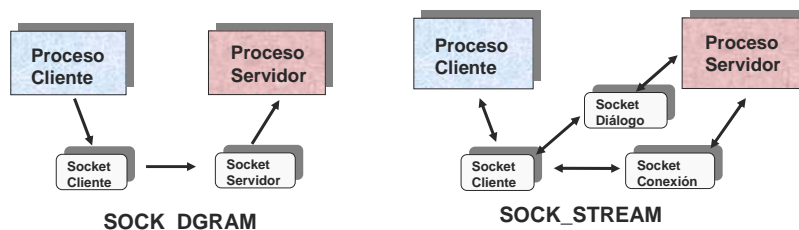
Público: IP, puerto

› TCP



Estructura de una Aplicación

- › Arquitectura Cliente-Servidor
 - **Proceso Servidor:** está continuamente escuchando el socket
 - › Un solo Thread: se atienden las peticiones secuencialmente
 - › Multithreading: se asigna un thread por cada solicitud
 - **Proceso Cliente:** toma la iniciativa en la transmisión
 - Cliente y servidor tienen sus propias direcciones IP y puertos.
 - El puerto del servidor debe ser fijo, pero el puerto del cliente se asigna dinámicamente



SITR: Aplicaciones Distribuidas. SOCKETS

9

Estructuras de Datos (1)

- › Estructuras de direcciones AF_INET

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

```
/* Estructura genérica de direcciones, válida para cualquier dominio
   (UNIX o INET). Las direcciones usadas deben ser forzadas a este tipo
   cuando se pasan como parámetro de una función. */
struct sockaddr {
    sa_family_t sa_family; /* address family */
    char        sa_data[14]; /* up to 14 bytes of direct address */
};
```

```
/* Estructura de direcciones INET. */
struct sockaddr_in {
    sa_family_t  sin_family; /* AF_INET */
    in_port_t    sin_port; /* Puerto */
    struct in_addr sin_addr; /* dirección IP */
    char        sin_zero[8];
};
```

SITR: Aplicaciones Distribuidas. SOCKETS

10

Estructuras de Datos (2)

- > Estructura para la dirección IP : `struct in_addr`
`#include <netinet/in.h>`
- > Varias opciones de acceso:

Entero 32 bits

```
struct in_addr {  
    uint32_t s_addr;  
};
```

2 Palabras 16 bits

```
struct in_addr {  
    struct { ushort_t s_w1, s_w2; } _s_un_w;  
};
```

4 bytes

```
struct in_addr {  
    struct { uchar_t s_b1, s_b2, s_b3, s_b4; } _s_un_b;  
};
```

SITR: Aplicaciones Distribuidas. SOCKETS

11

Funciones Auxiliares (1)

- > Funciones para el manejo de direcciones INET
 - Cada máquina tiene una forma de representar la información numérica por lo que se utilizan funciones de conversión a un formato estándar ("*Little Endian*")
`#include <arpa/inet.h>`

```
// convierte una cadena en formato W.X.Y.Z en una dirección IP de red  
in_addr_t inet_addr (const char *);  
  
// convierte una dirección IP de red en una cadena en formato W.X.Y.Z  
char *inet_ntoa (struct in_addr);  
  
// convierten puertos y direcciones de hosts a red y viceversa  
  
in_addr_t htonl (in_addr_t hostlong);  
in_port_t htons (in_port_t hostshort);  
in_addr_t ntohl (in_addr_t netlong);  
in_port_t ntohs (in_port_t netshort);
```

SITR: Aplicaciones Distribuidas. SOCKETS

12

Funciones Auxiliares (2)

- › Conversión de nombres simbólicos a direcciones IP

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

```
struct hostent {
    char    *h_name; /* official name of host */
    char    **h_aliases; /* alias list */
    int     h_addrtype; /* host address type */
    int     h_length; /* length of address */
    char    **h_addr_list; /* list of addresses from name server */
#define h_addr h_addr_list[0] /* address, for backward compatibility
*/};
```

```
// obtiene la estructura de direcciones a partir del nombre simbólico DNS
struct hostent *gethostbyname (const char *);

// Versión reentrante (Multithreading)
struct hostent *gethostbyname_r (const char *, struct hostent *,
                                char *, int, int *h_errnop);

// Obtiene la información de dirección/puerto de un socket local/remoto
int getsockname (int socket, struct sockaddr * addr, size_t *length_addr);
int getpeername (int socket, struct sockaddr * addr, size_t *length_addr);
```

Funciones Auxiliares (3)

- › Valores de puertos y protocolo para los servicios estándar (/etc/services)

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

```
struct servent {
    char    *s_name; /* official service name */
    char    **s_aliases; /* alias list */
    int     s_port; /* port # */
    char    *s_proto; /* protocol to use */
};
```

```
/* obtiene los datos de puerto y protocolo a partir de una cadena con el
nombre del servicio */
struct servent *getservbyname (const char *name, const char *proto);
```

Funciones Manejo de sockets (1)

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

```
// Crea un manejador para el socket
int socket (int domain, int type, int protocol);
    domain: PF_INET
    type: SOCK_STREAM, SOCK_DGRAM
    protocol: 0 (Protocolo por omisión)
    Devuelve el manejador (<0 si es un error)

int close (int s); /* elimina el manejador del socket */
```

```
// Nombra el socket (Asigna dirección y puerto )
// Nota: El socket cliente se nombra automáticamente
int bind (int s, const struct sockaddr * name, size_t namelen);
    s: identificador del socket
    name: puntero a la estructura dirección/puerto
        (se debe hacer un cast desde la estructura sockaddr_in
    namelen : tamaño de la estructura
    devuelve 0 si es correcto (-1 si se produce un error)
```

SITR: Aplicaciones Distribuidas. SOCKETS

15

Funciones Manejo de sockets (2)

› Sockets orientados a la conexión (TCP):

```
Servidor:
// Especifica el tamaño de la caché de solicitudes (Activa la
recepción de conexiones)
int listen (int s, int backlog);

// pone al servidor en espera de una conexión
int accept (int s, struct sockaddr *addr, size_t *addrlen);
    s: identificador del socket de escucha
    addr: almacena los datos del cliente
    addrlen : almacena el tamaño de la dirección del cliente
Si se establece una conexión devuelve un socket de diálogo
si es incorrecto devuelve -1
```

```
Cliente:
// establece una conexión
int connect(int s, const struct sockaddr *name, size_t namelen);
```

SITR: Aplicaciones Distribuidas. SOCKETS

16

Funciones Manejo de sockets (3)

> Lectura y escritura en un socket

```
ssize_t sendto (int socket, const void *message,
                size_t length, int flags,
                const struct sockaddr *dest_addr, size_t dest_len);

//solo en modo STREAM
ssize_t send(int socket, const void *buffer,
             size_t length, int flags);
ssize_t write(int fd, const void *buf, size_t nbytes);

ssize_t recvfrom(int socket, void *message,
                size_t length, int flags,
                const struct sockaddr *dest_addr, size_t *dest_len);

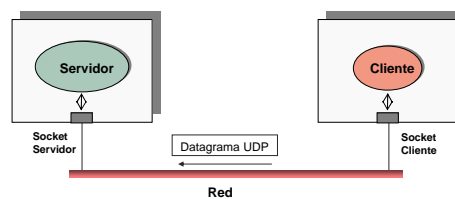
// solo en modo STREAM
ssize_t recv(int socket, void *buffer, size_t length, int flags);
ssize_t read(int fd, void *buf, size_t nbytes);
```

SITR: Aplicaciones Distribuidas. SOCKETS

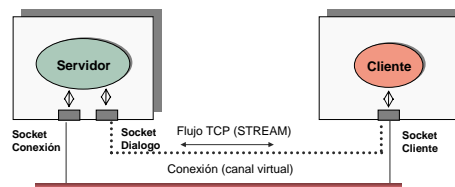
17

Ejemplo: Tipos Comunicación

> UDP



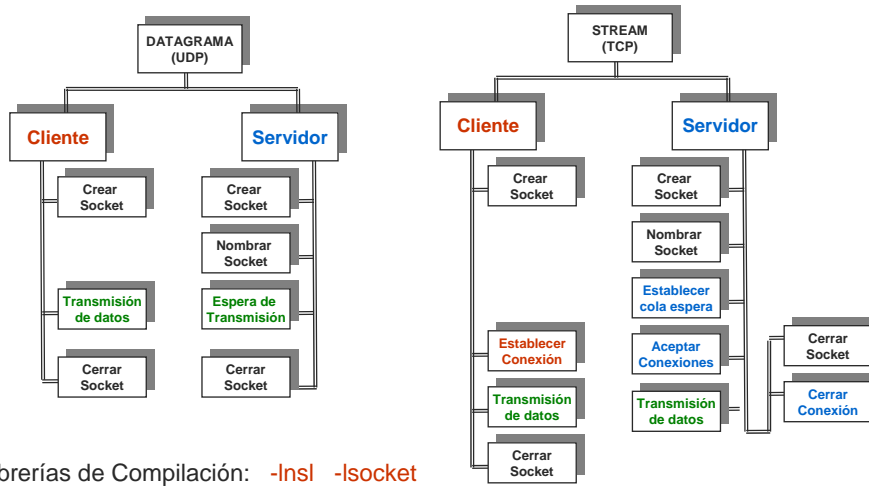
> TCP



SITR: Aplicaciones Distribuidas. SOCKETS

18

Etapas aplicación UDP/TCP



Librerías de Compilación: **-lnsl -lsocket**

SITR: Aplicaciones Distribuidas. SOCKETS

19

Comunicación UDP

```

lorca.umh.es - PuTTY
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]udpsrvl
---Servidor IP: 0.0.0.0, Puerto: 60538---

Esperando datos....
10 bytes Recibidos desde [193.147.145.36:60539]: Mensaje 1

Esperando datos....
10 bytes Recibidos desde [193.147.145.36:60540]: Mensaje 2
[u0/sitr/sitr001/sockets]

lorca.umh.es - PuTTY
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]
[u0/sitr/sitr001/sockets]udpc1t1 lorca.umh.es 60538 "Mensaje 1"
10 bytes Enviados a [193.147.145.36:60538]: Mensaje 1
---Cliente -> IP: 0.0.0.0, Puerto: 60539---

[u0/sitr/sitr001/sockets]udpc1t1 lorca.umh.es 60538 "Mensaje 2"
10 bytes Enviados a [193.147.145.36:60538]: Mensaje 2
---Cliente -> IP: 0.0.0.0, Puerto: 60540---

[u0/sitr/sitr001/sockets]
    
```

22

Comunicación UDP Bidireccional

Ciente UDP

```
// Espera datos por el socket
longdir = sizeof datosSocketSrv;
res = recvfrom(socketID, mensaje, DIM, 0,
              (struct sockaddr *) &datosSocketSrv, &longdir);
if(res>0)
{ // muestra lo que ha leído
  printf("%d bytes Recibidos desde [%s:%u]: %s\n", res,
        inet_ntoa(datosSocketSrv.sin_addr),
        ntohs(datosSocketSrv.sin_port), mensaje);
}
```

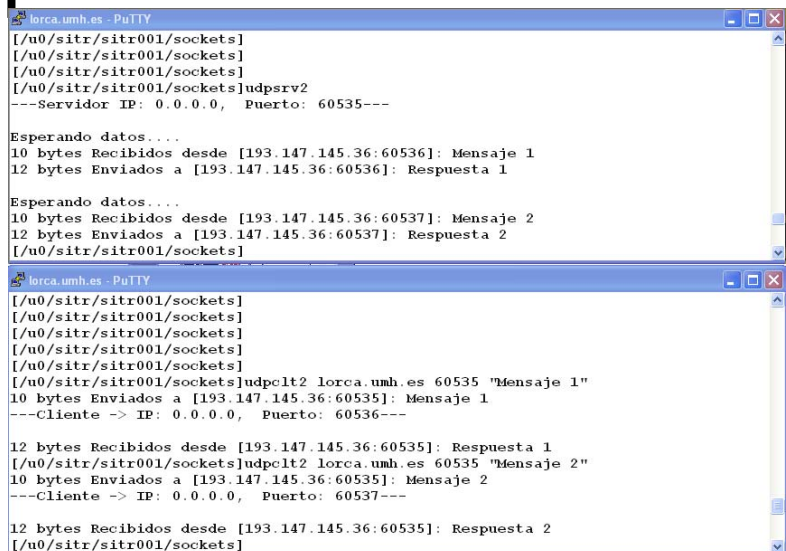
Servidor UDP

```
sprintf(mensaje, "Respuesta");
res = sendto(socketID, mensaje, strlen(mensaje)+1, 0,
            (struct sockaddr *)&datosSocketCl, sizeof datosSocketCl);
```

SITR: Aplicaciones Distribuidas. SOCKETS

23

Comunicación UDP (Bidireccional)



```
lorca.umh.es - PuTTY
[/u0/sitr/sitr001/sockets]
[/u0/sitr/sitr001/sockets]
[/u0/sitr/sitr001/sockets]
[/u0/sitr/sitr001/sockets]udpsrv2
---Servidor IP: 0.0.0.0, Puerto: 60535---

Esperando datos...
10 bytes Recibidos desde [193.147.145.36:60536]: Mensaje 1
12 bytes Enviados a [193.147.145.36:60536]: Respuesta 1

Esperando datos...
10 bytes Recibidos desde [193.147.145.36:60537]: Mensaje 2
12 bytes Enviados a [193.147.145.36:60537]: Respuesta 2
[/u0/sitr/sitr001/sockets]

lorca.umh.es - PuTTY
[/u0/sitr/sitr001/sockets]
[/u0/sitr/sitr001/sockets]
[/u0/sitr/sitr001/sockets]
[/u0/sitr/sitr001/sockets]
[/u0/sitr/sitr001/sockets]
[/u0/sitr/sitr001/sockets]udpc1t2 lorca.umh.es 60535 "Mensaje 1"
10 bytes Enviados a [193.147.145.36:60535]: Mensaje 1
---Cliente -> IP: 0.0.0.0, Puerto: 60536---

12 bytes Recibidos desde [193.147.145.36:60535]: Respuesta 1
[/u0/sitr/sitr001/sockets]udpc1t2 lorca.umh.es 60535 "Mensaje 2"
10 bytes Enviados a [193.147.145.36:60535]: Mensaje 2
---Cliente -> IP: 0.0.0.0, Puerto: 60537---

12 bytes Recibidos desde [193.147.145.36:60535]: Respuesta 2
[/u0/sitr/sitr001/sockets]
```

24

Comunicación TCP

```
lorca.umh.es - PuTTY
[/n0/sitr/sitr001/sockets]
[/n0/sitr/sitr001/sockets]
[/n0/sitr/sitr001/sockets]
[/n0/sitr/sitr001/sockets]
---Servidor IP: 0.0.0.0, Puerto: 41428---

Esperando Conexión (1) ....
[Conexión 1, Recibido (10 bytes) desde 193.147.145.36:41429 <-] Mensaje 1
[Conexión 1 Terminada]

Esperando Conexión (2) ....
[Conexión 2, Recibido (10 bytes) desde 193.147.145.36:41430 <-] Mensaje 2
[Conexión 2 Terminada]

Esperando Conexión (3) ....

lorca.umh.es - PuTTY
[/n0/sitr/sitr001/sockets]
[/n0/sitr/sitr001/sockets]
[/n0/sitr/sitr001/sockets]
[/n0/sitr/sitr001/sockets]
[/n0/sitr/sitr001/sockets]
[/n0/sitr/sitr001/sockets]
[/n0/sitr/sitr001/sockets]
---Cliente -> IP: 193.147.145.36, Puerto: 41429---

[Conexión Terminada]
[/n0/sitr/sitr001/sockets]tcpcli1 lorca.umh.es 41428 "Mensaje 1"
10 bytes Enviados a [193.147.145.36:41428]: Mensaje 1
---Cliente -> IP: 193.147.145.36, Puerto: 41429---

[Conexión Terminada]
[/n0/sitr/sitr001/sockets]tcpcli1 lorca.umh.es 41428 "Mensaje 2"
10 bytes Enviados a [193.147.145.36:41428]: Mensaje 2
---Cliente -> IP: 193.147.145.36, Puerto: 41430---

[Conexión Terminada]
[/n0/sitr/sitr001/sockets]
```

27

Comunicación TCP Bidireccional

Cliente TCP

```
// Espera datos por el socket
res = recv(socketID, mensaje, DIM, 0);
if(res>0)
{ // muestra lo que ha leído
    printf("%d bytes Recibidos desde [%s:%u]: %s\n", res,
        inet_ntoa(datosSocketSrv.sin_addr),
        ntohs(datosSocketSrv.sin_port), mensaje);
}
```

Servidor TCP

```
sprintf(mensaje, "Respuesta %d");
res = send(socketDialogo, mensaje, strlen(mensaje)+1, 0);
```

28

Comunicación TCP Bidireccional

```

lorca.umh.es - PuTTY
[~/u0/sitr/sitr001/sockets]tcpsrv2
---Servidor IP: 0.0.0.0, Puerto: 41431---

Esperando Conexión (1) ....
[Conexion 1, Recibido (10 bytes) desde 193.147.145.36:41432 <-] Mensaje 1
12 bytes Enviados a [193.147.145.36:41432]: Respuesta 1
[Conexion 1 Terminada]

Esperando Conexión (2) ....
[Conexion 2, Recibido (10 bytes) desde 193.147.145.36:41433 <-] Mensaje 2
12 bytes Enviados a [193.147.145.36:41433]: Respuesta 2
[Conexion 2 Terminada]

Esperando Conexión (3) ....

lorca.umh.es - PuTTY
[~/u0/sitr/sitr001/sockets]
[~/u0/sitr/sitr001/sockets]
[~/u0/sitr/sitr001/sockets]
[~/u0/sitr/sitr001/sockets]
[~/u0/sitr/sitr001/sockets]tcpclt2 lorca.umh.es 41431 "Mensaje 1"
10 bytes Enviados a [193.147.145.36:41431]: Mensaje 1
---Cliente -> IP: 193.147.145.36, Puerto: 41432---

12 bytes Recibidos desde [193.147.145.36:41431]: Respuesta 1
[Conexion Terminada]
[~/u0/sitr/sitr001/sockets]tcpclt2 lorca.umh.es 41431 "Mensaje 2"
10 bytes Enviados a [193.147.145.36:41431]: Mensaje 2
---Cliente -> IP: 193.147.145.36, Puerto: 41433---

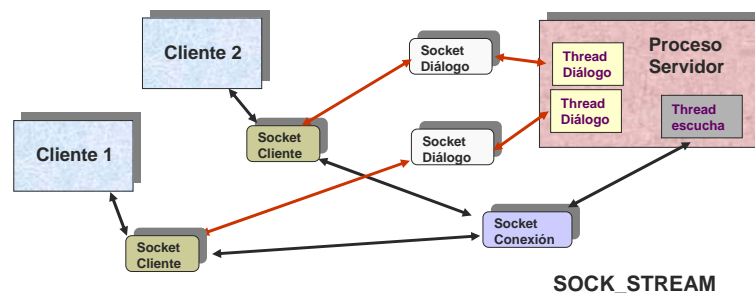
12 bytes Recibidos desde [193.147.145.36:41431]: Respuesta 2
[Conexion Terminada]
[~/u0/sitr/sitr001/sockets]

```

29

Servidor TCP Concurrente

- Proceso Servidor:**
 - Un **Thread (conexión)** está continuamente escuchando el socket
 - Con cada conexión aceptada se crea un **Thread (diálogo)** que se comunica con el cliente
 - El socket de dialogo se pasa como parámetro al thread



SITR: Aplicaciones Distribuidas. SOCKETS

30