

Sistemas Operativos Distribuidos

Sistemas Distribuidos

Sockets

Sockets

Aparecieron en 1981 en UNIX BSD 4.2

- Intento de incluir TCP/IP en UNIX.
- Diseño independiente del protocolo de comunicación.

Un socket es punto final de comunicación (dir. IP y puerto).

Abstracción que:

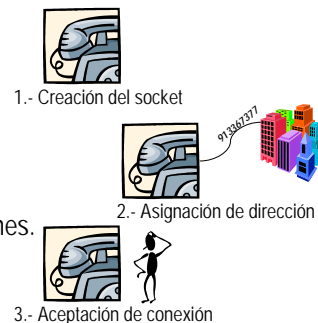
- Ofrece interfaz de acceso a servicios de red en nivel de transporte
- Representa extremo de comunicación bidireccional.

Actualmente:

- Disponibles en casi todos UNIX y prácticamente todos los SSOO
 - WinSock: API de sockets de Windows.
- En Java como clase nativa.

Conceptos básicos sobre sockets

- Dominios de comunicación.
- Tipos de *sockets*.
- Creación de un *socket*.
- Direcciones de *sockets*.
- Asignación de direcciones.
- Solicitud de conexión.
- Preparar para aceptar conexiones.
- Aceptar una conexión.
- Transferencia de datos.



Dominios de comunicación

- Un dominio representa una familia de protocolos.
- Un *socket* está asociado a un dominio desde su creación.
- Sólo se pueden comunicar *sockets* del mismo dominio.
- Los servicios de *sockets* son independientes del dominio.

Algunos ejemplos:

- **PF_UNIX** (o **PF_LOCAL**): comunicación dentro de una máquina.
- **PF_INET**: comunicación usando protocolos TCP/IP.

Tipos de sockets

- **Stream (SOCK_STREAM):**
 - Orientado a conexión.
 - Fiable, se asegura el orden de entrega de mensajes.
 - No mantiene separación entre mensajes (*stream*).
 - Si **PF_INET** se corresponde con el protocolo TCP.
- **Datagrama (SOCK_DGRAM):**
 - Sin conexión.
 - No fiable, no se asegura el orden en la entrega.
 - Mantiene la separación entre mensajes.
 - Si **PF_INET** se corresponde con el protocolo UDP.
- **Raw (SOCK_RAW):**
 - Permite el acceso a los protocolos internos como IP.

Creación de un socket

La función **socket** crea uno nuevo:

- ```
int socket(int dom,int tipo,int proto)
```
- Devuelve un descriptor de fichero (igual que un **open** de fichero).
  - Dominio (**dom**): **PF\_XXX**
  - Tipo de socket (**tipo**): **SOCK\_XXX**
  - Protocolo (**proto**): Dependiente del dominio y del tipo:
    - 0 elige el más adecuado.
    - Especificados en **/etc/protocols**.

El socket creado **no** tiene dirección asignada.

## Direcciones de sockets

- Cada *socket* debe tener asignada una dirección única.
- Dependientes del dominio.
- Las direcciones se usan al:
  - Asignar una dirección local a un socket (**bind**).
  - Especificar una dirección remota (**connect** o **sendto**).
- Se utiliza la estructura genérica de dirección:
  - **struct sockaddr mi\_dir;**
- Cada dominio usa una estructura específica.
  - Uso de *cast* en las llamadas.
  - Direcciones en **PF\_INET** (**struct sockaddr\_in**).
  - Direcciones en **PF\_UNIX** (**struct sockaddr\_un**).

## Direcciones de sockets en PF\_INET

Una dirección destino viene determinada por:

- Dirección del *host*: 32 bits.
- Puerto de servicio: 16 bits (Reservados: 0..1023)
  - Espacio de puertos TCP y UDP independientes

Estructura **struct sockaddr\_in**:

- Debe iniciarse a 0 (**bzero**).
- **sin\_family**: dominio (AF\_INET).
- **sin\_port**: puerto.
- **sin\_addr**: dirección del *host*.

Una transmisión está caracterizada por cinco parámetros únicos:

- Dirección *host* y puerto origen.
- Dirección *host* y puerto destino.
- Protocolo de transporte (UDP o TCP).

## Obtención de la dirección del host

Usuarios manejan direcciones en forma de texto:

- decimal-punto: 138.100.8.100
- dominio-punto: laurel.datsi.fi.upm.es

- Conversión a binario desde decimal-punto:

```
int inet_aton(char *str, struct in_addr *dir)
```

- **str**: contiene la cadena a convertir.
- **dir**: resultado de la conversión en formato de red.

- Conversión a binario desde dominio-punto:

```
struct hostent *gethostbyname(char *str)
```

- **str**: cadena a convertir.
- Devuelve la estructura que describe al *host*.

## Asignación de direcciones

La asignación de una dirección a un *socket* ya creado:

```
int bind(int s, struct sockaddr* dir, int tam)
```

- Socket (**s**): Ya debe estar creado.
- Dirección a asignar (**dir**): Estructura dependiendo del dominio.
- Tamaño de la dirección (**tam**): `sizeof()`.

Direcciones en dominio **PF\_INET**

- Si se le indica puerto 0, el sistema elige uno (puerto efímero).
- Host: una dirección IP de la máquina local.
  - **INADDR\_ANY**: elige cualquiera de la máquina.

Si no se asigna dirección (típico en clientes)

- automáticamente en primer uso (**connect** o **sendto**).

## Solicitud de conexión

Realizada en el cliente por medio de la función:

```
int connect(int s, struct sockaddr* d, int tam)
```

- Socket creado (**s**).
- Dirección del servidor (**d**).
- Tamaño de la dirección (**tam**).

Un socket *stream* sólo permite un único *connect* durante su vida

- Para conectarse con el mismo u otro hay que crear un nuevo socket

Normalmente se usa con *streams* pero también con datagramas

- Más adelante se analiza uso con datagrama

## Preparar para aceptar conexiones

- Realizada en servidor *stream* después de **socket** y **bind**

```
int listen(int sd, int backlog)
```

- Socket (**sd**): Descriptor de uso del socket.
- Tamaño del buffer (**backlog**):
  - N<sup>o</sup> máximo de peticiones pendientes de aceptar que se encolarán

- Hace que el socket quede preparado para aceptar conexiones
- Con un socket se pueden aceptar n<sup>o</sup> ilimitado de conexiones

## Aceptar una conexión

Realizada en el servidor *stream* después de **listen**:

```
int accept(int s, struct sockaddr *d, int *tam)
```

- Socket (**sd**): Descriptor de uso del socket.
- Dirección del cliente (**d**): Dirección del socket del cliente devuelta.
- Tamaño de la dirección (**tam**): Parámetro valor-resultado
  - Antes de la llamada: tamaño de dir
  - Después de la llamada: tamaño de dirección del cliente devuelta

## Aceptar una conexión

- Cuando se produce la conexión, el servidor obtiene:
  - La dirección del socket del cliente.
  - Un nuevo descriptor (socket) conectado al socket del cliente.
- Después de conexión quedan activos 2 sockets en el servidor:
  - El original para aceptar nuevas conexiones
  - El nuevo para enviar/recibir datos por la conexión establecida.
- Facilita construcción de servidores concurrentes
  - En servidores *multithread*, cuidado con condición de carrera en:

```
while (true) {
 n=accept(s, ...);
 pthread_create(..., &n);
}
```
  - Solución: pasar **n** por valor o usar memoria dinámica
    - flujo principal realiza *malloc* y *thread* el *free*

## Múltiples clientes con *streams*

- Con servidor iterativo no concurrente
  - Si 1 conexión por petición
    - Se intercalan peticiones de los clientes (1 por iteración)
  - Si varias peticiones de cliente usan misma conexión
    - No se trata a otro cliente hasta que no termine el actual o bien...
    - *select* espera simultánea de más peticiones de conexión o datos
- Con servidor concurrente
  - Si 1 conexión por petición
    - Cada *thr./proc.* sirve una petición y termina su labor
  - Si varias peticiones de cliente usan misma conexión
    - Cada *thr./proc.* sirve peticiones hasta que cliente cierra el socket
  - En ambos casos también puede usarse conjunto de *thr/proc* o híbrido

## Otras funcionalidades

Obtener la dirección a partir de un descriptor:

- Dirección local: **getsockname()**.
- Dirección del socket en el otro extremo: **getpeername()**.

Transformación de valores:

- De formato *host* a red:
  - Enteros largos: **htonl()**.
  - Enteros cortos: **htons()**.
- De formato de red a *host*:
  - Enteros largos: **ntohl()**.
  - Enteros cortos: **ntohs()**.

Cerrar la conexión:

- Para cerrar ambos tipos de sockets: **close()**.
  - Si el socket es de tipo *stream* cierra la conexión en ambos sentidos.
- Para cerrar un único extremo: **shutdown()**.

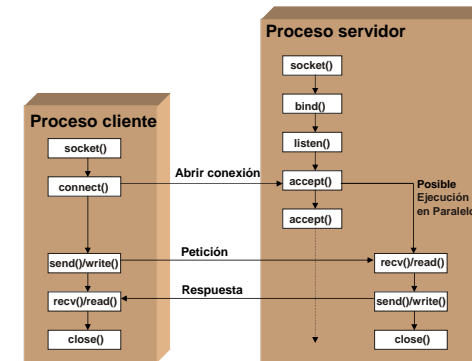
# Sistemas Operativos Distribuidos

## Transferencia de datos con *streams*

- Modo de operación asíncrono
- Envío:  
`int send(int s, char *mem, int tam, int flags)`
  - Devuelve el nº de bytes enviados.
  - Puede usarse `write` (o `writen`) sobre el descriptor de socket.
- Recepción:  
`int recv(int s, char *mem, int tam, int flags)`
  - Devuelve el nº de bytes recibidos (0 si cliente ha cerrado socket)
  - Puede usarse `read` (o `readv`) sobre el descriptor de socket.
  - Lectura puede devolver menos bytes de los pedidos
    - Si se requiere leer *N* bytes hay que usar un bucle
- No requiere correspondencia entre nº de `send` y de `recv`
- Los *flags* implican aspectos avanzados
  - como enviar o recibir datos urgentes (*out-of-band*).

Sistemas Operativos Distribuidos 17 Fernando Pérez Costoya — José M<sup>o</sup> Peña Sánchez  
M<sup>o</sup> de los Santos Pérez Hernández

## Escenario de uso de sockets *streams*



Sistemas Operativos Distribuidos 18 Fernando Pérez Costoya — José M<sup>o</sup> Peña Sánchez  
M<sup>o</sup> de los Santos Pérez Hernández

## Transferencia de datos con datagramas

- Envío:
- ```
int sendto(int s, char *mem, int tam,
           int flags, struct sockaddr *dir, int *tam)
```
- Recepción:
- ```
int recvfrom(int s, char *mem, int tam,
 int flags, struct sockaddr *dir, int *tam)
```
- `recv` (`read`) si no necesita conocer dir. de envío (cliente)
  - No se establece una conexión (`connect/accept`) previa.
  - Para usar socket basta con crear socket y reservar dirección
    - En el cliente no sería necesario `bind`

Sistemas Operativos Distribuidos 19 Fernando Pérez Costoya — José M<sup>o</sup> Peña Sánchez  
M<sup>o</sup> de los Santos Pérez Hernández

## Más sobre datagramas

- Mismo socket puede usarse para enviar a diferentes sockets
- Por un socket puede llegar información de distintos clientes
  - En *stream* por socket conectado llega información de un solo cliente
- Se mantiene separación entre mensajes:
  - Una lectura consume un mensaje
  - Si el tamaño leído es menor que mensaje, el resto se pierde
  - Correspondencia entre nº de `sendto` y de `recv/recvfrom`
- Se permite `connect` en socket datagrama:
  - No realiza conexión física: sólo especifica destino para `send/write`
  - No afecta al servidor pero...
  - permite que cliente pueda ser idéntico usando *stream* o datagrama

Sistemas Operativos Distribuidos 20 Fernando Pérez Costoya — José M<sup>o</sup> Peña Sánchez  
M<sup>o</sup> de los Santos Pérez Hernández

# Sistemas Operativos Distribuidos

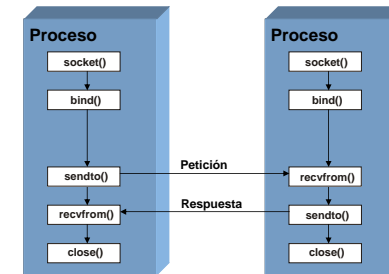
## Múltiples clientes con datagramas

- Con servidor iterativo no concurrente
  - Las peticiones de clientes ya llegan intercaladas
- Con servidor concurrente
  - Después de recibir mensaje, se crea thr/proc que lo trata y responde
  - Puede usarse modelo dinámico, de bolsa de thr/proc o híbrido

Sistemas Operativos Distribuidos  
21

Fernando Pérez Costoya — José M<sup>a</sup> Peña Sánchez  
M<sup>a</sup> de los Santos Pérez Hernández

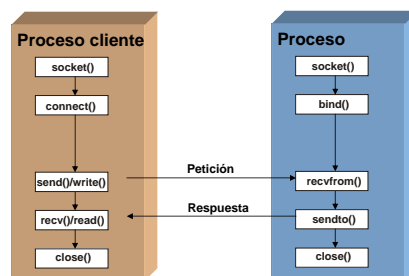
## Escenario de uso de sockets datagrama



Sistemas Operativos Distribuidos  
22

Fernando Pérez Costoya — José M<sup>a</sup> Peña Sánchez  
M<sup>a</sup> de los Santos Pérez Hernández

## Uso de datagramas con conexión



Sistemas Operativos Distribuidos  
23

Fernando Pérez Costoya — José M<sup>a</sup> Peña Sánchez  
M<sup>a</sup> de los Santos Pérez Hernández

## Datagramas vs streams

- Uso mayoritario de *streams*
- Datagramas si no tolera sobrecarga y admiten pérdida de info.
  - p.ej. transmisión de voz
- En cliente/servidor más conveniente *stream* excepto si:
  - Los mensajes de petición y respuesta son pequeños
    - No se puede tolerar la sobrecarga de la conexión
    - Además, si son grandes hay que fragmentar y compactar
  - Las operaciones son idempotentes
    - Evita sobrecarga de gestionar caché de respuestas en servidor
  - Se quiere dar servicio a un n<sup>o</sup> muy elevado de clientes
    - Datagramas no requieren tanta información en S.O. como *streams*

Sistemas Operativos Distribuidos  
24

Fernando Pérez Costoya — José M<sup>a</sup> Peña Sánchez  
M<sup>a</sup> de los Santos Pérez Hernández

## Configuración de opciones

Consultar opciones asociadas a un socket: **getsockopt ( )**

Modificar opciones asociadas a un socket: **setsockopt ( )**

Varios niveles dependiendo de protocolo afectado:

- **SOL\_SOCKET**: opciones independientes del protocolo.
- **IPPROTO\_TCP**: nivel de protocolo TCP.
- **IPPROTO\_IP**: nivel de protocolo IP.

Ejemplos

- Nivel **SOL\_SOCKET**. Para reutilizar direcciones: **SO\_REUSEADDR**
- Nivel **IPPROTO\_IP**. Para usar multicast IP:  
**IP\_MULTICAST\_TTL**, **IP\_MULTICAST\_IF**,  
**IP\_MULTICAST\_LOOP**, **IP\_ADD\_MEMBERSHIP**,  
**IP\_DROP\_MEMBERSHIP**

## Multidifusión IP

- Implementación de comunicación de grupo sobre IP
- Emisor envía datagrama a dirección IP de multidifusión
  - Empieza por 1110. De 239.0.0.0 a 239.255.255.255 para temporales.
- Emisor puede formar parte del grupo o no
- Procesos se incorporan y abandonan el grupo
- Control del ámbito de propagación mediante *time-to-live*:
  - el host (0), la subred (1), el *site* (32), ...
- No atomicidad.
- No garantiza ningún orden de entrega