Grado en Ingeniería de Computadores

Programación con sockets

Profesor:

Dr. Juan Carlos Fabero Jiménez (UCM)

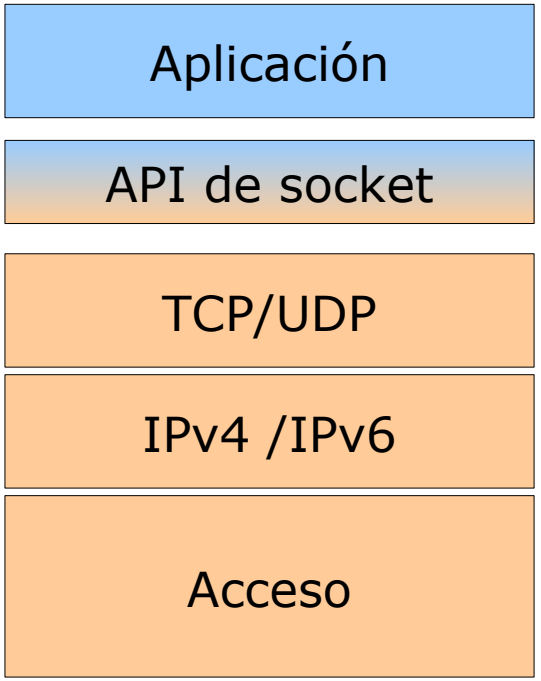
jcfabero@ucm.es

Contenidos

- **Tema 1: Ampliación de TCP.**
- **Tema 2: IP de nueva generación: IPv6.**
- **Tema 3: Configuración dinámica de la red: DHCP.**
- **Tema 4: El sistema de nombres de dominio DNS.**
- **Tema 4: Encaminamiento interno. RIP y OSPF.**
- **Tema 5: Encaminamiento externo. BGP.**
- **Tema 5: Cortafuegos y NAT.**
- **Tema 6: Criptografía. Infraestructura de clave pública.**
- **Tema 7: Introducción a la programación con *sockets.***

Introducción

- **Socket**
  - Una comunicación bidireccional queda especificada por sus extremos.
  - Cada extremo se denomina socket.
  - Un socket está caracterizado por:
    - Dirección IP
    - Número de puerto
    - Protocolo
  - API (Application Programming Interface)
    - BSD (Berkeley sockets)
    - Windows (winsock)
    - Otros idiomas (Java, Python...)



Tipos de sockets

- **DGRAM**
  - Servicio no fiable y no orientado a conexión.
  - Operaciones: socket, bind, [connect], recvfrom, sendto...
- **STREAM**
  - Servicio fiable orientado a conexión.
  - Similar a una tubería (pipe).
  - Operaciones: socket, bind, listen, accept, connect, recv, send...
- **RAW**
  - Permite acceder a protocolos de capas inferiores que no están implementados en la API.
  - Generalmente, servicio no fiable y sin conexión.

Tipos de sockets

- **Dominios y Protocolos**
  - Cada tipo de socket se puede utilizar dentro de un “dominio” (abstracto), asociado con un modo de direccionamiento:
    - AF_UNIX: comunicación entre procesos locales.
    - **AF_INET, AF_INET6**: direccionamiento IPv4 e IPv6.
    - AF_IPX, AF_X25, ...: otros protocolos.
    - AF_PACKET: para sockets de tipo RAW.
  - Cada tipo de socket puede utilizar los protocolos asociados:
    - TCP: para sockets de tipo SOCK_STREAM.
    - UCP: para sockets de tipo SOCK_DGRAM.

Estructuras de datos

- **sockaddr**
  - Se emplea en muchas de las llamadas al sistema: bind, connect, sendto...
  - sa_family: la familia de direcciones del socket (AF_INET, AF_INET6, AF_IPX...)
  - sa_data: espacio variable para la información de la dirección.

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

struct sockaddr {
    sa_family_t sa_family;
    char        sa_data[14]; }
```

- **sockaddr_storage**
  - Para definir variables genéricas que puedan contener cualquier sockaddr, se utiliza esta estructura.

```
struct addrinfo hints;
struct addrinfo *result, *rp;
struct sockaddr_storage peer_addr;
...
s = getaddrinfo(NULL, argv[1], &hints, &result);
...
peer_addr_len = sizeof(struct sockaddr_storage);
nread = recvfrom(sfd, buf, BUF_SIZE, 0,  (struct sockaddr *) &peer_addr, &peer_addr_len);
```

Estructuras de datos

- **addrinfo**
  - Contiene información sobre la dirección asociada con un socket, o criterios de búsqueda para getaddrinfo(3)
  - ai_flags: AI_NUMERICHOST, AI_PASSIVE, IN[6]ADDR_ANY, AI_NUMERICSERV
  - ai_family: AF_INET, AF_INET6, AF_UNSPEC
  - ai_socktype: SOCK_DGRAM, SOCK_STREAM
  - ai_protocolo: IPPROTO_UDP, IPPROTO_TCP; normalmente 0.
  - ai_addrlen: longitud, en bytes, de la dirección del socket.
  - ai_addr: dirección del socket (dirección IP + puerto)
  - ai_canonname: nombre oficial del host.
  - ai_next: puntero para crear una lista enlazada.

```
struct addrinfo {
    int              ai_flags;
    int              ai_family;
    int              ai_socktype;
    int              ai_protocol;
    socklen_t        ai_addrlen;
    struct sockaddr *ai_addr;
    char            *ai_canonname;
    struct addrinfo *ai_next; };
```

Estructuras de datos

- **sockaddr**
  - Es una estructura genérica para albergar la dirección del socket: IPv4/IPv6 + Puerto

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

- **Estructuras específicas para IPv4 o IPv6**
  - Normalmente no hará falta acceder a ellas explícitamente

```
struct sockaddr_in {
 sa_family_t sin_family; /* AF_INET */
 in_port_t sin_port;
 struct in_addr sin_addr;
 char sin_zero[8];
};

struct in_addr {
 uint32_t s_addr;
};
```

```
struct sockaddr_in6 {
        sa_family_t  sin6_family;   /* AF_INET6 */
        in_port_t  sin6_port;     /* port number */
        uint32_t  sin6_flowinfo; /* IPv6 flow label */
        struct in6_addr  sin6_addr;     /* IPv6 addr. */
        uint32_t  sin6_scope_id; /* Scope ID */
      };

      struct in6_addr {
        unsigned char  s6_addr[16];   /* IPv6 addr.*/
      };
```

Funciones básicas

- **socket(2)**
  - Crea un socket
    - domain: familia de dirección
      - AF_INET, AF_INET6, AF_UNIX...
    - type: tipo de socket
      - SOCK_DGRAM, SOCK_STREAM, SOCK_RAW...
    - protocol: protocolo asociado. Generalmente se fija a 0.
      - IPPROTO_UDP, IPPROTO_TCP

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

int socket(int domain, int type, int protocol);
```

```
int sd;

/* crea un socket TCP sobre IPv6 */
sd = socket(AF_INET6, SOCK_STREAM, 0);
```

Funciones básicas

- **getaddrinfo(3)**
  - Dado un host y servicio, devuelve una lista de addrinfo que se puede utilizar en bind o connect.
- **freeaddrinfo(3)**
  - Libera la estructura.

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

int getaddrinfo(const char *node, const char
*service, const struct addrinfo *hints,
                struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

const char *gai_strerror(int errcode);
```

```
struct addrinfo hints, *result, *rp;
…
memset(&hints, 0, sizeof(struct addrinfo));
   hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
   hints.ai_socktype = SOCK_DGRAM; /* Datagram socket */
   hints.ai_flags = AI_PASSIVE;    /* For wildcard IP address */
   hints.ai_protocol = 0;          /* Any protocol */
   hints.ai_canonname = NULL;  hints.ai_addr = NULL;  hints.ai_next = NULL;

   s = getaddrinfo(NULL, argv[1], &hints, &result);
   if (s != 0) {
      fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
      exit(EXIT_FAILURE);
   }
```

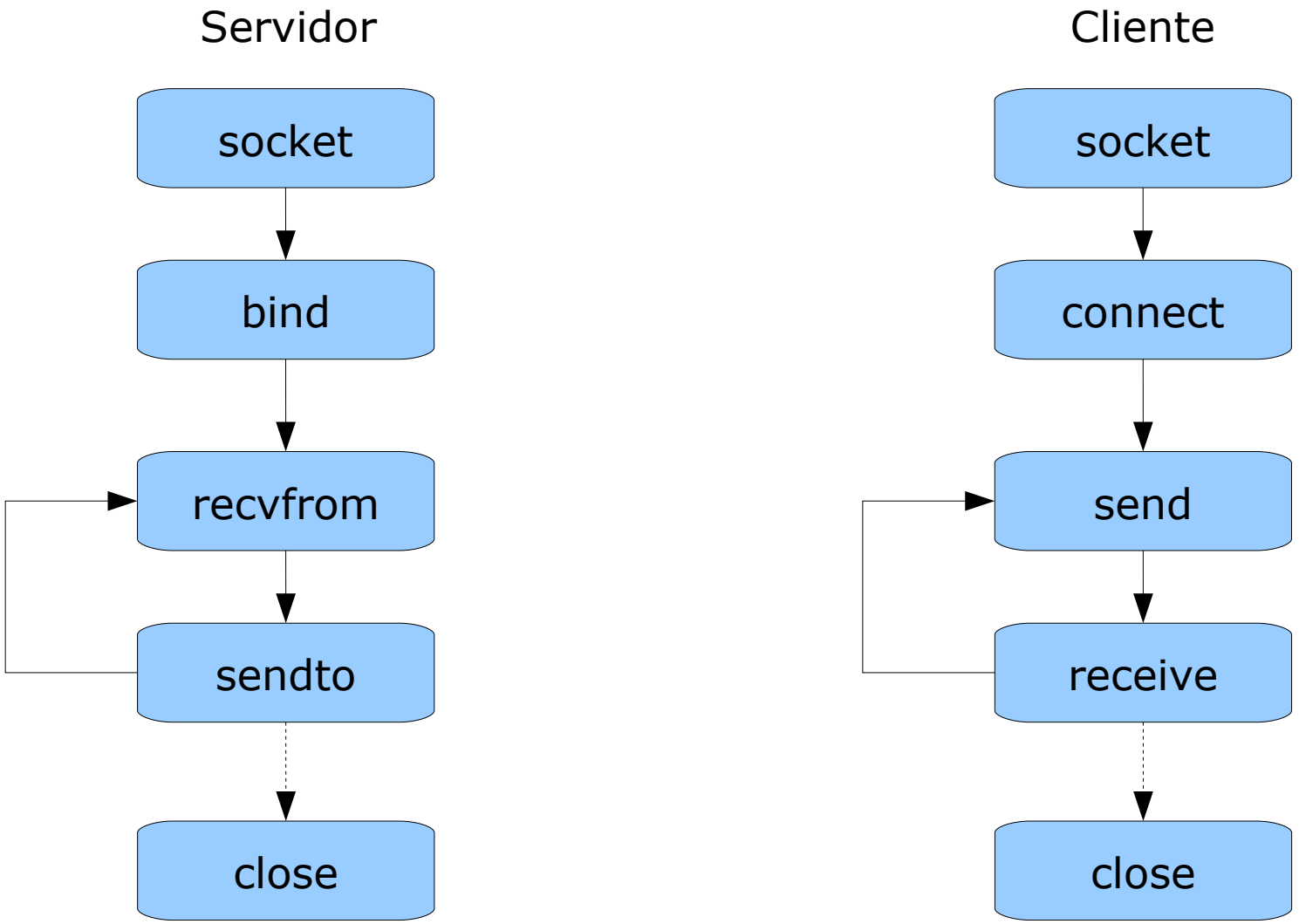
Funciones básicas

- **bind(2)**
  - Asocia una dirección (local) al socket.
    - sockfd: identificador del socket
    - addr: dirección (dirección IPv4/IPv6 + puerto)
    - addrlen: longitud de la dirección

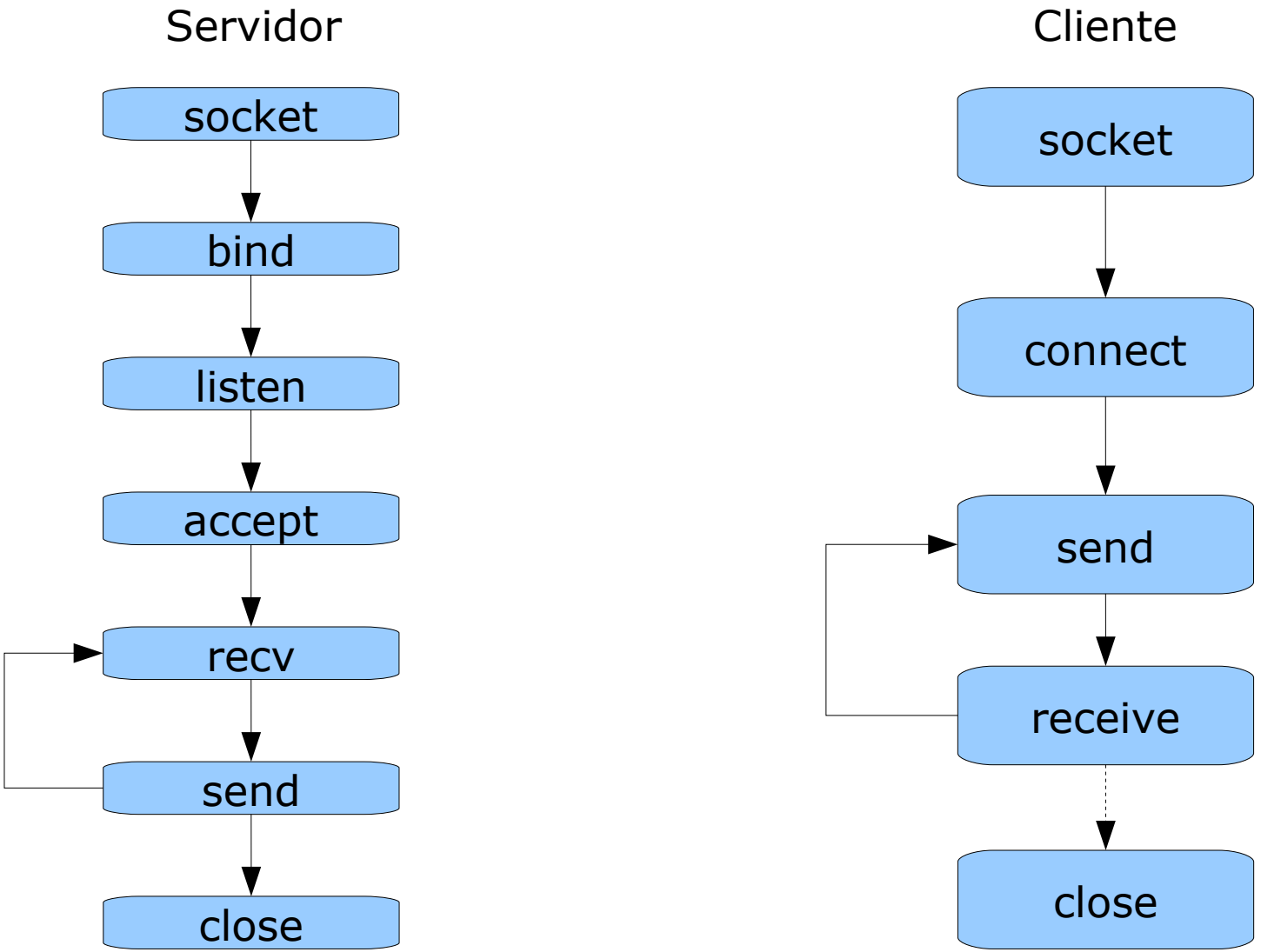
```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd,  const struct sockaddr *addr,
         socklen_t addrlen);
```

```
struct addrinfo hints, *result, *rp;
...
s = getaddrinfo(NULL, argv[1], &hints, &result);
...
for (rp = result; rp != NULL; rp = rp->ai_next) {
       sfd = socket(rp->ai_family, rp->ai_socktype,
              rp->ai_protocol);
       if (sfd == -1)
           continue;

       if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
           break;                  /* Success */

       close(sfd);
   }
```

Sockets UDP

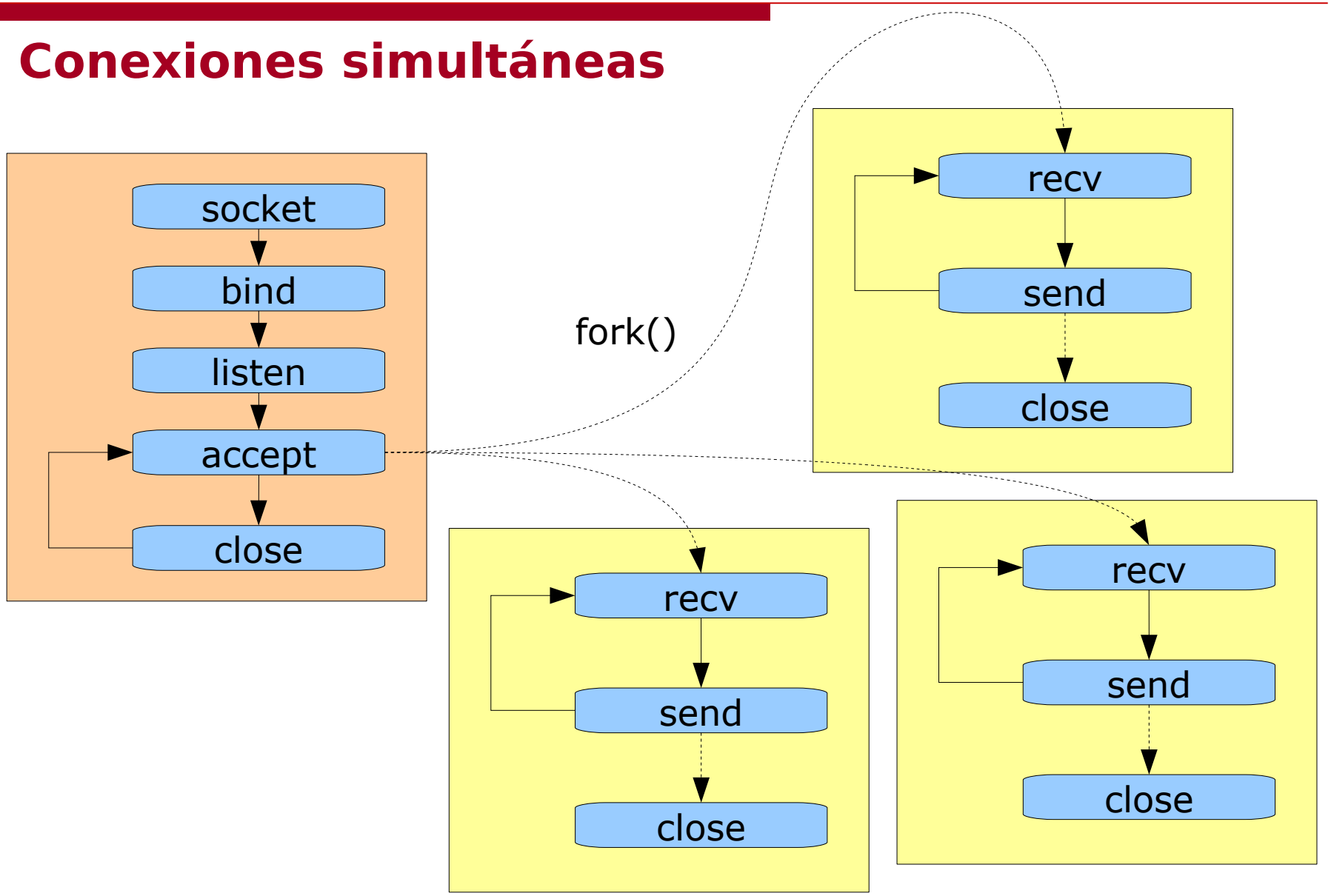

Servidor
socket
bind
recvfrom
sendto
close
Cliente
socket
connect
send
receive
close

Sockets TCP


Servidor
socket
bind
listen
accept
recv
send
close
Cliente
socket
connect
send
receive
close

Sockets TCP

- **Conexiones simultáneas**



Recepción

- **recv(2) / recvfrom(2)**
  - Reciben datos desde un socket.
    - sockfd: descriptor del socket.
    - buf: buffer de recepción.
    - len: longitud del buffer.
    - flags: indicadores (MSG_DONTWAIT...)
    - src_addr: estructura con la dirección del otro extremo.
    - addrlen: longitud de la dirección src_addr.
  - recv(2) se puede utilizar en socket en estado connected.
    - Equivalente a recvfrom(sockfd, buf, len, flags, NULL, NULL)

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

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

ssize_t recvfrom(int sockfd, void *buf, size_t
len, int flags, struct sockaddr *src_addr,
socklen_t *addrlen);
```

```
struct sockaddr_storage peer_addr;
socklen_t peer_addr_len;
ssize_t nread;
char buf[BUF_SIZE];
…
peer_addr_len = sizeof(struct sockaddr_storage);
nread = recvfrom(sfd, buf, BUF_SIZE, 0, (struct sockaddr *) &peer_addr, &peer_addr_len);
if (nread == -1) handle_error("recvfrom");
```

Envío de mensajes

- **send(2) / sendto(2)**
  - Envían datos a través de un socket.
    - sockfd: descriptor del socket.
    - buf: buffer de envío.
    - len: longitud del buffer.
    - flags: indicadores (MSG_DONTWAIT...)
    - dest_addr: dirección del destino (IPv4/IPv6 + puerto)
    - addrlen: longitud de la dirección
  - send() se puede utilizar en sockets en estado connected.
    - Equivalente a sendto(sockfd, buf, len, flags, NULL, NULL)

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

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

ssize_t sendto(int sockfd, const void *buf,
             size_t len, int flags,
             const struct sockaddr *dest_addr,
             socklen_t addrlen);
```

```
struct sockaddr_storage peer_addr;
socklen_t peer_addr_len;
ssize_t nread;
char buf[BUF_SIZE];
…
peer_addr_len = sizeof(struct sockaddr_storage);
if (sendto(sfd, buf, nread, 0, (struct sockaddr *) &peer_addr,  peer_addr_len) != nread)
        handle_error("sendto");
```

Apertura pasiva

- **listen(2)**
  - Deja un socket en escucha.
    - sockfd: descriptor del socket.
      Debe ser del tipo SOCK_STREAM.
    - backlog: máximo número de conexiones pendientes.

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

int listen(int sockfd, int backlog);
```

```
#define LISTEN_BACKLOG 50
int sfd, cfd;
...
if (listen(sfd, LISTEN_BACKLOG) == -1)
        handle_error("listen");
```

Conexión entrante

- **accept(2)**
  - Acepta una nueva conexión sobre un socket y abre un nuevo socket.
  - sockfd: socket en estado listen.
  - addr: estructura que se rellenará con los datos del otro extremo.
  - addrlen: longitud de la dirección. Se inicializa al espacio disponible y nos devuelve la longitud real.

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

int accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);

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

```
struct sockaddr_storage my_addr, peer_addr;
socklen_t peer_addr_size;
...
peer_addr_size = sizeof(struct sockaddr_storage);
cfd = accept(sfd, (struct sockaddr *) &peer_addr,
             &peer_addr_size);
if (cfd == -1)
   handle_error("accept");
```

Apertura activa

- **connect(2)**
  - Conecta un socket con una dirección remota.
  - Se puede usar en SOCK_DGRAM (y luego usar send() y recv() en lugar de sendto() y recvfrom())
    - sockfd: identificador del socket.
    - addr: dirección del extremo remoto.
    - addrlen: longitud efectiva de la dirección.

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

int connect(int sockfd,
      const struct sockaddr *addr,
      socklen_t addrlen);
```

```
struct addrinfo hints;
struct addrinfo *result, *rp;
int sfd, s, j;
...
s = getaddrinfo(argv[1], argv[2], &hints, &result);
...
for (rp = result; rp != NULL; rp = rp->ai_next) {
   sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
...
   if (connect(sfd, rp->ai_addr, rp->ai_addrlen) != -1)
         break;
```

Funciones auxiliares

- **getnameinfo(3)**
  - Es la inversa de getaddrinfo(3)
  - Evita dependencias sobre IPv4/IPv6
    - flags: NI_NAMEREQD, NI_DGRAM, NI_NUMERICHOST, NI_NUMERICSERV, NI_NOFQDN

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

int getnameinfo(const struct sockaddr *addr,
        socklen_t addrlen,
        char *host, socklen_t hostlen,
        char *serv, socklen_t servlen,
        int flags);
```

```
struct sockaddr *addr; /* input */
socklen_t addrlen; /* input */
char hbuf[NI_MAXHOST], sbuf[NI_MAXSERV];

if (getnameinfo(addr, addrlen, hbuf, sizeof(hbuf), sbuf,
            sizeof(sbuf), NI_NUMERICHOST | NI_NUMERICSERV) == 0)
    printf("host=%s, serv=%s\n", hbuf, sbuf);
/* comprobamos si hay resolución inversa */
if (getnameinfo(addr, addrlen, hbuf, sizeof(hbuf), NULL, 0, NI_NAMEREQD))
    printf("could not resolve hostname");
else
    printf("host=%s\n", hbuf);
```

Funciones auxiliares

- **htons(3)**
  - Convirte números entre la representación de la máquina y la representación de la red.
  - Útil para, por ejemplo, los números de puertos.
  - Generalmente no es necesario invocarlas directamente, sino que se puede manejar a través de getaddrinfo(3) y getnameinfo(3).

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

Ejemplo servidor UDP

- **Servidor simple (echo)**

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

#define BUF_SIZE 500

int main(int argc, char *argv[]) {
    struct addrinfo hints;
    struct addrinfo *result, *rp;
    int sfd, s;
    struct sockaddr_storage peer_addr;
    socklen_t peer_addr_len;
    ssize_t nread;
    char buf[BUF_SIZE];

    if (argc != 2) {
        fprintf(stderr, "Usage: %s port\n", argv[0]);
        exit(EXIT_FAILURE);
    }
```

Ejemplo servidor UDP

- **Servidor simple (echo)**

```
memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = AF_UNSPEC;    /* Permite IPv4 o IPv6 */
hints.ai_socktype = SOCK_DGRAM; /* Datagrama */
hints.ai_flags = AI_PASSIVE;    /* Es el servidor. Dejaremos dirección local sin especificar */
hints.ai_protocol = 0;          /* Cualquier protocolo. Puesto que es DGRAM, será UDP */
hints.ai_canonname = NULL;
hints.ai_addr = NULL;
hints.ai_next = NULL;

s = getaddrinfo(NULL, argv[1], &hints, &result);
if (s != 0) {
   fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
   exit(EXIT_FAILURE);
}

/* getaddrinfo() devuelve una lista de direcciones posibles. Vamos probando una a una. */

for (rp = result; rp != NULL; rp = rp->ai_next) {
   sfd = socket(rp->ai_family, rp->ai_socktype,
          rp->ai_protocol);
   if (sfd == -1)
      continue;
```

Ejemplo servidor UDP

- **Servidor simple (echo)**

```
    if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
        break;                  /* Ya tenemos una válida */

    close(sfd);
}

if (rp == NULL) {               /* Ninguna ha servido */
    fprintf(stderr, "Could not bind\n");
    exit(EXIT_FAILURE);
}

freeaddrinfo(result);           /* Ya no la necesitamos. Se libera la lista de estructuras */
```

Ejemplo servidor UDP

- **Servidor simple (echo)**

```
    /* Devolvemos al emisor cada datagrama recibido */

  for (;;) {
      peer_addr_len = sizeof(struct sockaddr_storage);
      nread = recvfrom(sfd, buf, BUF_SIZE, 0, (struct sockaddr *) &peer_addr, &peer_addr_len);
      if (nread == -1)
          continue;               /* Descartamos las peticiones erróneas */

      char host[NI_MAXHOST], service[NI_MAXSERV];

      s = getnameinfo((struct sockaddr *) &peer_addr, peer_addr_len, host, NI_MAXHOST,
                      service, NI_MAXSERV, NI_NUMERICSERV);
     if (s == 0)
          printf("Recibidos %zd bytes desde %s:%s\n", nread, host, service);
      else
          fprintf(stderr, "getnameinfo: %s\n", gai_strerror(s));

      if (sendto(sfd, buf, nread, 0, (struct sockaddr *) &peer_addr,  peer_addr_len) != nread)
          fprintf(stderr, "Error al enviar la respuesta\n");
  }
}
```

Ejemplo cliente UDP

- **Cliente simple**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUF_SIZE 500

int main(int argc, char *argv[]) {
    struct addrinfo hints;
    struct addrinfo *result, *rp;
    int sfd, s, j;
    size_t len;
    ssize_t nread;
    char buf[BUF_SIZE];

    if (argc < 3) {
        fprintf(stderr, "Usage: %s host port msg...\n", argv[0]);
        exit(EXIT_FAILURE);
    }
```

Ejemplo cliente UDP

Cliente simple

```
/* Los parámetros son: host puerto */

memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = AF_UNSPEC;    /* Resuelve a IPv4 o IPv6; podemos poner AF_INET6 */
hints.ai_socktype = SOCK_DGRAM; /* Datagrama */
hints.ai_flags = 0;
hints.ai_protocol = 0;          /* Cualquier protocolo, que será UDP */

s = getaddrinfo(argv[1], argv[2], &hints, &result);
if (s != 0) { /* no se ha encontrado el host y/o el puerto */
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
    exit(EXIT_FAILURE);
}
```

Ejemplo cliente UDP

Cliente simple

```
/* getaddrinfo() devuelve una lista de estructuras addr. Vamos probando una a una,
   hasta que conseguimos hacer socket(2) y connect(2) sobre alguna de ellas.
   Si socket(2) (o connect(2)) falla, (cerramos el socket) y probamos la siguiente */

for (rp = result; rp != NULL; rp = rp->ai_next) {
    sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
    if (sfd == -1)
        continue;

    if (connect(sfd, rp->ai_addr, rp->ai_addrlen) != -1)
        break;                  /* Success */

    close(sfd);
}

if (rp == NULL) {               /* Todas fallaron */
    fprintf(stderr, "Could not connect\n");
    exit(EXIT_FAILURE);
}

freeaddrinfo(result);           /* Ya no se necesita */
```

Ejemplo cliente UDP

- **Cliente simple**

```
/* Se envían los siguientes argumentos y se esperan las respuestas del servidor */
 for (j = 3; j < argc; j++) {
    len = strlen(argv[j]) + 1;       /* +1 para el byte \0 del final de la cadena */
    if (len + 1 > BUF_SIZE) {
       fprintf(stderr, "Se descarta el argumento %d, demasiado largo\n", j);
       continue;
    }

    if (write(sfd, argv[j], len) != len) {
       fprintf(stderr, "write incompleto/truncado\n");
       exit(EXIT_FAILURE);
    }

    nread = read(sfd, buf, BUF_SIZE);
    if (nread == -1) {
       perror("read");
       exit(EXIT_FAILURE);
    }

    printf("Recibidos %zd bytes: %s\n", nread, buf);
 }
 exit(EXIT_SUCCESS);
}
```