

Network Programming

Berkeley Sockets API

Berkeley Sockets API

- ❖ Es una API para comunicar 2 procesos, tanto locales o ejecutándose en máquinas diferentes.
- ❖ Nacida en 4.2BSD Unix en 1982. De-facto estándar.
- ❖ Un socket es una representación (handle) a un endpoint de un canal de comunicación. Similar a un file descriptor.
- ❖ Interface agnóstica de los distintos protocolos. Protocol Family.
 - PF_UNIX: para comunicaciones locales en el mismo sistema.
 - PF_INET: familia de protocolos IPv4.
 - PF_INET6: familia de protocolos IPv6.
 - Otras: AF_IPX, AF_X25, AF_NETLINK, etc.

socket()

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

```
int socket(int socket_family, int socket_type, int protocol);
```

- **socket_family**: PF_UNIX, PF_INET, PF_INET6
- **socket_type**: tipo de comunicación:
 - SOCK_STREAM: reliable, connection oriented, byte stream bidireccional.
 - SOCK_DGRAM: unreliable, connectionless datagrams.
 - SOCK_RAW: interface raw al protocolo de comunicación.
 - SOCK_SEQPACKET: reliable datagrams con secuencia garantizada.
 - SOCK_RDM: reliable datagrams sin garantizar orden.
- **protocol**: protocolo específico de ese tipo. En general 0.
 - IPPROTO_TCP o IPPROTO_UDP para PF_INET.

Addressing: `bind()`

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

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

- ❖ Como la dirección del endpoint de comunicación depende del protocolo, la estructura `sockaddr` es genérica, define el campo común `sa_family`.
- ❖ Distintos valores para `sa_family`, dependiendo del protocol family. A su vez cada protocolo define una estructura adecuada a su address:
 - `PF_UNIX`: `AF_UNIX` y `struct sockaddr_un`.
 - `PF_INET`: `AF_INET` y `struct sockaddr_in`.
 - `PF_INET6`: `AF_INET6` y `struct sockaddr_in6`.

PF_UNIX o PF_LOCAL

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

```
int unix_socket = socket(AF_UNIX, type, 0);
int error = socketpair(AF_UNIX, type, 0, int sv[2]);
```

- ❖ Los addresses en sockets locales son nombres de files:

```
struct sockaddr_un {
    sa_family_t sun_family;           /* AF_UNIX */
    char        sun_path[UNIX_PATH_MAX]; /* pathname */
};
```

- ❖ `socketpair` crea un par de sockets ya conectados en `sv[0]` u `sv[1]`.

Connection oriented sockets

- ❖ En los sockets connection oriented es necesario establecer la conexión antes de poder enviar datos.
- ❖ Asimetría, comportamientos distintos para quien comienza una conexión (*cliente*) y el que espera recibirla (*servidor*).
- ❖ El lado server de la conexión que va recibir conexiones tiene que poner al socket este modo llamando a la función `listen()`.
- ❖ El cliente utiliza la función `connect()` para conectarse a un servidor.
- ❖ El servidor para recibir una conexión específica llama a la función `accept()` que le retorna un nuevo socket handle que hace referencia a la conexión establecida.

Connection oriented sockets

- ❖ En los sockets connection oriented es necesario establecer la conexión antes de poder enviar datos.
- ❖ Asimetría, comportamientos distintos para quien comienza una conexión (*cliente*) y el que espera recibirla (*servidor*).
- ❖ El lado server de la conexión que va recibir conexiones tiene que poner al socket este modo llamando a la función `listen()`.
- ❖ El cliente utiliza la función `connect()` para conectarse a un servidor.
- ❖ El servidor para recibir una conexión específica llama a la función `accept()` que le retorna un nuevo socket handle que hace referencia a la conexión establecida.

Connection oriented sockets

❖ Servidor

```
int ss = socket(family, SOCK_STREAM, 0);

bind(ss, /* server address */);

listen(s, backlog);

int cs = accept(s, /* &client addr */ );

/*
 * do protocol con socket cs
 */
```

❖ Cliente

```
int s = socket(family, SOCK_STREAM, 0);

bind(s, /* client address */);

connect(s, /* server address */);

/*
 * do protocol con socket s
 */
```


PF_UNIX o PF_LOCAL

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

```
int unix_socket = socket(PF_UNIX, type, 0);
int error = socketpair(PF_UNIX, type, 0, int sv[2]);
```

- ❖ Los addresses en sockets locales son nombres de files:

```
struct sockaddr_un {
    sa_family_t sun_family;           /* AF_UNIX */
    char        sun_path[UNIX_PATH_MAX]; /* pathname */
};
```

- ❖ `socketpair` crea un par de sockets ya conectados en `sv[0]` u `sv[1]`.

PF_INET

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

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

```
#include <netinet/ip.h>
```

```
tcp_socket = socket(PF_INET, SOCK_STREAM, 0);
```

```
udp_socket = socket(PF_INET, SOCK_DGRAM, 0);
```

```
raw_socket = socket(PF_INET, SOCK_RAW, protocol);
```

❖ Los addresses tienen una dirección de IP y un puerto:

```
struct sockaddr_in {  
    sa_family_t    sin_family;    /* AF_INET */  
    in_port_t      sin_port;      /* port in network byte order */  
    struct in_addr sin_addr;      /* internet address */  
};
```

Enviando datos por sockets

```
#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);
```

```
ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
```

- `send()` solo se puede utilizar para sockets conectados. Similar a `write()`, salvo por los flags.
- Si `sendto()` se utiliza en sockets conectados, los argumentos de destino son ignorados.
- `sendmsg()` se utiliza para evitar copias de buffers. Scatter/gather list.

Recibiendo datos por sockets

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

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

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

```
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

- `recv()` solo se puede utilizar para sockets conectados. Similar a `read()`, salvo por los flags.
- `recv()` es similar a `recvfrom()` con argumentos de dirección de origen en `NULL`.
- `recvmsg()` se utiliza para evitar copias de buffers. Scatter/gather list.

Cerrando un socket

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

```
int shutdown(int sockfd, int how);
```

```
int close(int sockfd);
```

- `shutdown()` cierra una o todas las partes de una conexión bidireccional, valores de `how`:
 - `SHUT_RD`: cierra canal de lectura.
 - `SHUT_WR`: cierra canal de escritura.
 - `SHUT_RDWR`: cierra ambos canales.
- `close()` cierra el socket handle.