Apunte de Sockets

1. Sockets

Un file descriptor, en particular uno de socket, tiene tipo int. Recordar de la clase que se puede crear un socket usando:

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

Para trabajar con sockets de internet usaremos el *domain* AF_INET. Para trabajar con mensajes UDP (sin conexión), usaremos el *type* SOCK_DGRAM.

Para TCP, usaremos el *type* SOCK_STREAM. Recordar que en protocol en general se utiliza un 0.

Un socket se cierra con close(int socket).

2. Esquema de conexión

A continuación se muestra el esquema de conexión con sockets AF_INET que utilizan SOCK_STREAM como protocolo. El servidor hace *passive open* y el cliente *active open*.
Tener en cuenta que el servidor no necesita abrir puertos para cada nueva conexión entre clientes.

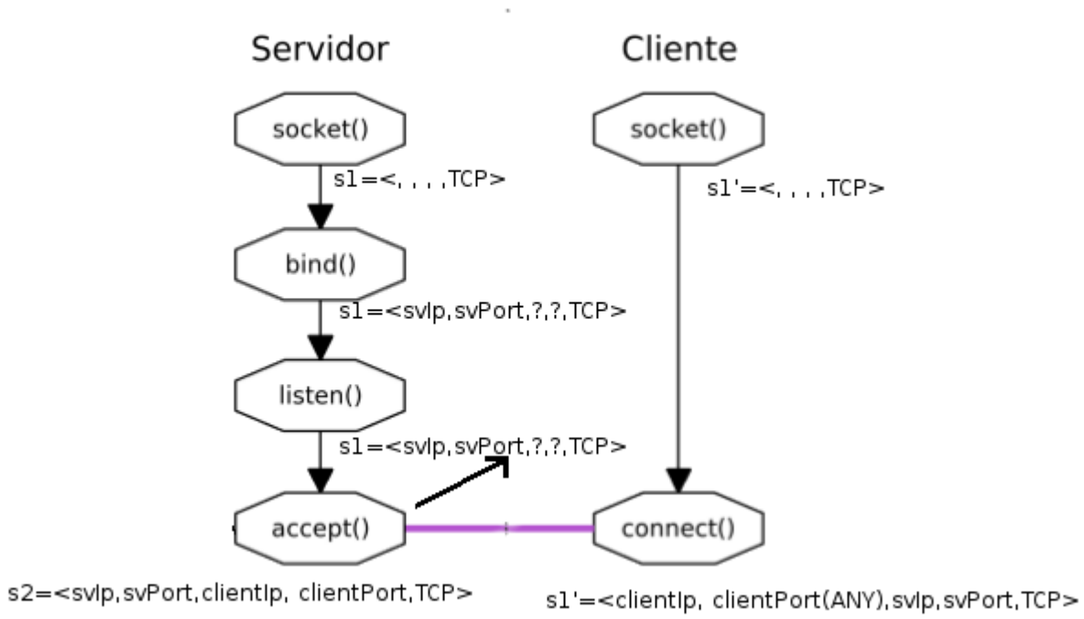

Figura 1: Esquema de conexión TCP

3. Direcciones de Internet

Para representar una dirección de internet se usa la estructura presentada a continuación:

```
struct sockaddr_in {
     short sin_family; //dominio, usamos AF\_INET
     unsigned short sin_port; //numero de puerto
     struct in_addr sin_addr; //direccion IP
     char sin_zero[8 //relleno(no se usa)
};
```

```
Donde la estructura que contiene la dirección IP es la siguiente:
struct in_addr {
     unsigned long s_addr;  //esto es un long de 32 bits
};
```

4. Network byte order

Las estructuras mencionadas arriba necesitan tener el puerto y la dirección IP almacenadas en un formato conocido como Network byte order. Para ello contamos con funciones de conversión:

- unsigned short htons(unsigned short us) convierte un short del host (máquina local) en un short de la red.
- unsigned long htonl(unsigned long ul) análoga pero convierte longs.

5. Resolver direcciones IP

Para convertir una cadena de caracteres que contiene una dirección IP (por ejemplo: "127.0.0.1") en una estructura in_addr usamos, por ejemplo:

- inet_pton(AF_INET, "127.0.0.1", &(remote.sin_addr));

Esta función ya nos deja la dirección IP en formato Network byte order dentro del in_addr apuntado por el último parámetro (en este caso, llamado remote).

6. Enviar paquetes UDP

Para enviar o recibir spaquetes UDP podemos usar la siguiente llamadas al sistema:

- ssize_t sendto(int s, const void *buf, size_t len, int flags, const sockaddr* to, socklen_t tolen);
- recvfrom(int s, void *buf, size_t len , int flags, struct sockaddr *src_addr, socklen_t *addrlen);

En el caso de sendto, se usa el socket *s* para enviar len bytes de datos desde el buffer apuntado por *buf* hacia la dirección apuntada por *to*, cuya longitud es de *tolen*. EL caso de recvfrom es análogo pero para la recepción de mensajes.

7. Includes recomendados

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
```

8. Funciones útiles

- char* fgets(char* s, int size, FILE* stream); Leer una linea de a lo sumo size
- int strncmp(const char *s1, const char *s2, size_t n); Comparar dos cadenas s1 y s2 de longitud a lo sumo n.
- char *strcat (char * destination, const char * source); Concatena una copia de la cadena source a la de destintation.
- char * strtok (char * str, const char * delimiters); Sucesivas llamadas a esta función permiten separar str en tokens, secuencias de caracteres contiguos separados por alguno de los caracteres especificados en los delimitadores