Oscar García Lorenzo
Tomás Fernández Pena
José Carlos Cabaleiro Domínguez

Grao en Robótica

Universidade de Santiago de Compostela

Redese Comunicacións, 3º Curso GrR

citius.usc.es

Índice

Índice

Sockets

Un **socket** é unha estructura software nun nodo que serve coma punto final para enviar e recibir datos

- Establecen unha interface entre a aplicación e a capa de transporte
- Analoxía: servizo de correo postal
  - Aplicación: usuario que deposita a carta nunha caixa de correo
  - Capa de transporte: carteiro que recolle a carta da caixa de correo
  - Socket: a caixa de correo
- Hai diferentes tipos de sockets: locais, TCP, UDP, raw

Para establecer unha comunicación entre dous puntos necesitamos dous sockets, un en cada punto da mesma

- Socket orixe
- Socket destino

Parámetros dos sockets

- O número de socket: variábel `int` identificadora do socket
- O dominio de comunicación (`AF_LOCAL`, `AF_INET`, `AF_INET6`, etc.)
  - Indica a familia da dirección a usar na comunicación
- O tipo de socket (`SOCK_STREAM`, `SOCK_DGRAM`, `SOCK_RAW`, etc.)
- O protocolo a usar na comunicación (`IPPROTO_TCP`, `IPPROTO_UDP`, `IPPROTO_ICMP`, etc.)
  - Normalmente, para cada dominio/tipo hai un único protocolo (pe, `AF_INET`/`SOCK_STREAM` → `IPPROTO_TCP`)
  - Pódese usar 0 como “calquer protocolo”
  - A lista de protocolos e o seu número correspondente pódese ver no ficheiro `/etc/protocols` ou en https://www.iana.org/assignments/protocol-numbers/
- La dirección del socket: se introduce a través de la estructura `struct sockaddr`

Sockets

- Sockets orientados a conexión (TCP)
  - Dous programas con roles (estructura) diferentes
    - Cliente: solicita conexión
    - Servidor: acepta a conexión
    - Despois transmitense datos
- Sockets non orientados a conexión ou sen conexión (UDP)
  - Dous programas con roles (estructura) similares
    - Extremo 1: envía e/ou recibe datos
    - Extremo 2: recibe e/ou envía datos

Principais funcións relacionadas cos sockets

En C actualizan a variable `int errno` e imprimese ca función `perror(const char *s)`, en Python lanzan eventos e excepcións

- `int socket()` crear un socket
- `int bind()` asignar dirección a un socket
- `int listen()` marcar un socket como pasivo (para poder atender peticións de clientes)
- `int connect()` en clientes, para solicitar a conexión cun servidor
- `int accept()` en servidores, para aceptar a conexión de clientes
- `int send()/recv()` enviar/recibir datos entre clientes e servidores
- `int sendto()/recvfrom()` enviar/recibir datos cando non existe conexión establecida
- `int close` pechar un socket

Principais funcións relacionadas cos sockets

En Python existen funcións de máis alto nivel que xuntan varias das anteriores, pero non as usaremos, xa que ocultan fases do funcionamento

- `socketpair()` Crea un par de sockets xa conectados
- `create_connection()` Crea unha conexióne devolve o socket directamente
- `create_server()` Crea un socket con dirección xa asignada

Función `socket()`: creación dun socket: C

```
int socket(int domain, int style, int protocol)
```

- Parámetros:
  - `domain` enteiro que debe valer `AF_INET` para direcciones IPv4 ou `AF_INET6` para IPv6
  - `style` usaremos `SOCK_STREAM` (orientado a conexión, TCP) ou `SOCK_DGRAM` (non orientado a conexión, UDP)
  - `protocol` por defecto para cada par dominio/tipo (protocolo 0)
- Valor devolto: enteiro identificador do socket en caso de éxito, -1 en caso de erro
- Necesario incluir `sys/socket.h` y `sys/types.h`

Función `socket()`: creación dun socket: Python

```
socket.socket(family=AF_INET, type=SOCK_STREAM,
  proto=0, fileno=None)
```

- Parámetros:
  - `family` enteiro que debe valer `AF_INET` para direcciones IPv4 ou `AF_INET6` para IPv6
  - `type` usaremos `SOCK_STREAM` (orientado a conexión, TCP) ou `SOCK_DGRAM` (non orientado a conexión, UDP)
  - `protocol` por defecto para cada par dominio/tipo (protocolo 0)
  - `fileno` para coller os parámetros desde ficheiro
- Valor devolto: o socket creado

Función `bind()`: asignación de dirección a un socket: C

```
int bind(int socket, struct sockaddr *addr,
         socklen_t length)
```

- Parámetros:
  - `socket` enteiro identificador do socket
  - `addr` punteiro a un `struct sockaddr` coa dirección que se quere asignar ao socket
    - Pode ser IPv4 o IPv6, pero hai que convertila ao tipo xenérico (`struct sockaddr *`) na chamada á función
    - No caso servidor debe poñerse `INADDR_ANY` para que poda aceptar conexións a través de cualquera das interfaces do mesmo[1]
  - `length` tamaño da estructura addr
- Valor devolto: 0 en caso de éxito, -1 en caso de erro

[1] Se se pon localhost, so aceptará conexións á interface de lazo de volta

Función `bind()`: asignación de dirección a un socket: Python

`socket.bind(address)`

- Parámetros:
  - `address` dirección que se quere asignar ao socket
    - Pode ser IPv4 o IPv6
    - O fomato de `address` cambia segundo a familia, para IPv4, recordade, é un par (dirección, porto)
    - No caso servidor, para que poda aceptar conexións a través de cualquera das interfaces do mesmo, a dirección debe de estar baleira e ter so un porto.

Función `listen()`: indicación de poñerse á escoita: C

```
int listen(int socket, unsigned int n)
```

- Usase en servidores:
  - Marcar o socket coma pasivo, é dicir, poderá escoitar posibles conexións de clientes $\Longrightarrow$ **socket servidor**
- Parámetros:
  - `socket` enteiro identificador do socket
  - `n` número de peticións de clientes que poden estar en cola á espera de seren atendidas
- Valor devolto: 0 en caso de éxito, -1 en caso de erro

Función `listen()`: indicación de poñerse á escoita: Python

`socket.listen([backlog]`

- Usase en servidores:
  - Marcar o socket coma pasivo, é dicir, poderá escoitar posibles conexións de clientes $\Longrightarrow$ **socket servidor**
- Parámetros:
  - `backlog` número de peticións de clientes que poden estar en cola á espera de seren atendidas, se non se escpecifica escollese unha razonable

Función `connect()`: solicitude de conexión: C

```
int connect(int socket, struct sockaddr *addr,
            socklen_t length)
```

- Usase en clientes:
  - Solicitar a conexión cun servidor
- Parámetros: os mesmos que a función `bind()`
  - `socket` entero identificador do socket
  - `addr` punteiro a un `struct sockaddr` coa dirección do socket do servidor
    - Pode ser IPv4 o IPv6, pero hai que convertila ao tipo xenérico (`struct sockaddr *`) na chamada á función
  - `length` tamaño da estructura `addr`
- A función espera a que o servidor responda[2], pero se o servidor non se está executando devolve un erro
- Valor devolto: 0 en caso de conexión, -1 en caso de erro

[2] Modificando as opcións de creación do socket podería facerse non bloqueante

Función `connect()`: solicitude de conexión: Python

`socket.connect(address)`

- Usase en clientes:
  - Solicitar a conexión cun servidor
- Parámetros: os mesmos que a función `bind()`
  - `address` dirección
    - Pode ser IPv4 o IPv6, pero recordade que son diferentes (en IPv4 par ((dirección, porto)))
- A función espera a que o servidor responda[3], pero se o servidor non se está executando devolve un erro
- Devolve diversas excepcións se hai problemas
- Existe a función `socket.connect_ex(address)` que devolve erros estilo C, para compatibilidade

[3] Modificando o socket podería facerse non bloqueante, campos `socket.timeout`

Función `accept()`: aceptar a conexión: C

```
int accept(int socket, struct sockaddr *addr,
           socklen_t *length_ptr)
```

- Usase en servidores:
  - Atender as peticións de clientes
- Parámetros:
  - `socket` enteiro identificador do socket de servidor
  - `addr` saída que é un punteiro a un `struct sockaddr` coa dirección do cliente que se conectou
  - `length_ptr` punteiro que é parámetro de entrada indicando o tamaño da estructura `addr` e de saída co espazo real consumido
- A función queda á espera[4] e cando se acepta a conexión, devolve un novo socket $\Longrightarrow$ **socket de conexión**
- Valor devolto: enteiro identificador do socket de conexión, -1 en caso de erro

[4] Modificando as opcións do socket podería facerse non bloqueante

Función `accept()`: aceptar a conexión: Python

`socket.accept()`

- Usase en servidores:
  - Atender as peticións de clientes
- Parámetros:
  - Nada, pero o socket ten que estar enlazado a unha dirección
- A función queda á espera[5] e cando se acepta a conexión, devolve un novo socket $\Longrightarrow$ **socket de conexión**
- Valor devolto: Par `(conn, address)`
  - `conn` é o novo socket de conexión
  - `address` é a dirección desde a que se conectou (realmente a dirección enlazada co socket do outro lado) (recordade, par `(dirección, porto)` para IPv4)

[5] Modificando as opcións do socket podería facerse non bloqueante

Función `send()`: envío de datos: C

```
ssize_t send(int socket, void *buffer,
             size_t size, int flags)
```

- Envío de datos entre clientes e servidores
- Parámetros:
  - `socket` enteiro identificador do socket
  - `buffer` punteiro aos datos a enviar
  - `size` número de bytes a enviar
  - `flags` opcións do envío. Por defecto 0
- Valor devolto[6]: número de bytes transmitidos (non implica que se reciban sen erros), -1 en caso de erro

[6] O tipo `ssize_t` é como `size_t` pero además pode ser negativo

Función `send()`: envío de datos: Python

`socket.send(bytes[, flags])`

- Envío de datos entre clientes e servidores
- Parámetros:
  - `bytes` datos a enviar
  - `flags` opcións do envío, definidas polo SO. Por defecto 0
- Valor devolto: O número de bytes enviados
- En Python 3 para enviar *string* podedes usar o método *encode()*

Función `recv()`: recepción de datos: C

```
ssize_t recv(int socket, void *buffer,
             size_t size, int flags)
```

- Recepción de datos entre clientes e servidores
- Parámetros:
  - `socket` enteiro identificador do socket
  - `buffer` punteiro onde se recibirán os datos
  - `size` número máximo de bytes a recibir
  - `flags` opcións da recepción[7]. Por defecto 0
- A función espera a que os datos cheguen[8] mentras o socket de conexión esté aberto
- Se se pecha o socket de conexión, sae devolvendo un 0
- Valor devolto: número de bytes recibidos, -1 en caso de erro

[7] Por exemplo, non borrar datos ou facela non bloqueante

[8] Salvo que se cambiase o comportamento por defecto

Función `recv()`: recepción de datos: Python

`socket.recv(bufsize[, flags])`

- Recepción de datos entre clientes e servidores
- Parámetros:
  - `buffsize` O tamaño do buffer de recepción
    - Debe de ser un valor non moi grande, por como funcioan as cousas, potencia de 2 ata 4096
    - OLLO: Non implica que a mensaxe teña que ter ese tamaño, podese recoller en varios `recv()`
  - `flags` opcións da recepción[9]. Por defecto 0
- A función espera a que os datos cheguen[10] mentras o socket de conexión esté aberto
- Se se pecha o socket de conexión, sae devolvendo unha exepción
- Valor devolto: os datos recibidos
  - En Python 3 para pasar a *string* podedes usar o método *decode()*

[9] Por exemplo, non borrar datos ou facela non bloqueante

[10] Salvo que se cambiase o comportamento por defecto

Notas sobre envíos e recepcións

- En Python2 sempre se envían cadeas de caracteres (Que son o máis próximo a enviar bytes tal cual)
- En Python3 envíanse bytes
  - Usar `encode()` e `decode()`
- As mensaxes poden ser de maior tamaño que os buffers, hai que seguir chamando a `recv` ata recibir toda a mensaxe ou `send` se non entrou completa
- Pero non se sabe cando parar, xa que non se coñece o tamaño final, podes estar esperando `recv` para sempre
- Solución típica: Enviar primeiro a lonxitude total dos datos.
- Se sempre se pecha o socket despois a transmisión, recibir 0 bytes indica o final (HTTP con conexión non persistentes), pero se queda aberto, non é seguro
- Nas prácticas usaremos mensaxes pequenas e non debería haber problema

Función `sendto()`: envío de datos: C

```
ssize_t sendto(int socket, void *buffer, size_t
               size, int flags, struct sockaddr
               *addr, socklen_t length)
```

- Usase para enviar datos cando non hai conexión
- Parámetros:
  - `socket` enteiro identificador do socket
  - `buffer` punteiro aos datos a enviar
  - `size` número de bytes a enviar
  - `flags` opcións do envío. Por defecto 0
  - `addr` punteiro a un `struct sockaddr` coa dirección á que se quere enviar
  - `length` tamaño da estructura `addr`
- Valor devolto: número de bytes transmitidos (non implica que se reciban sen erros), -1 en caso de error

Función `sendto()`: envío de datos: Python

```
socket.sendto(bytes, flags, address)
```

- Usase para enviar datos cando non hai conexión
- Parámetros:
  - `bytes` datos a enviar
  - `flags` opcións do envío, definidas polo SO. Por defecto 0
  - `address` dirección de envío (recordade, formato depende de se IPv4 ou IPv6)
- Valor devolto: O número de bytes enviados

Función `recvfrom()`: recepción de datos: C

```
ssize_t recvfrom(int socket, void *buffer, size_t
                 size, int flags, struct sockaddr
                 *addr, socklen_t *length-ptr)
```

- Usase para recibir datos cando non hai conexión
- Parámetros:
  - `socket` enteiro identificador do socket
  - `buffer` punteiro onde se recibirán os datos
  - `size` número máximo de bytes a recibir
  - `flags` opcións da recepción (igual que `recv()`). Por defecto 0
  - `addr` saída que é un punteiro a un `struct sockaddr` ca dirección da procedencia do paquete
  - `length_ptr` punteiro que é parámetro de entrada indicando o tamaño da estructura `addr` e de saída co espazo real consumido[11]
- A función espera a que cheguen datos[12]
- Valor devolto: número de bytes recibidos, -1 en caso de erro

[11] Se `addr` é `NULL` e `length` es 0, indica que non interesa a procedencia

[12] Salvo que se cambiase o comportamento por defecto

Función `recvfrom()`: recepción de datos: Python

`socket.recvfrom(bufsize[, flags])`

- Usase para recibir datos cando non hai conexión
- Parámetros:
  - `buffsize` O tamaño do buffer de recepción
    - Debe de ser un valor non moi grande, por como funcioan as cousas, potencia de 2 ata 4096
    - OLLO: Non implica que a mensaxe teña que ter ese tamaño, pero se é maior outro `recvfrom()` non a recuperará, pérdese a parte que non entra (en UDP)
  - `flags` opcións da recepción (igual que `recv()`). Por defecto 0
- A función espera a que cheguen datos[13]
- Valor devolto: par `(bytes, address)` coa dirección da que veñen os datos e os datos recibidos

[13] Salvo que se cambiase o comportamento por defecto

Resumen das funcións de envío e recepción

Funciones de envío:

- `send()` para envíos orientados a conexión
- `sendto()` para envíos orientados a conexión e sen conexión
  - `send(sockfd, buf, len, flags);` equivale a `sendto(sockfd, buf, len, flags, NULL, 0);`

Funcións de recepción:

- `recv()` para recepcións orientados a conexión
- `recvfrom()` para recepcións orientadas a conexión e sen conexión
  - `recv(sockfd, buf, len, flags);` equivale a `recvfrom(sockfd, buf, len, flags, NULL, 0);`

Na práctica

- `send()` e `recv()` para orientado a conexión (TCP)
- `sendto()` e `recvfrom()` para sin conexión (UDP)

Función `close()`: peche do socket: C

```
int close(int socket)
```

- Pecha o socket
- Parámetros:
  - `socket` enteiro identificador do socket

Función `close()`: peche do socket: Python

`socket.close()`

- Pecha o socket
- Pechanse automáticamente co recolector de basura, pero recomendase pechalos manualmente

Outras funciónsrelacionadas con los sockets

```
int shutdown(int socket, int how)
```

- Pecha o socket, pero permite axustar as accións: 0 pecha a recepción, 1 pecha a emisión e 2 pecha ambas[14]
- Mesmo en Python: `socket.shutdown(how)`

```
int getsockname(int socket, struct sockaddr *addr,
                socklen_t *length-ptr)
```

- A partir dun socket, proporciona a dirección e o seu tamaño
- `socket.getsockname()`

[14] Como a función `close()`

Outras funcións relacionadas con los sockets

```
int getpeername(int socket, struct sockaddr *addr,
                socklen_t *length-ptr)
```

- A partir dun socket, proporciona a dirección e o seu tamaño de quen está conectado
- `socket.getpeername()`

`getsockopt()`, `setsockopt()`, `fcntl()` e `ioctl()`

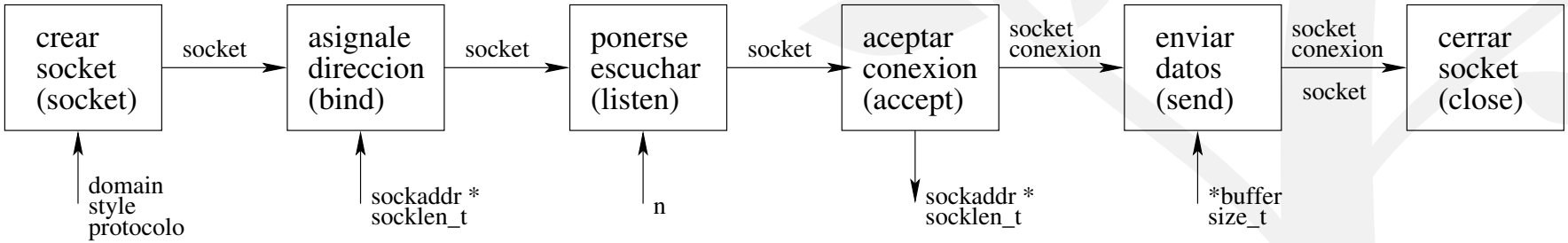
- Ler ou modificar as opcións do socket, por exemplo, o carácter non bloqueante
- `socket.getsockopt(level, optname[, buflen])` pero por compatibilidade con C
- `socket.setdefaulttimeout(timeout)`, `socket.sethostname(name)`

Índice

Sockets IPv4 orientados a conexión

Programa servidor e programa cliente básicos: o servidor envía unha mensaxe ao cliente

- **Programa servidor**
  - Elixir un número de porto[15] no que escoitará peticións de clientes
- Programa cliente
  - IP do servidor
  - Porto que o servidor usa para escoitar peticións
- **Esquema do programa servidor**



[15] Porto maior que `IPPORT_USERRESERVED`

Programa servidor TCP: C

- Incluir as cabeceiras necesarias. Usar `man`
- Declarar variabeis:
  - Dúas `int` para os sockets (socket de servidor e socket de conexión)
  - Un `struct sockaddr_in` para a dirección
  - Un `socklen_t` para o tamaño
  - Inicializar un string para a mensaxe[16]
- **Crear o socket** coa función `socket()`
  - `domain` AF_INET (IPv4)
  - `style` SOCK_STREAM (orientado a conexión)
  - `protocol` 0 (valor por defecto)
  - Devolve o `int` que identifica o **socket de servidor**
  - Comprobar que se creou satisfactoriamente: algo como

```
if(socket < 0){
    perror("Non se puido crear o socket"); exit(EXIT_FAILURE);
}
```

[16] Con `strlen` obtense a lonxitude da mensaxe sen o terminador de string

Programa servidor TCP: C

- **Asignar dirección ao socket** coa función `bind()`
  - Na estructura `sockaddr_in` indicanse:
    - AF_INET
    - O porto en orden de rede
    - E como IP cualquera coa macro `INADDR_ANY` con algo similar a `direccion.sin_addr.s_addr=htonl(INADDR_ANY)`
  - Convertila a `struct sockaddr *` na chamada á función
  - O tamaño obtiense con `sizeof(struct sockaddr_in)`
  - Comprobar que se creou satisfactoriamente: algo como

```
if(bind(socket, (struct sockaddr *) direccion, tamano) < 0){
    perror("Non se puido asignar direccion"); exit(EXIT_FAILURE);
}
```

- Marcar o socket como pasivo (para que poida **escoitar** peticións) ca función `listen()`
  - Número de solicitudes á espera: calquer valor > 0
  - Comprobar que non se produciu erro

Programa servidor TCP: C

- **Aceptar a conexión** coa función `accept()`
  - Parámetros iguais que na función `bind()`
    - A estrutura `addr` é de saída: dirección do cliente
    - O tamaño da dirección é de entrada e de salida
  - Devolve un número de socket $\Longrightarrow$ **socket de conexión**, que se usará coas funcións de envío e recepción
  - Quedase esperando a que un cliente solicite conexión
  - Comprobar que se creou satisfactoriamente: algo como

```
if((sc = accept(ss, (struct sockaddr *) &direccion, &tam)) < 0){
    perror("Non se puido aceptar a conexion");
    exit(EXIT_FAILURE);
}
```

  - Imprimir a **IP e o porto** de quen se conectou

Programa servidor TCP: C

- **Enviar**[17] un mensaxe ao cliente coa función `send()`
  - Parámetros:
    - O socket de conexión
    - A mensaxe
    - O tamaño da mensaxe
    - `flags` o valor por defecto, 0
  - Comprobar que se enviou correctamente
  - Imprimir o **número de bytes enviados**
- **Pechar os sockets** coa función `close()`
  - Parámetro: o identificador do socket que se desexa pechar
- **Comprobar** o seu funcionamento
  - O programa bloquease esperando unha conexión
  - Nun terminal escribese `telnet IP porto`
    - `IP` é a IP onde se executa o servidor
    - `porto` é o porto elexido
  - Recibirase a mensaxe do servidor e o programa terminará

[17] Tamén se podería recibir unha mensaxe do cliente coa función `recv`

Programa servidor TCP: C

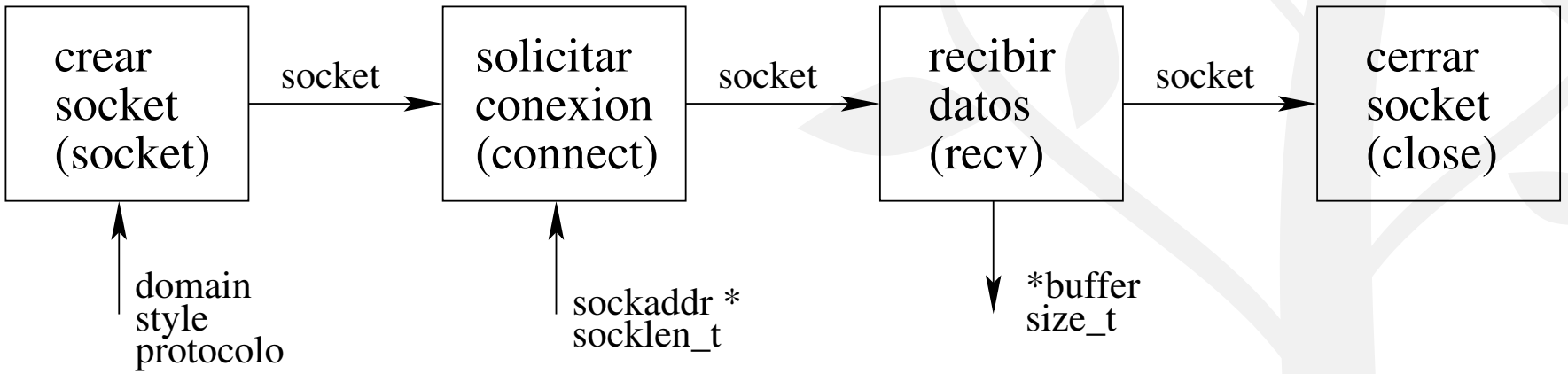
- Permitir elexir o porto en tempo de execución usando os argumentos do `main()`
- Permitir que o servidor atenda múltiples conexións de clientes (secuencialmente) con un esquema do tipo:

```
while(1){
  accept();
  send();
  close(socket_conexion);
}
```

Sockets IPv4 orientados a conexión

- Programa servidor
  - Elixir un número de porto no que escoitará peticións de clientes
- **Programa cliente**
  - IP del servidor
  - Porto que o servidor usa para escoitar peticións

- **Esquema do programa cliente**



Programa cliente TCP: C

- Incluir as cabeceiras: as mesmas que no servidor
- Declarar un `int` para o socket, un `struct sockaddr_in` para a dirección, un `socklen_t` para o tamaño e declarar un string para recibir a mensaxe[18]
- **Crear el socket** con la función `socket()`
  - Do mesmo modo que no servidor
- **Inicializar** a estrutura `sockaddr_in` coa dirección e porto do servidor
  - A IP é a dirección onde se executará o servidor
    `inet_pton(AF_INET,IP_text,&direccion.sin_addr);`
  - Se se executa no mesmo computador, `IP_text` é a do lazo de volta, 127.0.0.1
- **Solicitar a conexión** coa función `connect()`
  - Mesmos argumentos que a función `bind()` do servidor

[18] Asegurarse que ten espazo suficiente para a mensaxe

Programa cliente TCP: C

- **Recibir**[19] a mensaxe do servidor coa función `recv()`
  - Parámetros:
    - O socket
    - A mensaxe
    - O tamaño máximo da mensaje
    - `flags` o valor por defecto, 0
  - Comprobar que se enviou correctamente
  - Imprimir a mensaxe e o número de bytes recibidos[20]
- **Pechar o socket** coa función `close()`
- **Comprobar** o seu funcionamento
  - Executar o programa servidor
  - Noutro terminal executar o cliente
    - Se todo é correcto, verase a mensaxe do servidor e terminará
  - Tamén se pode executar un servidor con `nc -lk puerto`
  - Noutro terminal executar o cliente
  - No primer terminal escribir unha mensaxe

[19] Tamén se podería enviar unha mensaxe ao servidor coa función `send`

[20] O que devolve a función `recv()`, non do tamaño do string

Programa cliente TCP

- Facer que o programa cliente se poda conectar co servidor executándose noutro computador
  - Permitir elexir a IP e o porto do servidor en tempo de execución usando os argumentos do `main()`
  - Cambiase a IP do lazo de volta pola IP do servidor
  - Para conectarse a outro equipo usase: `ssh [usuario@]ip_ordenador` o `ssh [usuario@]nome_computador`
    - Se o usuario é o mesmo, non é necesario especificalo

Programas en Python

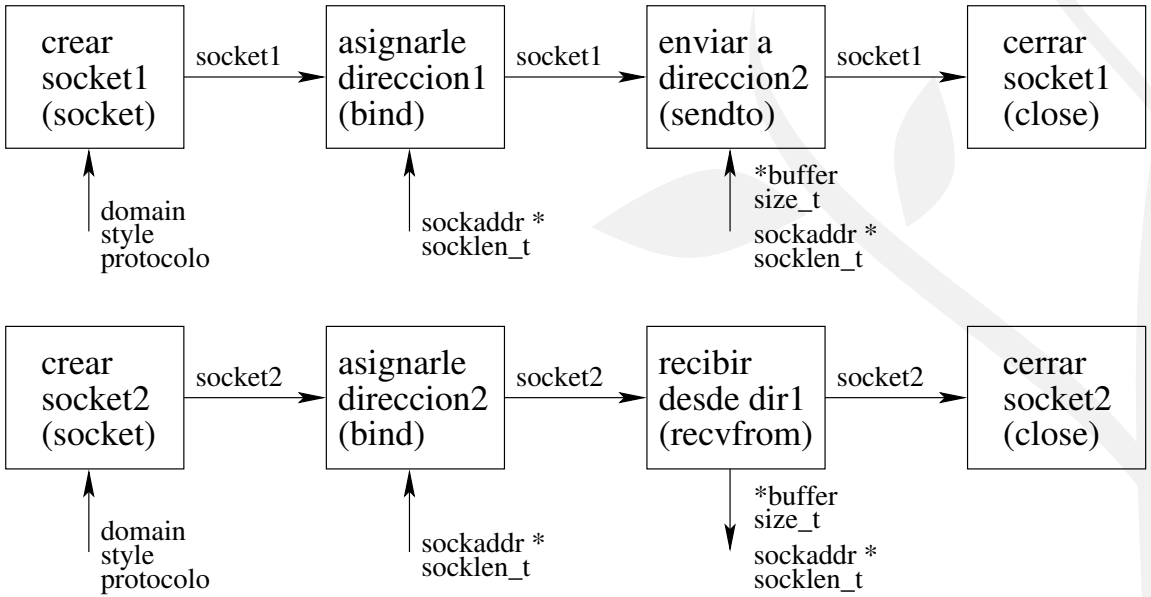
- Todo igual que en C pero mais fácil
- Non hai estruturas nin declaración de variabeis
- Usar as funcións de baixo nivel explicadas, non as de máis alto nivel

Índice

Sockets IPv4 sen conexión

O Programa 1 envía unha mensaxe ao Programa 2

- Programa 2, que recibe datos
  - Elexir un porto[21] no que recibirá datos (porto propio 2)
- Programa 1, que envía datos
  - Elexir un porto que usará para enviar datos (porto propio 1)
  - IP do equipo onde se executa o programa 2
  - Porto no que o programa 2 recibirá datos (porto remoto 1, que coincidirá co porto propio 2)



[21] Porto maior que `IPPORT_USERRESERVED`

Programas de envío/recepción UDP: C

- Os dous programas seguen o mesmo esquema
- Incluir as cabeceiras: as mesmas que no caso de TCP
- Declarar un `int` para o socket, dous `struct sockaddr_in`, unhaa para a dirección do socket propio e outra para o remoto, un `socklen_t` para o tamaño e un string para a mensaxe
- **Crear o socket** coa función `socket()`
  - `domain` AF_INET (IPv4)
  - `style` SOCK_DGRAM (sen conexión)
  - `protocol` 0 (valor por defecto)
  - Devolve o `int` que identifica o socket
  - Comprobar que se creou satisfactoriamente

Programas de envío/recepción UDP: C

- Inicializar as estruturas `struct sockaddr_in`
- Programa que envía
  - Unha coa IP e porto propios (`INADDR_ANY` e o porto elexido)
  - A outra coa IP[22] e porto remotos (a donde enviar datos)
- Programa que recibe
  - Unha coa IP e porto propios (`INADDR_ANY` e o porto elexido)
  - A outra se sobreescribirá coa IP e porto remotos, que os obterá da mensaxe que reciba[23]
- **Asignar dirección ao socket** coa función `bind()`
  - Ao socket so se lle asigna a dirección local
  - A dirección remota usase nas funcións de transmisión de datos
  - Comprobar que se creou satisfactoriamente

[22] Se se usa 127.0.0.1, os portos teñen que ser distintos

[23] O tamaño si se debe inicializar de todas formas

Programa de envío UDP: C

- **Enviar** a mensaxe coa función `sendto()`
  - Parámetros:
    - O número socket
    - A mensaXe
    - O tamaño da mensaxe
    - `flags` o valor por defecto, 0
    - A estructura coa dirección do socket destino
    - O tamaño da dirección
  - Comprobar que se enviou correctamente
  - Imprimir o **número de bytes enviados**
- **Pechar o socket** coa función `close()`

Programa de recepción UDP: C

- **Recibir** a mensaxe coa función `recvfrom()`
  - Parámetros:
    - O número socket
    - A mensaxe
    - O tamaño máximo da mensaje
    - `flags` o valor por defecto, 0
    - A estructura onde obteremos a dirección do socket destino
    - O tamaño da dirección[24]
  - Comprobar que se recibiu correctamente
  - Imprimir o **número de bytes recibidos**
  - Imprimir a **IP e o porto** de quen enviou os datos
- **Pechar os socket** coa función `close()`

[24] É un parámetro de entrada e de saída, é dicir, debe estar indicado

Programas de envío/recepción UDP: C

- **Comprobar** o seu funcionamento
  - En primeiro lugar execútase o programa que recibe
  - Noutro terminal execútase o que envía
    - Se todo é correcto, o programa que recibe imprimirá a mensaxe e terminará
  - O programa que envía pódese comprobar
    - Nun terminal execútase `nc -lku portorecepción`
    - Noutro terminal o programa que envía
    - No primero terminal sairá a mensaxe enviada
  - O programa que recibe podese comprobar
    - Nun terminal execútase o programa que recibe
    - Noutro terminal execútase `echo "mensaxe"| nc -u -q1 localhost portoenvío`
    - No primero terminal sairá a mensaxe enviada
- Facer que ambos programas poidan executarse en distintos computadores, como en TCP, usando os argumentos do `main()` para introducir as direccións e portos necesarios

Programas de envío/recepción UDP: Python

- Todo igual que en C pero mais fácil
- Non hai estruturas nin declaración de variabeis
- Usar as funcións de baixo nivel explicadas, non as de máis alto nivel