Técnicas de programación de software multimedia
(parte 2)

Programar señales

- **UNIX permite activar la ejecución asíncrona de código ante la ocurrencia de eventos especiales que se llaman *señales* (=cuando se activa la señal, el proceso deja lo que estaba para ejecutar otra función que previamente se asoció a la señal)**
- **`SIGINT` es una señal que se activa cuando se pulsa `CTRL-C`**
  - **Código para conectar la ocurrencia de la señal SIGINT con la ejecución de una función llamada gestorSenal**

```
#include <signal.h>
struct sigaction infoSenal;
```

```
void gestorSenal (int senalCausante)
{
  /* código que se activa con la señal.
    En senalCausante la función RECIBE del SO el número de la señal que generó la
    activación- es decir, el número 'SIGINT' (interesante si se usa la misma
    función para tratar varias señales – que no es nuestro caso) */
};
```

```
infoSenal.sa_handler = gestorSenal;
sigemptyset (&infoSenal.sa_mask); /* no hace falta manipular la mascara .sa_mask
   si no se desea variar las senales bloqueadas*/
infoSenal.sa_flags = 0;
CALL (sigaction (SIGINT, &infoSenal, NULL), "Error al instalar senal (SIGINT)");
```

Leer el instante de tiempo actual

- Lectura del tiempo

```
#include <sys/time.h>
#include <unistd.h>

struct timeval tiempo;
long segundos, microsegundos;

CALL (gettimeofday (&tiempo, NULL), "Fallo en gettimeofday");

segundos = tiempo.tv_sec; /* leer tv_sec como 'time value seconds' */
microsegundos = tiempo.tv_usec;
printf ("El tiempo medido es %ld.%6ld", tiempo.tv_sec,
   tiempo.tv_usec); /* ld = long int printed as decimal value */
```

  - El tiempo actual es UNA variable (`struct`) que contiene DOS campos (los dos hay que considerarlos!)

- Las restas de tiempo hay que hacerlas con cuidado: función

```
void timersub(struct timeval *a, struct timeval *b,struct timeval *res);
```

Network Byte Order

- Es bien conocido que existen distintos convenios sobre **cómo ordenar el almacenamiento de datos compuestos** (ej. un entero de 4 bytes) en memoria
  - ***Big-endian*** (byte más significativo delante) y ***little-endian*** (byte menos significativo delante)
  - Ej: almacenar 256 como un entero sin signo de 16 bits en posiciones de memoria [0] y [1]
    - ✓ Big-endian: [0]: 1, [1]:0
    - ✓ Little-endian: [0]: 0, [1]:1

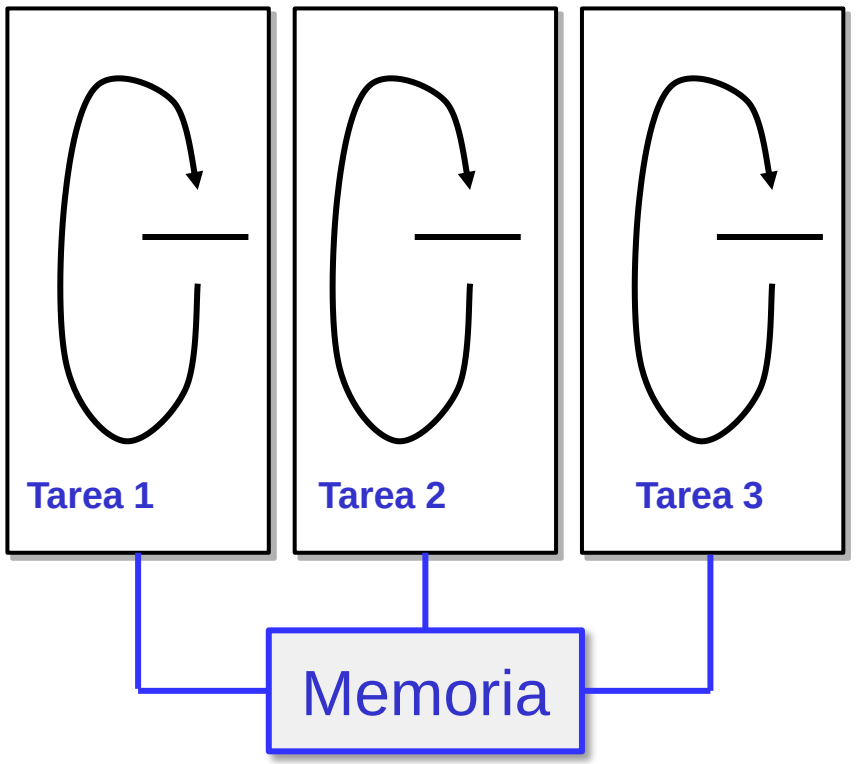
    Problema si hay que comunicar este tipo de datos entre máquinas de arquitecturas distintas
- Para el envío de datos por la red en IP, RFC791 dice
  - ✓ Enviar primero los bytes en posiciones de memoria menores
  - ✓ En cada byte, transmitir primero los bits más significativos
  - ✓ Colocar cantidades en varios octetos con el byte más significativo primero (=colocado como big-endian)
  - Este convenio se conoce como '***network byte order***'
- Si hay que transmitir estructuras de tamaño mayor que 1 byte, hay que colocarlas bien en la memoria, y luego transmitirlas
  - Es responsabilidad de la aplicación colocar bien los datos

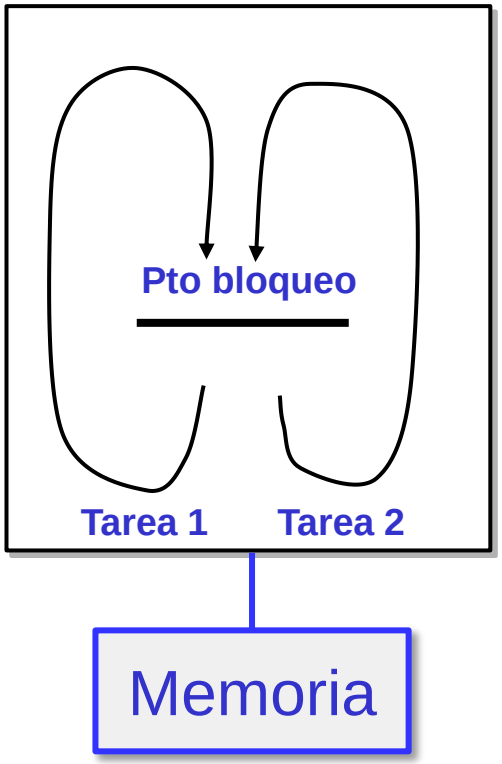
Network Byte Order

- **Un elemento de tamaño > 1 byte tiene que ser ordenado antes de mandarlo**
  - **Convertir de orden 'interno' o de 'host' a 'network byte order'**
    - ✓ `uint32_t htonl(uint32_t hostlong);`
      - ➢ **h**ost **to** **n**etwork **l**ong: Recibe un valor de 32 bits con orden 'host' y lo devuelve con 'network byte order'
    - ✓ `uint16_t htons(uint16_t hostshort);`
  - **Convertir de 'network byte order' a orden de 'host'**
    - ✓ `uint32_t ntohl(uint32_t netlong);`
    - ✓ `uint16_t ntohs(uint16_t netshort);`
- **El uso de estas funciones hace que el código que generemos sea *portable***
  - **El compilador decide cómo se colocan los bytes en función de la arquitectura del 'host'**
  - **Un código que llame a estas funciones siempre lo hace bien**

Arquitectura de una aplicación multimedia

- **Dos arquitecturas alternativas**



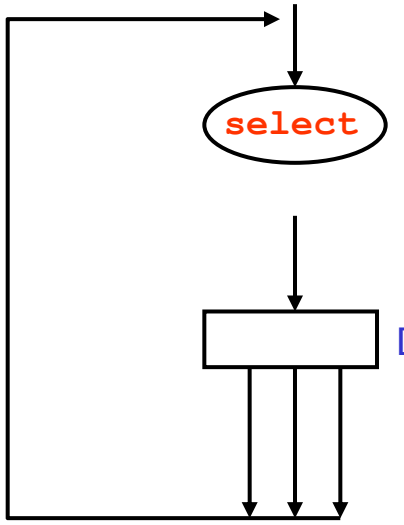


select

- Más información ejecutando `man select_tut`
- `select` se **bloquea** hasta que pasa lo primero de
  - Alguno de entre varios descriptores esté en disposición de ser leído o ser escrito,
    - ✓ El Sistema Operativo representa un descriptor como un número (es un `int`)
    - ✓ Un descriptor puede referirse a dispositivo (tarjeta de sonido) o un socket

```
descriptorTarjetaSonido = open ("/dev/dsp", O_RDWR);
```

  - Vence un temporizador asociado al `select`
- Objetivo de una arquitectura con `select`: que el proceso sólo se bloquee en el `select` (no en ningún `read` o `write` que se haga como consecuencia del `select`)
  - De esa forma la aplicación responderá inmediatamente a cualquier evento



Espera a lo primero que ocurra de que
- lleguen datos a socket 5,
- lleguen datos a la tarjeta de sonido (descriptor 8), o
- venza un temporizador de 10 segundos

Descubrir qué evento ocurrió para salir del `select`, y ejecutar cosas distintas (que no se bloqueen) en función de eso

select

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

/* definición función select */
int select(int n,
   fd_set *conjunto_lectura,   /*conjunto de descriptores
   esperando en lectura*/
   fd_set *conjunto_escritura, /*conjunto de descriptores
   esperando en escritura*/
   fd_set *conjunto_excepciones, /*conjunto de descriptores
   esperando una excepción*/
   struct timeval *timeout   /* temporizador */ );
```

Ejemplo de uso de `select`

```
/* Ej: generar conjunto de descriptores para poder esperar en lectura
   la llegada de datos en el descriptor 5 o en el 8.
   Esperamos como máximo 10 s a alguno de los eventos anteriores*/

fd_set conjunto_lectura;
struct timeval tiempo;

/* programar la espera de 10 segundos */
tiempo.tv_sec = 10;
tiempo.tv_usec = 0;

/* configurar el conjunto de lectura */
FD_ZERO(&conjunto_lectura); /* borro cualquier resto que pudiera haber
   en la variable */
FD_SET(5, &conjunto_lectura); /* añado el descriptor número 5 al
   conjunto */
FD_SET(8, &conjunto_lectura); /* añado el descriptor número 8 al
   conjunto */

CALL (res = select (9, &conjunto_lectura, NULL, NULL, &tiempo));
```

descriptor usado en el `select` con mayor número + 1

Ejemplo de uso de `select`

- Chequeo de resultados, saber qué pasó en el `select`

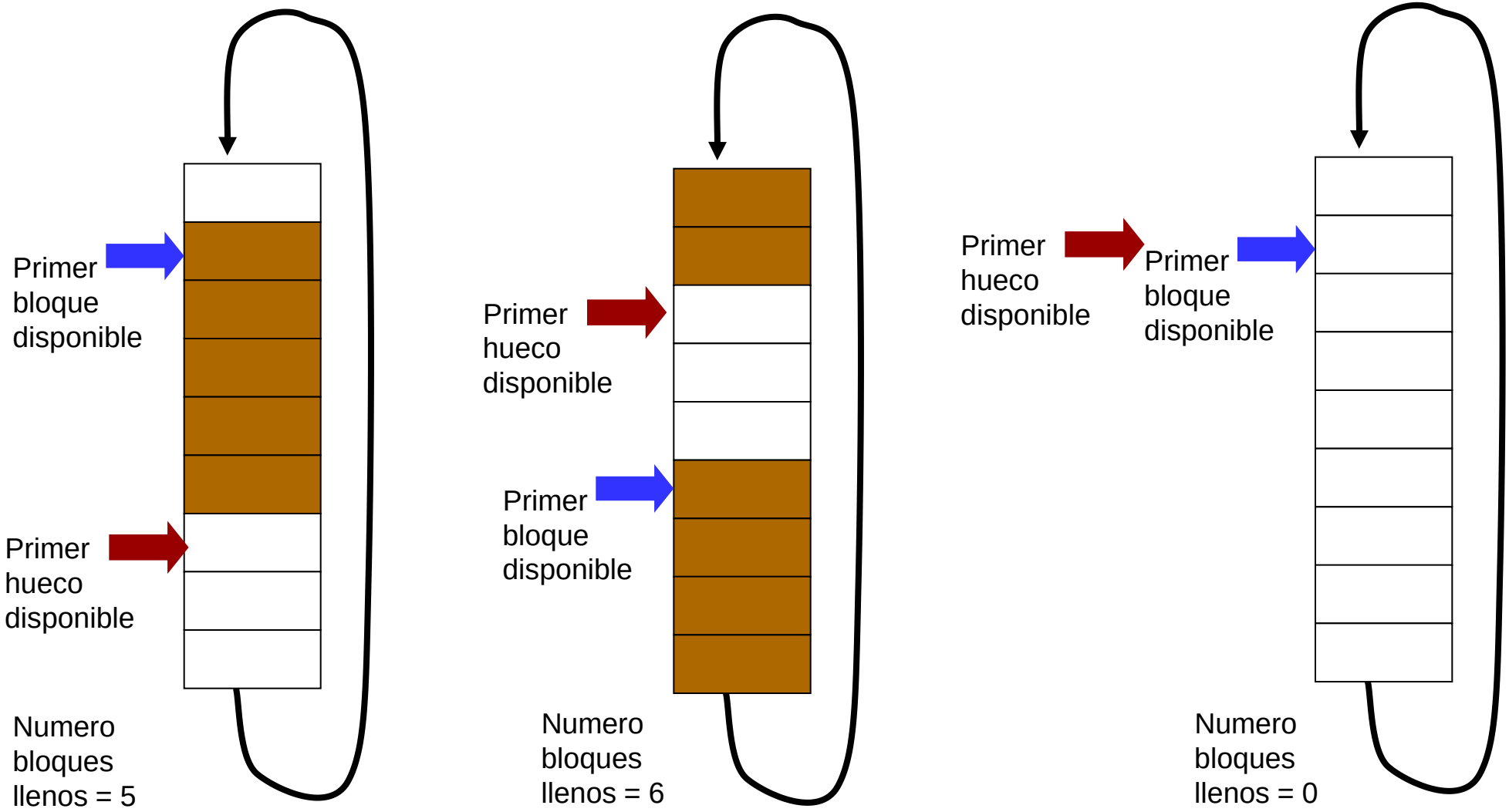
```
if (res == -1) { /* error - gestionarlo */
else if (res == 0) { /* han pasado los 10 segundos y ha
   vencido el timer */
else { /* en res queda el número de eventos exitosos*/
   if ( FD_ISSET (5, &conjunto_lectura) == 1)
   { /* han llegado datos en el descriptor 5 */       }
   if ( FD_ISSET (8, &conjunto_lectura) == 1)
   { /* han llegado datos en el descriptor 8 */       }
```

- Ojo: hay que comprobar TODOS los casos programados, porque al salir del `select` pueden haberse cumplido VARIOS
- Si se vuelve a llamar a `select`, hay que volver a poner a cero (`FD_ZERO`...) conjuntos (`conjunto_lectura`, etc.), añadir elementos a cada conjunto y actualizar timer

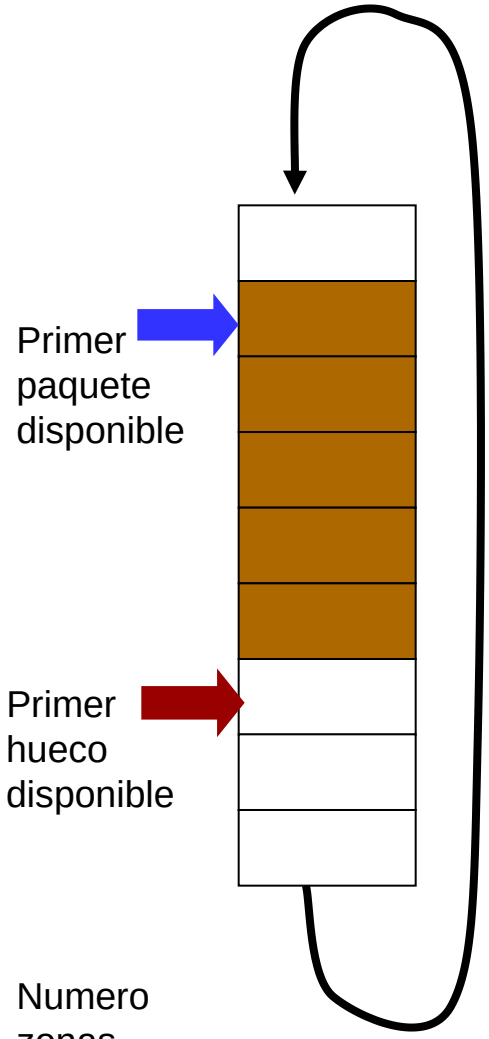
¿Cuándo se desbloquea un `select`?

- **Muy intuitivo si el `select` se bloquea en lectura en un socket**
  - Retorna cuando el socket esté dispuesto para entregar datos
    - ✓ Ha llegado un paquete UDP
    - ✓ Ha llegado información TCP, y el nivel TCP la quiere pasar a los niveles superiores
- **… pero ¿qué tiene que ocurrir para despertar a un `select` en lectura a la tarjeta de sonido?**
  - Nota: al llamar al `select`, ¡el sistema operativo no sabe cuántos datos queremos! (eso se sabe en el `read`)
  - Respuesta: `select` se despierta cuando la tarjeta ha llenado 1 fragmento
  - Recordatorio: queremos que la aplicación SOLO se bloquee en el `select`, no en el `read`, por lo que deberemos ajustar el tamaño del fragmento a los datos que pedimos en el `read` !
    - ✓ Si el `read` ejecutado después de un `select` pide dos fragmentos, se bloquea, porque hay 1 fragmento
- **Escritura en la tarjeta de sonido con `select`**
  - Si el `select` se bloquea porque la tarjeta está llena de datos para ser escritos, se desbloquea cuando hay libre 1 fragmento
- **El temporizador del `select` tiene también una granularidad limitada**
  - Depende del reloj del sistema operativo, no de un reloj hardware
  - Ojo: es un error configurar el *timer* con valores de tiempo NEGATIVOS

Buffer circular



Buffer circular: código ofrecido



El código ofrecido sólo se puede usar con un proceso y gestión de eventos con `select` (no con varios procesos o hilos de ejecución, ya que puede tener problemas de concurrencia)

```
/* devuelve puntero a zona de memoria que identifica el buffer
   circular. La función CREA la memoria. blockSize es el número
   de bytes que caben en un bloque */
buffer = void * createCircularBuffer (int
   numberOfBlocks, int blockSize);

/* devuelve puntero al primer hueco libre, o NULL si no hay huecos
   libres. Utilizado para 'meter datos' en el buffer. Su código debe
   tratar el caso en el que no haya huecos libres*/
void * pointerToInsertData (void * buffer);

/* devuelve puntero al primer bloque de datos disponible para ser
   procesado, o NULL si no hay bloques libres. Utilizado para
   'sacar datos' del buffer. Su código debe tratar el caso en el que
   no haya paquetes disponibles*/
void * pointerToReadData (void * buffer);

/* ejecutarlo antes de que termine el proceso */
void destroyBuffer (void * buffer);
```

Comunicaciones

- Programamos con sockets (herramienta estándar de comunicaciones con Linux/Unix…)
- Recordar requisito de RFC 4961: que el puerto de origen y destino de los paquetes UDP sea el mismo (para RTP y RTCP)
- Una buena referencia para repasar es “Internetworking with TCP/IP, Vol 1”, Douglas Comer

Repaso de sockets

- **Crear socket – consideramos sólo sockets para comunicación TCP – UDP / IP**

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

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

- **Ejemplo de uso**

```
sock = socket (PF_INET, /* Protocol Family Internet,
            socket de 'Internet';
            constante definida en socket.h */
       SOCK_DGRAM, /* SOCK_DGRAM ~ UDP; SOCK_STREAM ~ TCP */
       0 /* sólo hay un protocolo DGRAM en Internet */ );
```

Definiendo dirección y puerto: `struct sockaddr_in`

- **`struct sockaddr_in`: estructura que contiene un elemento <dirección IP, puerto>.**
  - Se dice que define un "address", pero es en este caso se refiere a una combinación '**dirección IP + puerto**'

```
struct sockaddr_in var_sockaddr_in;  /* declaro variable */
```

  - Definición (`netinet/in.h`): contiene al menos

✓ `sa_family_t`	`sin_family`	AF_INET (Address Family Internet)
✓ `in_port_t`	`sin_port`	Port number.
✓ `struct in_addr`	`sin_addr`	IP address.

  - Se utiliza en `bind`, `sendto`, `recvfrom` ...
  - Para rellenar valores, accedo a los campos `sin_port` y `sin_addr` de la estructura
    - `var_sockaddr_in.sin_family = AF_INET;`
    - `var_sockaddr_in.sin_port = htons(PUERTO);`
    - `var_sockaddr_in.sin_addr = htons(INADDR_ANY); /* = cualquier dirección */`

Rellenando struct sockaddr_in

- Si tengo una dirección IP en formato 'cadena de caracteres', ¿cómo pasarlo a una estructura `sockaddr_in`?

```
#include <arpa/inet.h>
int inet_pton(int address_family, const char *src, void *dst);
```

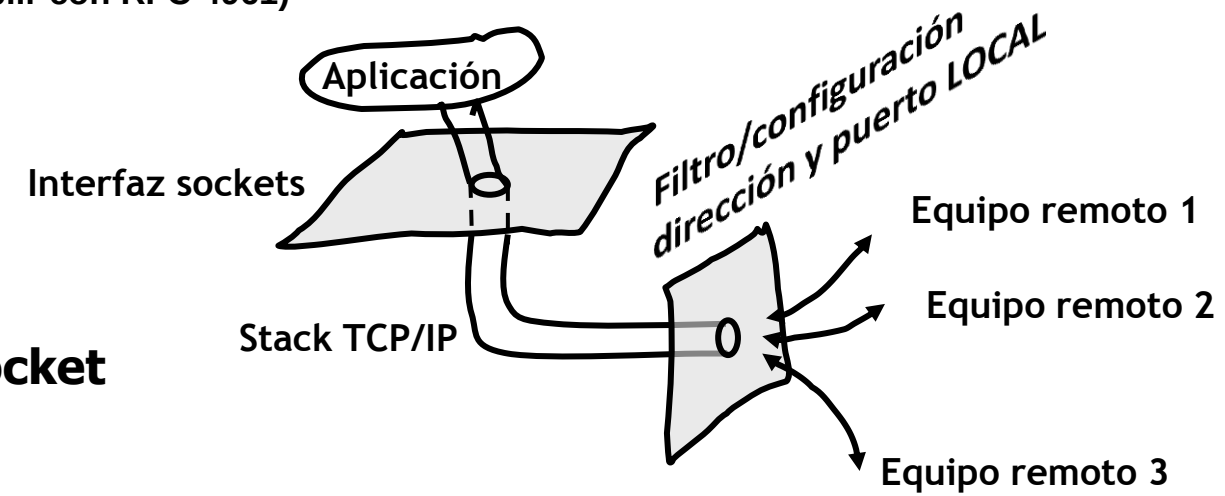
- `inet_pton` transforma una cadena de caracteres que representa una dirección IP en 'dotted notation' en un entero de 32 bits (es decir, la representación compacta de una IP) y la deja en la parte apropiada de una variable `struct sockaddr_in`
- Ejemplo
- `result = inet_pton(AF_INET, "163.117…", &var_sockaddr_in.sin_addr)`
  - `.sin_addr` es un número de 32 bits que representa una IP (¡no es la cadena de caracteres que representa cómo escribimos los humanos una dirección IP!)
  - Chequea el valor de `result` (error si la cadena de caracteres no contiene una IP, si no se utiliza la *address family* correcta (`AF_INET`)…
- Esta función NO interactúa con el DNS
  - Si desea que interactúe con DNS, utilice `getaddrinfo()` para transformar un nombre DNS en una dirección

bind

(suponemos de aquí en adelante que utilizamos UDP)

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

- `bind`: permite fijar dirección IP y puerto LOCAL para un socket
  - Afecta al filtrado de paquetes entrantes: sólo se reciben por este socket paquetes que tengan como dirección IP destino y puerto destino los especificados por el `bind`
  - Afecta a los parámetros utilizados para paquetes salientes: los paquetes enviados por este socket tienen como dirección IP origen y puerto origen los especificados por el `bind`
    - ✓ Nota: es la forma más apropiada para fijar un puerto origen (tal y como necesitamos para cumplir con RFC 4961)



Unconnected Socket

(solo `bind`)

bind

- Ej_1 de `bind`: si el argumento de tipo `sockaddr_in` fija puerto `5004` y la dirección es `INADDR_ANY`, entonces
  - El socket acepta paquetes recibidos para el puerto `5004` en cualquiera de las direcciones locales que pueda tener el host.
  - Paquetes mandados tienen como puerto origen el puerto `5004`, y escogerá una de las direcciones locales disponibles (si hubiera varias) para enviar
- Ej_2 de `bind`: si el argumento de tipo `sockaddr_in` fija puerto `5005` y la dirección es 226.7.1.2 (una dirección multicast), entonces
  - El socket acepta paquetes recibidos que han sido enviados a la dirección 226.7.1.2 y para el puerto `5005`
  - Paquetes mandados tienen como puerto origen el puerto `5005`, y escogerá una de las direcciones locales disponibles (si hubiera varias) para enviar
    - ✓ No se pueden utilizar direcciones multicast como dirección de fuente!

- Interpretación error '`bind: Address already in use`'
  - Address es aquí dirección + puerto
  - Sólo puede haber UNA aplicación escuchando de UN puerto
  - Error cuando ya hay otra aplicación escuchando en ese puerto
    - ✓ Identifique cuál

```
lsof –i :puerto
lsof –i :5004
```

sendto

DESPUES de haber hecho `bind`, para enviar datos podemos utilizar

```
ssize_t sendto(int socket_id, const void *msg, size_t
msg_len,    int flags, const struct sockaddr *dest,
socklen_t dest_len);
```

- **Uso**
  - **Poner los datos en una zona contigua de memoria, cuya dirección de comienzo hemos almacenado en `void *msg`, y tiene tamaño `msg_len`**
    - ✓ **Esa zona puede haberse creado con `malloc` (o como `char buffer[tamanho]`)**
  - **Rellenamos correctamente `dest` , que será una variable de tipo `struct sockaddr_in` para indicar dirección IP y puerto del nodo remoto al que se manda el paquete**
  - **Flags permite indicar envío de datos 'fuera de banda'… Nosotros no lo utilizamos (poner a 0)**

```
datosEnviados = sendto (socket, msg, msg_len, 0, &dest,
sizeof (struct sockaddr_in) );
```

recvfrom

DESPUES de haber hecho `bind`, para recibir datos podemos utilizar

```
ssize_t recvfrom(int socket_id, void *msg,
size_t msg_len, int flags, struct sockaddr
*desde, socklen_t *desde_len);
```

- Los datos los deja de forma contigua en la posición de memoria que empieza en `msg`
  - ✓ Esa zona puede haberse creado con `malloc` (o como `char buffer[tamanho]`)
- Al retornar la función, en `desde` queda la dirección y puerto remotos del paquete recibido (dirección y puertos de fuente)
  - ✓ No importa los valores que tenía `desde` al llamar `recvfrom`
- Pero cuidado: ANTES de llamar a `recvfrom` hay que rellenar la variable `desde_len`
  - ✓ `desde_len = sizeof (struct sockaddr_in);`
  - ✓ De esta forma, `recvfrom` sabe cuántos datos puede escribir a partir de la posición de memoria definida por `desde`

recvfrom

- Si sólo ha hecho `bind`, `recvfrom` puede recibir paquetes enviados desde CUALQUIER origen (eso sí, dirigidos al puerto que se haya especificado en el `bind`)
  - ¡No se filtran paquetes por la IP de origen ni el puerto de origen de los datos!
  - Si quiero filtrar (para escuchar sólo paquetes que vengan de IP x.y.z.k), necesito otras herramientas, como `connect`
- Nota: `sendto` y `recvfrom` no son la única forma de enviar y recibir: para sockets 'conectados' se puede utilizar `read` y `write` (y hay más formas)
  - ¡Pero para los requisitos de nuestro programa (multicast y puertos simétricos), `sendto` y `recvfrom` tienen ventajas!

Multicast

- ¡Sólo se puede utilizar con UDP!
- Para enviar no hace falta hacer nada especial (sólo utilizar una dirección de destino multicast)
- Para recibir hace falta que el host se suscriba al grupo multicast (IGMP)… El programador tiene que:
  - Hacer `bind` con variable de tipo `struct sockaddr_in` que contiene el puerto apropiado, y como dirección la del grupo multicast a suscribirse
  - Rellenar estructura `ip_mreq`

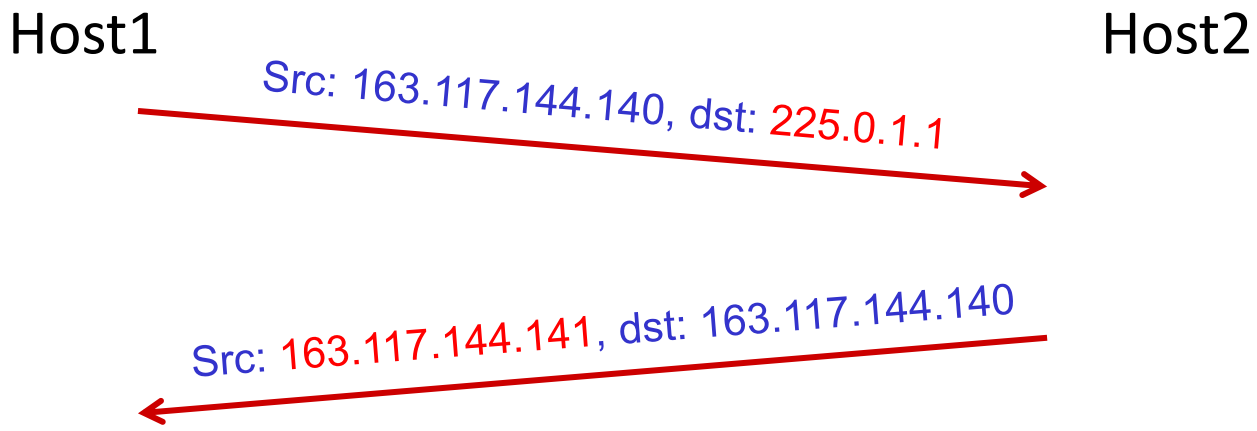
```
struct ip_mreq mreq_ipv4;
mreq_ipv4.imr_multiaddr = dirMcastIPv4 /* un struct
  in_addr; conversion desde cadena de caracteres con
  funcion  inet_pton(AF_INET, "227. …") */;
mreq_ipv4.imr_interface.s_addr = INADDR_ANY;
```

  - Llamar a `setsockopt` para el socket en el que se quiere escuchar a la dirección multicast

```
setsockopt(socket_id, IPPROTO_IP, IP_ADD_MEMBERSHIP,
  &mreq_ipv4, sizeof(mreq_ipv4))
```

Multicast

- Una dirección multicast NO puede utilizarse como dirección fuente
- Ejemplo de comunicación entre `host1` (utiliza dirección local unicast) y `host2` (utiliza dirección local multicast)



Multicast y RFC 4961

RFC4961 exige que para RTP y RTCP el puerto origen y destino debe ser el mismo en los paquetes UDP

UNA forma de comunicar dos nodos, respetando RFC 4961 (hay más), de forma que host1 manda un paquete a host2, que escucha en dirección multicast, y host2 responde a host1. Utilizan PORT para origen y destino

host1 (escucha en unicast)

- `bind` a INADDR_ANY y PORT (fija puerto local),
- Configura variable **remote** de tipo `struct sockaddr_in` con IPmcast y PORT
- Envia datos: `sendto` con dirección destino **remote**

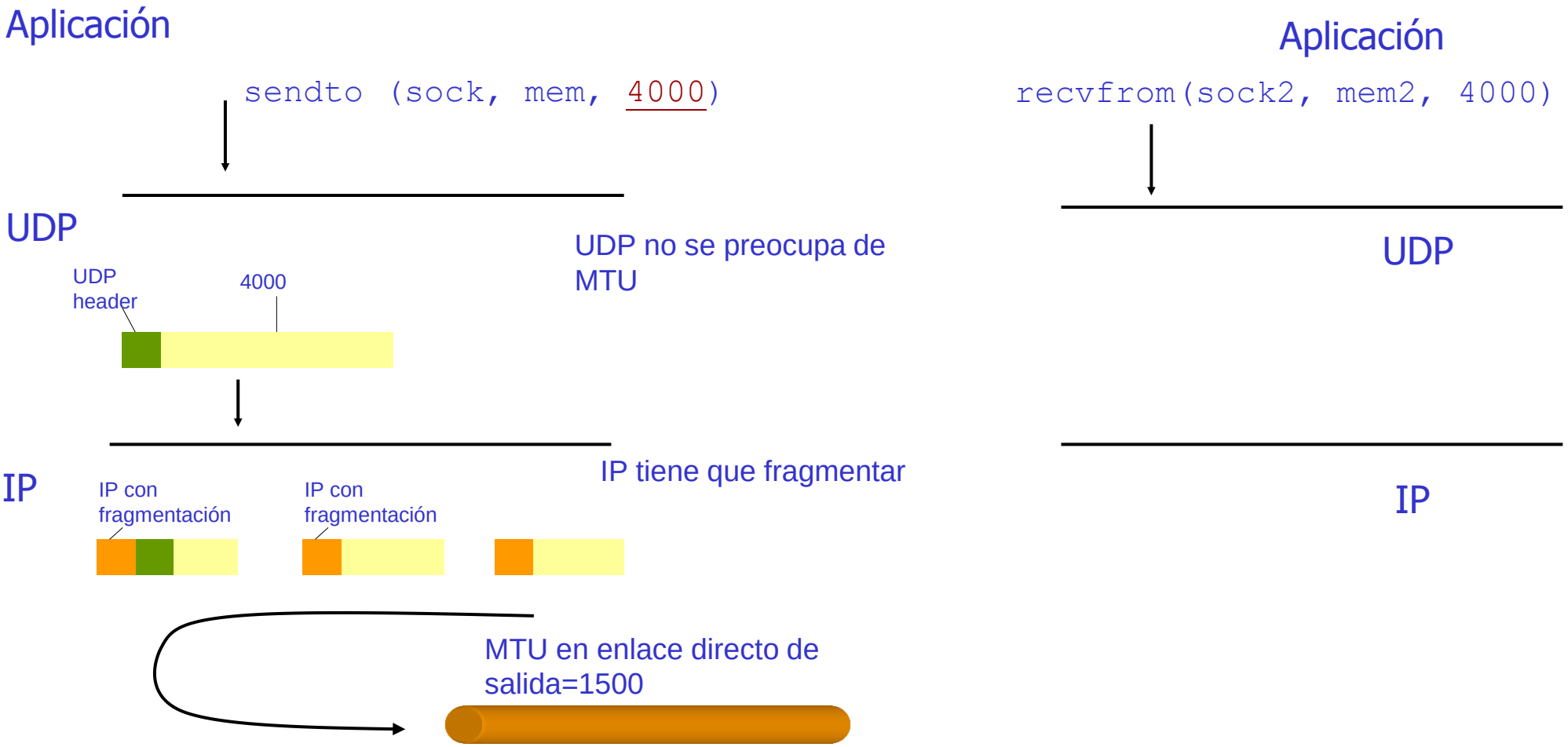
- Recibir datos: `recvfrom`
  - En variable de tipo `struct sockaddr_in` recibe IP unicast de host2 (no IPmcast!) y PORT

host2 (escucha en dirección multicast IPmcast)

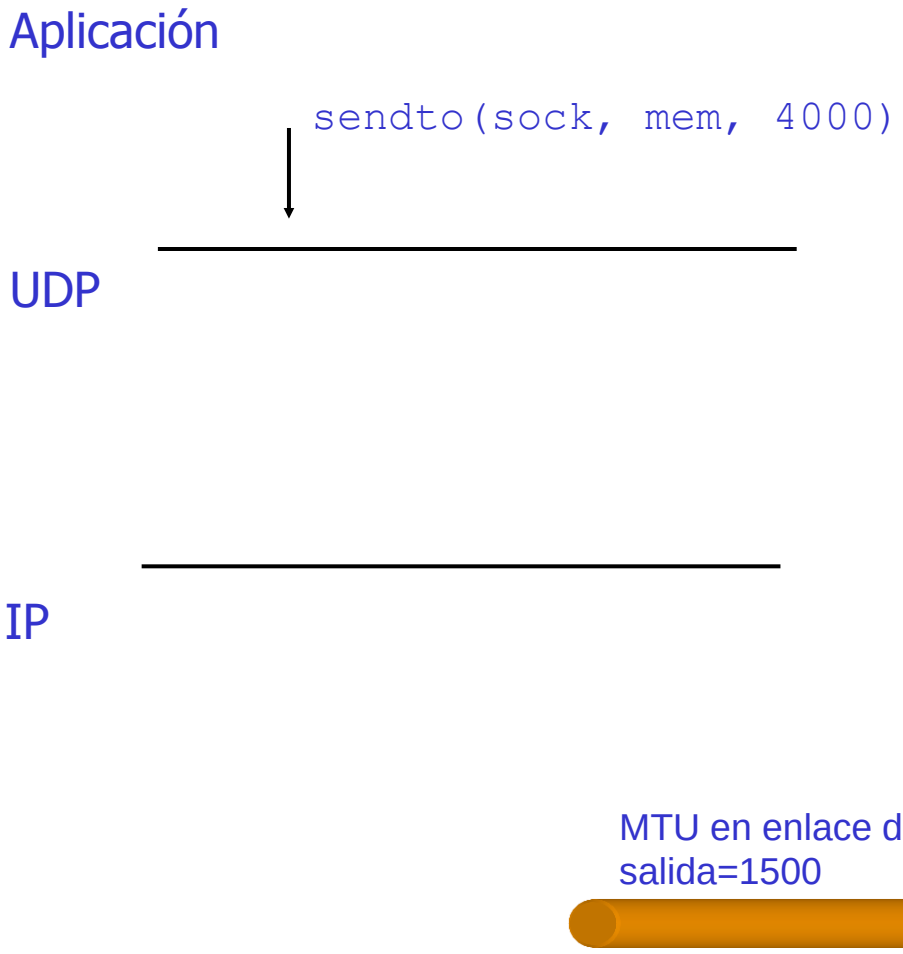
- `bind` a IPmcast y PORT
- `setsockopt` para suscribirse a dirección mcast IPmcast

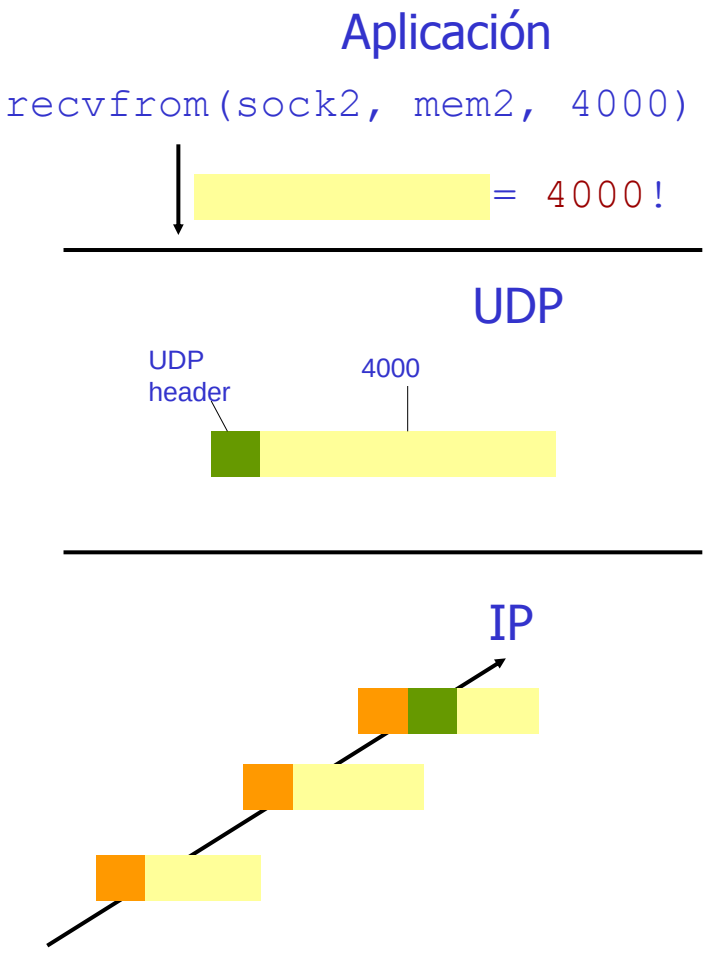
- Recibe datos con `recvfrom`
  - En variable de tipo `struct sockaddr_in` recibe IP unicast de host1 y PORT
- Envía datos con `sendto` utilizando la variable de tipo `struct sockaddr_in` que rellenó `recvfrom`

Delimitación de datos en UDP



Delimitación de datos en UDP





MTU en enlace directo de salida=1500

Orden sugerido para programar `conf`

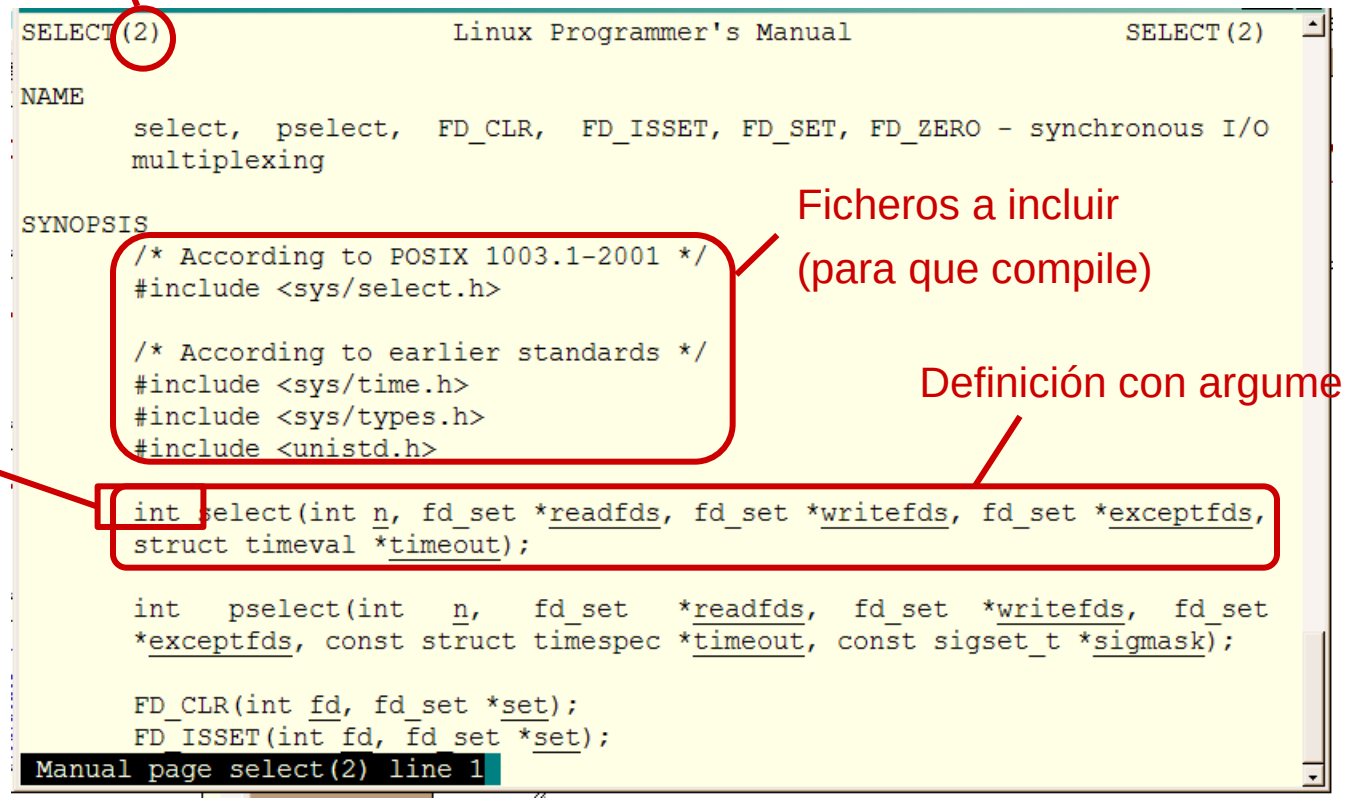
1. **Utilice la función de captura de argumentos**
   - Esta función YA ESTÁ implementada (no la tiene que hacer usted), vea `confArgs.h`
   - Tendrá información sobre si es `first` o `second`, calidad del audio, tamaño de paquetes y de buffer…
     - Imprima los valores con la función `printValues`
2. **Considere los casos de `first` y `second`**
   - Empiece programando `second`
     - Configure comunicaciones
     - Configure tarjeta de sonido
     - Grabe un paquete
     - Ponga una cabecera RTP básica (sólo payload)
   - Programe `first`
     - Configura comunicaciones, tanto unicast como multicast
     - Recibe paquete, aprende tamaño paquete, aprende payload
     - Configura tarjeta de sonido

Orden sugerido para programar `conf`

3. **Implemente envío/recepción de datos**
   - Posiblemente esta parte puede ser igual para `first` y `second`
   - Cree estructuras de datos (buffer circular…)
   - Programe bucle que incluye el `select` para
     - Recibir paquetes de la red y escribirlos en el buffer circular
     - Coger paquetes del buffer y escribirlos en la tarjeta de sonido (quizás esta tarea empiece más tarde, en función del tiempo de buffering)
     - Recibir datos de la tarjeta de sonido y mandarlos al destino
   - Complete procesado de RTP
     - Paquetes perdidos, …

PROGRAMANDO Y DEPURANDO

Comando `man`



- **Nos interesan secciones 2 (llamadas al SO) y 3 (funciones)**
  - **`man 2 open`**
    - ✓ **si hacemo `man open` nos informa sobre el 'comando' open (= línea de comandos), no sobre la llamada al sistema operativo open)**
- **Describe toda la información necesaria para programar con una función o una llamada al SO**

Comando `man`

- **Detalla:**
  - **Tipos de los argumentos, valores iniciales razonables para dichos argumentos, y valores retornados en los argumentos**
  - **Significado de los valores retornados (sección RETURN VALUE)**
    - **Si hay errores, se indica en el return value**
      - Cada error tiene un código, y `man` indica posibles causas (sección ERRORS)

Depurando

- **Utilidad `strace`**
  - **Le permite ver la secuencia de llamadas al sistema operativo, los parámetros, si tienen éxito o fallan, y una indicación de por qué**
    - ✓ **Por ejemplo, útil al depurar `recvfrom`, `sendto`, ...**

  `-o fichero` -> vuelca información a un fichero

  `-tt` -> muestra el instante de tiempo en el que se ejecuta una operación

    - ✓ **Útil para depurar temporizaciones**

Ejemplo de salida de `strace`

```
doc010:~/larc> strace ./host1
mprotect(0xb7f96000, 4096, PROT_READ)   = 0
munmap(0xb7f9c000, 147212)              = 0
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(5000), sin_addr=inet_addr("0.0.0.0")},
16) = 0
sendto(3, "Sent from host1\0"..., 16, 0, {sa_family=AF_INET, sin_port=htons(5000),
sin_addr=inet_addr("225.0.1.1")}, 16) = 16
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 12), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7fbf000
write(1, "Host1: Using sendto to send data "..., 58Host1: Using sendto to send data
to multicast destination
) = 58
recvfrom(3, "Sent from host2\0"..., 256, 0, {sa_family=AF_INET,
sin_port=htons(5000), sin_addr=inet_addr("163.117.144.141")}, [16]) = 16
```

Nombre de la llamada

Valores de los argumentos. En este caso son constantes bien conocidas (conocidas también por strace)

Resultado de la llamada (ver `man 3 socket`)

Ejemplo de salida de `strace`

```
doc010:~/larc> strace ./host1

mprotect(0xb7f96000, 4096, PROT_READ)   = 0
munmap(0xb7f9c000, 147212)              = 0
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(5000),
sin_addr=inet_addr("0.0.0.0")}, 16) = 0
sendto(3, "Sent from host1\0"..., 16, 0, {sa_family=AF_INET,
sin_port=htons(5000), sin_addr=inet_addr("225.0.1.1")}, 16) = 16
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 12), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7fbf000
write(1, "Host1: Using sendto to send data "..., 58Host1: Using sendto
to send data to multicast destination
) = 58
recvfrom(3, "Sent from host2\0"..., 256, 0, {sa_family=AF_INET,
sin_port=htons(5000), sin_addr=inet_addr("163.117.144.141")}, [16]) =
16
```

Socket UDP abierto

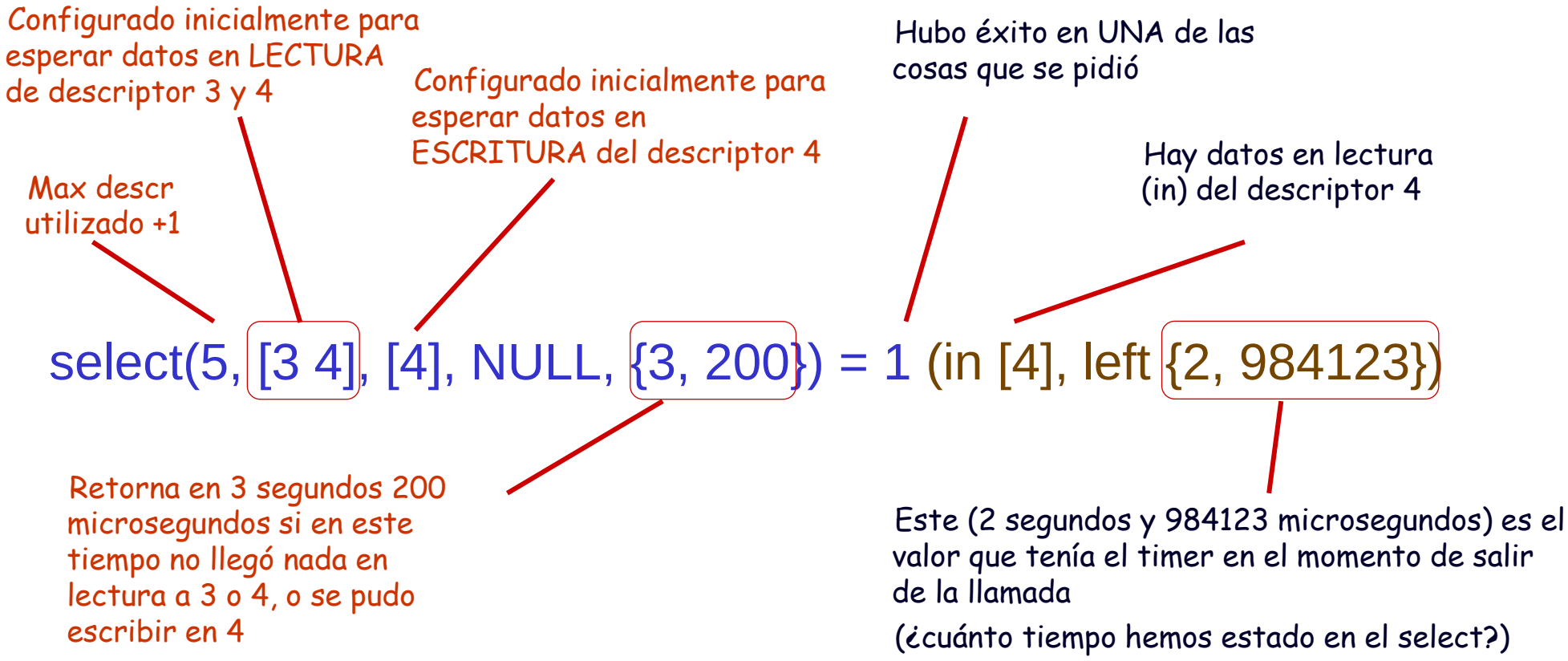
bind

Envía a dirección multicast y puerto destino 5000

Recibe de puerto remoto 5000 y dirección remota …141

Ejemplo de salida de `strace`

- **Interpretando llamada a `select` en `strace`**
  - **Parte de la izquierda son loa valores de los argumentos ANTES de la llamada; parte de la derecha, valores DESPUÉS de la llamada**



Buscando una función (o una declaración de una estructura…)

- Quiero saber en qué fichero (de entre muchos que tengo en un directorio) se ha declarado una función llamada `defaultValue`

```
doc003:~/larc> grep defaultV *.h
audioSimpleArgs.h:void defaultValues (struct structSndQuality
   *datosDsp, /* structure defined in audioConfig.h, contains
   format, channels and sampling frequency */
```

- Restrinja a `*.h` para declaraciones y `*.c` para código o llamadas.

Depurando

- **Escriba poco código, compile y pruebe**
- **Repita ciclo de 'escribir -> compilar -> probar' muchas veces**
  - **Si escribe todo el código de un golpe, le será difícil saber dónde está el fallo, no se acordará de por qué escribió ese trozo de código, ...**
- **Herramientas**
  - **Trazado manual (`printf`), permite saber**
    - ✓ **Por dónde pasó su código**
    - ✓ **Valores de variables importantes para la ejecución de su código**
      - En general, para este proyecto NO necesita depuradores de tipo `gdb`
        - Aunque puede utilizarlos si lo desea y los domina
  - **`strace` permite conocer qué pasó con las llamadas al sistema operativo**
    - ✓ **También podría obtenerlo mediante trazado manual, pero `strace` es MUCHO más conveniente**
- **Información**
  - **Documentación de la asignatura!!**
  - **Comando `man`**
  - **Buscar en Internet (stackoverflow, etc.) – sólo si se da información suficientemente precisa sobre lo que se busca, típicamente en inglés**

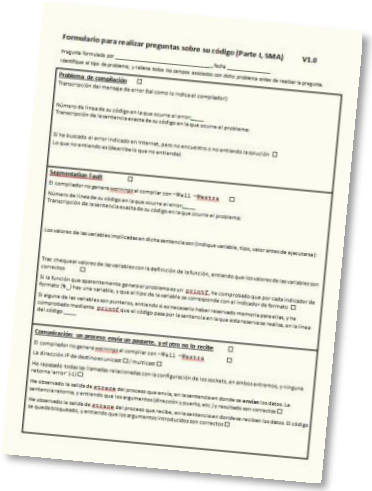
Depuración con `printf`

- El texto incluido en un `printf` **NO** se asegura que se imprima en el momento en que se invoca la función
  - Por motivos de eficiencia, el S.O. puede decidir guardarlo para acumular más datos, e imprimirlo después
  - Si utilizamos `printf`, y en alguna línea posterior el código detiene su ejecución (ej, por un `Segmentation Fault`), no podemos estar seguros de que el texto del `printf` se imprimió
- Para asegurarse de que `printf` imprime antes de ejecutar la siguiente instrucción:

```
printf ('Pasa por aquí\n'); fflush (stdout);
```

Preguntando al profesor cuando tengo problemas

- A partir del 20 de octubre NO se responderán preguntas MAL enunciadas sobre la parte I
- Ejemplo de preguntas MAL enunciadas:
  - No compila
  - No funciona
  - Me sale segmentation fault
  - *First* no manda paquetes a *second*
  - El timer expira y no sé por qué


Formulario para realizar preguntas sobre su código (Parte I, SMA) V1.0

- Para facilitar que las preguntas sobre su código estén bien formuladas, tendrá que rellenar formulario
  - Disponible en Aula Global (el profesor tendrá copias)
- Preguntas GENERALES: sobre los enunciados, sobre la estructura de la práctica, sobre la semántica de funciones de Linux… se hacen SIN formulario

Preguntas BIEN enunciadas

- **Sobre 'segmentation faults'**
  - **Prerrequisitos:**
    - ✓ **Asegúrese de que el compilador (opciones `-Wall -Wextra`) NO genera warnings, o si genera alguno, se entiende perfectamente lo que significa**
      - En general, los warnings son autoexplicativos, pero si tiene dudas, copie una cantidad suficiente del texto generado, y busque en Internet
    - ✓ **Identifique la sentencia C exacta en la que ocurre el segmentation fault con `printf(…); fflush(stdout);`**
      - Si la sentencia utiliza punteros, asegúrese de que la memoria a la que apunta estaba inicializada correctamente
    - ✓ **Identifique los valores de las variables implicados en la sentencia C (valor del puntero…), imprimiéndolo ANTES de la ejecución del código con el problema con `printf`…**
    - ✓ **Compruebe con la declaración de la función (`man función` o similar) que los valores son correctos**
      - Si las variables implicadas son punteros, y requieren haber reservado memoria con anterioridad, he comprobado que se reserva memoria correctamente

Pregunta BIEN enunciada: "Al ejecutar la sentencia X con los valores V1, V2 y V3 en los argumentos A1, A2, A3, ocurre un segmentation fault, a pesar de que entiendo que los valores son correctos para dicha sentencia"