

INFORME PRÁCTICA 3

Ejercicio 1.c:

En primer lugar definimos el tamaño máximo que va a recibir el receptor como 5

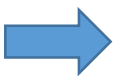
```
#define MAX 5
```

En el emisor mandamos el siguiente mensaje de 26 bytes:

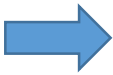
```
char mensaje[] = "ABCDEFGHJKLMNOPQRSTUVWXYZ";
```

```
if((bytes_enviados=sendto(sock,mensaje,strlen(mensaje),0,(struct sockaddr *) &dir_receptor,tamano))==-1){  
    perror("\nError al enviar el mensaje\n");  
    exit(EXIT_FAILURE);  
}  
  
printf("\nNúmero de bytes enviados: %d\n",bytes_enviados);
```

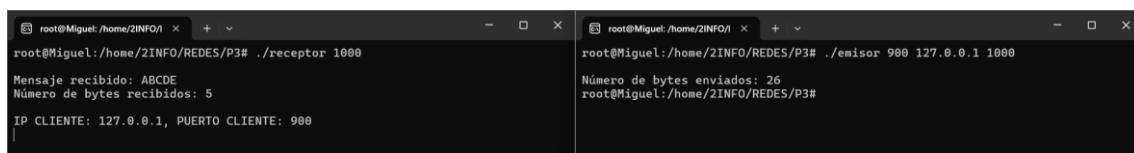
Finalmente en el emisor añadimos dos funciones recvfrom



```
if((bytes_recibidos=recvfrom(sock,mensaje,MAX,0,(struct sockaddr *)&dir_emisor,&tamano))==-1){  
    perror("\nError al recibir el mensaje\n"); // si hay algún error lo notificamos  
    exit(EXIT_FAILURE);  
}  
  
mensaje[bytes_recibidos] = '\0'; // colocamos un salto de línea al final de nuestro mensaje recibido  
  
printf("\nMensaje recibido: %s",mensaje); // imprimimos el mensaje recibido  
printf("\nNúmero de bytes recibidos: %d\n",bytes_recibidos); // imprimimos el número de bytes recibidos  
  
char ip_cli_dest[INET_ADDRSTRLEN]; // si se ha podido crear, creamos una variable para imprimir la IP  
inet_ntop(AF_INET, &(dir_emisor.sin_addr),ip_cli_dest,INET_ADDRSTRLEN); // guardamos la IP en texto  
printf("\nIP CLIENTE: %s, PUERTO CLIENTE: %d\n",ip_cli_dest,ntohs(dir_emisor.sin_port)); // imprimimos la IP y el puerto  
  
// SEGUNDO RECVFROM (NECESARIO PARA EL APARTADO 1.C)  
  
if((bytes_recibidos=recvfrom(sock,mensaje,MAX,0,(struct sockaddr *)&dir_emisor,&tamano))==-1){  
    perror("\nError al recibir el mensaje\n");  
    exit(EXIT_FAILURE);  
}  
  
mensaje[bytes_recibidos] = '\0'; // colocamos un salto de línea al final de nuestro mensaje recibido  
  
printf("\nMensaje recibido: %s",mensaje); // imprimimos el mensaje recibido  
printf("\nNúmero de bytes recibidos: %d\n",bytes_recibidos); // imprimimos el número de bytes recibidos  
  
inet_ntop(AF_INET, &(dir_emisor.sin_addr),ip_cli_dest,INET_ADDRSTRLEN); // guardamos la IP en texto  
printf("\nIP CLIENTE: %s, PUERTO CLIENTE: %d\n",ip_cli_dest,ntohs(dir_emisor.sin_port)); // imprimimos la IP y el puerto
```



Tras ejecutar el código, obtenemos lo siguiente por consola



```
root@Miguel:/home/2INFO/REDES/P3# ./receptor 1000  
Mensaje recibido: ABCDE  
Número de bytes recibidos: 5  
IP CLIENTE: 127.0.0.1, PUERTO CLIENTE: 9000  
  
root@Miguel:/home/2INFO/REDES/P3# ./emisor 900 127.0.0.1 1000  
Número de bytes enviados: 26  
root@Miguel:/home/2INFO/REDES/P3#
```

Como podemos observar, el receptor no es capaz de recibir todo el mensaje que manda el emisor (26 bytes), sino que solo recibe los 5 primeros bytes (el primer recvfrom). A parte de esto, nunca termina su ejecución ya que se queda en el segundo recvfrom esperando otro mensaje.

Esto sucede ya que estamos creando un transporte UDP sin conexión, el cual no ofrece una entrega fiable.

A diferencia de TCP, que mantiene información acerca de los buffers de recepción y envío, y por lo tanto es capaz de acceder a dicha información con una segunda función `recv`, UDP no mantiene información acerca de estos buffers, por lo que una segunda función `recvfrom` esperaría un nuevo mensaje.

Ejercicio 1.d:

En nuestros programas añadimos las siguientes modificaciones para que se transmita un array de números tipo float en vez de una cadena de texto.

En el emisor:

-Definimos nuestra nueva variable mensaje como un array de floats.

```
float mensaje[] = {1.2, 2.3, 3.4, 4.5};
```

En el receptor:

-Redefinimos la variable mensaje como un array de floats. Con esto C almacena correctamente (en orden) los datos en el array, por lo que nosotros solo tenemos que controlar el número exacto de floats recibidos.

```
float mensaje[MAX];
```

-Calculamos el número de floats que se reciben. Conociendo que un float en C ocupa 4 bytes, utilizamos los bytes recibidos que nos devuelve la función `recv`, y lo dividimos por el factor de 4 bytes para obtener el número de floats recibidos. Este resultado se guarda en la variable `nfloats`.

```
nfloats=nbytes/4;
```

- Con un bucle `for` imprimimos sucesivamente los floats recibidos (con la variable `nfloats` que calculamos arriba).

```
for(int i=0;i<nfloats;i++){  
    printf("Número recibido: %f\n",mensaje[i]);  
}
```

Con esto obtenemos el siguiente resultado.

-En el emisor:

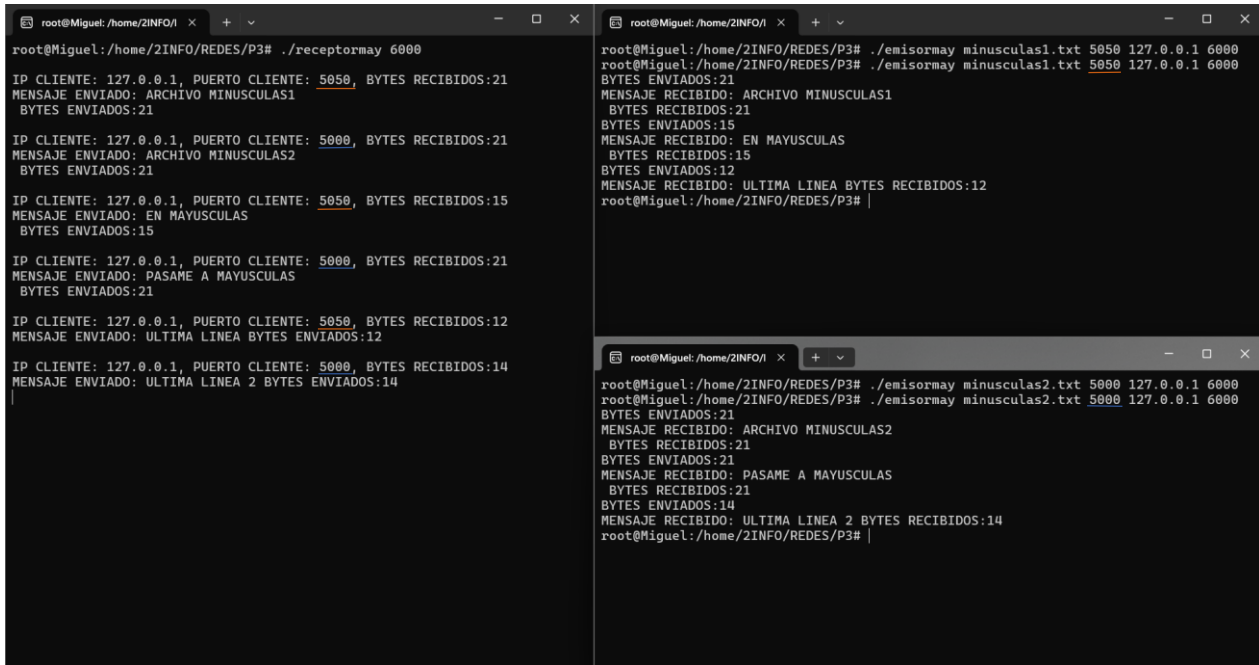
```
isabel@ISA-PORTATIL:~/c/segundo/Redes/practica3/Ej1$ ./emisor 5000 127.0.0.1 6000  
Número de bytes enviados 16
```

-En el receptor:

```
isabel@ISA-PORTATIL:~/c/segundo/Redes/practica3/Ej1$ ./receptor 6000  
Número de bytes recibidos: 16  
Número recibido: 1.200000  
Número recibido: 2.300000  
Número recibido: 3.400000  
Número recibido: 4.500000  
Ip del emisor: 127.0.0.1  
Puerto del emisor: 5000
```

Ejercicio 3:

Al introducir un sleep (de un segundo) en el lazo del emisor, donde se van leyendo las líneas del archivo, nos da tiempo de lanzar un segundo emisor que se conecte al mismo receptor, de esta forma, nuestro receptor es capaz de atender a dos emisores simultáneamente. A continuación podemos verlo:



```
root@Miguel:/home/2INFO/1 x + v - □ ×
root@Miguel:/home/2INFO/REDES/P3# ./receptormay 6000
IP CLIENTE: 127.0.0.1, PUERTO CLIENTE: 5050, BYTES RECIBIDOS:21
MENSAJE ENVIADO: ARCHIVO MINUSCULAS1
BYTES ENVIADOS:21
IP CLIENTE: 127.0.0.1, PUERTO CLIENTE: 5000, BYTES RECIBIDOS:21
MENSAJE ENVIADO: ARCHIVO MINUSCULAS2
BYTES ENVIADOS:21
IP CLIENTE: 127.0.0.1, PUERTO CLIENTE: 5050, BYTES RECIBIDOS:15
MENSAJE ENVIADO: EN MAYUSCULAS
BYTES ENVIADOS:15
IP CLIENTE: 127.0.0.1, PUERTO CLIENTE: 5000, BYTES RECIBIDOS:21
MENSAJE ENVIADO: PASAME A MAYUSCULAS
BYTES ENVIADOS:21
IP CLIENTE: 127.0.0.1, PUERTO CLIENTE: 5050, BYTES RECIBIDOS:12
MENSAJE ENVIADO: ULTIMA LINEA BYTES ENVIADOS:12
IP CLIENTE: 127.0.0.1, PUERTO CLIENTE: 5000, BYTES RECIBIDOS:14
MENSAJE ENVIADO: ULTIMA LINEA 2 BYTES ENVIADOS:14

root@Miguel:/home/2INFO/1 x + v - □ ×
root@Miguel:/home/2INFO/REDES/P3# ./emisormay minusculas1.txt 5050 127.0.0.1 6000
root@Miguel:/home/2INFO/REDES/P3# ./emisormay minusculas1.txt 5050 127.0.0.1 6000
BYTES ENVIADOS:21
MENSAJE RECIBIDO: ARCHIVO MINUSCULAS1
BYTES RECIBIDOS:21
BYTES ENVIADOS:15
MENSAJE RECIBIDO: EN MAYUSCULAS
BYTES RECIBIDOS:15
BYTES ENVIADOS:12
MENSAJE RECIBIDO: ULTIMA LINEA BYTES RECIBIDOS:12
root@Miguel:/home/2INFO/REDES/P3# |

root@Miguel:/home/2INFO/1 x + v - □ ×
root@Miguel:/home/2INFO/REDES/P3# ./emisormay minusculas2.txt 5000 127.0.0.1 6000
root@Miguel:/home/2INFO/REDES/P3# ./emisormay minusculas2.txt 5000 127.0.0.1 6000
BYTES ENVIADOS:21
MENSAJE RECIBIDO: ARCHIVO MINUSCULAS2
BYTES RECIBIDOS:21
BYTES ENVIADOS:21
MENSAJE RECIBIDO: PASAME A MAYUSCULAS
BYTES RECIBIDOS:21
BYTES ENVIADOS:14
MENSAJE RECIBIDO: ULTIMA LINEA 2 BYTES RECIBIDOS:14
root@Miguel:/home/2INFO/REDES/P3# |
```