

**Práctica 2:**  
**Sockets orientados a conexión**  
Servidor y cliente TCP

Informe de comportamiento

Presentado por:

**Carlos Cao López**  
**Pedro Vidal Villalba**

Para la materia:

**Redes**

Santiago de Compostela

27 del 10, 2023

# Índice

1	Introducción	3
2	Ejercicio 1. Apartado c	4
3	Ejercicio 1. Apartado d	6
4	Ejercicio 3	8

## Índice de figuras

1	(1.c) – Salida por terminal sin y con <code>sleep()</code> . . . . .	4
2	(1.d) – Salida por terminal del cliente con <code>MAX_BYTES_RECV = 5</code> . . .	6
3	(1.d) – Salida por terminal del cliente con <code>MAX_BYTES_RECV = 10</code> . . .	6
4	(1.d) – Salida por terminal del cliente con <code>MAX_BYTES_RECV = 15</code> . . .	7
5	(3) – Salida al ejecutar el servidor de mayúsculas y dos clientes . . .	9

## 1. Introducción

En este informe se recogerán los distintos comportamientos asociados a los códigos de los apartados (1.c), (1.d), y (3) de la Práctica 2 de la materia de *Redes*.

En el primer ejercicio de esta práctica se han implementado sendos programas `servidor.c` y `cliente.c`. El primero de ellos se encarga de enviar un mensaje de saludo al cliente y, este debe imprimir el mensaje recibido así como los bytes que ocupa el mensaje.

En el apartado (1.c) comprobamos si es posible que el servidor envíe dos mensajes con sendas funciones `send()` y que el cliente reciba ambos mensajes con una única sentencia `recv()`.

En el apartado (1.d) probaremos a implementar un bucle `while` para la recepción de los mensajes.

En el segundo ejercicio de la práctica se han implementado también un par de programas `servidormay.c` y `clientemay.c`. En estos, el cliente lee un archivo de texto de entrada y se lo envía al servidor línea a línea a través de una conexión previamente creada. El servidor debe pasar los caracteres de cada línea a mayúsculas y devolverle al cliente la línea convertida. Por último, el cliente va recibiendo las líneas y las va escribiendo en un archivo de salida, que tendrá el mismo nombre que el archivo de entrada pero todo en mayúsculas.

En el ejercicio (3) se debe comprobar que el servidor programado en el ejercicio 2 puede atender a varios clientes secuencialmente.

## 2. Ejercicio 1. Apartado c

Como se ha comentado en la introducción, en este apartado trataremos de que el cliente reciba dos mensajes utilizando un único `recv()`. Para ello, se han introducido los siguientes cambios en el servidor.

- En `servidor.c`:

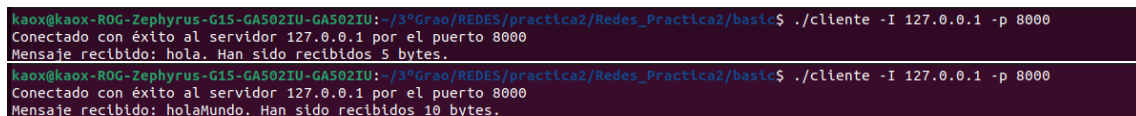
```

1  /* Codigo asociado al apartado (1.c)*/
2
3  /* Enviamos un primer mensaje*/
4  strcpy(message, "hola");
5  if ( (sent_bytes = send(client.socket, message, strlen(message)
6      + 1, 0)) < 0) {
7      perror("No se pudo enviar el mensaje");
8      exit(EXIT_FAILURE);
9  }
10 /*Enviamos un segundo mensaje*/
11 strcpy(message, "Mundo");
12 if ( (sent_bytes = send(client.socket, message, strlen(message)
13     + 1, 0)) < 0) {
14     perror("No se pudo enviar el mensaje");
15     exit(EXIT_FAILURE);
16 }
17 /* Fin del codigo asociado al apartado (1.c)*/

```

- En `cliente.c` no hay cambios.

Al ejecutar los programas con estos cambios observamos la salida de la Figura 1. Como se puede observar, el programa cliente solo recibe el primero de los mensajes pues al servidor no le da tiempo a ejecutar el segundo `send()` antes del `recv()` del cliente.



```

kaox@kaox-ROG-Zephyrus-G15-GA502IU-GA502IU:~/3ºGrao/REDES/practica2/Redes_Practica2/basic$ ./cliente -I 127.0.0.1 -p 8000
Conectado con éxito al servidor 127.0.0.1 por el puerto 8000
Mensaje recibido: hola. Han sido recibidos 5 bytes.
kaox@kaox-ROG-Zephyrus-G15-GA502IU-GA502IU:~/3ºGrao/REDES/practica2/Redes_Practica2/basic$ ./cliente -I 127.0.0.1 -p 8000
Conectado con éxito al servidor 127.0.0.1 por el puerto 8000
Mensaje recibido: holaMundo. Han sido recibidos 10 bytes.

```

**Figura 1:** (1.c) – Salida por terminal sin y con `sleep()`

Para hacer que el programa `cliente.c` reciba los dos mensajes, basta con introducir un `sleep()` en el cliente para que este espere un breve periodo de tiempo entre el establecimiento de la conexión y la recepción, permitiendo al servidor el envío de los dos mensajes. Debe apuntarse que en la parte de servidor se ha implementado un cambio en el número de bytes enviados para forzar el no envío del carácter de

terminación del primer `string`. En caso de ser enviado, aunque el cliente recibiría los dos mensajes como podríamos apreciar en el número de bytes recibidos tan solo se imprimiría el primero de ellos pues se detectaría el carácter de terminación de string antes del segundo mensaje no llegando así a imprimirse.

Se puede ver en la Figura 1 el comportamiento asociado.

- En `servidor.c`.

```
1 /* Enviamos un primer mensaje */
2 strcpy(message, "hola");
3 if ((transmited_bytes = send(client.socket, message, strlen(
   message), 0)) < 0) {
4     perror("No se pudo enviar el mensaje");
5     exit(EXIT_FAILURE);
6 }
```

- En `cliente.c`.

```
1 connect_to_server(client);
2 sleep(3); /* Código asociado al apartado (1.c) */
3 handle_data(client);
```

### 3. Ejercicio 1. Apartado d

Partiendo del anterior apartado, en este eliminaremos la espera introducida a fin de recibir los dos mensajes en el cliente por separado e introduciremos un bucle de tipo `while` como se muestra abajo. Como se puede observar en las distintas figuras, el bucle termina una vez han sido recibidos todos los bytes.

- En `servidor.c` no hay cambios.
- En `cliente.c`.

```
1 /*Codigo asociado al apartado 1(d) */
2 while((recv_bytes = recv(client.socket, server_message,
   MAX_BYTES_RECV, 0)) > 0){
3     printf("Mensaje recibido: %s. Han sido recibidos %ld bytes
       .\n", server_message, recv_bytes);
4 }
5 /*Codigo asociado al apartado (1.d) */
```

- $MAX\_BYTES\_RECV = 5$ . Su comportamiento aparece en la Figura 2. Como se puede observar se reciben los mensajes en bloques de 5 bytes, siendo el último el relativo al carácter de terminación de `string`.

```
kaox@kaox-ROG-Zephyrus-G15-GA502IU-GA502IU: ~/3ºGrao/REDES/practica2/Redes_Practica2/basico$ ./cliente -I 127.0.0.1 -p 8002
Conectado con éxito al servidor 127.0.0.1 por el puerto 8002
Mensaje recibido: hola. Han sido recibidos 5 bytes.
Mensaje recibido: Mundo. Han sido recibidos 5 bytes.
Mensaje recibido: . Han sido recibidos 1 bytes.
```

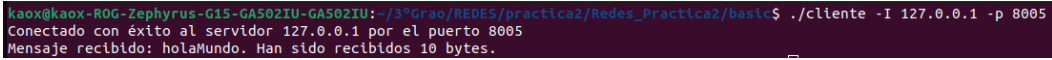
**Figura 2:** (1.d) – Salida por terminal del cliente con  $MAX\_BYTES\_RECV = 5$

- $MAX\_BYTES\_RECV = 10$ . Su comportamiento se refleja en la Figura 3. El cliente recibe en dos bloques distintos cada mensaje. Nótese que esta vez no hemos tenido que eliminar el carácter terminador de `strings` pues los mensajes se reciben de forma independiente.

```
kaox@kaox-ROG-Zephyrus-G15-GA502IU-GA502IU: ~/3ºGrao/REDES/practica2/Redes_Practica2/basico$ ./cliente -I 127.0.0.1 -p 8004
Conectado con éxito al servidor 127.0.0.1 por el puerto 8004
Mensaje recibido: hola. Han sido recibidos 5 bytes.
Mensaje recibido: Mundo. Han sido recibidos 6 bytes.
```

**Figura 3:** (1.d) – Salida por terminal del cliente con  $MAX\_BYTES\_RECV = 10$

- $MAX\_BYTES\_RECV = 15$ . En este caso, y de forma análoga al ejercicio (1.c), hemos tenido que gestionar el carácter de terminación de `strings` para la correcta impresión del mensaje.



```
kaox@kaox-ROG-Zephyrus-G15-GA502IU-GA502IU:~/3ºGrao/REDES/practica2/Redes_Practica2/bast$ ./cliente -I 127.0.0.1 -p 8005
Conectado con éxito al servidor 127.0.0.1 por el puerto 8005
Mensaje recibido: holaMundo. Han sido recibidos 10 bytes.
```

**Figura 4:** (1.d) – Salida por terminal del cliente con  $MAX\_BYTES\_RECV = 15$



## 4. Ejercicio 3

En este ejercicio debemos comprobar que el servidor de mayúsculas programado en el ejercicio (2) puede atender varios clientes secuencialmente. Para ello, modificamos el lazo del cliente de mayúsculas en el que se van leyendo las líneas del archivo, introduciendo un `sleep()` para que dé tiempo a lanzar un segundo cliente desde otra terminal.

Se realizaron los siguientes cambios en los códigos del ejercicio (2):

- En `servidormay.c` no hay cambios.
- En `clientemay.c`:

```
1 while (!feof(fp_input)) {
2     /* Leemos hasta que lo que devuelve getline es EOF,
3     cerramos la conexion en ese caso */
4     if(getline(&send_buffer, &buffer_size, fp_input) == EOF){
5         /* Escaneamos la linea hasta el final del archivo */
6         shutdown(client.socket, SHUT_RD); /* Le decimos al
7         servidor que pare de recibir */
8         continue;
9     }
10    if ( (sent_bytes = send(client.socket, send_buffer, strlen(
11    send_buffer) + 1, 0)) < 0) fail("No se pudo enviar el
12    mensaje");
13
14    /*Esperamos a recibir la linea*/
15    if( (recv_bytes = recv(client.socket, recv_buffer,
16    MAX_BYTES_RECV,0)) < 0) fail("No se pudo recibir el
17    mensaje");
18
19    fprintf(fp_output, "%s", recv_buffer);
20    /**** < CAMBIO PARA EL EJERCICIO 3 > ***/
21    sleep(1);
22    /**** </ CAMBIO PARA EL EJERCICIO 3 > ***/
23 }
```

Ejecutando los programas con estos cambios, con un servidor y dos clientes conectándose a él de forma simultánea, observamos que, en efecto, el servidor sí es capaz de atender a los dos conexiones de forma secuencial. En la Figura 5 se puede observar el resultado de la ejecución.

Como debería ocurrir, el servidor comienza a ejecutarse y espera por clientes que se conecten a él. Entonces, el primer cliente solicita una conexión, el servidor la acepta y se comienza a manejar. Paralelamente, el segundo cliente solicita a su vez

```

pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ ./servidormay 8080
Ejecutando servidor de mayúsculas con parámetros: PORT=8080, BACKLOG=16, LOG=log.
Servidor creado con éxito y listo para escuchar solicitudes de conexión.
Hostname: pedro-GL65-95EK; IP: 193.144.81.195; Puerto: 8080

Cliente conectado desde 127.0.0.1:42468.
Manejando la conexión del cliente 127.0.0.1:42468...
Cerrando la conexión del cliente 127.0.0.1:42468.
Cliente conectado desde 127.0.0.1:42476.
Manejando la conexión del cliente 127.0.0.1:42476...
Cerrando la conexión del cliente 127.0.0.1:42476.
^C
Cerrando el servidor y saliendo...
pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ |

pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ cat archivo1.txt
soy el archivo
creado para que me pasen
a mayúsculas en el cliente
numero 1

pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ ./clientemay archivo1.txt localhost 8080
Conectado con éxito al servidor 127.0.0.1 por el puerto 8080
Se procede a enviar el archivo: archivo1.txt
pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ cat ARCHIVO1.TXT
SOY EL ARCHIVO
CREADO PARA QUE ME PASEN
A MAYÚSCULAS EN EL CLIENTE
NUMERO 1
pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ |

pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ cat archivo2.txt
soy otro archivo
creado para que me pasen
a mayúsculas en el cliente
numero 2

pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ ./clientemay archivo2.txt localhost 8080
Conectado con éxito al servidor 127.0.0.1 por el puerto 8080
Se procede a enviar el archivo: archivo2.txt
pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ cat ARCHIVO2.TXT
SOY OTRO ARCHIVO
CREADO PARA QUE ME PASEN
A MAYÚSCULAS EN EL CLIENTE
NUMERO 2
pedro@pedro-GL65-95EK: ~/OneDrive/Tercero/Primer_Cuatril/Redes/Practicas/P2/mayus
$ |

```

**Figura 5:** (3) – Salida al ejecutar el servidor de mayúsculas y dos clientes

una conexión; este se queda bloqueado esperando a que su conexión se acepte, ya que el servidor está ocupado atendiendo la conexión del anterior cliente. Cuando se termina de procesar el archivo del primer cliente, este acaba y el servidor vuelve a escuchar por conexiones pendientes. Como la del segundo cliente está pendiente, la acepta y maneja su conexión a continuación.

Así, hemos verificado que el servidor es efectivamente capaz de manejar secuencialmente varias conexiones que le lleguen a la vez desde distintos clientes.