

# Informe Redes práctica 3

Autores: Álvaro Schwiedop Souto y  
Santiago Vilas Pampín

## 1

### C.

Para este apartado se pedía ver si el receptor podía recuperar el resto de los datos que envía el emisor si hacíamos que el *recvfrom()* tuviese como tamaño máximo del mensaje un número menor a la longitud del mensaje que se pretende recibir. Para ello limitamos el tamaño máximo del *recvfrom()* y añadimos otro:

```
socklen_t tamSocketRemoto = sizeof(socket_remoto);
ssize_t bytesRecv = recvfrom(receptor_fd, mensaje, 2, 0, (struct sockaddr
*)&socket_remoto, &tamSocketRemoto);
if (bytesRecv < 0)
    error("recvfrom fallo");
mensaje[bytesRecv] = '\n';
mensaje[bytesRecv+1] = '\0';

printf("Bytes recibidos: %ld\n", bytesRecv);
int remoto_port = ntohs(socket_remoto.sin_port);
char remoto_ip[INET_ADDRSTRLEN];
/* Convertimos una IP binaria en orden de red a IP legible */
inet_ntop(AF_INET, &socket_remoto.sin_addr, remoto_ip, INET_ADDRSTRLEN);
printf("Receptor recibió un mensaje en el puerto %d IP %s...\n", remoto_port,
remoto_ip);
printf("Mensaje recibido: %s", mensaje);

bytesRecv = recvfrom(receptor_fd, mensaje, 2, 0, (struct sockaddr
*)&socket_remoto, &tamSocketRemoto);
if (bytesRecv < 0)
    error("recvfrom fallo");
mensaje[bytesRecv] = '\n';
mensaje[bytesRecv+1] = '\0';

printf("Bytes recibidos: %ld\n", bytesRecv);
/* Convertimos una IP binaria en orden de red a IP legible */
inet_ntop(AF_INET, &socket_remoto.sin_addr, remoto_ip, INET_ADDRSTRLEN);
printf("Receptor recibió un mensaje en el puerto %d IP %s...\n", remoto_port,
remoto_ip);
printf("Mensaje recibido: %s", mensaje);
```

Pero se obtuvo que es imposible recibir la parte del mensaje perdida.

**d.**

receptor.c

```
#define MAX_SIZE 1000 * sizeof(float) // Cambiamos la macro

int main(int argc, char *argv[]) {

    . . .

    float mensaje [MAX_SIZE];

    . . .

    // sizeMssg es igual a MAX_SIZE
    ssize_t bytesRecv = recvfrom(receptor_fd, mensaje, sizeMssg, 0, (struct sockaddr
*)&socket_remoto, &tamSocketRemoto);

    . . .

    // Obtenemos en el receptor el tamaño del mensaje dividiendo el número
    // de bytes recibidos totales por el tamaño de un float
    int tamMens = bytesRecv / sizeof(mensaje[0]);

    printf("Bytes recibidos: %ld\n", bytesRecv);

    . . .

    printf("Mensaje recibido:\n");
    for (int i = 0; i < tamMens; ++i) {
        //Se imprimen uno a uno los floats
        printf("Mensaje[%d]: %f\n", i, mensaje[i]);
    }
    // Se imprime el número de float recibido
    printf("Número de floats: %d\n", tamMens);

    /* Cerramos la conexión del socket del servidor */
    close(receptor_fd);
    return 0;
}
```

emisor.c

```
#define N_FLOATS 1000

. . .

int main(int argc, char *argv[]) {

    . . .
    // Definimos nuestro mensaje como tipo float
```

```

float mensaje [N_FLOATS];

// El tamaño del mensaje a enviar es la cantidad de bytes
size_t sizeMssg = sizeof(mensaje);

// Se rellena el array con floats y los mostramos por pantalla
for (int i = 0; i < N_FLOATS; ++i)
{
    mensaje[i] = 2.5 * (i+2);
    printf("Mensaje[%d]: %f\n", i, mensaje[i]);
}

. . .

socklen_t tamSocketRemoto = sizeof(socket_remoto);

// La cantidad de bytes que enviamos es la del tamaño en bytes del array de
floats
ssize_t bytesEnv = sendto(emisor_fd, mensaje, sizeMssg, 0, (struct sockaddr
*)&socket_remoto, tamSocketRemoto);
if (bytesEnv < 0) error("sendto fallo");

printf("Bytes enviados: %ld\n", bytesEnv);

/* Cerramos la conexión del socket del servidor */
close(emisor_fd);
return 0;
}

```

El programa emisor genera un array de float con N\_FLOATS elementos y lo envía mediante UDP. El programa receptor recibe los datos, calcula automáticamente cuántos float fueron enviados y los imprime por pantalla. Esto permite que la cantidad de datos transmitidos sea determinada únicamente por el emisor, mientras que el receptor se adapta dinámicamente a la cantidad recibida.

### 3

clienteUDP.c:

```

#define MAX_SIZE 1000

/* Convertimos el nombre del archivo a mayúsculas */
char* to_uppercase_filename(const char *filename) {
    char *upper_filename = strdup(filename);

    if (!upper_filename) error("Memory allocation failed");
}

```

```

    for (int i = 0; upper_filename[i]; i++) {
        upper_filename[i] = toupper(upper_filename[i]);
    }
    return upper_filename;
}

int main(int argc, char *argv[]) {

    . . .

    while (fgets(mensaje, MAX_SIZE, fichero_entrada) != NULL) {

        . . .
        // Pausa para tener tiempo lanzar más clientes
        sleep(2);

        . . .

    }

    . . .

}

```

El servidor servidorUDP.c utiliza recvfrom dentro de un bucle infinito para recibir mensajes de cualquier cliente. Como el protocolo UDP no establece conexiones permanentes, por lo que el servidor puede recibir y responder a múltiples clientes de forma intercalada.

Al ejecutar un primer cliente, este empieza a enviar líneas con pausas de 2 segundos. Durante esos 2 segundos, se puede lanzar un segundo cliente en otra terminal que empieza a enviar sus líneas. El servidor recibe mensajes de ambos clientes en el orden en que llegan, los procesa (convierte a mayúsculas) y responde a cada cliente de manera independiente. Sin esa pausa, el proceso terminaría muy rápido y no podríamos lanzar las otras terminales en paralelo.

The screenshot shows two terminal windows. The left window displays the output of the server program (servidorUDP.c), which receives messages from clients and responds with the uppercase version of the received text. The right window shows the execution of the client program (clienteUDP.c) from a shell, demonstrating how multiple clients can be launched simultaneously, each sending a message to the server.

```

[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:20]
[{$}]> ./servidorUDP 5001
Enviados 5 bytes convertidos a mayúsculas
Bytes recibidos: 5
Receptor recibió un mensaje en el puerto 5003 IP 127.0.0.1...
Mensaje: TEST

Número de chars: 5
Enviados 8 bytes convertidos a mayúsculas
Bytes recibidos: 8
Receptor recibió un mensaje en el puerto 5003 IP 127.0.0.1...
Mensaje: ARCHIVO

Número de chars: 8
Enviados 5 bytes convertidos a mayúsculas
Bytes recibidos: 5
Receptor recibió un mensaje en el puerto 5004 IP 127.0.0.1...
Mensaje: TEST

Número de chars: 5
Enviados 7 bytes convertidos a mayúsculas
Bytes recibidos: 7
Receptor recibió un mensaje en el puerto 5003 IP 127.0.0.1...

[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:22]
[{$}]> ./clienteUDP 5004 127.0.0.1 5001 texto3.txt
Bytes enviados: 6
[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:22]
[{$}]>

[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:21]
[{$}]> ./clienteUDP 5001 5001 texto.txt
[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:21]
[{$}]> ./clienteUDP
Usó: ./clienteUDP <puerto_propio> <ip_destino> <puerto_destino> <archivo_texto>
[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:22]
[{$}]> cd ..
[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:24]
[{$}]> ./clienteUDP 5003 127.0.0.1 5001 texto2.txt
Bytes enviados: 6
[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:25]
[{$}]>

[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:24]
[{$}]> ./clienteUDP 5002 127.0.0.1 5001 texto1.txt
Bytes enviados: 6
[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:25]
[{$}]> ls
clienteUDP Makefile servidorUDP.c TEXT02.TXT
clienteUDP.c receptor texto1.txt texto3.txt
emisor receptor.c TEXT01.TXT TEXT02.TXT
emisor.c servidorUDP texto2.txt TEXT0.TXT
[alvaro omarchy-btw] - [~/Escritorio/USC/Redes/Practica3] - [jue oct 30, 11:26]
[{$}]>

```