INFORME PRÁCTICA SERVIDOR REDES

César Poza y Jaime Romero

24 de octubre de 2024

1. Introducción

El objetivo de esta práctica es establecer una conexión TCP entre un cliente y un servidor básicos implementados en C. Se crearán dos versiones de cliente y servidor: una básica, en la que el servidor envía dos simples mensajes al cliente, y otra con mayor complejidad. En la versión avanzada, el cliente lee un archivo de texto y lo envía al servidor, esperando a que el servidor transforme el contenido a mayúsculas y lo devuelva, guardándo finalmente el resultado en otro archivo.

2. Cliente/Servidor básicos

Para la creación de un cliente y un servidor básicos nos apoyamos en algunas de las librerías para la creación de sockets en C, incluyendo:

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

Gracias a estas librerías se podrán emplear, por ejemplo, las funciones de la API de sockets en C, permitiéndonos la comunicación entre programas mediante el uso del protocolo TCP/IP.

2.1. Dos mensajes

En este apartado se buca comprobar si es posible que el servidor envíe dos mensajes al cliente con dos llamadas a send(), y el cliente los reciba con una única llamada a recv(). Para ello el cliente tendrá que esperar entre que establece la conexión y recibe el mensaje, para que al servidor le dé tiempo a enviar los mensajes. Con una llamada a sleep de un par de segundos será más que suficiente.

servidor.c - El programa quedará igual que el del servidor básico, (leyendo el número de puerto desde termianl, creando las variables de los sockets y direcciones, las cadenas de mensajes) con las mismas llamadas a funciones (socket, bind, listen).

```
sockservidor = socket(AF_INET, SOCK_STREAM, 0);
bind(sockservidor, (struct sockaddr *) &direccionserv, sizeof(direccionserv))
listen(sockservidor, peticiones)
```

Sin embargo, ahora debemos modificar el bucle infinito que realiza los envíos. Primero acepta la petición del cliente con accept().

```
sockcliente = accept(sockservidor, (struct sockaddr *) &direccioncliente, &tamano)
```

Y a continuación realiza las dos llamadas consecutivas a send(), con el mismo parámetro del socket cliente, pero añadiendo mensaje uno o dos (y sus tamaños) según corresponda.

```
if(send(sockcliente, mensaje1, strlen(mensaje1), 0) < 0){
    perror("No se pudo enviar el mensaje");
    exit(EXIT_FAILURE);
}
if(send(sockcliente, mensaje2, strlen(mensaje2), 0) < 0){
    perror("No se pudo enviar el mensaje");
    exit(EXIT_FAILURE);
}
```

cliente.c - El programa cliente queda prácticamente igual (leyendo argumentos, creando socket y dirección del servidor), conectando con el servidor con socket() y connect():

```
sockservidor = socket(AF_INET, SOCK_STREAM, 0);
connect(sockservidor, (struct sockaddr *) &dirservidor, sizeof(dirservidor))
```

Ahora, antes de recibir los mensajes, es cuando llamamos a sleep(), para esperar un momento a que el servidor envíe ambos mensajes. Y después de esto, ya tendríamos la única llamada a recv() que guarda los mensajes que una cadena de texto, de la siguiente forma:

```
if(recv(sockservidor, mensaje, MAX, 0) < 0){
    perror("\nNo se pudo recibir el mensaje\n");
    exit(EXIT_FAILURE);
}else printf("\nMensaje recibido: \n%s\n", mensaje);
```

2.2. Sin espera

Ahora, sobre el apartado anterior, eliminaremos la espera y usaremos un bucle para recibir información secuencialmente. Para esto simplemente borramos (o comentamos) la llamada a sleep(), y sustituímos el código anterior que llama a recv() por el siguiente:

```
ssize_t n;
while((n = recv(sockservidor, mensaje, 100, 0)) > 0){
    printf("\nMensaje recibido: \n%s\n", mensaje);
}
```

Este bucle llama a recv() las veces que haga falta, mientras que el número de bytes que se reciban sea positivo (termina cuando se deja de recibir información).

El tercer parámetro de la función son los bytes a recibir, por lo tanto si cambiamos su valor a 2, por ejemplo, en vez de recibir el mensaje completo, lo recibiríamos por tandas de 2 caracteres en 2 caracteres:
Antes:

```
"Hola que tal"
```

Después:

```
"Ho", "la", " q", "ue", " t", "al"
```

3. Cliente/Servidor mayúsculas

Para esta versión del cliente-servidor, el objetivo es implementar un sistema algo más complejo que permite el intercambio de información entre el cliente y el servidor. En particular, crearemos dos programas, `servidormay.c` `clientemay.c`.
El cliente lee un archivo de texto (introducido por línea de comandos) y envía el contenido línea a línea al servidor (enviando la siguiente línea una vez reciba la línea transformada). El servidor, al recibir cada línea, la transforma a mayúsculas con la función `toupper()` y devuelve la línea transformada al cliente. A su vez, el cliente, cuando recibe cada línea la escribe en otro archivo de texto.

3.1. Varios Clientes Secuencialmente

Para realizar esta operación, partimos de los códigos servidor/cliente de mayúsculas, pero los modificaremos para que el servidor, además de recibir información las veces que haga falta del cliente, también la reciba de varios clientes secuencialmente.
Para conseguirlo, tendremos que introducir el código que acepta conexión, recibe, y envía línea; en un bucle infinito mayor. Así, el servidor queda activo atentiendo a varios clientes, hasta que lo cerremos manualmente con la señal SIGINT.
El código será el siguiente:

```
while(1){

    tamano = sizeof(direccioncliente);
    if((socketCliente = accept(socketServidor, (struct sockaddr *) &direccioncliente, &tam
        perror("No se pudo aceptar la conexion");
        continue;
    }

    while(1) {
        ssize_t n = recv(socketCliente, linea, MAXLINEA, 0);
        if (n < 0) {
            perror("No se pudo recibir el mensaje");
            close(socketCliente);
            break;
        } else if (n == 0) {
            printf("\nConexión cerrada por el cliente\n");
```

```
            close(socketCliente);
            break;
        }

        linea[n] = '\0';
        printf("\nLínea recibida: %s", linea);

        mayus(linea);

        if(send(socketCliente, linea, strlen(linea), 0) < 0){
            perror("No se pudo enviar el mensaje");
            close(socketCliente);
            break;
        }
        printf("\nLínea enviada: %s", linea);
    }
}
```

Con el bucle while grande, el servidor está activo y recibe peticiones de clientes, las cuales acepta con accept(). Una vez establecida la conexión, entra en el segundo bucle, que se encarga de recibir y enviar las líneas, convirtiéndolas a mayúsculas. Esto lo hace con las funciones que ya estuvimos usando, recv() y send(), y la función mayus() que tuvimos que implementar para el ejercicio.