Hugo Chalard Collado y Juan Calzada Bernal

Finger sockets tdp/udp

# Detalles relevantes del desarrollo

Explicaremos como hemos desarrollado el UDP por una parte y el TCP por otra.

## TCP

### Para hacer la recepción TCP y asegurarnos de que el otro peer recibe todo lo enviado, es necesario seguir los pasos especificados en este blog con claridad: <https://blog.netherlabs.nl/articles/2009/01/18/the-ultimate-so_linger-page-or-why-is-my-tcp-not-reliable> No es suficiente con hacer el linger. Estos pasos, en general, son:

#### Una vez terminado el envío (no se va a volver a llamar a send()), se hace un shutdown(s, SHUT\_WR).

#### Antes de llamar a close(socket), se debe llamar a recv() para limpiar el buffer de entrada.

## UDP

### Para hacer la comunicación UDP, es necesario establecer unos timeouts y reintentos en el cliente UDP, ya que este protocolo de transporte no asegura la recepción del mensaje en el servidor. *reintentos = REINTENTOS while(reintentos--) { sendto() alarm(TIMEOUT) recv() if(!recv && (errno == EINTR)) continue; else { // Tratar respuesta recibida } }*

### Es necesario establecer unos tamaños máximos de envío para los paquetes UDP, ya que el tratamiento en el receptor se hace sobre solamente un paquete.

## En general, nuestra práctica se compone por:

### Cliente:

#### cliente.c: Contiene el main de cliente, llama a cliente\_tcp o cliente\_udp en función de los parámetros de entrada.

#### client\_tcp.h: Contiene el código para hacer las peticiones al servidor de finger en protocolo TCP.

#### client\_udp.h: Lo mismo para UDP.

#### common\_client.h: Contiene declaraciones compartidas necesarias para el código del cliente (tanto TCP como UDP)

### Servidor:

#### servidor.c: Programa servidor principal que configura los servidores TCP y UDP, gestiona las conexiones entrantes mediante select(), bifurca los procesos hijo para gestionar las peticiones y gestiona el apagado graceful del servidor.

#### compose\_finger.h: Implementa la funcionalidad central del protocolo finger, gestionando la visualización de la información del usuario incluyendo el estado del login, directorio, shell, estado del correo y archivo plan, usando UTMP/UTMPX para el seguimiento de las sesiones de usuario.

#### parse\_client\_request.h: Maneja el análisis sintáctico y la validación de las peticiones del protocolo finger del cliente, soportando varios formatos incluyendo nombre de usuario, nombre de host, y combinaciones de banderas /W.

#### server\_tcp.h: Implementa la funcionalidad de servidor TCP para el protocolo finger, manejando las conexiones de cliente, recibiendo peticiones, generando respuestas a través de compose\_finger, y gestionando el ciclo de vida de la conexión.

#### server\_udp.h: Implementa la funcionalidad del servidor UDP con características similares a las del servidor TCP pero adaptadas para el protocolo sin conexión, incluyendo el manejo del tiempo de espera y las limitaciones de tamaño de las respuestas.

#### common\_server.h: Contiene funciones de ayuda para el registro de eventos y la gestión de errores, además de una función para analizar y manipular los nombres de host en las solicitudes que implican varios hosts en una cadena.

### common.h: Contiene definiciones y funciones comunes utilizadas por el servidor y cliente finger, incluyendo constantes de configuración (puerto, tamaños de buffer, timeouts) y una función para validar el formato CRLF de los mensajes.

## Hemos decidido implementar las funcionalidades externas en archivos .h debido a la facilidad que aporta para la reutilización de código y su mantenibilidad. Se podría haber implementado igualmente en archivos .c

# pruebas de funcionamiento realizadas

## Pruebas de funcionamiento de funciones auxiliares

### Hemos realizado pruebas de funcionamiento de las funciones auxiliares con estos archivos fuente.

#### compose\_finger.c:  verifica la funcionalidad de composición de respuestas finger, tanto para listado de todos los usuarios como para usuarios específicos, validando también el formato CRLF de las respuestas.

#### test\_check\_crlf\_format.c: pruebas unitarias para la función que verifica el formato CRLF (retorno de carro + nueva línea) en las cadenas de texto, comprobando diferentes casos de uso y formatos.

#### test\_parse\_client\_request.c: verifica el correcto funcionamiento del parser de peticiones del cliente finger, probando diferentes formatos de entrada (usernames, hostnames, flags) y validando las respuestas esperadas.

## Pruebas de funcionamiento de conexión UDP y TCP.

### Para poder comprobar que la conexión se realizaba de forma correcta con peticiones a servidores externos (i.e. diferentes al localhost) hemos levantado un servidor en un VPS (con nombre finger.run.place) y hemos comprobado que se pueden enviar grandes volúmenes de informacion (en TCP) aunque el servidor no sea local. Fue al probar esto cuando nos dimos cuenta de que algo fallaba al cerrar el socket TCP, si se cerraba antes de que el peer hubiese asentido toda la información, con *close()*, se forzaba el cierre de la conexión y linger no lo solucionaba. Fue necesario hacer un bucle con *recv()* (y desechar lo recibido) en el servidor antes de llamar a *close().* Visto antes en: <https://blog.netherlabs.nl/articles/2009/01/18/the-ultimate-so_linger-page-or-why-is-my-tcp-not-reliable>

### Para enviar grandes volúmenes de información cambiamos temporalmente las funciones que componen el finger en *compose\_finger.h* En la siguiente captura se muestra como conseguimos enviar 600 MB de datos sin problema, en una conexión TCP.

### El servidor, en principio, está levantado ahora mismo, si se quiere comprobar el funcionamiento, solo que no enviará 600 MB de datos, enviará una respuesta normal de finger. (en caso de que se quiera comprobar el funcionamiento y no esté levantado contactar con cualquiera de los dos alumnos)

### Para probarlo en nogal.usal.es simplemente hemos comprobado que lanzaServidor.sh se ejecuta correctamente y produce la salida deseada.