Hugo Chalard Collado y Juan Calzada Bernal

FINGER SOCKETS TDP/UDP

1. DETALLES RELEVANTES DEL DESARROLLO

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

A. TCP

i. 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:

1. Una vez terminado el envío (no se va a volver a llamar a *send())*, se hace un *shutdown(s, SHUT_WR)*
2. Antes de llamar a *close(socket)*, se debe llamar a *recv()* para limpiar el buffer de entrada.

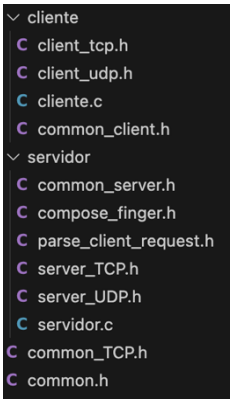
B. UDP

i. 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
  }
}
```

ii. 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.

C. En general, nuestra práctica se compone por:



i. Cliente:
   1. *cliente.c:* Contiene el main de cliente, llama a cliente_tcp o cliente_udp en función de los parámetros de entrada.
   2. *client_tcp.h:* Contiene el código para hacer las peticiones al servidor de finger en protocolo TCP.
   3. *client_udp.h:* Lo mismo para UDP.
   4. *common_client.h:* Contiene declaraciones compartidas necesarias para el código del cliente (tanto TCP como UDP)

ii. Servidor:
   1. *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.
   2. *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.
   3. *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.
   4. *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.
   5. *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.
   6. *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.

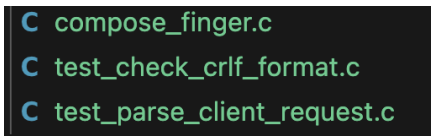
iii. common.h: Contiene definiciones y funciones comunes utilizadas por el servidor y cliente finger, incluyendo constantes de configuración (puerto, tamaños de buffer, timeouts) y una función para validar el formato CRLF de los mensajes.

D. 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

2. PRUEBAS DE FUNCIONAMIENTO REALIZADAS

A. Pruebas de funcionamiento de funciones auxiliares

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



1. *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.

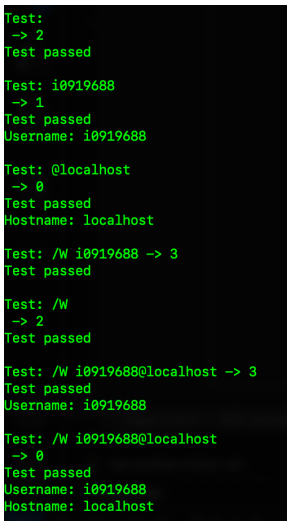
```
<nogal>/home/i0919688/PracticaRedesSocket/RedesISocket/temp$ ./composeFinger
Login: i3307307                 Name: Vicente Flores Sergio
Directory: /home/i3307307       Shell: /bin/bash
On since Wed Dec 11 12:44 (CET) on pts/0 from 10.50.22.24
No mail.
No Plan.

Login: i1042457                 Name: Enriquez Gago Angel
Directory: /home/i1042457       Shell: /bin/bash
On since Wed Dec 11 13:46 (CET) on pts/3 from 93.156.211.116
On since Wed Dec 11 13:46 (CET) on pts/12 from 93.156.211.116
No mail.
No Plan.

Login: i2259918                 Name: Lobo Miron Diego
Directory: /home/i2259918       Shell: /bin/bash
On since Wed Dec 11 12:45 (CET) on pts/7 from 79.117.136.5
No mail.
No Plan.
```

2. *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.

```
juancalzadabernal@MacBook-Air-de-Juan test % ./crlf
1 Test passed for buffer: hola


2 Test passed for buffer: hola


3 Test passed for buffer: hola


4 Test passed for buffer:


5 Test passed for buffer:


6 Test passed for buffer: hola
```

```
7 Test passed for buffer: hola


8 Test passed for buffer: hola


9 Test passed for buffer: hola


10 Test passed for buffer: hola


12 Test passed for buffer: Response doesn't fit in UDP packet
```

3. *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.



B. Pruebas de funcionamiento de conexión UDP y TCP.

i. 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 lo cierras antes de que el peer asienta toda la información, con *close()*, se fuerza el cierre de la conexión y linger no lo soluciona. Es necesario hacer un bucle con *recv()* (y desechar lo recibido) en el servidor antes de llamar a *close()*

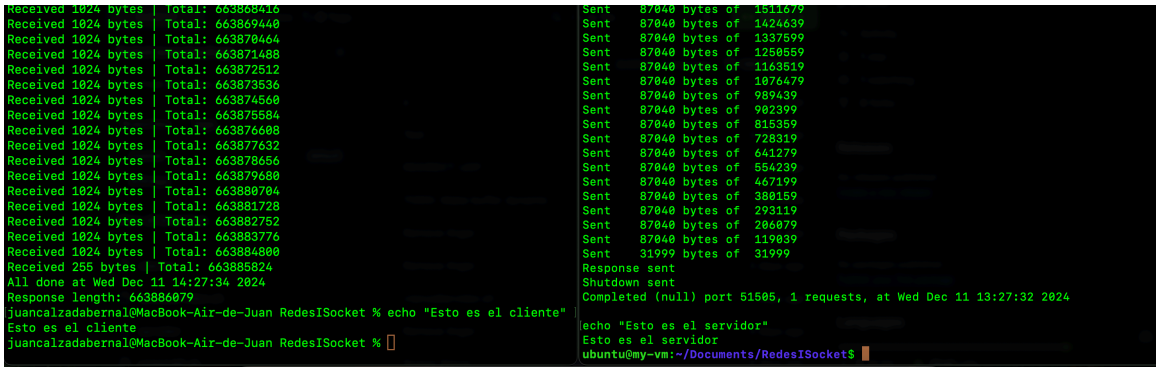
```c
char dummy_buffer[1024];
alarm(TIMEOUT);
while (recv(s, dummy_buffer, sizeof(dummy_buffer), 0) > 0)
  ;
alarm(0);
// Now we must close the connection
close(s);
```

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 hemos cambiado temporalmente las funciones que componen el finger en *compose_finger.h*

```c
#ifdef SEND_BIG_CHUNK

const int CHUNK_SIZE = 663886080; // 660 mb

char *all_users_info() {
  char *info = (char *)malloc(size: CHUNK_SIZE); // 660 mb
  if (!info) {
    return NULL;
  }
  memset(b: info, c: 'A', len: CHUNK_SIZE - 1);
  info[CHUNK_SIZE - 1] = '\0';
  info[CHUNK_SIZE - 2] = '\n';
  info[CHUNK_SIZE - 3] = '\r';
  return info;
}

char *just_one_user_info(char *username) {

  char *info = (char *)malloc(size: CHUNK_SIZE); // 660 mb
  if (!info) {
    return NULL;
  }
  memset(b: info, c: 'A', len: CHUNK_SIZE - 1);
  info[CHUNK_SIZE - 1] = '\0';
  info[CHUNK_SIZE - 2] = '\n';
  info[CHUNK_SIZE - 3] = '\r';
  return info;
}

#else
```

En la siguiente captura se muestra como hemos conseguido enviar 600 MB 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)

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

```
<nogal>/home/i0919688/PracticaRedesSocket/RedesISocket$ make run
gcc -O3  -o servidor src/servidor/servidor.c
gcc -O3  -o cliente src/cliente/cliente.c
chmod 700 lanzaServidor.sh
./lanzaServidor.sh
<nogal>/home/i0919688/PracticaRedesSocket/RedesISocket$ ls
45237.txt  46560.txt  49924.txt  59633.txt         Makefile        spawnAll.sh   temp
46548.txt  46574.txt  52531.txt  cliente           peticiones.log  spawnUser.sh
46558.txt  46584.txt  55903.txt  lanzaServidor.sh  servidor        src
<nogal>/home/i0919688/PracticaRedesSocket/RedesISocket$
```

```
<nogal>/home/i0919688/PracticaRedesSocket/RedesISocket$ cat 46574.txt
Login: p1777001                 Name: Moreno Montero Angeles MarIa
Directory: /home/p1777001       Shell: /bin/bash
Last login Tue Dec 03 12:53 (CET) on pts/34 from 172.20.1.150
No mail.
No Plan.
```