Práctica 2. Sockets C y Java en modo stream (TCP)

SCS, 2010/11

5 de octubre de 2010

Índice

1. Ejercicio 1: Sockets TCP en C

Se mostrará la creación y uso de sockets TCP en C y la interacción entre servidores y clientes escritos en diferentes lenguajes.

Se parte de un código de ejemplo que implementa un servicio de cifrado de cadenas de caracteres sencillo.

Descarga: sockets1.tar.gz

1.1. Funcionamiento

El servidor escucha en el puerto 12345 y maneja 2 tipos de peticiones con la siguiente sintaxis:

```
CIFRAR:<cadena a cifrar>
DESCIFRAR:<cadena a descrifrar>
```

Nota: todos los mensajes intercambiados (peticiones y respuestas) terminan con `'\n'`

Recibida la petición, en función del comando indicado, el servidor responde con la versión cifrada o descifrada de la cadena recibida como parámetro.

- El método de cifrado implementado se conoce como algoritmo del César.

- En el cifrado se reemplaza cada letra por la que está 3 posiciones más adelante en el alfabeto. El descrifrado realiza la operación complementaria.

1.2. Componentes y compilación

Se incuye una implementación en C del servidor monoproceso y otra multiproceso. Se incluyen también 2 clientes para acceder al servicio, uno escrito en C y otro en Java.

- Servidores:
  - `servidor_cifrado.c`
  - `servidor_cifrado_multi.c`
- Clientes:
  - `cliente_cifrado.c`
  - `ClienteCifrado.java`
- Código adicional
  - `mierror.h, mierror.c`: gestión de errores
  - `cifrado.h, cifrado.c`: implementación del *cifrado del César*

Compilación:

```
$ tar xzvf socket1.tar.gz
$ cd sockets1
$ make
```

Uso:

- Lanzar servidor

  ```
  $ ./servidor   ó    $ ./servidor_multi
  ```

- Lanzar los clientes en un terminal distinto

  ```
  $ ./cliente  localhost -c "cadena a cifrar"
  $ ./cliente  localhost -d "cadena a descifrar"

  $ java ClienteCifrado  localhost -c "cadena a cifrar"
  $ java ClienteCifrado  localhost -d "cadena a descifrar"
  ```

1.3. Tareas a realizar

- Inspeccionar el código proporcionado y ver las diferencias y similitudes entre el servidor monoproceso y el multiproceso y entre los clientes C y Java
- Verificar con `netstat` que el servidor está a la escucha
- Comprobar el funcionamiento de los dos tipos de clientes
- Comprobar con `nc` el envio de mensajes al servidor. Repetirlo con `nc` en modo escucha para comprobar el formato de los mensajes enviados por los clientes.
- Comprobar la diferencia de funcionamiento entre el servidor monoproceso y multiproceso

  Nota: para hacer más evidente la diferencia puede ser util incluir un retardo de 5 s. o 10 s. al inicio de la función `procesar_peticion(...)` de ambos servidores
  - Usar la función `sleep(5);` (requiere añadir `#include <unistd.h>`)

2. Ejercicio 2: Sockets TCP en Java

Se parte de un código de ejemplo que implementa un servidor de diccionarios.

Descarga: sockets2.tar.gz

2.1. Funcionamiento

El servidor escucha en el puerto 22222.

El protocolo que implementa maneja los siguientes mensajes (todos los mensajes intercambiados [peticiones y respuestas] terminan con un salto de línea `'\n'`)

`BUSCAR <palabra>\n`	busca la palabra en el diccionario y devuelve su entrada o informa de que no la ha encontrado
`LISTAR \n`	devuelve la lista de palabras contenidas en el diccionario
`AYUDA \n`	devuelve la lista de comandos reconocidos por el servidor
`SALIR \n`	termina la sesión con el servidor de diccionarios

2.2. Componentes y compilación

- Clases y ficheros:
  - `ServidorDiccionario`: implementación monohilo del servidor
  - `GestorCliente`: gestiona el diálogo con los clientes y procesa las peticiones
  - `ClienteDiccionario`: ejemplo de cliente, espera recibir comandos en `STDIN`, los envía al servidor y muestra la respuesta por pantalla
  - `Diccionario`: clase que encapsula un diccionario, ofrece métodos en los que delega el servidor para implentar los comandos de gestión del diccionario (`buscarPalabra()`, ...)
  - `PeticionDiccionario`: clase que encapsula una petición al servidor de diccionarios, contiene un String con el comando enviado y un array de 1 ó 2 Strings con sus argumentos.
  - `general.dic`: diccionario general usado por el servidor

  f
- Compilación y ejecución:

```
$ tar xzvf sockets2.tar.gz
$ cd sockets2

$ javac ServidorDiccionario.java
$ javac ClienteDiccionario.java

$ java ServidorDiccionario
$ java ClienteDiccionario localhost   (en un terminal distinto)
```

Verificar que el servidor no soporta el acceso concurrente desde dos clientes.

2.3. Tareas a realizar

Se tratará de extender el servidor de partida con 2 ampliaciones

1. Modificar el servidor para construir una versión multihilo que permita atender a varios clientes a la vez
   - Usar el esquema presentado en los apuntes (pág. 23, Tema 2)

2. Ampliar el protocolo para crear un servidor con estado que maneje diccionarios personalizados y que soporte los siguientes comandos:

   - `LOGIN <usuario>\n`
     - Inicia una sesión para el usuario indicado (devuelve el mensaje `USUARIO CONECTADO`)
     - Crea un nuevo objeto `Diccionario` usando como nombre de diccionario el identificador de usuario indicado
     - Una vez creado ese objeto `Diccionario` (`new Diccionario(''<usuario >'')`) se cargará el fichero `<usuario>.dic` (método `cargar()`) con los contenidos del diccionario del usuario.
       - Si ese usuario ya existía se leerán las palabras del fichero `usuario.dic`
       - Si no existía el usuario, continúa con un diccionario vacio
     - El objeto `Diccionario` representa al diccionario privado de ese usuario y sólo será accesible mientras dure la sesión actual
       - Al recibir un comando `SALIR` se guarda la versión actual del diccionario en el fichero `usuario.dic` (método `guardar()`) y desaparece el objeto `Diccionario` [`null`])
   - `LOGOUT`
     - Finaliza la sesión del usuario actual (si lo hubiera), devuelve el mensaje `USUARIO DESCONECTADO`
     - Si se ha iniciado sesión, libera el diccionario personal del correspondiente usuario y el servidor pasará funcionar con el diccionario general y los comandos básicos.
     - Si no se ha iniciado sesión, informa del error con la respuesta `NINGÚN USUARIO CONECTADO`
   - `BUSCAR <palabra>\n`
     - Si se ha iniciado sesión de usuario, se busca primero en su diccionario privado y después en el general
     - Si no se ha iniciado sesión, se busca únicamente en el diccionario general
   - `LISTAR \n`
     - Si se ha iniciado sesión de usuario, muestra la lista de palabras del diccionario privado
     - Si no se ha iniciado sesión, muestra las palabras del diccionario general
   - `ANADIR <palabra> <texto_definición>\n` (un espacio separa palabra y definición)
     - Si se ha iniciado sesión de usuario, añade el par (palabra, definición) al diccionario privado
     - Si no se ha iniciado sesión, devuelve mensaje `COMANDO INCORRECTO`
   - `BORRAR <palabra>\n`
     - Si se ha iniciado sesión de usuario, borra la entrada asociada a palabra
     - Si no se ha iniciado sesión, devuelve mensaje `COMANDO INCORRECTO`
   - `SALIR \n`
     - Si se ha iniciado sesión de usuario, guarda el diccionario privado a disco [método `guardar()` de la clase `Diccionario`]
   - `AYUDA \n`
     - Muestra la lista de (todos) los comandos disponibles

3. Documentación a entregar

- Para el **ejercicio 1** basta un comentario de las pruebas realizadas y los resultados obtenidos.
- Para el **ejercicio 2** se entregarán los fragmentos de código incorporados (funciones, clases o fragmentos nuevos) para dar soporte a las ampliaciones introducidas, junto con una muestra de su ejecución y uso.