Universidad
Rey Juan Carlos

Práctica 1: Comunicación mediante sockets

Sistema distribuidos y concurrentes

2021/2022

El objetivo de esta primera práctica es afianzar las nociones de comunicaciones, sockets y su implementación en C.

Servidor-Cliente Simple

Desarrollar 2 procesos distintos, un servidor y cliente en C que se comuniquen mediante sockets.

- La comunicación debe ser síncrona y bidireccional.
- El cliente y servidor deberán intercambiar mensajes de tipo char (no se utilizarán structs). Tanto el cliente como el servidor leerán mensajes de la entrada estándar y los enviarán por el socket.
- Ambos procesos deben mostrar en el terminal el símbolo '>' para indicar que se requiere escribir en la entrada estándar (man fgets, scanf)
- Los mensajes recibidos serán mostrados por pantalla. Los mensajes recibidos en cada proceso deben mostrarse con el prefijo "+++".
- El cliente y el servidor estarán leyendo de la entrada estándar infinitamente. Hasta que alguno de los dos termine. Ambos procesos deben permitir finalizar correctamente usando la combinación Control+C. (Consultar man signal, SIGINT)
- El servidor solo puede atender a un cliente a la vez.

Ejemplo lado servidor:

```
$ ./server
Socket successfully created...
Socket successfully binded...
Server listening…
+++ Hello server!
> Hello client!
```

Ejemplo lado cliente:

```
$ ./client
Socket successfully created…
connected to the server...
> Hello server!
+++ Hello client!
```

Servidor-Cliente Simple No Bloqueante

Mejora el código del servidor-cliente simple desarrollado en el apartado anterior para que permita la siguiente funcionalidad:

- Los recv() del cliente y servidor deben ser no bloqueantes.
- Si alguno de los dos procesos (cliente o servidor), no tiene nada que leer del socket debe mostrar de nuevo el carácter '>'
- El cliente debe esperar al menos 0.5 segundos
- Para el servidor no hay especificaciones en cuanto al tiempo máximo de espera en el recv().

Servidor-Cliente Complejo (multi-hilo)

Mejora el código del servidor-cliente anteriores para que permita la siguiente funcionalidad:

- El servidor tiene que permitir comunicación bidireccional y no-bloqueante
- El servidor puede atender como máximo a 100 clientes a la vez. Para ello generará un thread por cada conexión nueva entrante.
- Usa el cliente simple como cliente para realizar las pruebas, con una modificación. El cliente debe aceptar como parámetro un número que servirá como client_id.
- El mensaje a enviar desde el cliente será del tipo `"Hello server! From client: 1"`. Puedes usar structs para intercambiar mensajes a través de sockets.
- El mensaje de respuesta del servidor será siempre `"Hello client!"`. Una vez el cliente reciba el mensaje, debe finalizar correctamente.

Para realizar las pruebas de multicliente se recomienda usar un código bash de este estilo:

```
for i in `seq 1 100`; do
   WAIT=`printf '0.%06d\n' $RANDOM`;
  (sleep $WAIT; echo "Lanzando cliente $i ..."; ./client $i) &
done
```

Requisitos mínimos de la entrega:

1. Se entregarán las 3 versiones del cliente-servidor descritas en el enunciado.
2. El servidor y cliente deben funcionar aún ejecutando en máquinas diferentes.
3. El código debe compilar y ejecutar correctamente en los ordenadores del laboratorio. Deberás incluir un fichero Makefile o script para la compilación correcta de los proyectos.
4. Deben llegar los mensajes de todos los clientes.
5. Será necesario realizar captura de errores para todas las primitivas de comunicación utilizadas.

Se valorará:

1. Correcto funcionamiento de la aplicación
2. Limpieza y correcta tabulación del código.
3. Estilo de código
4. Descomposición en funciones simples.