Práctica 5: Programación en C++ de sistemas cliente–servidor basados en comunicación síncrona

Programación de Sistemas Concurrentes y Distribuidos

Dpto. de Informática e Ingeniería de Sistemas,
Grado de Ingeniería Informática
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

1. Objetivos

Esta práctica consiste en programar un sistema cliente–servidor mediante un modelo de comunicación síncrona, análogo al presentado en las clases de teoría. Los objetivos concretos de la práctica son:

- Programar procesos distribuidos que se comuniquen mediante *sockets* síncronos en C++.
- Aprender a implementar sistemas cliente–servidor en C++.
- Profundizar en el modelo de concurrencia de C++.

2. Trabajo previo a la sesión en el laboratorio

Antes de la correspondiente sesión en el laboratorio, cada estudiante deberá leer el enunciado, analizar los problemas que en él se proponen y realizar un diseño previo de las soluciones sobre las que va a trabajar. Los resultados de su trabajo de análisis y diseño los tendrá que expresar en un documento que entregará en papel a los profesores antes del inicio de la sesión (en el mismo laboratorio). El documento debe contener como mínimo el nombre completo y el NIP del estudiante y, para cada ejercicio,

- la descripción de los procesos involucrados en el sistema y los protocolos de interacción que determinan la comunicación entre los procesos (contenido de los mensajes intercambiados, orden de los mensajes, gestión de errores en la interacción, etc.).
- un esbozo de alto nivel del código de los procesos cliente y servidor indicando las zonas que están afectadas por la comunicación y la sincronización.

- una descripción de los mecanismos utilizados en el lado del servidor para gestionar las restricciones de sincronización.

La entrega de este documento es un pre-requisito para la realización y evaluación de la práctica. El documento debe estar correctamente escrito, sin faltas de ortografía, y cumplir los mínimos exigibles de presentación. Caso contrario no se valorará.

Por otro lado, el enunciado de la práctica también propone una serie de ejercicios previos con objeto de ayudar a los alumnos a comprender cómo programar una aplicación cliente–servidor (sección 4). Se recomienda la realización de estos ejercicios antes de la sesión de laboratorio.

3. La abstracción de socket y el modelo cliente–servidor

Un modelo arquitectural fundamental para el desarrollo de aplicaciones distribuidas es el modelo *cliente–servidor*. Un *servidor* es un proceso que ofrece una o más operaciones que son accesibles a través de la red. Por otro lado, un *cliente* es un proceso que invoca (normalmente de forma remota) las operaciones ofrecidas por un servidor. El proceso servidor está a la espera de recibir invocaciones por parte de sus clientes. Una vez recibida una invocación, lleva a cabo el procesamiento requerido, usando los parámetros suministrados por el cliente, y le devuelve resultado.

La comunicación entre los procesos cliente y servidor requiere de una infraestructura de red. Una forma común de comunicación, presente en la mayoría de los entornos y lenguajes de desarrollo, es la basada en *sockets*. Los *sockets* representan los extremos de la conexión bidireccional que se establece para llevar a cabo esta comunicación. Cuando dos procesos requieren comunicarse solicitan al sistema operativo la creación de un *socket*. La respuesta a esta solicitud es un identificador que permite al proceso hacer referencia al nuevo *socket* creado.

Atendiendo a la pila de protocolos de Internet existen dos tipos de *sockets*:

- *Sockets* orientados a comunicación síncrona. El uso de este tipo de *sockets* proporciona una transmisión bidireccional, continua y fiable (los datos se comunican ordenados y sin duplicados) con conexión mediante el protocolo TCP (*Transport Control Protocol*).

- *Sockets* orientados a comunicación asíncrona. El uso de este tipo de *sockets* proporciona una transmisión bidireccional, no fiable, de longitud máxima prefijada y sin conexión mediante el protocolo UDP (*User Datagram Protocol*).

En esta práctica se proporciona una librería sencilla que abstrae los detalles de bajo nivel de comunicación vía sockets (la especificación de la librería se encuentra en los fuentes adjuntos en el web de la asignatura). La librería define una clase `Socket` con métodos públicos análogos a los utilizados en clase, enmascarando algunos detalles de manipulación de estructuras de bajo nivel, que estudiaréis en profundidad en otras asignaturas. Concretamente, en esta práctica nos centramos en comunicación síncrona entre procesos cliente y servidor.

Tanto para el proceso cliente como para el proceso servidor el primer caso será crear el objeto `Socket` mediante su constructor. Para su creación, el servidor tiene que suministrar el puerto en el que se va a publicar el servicio. Por su parte, el cliente tiene que suministrar tanto la IP donde se encuentra el servidor como el puerto del servicio.

Desde el punto de vista del **servidor** se requiere:

1. **Bind()**. Aviso al sistema operativo de que se quiere asociar el programa actual al socket abierto.
2. **Listen()**. Aviso al sistema operativo de que se procede a escuchar en la dirección establecida por el socket.
3. **Accept()** Aceptación de clientes. Si no hay ninguna petición de conexión, la función permanecerá bloqueada hasta que se produzca alguna. Cada vez que se ejecute, se acepta la conexión con un cliente que lo está solicitando.
4. **Recv()** Recepción de información a través del puerto. Análoga a la instrucción => utilizada en clase.
5. **Send()** Envío de información a través del puerto. Análoga a la instrucción <= utilizada en clase.
6. **Close()** Cierre de la comunicación y del socket.

En el **cliente**:

1. **Connect()**. Conexión con el servidor en la dirección y el puerto especificados.
2. **Send()** Envío de información a través del puerto.
3. **Recv()** Recepción de información a través del puerto.
4. **Close()** Cierre de la comunicación y del socket.

4. Ejercicios previos

El propósito de los dos siguientes ejercicios es introducir y formar al alumno en la programación de sistemas cliente–servidor. Se recomienda encarecidamente programar, ejecutar y comprender el funcionamiento de estos dos sencillos ejemplos como paso previo a la realización de la práctica.

4.1. Ejercicio 1

Los fuentes adjuntos en el web de la asignatura contienen un ejemplo de proceso servidor que escucha peticiones de un cliente en un puerto específico, así como el código de un cliente que usa dicho servicio. La instrucción

```
string SERVER_ADDRESS = "localhost";
```

obliga a que el cliente se ejecute en la misma máquina que el servidor.

El proceso servidor está asociado al número de puerto indicado en la variable `SERVER_PORT`. El proceso cliente solicita al proceso servidor que cuente el número de vocales de las frases que debe introducir el usuario por la entrada estándar. El proceso servidor atiende las peticiones del cliente y le comunica la respuesta, proceso que se repite hasta que recibe la cadena `END OF SERVICE`, momento en que el servidor finaliza su ejecución.

El cliente, en primer lugar, establece una conexión con el servidor para solicitarle sus servicios. A continuación, envía las diferentes peticiones de servicio, recibe las respuestas del servidor e informa al usuario (mediante la salida estándar) del número de vocales contabilizadas por el servidor para cada una de las frases introducidas. Cuando el usuario introduce la cadena `END OF SERVICE`, el cliente la remite al servidor y finaliza su ejecución.

Ejecutad el servidor en un terminal. A continuación, abrid un nuevo terminal y ejecutad el cliente. En este caso el servidor y el cliente se están ejecutando en la misma máquina; es decir, en modo local. Analizad el comportamiento de ambos procesos y las comunicaciones que se establecen entre ellos.

El ejercicio pide modificar ambos programas. En el caso del servidor, el puerto será un parámetro de invocación desde línea de comandos. En el caso del cliente, tomará dos parámetros: la IP donde se encuentra el servidor, así como el puerto en el que escucha. Para probar su ejecución, debéis averiguar la dirección IP del ordenador en el que vais a ejecutar el servidor (mediante el comando `host hendrix01.cps.unizar.es`, por ejemplo) y ponerlo en marcha. A continuación, ejecutad el cliente en un ordenador diferente al que ejecuta el servidor (por ejemplo, que el compañero que tengas al lado ejecute su cliente para que invoque el servicio que ofrece tu servidor). Analizad el comportamiento de ambos procesos y las comunicaciones que se establecen entre ellos.

4.2. Ejercicio 2

En la versión anterior el servidor queda asociado y conectado al primer cliente que se conecte. Lo habitual es que un servidor atienda simultáneamente a más de un cliente. Se pide modificar el código del ejercicio anterior para que el servidor atienda a múltiples clientes. Para ello, cuando reciba la petición de conexión por parte de un nuevo cliente (el servidor ejecuta un `Accept()`), debe crear un *thread*, adecuadamente parametrizado, para atender a ese cliente (podemos suponer que no llegará a haber más de 10 clientes).

Ejecutad el servidor en un terminal. A continuación, ejecutad un cliente en local y otro en remoto y analizad el comportamiento de los procesos que se crean y las comunicaciones que se establecen entre ellos.

5. Ejercicio

A continuación se describe el ejercicio a resolver durante esta práctica y las tareas que deben completar los alumnos.

5.1. Enunciado

En este ejercicio se va a implementar un programa que simula un servicio de reserva de plazas en un avión. Para simplificar, vamos a asumir que el avión contiene 4 filas de 4 asientos. Cada plaza se identifica de forma única por el número de fila (valor entero comprendido entre 1 y 4) y un número de asiento (en este caso se usa una letra mayúscula entre A y D). Por ejemplo, el identificador "2C" corresponde con la plaza ubicada en la segunda fila y el tercer asiento.

El objetivo del ejercicio es programar un sistema cliente-servidor en el que varios clientes se conectan al servidor para reservar plazas del avión. Un cliente tiene como objetivo reservar el mayor número posible de plazas, pero de una en una conforme el siguiente protocolo de interacción:

- El cliente, una vez establecida la comunicación con el servidor, le enviará como primer mensaje "INICIO COMPRA:Cliente_1", donde *Cliente_1"* es la identidad del cliente (su nombre, por ejemplo). El servidor procesa internamente esta petición y responde al cliente con el mensaje "BIENVENIDO AL SERVICIO".

- El cliente decide a nivel local qué plaza desea reservar (vamos a suponer la plaza "2C") y envía al servidor una petición "RESERVAR:2C". Recibida la petición, el servidor comprobará si esa plaza está disponible. Si lo está, responderá al cliente con el mensaje "RESERVADA:2C". En caso contrario, la respuesta enviada por el servidor informará de que la plaza está ocupada y contendrá también el estado actual del avión, representado como se explica en el siguiente ejemplo. La respuesta "PLAZA OCUPADA:XXXL-LXXL-XXXX-LLLL" informaría al cliente de que la plaza "2C" está ocupada (X) y que las plazas aún libres (L) son la "1D", "2A" y "2D", y toda la cuarta fila. Nótese que el estado se codifica como una cadena de caracteres, donde se utiliza un carácter para cada plaza (X y L, ocupado y libre, respectivamente), y la información de cada fila se separa por medio del carácter "-" (las filas se codifican por orden numérico, de la 1 a la 4). Con la información recibida, el cliente podrá decidir qué plaza alternativa reserva, invocando al servidor con la misma sintaxis.

- Un cliente reservará plazas mientras haya disponibles, conforme al modelo de interacción anterior. Cuando el avión esté completo el servidor responderá a una petición de reserva con la respuesta "VUELO COMPLETO". En ese instante, el cliente finalizará su ejecución, e informara por su salida estándar del número de plazas que logró reservar en total.

Desde el punto de vista del servidor, cada nuevo cliente que se conecta será atendido por un nuevo *thread* que se encarga de interactuar con el cliente para tramitar todas sus peticiones de reserva. Por otro lado, el cliente consta de un bucle en el que intenta reservar el número máximo de plazas posible, una a una, y conforme el protocolo descrito con anterioridad.

5.2. Tareas a realizar

En esta práctica tenéis que implementar un servidor capaz de atender a múltiples clientes conforme el protocolo descrito, usando comunicación síncrona

para el envío de los mensajes. El servidor deberá poder ejecutarse en cualquier máquina que esté conectada a la red. También deberéis programar un fichero `ejecuta_p5_SERVER.bash` que compile el servidor, a través de un `Makefile_p5_SERVER` que hayáis escrito previamente y, en caso de que la compilación haya sido correcta, lo ponga en funcionamiento en un puerto especificado como parámetro de invocación al script (por ejemplo, "`ejecuta_p5_SERVER.bash 2000`" pondrá el servidor a escuchar en el puerto 2000).

Los profesores programarán los clientes que serán utilizados durante la evaluación de la práctica. Para que estos clientes y el servidor puedan interaccionar es crucial que el alumno respete el protocolo especificado (formato de los mensajes, espacios en blanco, codificación de las plazas, etc.). No obstante, los alumnos deberán programarse también como parte del ejercicio un cliente de prueba que garantice el correcto funcionamiento de su servidor.

6. Entrega de la práctica

La práctica se realizará de forma individual. Cuando se finalice se debe entregar un fichero comprimido `practica5_miNIP.zip` (donde `miNIP` es el NIP del autor de los ejercicios) con el siguiente contenido:

1. Todos los ficheros fuente del servidor programado.
2. El fichero `ejecuta_p5_SERVER.bash` acorde a la especificación anterior.
3. Un fichero `Makefile_p5_SERVER` acorde a la especificación anterior.
4. Un documento en formato PDF, denominado `informe_p5.pdf` que debe contener la siguiente información:
   - el nombre completo del alumno y su NIP
   - una breve explicación de las decisiones de diseño adoptadas en su solución
   - una descripción de las principales dificultades encontradas para la realización de la práctica
   - una explicación del comportamiento observado en las ejecuciones del ejercicio
   - Los nombres de los ficheros fuente que conforman la solución solicitada.

Para la entrega del fichero `practica5_miNIP.zip` se utilizará el comando `someter` en la máquina `hendrix`. Los alumnos pertenecientes a grupos de prácticas cuya primera sesión de prácticas se celebra el día 28 de noviembre de 2018 deberán someter la práctica no más tarde del día 11 de diciembre de 2018 a las 23:59. Los alumnos pertenecientes a grupos de prácticas cuya primera sesión de prácticas se celebra el día 12 de diciembre de 2018 deberán someter la práctica no más tarde del día 25 de diciembre de 2018 a las 23:59.

Hay que asegurarse de que la práctica funciona correctamente en los ordenadores del laboratorio (vigilar aspectos como los permisos de ejecución, juego de caracteres utilizado en los ficheros, etc.). También es importante someter código

limpio (donde se ha evitado introducir mensajes de depuración que no proporcionan información al usuario). El tratamiento de errores debe ser adecuado, de forma que si se producen debería informarse al usuario del tipo de error producido. Además se considerarán otros aspectos importantes como calidad del diseño del programa, adecuada documentación de los fuentes, correcto formateado de los fuentes, etc.

Para el adecuado formateado de los fuentes, es conveniente seguir unas pautas. Hay varias, y es posible que podáis configurar el entorno de desarrollo para cualquiera de ellas. Una posible, sencilla de seguir, es la "Google C++ Style Guide", que se puede encontrar en

`https://google.github.io/styleguide/cppguide.html`

Alternativamente, cualquiera que uséis en otras asignaturas de programación.

7. Evaluación de la práctica

La evaluación de esta práctica será presencial, en la fecha y hora establecida con suficiente antelación por los profesores. Si un alumno no se presenta a la evaluación, su práctica será calificada como "no presentada".

El alumno deberá ejecutar su servidor en una máquina cualquiera del laboratorio L0.01 del edificio Ada Byron, utilizando los fuentes y el fichero `bash` que sometió en las fechas previstas. El servidor deberá ser capaz de atender, por lo menos, 5 clientes concurrentes, conforme el protocolo descrito en el enunciado. Los clientes utilizados en el proceso de evaluación serán proporcionados por los profesores. Durante la evaluación no se permitirá el uso de otros ficheros alternativos a los entregados, ni la modificación de estos. Por tanto, antes de someter la práctica os debéis asegurar que el servidor programado es correcto y funciona y se ejecuta conforma las pautas del enunciado.