Ejercicio evaluable 4 - Diseño de una aplicación distribuida

Sistemas distribuidos

Adrián Martínez Cruz 100451213

César López Navarro 100451326

Grupo 86

uc3m | Universidad Carlos III de Madrid

Introducción

En este ejercicio se diseñará una aplicación distribuida que gestiona el acceso a un aparcamiento público mediante varios computadores de puerta y un computador central, utilizando sockets para la comunicación.

Contexto Previo

El sistema consta de un computador central que gestiona el número de plazas libres y **N** puertas de acceso, cada una equipada con:

- Una barrera de acceso.
- Un botón de entrada.
- Una cámara para fotografiar matrículas.
- Una pantalla que muestra plazas libres.
- Un botón de salida.

Cada puerta debe:

- Permitir el acceso solo si hay plazas disponibles.
- Tomar una foto de la matrícula al entrar.
- Informar al servidor de entradas y salidas.
- Recibir del servidor las actualizaciones de plazas libres.

Funciones en la aplicación distribuida

El funcionamiento de la aplicación consistirá en la interacción entre cliente-servidor, es decir, computadores de las puertas-computador central.
El servidor manda a los computadores de las puertas el número actualizado de plazas libres cada vez que entra un vehículo al aparcamiento, para que las muestren en pantalla.
Además de esto, cada vez que entra un vehículo en el aparcamiento se toma una fotografía de la matrícula del vehículo y se envía al computador central para su procesamiento. El computador de cada puerta puede abrir la barrera para dejar acceso a un vehículo siempre que haya plazas en el interior.

A continuación se describe cada función.

vehicle_entry_request

Cuando un conductor pulsa el botón de entrada, la puerta solicita al servidor la posibilidad de acceso. Si hay espacios libres, se activa la barrera y se registra una foto de la matrícula del coche que entra.

vehicle_exit_request

Cuando un conductor pulsa el botón de salida, la puerta notifica al servidor que un vehículo ha salido. Se activa la barrera y se borra la matrícula de la lista de matrículas.

update_free_spaces

El servidor envía actualizaciones, cada vez que llega un vehículo al aparcamiento o sale de él, a todas las puertas con el nuevo número de plazas libres. Internamente, se actualiza la lista de las matrículas.

send_license_plate

La cámara hace una foto a la matrícula cada vez que entra un vehículo nuevo al aparcamiento y se envía al servidor para su procesamiento.

En todas estas funciones, aparte de los datos ya mencionados, se enviará también el código de operación representativo de cada una, de manera que el servidor pueda identificar qué funcionalidad ha sido solicitada.

Identificación de las partes cliente y servidor

El servidor sería el computador central que gestiona las plazas libres del aparcamiento y es el que manda la información a los demás computadores que están en las puertas.

Los clientes serían los computadores de las puertas que son las que se encargan de recibir la información actualizada del servidor con las plazas libres.

Por una parte, el cliente gestiona las solicitudes de apertura y cierre de puertas, toma las fotos de las matrículas de los coches que entran y notifica la salida de coches. Estos datos, junto al código de la función solicitada en la petición y otros tipos de información necesaria para cada tipo de función, serán enviados al sistema central. En este ejercicio evaluable, las puertas del aparcamiento se representarán como **PAN** (**P**uerta de **A**cceso nº **N**).

El sistema central será quien reciba y administre los datos y solicitudes mencionadas. Como se explica en el apartado anterior, hay cuatro tipos de funciones. Estas serán ejecutadas cuando el cliente envíe una petición cuyo primer dato será el código de la función solicitada. Cuando el servidor reciba esta petición, tomará los datos necesarios y ejecutará la funcionalidad deseada. El servidor se identificará como **CC** (**C**omputador **C**entral).

Peticiones, respuestas y secuencia de intercambio de mensajes

A continuación se describe el intercambio de mensajes entre puerta de entrada - computador central. Se recuerda que *cliente* es la puerta de entrada nº N y el *servidor* es el computador central.

vehicle_entry_request

```C/C++
struct vehicle_entry_request_args {
```

```
    int id_puerta;
};
```

- **Primer mensaje cliente -> servidor:** se enviará el código de operación 1 int op = 1, que será el código de operación de vehicle_entry_request.
- **Segundo mensaje cliente -> servidor:** struct vehicle_entry_request_args.
- **Tercer mensaje servidor -> cliente:** int resultado que podrá ser 0 o 1, dependiendo de si la operación se ha ejecutado correctamente o incorrectamente.

vehicle_exit_request

```C/C++
struct vehicle_exit_request_args {
    int id_puerta;
};
```

- **Primer mensaje cliente -> servidor:** se enviará el código de operación 2 int op = 2, que será el código de operación de vehicle_exit_request.
- **Segundo mensaje cliente -> servidor:** struct vehicle_exit_request_args.
- **Tercer mensaje servidor -> cliente:** int resultado que podrá ser 0 o 1, dependiendo de si la operación se ha ejecutado correctamente o incorrectamente.

update_free_spaces

```C/C++
struct update_free_spaces_args {
    int plazas_libres;
};
```

- **Primer mensaje servidor -> cliente:** se enviará el código de operación 3 int op = 3, que será el código de operación de update_free_spaces.

- **Segundo mensaje servidor -> cliente:** struct update_free_spaces_args.
- **Tercer mensaje cliente -> servidor:** int resultado que podrá ser 0 o 1, dependiendo de si la operación se ha ejecutado correctamente o incorrectamente.

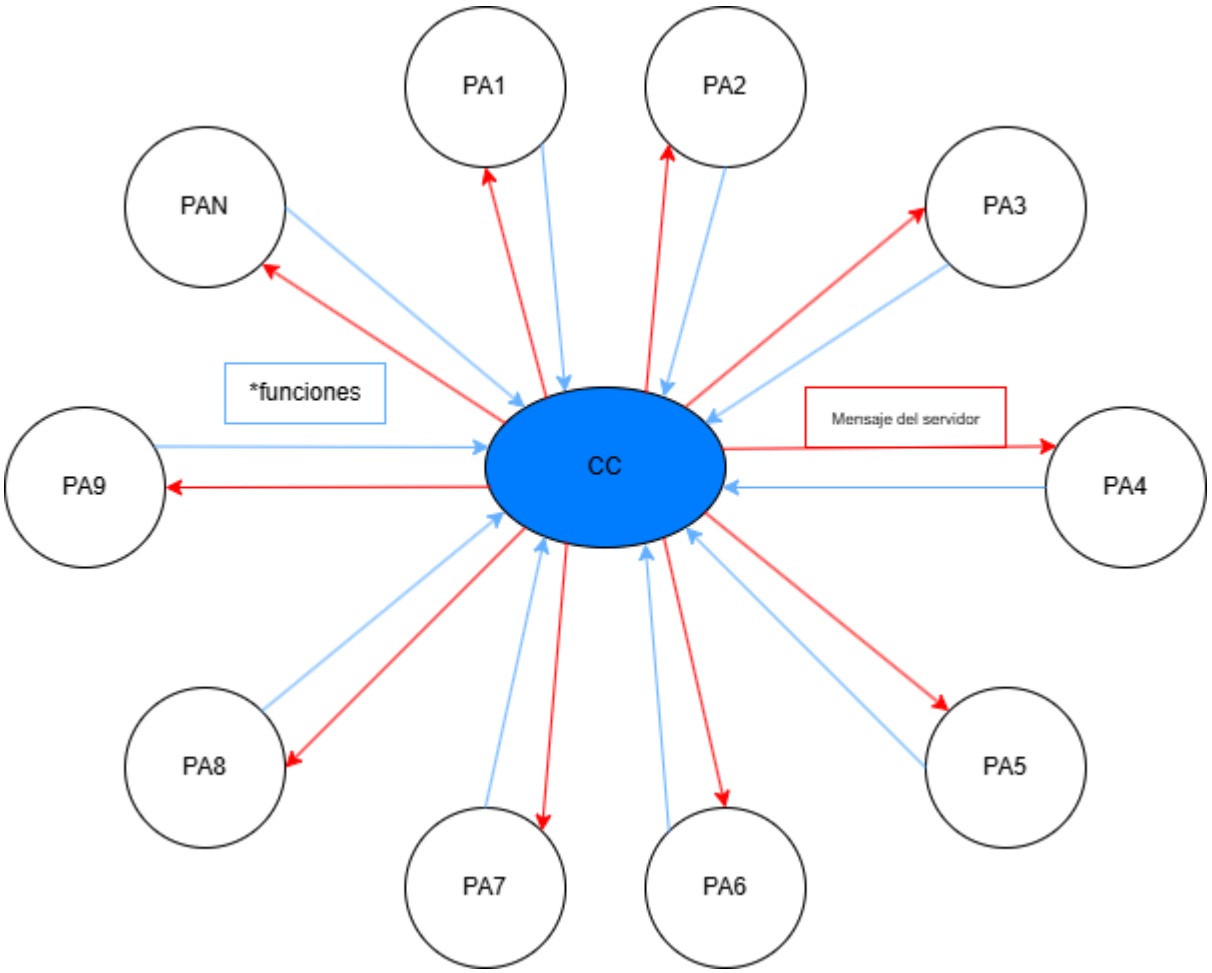
send_license_plate

```C/C++
struct send_license_plate_args {
    int id_puerta;
    char imagen_matricula[2048];
};
```

- **Primer mensaje cliente -> servidor:** se enviará el código de operación 4 int op = 4, que será el código de operación de send_license_plate.
- **Segundo mensaje cliente -> servidor:** struct send_license_plates_args.
- **Tercer mensaje servidor -> cliente:** int resultado que podrá ser 0 o 1, dependiendo de si la operación se ha ejecutado correctamente o incorrectamente.

Esquema gráfico

A continuación se encuentra un esquema gráfico de la comunicación que se lleva a cabo en esta aplicación:



Aclaraciones:

- CC significa Computador Central, y PAN (siendo N un número) corresponde a Puerta de Acceso nº N, como se aclara al comienzo del documento.
- ***funciones** consiste en el código de la función solicitada entre *vehicle_entry_request*, *vehicle_exit_request*, *update_free_spaces* y *send_license_plate* y los datos necesarios que han sido explicados previamente. En esas mismas flechas azules, el computador central enviará de vuelta el resultado de la operación.
- Las **flechas rojas** son exclusivamente para representar los mensajes que envía el servidor al cliente en la función *update_free_spaces*.

Protocolo de red del servidor

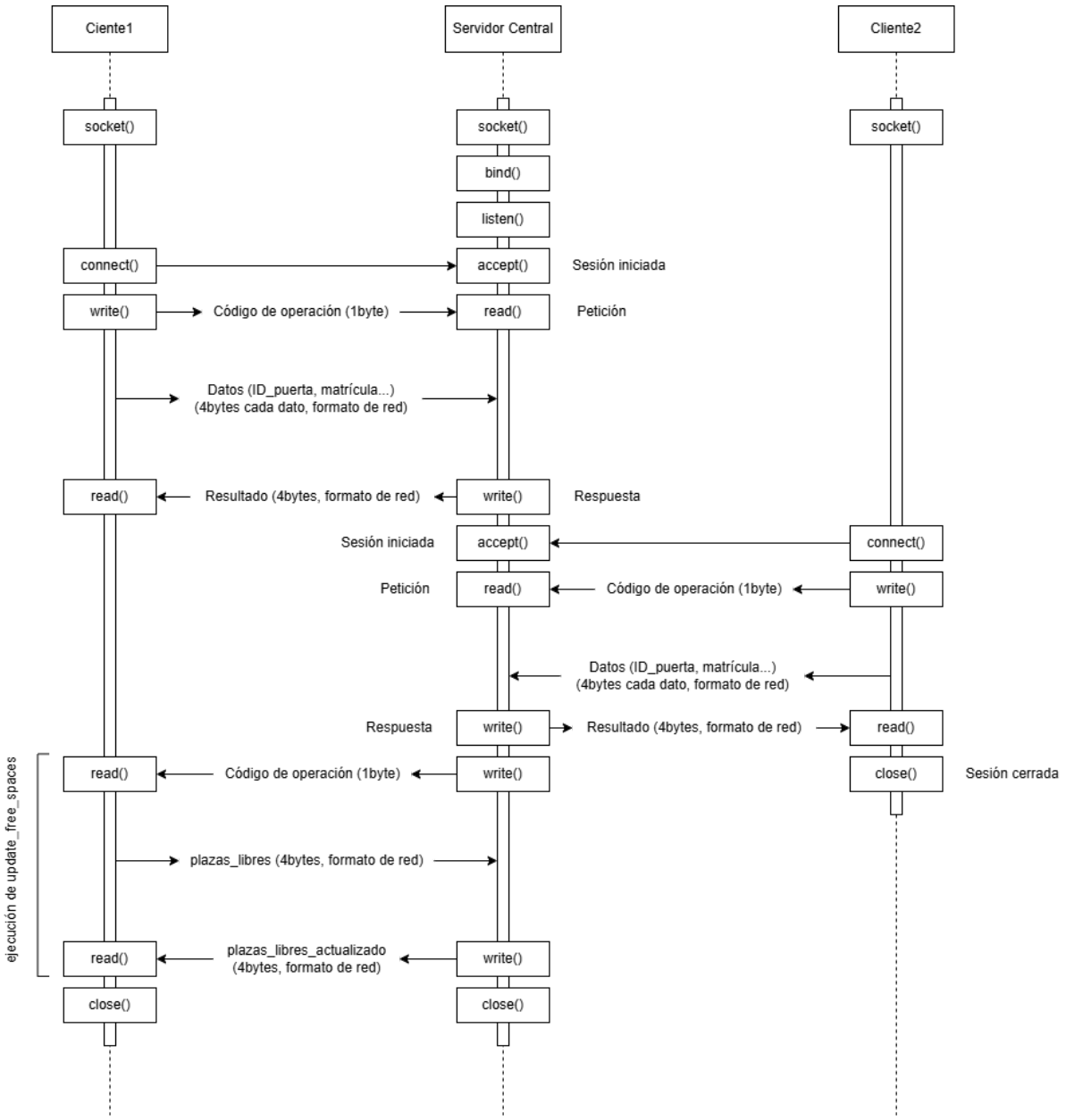
Dado que los datos que se intercambian no se deben perder, el protocolo de red escogido será **TCP**, para así asegurar un servicio orientado a conexión así como la retransmisión de paquetes perdidos, garantizar el orden de llegada de mensajes, y principalmente que los datos lleguen correctamente y completos a ambas partes.

Tendrá que haber obligatoriamente un socket abierto en el servidor todo el tiempo, dado que las puertas irán mandando continuamente mensajes de entrada o de salida de vehículos, así como las fotos de las matrículas de los vehículos. Las puertas también deberán tener un socket abierto todo el tiempo, ya que el servidor podrá enviarles la actualización de *update_free_spaces* en el momento en el que un vehículo entre por cualquier puerta.

Además, se considera necesario en el diseño del servidor la gestión de la conexión por sesiones, ya que en este caso los clientes solo pueden enviar peticiones cuando su sesión está iniciada. Se necesita una conexión abierta de manera constante para poder enviar correctamente las actualizaciones a las puertas de acceso.

Formato y paso de mensajes con API sockets

En este apartado se encuentra la representación gráfica de la secuencia de paso de mensajes que se lleva a cabo en esta aplicación, así como su formato y las llamadas al API de sockets:



Podemos observar que tanto servidor como cliente disponen de funciones pertenecientes tanto a procesos servidores como a procesos clientes.

Concurrencia

Ambas partes de la aplicación presentan concurrencia.

En primer lugar, el servidor contará con un hilo que permanecerá activo constantemente, esperando recibir solicitudes. Una vez que esto ocurra, generará

otro hilo paralelo e independiente que maneja la función de la solicitud de manera simultánea y se extinguirá al completar su ejecución.
Por otra parte, el cliente tendrá siempre un hilo en escucha para recibir mensajes de la función *update_free_spaces* por parte del servidor. Además, se crearán hilos cada cierto tiempo para enviar peticiones de las funciones *vehicle_entry_request*, *vehicle_exit_request* y *send_license_plate*.

Direccionamiento (IP y puertos)

El direccionamiento de esta aplicación será estático ya que las puertas deben comunicarse de forma constante con el servidor (por ejemplo, para enviar matrículas o recibir actualizaciones de plazas). Con direcciones IP fijas, el servidor sabe siempre a qué IP debe enviar las actualizaciones, evitando pérdidas o fallos de conexión.
Un buen ejemplo sería:
Puerto al que atiende el servidor:

- Puerto 8000, las peticiones de los clientes deben ejecutarse aquí.

DIrección IP del servidor:

- 192.168.1.101, todos los clientes deben conocer esta dirección.

Además, este tipo de direccionamiento asegura mayor seguridad, así como gran facilidad de configuración y monitoreo frente al direccionamiento dinámico.

Servidor con o sin estado

El servidor central se implementa como un servidor con estado, ya que necesita mantener información actualizada sobre el estado de las plazas del aparcamiento público. Esto permite una gestión eficiente del *parking* y una visión en tiempo real de la disponibilidad de las plazas.

Seguridad

Para proteger la integridad y la confidencialidad de los datos, se pueden implementar técnicas de seguridad como el cifrado de datos, los firewalls y la autenticación para que la comunicación sea segura y que solo los usuarios que estén autorizados sean capaces de acceder a la información.