SISTEMAS DISTRIBUIDOS

EJERCICIO 4 DISEÑO DE UNA APLICACIÓN DISTRIBUIDA

ID de grupo de prácticas: 81
Nombre de los participantes:
- Liang Ji Zhu
- Paolo Michael Webb

Correos electrónicos de los participantes:
- 100495723@alumnos.uc3m.es
- 100495955@alumnos.uc3m.es

Índice

1. Introducción

En los ejercicios 1-3 realizamos el paso de una aplicación monolítica a un servicio distribuido empleando tres tecnologías (colas de mensajes POSIX, sockets TCP y RPC). Gracias a eso, se establecieron los patrones como la separación proxy/servidor, el uso de mensajes autocontenidos y la validación de errores en los extremos.

En este ejercicio vamos a utilizar esos aprendizajes para diseñar un sistema que controla el acceso a un aparcamiento público, donde:

- Cada una de las N puertas dispone de un pequeño ordenador con botón de entrada / salida, barrera, cámara y pantalla.
- Un servidor central gestiona las plazas libres, valida cada acceso y distribuye la información actualizada a todas las puertas.

El ordenador central gestionará el número de plazas libres y notificará a todas las puertas los cambios en tiempo real. Además, recibirá las fotos de las matrículas en cada acceso.

2. Objetivos

2.1 Objetivo General

Diseñar una aplicación distribuida basada en sockets que permite controlar, en tiempo real y de manera segura, las operaciones de entrada y salida de vehículos de un aparcamiento público.

2.2 Objetivos Específicos

- Actualización inmediata del número de plazas libres en todas las puertas.
- Autorización local de la barrera solo cuando queden plazas.
- Trazabilidad de cada vehículo mediante la matrícula e instantánea asociada.
- Escalabilidad horizontal: añadir nuevas puertas sin modificar el núcleo.
- Robustez ante fallos de red mediante reintentos y reconexiones.
- Portabilidad: el protocolo se define con JSON para ser independiente del lenguaje.

3. Diseño

Cada puerta funciona como cliente que se conecta por TCP al servidor central. Cuando la puerta detecta una solicitud (pulsación de botón), envía un mensaje, recibe la respuesta y actúa sobre su barrera. El servidor mantiene el contador de plazas y emite avisos broadcast a todas las puertas cuando cambia el aforo.

3.1 Arquitectura Lógica

Con una topología de tipo estrella, el Servidor Central mantiene el estado global (plazas libres + histórico de matrículas) y se comunica por sockets TCP con cada doorClient:

- doorClient (x N puertas):

- Detecta pulsaciones ENTRADA / SALIDA.
- Envía mensaje PETICIÓN en JSON.
- Ejecuta la orden recibida (levantar barrera, mostrar aforo).
- Reintentar si no hay respuesta tras 2 s; reconecta cada 5 s ante fallo.

- centralServer:

- Procesa las PETICIONES secuencialmente (cola interna FIFO).
- Actualiza el contador de plazas (–1 / +1).
- Devuelve RESPUESTA al emisor y difunde AFORO al resto.
- Registra {timestamp, idPuerta, matrícula, tipo}.

3.2 Modelo de Datos Mínimo

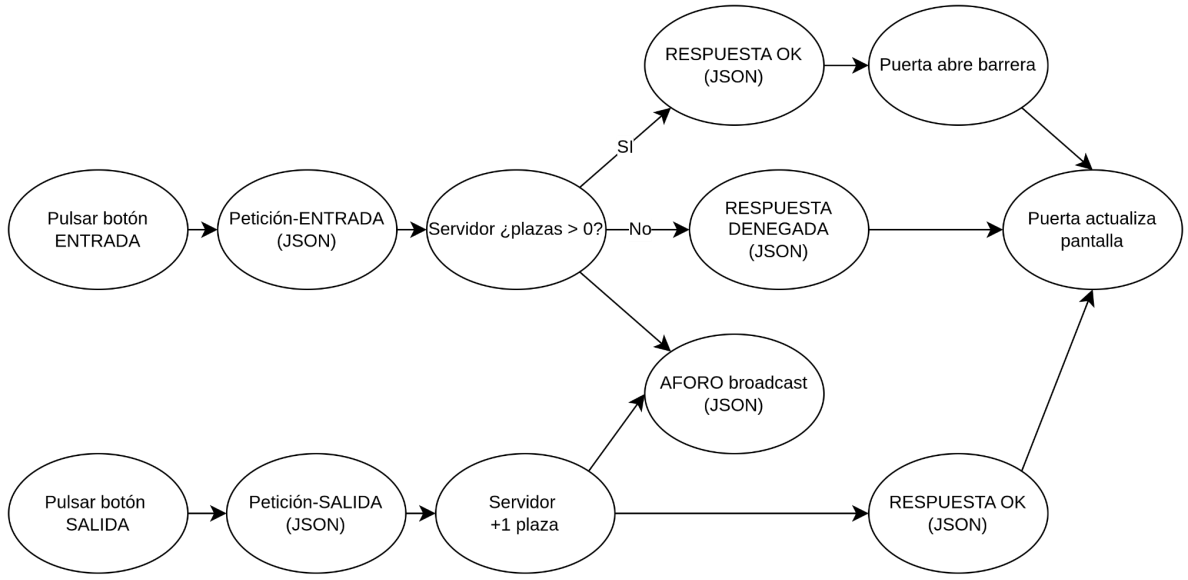
- plazasLibres: entero.
- peticiónAcceso: { tipo: “ENTRADA” | “SALIDA”, idPuerta, fotoBase64 }
- respuestaAcceso: { resultado: “OK” | “DENEGADO”, plazasActuales }
- notificaciónAforo: { plazasActuales }

3.2 Modelo de Comunicación

Mensaje (JSON)	**Quién → Quién**	**Campos clave**	**Comentario**
HELLO	Puerta → Server	idPuerta, versión	Handshake y sincronización de aforo
PETICIÓN	Puerta → Server	tipo (“ENTRADA” / “SALIDA”), foto64 (solo ENTRADA)	Solicitar permiso
RESPUESTA	Server → Puerta	resultado (“OK” / “DENEGADO”), plazasActuales	Acción inmediata
AFORO	Server → Todas	plazasActuales	Broadcast tras cada cambio
BYE	Puerta → Server	—	Cierre ordenado antes de apagar

3.3 Diagrama de Flujo

En el siguiente diagrama se resume la entrada y la salida de vehículos, así como la difusión de cambios en el aforo.



3.4 Gestión de Errores y Reconexión

- Reconexión: ante fallo de socket, la puerta intentará enviar HELLO cada 5 s hasta tener éxito.
- Timeout de respuesta: si la puerta no recibe RESPUESTA en 2s se aplica la PETICIÓN una vez más.
- Reconexión automática: tras 3 fallos, la puerta pasa a modo “offline” y muestra “Acceso manual” al operario.

3.5 Secuencias Habituales

1. Entrada de vehículo
    a. Conductor pulsa botón.
    b. doorClient envía PETICIÓN-ENTRADA con foto.
    c. Servidor comprueba plazas > 0 → responde OK y decrementa contador.
    d. Puerta sube barrera y muestra nuevo aforo.
    e. El servidor manda AFORO al resto de puertas.
2. Salida de vehículo
    a. Pulsación de botón de salida.
    b. doorClient envía PETICIÓN-SALIDA.
    c. Servidor responde OK, incrementa contador y emite AFORO.

4. Conclusión

El diseño propuesto cumple los requisitos esenciales y contiene las técnicas que aprendimos de las prácticas anteriores. Se ha definido la topología de tipo estrella y el uso de JSON con prefijo de longitud que facilitan la lectura, el mantenimiento y la portabilidad a otros lenguajes. Las puertas reciben el aforo en tiempo real, solo levantan la barrera cuando corresponde y almacenan todas las matrículas para lograr la trazabilidad. Añadir o quitar puertas no exige cambiar la lógica central; basta con que la nueva puerta conozca la IP y el puerto del servidor. Se definieron también reconexiones automáticas, reintentos con tiempo de espera y un modo “offline” visible al operario para evitar bloqueos del sistema. El actual protocolo está pensado para crecer (TLS, balanceo activo-pasivo, panel web de supervisión) sin romper la compatibilidad con nuestra versión inicial.