Fundamentos de los Sistemas Operativos
Ficha de entrega de práctica

**: campo obligatorio*
***IMPORTANTE:** esta ficha no debe superar las DOS PÁGINAS de extensión*

Grupo de prácticas*:14
Miembro 1: Luis Muñoz Sanz
Miembro 2: Jesús Toro del Dedo

Número de la práctica*: 3	**Fecha de entrega*:** 4 de mayo de 2025

Descripción del trabajo realizado*
Para este trabajo hemos desarrollado dos programas en C: sala.c, que contiene la lógica principal de gestión de la sala de teatro, y sala_app.c, que actúa como interfaz de línea de comandos y procesa los argumentos usando getopt().

En sala.c hemos implementado dos grandes tipos de operaciones: lectura y escritura sobre ficheros.

- En las operaciones de lectura (recupera_estado_sala y recupera_estado_parcial_sala), se abre el archivo en modo de solo lectura, se leen los datos de la cabecera (nombre, capacidad, etc.) validando cada paso, y luego se leen los asientos. Finalmente, se actualiza el estado en memoria y se cierra el fichero.
- En las operaciones de escritura (guarda_estado_sala y guarda_estado_parcial_sala), se abre el fichero en modo escritura. La versión completa puede truncar el contenido o crearlo de nuevo; la versión parcial escribe únicamente los datos deseados. En ambos casos, se escriben primero los datos de cabecera (ciudad, capacidad, libres) y luego los asientos.

Para gestionar el desplazamiento dentro del fichero usamos lseek(), calculando el offset correspondiente con funciones auxiliares como posicion_cursor_asiento() accediendo directamente a una posición específica del archivo sin necesidad de leerlo entero, lo que mejora la eficiencia.

El tratamiento de errores se realiza consistentemente mediante fprintf(stderr, ...), informando al usuario cuando ocurre una operación fallida, ya sea por errores de fichero, argumentos incorrectos o estado inválido de la sala.

En sala_app.c, usamos getopt() para parsear los argumentos desde línea de comandos. Dentro de cada comando (como crea, reserva, anula, etc.), las opciones pueden pasarse en cualquier orden, lo que mejora la robustez y usabilidad del programa.

También implementamos las funcionalidades extra propuestas en el reto:

- **anula -personas**: Recorre todos los asientos buscando coincidencias con los identificadores de persona indicados, y libera cada uno encontrado.
- **compara ruta1 ruta2**: Utiliza stat() para validar tamaño de ficheros y, si son iguales, compara byte a byte el contenido usando read() y memcmp(), devolviendo si son iguales o no.

Horas de trabajo invertidas* Miembro 1: 20 **Miembro2: 20**
(indicar las horas de todos los integrantes)

Cómo probar el trabajo*
Se deberá usar el comdando
gcc -o <nombre_compilado> sala_app.c sala.c
crear al menos una sala:
./<nombre_compilado> crea -f <nombre_sala> -c <capacidad>
ls <nombre_sala> //permite ver si se guardo el estado de la sala
y aplicar las diferentes operaciones
./<nombre_compilado> estado -f <nombre_sala>
Probar las diferentes funcionalidades pedidas en la ficha.
Para probar el reto:

Grupo de prácticas*:14 **Miembro 1: Luis Muñoz Sanz** **Miembro 2: Jesús Toro del Dedo**	
Número de la práctica*: 3	**Fecha de entrega*:** 4 de mayo de 2025
./<nombre_compilado> anula <nombre_sala> ***-persona*** ids ./<nombre_compilado> compara <nombre_sala1> <nombre_sala2>	
Incidencias *En el guión de la práctica hay una pequeña discrepancia en el reto, por un lado menciona que se debe eliminar por el id de la persona y posteriormente por el asiento. Hemos aplicado sobre el ID de la persona debido a que, para acceder al id de la persona era necesario saber acceder al asiento. No pudimos exponer la duda previamente porque nos dimos cuenta el día de la entrega.*	
Comentarios *En Compara si es 0 es que las salas son iguales si es 1 diferentes.* *Las pruebas realizadas sobre el trabajo están en el fichero pruebas.txt*	