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*: 44
Miembro 1: Kilian Armas Pérez

Número de la práctica*: 2	**Fecha de entrega*:** 03 de junio de 2023

Descripción del trabajo realizado*
Para la implementación de la gestión de reservas de los asientos de una sala de teatro se ha usado una estructura de datos basada en un vector, cuyo tamaño es la capacidad de la sala en cuestión y que representa tanto los asientos libres como los ocupados. El código de la práctica se ha repartido en tres ficheros .c: el primero de estos es ***main.c*** (programa principal de ejecución), el siguiente es ***sala.c*** (funciones básicas para el tratamiento de la sala) y el último es ***test.c*** (funciones de prueba más específicas). Además de estos ficheros, podemos encontrar dos ficheros .h (***sala.h*** y ***test.h***), que se encargan de la implementación de las funciones tanto de sala.c como de test.c.

Además de todas las funciones implementadas anteriormente en el fichero ***sala.c***, se han agregado 6 nuevas funciones para poder almacenar la información de la sala en un fichero .txt. En primer lugar, se halla la función ***guarda_estado_sala(char* ruta_fichero)***, a la cuál se le pasa como parámetro el nombre del archivo en el que se desea guardar la información. Para poder recuperar dicha información se dispone de ***recupera_estado_sala(char* ruta_fichero)***, que abre el archivo pasado como parámetro y sobrescribe el estado de la sala actual con el de la almacenada en el fichero *(para actualizar la ocupación de la sala se llama a la función **estado_sala_recuperada()**)*. Ambas funciones para guardar y recuperar, tiene sus propias versiones parciales (***guarda_estadoparcial_sala()*** y ***recupera_estadoparcial_sala()***), cuyo propósito es guardar o recuperar el estado de un conjunto de asientos. Además, se ha creado una función para recuperar la capacidad de la sala guardada en el fichero, ***recupera_capacidad()***.

Con la finalidad de facilitar el tratamiento de órdenes y argumentos se ha decidido utilizar getopt, más concretamente ***getopt_long()***, lo cual, además del tratamiento de órdenes cortas como *"-f"* o *"-c"*, permite el tratamiento de órdenes largas como *"--crea"* o *"--reserva"*. Para poder identificar las órdenes que han sido ejecutadas en cada ejecución, se ha utilizado un ***switch()*** con todos las posibles órdenes para activar sus correspondientes flags en cada momento *(por ejemplo: cuando se llama a "-a", **"flag_a"** se pone a uno; o cuando se llama a "--anula", la variable **flag** se iguala al índice de dicha orden (3))*. Gracias a todo esto, se han podido implementar todas las órdenes requeridas y las correspondientes al reto.

Horas de trabajo invertidas* Miembro 1: 30 horas

Cómo probar el trabajo*
Con el propósito de compilar el código propuesto en esta práctica, se deberá generar un ejecutable con un nombre definido por el usuario (en este caso se ha optado por el nombre *"misala"*) y posteriormente ejecutarlo haciendo uso de las siguientes instrucciones:

- ***gcc -fcommon main.c sala.c test.c -o misala***
- ***./misala orden argumentos***

En primer lugar, se puede apreciar la orden correspondiente a la creación de una nueva sala y a su posterior guardado en un fichero. Existen dos opciones diferentes para ejecutar esta orden, la primera es:

- ***./misala --crea -f ruta -c capacidad***

Esta última solo funciona si el archivo seleccionado en *"ruta"* no existe, en caso de que exista se imprime un mensaje de error por stderr. Para evitar esto se hace uso de la orden *"-o",* que en caso de que el fichero ya exista se sobrescribe su información:

- ***./misala --crea -f ruta -o -c capacidad***

Grupo de prácticas*: 44
Miembro 1: Kilian Armas Pérez

Número de la práctica*: 2	**Fecha de entrega*:** 03 de junio de 2023

En segundo lugar, se halla la orden cuya finalidad es la de reservar N asientos para N personas en la sala guardada en *"ruta"* con sus respectivos identificadores:

- ***./misala --reserva -f ruta -n número_de_asientos id_persona1 id_persona2 …***

En tercer lugar, está la orden para liberar N asientos de la sala almacenada en *"ruta"*. Hay dos alternativas para ejecutar esta orden, una que utiliza los identificadores de los asientos para liberarlos y otra que usa los identificadores de las personas para realizar la liberación:

- ***./misala --anula -f ruta -a id_asiento1 id_asiento2 …***
- ***./misala --anula -f ruta -i id_persona1 id_persona2 …***

A continuación, se observa una orden que sirve para visualizar el estado de la sala recuperada de *"ruta"*:

- ***./misala --estado -f ruta***

La siguiente orden se utiliza para comparar si dos salas (almacenadas en *"ruta1"* y *"ruta2"*) son iguales, es decir, tienen el mismo tamaño y el estado de cada uno de sus asientos es el mismo en ambos casos:

- ***./misala --compara ruta1 ruta2***

Por último, se ha implementado una nueva orden para poder ejecutar los tests propuestos desde la entrega anterior, además de dos nuevos tests para probar las funciones de guardado y recuperación, tanto al completo (***test_GuardadoCompleto()***) como de manera parcial (***test_GuardadoParcial()***). Para ejecutar cada test se deberá pasar un identificador de la prueba a realizar:

- ***./misala --test id_prueba***

Incidencias

Todos los errores y anomalías que han ido apareciendo a lo largo del desarrollo de la práctica, de sus pruebas y de la ejecución de las diferentes órdenes implementadas han sido solucionados. Además se han realizado las correcciones pertinentes a raíz de las observaciones de las anteriores entregas.

Comentarios

Con el fin de ayudar a la ejecución de los tests, a continuación se proporcionan los identificadores de cada una de las pruebas: test_ReservaBásica = 0, test_ReservaSimple = 1, test_ReservaMultiple = 2, test_SalaVacía = 3, test_SalaLlena = 4, test_ReservaNegativa = 5, test_ReservaTeclado = 6, test_GuardadoCompleto = 7 y test_GuardadoParcial = 8.

Se han extraído las funciones internas del fichero .h y se ha eliminado la conversión a char de la sala. A partir de ahora se comprueba si la llamada al sistema ***stat()*** en la función ***tamaño_bloque()*** da error y en caso de error ya no se aborta el programa con exit sino que se retorna un -1. Además, ya no se crea una nueva sala dentro de la función ***recupera_estado_sala()***, en cambio, se sobrescribe la sala actual si es posible. Por último, se ha empaquetado el código del main en funciones para hacerlo más modular.

Se ha cambiado la expresión errónea *"buffer+i"* de las escrituras y lecturas en el fichero, que se incrementaba en el número de bytes que ocupa un bloque, por la expresión *"buffer+i/sizeof(int)"*, que ahora se incrementa de manera, es decir, en el número de asientos que hay en un bloque. Además, ahora ya se comprueba la compatibilidad de los tamaños de la sala almacenada y en el fichero a la hora de hacer un guardado parcial, y también se comprueba si los id de los asientos que se van a escribir o leer son correctos (ni negativos ni mayores que la capacidad de la sala) en ambas funciones parciales.

Por último, a partir de esta actualización a la hora de reservar por línea de comandos, ya no se guarda el estado de toda la sala, sino que solo se guarda el estado de los asientos reservados. Del mismo modo, a la hora de anular asientos por línea de comandos se guarda el estado de forma parcial, solo de los asientos que se han modificado, y no se guarda el estado de toda la sala al completo. Por último, se ha sustituido ***ftruncate*** por ***lseek*** en la función ***guarda_estado_parcial()***, para que a la hora de situar el puntero en el fichero no se pierdan los datos almacenados en posiciones superiores.