Fundamentos de los Sistemas Operativos
Ficha de entrega de práctica

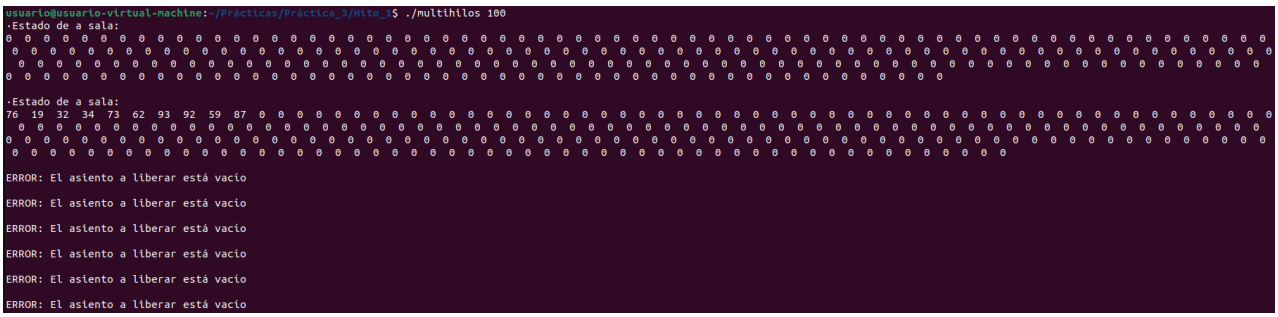
Grupo de prácticas*: 44
Miembro 1: Kilian Armas Pérez

Número de la práctica*: 3	**Fecha de entrega*:** 5 de junio de 2023

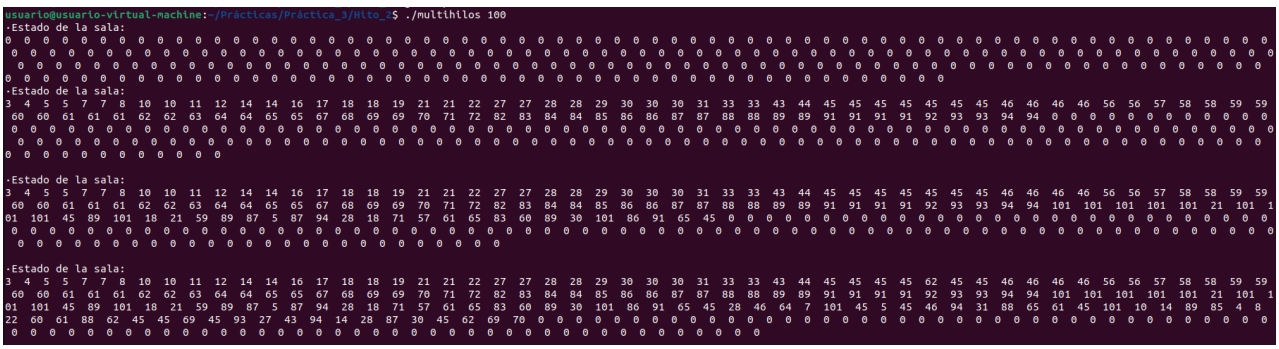
Descripción del trabajo realizado*
(Algoritmos que han implementado, llamadas al sistema utilizadas, estrategia que han seguido para resolver el problema, etc.)

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, desde donde se crea la sala y se lanzan los hilos), el siguiente es ***sala.c*** (funciones básicas para el tratamiento de la sala). Además de estos ficheros, podemos encontrar un ficheros .h (***sala.h***), que se encarga de la implementación de las funciones de ***sala.c***, contando con todas las funciones desarrolladas anteriormente en las prácticas 1 y 2.

En el primer hito, se ha logrado evidenciar el problema de la sección crítica en el acceso compartido a la sala por múltiples hilos en ejecución que realizan 3 reservas y 3 liberaciones cada uno, así como otro hilo que se encarga de mostrar el estado de la sala cada cierto tiempo. Esto se ha conseguido forzando a que antes de completar la reserva o la liberación haya un retardo aleatorio de máximo 0,5 segundos. Lo que permite que dos hilos seleccionen el mismo asiento para reservarlo, sobrescribiendo uno la reserva del otro, lo que a la hora de liberar resulta en dos intentos de liberación en el mismo asiento, uno terminando satisfactoriamente y el otro generando una situación de error que nos indica que existe algún tipo de corrupción en los datos.



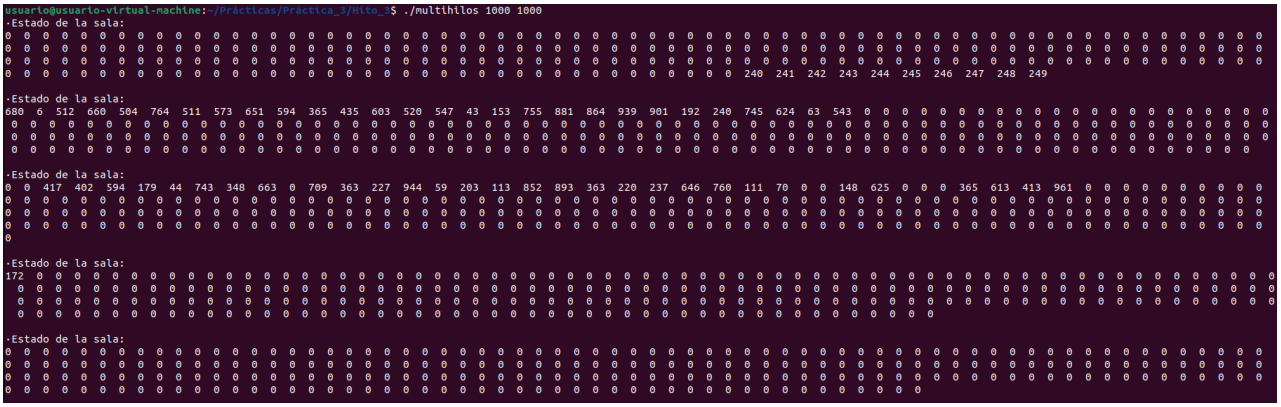
Para solucionar este problema, en el hito 2 se han realizado las modificaciones pertinentes para conseguir que la API de la sala sea ***thread-safe***, es decir, eliminar la posibilidad de que aparezca cualquier inconsistencia de datos por el acceso concurrente de diversos hilos. Para alcanzar dicho objetivo se han empleado los cerrojos o ***mutex***, de la librería ***pthread***, para bloquear el acceso al código de la sección crítica a múltiples hilos concurrentemente (***pthread_mutex_lock()*** y ***pthread_mutex_unlock()***), siendo el primero en llegar el único que obtiene acceso, hasta que este termina y libera la sección para que otro hilo puede entrar y ejecutar dicho código. Evitando de este modo que dos hilos seleccionen el mismo asiento para reservarlo, y por consiguiente que se intenten liberar asientos no ocupados.



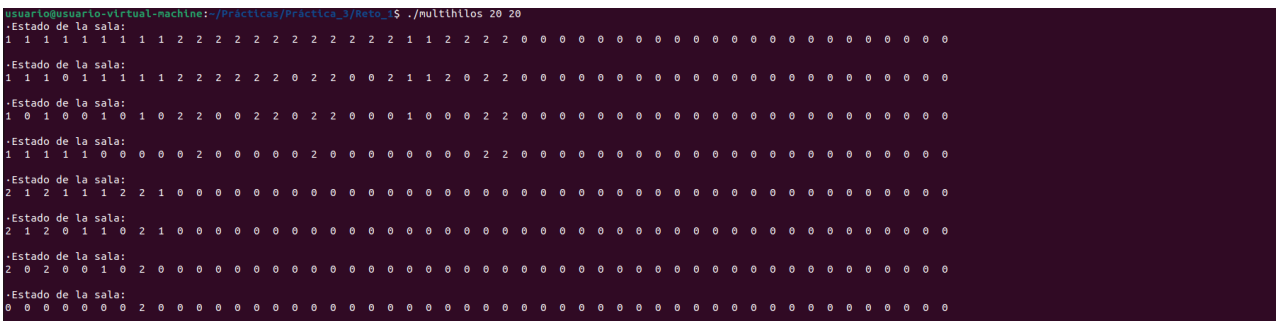
Grupo de prácticas*: 44
Miembro 1: Kilian Armas Pérez

Número de la práctica*: 3	**Fecha de entrega*:** 5 de junio de 2023

Adicionalmente, en el hito 3 se ha mejorado la API de la sala para lograr que un hilo al reservar o liberar se quede en espera hasta que haya asientos libres o hasta que haya asientos ocupados, respectivamente. Este propósito se ha alcanzado gracias a la utilización de esperas condicionales, mediante las funciones ***pthread_cond_wait()*** y ***pthread_cond_signal()***. Estas funciones se usan junto con dos variables condición, ***cond_reserva*** y ***cond_libera***. La función ***wait()*** se utiliza tanto en ***reserva_asiento()*** como en ***libera_asiento()*** en un bucle while (***while (libres == 0)*** en el caso de reservar y ***while (ocupados == 0)*** en el de liberar). Si una sala está vacía y un hilo desea liberar un asiento, primero se deberá reservar un asiento para que se desbloquee el hilo de liberación; lo mismo ocurre en el caso de que un hilo quiera reservar un asiento en una sala llena, primero debe liberarse un asiento para que el hilo de reserva sea desbloqueado.



Por último, en el reto 1, se ha modificado la API para identificar entre reservas realizadas por hombres y mujeres, para así poder hacer un control del sexo, impidiendo que a la hora de realizar una reserva ningún sexo quede por encima del 60% de la ocupación actual. Para implementar esto se hace uso de dos variables globales ***"hombres"*** y ***"mujeres"***, que llevan la cuenta de cuenta de las reservas de cada sexo, y una variable de tipo int llamada ***"ratio"*** que cuando el id es 1 (hombre) se iguala a la expresión ***hombres+1 > (ocupados+1) * 0.6***, mientras que cuando el id es 2 (mujer) se iguala a ***mujeres+1 > (ocupados+1) * 0.6***. Además, se añade una nueva condición al bucle que se encuentra en reserva_asiento(), de tal forma que se quedaría como: ***while (libres == 0 || (ocupados+1 >= 10 && ratio))***. Gracias a esto, la reserva en cuestión se quedará en espera mientras no haya asientos libres, o mientras la ocupación sea mayor o igual que 10 y se cumpla que el sexo del que se desea realizar la reserva pase a ocupar más del 60%.



Horas de trabajo invertidas* Miembro 1: 28 horas *(indicar las horas de todos los integrantes)*

Cómo probar el trabajo*
(qué debe hacer el profesor para utilizar el programa entregado: nombre de los programas, instrucciones de compilación, opciones de menú, datos de prueba, argumentos de invocación al programa, etc.)

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 *"multihilos"*) mediante el siguiente comando por línea de órdenes:

- ***gcc -lpthread -fcommon main.c sala.c retardo.c -o multihilos***

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

Número de la práctica*: 3	**Fecha de entrega*:** 5 de junio de 2023

Para ejecutar tanto el hito 1 como el hito 2 se deberá hacer uso de la siguiente instrucción:

- ***./multihilos n***

Donde n es el número de hilos que se desean lanzar a ejecución. A diferencia de los hitos 1 y 2, el hito 3 se ejecuta mediante una instrucción distinta, la cuál es:

- ***./multihilos n m***

Donde n es el número de hilos de reserva que se desean lanzar y m el número de hilos de liberación que se quieran mandar a ejecutar. Por otro lado, para ejecutar el reto 1 se usará la instrucción:

- ***./multihilos nhombres nmujeres***

Donde nhombres es el número de hilos hombres que se deben lanzar a ejecución y nmujeres el número de hilos mujeres que se deben mandar a ejecutar.

Incidencias
(errores no resueltos, anomalías, cualquier cosa que se salga de lo normal)

Todos los errores y anomalías que han ido apareciendo a lo largo del desarrollo de la práctica, de sus pertinentes pruebas y de la ejecución de los diferentes hitos desarrollados han sido solucionados, además de que se han realizado todas las correcciones pertinentes en relación a las observaciones de la primera entrega.

Comentarios
(observaciones adicionales que quieran anotar)

Como observación adicional, hay que destacar que en el hito 2 y por consiguiente en el hito 3, no solo se han protegido las funciones principales como lo son la reserva, la liberación y el muestreo del estado de la sala, sino que también se han asegurado las secciones críticas del resto de funciones de la API, así como las de tratamiento de ficheros (***guarda_estado_sala()***, ***recupera_estado_sala()***, etc.).

A raíz de las observaciones de la primera entrega se han realizado los cambios pertinentes. En primer lugar, ahora ya se comprueba que las funciones ***lock***, ***unlock***, ***wait*** y ***signal*** funcionen correctamente y no se asume que lo hagan desde un inicio. En el hito 2, se ha incluido dentro de la protección del cerrojo la consulta a la variable ***"ocupados"*** que anteriormente se situaba fuera de esta.

Por último, a partir de esta revisión, en el hito 3 se han resituado las operaciones de signal en las funciones reserva_asiento() y libera_asiento() para que se realicen después del código que reserva o libera un asiento, respectivamente. Además se ha creado una nueva función ***destroy()*** que libera los recursos (mutex y variables condición) al finalizar el programa, mediante el uso de las operaciones ***pthread_mutex_destroy()*** y ***pthread_cond_destroy()***.