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*:
Miembro 1: Romen Adama Caetano Ramirez

Número de la práctica*: 3	**Fecha de entrega*:** 17 de Mayo

Descripción del trabajo realizado*

El objetivo de esta actividad es iniciarte en la programación concurrente, a través de la creación de hilos que tendrán que coordinarse entre sí para poder resolver una misión común.

Hito 1 (Problema de la Sección Crítica):
En este hito, se debe evidenciar el problema de la sección crítica en la implementación actual de la API, donde múltiples hilos acceden concurrentemente a la estructura de datos que guarda el estado de los asientos de la sala.
Archivo "multihilos.c":

- Implementa la función principal "main":
  - Verifica la cantidad de argumentos pasados por la línea de comandos y muestra un mensaje de uso si no es correcta.
  - Crea una sala con capacidad para 30 asientos.
  - Crea un hilo de visualización del estado de la sala.
  - Crea hilos de reserva y liberación de asientos según el número especificado por el usuario.
  - Espera a que los hilos de reserva y liberación terminen.
  - Detiene el hilo de visualización del estado de la sala.
  - Elimina la sala.

Hito 2 (API Thread-Safe):
En este hito, se debe modificar la implementación de la API para que sea thread-safe, es decir, garantizar que no se generen inconsistencias en los datos debido al acceso concurrente por parte de múltiples hilos.

Para lograr una implementación "thread-safe", se utiliza un mutex (bloqueo) para controlar el acceso a la estructura de datos que guarda el estado de los asientos.

- *Se agregó la declaración y la inicialización del mutex `sala_mutex`:*
  - *pthread_mutex_t sala_mutex = PTHREAD_MUTEX_INITIALIZER;*
- *2. En las funciones `realizar_reservas` y `realizar_liberaciones`, se bloquea el mutex antes de acceder a la sala y se desbloquea después de realizar las operaciones correspondientes:*
  - *pthread_mutex_lock(&sala_mutex); // Bloquear el mutex antes de acceder a la sala*
  - *pthread_mutex_unlock(&sala_mutex); // Desbloquear el mutex después de acceder a la sala*
- *3. En la función `visualizar_estado`, también se bloquea y desbloquea el mutex antes y después de acceder a la sala, ya que también es una operación crítica:*
  - *pthread_mutex_lock(&sala_mutex); // Bloquear el mutex antes de acceder a la sala*
  - *pthread_mutex_unlock(&sala_mutex); // Desbloquear el mutex después de acceder a la sala*

Hito 3 (Espera Condicional):
Archivo "multihilos.c":

- Se utilizan variables globales para el mutex de la sala y las variables de condición.
- Se declara una variable global "asientos_ocupados_hilos" para llevar la cuenta de los asientos ocupados.
- La función "realizar_reservas" se encarga de realizar reservas de asientos de forma aleatoria.
- La función "realizar_liberaciones" se encarga de realizar liberaciones de asientos de forma aleatoria.
- Se definen las funciones auxiliares "reservas_wrapper" y "liberaciones_wrapper" para envolver las funciones principales y pasarlas como argumentos a los hilos.
- Se define la función "visualizar_estado" que muestra el estado actual de la sala.
- En la función principal "main":
  - Se capturan los argumentos de línea de comandos para el número de hilos de reserva y liberación.

Grupo de prácticas*:
Miembro 1: Romen Adama Caetano Ramirez

Número de la práctica*: 3	**Fecha de entrega*:** 17 de Mayo

- Se crea la sala con capacidad para 30 asientos.
- Se crea un hilo para la visualización del estado de la sala.
- Se crean los hilos de reserva y liberación según los argumentos ingresados.
- Se utilizan las funciones pthread_join() para esperar a que los hilos de reserva y liberación terminen su ejecución.
- Se cancela el hilo de visualización y se espera a que termine su ejecución.
- Se destruyen el mutex y las variables de condición.

- El programa finaliza con un código de retorno 0.

Horas de trabajo invertidas* Miembro 1: 30 horas

Cómo probar el trabajo*

- Descomprimir él .zip y mediante comandos de terminal (cd) desplazarse hasta el directorio:
- ***…/pract3*** (seguramente sea el directorio por defecto al descomprimir)

Ejecutar el script desarrollado para facilitar la manipulación del programa:
(se encarga de compilar los archivos en caso de alguna modificación en los mismos)

- ***bash script.sh***

Dirigirse hacia el hito a trabajar:

- ***cd …/pract3/pract3_hito1***

Una vez ahí ejecutar:

- ./hito1 n

Incidencias
Como no he tenido una revisión hasta el momento en que entrego de la practica anterior, he intentado trabajar con lo que tenía para realizar la práctica.
Además, en esta semana de entrega han habido numerosos exámenes y trabajos y no he podido dedicarle todo el tiempo necesario para terminar de pulir y debugear la aplicación.
Los programas funcionan y utilizan lo solicitado a desarrollar, pero estoy seguro de que esta por pulir los programas entregados.

Comentarios
Para ver el contenido completo y el desarrollo evolutivo ver el GitHub personal.
https://github.com/Romen-Adama-Dev/FSO