Universidad Nacional Autónoma De México
Facultad De Ciencias

# Computación Concurrente

## *Práctica Reposición*


Fecha:

13 de Mayo del 2019

Nombre:

Muñiz Patiño, Andrea Fernanda.
No. de cuenta 311175318

# 1. Problema a resolver

## 1.1. Conceptos previos

- **Exclusión mutua:** Para resolver el problema de la exclusión mutua se debe cumplir con las siguientes 3 condiciones:
  - Mutex: En cualquier momento hay como máximo un proceso en la sección crítica
  - Garantía de entrada: cada hilo/proceso que tenga que entrar a la sección crítica, lo logrará eventualmente.
  - Ausencia de Deadlocks: No habrá ningún bloqueo, es decir aunque varios procesos intenten entrar a la sección crítica un proceso lo lograra.
- **Semáforo:** Estructura de datos concurrente que consta de un contador, una cola de procesos y tiene las operaciones *wait* y *signal*

## 1.2. Definición y solución del problema: Lectores - Escritores

Este problema modela el acceso a una base de datos. Donde tenemos dos usuarios, los lectores y los escritores.

Los lectores buscan únicamente consultar/leer la base de datos. Los escritores buscan modificar esta base de datos para actualizar algún dato.

Varios lectores pueden estar consultando la base de datos, pero solo puede un escritor estar actualizando, ya que pensemos en el caso en que dos escritores quieren actualizar una misma tabla, actualizan la misma fila de la base pero con datos diferentes, ¿Qué datos se deben guardar?, ¿Cómo decidir que dato guardar de manera definitiva?, esto es un conflicto que puede decirse que proviene de aplicar escritura concurrente donde no debería.

En el programa entonces admitiremos a varios lectores a la vez, pero solo un escritor modificando datos a la vez.

Para garantizar que un escritor haga los cambios correspondientes se deben de definir *prioridades*, digamos cuando estén los lectores en la base de datos e intente un escritor actualizar la base, entonces este debe esperar a que los lectores acaben su consulta, para luego llevar acabo los cambios. Pero ¿Qué pasa si mientras el escritor espera que terminen los lectores su consulta, llegan más lectores? ¿También hay que esperarlos? Si fuera así no se podría garantizar la entrada a los escritores a la base de datos y entonces no estaríamos cumpliendo una de las 3 condiciones de la Exclusión mutua que debe tener la solución de nuestro problema.

Sin embargo esto es tener menor concurrencia y por ende un menos rendimiento, pero dado el problema tiene que hacerse de esa manera.

# 2. Observaciones

Los semáforos deben ser globales por que se llamarán las funciones *wait* y *signal* desde distintos procesos, en la biblioteca *semaphore*, son las funciones: *wait* y *post*.

Sea $l$ el número de lectores en el programa y $e$ el número de escritores.

De manera arbitraria se establece que los hilos en el intervalo

$$0 \leq id\_Hilo < l$$

serán los hilos en representación de los lectores y los hilos en el intervalo

$$l < id\_Hilo < l + e$$

serán los escritores.
Esto además por que el identificador de los hilos inicia en 0.

Para las ejecuciones de prueba se establece que el número de veces que itera cada hilo es la CONSTANTE 4, es decir cada hilo itera 4 veces.

Haciendo varias pruebas, me he percatado que la implementación hace que los escritores tengan acceso a la base hasta que los lectores han terminado de acceder a la "base de datos".

A continuación se anexan algunos ejemplos:

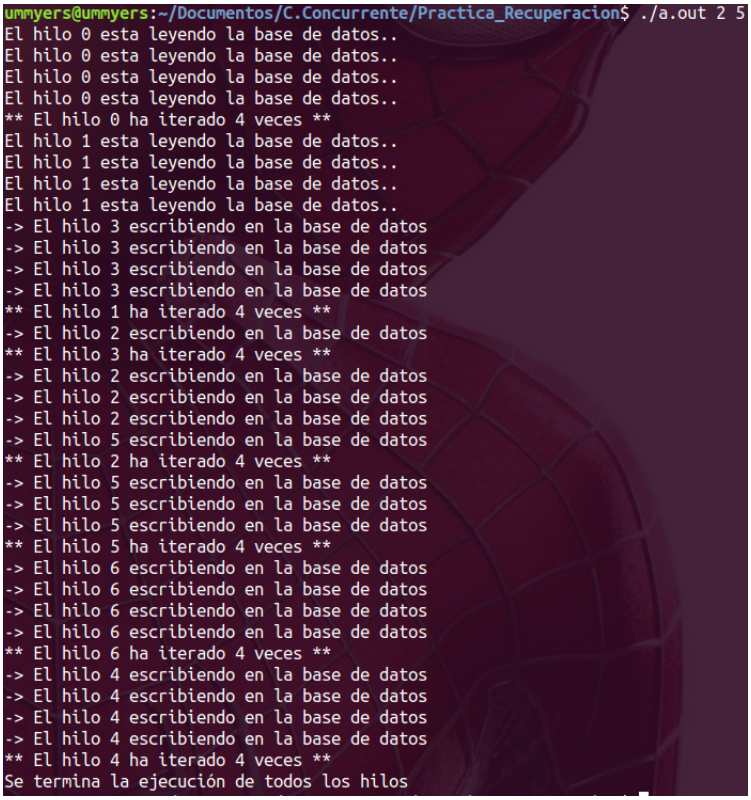


Figura 1: Donde es mayor el número de escritores

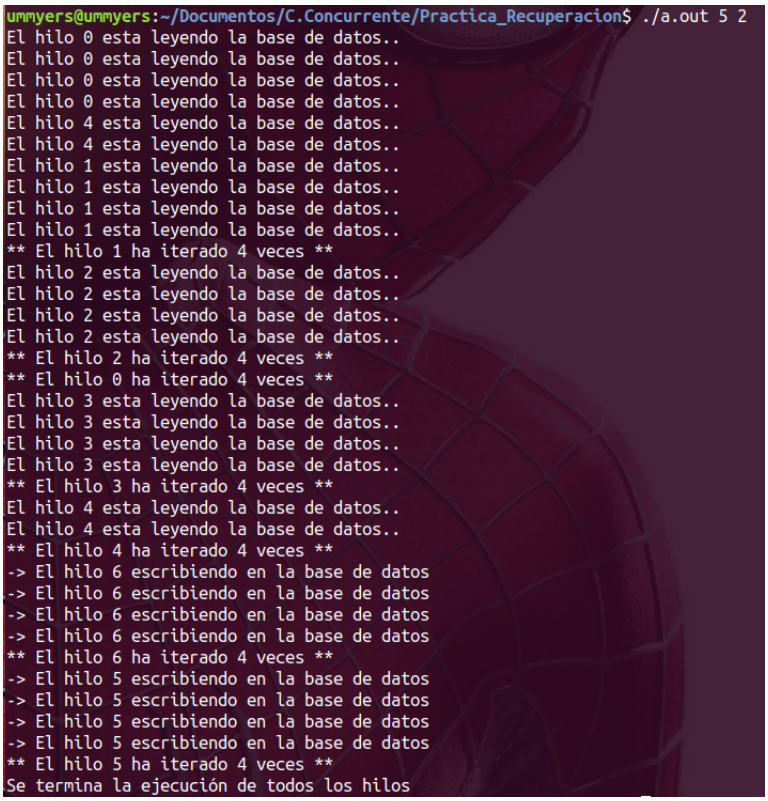


Figura 2: Donde es mayor el número de lectores

Observemos que aunque sea mayor el número de escritores o mayor el número de lectores, no hay *deadlocks* y todos los hilos hacen su ejecución. Es decir todos tienen acceso a la sección crítica.

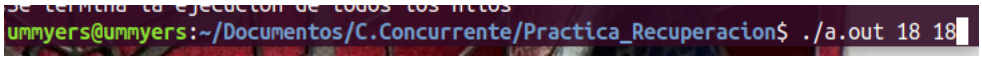


Figura 3: Ejecución con parámetros mayores

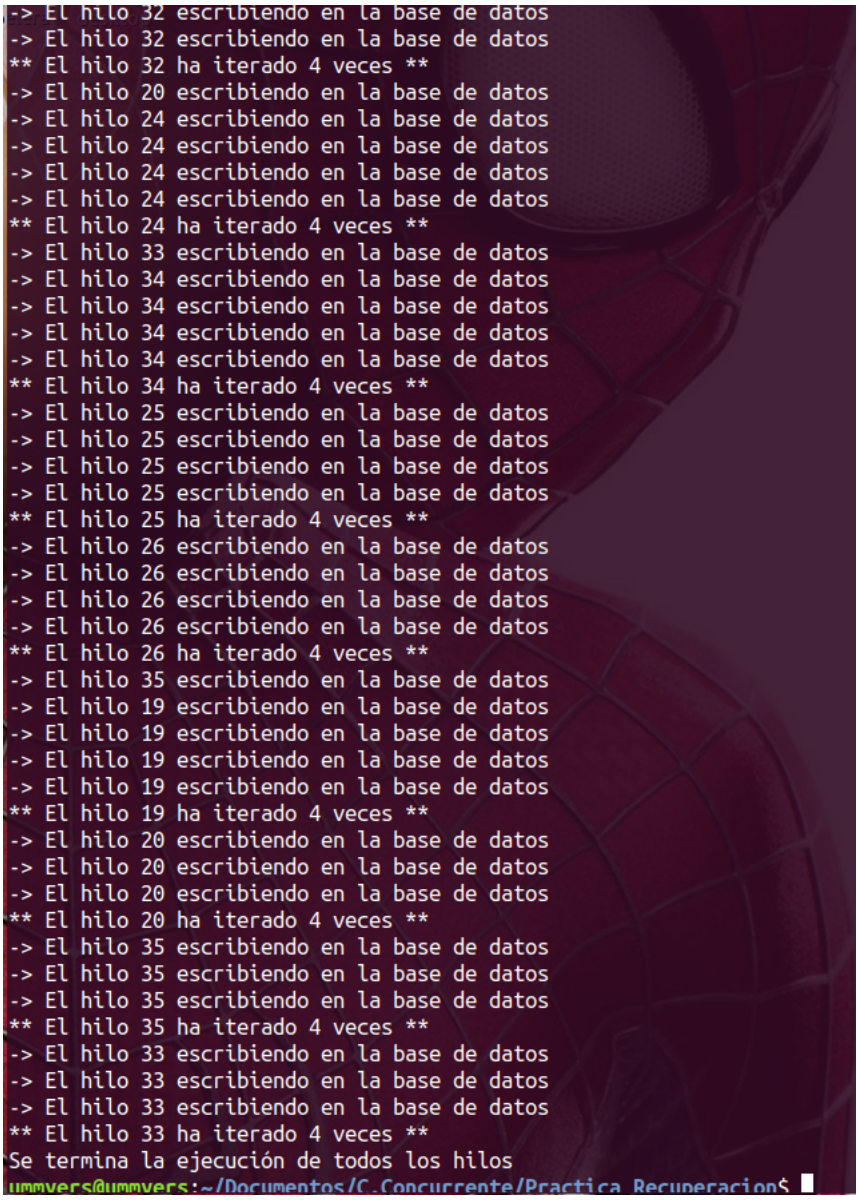


Figura 4: Ejecución con parámetros, lectores 18 y escritores 18

Como podemos observar los escritores con el *id_Hilo* 24, 20, 33, 34, 25, 26, 19 etc.. tuvieron acceso al final.

Para la implementación de este problema, se simulo unicamente el acceso a la ***BASE DE DATOS***, esto pudo implementarse con archivos de texto, pero el objetivo es resolver el problema de la sincronización.

## 3. Sobre compilación

Para compilar el programa basta con hacer

**$ gcc Concurrente_PracticaR.c -fopenmp -o salida.out**

Sea $l$ el número de lectores en el programa y $e$ el número de escritores.

Para ejecutar el programa:

**$ ./salida.out** «**l**» «**e**»

## Referencias

[1] Andrew S. Tanenbaum, *Sistemas operativos modernos*, tercera edicion, Prentice Hall, Mexico, DF, 2009.