# Universidad Nacional Autónoma de México

# Facultad de ciencias


## Computación Concurrente 2024-1

**Profesora:**

Gilde Valeria Rodríguez Jiménez

**Ayudantes:**

Gibrán Aguilar Zuñiga

Luis Angel Leyva Castillo

Rogelio Alcantar Arenas

**Equipo:**

Concorriente

**Integrantes:**

DJLP

ADLG

# TEORÍA

1. Explica por qué tu solución si funciona, para esto puedes demostrar las propiedades de deadlock-free, libre de hambruna, lock-free, wait-free.

**Deadlock-free:** El código utiliza el algoritmo de Peterson para la cuestión del consumidor y los productores por lo que se reduce la posibilidad de un bloqueo de este tipo. Además de que no hay manera en que los hilos se bloqueen indefinidamente esperando unos a otros ya que contamos con el uso de lock() y unlock() en el método run.

**Libre de hambruna:** De nuevo cómo utilizamos peterson aseguramos que los productores y consumidor no esperen indefinidamente tanto para producir tortillas como para contarlas y añadirlas al arreglo del consumidor. Además la implementación garantiza que todos los productores tengan oportunidad de generar su tortilla sin quedar atrapados en un ciclo sin fin.

**Lock-free:** Como vimos la implementación de los locks con Peterson nos garantiza que siempre hay un progreso controlado gracias a que este candado administra la entrada de dos hilos a la SC, de esta forma tendremos un progreso en toda la ejecución del código.

**Wait-free:** Dadas las explicaciones anteriores también aseguramos que siempre hay un progreso por el uso del algoritmo de Peterson y el uso de las adquisiciones del candado y la liberación de este.

2. ¿Qué modificación tendrías que hacer para que tuviéramos m consumidores?

La modificación sería que en vez de inicializar un hilo serían los m hilos consumidores junto con su join. También sería cuestión de tratar la SC de cada consumidor para que cada uno tenga sus propias tortillas, y como ya tenemos Peterson no habría que mover mucho en esa parte más que la cuestión mencionada.

3. Supón que posees más atributos, de tal manera que tienes un atributo que almacena los kg de masa disponibles, así también de otro que almacena la cantidad de masa requerida para hacer 1 tortilla, ¿Qué modificación tendrías que hacer para que esto funcionara?

Usando el mismo WFSnapshot seria agregar dichos valores en StampedSnap para los kg de masa disponibles y la masa para crear una tortilla. Ya de ahí se manejan con métodos en el main y run como análogo como con el uso del método update().

4. Para el problema de la tortillería, cómo podríamos hacer para tener una fila dinámica, es decir que las personas llegaran sin tener que agruparlas o en bloques estáticos. Explica tu solución.

La solución sería ir despachando a las personas conforme van llegando, pero para eso debería haber una fábrica de tortillas bastante eficiente ya que si se van generando las tortillas conforme llega gente se produciría una fila. La otra forma es simplemente agregar las marcas de tiempo a una cola y ya así se sabría que la primera marca de tiempo que sale es de la primera que llegó.

5. Da comentarios de lo que aprendiste en esta práctica.

Vimos la implementación del WFSnapshot junto con sus Stamps, su uso no muy bien porque en general nos dieron muchos problemas para agregar, mostrar y utilizar información de las variables. Por ello simplemente simulamos las problemáticas en arreglos normales o manejables ya que si tratamos de usar los Snapshots pero nos dieron muchos errores de tipos.

Los proyectos mvn de esta práctica se encuentran en el zip “Concorriente.zip”.

Referencias


- [HS08] Maurice Herlihy and Nir Shavit. The Art of Multiprocessor Programming. Morgan Kauf-mann Publishers Inc., San Francisco, CA, USA, 2008.