Universidad Nacional Autónoma de México
Facultad de Ciencias

Computación Concurrente

Tarea 3

Bonilla Reyes Dafne - 319089660
García Ponce José Camilo - 319210536
Juárez Ubaldo Juan Aurelio - 421095568

Objetos concurrentes: Condiciones de progreso y de corrección

Ejercicio 1

Contesta de forma breve (máximo 3 líneas por pregunta) lo siguiente:

a) ¿Cuál es la diferencia entre una historia secuencial y una concurrente?

Las principales diferencias son las llamadas al mismo tiempo y el match de los métodos. En las ejecuciones secuenciales ni llamadas ni respuestas suceden al mismo tiempo, por lo que tenemos una respuesta inmediatamente después, y por el contrario, en las concurrentes no tenemos un mach inmediatamente después de llamar a un método.

b) Describe con tus propias palabras, ¿en qué consiste la linealizabilidad?

La linealizabilidad asegura que en un sistema concurrente las operaciones den la impresión de ocurrir instantáneamente, respetando el orden en que terminan en el tiempo real. Esto es, que una ejecución concurrente se comparta como una ejecución secuencial con respecto a un objeto y sus comportamientos de dichos objetos.

c) Describe con tus propias palabras, ¿cuál es la diferencia entre la consistencia secuencial y la linealizabilidad?

La principal diferencia es el orden. Por un lado, la consistencia pide un orden sobre el periodo de inactividad, es decir, un orden parcial, mientras que la linealizabilidad pide un orden sobre los métodos, es decir, un orden total.

Ejercicio 2

¿La siguiente propiedad es equivalente a decir que un objeto x es *wait-free*? Argumenta por qué.

Propiedad: Para toda historia infinita H de x, cada hilo que toma un número infinito de pasos en H completa un número infinito de llamadas a métodos.

Sí, es equivalente. Para ver porque, primero veamos la definición de *wait-free*:

"Un método es wait-free si garantiza que cada llamada termina su ejecución en un número finito de pasos."

Tomando esto en cuenta, veamos que son equivalentes:

- **Veamos la ida: $\Rightarrow$**

  Supongamos que un objeto x tiene un método y que es *wait-free* entonces para cualquier historia infinita, H si los hilos de la historia hacen llamadas al método y terminan su ejecución en un número finito de pasos, debido a que el método es *wait-free*. Por lo tanto, si se hacen infinitas llamadas al método, estas llamadas se terminarán en un número infinito de pasos, ya que son infinitas las llamadas.

- **Veamos el regreso: $\Leftarrow$**

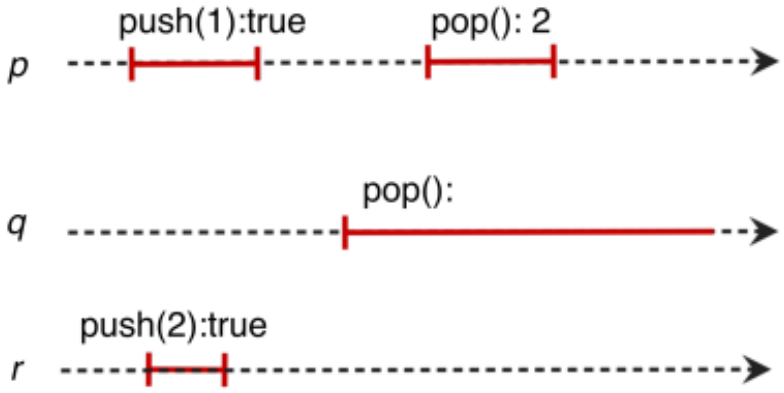
  Sea una historia infinita H donde los hilos de la historia hacen llamadas infinitas al método y de un objeto x y las llamadas se completan, entonces cada llamada al método y debe terminan en un número finito de pasos, ya que si alguna llamada no termina en un número finito de pasos tendremos que la llamada no puede terminar y, por lo tanto, si la llamada realizada por un hilo no termina significa que ese hilo solo completo una cantidad finita de llamadas (ya que no puede hacer más si no termino una llamada) por lo cual el método y del objeto x debe terminar cada llamada en una cantidad finita de pasos y concluimos que el método debe es *wait-free*.

$\therefore$ Si es equivalente.

Ejercicio 3

Considera la siguiente ejecución de una implementación de una pila para tres hilos *p, q, r* y sobre un solo objeto (de tipo pila).

La especificación secuencial de una pila es que el método *push(x)* añade a x al inicio de la pila y el método *pop()*:x elimina el primer elemento x al inicio de la pila (mantiene *LIFO*).



a) ¿Es linealizable con respecto a una pila? De ser así, incluye una linearización y especifica la *extensión* de *H* y *complete(H')*.

Sí, es linealizable, para esto tendremos las siguientes historias (diremos que el objeto pila se llama O):

- Tenemos que H es:

$$p\ O.push(1)$$

$$p\ O : true$$
$$p\ O : true$$
$$r\ O.push(2)$$
$$r\ O : true$$
$$p\ O.pop()$$
$$p\ O : 2$$
$$q\ O.pop()$$

- Después tenemos que la *extensión* de H o H' es:

$$p\ O.push(1)$$
$$p\ O : true$$
$$r\ O.push(2)$$
$$r\ O : true$$
$$p\ O.pop()$$
$$p\ O : 2$$
$$q\ O.pop()$$
$$q\ O : 1$$

Ahora, solo agregamos la respuesta $O : 1$.

Por otro lado, para *complete(H')* será la misma H', ya que no tiene llamadas incompletas (sin respuesta) y, por último, la linearización S será *complete(H')* (entonces también es *complete(H')*), así tenemos que sí es linealizable con respecto a una pila.

b) ¿Es secuencialmente consistente? Incluye una historia que lo justifique.

Sí, es secuencialmente consistente, esto debido a que como si es linealizable entonces es secuencialmente consistente, para esto usaremos la historia S del inciso a:

$$p\ O.push(1)$$
$$p\ O : true$$
$$r\ O.push(2)$$
$$r\ O : true$$
$$p\ O.pop()$$
$$p\ O : 2$$
$$q\ O.pop()$$
$$q\ O : 1$$

Notemos que el orden de las llamadas de métodos de cada hilo se respetan (tenemos que para el hilo p primero pasa él $push(1)$ y luego él $pop()$), entonces tenemos que sí es secuencialmente consistente.

c) ¿Es consistente en la inactividad? Incluye una historia que lo justifique.

Sí, es consistente en la inactividad, para esto usaremos la historia S del inciso a:

$$p\ O.push(1)$$

$$p\ O : true$$

$$r\ O.push(2)$$

$$r\ O : true$$

$$p\ O.pop()$$

$$p\ O : 2$$

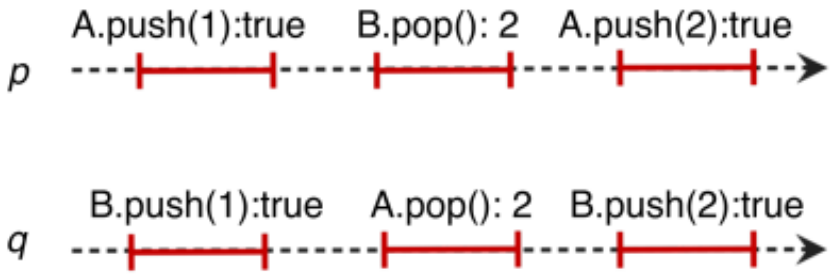
$$q\ O.pop()$$

$$q\ O : 1$$

La única sección de inactividad que tenemos es entre los métodos $push()$ y los métodos $pop()$, por lo tanto, la historia que usamos si sirve, ya que primero se hacen las llamadas y respuestas de los métodos $push()$ y luego los de $pop()$, respetando el periodo de inactividad.

Ejercicio 4

Considera la siguiente ejecución de una implementación de una pila para dos hilos *p, q* y sobre dos objetos (de tipo pila) *A, B*. (Considera la especificación secuencial del ejercicio 2).



a) Considera que *H* es la historia de la ejecución, ¿*H*|*A* y *H*|*B* son secuencialmente consistentes?

Primero definamos a $H|A$ y $H|B$:

- $H|A$:

```
p A.push(1)
p A : true
q A.pop()
q A : 2
p A.push(2)
p A : true
```

- $H|B$:

```
q B.push(1)
q B : true
p B.pop()
p B : 2
q B.push(2)
q B : true
```

Ahora, revisemos si $H|A$ es secuencialmente consistente. Veamos que para que sea secuencialmente consistente tenemos que hacer que se comporte como una pila, respetando las llamadas y respuestas de los hilos.

Entonces, tenemos esta historia:

```
p A.push(1)
p A : true
p A.push(2)
p A : true
q A.pop()
q A : 2
```

Notemos que el orden de las llamadas y las respuestas de p y q se mantuvieron, por lo que podemos decir que si es secuencialmente consistente.

De manera análoga, podemos hacer lo mismo para $H|B$, obteniendo así la siguiente historia:

```
q B.push(1)
q B : true
q B.push(2)
q B : true
p B.pop()
p B : 2
```

Por lo tanto, $H|B$ también es secuencialmente consistente.

b) Considera que H es la historia de la ejecución, ¿$H|A$ y $H|B$ son linealizables?

Primero, revisemos si $H|A$ es linealizable, para ello notemos que tenemos la siguiente historia:

```
p A.push(1)
p A : true
q A.pop()
q A : 2
p A.push(2)
p A : true
```

Notemos que para que la ejecución se comporte como una pila, necesitamos que `push(2)` vaya primero y luego `pop()`. Sin embargo, notemos que en la ejecución `pop()` va antes que `push(2)`, pero recordemos que no podemos reordenarlos porque se rompería el orden parcial. Por lo tanto, podemos decir que $H|A$ no es linealizable.

Ahora bien, notemos que esto mismo pasa también para $H|B$, ya que `pop()` va primero que `push(2)`:

```
q B.push(1)
q B : true
p B.pop()
p B : 2
q B.push(2)
q B : true
```

Por lo tanto, $H|B$ tampoco es linealizable.

c) ¿La historia *H* es secuencialmente consistente y linealizable?

La historia H no es secuencialmente consistente, ya que para que fuera secuencialmente necesitamos tener primero los `push(2)` antes que los `pop()` (para los dos objetos). Sin embargo, notemos que de ser así, tendríamos que romper el orden de los métodos de los hilos, porque tenemos `A.pop():2` y `B.push(2)` en el hilo q, de manera similar que en B. De hecho, notemos que en el hilo `p` tenemos `B.pop()` y `B.push(2)`, por lo que no podemos reordenar de tal manera que se respete como funciona una pila y el orden de ambos hilos.

Ahora bien, revisemos si la historia H es linealizable. Para ello, primero recordemos la siguiente propiedad:

"H es linealizable si y solo si para cada objeto X, H|X es linealizable".

- (Clase del 4/9/24: definición de composicional).

Notemos que en el inciso **b)** vimos qué $H|A$ y $H|B$ no son linealizables. Por lo tanto, la historia H no es linealizable.

Ejercicio 5

Considera la siguiente clase *Visibility*:

```
class Visibility {
  Shared Variables:

1   int x = 0;
2   volatile Boolean flag = false;
3   void writerThread() {
4     try {
5       Thread.sleep(1000);
6       x = 1;
7       flag = true;
8     } catch (InterruptedException e) {}
9   }
10  void readerThread() {
11    while(!flag) {}
12    int y = 100 / x;
13  }
14 }
```

a) ¿Es posible que el método *readerThread()* divida entre cero? Argumenta porqué.

En este caso, no es posible que readerThread() divida entre cero. Ya que aunque la variable x no es volatile, java actualiza las variables antes de volatile para todos los hilos cuando la variable volatile flag se actualiza, por lo que cuando readerThread() acceda a x, el valor ya estara actualizado.

b) Si ambas variables son *volatile*, ¿es posible que el método *readerThread()* divida entre cero? Argumenta por qué.
En este caso, no es posible que el readerThread() divida entre cero, ya que ambas variables son volatile, por lo que los cambios en ellas son visibles inmediatamente para el hilo lector, lo que asegura que cuando lea la variable x, esta sera igual a 1.

c) Si ninguna de las dos variables es *volatile*, ¿es posible que el método *readerThread()* divida entre cero? Argumenta por qué.
Este caso es similar al primero, ya que al no tener variables volatile, ya que no podemos asegurar que cuando los valores de x o $flag$ se actualicen, estos sean visibles entre hilos, por lo que es posbile que el hilo lector lea el valor antiguo de x y divida entre 0

Referencias

[1] Ayudantías, clases de laboratorio y teoría de la semana del 9 a 13 de septiembre del 2024.