Sistemas Distribuidos y Verificación
Computación Concurrente
Tarea 10

Prof: Sergio Rajsbaum rajsbaum@im.unam.mx
Ayudantes: David Méndez MedezD.lopez@ciencias.unam.mx
Juan Onofre barttcarl@gmail.com

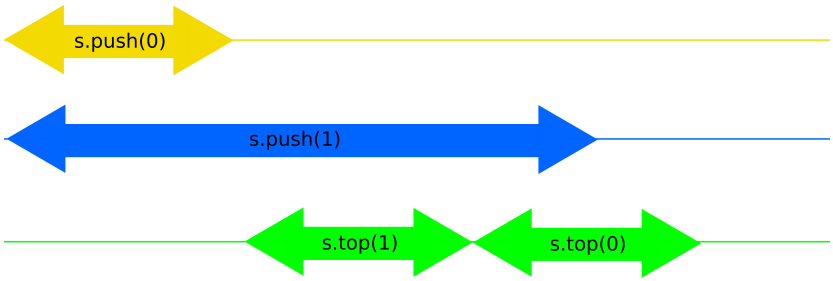
Entrega: 20 Mayo 2014

Argumenta en detalle tus respuestas. Ejercicios sin demostrar no cuentan.
Es necesario demostrar la correctez y complejidad de tus algoritmos.
No se aceptan tareas después de la fecha límite

1. Hacer un resumen de a lo más una cuartilla de *Don't Settle for Eventual Consistency*

2. En la figura se muestra una historia para tres procesos, cada línea corresponde a un proceso diferente. Los tres procesos trabajan sobre una pila s. Este objeto tiene dos métodos:

   - $s.top(i)$ que indica que el elemento i está en la casilla superior de la pila, pero no elimina al elemento i.
   - $s.push(i)$ que pone el elemento i en la casilla superior de la pila s.



   a) Indica si la historia es linearizable

   b) Indica si la historia es secuencialmente consistente.

3. Considera el siguiente código.
   La clase `AtomicInteger` es un contenedor para un valor entero. Esta clase contiene el método `boolean CompareAndSet(int expect, int update)` que compara el valor actual del objeto con `expect`. Si los valores son iguales, entonces atómicamente se reemplaza el valor del objeto con el valor `update` y se devuelve `true`. Si los valores no son iguales, el objeto no cambia y se regresa `false`. La clase también contiene el método `int get()` que regresa el valor actual del objeto.
   El código muestra la implementación de una cola FIFO. La cola guarda a los elementos en el arreglo `items`, el cual supondremos que tiene capacidad infinita; también contiene dos campos de la clase

`AtomicInteger`. `head` que es el índice de la casilla del siguiente elemento a ser eliminado y `tail` que es el índice de la casilla en la que se guardará el siguiente elemento.

```
class IQueue<T> f
      AtomicInteger head = new AtomicInteger(0);
      AtomicInteger tail = new AtomicInteger(0);
      T[] items = (T[]) new Object[Integer.MAX VALUE];
      public void enq(T x){
            int slot ;
            do{
                  slot = tail.get () ;
            }while (! tail.compareAndSet(slot, slot + 1));
            items[slot] = x;
      }

      public T deq() throws EmptyException{
            T value;
            int slot ;
            do{
                  slot = head.get() ;
                  value = items[slot];
                  if (value == null){
                        throw new EmptyException();
                  }
            }while(!head.compareAndSet(slot, slot + 1));
            return value;
      }
}
```

Da un ejemplo que muestre que esta implementación no es linearizable.

4. En clase se vio como construir registros a partir de otros. Describir detalladamente el algoritmo que construye registros *Multi-Writer Atomic* a partir de los registros *Multi-Reader Atomic*. Demostrar de manera intuitiva que el algoritmo es correcto.