Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente

Verão de 2021/2022, Segunda Série de Exercícios

Resolva os seguintes exercícios e apresente os testes com os quais validou a correção da implementação de cada exercício. A entrega deve ser feita através da criação da tag **`0.2.0`** no repositório individual de cada aluno.

1. Implemente a seguinte classe *thread-safe*, sem recurso à utilização de *locks*.

```
class CounterModulo(moduloValue: Int) {
   val value: Int
   fun increment(): Int
   fun decrement(): Int
}
```

   Este classe implementa um contador, cujo valor pode estar entre 0 e **`moduloValue`** (exclusive). O método **`increment`** incrementa o valor do contador e caso este seja **`moduloValue-1`**, então o resultado do incremento deve ser **`0`**. O método **`decrement`** decrementa o valor do contador e caso este seja **`0`**, então o resultado do decremento deve ser **`moduloValue-1`**. Ambos os métodos retornam o resultado da operação.

2. Considere a classe **`UnsafeMessageBox`**, cuja implementação se apresenta a seguir:

```
class UnsafeMessageBox<M> {
   private class Holder<M>(val msg: M, initialLives: Int) {
       var lives: Int = initialLives
   }
   private var holder: Holder<M>? = null
   fun publish(msg: M, lives: Int) {
       holder = Holder(msg, lives)
   }
   fun tryConsume(): M? =
       if (holder != null && holder.lives > 0) {
           holder.lives -= 1
           holder.msg
       } else {
           null
       }
}
```

   Esta implementação reflete a semântica de um sincronizador *message box* contendo no máximo uma mensagem que pode ser consumida múltiplas vezes, até ao máximo de **`lvs`**. Contudo esta classe não é *thread-safe*. Implemente, sem utilizar *locks*, uma versão *thread-safe* deste sincronizador.

3. Realize uma implementação da interface **`Lazy`**, apresentada em seguida, sem recurso a *locks* e usando espera activa.

```
class Lazy<T>(initializer: ()->T) {
   val value: T
       get() = TODO()
}
```

   O valor de **`value`** é calculado pela primeira *thread* que ler esta propriedade, através da execução da função definida por **`initializer`**.

4. Realize na linguagem Kotlin uma implementação de uma fila de mensagens, sem recurso a *locks*, usando o algoritmo de Michael-Scott.

   Data limite de entrega: 14 de maio de 2022

   ISEL, 26 de abril de 2022