Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente

Inverno de 2021/2022, Segunda Série de Exercícios

Resolva os seguintes exercícios e apresente os programas de teste 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. Considere o sincronizador **`Exchanger<V>`** presente na biblioteca de classes da plataforma Java.
   a. Realize uma implementação deste sincronizador usando monitores.
   b. Realize uma implementação deste sincronizador sem recurso ao uso de *locks* e usando espera activa. Implemente apenas a variante do método **`exchange`** que não recebe *timeouts*. Em adição, este método não precisa de estar sensível a interrupções da *thread*. Use **`Thread.yield`** na espera activa.
   c. Adicione o suporte para cancelamento da espera por interrupção à resolução da alínea anterior.

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

```
public class UnsafeMessageBox<M> where M : class {
  private class MsgHolder {
    internal readonly M msg;
    internal int lives;
  }
  private MsgHolder msgHolder = null;
  public void Publish(M m, int lvs) {msgHolder = new MsgHolder { msg = m, lives = lvs };}
  public M TryConsume() {
    if (msgHolder != null && msgHolder.lives > 0) {
      msgHolder.lives -= 1;
      return msgHolder.msg;
    }
    return 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 em *Java* ou em C#, sem utilizar *locks*, uma versão *thread-safe* deste sincronizador.

3. Implemente o sincronizador *message queue*, para suportar a comunicação entre *threads* produtoras e consumidoras através de mensagens do tipo genérico **`E`**. A entrega de mensagens deve usar o critério FIFO (*first in first out*): dadas duas mensagens colocadas na fila, a primeira a ser entregue a um consumidor deve ser a primeira que foi colocada na fila. Contudo esse critério FIFO não tem de ser garantido nas *threads* consumidoras: um pedido de remoção pode ser satisfeito antes de outro pedido de remoção realizado previamente. A interface pública deste sincronizador, em *Java*, é a seguinte:

```
public class MessageQueue<E> {
  public void enqueue(E message);
  public Optional<E> dequeue(long timeout) throws InterruptedException;
}
```

O método **enqueue** não é bloqueante, retornando imediatamente após entregar a mensagem na fila. Optimize este sincronizador, usando as técnicas *non-blocking* apresentadas nas aulas teóricas. Note que o método **dequeue** continua a ser potencialmente bloqueante. A resolução deste exercício deve incluir uma implementação de uma fila *lock-free* usando o algoritmo de Michael & Scott.

Data limite de entrega: 4 de dezembro de 2021

ISEL, 15 de novembro de 2021