Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Programação Concorrente
Teste Global da Época de Recurso, Inverno de 2019/2020

---

1. [2] Considere a definição em *Java* da classe **`UnsafeCreateOnceSpinLock`,** como uma tentativa para implementar o sincronizador *create-once lock*, que suporta toda a sincronição necessária em cenários de partilha entre *threads* de objectos *singleton* com inicialização *lazy*.

```
class UnsafeCreateOnceSpinLock {
  private long state = -1L;   // uncreated

  // called in order to create the shared object
  public boolean tryCreate() {
    while (state != 0L) {
      if (state == -1L) {
        state = Thread.currentThread().getId();
        return true;  // the current thread must create the shared object
      }
      Thread.yield();
    }
    // the shared object has already been created
    return false;
  }

  // called after a successful creation of the shared object
  public void onCreationSucceeded() {
    if (state != Thread.currentThread().getId()) throw new IllegalStateException();
    state = 0L;
  }

  // called after a failed creation of the shared object
  public void onCreationFailed() {
    if (state != Thread.currentThread().getId()) throw new IllegalStateException();
    state = -1L;
   }
}
```

   Esta classe ilustra um algoritmo que implementa o sincronizador *create-once lock*, contudo não é *thread safe*. Sem usar *locks*, implemente, em C# ou *Java*, uma versão *thread safe* desta classe (**`CreateOnceSpinLock`**).

2. [5] Implemente em *Java* ou C#, com base nos monitores implícitos ou explícitos, o sincronizador *n-ary exchanger*. A interface pública deste sincronizador, na linguagem *Java*, é a seguinte:

```
public class NaryExchanger<T> {
  public NaryExchanger(int size);
  public List<T> exchange(T elem, int timeout) throws InterruptedException;
}
```

   Este sincronizador, que é uma generalização do sincronizador *exchanger*, suporta a troca de informação entre grupos de **`size`** *threads*, para **`size`** maior ou igual a 2. As *threads* que utilizam este sincronizador manifestam a sua disponibilidade para iniciar uma troca invocando o método **`exchange`**, especificando o objecto que pretendem partilhar com o grupo (**`elem`**) e o tempo limite de espera (**`timeout`**). Um grupo é formado quando **`size`** threads tiverem chamado o método **`exchange`**.
   O método **`exchange`** termina: (a) devolvendo uma lista com todos os objectos partilhados pelas **`size`** *threads* do grupo à qual a *thread* invocante pertence; (b) devolvendo **`null`** se for excedido o tempo de espera especificado, ou; (c) lançando **`InterruptedException`** quando a espera da *thread* for interrompida. A lista retornada pelo método **`exchange`** inclui o próprio elemento passado como parâmetro. Se a *thread* A recebeu uma lista incluindo o objecto partilhado pela *thread* B, então a *thread* B recebeu obrigatoriamente uma lista que inclui o objecto partilhado pela *thread* A. Um objecto só pode ser partilhado entre um grupo de *threads*.

3. [5] Implemente em *Java* ou C#, com base nos monitores implícitos ou explícitos, o sincronizador *synchronous semaphore*, em que a operação de entrega de autorizações aguarda a aquisição das mesmas. A interface pública deste sincronizador, na linguagem *Java*, é a seguinte:

```
public class SynchronousSemaphore {
  public boolean release(int releases, int timeout) throws InterruptedException;
  public boolean acquire(long timeout) throws InterruptedException;
}
```

O método **release** entrega **releases** autorizações ao semáforo, retornando: (a) o valor **true**, quando todas as autorizações entregues tiverem sido adquiridas; (b) o valor **false**, se for excedido o tempo de espera especificado; (c) lançando **InterruptedException** quando a espera da *thread* for Interrompida. Se o método retornar **false** ou lançar uma excepção, então nenhuma das autorizações poderá ser adquirida. O método **acquire** adquire uma autorização do semáforo, retornando: (a) o valor **true** quando tiver sido adquirida a autorização; (b) o valor **false**, se for excedido o tempo de espera especificado; (c) lançando **InterruptedException** quando a espera da *thread* for Interrompida. A entrega e consumo das autorizações deve seguir a política FIFO (*first in first out*).

4. [6] Considere o seguinte método:

```
public R Compute(T[] elems, R initial)  {
  R acc = initial;
  foreach (var elem in elems)
    acc = C(B(A(elem)), acc);
  return acc;
}
```

Os métodos **A** e **B** são funções sem efeitos colaterais e passíveis de múltiplas execuções em paralelo. Ambos os métodos recebem **T** e retornam **T**. O método **C** realiza uma operação não associativa.

a) Realize uma versão assíncrona do método **Compute** seguindo o padrão TAP (*Task-based Asynchronous Pattern*) usando os métodos assíncronos do C# e/ou a funcionalidades disponíveis na TPL. Assuma que tem disponível as versões TAP dos métodos **A**, **B**, e **C**. Tire partido do paralelismo potencial existente.

b) Realize a variante da versão assíncrona do método implementado no exercício da alínea anterior, onde o número de operações **A** pendentes (i.e., iniciadas e não concluídas) está limitado a 10. Assuma que tem disponível uma implementação do semáforo com interface assíncrona realizado nas aulas.

5. [2] No método **M**, apresentado em seguida, as invocações a **A** podem decorrer em paralelo, o que seria vantajoso já que é nessa operação que se concentra a maior componente de processamento. O método **B** implementa uma operação, comutativa e associativa, sendo **default(T)** o respectivo elemento neutro. Use a *Task Parallel Library*, para implementar o método **ParallelM**, de forma a tirar partido de todos os *cores* de processamento disponíveis.

```
public T[] M<T>(T[] items) {
 var res = new T[items.Length + 1]; var acc = default(T);
 for (var i = 0; i < items.Length; i++) {
   res[i] = A(items[i]);
   acc = B(acc, res[i]);
 }
 res[items.Length] = acc;
 return res;
}
```

Duração: 2 horas e 30 minutos
ISEL, 4 de fevereiro de 2020