Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente

Teste Global de 2ª Época, Inverno de 2014/2015

---

1. [3] Considere a classe `UnsafeSpinSemaphore`, cuja implementação em C# é apresentada a seguir:

```
public class UnsafeSpinSemaphore {
  private readonly int maximum;
  private int count;

  UnsafeSpinSemaphore(int i, int m) {
    if (i < 0 || m <= 0)
      throw new ArgumentException("i/m");
    count = i; maximum = m;
  }

  public void Acquire {
    SpinWait sw = new SpinWait();
    while (count == 0)
      sw.SpinOnce();
    count -= 1;
  }

  public Release(int rs) {
    if (rs < 0 || rs + count > maximum)
      throw new ArgumentException("rs")
    count += r;
  }
}
```

   Como o próprio nome inidca esta classe não é *thread-safe*. Sem recorrer à utilização de *locks*, implemente a classe `SafeSpinSemaphore` com a mesma semântica de sincronização, mas *thread-safe*.

2. [4] Implemente em Java ou C#, com base nos monitores implícitos ou explícitos, o sincronizador *data distributor* para suportar a comunicação entre *threads* produtoras e consumidoras em cenários onde as *threads* produtoras entregam vários elementos e as consumidoras recolhem até *n* elementos, com *n* definido na construção.

```
public class DataDistributor<D> {
   public DataDistributor(int n);
   public void Put(List<D> data); // adiciona elementos (sem limite)
   public List<D> Take();         // obtém entre 1 e n elementos
}
```

   O método `Put` entrega um conjunto de itens de dados ao sincronizador, sem limite de número. O método `Take` recolhe entre 1 e *n* itens de dados, conforme o máximo que estiver disponível, bloqueando a *thread* invocante enquanto não existir pelo menos um item disponível. As chamadas ao método `Take` devem ser servidas com disciplina LIFO (*last-in-first-out*) com os itens passados em `Put` a serem entregues por ordem FIFO (*first-in-first-out*). O sincronizador deve suportar a interrupção das *threads* bloqueadas no método `Take`.

3. [4] Implemente em *Java* ou C#, com base nos respectivos monitores implícitos ou explícitos, o sincronizador *bulletin board*, cuja interface pública se apresenta a seguir:

```
public class BulletinBoard<W> where W : class {
   public void Post(W warning, uint validity);
   public W Receive();
   public void Clear();
}
```

   Este sincronizador suporta a comunicaçao entre *threads* através de mensagens de aviso, do tipo de dados genérico `W`. As mensagens de aviso são afixadas no boletim invocando o método `Post`; a cada aviso é associada uma validade temporal; se o valor do argumento `validity` for zero, o aviso apenas será transmitido às correntemente bloqueadas pelo método `Receive`; caso `validity` tenha um valor superior a zero, este especifica o tempo de validade do aviso, em milissegundos. Se for afixado um aviso quando ainda estiver outro válido, o novo aviso substitui o anterior. As *threads* que pretendam ser notificadas da ocorrência de avisos invocam o método `Receive`. Este método retorna de imediato se existir um aviso válido ou bloqueia a *thread* invocante até que um seja afixado ou que ocorra cancelamento por interrupção. O método `Clear` limpa a mensagem de aviso afixada no boletim, se existir. Note que as *threads* bloqueadas em `Receive` têm de receber o aviso veiculado por `Post`, independentemente de qualquer chamada posterior a `Clear`.

4. [6] A interface `Services` representa serviços síncronos de um sistema de gestão de equipamentos. O método síncrono `CheckDeviceVersion` usa operações de `Services` para verificar se a versão de um equipamento coincide com a registada anteriormente. Todas as operações de `Services` envolvem comunicações por redes de dados com os equipamentos ou com servidores de bases de dados, sendo vantajosa a disponibilização de variantes assíncronas dessas operações, bem como de `CheckDeviceVersion`. Para além disso, em `CheckDeviceVersion`, a operação `GetStoredVersion` poderia decorrer em paralelo com a sequência `GetDeviceAddress` → `GetVersionFromDevice`.

```
public class SyncOps {
   public interface Services {
      String GetDeviceAddress(int devId);
      int GetVersionFromDevice(String addr);
      int GetStoredVersion(int devId);
   }
   public bool CheckDeviceVersion(Services svc, int devId) {
      String addr = svc.GetDeviceAddress(devId);
      int devVer = svc.GetVersionFromDevice(addr);
      int stoVer = svc.GetStoredVersion(devId);
      return devVer == stoVer;
   }
}
```

   a. [4] A classe `APMOps` será a variante assíncrona de `SyncOps`, seguindo o estilo *Asynchronous Programming Model* (APM). Apresente os métodos `BeginCheckDeviceVersion` e `EndCheckDeviceVersion`, que usam a interface `APMServices` (variante APM de `Services` que não tem de apresentar).
   NOTA: não pode usar a TPL e só se admitem esperas de controlo dentro das operações `End`, estritamente onde o APM o exige.

   b. [2] A classe `TAPOps` será a variante assíncrona de `SyncOps`, seguindo o estilo *Task-based Asynchronous Pattern* (TAP). Tirando partido da *Task Parallel Library* (TPL) ou dos métodos `async` do C#, implemente o método `CheckDeviceVersionAsync`, que usa a interface `TAPServices` (variante TAP de `Services` que não tem de apresentar).
   NOTA: na implementação não se admite a utilização de operações com bloqueios de controlo.

5. [3] Considere o método `ProcessData` apresentado a seguir. As invocações a `ProcessItem` podem decorrer em paralelo, o que se considera vantajoso já que é nessa operação que se concentra a maior parte do tempo total de execução. O método `CombineResults` implementa uma operação comutativa e associativa e `new Result()` produz o seu elemento neutro. A função `ProcessItem` só realiza operações de leitura sobre a instância de `Item` que recebe como argumento. Tirando partido da *Task Parallel Library*, apresente uma versão de `ProcessData` que use invocações paralelas a `ProcessItem` para tirar partido de todos os *cores* de processamento disponíveis.

```
static Result ProcessData(IEnumerable<Data> items, Info info) {
   Result res, total = new Result();
   foreach (Data item in items) {
      res   = ProcessItem   (item, info);
      total = CombineResults(total, res);
   }
   return total;
}
```

Duração: 2 horas e 30 minutos
ISEL, 20 de Fevereiro de 2015