LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO
MESTRADO INTEGRADO EM ENGENHARIA FÍSICA

Programação Concorrente

Teste de Avaliação[1]

22 de Maio de 2019 Duração: 2h00m

I

1 Explique o conceito de *starvation*, descreva um cenário que seja propício à ocorrência deste fenómeno, e que solução adoptaria nesse cenário para o evitar.

2 Explique a utilidade de poder ter mais do que uma variável de condição num monitor (e.g., na biblioteca de concorrência em Java) e porque tal não é equivalente a usar vários monitores, cada um com uma variável de condição (e.g., os monitores nativos de Java).

3 Compare as abordagens usadas para construir uma abstracção para uso concorrente por threads clientes em Java e processos clientes em Erlang.

II

Pretende-se que escreva em Java, fazendo uso de primitivas baseadas em monitores, código que permita jogadores participarem num jogo de adivinha. Cada partida envolve 4 jogadores (cada um representado por uma thread), que competem para ver quem adivinha primeiro um número gerado aleatoriamente entre 1 e 100. Cada partida é limitada a um minuto e a 100 tentativas de resposta (total para todos os jogadores). Devem ser suportadas várias partidas a decorrer em simultâneo. As interfaces a implementar são:

```
interface Jogo {
    Partida participa();
}

interface Partida {
    String adivinha(int n);
}
```

A operação `participa()` deverá bloquear até poder começar uma partida (4 jogadores a quererem participar), devolvendo o objecto que representa a partida. Sobre este objecto, a operação `adivinha(n)`, usada para jogar, devolve um de: `GANHOU` se esta tentativa foi a primeira a acertar (dentro dos limites de tempo e tentativas de resposta); `PERDEU` se algum jogador já ganhou; `TEMPO` se esgotou o limite de tempo da partida (um minuto); `TENTATIVAS` se foi excedido o limite de tentativas; `MAIOR` / `MENOR` se o número escondido está acima/abaixo de `n`.

III

Apresente o código Erlang para implementar o mesmo sistema descrito no grupo II, através de um ou mais processos. Supondo que os jogadores são processos Erlang que comunicam pelo mecanismo nativo de mensagens, implemente também as funções de interface apropriadas para serem usadas por estes para interagirem com o(s) processos relevante(s).

---

[1] Cotação — 6+8+6