Monitores nativos em Java

Grupo de Sistemas Distribuídos
Universidade do Minho

Objectivos

Uso de monitores nativos de Java em problemas de ordem de execução.

Mecanismos

Keyword `synchronized`. Métodos `wait`, `notify` e `notifyAll` de `Object`.

Exercícios propostos

1. Implemente um *bounded buffer* para uso concorrente, como uma classe que disponibilize os métodos `put` e `get`, que bloqueiam quando o buffer está cheio/vazio respectivamente. O buffer deverá usar um array de tamanho fixo N, passado no construtor.

```
class BoundedBuffer<T> {
  BoundedBuffer(int N) { ... }
  T get() throws InterruptedException { ... }
  void put(T x) throws InterruptedException { ... }
}
```

2. Considere um cenário produtor/consumidor sobre o BoundedBuffer do exercício anterior, com P produtores e C consumidores, com um número total de threads $C + P = N$ e tempos de produção e consumo T_p e T_c. Obtenha experimentalmente o número óptimo de threads de cada tipo a utilizar para maximizar o débito.

3. Escreva uma abstracção para permitir que N threads se sincronizem:

```
class Barreira {
  Barreira (int N) { ... }
  void await() throws InterruptedException { ... }
}
```

   A operação `await` deverá bloquear até que as N threads o tenham feito; nesse momento o método deverá retornar em cada thread. a) Suponha que cada thread apenas vai invocar `await` uma vez sobre o objecto. b) Modifique a implementação para permitir que a operação possa ser usada várias vezes por cada thread (barreira reutilizável), de modo a suportar a sincronização no fim de cada uma de várias fases de computação.