LICENCIATURA EM ENGENHARIA INFORMÁTICA
UNIVERSIDADE DO MINHO

Sistemas Distribuídos

Teste[1]

9 de dezembro de 2024 Duração: 2h00m

Responda a cada grupo numa folha separada, entregando obrigatoriamente três folhas.

I

Responda diretamente a cada pergunta, omitindo considerações genéricas sobre cada um dos assuntos:

1 Caracterize, comparativamente, as arquiteturas de sistemas distribuídos estudadas.

2 Indique os cuidados a ter para minimizar o impacto da utilização de locks no desempenho dos programas.

3 Explique a importância da serialização e reestruturação de dados para a implementação de sistemas distribuídos e indique obstáculos à sua realização.

4 Indique que relação entre eventos, locais e remotos, captura um relógio lógico e em que medida é ou não suficiente para a implementação de um mecanismo de exclusão mútua distribuído.

II

Considere um serviço de gestão do desembarque de navios porta-contentores. O comandante de um navio primeiro pede permissão para atracar numa doca. Depois de lhe ser indicado o número da doca, dirige-se a esta. Espera então até ser informado que o desembarque está completo, para depois sair da doca. Considere que existem 7 classes de tamanho de porta-contentores (de 0 a 6), e 28 docas no porto (de 0 a 27), sendo que um navio de classe C só pode atracar em docas com número $D \geq 4 * C$.

Apresente duas classes Java (para serem usadas no servidor), usando primitivas baseadas em monitores, que implementem as interfaces abaixo, tendo em conta que os seus métodos serão invocados num ambiente *multi-threaded*.

```
interface Manager {
  Trip permission(int size) throws InterruptedException;
}
interface Trip {
  int dockId();
  void waitDisembark() throws InterruptedException;
  void finishedDisembark();
  void depart();
}
```

O método `permission` é usado para pedir permissão para atracar, devendo bloquear até ser atribuída uma doca que esteja livre, apropriada para a classe de tamanho do navio. Devolve um objeto `Trip` para ser usado durante a estadia no porto. O método `dockId` devolve o número da doca que foi atribuída; `waitDisembark` deve bloquear até o desembarque estar terminado; `finishedDisembark` serve para informar que o desembarque terminou; `depart` serve para informar que o navio saiu da doca. Quando há vários navios à espera, dê prioridade ao uso de docas por navios maiores, que só possam ser servidos por estas, em detrimento de navios menores. Minimize as *threads* que são acordadas.
Simplificação (70% da cotação): assuma que não há classes de tamanho e todas as docas servem para todos os navios.

III

Considere o sistema descrito acima, ao qual clientes, que representam comandantes de navios, se ligam por TCP. Implemente só o programa servidor usando *threads*, *sockets* TCP e as interfaces apresentadas no pergunta anterior. Assuma que no servidor existe uma *thread* que irá invocar o método `finishedDisembark` quando apropriado. Descreva o protocolo usado, que deve ser o mais simples possível, por exemplo, baseado em linhas de texto.

---

[1] Cotação — 10+7+3