ENGENHARIA INFORMÁTICA
UNIVERSIDADE DO MINHO

Sistemas Distribuídos

Exame[1]

24 de janeiro de 2024 Duração: 2h00m

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

I

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

1 "*Não posso utilizar o algoritmo de Peterson num programa com muitos threads porque vai ficar bloqueado.*" Discuta esta afirmação.

2 Justifique a migração de código do servidor para o cliente em sistemas distribuídos.

3 Considere o algoritmo distribuído de exclusão mútua baseado em relógios lógicos (Ricart-Agrawala). Justifique, neste contexto, uma vantagem e uma desvantagem da utilização de relógios lógicos.

4 Um sistema de gestão de *cloud computing* como o Kubernetes tem uma arquitetura cliente-servidor e coloca máquinas virtuais (pedidos) a máquinas físicas (reursos) dependendo dos disponibilidades e necessidades (memória, CPUs, discos, etc). Qual lhe parece ser o problema de sistemas distribuídos principal para garantir que este serviço está sempre disponível, mesmo quando as máquinas físicas podem falhar? Proponha uma solução apoiada nos algoritmos estudados, justificando.

II

Considere um serviço de transferência em que o servidor não armazena os ficheiros na sua totalidade, mas serve apenas de intermediário entre emissor e recetor ligados em simultâneo. Um cliente que pretende enviar um ficheiro liga-se ao servidor e informa-o da sua intenção, obtendo um identificador da transferência, para partilhar com o recetor. De seguida, começa a enviar os bytes do ficheiro. O cliente que deseja receber o ficheiro liga-se ao servidor, enviando o identificador partilhado e fica à espera de receber o conteúdo. Quando o emissor termina, fecha a conexão. O uso de um identificador de transferência por um terceiro cliente deve ser sinalizado como erro.

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

```
interface Manager {
  String newTransfer();
  Transfer getTransfer(String identifier);
}
interface Transfer {
  void enqueue(byte[] b) throws InterruptedException;
  byte[] dequeue() throws InterruptedException;
}
```

O método `newTransfer` é usado para preparar uma nova transferência, devolvendo o identificador; O método `getTransfer` devolve o objeto que gere essa transferência, se ainda puder ser usado, servindo de *buffer*. O método `enqueue` serve para adicionar novos bytes ao *buffer*, e bloqueia até tal ser possível sem exceder a capacidade de memória. Existem dois limites de capacidade: para cada transferência só podem estar guardados em cada momento 16k bytes; em todo o servidor só podem estar guardados 16M bytes. O fim de transferência é assinalado passando `null` como argumento. O método `dequeue` serve para retirar bytes ao *buffer*, devendo bloquear até existir conteúdo, ou até ao fim de transferência, caso em que devolve `null`. Quando é o limite global de memória que está em causa, tente dar às transferências a oportunidade de progredirem de uma forma justa.

Simplificação: (max. 60% da cotação) Ignore o limite de capacidade global.

III

Considere o sistema descrito no grupo II, ao qual clientes se ligam por TCP. Implemente só o programa servidor usando *threads*, *sockets* TCP e as interfaces apresentadas na pergunta anterior. Descreva o protocolo usado, que deve ser o mais simples possível.

---

[1] Cotação — (2+2+2+4)+(7+3)