LICENCIATURA EM ENGENHARIA INFORMÁTICA
UNIVERSIDADE DO MINHO

Sistemas Distribuídos

Exame[1]

27 de janeiro de 2025 Duração: 2h00m

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

I

Responda diretamente a cada pergunta e com grafia bem legível. Omita preâmbulos e considerações genéricas sobre cada um dos assuntos.

1 Nas três seguintes arquiteturas de sistemas distribuídos que estudámos, **Layered**, **Event-Based**, e **Peer-to-Peer**, onde encontramos exemplos da arquitetura **Cliente-Servidor** (Client-Server)?

2 A utilização de **váriaveis de condição** exige que sejam manipuladas em **estreita coordenação com um *lock***. Explique porquê e como. Auxilie-se de um exemplo conciso em pseudo-código que suporte a sua resposta.

3 Considere a arquitetura genérica Cliente-Servidor. Explique porque razão devemos evitar que, no servidor, uma mesma *thread* seja responsável por escrever para múltiplos *sockets*, associados a diferentes clientes.

4 Considere um **algoritmo distribuído** para garantia de **exclusão mútua** no acesso a recursos partilhados **baseado nos relógios de tempo real** (*hardware clocks*) dos nós. O que é usual assumirmos sobre o tempo de comunicação entre dois nós. Porquê?

II

Considere um serviço de controlo de concorrência de uma base de dados distribuída. Antes de executar uma transação, um cliente pede primeiro permissão ao serviço, enviando o conjunto de chaves que irá usar. O serviço deixará a transação prosseguir apenas se não existir nenhuma outra em conflito a executar. Caso contrário, terá que esperar. Para a **mesma** chave, só pode existir no máximo uma transação a executar em simultâneo. Chaves diferentes não originam conflitos. Uma transação que não consiga prosseguir **não deverá bloquear** outras transações que consigam.

Apresente uma classe Java (para ser usada no servidor), usando primitivas baseadas em monitores, que implementa a interface abaixo, tendo em conta que os seus métodos serão invocados num ambiente *multi-threaded*.

```
interface Manager {
    String begin(Set<String> keySet) throws InterruptedException;
    void commit(String id);
    char getTransactionStatus(String id);
}
```

Funcionalidade básica (até 80%): O método `begin` é usado para pedir permissão para executar uma transação, recebendo o conjunto de chaves usadas e retornando um identificador único, devendo bloquear até a permissão estar concedida. O método `commit` é usado para indicar o fim da execução de uma transação, recebendo o respetivo identificador. O método `getTransactionStatus` retorna o estado atual de uma transação ou `''` para um identificador de transação inválido. Existem três estados distintos: `'b'` (a iniciar), `'e'` (a executar), e `'c'` (terminada). Procure maximizar o paralelismo permitido e minimizar o número de *threads* que são acordadas.

Funcionalidade avançada (para 100%): Considere a existência de uma transação especial de manutenção, iniciada através de `begin(null)` e terminada com `commit(null)`. Assume-se que esta transação conflita com todas as outras. Para além disso, é pretendido que execute o mais cedo possível, sendo que tem prioridade sobre todas as outras transações em espera. Considera-se que existe no máximo uma instância desta transação a decorrer em simultâneo.

III

Considere o sistema descrito acima, ao qual clientes se ligam por TCP. Implemente só o programa servidor usando *threads*, *sockets* TCP, e a interface apresentada na pergunta anterior. Considere que cada cliente pode enviar múltiplos pedidos antes de receber resposta. Descreva o protocolo usado, que deve ser o mais simples possível, por exemplo, baseado em linhas de texto.

---

[1] Cotação — 10+7+3