+1/1/60+

Nome:

. . . . . .

Número:

☐0	☐0	☐0	☐0	☐0
☐1	☐1	☐1	☐1	☐1
☐2	☐2	☐2	☐2	☐2
☐3	☐3	☐3	☐3	☐3
☐4	☐4	☐4	☐4	☐4
☐5	☐5	☐5	☐5	☐5
☐6	☐6	☐6	☐6	☐6
☐7	☐7	☐7	☐7	☐7
☐8	☐8	☐8	☐8	☐8
☐9	☐9	☐9	☐9	☐9

ENGENHARIA INFORMATICA – UNIVERSIDADE DO MINHO

Teste de Sistemas Distribuídos

19 de janeiro de 2022 – Duração: 2h00

Instruções: *Preencha o nome e o número de aluno nesta folha pintando complemente as caixas correspondentes a cada algarismo; em cada pergunta de escolha múltipla há sempre uma ou mais respostas certas; para as assinalar pinte completamente as caixas correspondentes; não use as áreas sombreadas; preencha também o nome e número em cada folha de exame adicional.*

Grupo I

Responda a este grupo no próprio enunciado.

1. A implementação de exclusão mútua entre *threads* com o algoritmo *bakery* de Lamport não é normalmente usada na prática pois

- ☐ é incompatível com mais do que dois *threads* concorrentes
- ☐ não funciona corretamente com os processadores atuais
- ☐ leva a uma espera ativa que é ineficiente
- ☐ cada instância do *lock* ocuparia demasiado espaço em memória

2. Considere o problema de sincronização de relógios físicos em servidores dispersos na Internet e a sua resolução com o algoritmo *Network Time Protocol* (NTP). Este algoritmo:

- ☐ necessita de uma rede de difusão para funcionar
- ☐ torna dispensável a existência de uma referência de tempo fiável
- ☐ funciona melhor quando o atraso total na rede é em média mais pequeno
- ☐ funciona melhor quando os atrasos na rede entre os servidores envolvidos são simétricos (i.e., iguais nos dois sentidos)

3. Considere um modelo de concorrência *1-thread-por-ligação* na implementação do servidor num sistema cliente/servidor usando *sockets* TCP/IP. É verdade que este modelo:

- ☐ dispensa a utilização de primitivas de exclusão mútua na lógica da aplicação no cliente
- ☐ é compatível com a clientes multi-thread
- ☐ dispensa a utilização de primitivas de exclusão mútua na lógica da aplicação com estado partilhado do servidor
- ☐ dispensa a utilização de primitivas de exclusão mútua na manipulação do estado de sessão no servidor

4. Uma DHT como o Chord utilizada para a resolução de nomes para endereços num sistema distribuído:

- ☐ é adequada a um sistema administrativamente descentralizado
- ☐ perde dados quando um nó falha ou se desliga subitamente
- ☐ permite obter uma resposta consultando menos nós que um sistema hierárquico de dimensão semelhante
- ☐ obriga a que cada nó mantenha ligações com metade dos outros nós

5. Considere um sistema de *criptomoeda* na Internet como o Bitcoin em que cada participante guarda uma cópia do registo de todas as transações efetuadas e do conteúdo das carteiras e se quer garantir que uma mesma quantia não é gasta mais do que uma vez. Ignore os aspetos de segurança, admitindo que todos os participantes são honestos. Identifique aquele que lhe parece ser o problema de sistemas distribuídos mais importante e identifique uma solução típica. Discuta a adequação dessa solução ao ambiente em causa.

☐0 ☐.1 ☐.2 ☐.3 ☐.4 ☐.5 ☐.6 ☐.7 ☐.8 ☐.9 ☐1 *cotação*

Grupo II *Responda a cada pergunta deste grupo numa folha de exame separada.*

Considere um sistema cliente/servidor de apoio a mesas de voto em que um cliente pode verificar a sua identidade, garantindo que ainda não votou; esperar pela reserva de uma cabine de voto (número entre 1 e C); e votar (número entre 1 e E).

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

```
interface Votacao {
  boolean verifica(int identidade);
  int esperaPorCabine();
  void vota(int escolha);
  void desocupaCabine(int i);

  int vencedor(); // apenas para a alinea de Valorização
}
```

O método `verifica` testa se um dado eleitor pode votar, devolvendo `true` apenas uma vez para cada identidade; o método `esperaPorCabine` deve bloquear até uma cabine estar disponível, devolvendo o seu número; `vota` regista um dado voto dentro das E escolhas possíveis; `desocupaCabine` serve para assinalar que a cabine i ficou disponível.

Valorização: Implemente um método adicional `vencedor` que deve encerrar a eleição, impedindo novos votantes mas permitindo que todos os que já foram verificados ainda consigam votar e devolver o número do vencedor (assuma que não há empates).

7. Considere um serviço ao qual clientes se ligam por TCP, para votarem numas eleições. Ao chegar, um cliente envia o seu número de identificação, devendo o servidor verificar se o cliente pode votar e responder com `INVALIDO`, se já votou, ou `VOTE NA CABINE` i, logo que uma cabine esteja disponível. O cliente envia então o seu voto (apenas um número entre 1 e E), recebendo uma confirmação e desligando. O ato de votar liberta a cabine para poder ser usada por outros votantes.

Implemente só o programa servidor usando *threads*, *sockets* TCP, e a classe desenvolvida na pergunta anterior. Use um protocolo o mais simples possível, por exemplo, baseado em linhas de texto.