ENGENHARIA INFORMÁTICA
UNIVERSIDADE DO MINHO

Sistemas Distribuídos

Teste[1]

4 de janeiro de 2024 Duração: 2h00m

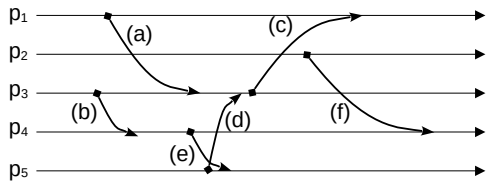
Responda a cada grupo num folha separada, entregando obrigatoriamente três folhas.

I

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

1 Como é que o algoritmo de exclusão mútua de Lamport (padaria ou *bakery*) garante um processo que pretende entrar na secção crítica não é ultrapassado mais do que uma vez por qualquer outro processo? Dê exemplos.

2 Considere o diagrama ao lado que representa os processos p_1 a p_5 que trocam mensagens enquanto usam relógios lógicos escalares com valores inicias de 5,1,1,2,2, respetivamente. Indique o valor do relógio para cada uma das mensagens *(a)*-*(f)*, justificando.



3 Qual é a estratégia partilhada pela disseminação epidémica e pelo Chord DHT para serem viáveis com um número muito grande de participantes? Justifique.

4 Considere uma agência de viagens em que o utilizador pode reservar transportes e estadias nos sistemas de diferentes transportadoras e hoteis alojados num mesmo centro de dados. Como podemos num sistema distribuído garantir que o utilizador reserva exatamente o programa completo? E se os sistemas em questão estiverem em centros de dados distantes?

II

Considere um sistema para organizar *raids* num jogo multi-jogador. Cada jogador indica qual o número mínimo de participantes no *raid*. Os participantes no próximo *raid* ficam definidos mal existem jogadores à espera em número suficiente para satisfazer os requisitos de todos eles (em termos de número de participantes). Os jogadores que apareçam mais tarde serão agrupados no *raid* seguinte. Logo que possível, o sistema 1) indica a cada jogador os nomes dos outros jogadores desse *raid*; 2) quando o *raid* pode começar pois o sistema só permite que estejam até R *raids* a decorrer em simultâneo.

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 {
  Raid join(String name, int minPlayers) throws InterruptedException;
}
interface Raid {
  List<String> players();
  void waitStart() throws InterruptedException;
  void leave();
}
```

A operação `join` deverá bloquear até estar formado o grupo de participantes no *raid*, devolvendo o objeto que o representa; tem como parâmetro o nome do jogador e o número mínimo de jogadores que o *raid* deve ter. A operação `players` devolve a lista de jogadores presentes no *raid*; `waitStart` deverá bloquear até o *raid* poder começar (só podem estar R a decorrer em simultâneo); `leave` é invocada quando um jogador abandona o *raid*, que termina quando todos os jogadores o tiverem feito.

Simplificação: (max. 60% da cotação) Apresente apenas uma classe Java com a interface `Manager` em que o método `join` devolve diretamente a lista de participantes (`List<String>` em vez de `Raid`) e ignore o limite R. Deve no entanto cumprir os requisitos em termos de número de participantes.

III

Considere o sistema de organização de *raids* descrito no grupo II, ao qual clientes, que representam jogadores, 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, por exemplo, baseado em linhas de texto.

---

[1] Cotação — (2+2+2+4)+(7+3)