

Sistemas Distribuídos - COS470 2021/1

Quinta Lista de Exercícios

Dica: Para ajudar no processo de aprendizado responda às perguntas integralmente, mostrando o desenvolvimento das respostas.

Questão 1: Por que um sistema transacional deve oferecer ACID? Explique também o significado de cada propriedade.

Questão 2: Considere um sistema bancário transacional e a seguinte implementação da função que transfere da conta $c1$ para a conta $c2$ o valor v .

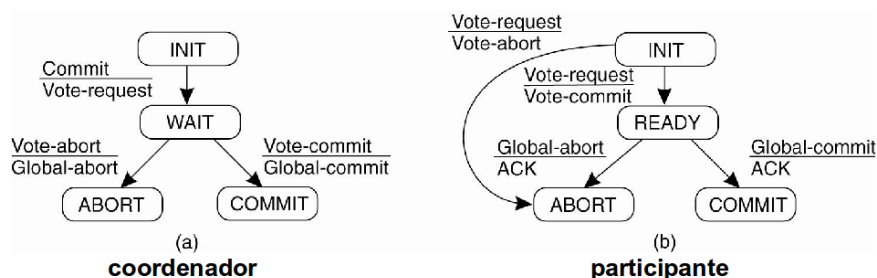
```
transferencia(c1, c2, v) {  
    acquire(c1)  
    se (retirada(c1,v) >= 0)  
        acquire(c2)  
        deposito(c2,v)  
        release(c1)  
        release(c2)  
        retorna 0  
    release(c1)  
    retorna -1  
}
```

Explique o que pode acontecer com esta implementação. Como você corrigiria a implementação?

Questão 3: Para que serve a técnica de *Two Phase Locking* (2PL)? Explique sucintamente como a mesma funciona.

Questão 4: Considere um sistema transacional distribuído e o uso do protocolo *Two Phase Commit* (2PC). Construa um exemplo onde temos uma condição de corrida que pode levar a um *deadlock*. Explique seu exemplo.

Questão 5: Considere o diagrama de transição de estados do protocolo *Two Phase Commit* (2PC):



1. Explique o que acontece quando um processo participante falha no estado **INIT**. Como o protocolo recupera desta falha?
2. Explique o que acontece quando um processo participante falha no estado **READY**. Como o protocolo recupera desta falha?

3. Explique o que acontece quando o coordenador falha no estado WAIT. Como o protocolo recupera desta falha?

Questão 6: Explique os conflitos *read-write* e *write-write* que surgem quando temos sistemas distribuídos com dados replicados.

Questão 7: Considere as seguintes execuções de instruções em diferentes processos, cada qual com sua memória local (assuma que inicialmente os valores das variáveis são zero). Determine quais casos (execuções) respeitam o modelo de consistência sequencial, indicando uma possível ordenação para as instruções.

- | | |
|--|---|
| 1. P1: W(x,1);
P2: R(x,0); R(x,1) | 5. P1: W(x,1);
P2: W(x,2);
P3: R(x,2); R(x,1);
P4: R(x,1); R(x,2); |
| 2. P1: W(x,1);
P2: R(x,1); R(x,0); | 6. P1: W(x,1); R(x,1); R(y,0);
P2: W(y,1); R(y,1); R(x,1);
P3: R(x,1); R(y,0);
P4: R(y,0); R(x,0); |
| 3. P1: W(x,1);
P2: W(x,2);
P3: R(x,1); R(x,2); | 7. P1: W(x,1); R(x,1); R(y,0);
P2: W(y,1); R(y,1); R(x,1);
P3: R(y,1); R(x,0); |
| 4. P1: W(x,1);
P2: W(x,2);
P3: R(x,2); R(x,1); | |

Questão 8: Considerando novamente as execuções de instruções acima em diferentes processos, determine quais casos (execuções) respeitam o modelo de consistência causal, indicando uma possível ordenação para as instruções.

Questão 9: Em se tratando de sistemas tolerante a falhas, qual é a diferença entre disponibilidade (*availability*) e confiabilidade (*reliability*)? Dê um exemplo que ilustre as diferenças.

Questão 10: Considere um componente com $MTTF = 2.5$ ano e $MTTR = 32$ horas. Considere o uso de componentes redundantes para projetar um sistema cujo componente tem disponibilidade de 99.99%. Assuma que falhas nestes componentes são independentes.

- Determine o número de componentes redundantes necessários.
- Determine a probabilidade do sistema com redundância ter ao menos um componente em falha.

Questão 11: Considere a organização de componentes redundantes TMR (*Triple Modular Redundancy*). Explique o que ocorre nos seguintes casos:

1. Exatamente um componente e um votador falha (*crash failure*) em cada linha.
2. Dois votadores falham (*crash failure*) na mesma coluna.
3. Dois votadores falham (*value failure*) em colunas diferentes mas com o mesmo valor de saída.

Questão 12: Explique por que falhas bizantinas são mais difíceis de lidar do que falhas que travam (*crash failures*).

Questão 13: Considere o protocolo de acordo bizantino com três participantes sendo um deles operando em falha bizantina. Mostre que o protocolo falha, ou seja, que os processos que não estão em falha não chegam a um consenso sobre o identificador do outro.