Cap 6 Exercicios

1) O tipo de aplicação que a sincronização de relógios físicos é indicada são em situações mais criticas em que necessitamos da sincronização do tempo para executa-los e marcando tempos bem próximos. As limitações são primeiro, um relógio não pode voltar atrás(se estiver marcando 300, o relógio não poderá ser corrigido para 299, tendo em vista que várias aplicações dependem desta semântica). Desta forma, se um computador necessita sincronizar−se com um outro computador cujo relógio está atrasado, ele terá que reduzir a freqüência de seu clock de modo que trabalhe um pouco mais lento e possa se igualar ao clock do computador com o qual está se sincronizando. Segundo, em sistemas distribuídos, não se pode determinar com precisão o delay de transmissão da rede. Por exemplo: Um cliente desejando sincronizar−se com um servidor envia−lhe uma requisição. O servidor responde devolvendo o tempo atual de seu clock. O problema é que quando o cliente recebe de volta a mensagem, o tempo retornado já está desatualizado, terceiro para algumas aplicações os relogios fisicos não são muito precisos

2) O tipo de aplicação que a sincronização de relógios lógicos é indicada quando não importa que todos os processos concordem a respeito da hora exata, mas que eles concordem sobre a ordem em que os eventos ocorreram. As limitações são não garante que as mensagens dos processos concorrentes sejam ordenadas, não permite decidir se dois eventos são concorrentes = relógios de Lamport não garantem que, se C(a) < C(b), a preceda causalmente b.

3) Sim, o Hybrid Time Clock(HTC) é um desses exemplos

4) A compensação é feita tendo um conhecimento preciso da posição relativa do transmissor ou receptor garantindo que a mensagem chega mais rapida ao receptor

5) A compensação do tempo de troca de mensagens é necessária na sincronização interna de um conjunto de relógios físicos porque é uma das dificuldades que os relógios físicos sofrem que é o delay da transmissão da mensagem de um processo pro outro, com esse relogio a gente sabe a hora exata da ocorrencia do evento mas não sabe a demora que foi chegar a mensagem de um lugar pro outro, por isso necessita dessa compensação

6) Precisão: O objetivo é manter os desvios entre dois relógios em quaisquer duas máquinas p e q dentro de um limite especificado, conhecido como a precisão π: $|Cp(t)−Cq(t)| <= \pi$ sendo Cp(t) o tempo de relógio computado da máquina p no tempo UTC t.

Acurácia: O objetivo é manter o desvio do relógio (em relação ao tempo UTC) limitado por um valor α: $|Cp(t)−t| <= \alpha$

Sincronização interna: manter os relógios precisos porque a gente está sincronizando os computadores de uma forma local, todos estão sendo sincronizados
Sincronização externa: manter os relógios com acurácia, isso porque a gente está tentando sincronizar um computador com uma hora externa que seria do UTC entao se a gente garante nesse computador p uma sincronização com o UTC, a gente consegue sincronizar através da precisão os outros computadores que se basearem nesse computador p

7) O relógio da segunda máquina:

- 990 vezes em um milissegundo
- 990.000 vezes em um segundo
- Em um segundo menos 10.000 ticks -> 10mseg

- Em um minuto 10mseg X 60 = 600mseg
O relógio da segunda máquina é 1% mais lento:
- 1% de 60seg = 0.01 x 60 = 0.6seg = 600mseg

8) GPS: Cada satélite tem até quatro relógios atômicos que são calibrados periodicamente, um satélite transmite sua posição em broadcast e anexa marcas de tempo a cada msg, informando a hora local, essa transmissão broadcast permite que todo receptor na Terra calcule com precisão sua própria posição (usando três satélites).

Exemplos de aplicações:
- Definição da rota entre usuários do aplicativo Uber Juntos: O App calcula a rota de a cordo com o GPS do cliente e do motorista, traçando a rota mais indicada para o ponto de encontro e o destino final.
- Atualização de um banco de dados compartilhado de aplicações financeiras
- Jogos multiníveis com múltiplos usuários

9) Esse resultado Vj[i] <= Vi[i] sempre vai ser assim porque a unica forma do valor i no processo i ser incrementado é se o processo i criar um evento, ou seja, todos os outros processos, a qual são j sempre recebera no máximo o valor Vi[i] como máximo, como indicado na regra 3. Então a gente nunca vai aumentar esse valor i no processo j, já que o único que pode fazer isso é o proprio processo i.

10) É como foi dito no exercicio passado, provamos que sempre Vj[i] <= Vi[i]. Então se um processo m2 chega em um processo primeiro que o processo m4 saindo de um mesmo processo, o valor de Vm2[i] < Vm4[i] porque criamos um evento pros dois e o último sempre tem um valor maior. E sempre que ele para em um outro um processo j ocorre a incrementação na posição j naquele vetor, ou seja, para que tenha essa causalidade precisamos que o valor de Vm2 < Vm4, indicando que Vm2 aconteceu bem antes dos acontecimentos de Vm4 tornando m2 precessor causalmente de m4

11) Sim, o multicast totalmente ordenado se faz necessário quando precisamos garantir a mesma sequência de eventos em todos os processos do sistema. Operação onde todas as mensagens são entregues na mesma ordem a cada receptor.

12) Em um algoritmo centralizado sempre existe um processo fixo que age como coordenador. A distribuição vem do fato de que outros processos rodam em máquinas separadas.

13) Caso tenha uma falha do coordenador uma forma de tratar de forma que isto não prejudique o funcionamento do sistema, é quando o próximo pedido tentar permissão ao coordenador e irá falhar, com isso, o processo requisitante pode inicial um novo processo de eleição de coordenador. Escolhendo um novo coordenador e fazendo o sistema continuar normalmente. A recuperação do estado do servidor(fila de pedidos para acesso ao recurso compartilhado) não é estritamente necessária porque suponha que o algoritmo seja tal que o coordenador sempre responda a um pedido, ou aceitando ou negando acessar um certo recurso. Se não existem processos acessando recursos, nem na fila, então a queda do coordenador não é fatal e precisamos so fazer uma eleição mas caso tenha uma quantidade grande de processos querendo acessar um recurso na fila, ai seria necessario

14) Se um processo falhou e não responde a uma requisição de um outro processo para acessar um recurso, a falta de resposta pode ser interpretada como uma recusa de permissão, ou seja, o processo não acessa o recurso já que ele acha que o processo X está vivo e está esperando uma permissão

futura do X que nunca virá. Para resolver essa situação o melhor seria permitir que o processo requisitante não fique bloqueado esperando uma resposta. O processo requisitante apenas dorme por um período fixo de tempo, e quando acordar, verifica se os processos dos quais ele espera uma resposta estão vivos.

15)

16) Sim porque o deadlock acontece quando dois processos(P1 e P2) possuem já um recurso P1 com A e P2 com B e querem usar um recurso que já sendo usado por outro, ou seja, P2 quer A e P1 quer B, com isso provoca o deadlock onde os dois ficam esperando os dois usarem o recurso pra depois usar

17) O algoritmo funcionará normalmente, não importando se foi dois ou mais processos que começou a eleição. Cada um dos processos com maiores números irá receber duas mensagens de ELEIÇÃO. A segunda mensagem provavelmente será ignorada e o processo de eleição continua

18) Pode haver duas mensagens de eleição circulando simultaneamente sim porque basicamente dois processos tentaram a comunicação com o coordenador e falharam, por isso, eles começaram uma nova eleição nos dois processos. Isso não é um problema, porque os dois garantiram sempre o mesmo resultado então não importa tanto.