Mestrado Integrado em Engenharia Informática – Algoritmos e Sistemas Distribuídos

Primeiro teste – 21 de outubro de 2013 – Duração: 1 hora e 30 minutos

Responda às questões no espaço reservado para o efeito após cada alínea, ou na última página caso necessite mais espaço. Indique claramente qualquer pressuposto que tenha de fazer para resolver a uma questão.

Número		Nome	

1. [9,5 valores] Considere a seguinte especificação para uma variante do problema da difusão. Como usualmente, um conjunto fixo de processos podem ter inputs sob a forma "send(m)" e outputs sob a forma "deliver(m)", e durante a execução todos os processos podem enviar e entregar um número arbitrário de mensagens. Um processo correcto é um processo que não sofre falhas por crash.

   [C1] Se um processo correcto enviar uma mensagem m, então todos os processos correctos fazem a certo ponto ("eventually") output (deliver) de m.
   [C2] Se um processo fizer output de uma mensagem m, então todos os processos correctos fazem a certo ponto ("eventually") output de m.
   [C3] Uma mensagem m não pode estar repetida em mais do que um output do mesmo processo, e se um processo fizer output de uma mensagem m, então m foi anteriormente enviada.
   [C4] Se, para um certo processo p, p.send(m) preceder p.send(m') num trace do sistema, então, para qualquer outro processo p', p'.deliver(m') não pode preceder p'.deliver(m) nesse trace do sistema.

   (a) Para cada condição da especificação, indique se esta é uma condição de safety ou de liveness. (Responda apenas safety ou liveness junto a cada condição.)

   C1: _____ C2:_____ C3:_____ C4:_____

   (b) Considere o seguinte trace de uma execução com dois processos p1,p2 que não falham. (A notação usada para descrever o trace indica, para cada elemento da sequência de inputs/outputs, o processo que faz o input ou o output, seguido de ":", seguido do input ou output correspondente.)

   p1:send(m1), p2:send(m2), p2:deliver(m1), p1:deliver(m3), p1:send(m3), p2:deliver(m3), p2:deliver(m2).

   Para cada condição da alínea anterior, indique se a condição é ou não verificada pelo trace. No caso das condições de liveness que não são verificadas, escreva também uma extensão do trace onde esta passe a ser verificada.

C1:
C2:
C3:
C4:

   (c) Este problema tem solução num sistema assíncrono com canais fiáveis onde os processos podem ter falhas por crash? Se sim, apresente o pseudo-código de um algoritmo que resolve o problema. Se não, prove que não existe solução para o problema.

2. [2,5 valores] Considere uma variante à especificação de linearizibilidade/atomicidade numa variável de leitura/escrita, que relaxa a propriedade da atomicidade de forma a que, no caso de haverem várias leituras e escritas concorrentes, as leituras possam também retornar qualquer dos valores a serem escritos concorrentemente com a leitura. Mais precisamente, dado um trace T de uma variável replicada:
    - Definem-se duas operações (A e B) como sequenciais se a resposta de uma das operações preceder a invocação da outra operação em T. Ou seja, resposta(B) < invocação(A) ou resposta(A) < invocação(B)
    - Definem-se duas operações (A e B) como concorrentes se não forem sequenciais, ou seja, resposta(B) > invocação(A) e resposta(A) > invocação(B)
    - Para cada operação OP existe um ponto de serialização, entre a invocação e a resposta, tal que se movermos a invocação e a resposta para esse ponto, o trace resultante T_{ser} obedece às seguinte condições:
        - No caso de existirem operações de escrita concorrentes a uma operação de leitura OP em T, esta leitura pode retornar em T quer o valor da última escrita que precede OP em T_{ser}, quer o valor de qualquer escrita concorrente com OP em T.
        - Nos restantes casos os traces são iguais tal como na atomicidade, ou seja, OP retorna o mesmo valor em T_{ser} e em T.

    a) Desenhe um diagrama temporal de invocações e respostas correspondente a um trace que obedeça a esta variante da atomicidade mas que não obedeça à atomicidade.

    b) Proponha uma alteração ao algoritmo ABD que o torne mais eficiente (em termos do número de fases de pelo menos uma das operações) e que obedeça à nova especificação

3. [5,5 valores] Consider a seguinte especificação do consenso com falhas Bizantinas, que difere da especificação original na primeira condição pelo facto que só incidir sobre as propostas iniciais dos processos correctos:

C1 [Validade] Se todos os processos correctos tiverem a proposta inicial vi = v, então v é o único valor do output permitido pelas réplicas correctas,

C2 [Acordo] Dois processos correctos não podem decidir valores diferentes,

C3 [Terminação] Todos os processos correctos têm de decidir a certo ponto (“eventually”).

(a) Qual destas condições é uma condição de liveness?

Número: Nome:

(b) Complete o seguinte trace, usando a notação da pergunta 1 b), com três processos p1, p2, p3, em que p1 sofre uma falha Bizantina numa execução no modelo de falhas Bizantino mas não falha numa execução no modelo de falhas por crash, de forma a este trace obedeça à especificação original do consenso para o modelo de crash mas não a esta especificação descrita acima.

p1: propose (1),_____

(c) Usando a mesma notação, dê um exemplo de um trace que não seja obedeça à condição de liveness, e depois um exemplo de uma extensão desse trace que já obedeça.

(d) Considere a seguinte solução para o consenso num sistema síncrono estudada na aula:

```
statesi:
   proposei  - input, proposta inicial
   decidei - output, decisão final
   valsi  - inicialmente {proposei}
   roundsi - inteiro que representa o número do round, inicialmente 0

msgsi:
   if (roundsi<=f)
      forall (j in V)
            sendi(j, valsi);    // send to all
      }


transi:
      roundsi=roundsi+1
      forall (j in V-{i})
            receive(S) from j
             valsi = valsi U S
      if (roundsi == f+1)
            decidei = min(valsi)
```

(e) A solução acima resolve o consenso num sistema síncrono com um máximo de f falhas Bizantinas e um total de n=3f+1 réplicas? Se sim dê um argumento de correcção. Se não mostre um contra-exemplo em que a especificação não é cumprida.

4. [2,5 valores] Complete as seguintes 4 linhas de código do lock server do yfs, correspondente ao handler da função release que liberta um lock.

```
enum lstate {
    FREE,
    LOCKED
};
struct lock {
    pthread_cond_t cond;
    lock_protocol::lockid_t lid;
    lstate state;
};
int lock_server::release(int clt, lock_protocol::lockid_t lid, int &r)
{

  _____________________________________________
  lock *l = locks[lid];
  r = 0;
  if (l==NULL) return lock_protocol::NOENT;

  _____________________________________________

  _____________________________________________

  _____________________________________________
  return lock_protocol::OK;
}
```

Espaço adicional para resolução do teste. Indique junto à pergunta caso tenha de usar este espaço.