TOLERÂNCIA A FALTAS DISTRIBUÍDA

2006-2007
10 de Janeiro de 2007

Nota prévia

- O exame tem duração de duas horas e trinta minutos.
- Não se esqueça de identificar *todas* as folhas com o seu nome e número.

Abstracções Básicas

Considere a seguinte abstracção, designada por "canais ponto-a-ponto perfeitos":

Events:

Request: $\langle$ *pp2pSend*, *dest*, m $\rangle$: Used to request the transmission of message m to process *dest*.

Indication:$\langle$ *pp2pDeliver*, *src*, m $\rangle$: Used to deliver message m sent by process *src*.

Properties:

PL1: *Reliable delivery:* Let p_i be any process that sends a message m to a process p_j. If neither p_i nor p_j crashes, then p_j eventually delivers m.

PL2: *No duplication:* No message is delivered by a process more than once.

PL3: *No creation:* If a message m is delivered by some process p_j, then m was previously sent to p_j by some process p_i.

Questão 1 *(*2 valores*) Assuma que, para concretizar esta abstracção possui canais FIFO ponto-a-ponto não fiáveis. Apresente um algoritmo para oferecer canais ponto-a-ponto perfeitos. Utilize pseudo-código para descrever o algoritmo. Não se preocupe com questões de eficiência e assuma que não existem limitações de memória.*

Questão 2 *(*1 valor*) Indique se o protocolo TCP (sobre IP) concretiza a abstracção descrita anteriormente. Justifique.*

Considere o seguinte algoritmo (incompleto) que pretende concretizar um detector de falhas alguma-vez (do Inglês, *eventually*) perfeito:

Implements:
EventuallyPerfectFailureDetector ($\Diamond\mathcal{P}$).

Uses:
PerfectPointToPointLinks (pp2p).

upon event $\langle$ *Init* $\rangle$ **do**
alive := Π;
suspected := $\emptyset$;
period := TimeDelay;
startTimer (period);

upon event $\langle$ *Timeout* $\rangle$ **do**
forall $p_i \in \Pi$ **do**
if ($p_i \notin$ alive) $\wedge$ ($p_i \notin$ suspected) **then**
suspected := suspected $\cup \{p_i\}$;
trigger $\langle$ *suspect*, p_i $\rangle$;
else if ($p_i \in$ alive) $\wedge$ ($p_i \in$ suspected) **then**
suspected := suspected $\setminus \{p_i\}$;
trigger $\langle$ *restore*, p_i $\rangle$;
trigger $\langle$ *pp2pSend*, p_i, [HEARTBEAT] $\rangle$;
alive := $\emptyset$;
startTimer (period);

upon event $\langle$ *pp2pDeliver*, src, [HEARTBEAT] $\rangle$ **do**
alive := alive $\cup$ {src};

Questão 3 *(*2 valores*) Complete o algoritmo, de forma a que este concretize um detector $\Diamond\mathcal{P}$.*

Difusão fiável

Questão 4 *(*2 valores*) Apresente um algoritmo que concretize difusão fiável uniforme usando canais FIFO ponto-a-ponto perfeitos assim como um detector de falhas perfeito (o algoritmo deve funcionar sem recorrer a maiorias).*

Registos

Considere as seguintes execuções de acesso a um registo:

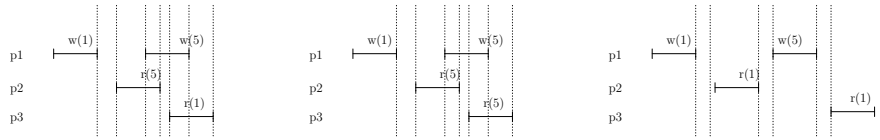

Figura 1: Execuções de registos

Questão 5 *(*2 valores*) Apenas uma das execuções é válida para registos atómicos, outra só válida para registos regulares (mas não atómicos) e outra não é válida nem em registos atómicos nem em registos regulares. Diga, justificando, a que categoria corresponde cada execução.*

Considere o seguinte algoritmo que concretiza um registo atómico $(1, N)$, ou seja, num sistema com 1 escritor e N leitores.

Implements:
$(1, N)$AtomicRegister (on-areg).

Uses:
BestEffortBroadcast (beb);
PerfectPointToPointLinks (pp2p).
PerfectFailureDetector ($\mathcal{P}$).

upon event $\langle$ *Init* $\rangle$ **do**
correct := Π;
sn := 0; v := 0;
acks := 0; reqid := 0;
readval := 0; reading := false;
readSet := $\emptyset$;

upon event $\langle$ *crash*, p_i $\rangle$ **do**
correct := correct $\setminus$ $\{p_i\}$;

upon event $\langle$ *on-aregWrite*, val $\rangle$ **do**
reqid := reqid+1; sn := sn + 1; v := val; acks := 1;
trigger $\langle$ *bebBroadcast*, [WRITE, reqid, sn, val] $\rangle$;

upon event $\langle$ *bebDeliver*, p_j, [WRITE, id, t, val] $\rangle$ **do**
if t > sn **then**
sn := t; v := val;
trigger $\langle$ *pp2pSend*, p_j, [ACK, id] $\rangle$;

(continua)

(continuacao)
upon event ⟨ *pp2pDeliver*, p_j, [ACK, id] ⟩ **do**
    if reqid = id **then**
        acks := acks + 1;

upon CONDICAO-1 **do**
    if reading = true **then**
        reading := false;
        trigger ⟨ *on-aregReadReturn*, readval ⟩;
    else
        trigger ⟨ *on-aregWriteReturn*, ⟩;

upon event ⟨ *on-aregRead*, ⟩ **do**
    reqid := reqid+1; readSet := ∅;
    trigger ⟨ *bebBroadcast*, [READ, reqid[r]] ⟩;

upon event ⟨ *bebDeliver*, p_j, [READ, r, id] ⟩ **do**
    trigger ⟨ *pp2pSend*, p_j, [READVALUE, id, sn[r], v[r]] ⟩;

upon event ⟨ *pp2pDeliver*, p_j, [READVALUE, id, ts, val] ⟩ **do**
    if reqid = id **then**
        readSet := readSet ∪ { (ts, val) };

upon CONDICAO-2 **do**
    (tstamp,readval) := *highest*(readSet);
    acks := 0; reading := true;
    trigger ⟨ *bebBroadcast*, [WRITE, reqid, tstamp, readval] ⟩;

Questão 6 *(*1 valor*) Seria possível definir a CONDICAO-2 da sequinte forma: "correct ⊆ readSet"? Em caso afirmativo, como deve ser expressa a CONDICAO-1?*

Questão 7 *(*1 valor*) Seria possível definir a CONDICAO-1 da sequinte forma: "acks = 1"? Em caso afirmativo, como deve ser expressa a CONDICAO-2?*

Questão 8 *(*1 valor*) Considere que não tinha acesso a um detector de faltas perfeito. Como seria necessário definir as CONDICAO-1 e CONDICAO-2?*

Acordo distribuído

Considere o seguinte algoritmo para concretizar o acordo uniforme.

Implements:
UniformConsensus (uc).

Uses:
ReliableBroadcast (rb);
BestEffortBroadcast (beb);
PerfectPointToPointLinks (pp2p);
PerfectFailureDetector ($\mathcal{P}$).

upon event $\langle$ *Init* $\rangle$ **do**
proposal := decided := $\perp$; round := 1;
detected := ack-set := $\emptyset$;
for i = 1 **to** N **do** proposed[i] := $\perp$;

upon event $\langle$ *crash*, p_i $\rangle$ **do**
detected := detected $\cup$ { *rank*(p_i) };

upon event $\langle$ *ucPropose*, v $\rangle$ $\wedge$ (proposal = $\perp$) **do**
proposal := v;

upon (round = *rank*(self)) $\wedge$ (proposal $\neq \perp$) $\wedge$ (decided = $\perp$) **do**
trigger $\langle$ *bebBroadcast*, [PROPOSE, round, proposal] $\rangle$;

upon event $\langle$ *bebDeliver*, p_i, [PROPOSE, r, v] $\rangle$ **do**
proposed[r] = v;
if r $\geq$ round **then**
trigger $\langle$ *pp2pSend*, p_i, [ACK, r] $\rangle$;

upon round $\in$ detected **do**
if proposed[round] $\neq \perp$ **then**
proposal := proposed[round];
round := round + 1;

upon event $\langle$ *pp2pDeliver*, p_i, [ACK, r] $\rangle$ **do**
ack-set := ack-set $\cup$ {*rank*(p_i)};

upon |ack-set $\cup$ detected| = N **do**
trigger $\langle$ *rbBroadcast*, [DECIDED, proposal] $\rangle$;

upon event $\langle$ *rbDeliver*, p_i, [DECIDED, v] $\rangle$ $\wedge$ (decided = $\perp$) **do**
decided := v;
trigger $\langle$ *ucDecide*, v $\rangle$;

Questão 9 *(*2 valores*) Explique de forma abreviada o funcionamento do algoritmo, indicando qual é o processo que tem a responsabilidade de fazer terminar o algoritmo.*

Variantes de consenso

Considere que pretendia resolver o seguinte problema. Possui um conjunto de processos. Cada processo possui um sensor de temperatura. Devido a imprecisões do sensor, existe um erro de leitura, pelo que cada processo pode ler um valor diferente, mesmo que o seu sensor esteja correcto. Para além disso, f sensores podem falhar, retornando valores inválidos ($f < N/2$). Os processos necessitam de realizar uma computação replicada determinista, pelo que devem acordar qual o valor da temperatura a usar (este valor deve ser correcto). Os processos só falham por paragem.

Questão 10 *(*3 valores*) Assuma que possui um detector de falhas perfeito, um serviço de consenso uniforme, e um serviço de difusão fiável. Usando estes componentes, invente um algoritmo para seleccionar qual o valor da temperatura a usar pelos processos na computação replicada.*

Replicação usando difusão em grupo fiável

Considere que possui à sua disposição um serviço de comunicação em grupo que oferece os serviços de filiação, sincronia na vista com entrega de mensagens regular ou uniforme e diferentes políticas de ordenação (sem ordem, causal e total).

Usando estes serviços pretende construir um serviço de reserva de computadores para depuração de programas. O serviço oferece as seguintes primitivas:

- "erro = Reservar (in string NomeDoPrograma, out string NomeDaMaquina);"
- "erro = Consultar (in string NomeDoPrograma, out string NomeDaMaquina);"
- "erro = Libertar (out string NomeDaMaquina);"

O serviço é responsável por escolher qual o computador onde deve ser executado o programa (retornando erro, caso não exista nenhum computador disponível).

Questão 11 *(*3 valores*) Apresente o pseudo-código para resolver este problema usando a comunicação em grupo para tornar o serviço tolerante a faltas. Use as primitivas de comunicação mais eficientes que satisfaçam a correcção do serviço.*