

Sistemi Distribuiti

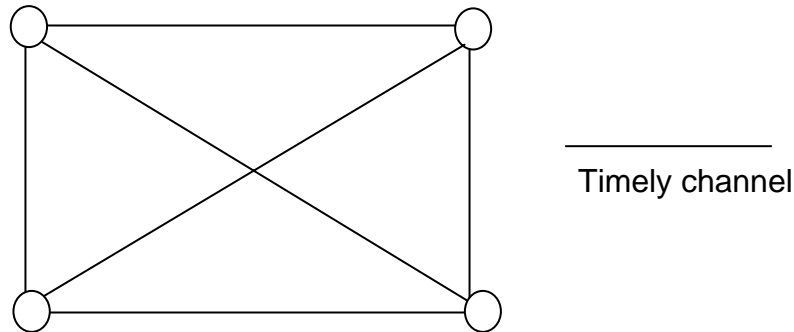
Esercizi d'Esame

Dott. Ing. Silvia Bonomi
bonomi@dis.uniroma1.it

Esame del 27/11/2006

Esercizio 2

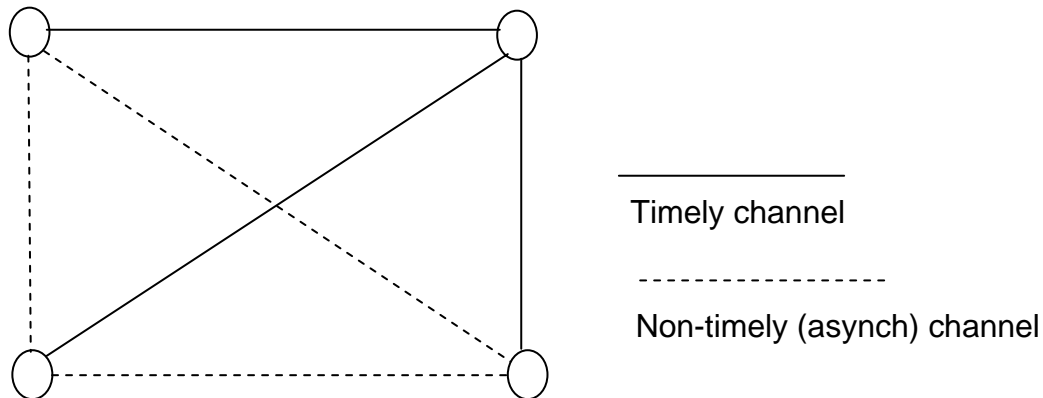
- Si consideri un sistema in cui ogni processo puo' comunicare con altri $n-1$ processi attraverso un canale di comunicazione (in figura e' rappresentato un sistema con 4 processi).
- Realizzare un algoritmo che implementi un failure detector perfetto P in ogni processo nel caso in cui il sistema è assunto sincrono:
 1. tempo di esecuzione di un passo elementare è trascurabile
 2. clock sono sincronizzati
 3. il tempo di trasferimento dei messaggi limitato ad un tempo max pari a D (canali timely).



Esame del 27/11/2006

Esercizio 3

- Con riferimento all'esercizio precedente, si può implementare un failure detector P in ogni processo in presenza di un sistema in cui un solo processo corretto che ha canali timely verso tutti gli altri?
- In caso affermativo proporre un'implementazione. In caso negativo discutere il perché'.

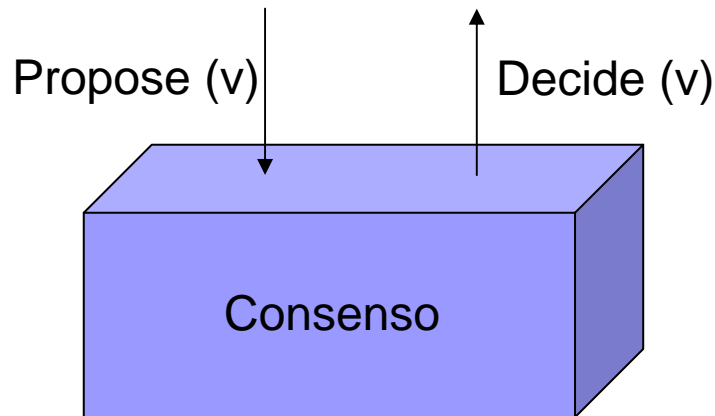


Esame del 20/07/2006

Esercizio 2

- Si consideri il seguente problema: ci sono n processi ed un numero di risorse $n-1$. Si ha a disposizione una primitiva di Consenso che espone l'operazione `propose` e ritorna con una `decide`. Scrivere un semplice algoritmo che soddisfi le seguenti specifiche:
 - Se f (con $f < n$) è il numero di processi guasti, allora **almeno** $n-1-f$ risorse saranno associate ad un processo.
 - Ogni risorsa è associata **al piu'** ad un processo.
- Quante istanze di consenso sono necessarie per soddisfare la specifica? Motivare la risposta.

Soluzione



- Si alloca una risorsa per volta
- ogni volta si toglie il processo dall'insieme di quelli che devono ancora avere la risorsa
- Assumiamo, per semplicità, che alla chiamata i -esima del consenso associamo la risorsa R_i

Init

```
Risorse = {R1, R2, ... Rn-1}  
Processi_senza_Risorsa =  $\Pi$   
trigger propose(<pi, Ri>)
```

Upon decide (<pi, Ri>)

```
Processi_senza_Risorsa = Processi_senza_Risorsa / {pi}  
Risorse = Risorse / {Ri}  
if (|Risorse|>0)  
    processo = selectRandom (Processi_senza_Risorsa)  
    risorsa = selectRandom (risorse)  
    trigger propose(<processo, risorsa>)  
endif
```

Esercizio

- Si consideri il seguente problema: ci sono n processi ed un numero di risorse $n-1$. Si ha a disposizione una primitiva di Consenso che espone l'operazione `propose` e ritorna con una `decide` ed un failure detector perfetto. Scrivere un semplice algoritmo che soddisfi le seguenti specifiche:
 - Se f (con $f < n$) è il numero di processi guasti, allora $n-1$ risorse saranno associate a processi corretti.
 - Ogni risorsa è associata **al piu'** ad un processo.

Esame del 20/04/2007

Esercizio 5

- Si consideri un sistema di N processi con identificativi totalmente ordinati $\{1,2,\dots,N\}$ e topologia ad anello con canali FIFO unidirezionali di tipo perfetto. Ogni processo puo' essere di colore verde (V) o rosso (R). I processi possono cambiare piu' volte colore durante la computazione. Ogni processo conosce gli identificativi degli altri processi. I processi non si guastano.

- Considerate ora la seguente specifica di Eventual Green Leader:
 - Se esiste un tempo t dopo il quale nessun processo cambia colore ed esiste almeno un processo verde, allora ogni processo elegge prima o poi come leader il processo verde che ha il minimo identificativo tra i processi verdi
 - (N.B. la variabile leader puo' essere settata piu' volte durante la computazione, puo' assumere temporaneamente valori diversi a processi diversi e puo' contenere l'identificativo di un processo rosso).

- Completare il codice seguente dando una spiegazione (obbligatoria) della sua correttezza nel soddisfare la specifica data.

```
when (init) do
    set mycolor                % il processo sceglie il suo colore tra R e V
    candidates:={1,...N}      % insieme dei processi candidati ad essere leader
    leader:=min(candidates)    % scelta del leader con identificativo minore
    fifopp2psend(mycolor, myid) to vicino sx

when (fifopp2pdeliver(color, id) from vicino dx) do
    if (color= /inserire qui)
        then
            /inserire qui
        else
            /inserire qui
    if (id <>myid)
        then /inserire qui

when (candidates changes) do
    /inserire qui

when (mycolor changes) do
    /inserire qui
```

```

when (init) do
    set mycolor                % il processo sceglie il suo colore tra R e V
    candidates:={1,...N}       % insieme dei processi candidati ad essere leader
    leader:=min(candidates)    % scelta del leader con identificativo minore
    fifopp2psend(mycolor, myid) to vicino sx

when (fifopp2pdeliver(color, id) from vicino dx) do
    if (color= G)
        then
            candidates = candidates  $\cup$  {id}
        else
            candidates = candidates / {id}
    if (id <>myid)
        then
            fifopp2psend(color, id) to vicino sx

when (candidates changes) do
    leader:=min(candidates)

when (mycolor changes) do
    fifopp2psend(mycolor, myid) to vicino sx

```

Esame del 28/3/2007

Esercizio 5

- Considerate un sistema di N processi P_1, P_2, \dots, P_N . Ogni processo P_i usa un oracolo che ritorna in output tramite l'istruzione `leader()` l'identificativo del processo che P_i considera leader corrente. L'oracolo soddisfa la specifica di Eventual Leader. Considerando che ogni processo P_i invia al più un messaggio m_i , scrivere un algoritmo che implementi la primitiva `eventualTOcast() / eventualTOdeliver()` secondo la seguente specifica:
- **Validity:** ogni messaggio m inviato in `eventualTOcast(m)` da un processo corretto è prima o poi consegnato tramite una `eventualTOdeliver(m)` da ogni processo corretto
- **Integrity:** se un messaggio m è consegnato tramite una `eventualTOdeliver(m)`, allora m è stato precedentemente inviato tramite una `eventualTOcast(m)`.

- **Finite Delivery Duplication:** se un messaggio m è inviato tramite una $\text{eventualTOcast}(m)$ allora può essere consegnato da uno stesso processo tramite una $\text{eventualTOdeliver}(m)$ un numero finito di volte.
- **Eventual Agreement:** per ogni coppia di processi corretti P_i, P_j , tali che:
 - P_i consegna due messaggi mh ed mk tali che $\text{eventualTOdeliver}(mh)$ precede $\text{eventualTOdeliver}(mk)$ e non esegue altre consegne di mh ed mk
 - P_j esegue $\text{eventualTOdeliver}(mk)$ e non esegue altre consegne di mk ,
- allora P_j non eseguirà mai, dopo la suddetta consegna $\text{eventualTOdeliver}(mk)$, una successiva consegna di mh $\text{eventualTOdeliver}(mh)$.
- Informalmente, la specifica prevede che tutti i processi corretti siano in grado di consegnare il set di messaggi inviati nello stesso ordine *prima o poi*. Prima di ciò possono consegnare i messaggi in ordine diverso (uno stesso messaggio può essere consegnato più volte).
- Nota: è possibile usare (se utile) primitive di broadcast ordinato di tipo FIFO/Causal, ma non una primitiva di TO broadcast.