Criteri di sincronizzazione

SCD

Anno accademico 2008/9
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@math.unipd.it

Criteri di valutazione – 1

- I costrutti di un linguaggio di programmazione vengono spesso valutati sotto 2 punti di vista
  - Potere espressivo (criterio oggettivo)
    - Misura la capacità del linguaggio di soddisfare direttamente i bisogni caratterizzanti del dominio applicativo
  - Usabilità (criterio soggettivo)
    - Misura il grado di interazione (**efficacia**) e di integrazione (**coerenza**) dei costrutti in esame, tra di loro e verso il resto del linguaggio
- Questo schema di valutazione può essere applicato anche ai costrutti di sincronizzazione
  - 1979, Toby Bloom, "Evaluating synchronisation mechanisms", Proc. 7th *ACM Symposium on Operating System Principles*, pp. 24-32

Criteri di valutazione – 2

- Problemi chiave nel progetto di modalità di sincronizzazione superiori a *exclusion synchronisation*
  1) Tipo di servizio richiesto
     - Canale tipato
  2) Ordine di arrivo delle richieste
     - Politiche di accodamento
  3) Stato del servente (conseguenza della storia d'uso)
     - Guardie [*avoidance synchronisation*]
  4) Priorità del cliente
  5) Parametri della richiesta

  Quale soluzione?

L'allocazione delle risorse – 1

- Problema assolutamente centrale a ogni forma di programmazione concorrente
  - Coinvolge tutte le dimensioni identificate da Toby Bloom
  - Particolarmente difficile da risolvere solo con guardie

Esempio: Si costruisca un controllore che debba allocare risorse a un gruppo di clienti, ove le risorse siano in numero finito e i clienti siano in competizione tra loro per averle. Ciascun cliente C_i può richiedere $N_i \geq 1$ risorse alla volta. Il protocollo di assegnazione sancisce che la richiesta, se accettata, debba essere soddisfatta integralmente, altrimenti sia tenuta in sospeso e il cliente bloccato sino a quando ne diventi possibile il soddisfacimento.

L'allocazione delle risorse – 2

- L'ampiezza della richiesta di un cliente appare come parametro del messaggio in ingresso sul canale tipato
- Per leggere il parametro (e determinare la soddisfacibilità della richiesta) occorre però prima accettare la sincronizzazione
  - Che fare se la richiesta non fosse soddisfacibile al momento?
  - Di sicuro non vogliamo ricorrere all'uso del "*busy wait*" !
  - Questione di usabilità prima che di potere espressivo

L'allocazione delle risorse – 3

- L'uso delle guardie di Dijkstra permette *avoidance synchronization*
  - Evitare sincronizzazione ove questa non sia ammissibile nello stato corrente della risorsa
- L'uso delle variabili di condizione del *monitor* di Hoare consente di ottenere sincronizzazione condizionale (*condition synchronization*)
  - Ciò che ci servirebbe in questo caso: prima valutare il parametro e poi, se necessario, imporre attesa
  - Ma il limite strutturale del *monitor* è proprio richiedere programmazione esplicita dell'attesa!

L'allocazione delle risorse – 4

- **Esistono almeno 2 estensioni del protocollo base di *avoidance synchronization***
  - **Consentire all'espressione di guardia di avere accesso ai parametri in ingresso della richiesta**
    - Linguaggio **SR** (*Synchronizing Resource*), http://www.cs.arizona.edu/sr/
  - **Consentire al processo servente di trasferire il cliente accettato ma non soddisfacibile su un'altra coda d'attesa**
    - Dove il cliente attenderà il verificarsi di condizioni più propizie
    - Consentendo così al canale di poter accogliere nuove richieste

L'allocazione delle risorse – 5

Forma base (1 risorsa per richiesta)

```
protected Controller is
 entry Allocate (R : out Resource);
 procedure Release (R : Resource);
private
 Free : Natural := Full_Capacity;
 …
end Controller;
protected body Controller is
 entry Allocate (R : out Resource)
  when Free > 0 is
 begin
  Free := Free – 1;
 …
 end Allocate;
 procedure Release (R : Resource) is
 begin
  Free := Free + 1;
 end Release;
end Controller;
```

La guardia usa un parametro in di ingresso

Soluzione SR per il nostro problema

```
type Request is range 1..Max_Requests;
protected Controller is
 entry Allocate
  (R      :     out Resource;
   Amount : in      Request);
 procedure Release
  (R      : Resource;
   Amount : Request);
private
 Free : Request := Request'Last; …
end Controller;
protected body Controller is
 entry Allocate
  (R      :     out Resource;
   Amount : in      Request)
  when Amount <= Free is
 begin
  Free := Free – Amount;
 end Allocate;
 procedure Release (…) is …
end Controller;
```


L'allocazione delle risorse – 6

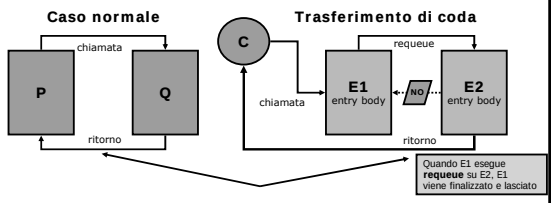
- **Il difetto principale dell'approccio SR è il suo eccessivo costo realizzativo**
  - **Legare direttamente la sincronizzazione (e dunque l'accodamento) al valore di un parametro della richiesta comporta che tutte le richieste accodate debbano essere rivalutate ogni volta che la condizione possa essere cambiata!**
- **L'alternativa è trasferire la richiesta attualmente non soddisfacibile a un'altra coda**
  - **Senza doverne rivalutare l'eventuale guardia**
  - **Con una singola operazione atomica (→ senza *race condition*)**

L'allocazione delle risorse – 7

- **Vediamo come il trasferimento di coda possa risolvere il nostro problema**
  - **Esercizio 10**
    Migliorare la soluzione proposta evitando di percorrere inutilmente la coda sul canale interno `Assign` (ossia quando le risorse disponibili non fossero ancora disponibili per soddisfare alcuna richiesta pendente)

Semantica del trasferimento di coda – 1

- **Il trasferimento di coda (`requeue`) non è una normale chiamata di procedura**



Semantica del trasferimento di coda – 2

- **Il trasferimento pone 2 problemi importanti**
  - **Verso quale coda di canale (`entry`)?**
    - La coda di destinazione (*target*) può essere una qualsiasi coda di canale, sia di processo che di risorsa protetta
      - Metodologicamente però, è preferibile restringerne l'ambito alla stessa entità di partenza
    - Il trasferimento verso coda di altra entità comporta rilascio dell'entità di partenza
      - Il che potrebbe non essere desiderabile
    - Il canale destinazione deve essere compatibile con quello di partenza
      - O nessun parametro o esattamente gli stessi della chiamata esterna
    - La direzione del trasferimento determina il destinatario finale del "lucchetto"
  - **Come trattare il *time-out* posto dal cliente sulla richiesta?**
    - Il trasferimento deve indicare esplicitamente se applicarlo al canale destinazione
    - Oppure considerarlo soddisfatto con l'attuale accettazione

Semantica del trasferimento di coda – 3

Che succede in questo caso?

Vi sono 2 possibilità:

1) La chiamata B.E1 non viene accolta entro il tempo T1 → chiamata annullata

2) La chiamata viene accolta in E1, ove esegue per un tempo T2, ma verrà annullata se E2 non venisse accolta entro T2+T1

```
-- A
select
 B.E1;
or
 delay T1;
end select;
```

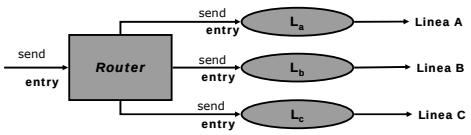
```
-- B
select
 accept E1 do
  … -- T2 seconds

  requeue E2 with abort;
 end E1;
or
 …
end select;
```

Questa clausola preserva l'eventuale *time-out* posto dal chiamante sulla coda di destinazione

Esercizio 11

Sull'utilità del trasferimento verso altra coda

Esercizio 11: Un *router* di rete può instradare pacchetti in ingresso verso 3 linee di comunicazione L_{a-c}, distinte ma funzionalmente equivalenti. La linea L_a rappresenta la scelta preferenziale, ma le altre linee (prima L_b e poi L_c) sono usate quando la precedente alternativa risulti sovraccarica. Si realizzi un programma concorrente che realizzi questa politica, considerando il diagramma qui riportato come possibile fonte di ispirazione.

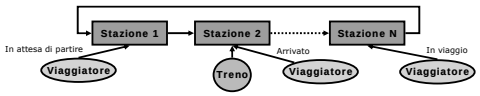


Un esempio complesso – 1

- **La possibilità di trasferimento tra code consente grande potere espressivo che aiuta a rappresentare situazioni complesse**
- **Vogliamo simulare il comportamento di un sistema di trasporto viaggiatori su linea circolare**
  - **(P.es. una linea metropolitana)**
  - **N stazioni su linea circolare → una risorsa protetta per stazione, presso la quale ciascun viaggiatore in partenza si blocca in attesa del treno**
  - **1 treno a capienza finita → entità attiva**
  - **K viaggiatori che si recano alla loro stazione di partenza avendo una stazione di arrivo → un'entità attiva per ogni viaggiatore**

Un esempio complesso – 2

- **Il trasferimento di coda è un modo pratico di simulare il trasporto dei viaggiatori**
  - **Quando il treno arriva in una stazione il programma trasferisce i viaggiatori in attesa in quella stazione (ma solo fino alla capienza massima del treno) nella coda della loro stazione di destinazione, dalla quale saranno rilasciati (= arrivati) quando il treno vi farà sosta**



Un esempio complesso – 3

- **Vediamo come programmare questa soluzione …**
- **Esercizio 12**
  Realizzare la stessa semantica della soluzione mostrata usando un altro linguaggio a piacere


PENINSULAR
RAIL ROAD LINE.
CHANGE OF HOURS.