Criteri avanzati di sincronizzazione

SCD

Anno accademico 2016/17
Sistemi Concorrenti e Distribuiti

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

Valutazione critica – 1

- Un linguaggio e i suoi costrutti possono essere valutati sotto due criteri generali
- Potere espressivo
  - La capacità del linguaggio di soddisfare i bisogni del dominio applicativo
- Usabilità
  - Il grado di interazione (misura efficacia) e di integrazione (misura di coerenza) dei costrutti in esame, tra di loro e verso il resto del linguaggio

Valutazione critica – 2

- È utile applicare questo schema di valutazione ai costrutti di sincronizzazione
  - 1979, Toby Bloom, "*Evaluating synchronisation mechanisms*", Proc. 7th ACM Symposium on Operating System Principles, pp. 24-32
- Bloom identifica 6 tipi di vincoli influenti sulla sincronizzazione
  - Per requisiti di potere espressivo superiori alla *exclusion synchronization* come la conosciamo finora

Condizioni di sincronizzazione – 1

- In funzione del tipo di richiesta
  - E.g., meglio letture che scritture
  - Espressa con canali tipati e guardie ☑
- In funzione del tempo della richiesta
  - Espressa con politiche di accodamento sulle code di guardia o di ordinamento (*scheduling*) dei clienti ☑
- In funzione dei parametri della richiesta ☒
  - E.g., algoritmo dell'ascensore

Condizioni di sincronizzazione – 2

- **In funzione dello stato di sincronizzazione della risorsa ☑**
  - **E.g., #utenti correnti per molteplicità >1 oppure # richieste in attesa (`'Count`)**
- **In funzione dello stato logico interno della risorsa ☑**
  - **E.g., *buffer is empty vs. buffer is full***
- **In funzione della storia d'esecuzione nella risorsa ☑**

L'allocazione delle risorse – 1

- **Problema ricorrente in ogni modello di programmazione concorrente**
  - **Coinvolge tutte le dimensioni identificate da 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

- **Nell'esempio, il volume di richiesta di un cliente appare come parametro in ingresso sul canale tipato**
- **Per leggere il parametro (e valutare se la richiesta è soddisfacibile) occorre però prima accettare la sincronizzazione**
  - **Che fare se la richiesta non fosse soddisfacibile al momento?**
  - **Di sicuro non vogliamo ricorrere all'uso di "*busy wait*"**
  - **Questione di usabilità prima che di potere espressivo**

L'allocazione delle risorse – 3

- **L'uso delle guardie abilita l'uso di *avoidance synchronization***
  - **Per evitare sincronizzazione ove questa non sia utile nello stato logico corrente della risorsa, prima di valutare la richiesta**
- **L'uso delle variabili di condizione del *monitor* di Hoare supporta attesa condizionale (`wait`, `signal`) all'interno della sincronizzazione**
  - **Sembrerebbe ciò che vogliamo: prima valutare il parametro e poi eventualmente imporre attesa**
  - **Ma il limite strutturale del *monitor* è proprio richiedere programmazione esplicita dell'attesa**

L'allocazione delle risorse – 4

- **Conviene estendere il protocollo base di *avoidance synchronization***
- **Ci sono almeno due soluzioni**
  - **Consentire all'espressione di guardia di accedere ai parametri in ingresso della richiesta**
    - Lo fa il linguaggio **SR** (*Synchronizing Resources*) http://www.cs.arizona.edu/sr/
  - **Consentire al processo servente di trasferire il cliente accettato ma non soddisfacibile, su 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;
```

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;
```

La guardia usa un parametro 'in' dell'*entry*

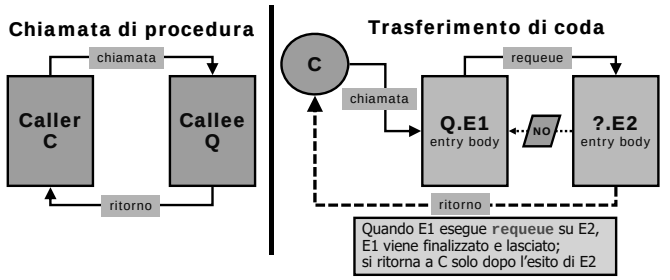
L'allocazione delle risorse – 6

- **Il fiasco dell'approccio SR è il suo eccessivo costo**
  - **Legando sincronizzazione e accodamento di chiamata al valore di parametri della richiesta rispetto allo stato della risorsa, tutte le richieste accodate devono essere rivalutate ogni volta che lo stato della risorsa cambi**
    - Una sorta di `notifyAll()` in peggio

- **Assai meglio trasferire ad altra coda la richiesta accettata ma non soddisfacibile**
  - **Senza dovere rivalutare l'eventuale guardia, la cui espressione potrà essere specializzata per la condizione di riaccodamento**
  - **Trasferendo la richiesta con singola operazione atomica**

Semantica del trasferimento di coda – 1

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



Semantica del trasferimento di coda – 2

- **Permettere il trasferimento di coda in modo programmatico pone due domande delicate**
  1. **Verso quali code di canale (`entry`) permetterlo**
  2. **Come trattare il *time-out* eventualmente posto dal cliente sulla sua invocazione iniziale**

Semantica del trasferimento di coda – 3

1. **Verso quale coda permettere trasferimento**
   - **Qualsiasi `entry` (sia di processo che di risorsa protetta)**
     - Ma per maggior coesione funzionale conviene restare nell'entità di partenza
   - **Il trasferimento verso coda di altra entità causa il rilascio dell'entità di partenza**
     - Necessari, ma potenzialmente indesirabile
   - **Il canale destinazione deve avere interfaccia compatibile con quello di partenza**
     - Esattamente la stessa della chiamata inziale oppure vuota
   - **La direzione del trasferimento (interna o esterna) determina il destinatario finale del "lucchetto" originale**

Semantica del trasferimento di coda – 4

2. **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

Cosa 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`**, dove 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 time units

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

La conseguenza è una distorsione temporale della tolleranza richiesta

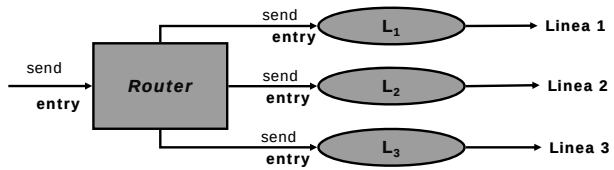
Questa clausola preserva l'eventuale *time-out* posto dal chiamante sulla coda di destinazione

Esempi d'uso – 1

- **Algoritmo di allocazione di risorse con trasferimento di coda**
- **Vedere soluzione base nell'esempio associato alla lezione di oggi**
- **Da migliorare evitando trasferimenti inutili**
  - **Come? Esercizio**

Esempi d'uso – 2

Un *router* di rete può instradare pacchetti in ingresso verso $N = 3$ linee di comunicazione $\mathbf{L}_{1\text{-}3}$, distinte ma funzionalmente equivalenti.
La linea $\mathbf{L}_1$ rappresenta la scelta preferenziale, ma le altre linee (prima $\mathbf{L}_2$ e poi $\mathbf{L}_3$) sono usate quando la precedente alternativa risulti sovraccarica.
Per realizzare questa soluzione il trasferimento di coda avverrebbe tra entità distinte (da Router verso L_i)



Esempi d'uso – 3

- **Il trasferimento tra code offre grande potere espressivo per rappresentare situazioni complesse**
- **Vogliamo simulare il comportamento di un sistema di trasporto viaggiatori su linea circolare**
  - **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**

Idea di soluzione

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