Criteri avanzati di sincronizzazione

SCD

Anno accademico 2019/2020
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@unipd.it

Valutazione critica – 1

- I costrutti di un linguaggio possono essere valutati da due punti di vista
- Potere espressivo
  - Capacità di soddisfare bisogni applicativi
- Usabilità
  - Qualità di interazione (efficacia) e grado di integrazione (coerenza) tra loro

Valutazione critica – 2

- Applichiamo 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, https://doi.org/10.1145/800215.806566
- L'autrice individua 6 tipi di vincoli influenti sulla sincronizzazione
  - Tutti con potere espressivo superiore alla *exclusion synchronization*

Condizioni di sincronizzazione – 1

- In funzione dello stato di sincronizzazione della risorsa ☑
  - #utenti correnti e #richieste in attesa (`'Count`) rispetto a disponibilità (aka molteplicità);
- In funzione dello stato logico della risorsa ☑
  - *Buffer is empty vs. buffer is full*
- In funzione della storia d'esecuzione nella risorsa ☑
  - Riflessa nella valorizzazione delle guardie

Condizioni di sincronizzazione – 2

- In funzione del tipo di richiesta ☑
  - Riflessa nella valorizzazione della guardia (e.g., preferenze a letture piuttosto che a scritture)
- In funzione del tempo della richiesta ☑
  - Espressa con politiche di accodamento sulle code di guardia o di ordinamento dei clienti
- In funzione dei parametri della richiesta ☒
  - Dove la disponibilità dipende non solo dal tipo di richiesta ma anche dalla sua dimensione

L'allocazione delle risorse – 1

- Problema ricorrente in ogni modello di programmazione concorrente
  - Coinvolge tutte le 6 dimensioni di interesse
  - Difficile da trattare solo con guardie

Esempio:
Un controllore deve assegnare un numero staticamente fissato N di risorse a un insieme di clienti $\{C_{i=1,\dots,M}\}$ concorrenti.
Ogni cliente C_i può richiedere $1 \leq n_i \leq N$ risorse alla volta.
Se accettata, la richiesta deve essere soddisfatta integralmente, altrimenti essa deve essere tenuta in sospeso (e il cliente bloccato) sino a quando ne diventi possibile il soddisfacimento.

L'allocazione delle risorse – 2

- Nel problema dato, il volume di richiesta viene presentato come parametro di ingresso della richiesta
- Per valutare se la richiesta sia soddisfacibile, occorre leggere il parametro e quindi accettare la sincronizzazione
  - Che fare a quel punto ove la richiesta non fosse immediatamente soddisfacibile?
  - Il principio di usabilità aborre il "*busy wait*"

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 (e 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***
- **Almeno due soluzioni**
  1. **L'espressione di guardia accede ai parametri in ingresso della richiesta**
  2. **Il processo servente trasferisce la chiamata (accettata ma non soddisfacibile) su altra coda**
     - 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 1 (N risorse per richiesta)

```
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*

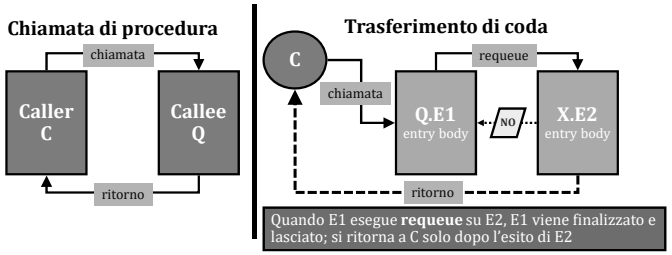
L'allocazione delle risorse – 6

- **La soluzione 1 ha costo insostenibile**
  - **Legare l'accodamento di chiamata a come il valore di parametri della richiesta concorda con lo stato corrente della risorsa, richiede di rivalutare tutte le richieste accodate a ogni cambio di stato**
    - Una sorta di `notifyAll()` in peggio

- **Assai meglio trasferire ad altra coda la richiesta accettata ma non soddisfacibile**
  - **Trasferendo la richiesta con singola operazione atomica**
  - **Senza valutare l'eventuale guardia di destinazione**

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` (di server, di RP)**
     - Per maggior coesione funzionale conviene restare nell'entità di partenza
   - **Il trasferimento verso coda di altra entità causa il rilascio dell'entità di partenza**
     - Necessario, ma potenzialmente indesiderabile
   - **Il canale destinazione deve avere interfaccia compatibile con quello di partenza**
     - LSP: la stessa della chiamata inziale oppure vuota
   - **La direzione del trasferimento (interna o esterna) determina il destinatario finale del *lock* originale**

Semantica del trasferimento di coda – 4

2. **Come trattare il *time-out* eventualmente 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

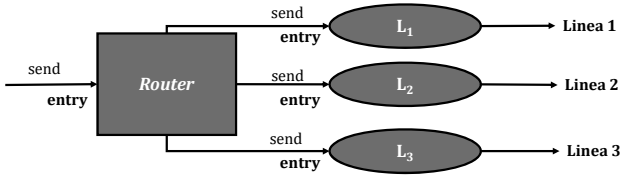
Questa clausola preserva l'eventuale *time-out* posto dal chiamante sulla coda di destinazione

Esempi d'uso – 1

- Algoritmo di allocazione di risorse con trasferimento di coda
- Vedere l'esempio associato alla lezione
- Migliorare la soluzione proposta, evitando trasferimenti di coda 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 di coda offre grande potere espressivo per trattare 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
  - L'arrivo del treno in una stazione abilità il trasferimento dei viaggiatori in attesa in essa (fino alla capacità massima del treno) nella coda della stazione di loro destinazione, dalla quale saranno rilasciati (= arrivati) all'arrivo del treno