Università degli Studi di Padova

Programmazione concorrente

SCD

Anno accademico 2004/5
Corso di Sistemi Concorrenti e Distribuiti

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

Premesse - 1

- L'espressività di un linguaggio (sia naturale che di programmazione) ne è anche il limite
  - Ciò che il linguaggio non prevede o non consente, di fatto non esiste
  - Questo è anche il caso della concorrenza
- Molti linguaggi "storici" sono sequenziali
  - Il flusso di programma è unico per ogni esecuzione, anche se dati variabili ne possono modificare il cammino

Premesse - 2

- La realtà, invece, è intrinsecamente concorrente, così come lo sono i moderni elaboratori con i loro dispositivi ed i sistemi operativi
  - Vi è una varietà di tecniche per rendere effettivo il parallelismo inerente di molte attività
- Quale scelta per il progetto di un linguaggio di programmazione inteso per sistemi dotati di concorrenza interna?

Linguaggi concorrenti - 1

- Un linguaggio sequenziale può esprimere concorrenza tramite l'uso di librerie di sistema, sia entro un programma che tra programmi distinti
  - L'espressione ed il controllo della concorrenza si pone in questo modo al di fuori del linguaggio
  - Problemi semantici e di portabilità
- Un linguaggio concorrente è invece capace di esprimere direttamente la presenza di più flussi di controllo coesistenti all'interno di uno stesso programma

Linguaggi concorrenti - 2

- Il progetto di un linguaggio concorrente necessita di un modello di concorrenza di riferimento
  - Molte scelte e forme sono possibili
  - La programmazione concorrente è di grande aiuto per rappresentare adeguatamente l'attività di sistemi complessi, ma è anche difficile ed esposta al rischio di importanti errori concettuali
  - Per questo occorre che il modello di riferimento sia valido

Forme di concorrenza - 1

- Chiameremo processo il singolo flusso di controllo all'interno di un programma
- Concettualmente, l'esecuzione di un processo può prevedere che:
  a) Tutti i processi condividano uno stesso elaboratore
  b) Ciascun processo possieda un elaboratore proprio e tutti gli elaboratori condividano un banco di memoria comune
  c) Ciascun processo possa avere un elaboratore proprio e gli elaboratori, pur connessi, non condividano memoria

Università degli Studi di Padova

Programmazione concorrente

Forme di concorrenza - 2

- Ciascuna di queste 3 forme (ed i loro ibridi) richiedono metodi di implementazione diversi
  - Definiamo paralleli quei processi che, ad ogni istante, sono simultaneamente in esecuzione → i casi b) e c)
  - Definiamo concorrenti quei processi che sono capaci di esecuzione parallela
- Parallelismo → concorrenza realizzata
- Concorrenza → parallelismo potenziale

Università degli Studi di Padova

Programmazione concorrente

Forme di concorrenza - 3

- La concorrenza è dunque più generale del parallelismo, pertanto concettualmente più importante

> Programmazione concorrente è il nome dato a notazioni e tecniche usate per esprimere parallelismo potenziale e per risolvere i problemi di sincronizzazione e comunicazione correlati con tale espressione.
>
> Il vantaggio della programmazione concorrente è di consentire lo studio del parallelismo senza doversi confrontare con le relative problematiche di implementazione.
>
> Ben-Ari, *Principles of Concurrent Programming*, 1982

Università degli Studi di Padova

Programmazione concorrente

Forme di concorrenza - 4

- La programmazione concorrente non è il solo modo di sfruttare *hardware* parallelo
  - Parallelismo a grana grossa → parallelismo a grana fine
- Altri modelli di elaborazione (*computational model*) sono più adatti ad ambiti di calcolo parallelo
  - Processori vettoriali (*vector processors*)
  - Architetture *data-flow*

Università degli Studi di Padova

Programmazione concorrente

Forme di concorrenza - 5

- Principi di base di ingegneria del *software* richiedono l'uso di strumenti e metodi di sviluppo adatti alle caratteristiche del dominio applicativo
- Applicazioni inerentemente concorrenti sono p.es. quelle il cui *software* interagisce direttamente con componenti *hardware* interne ed esterne → sistemi *embedded*

Università degli Studi di Padova

Programmazione concorrente

Un modello di concorrenza - 1

- Entità attive
  - Sono capaci di intraprendere azioni di propria iniziativa (se forniti delle necessarie risorse di elaborazione)
- Entità reattive
  - Eseguono azioni solo in risposta ad esplicite richieste
  - Risorse → hanno stato interno ed impongono condizioni di accesso (p.es. mutua esclusione)
  - Entità passive → non impongono condizioni di accesso

Università degli Studi di Padova

Programmazione concorrente

Un modello di concorrenza - 2

- L'implementazione di entità risorse richiede capacità di controllo sulle condizioni di accesso
  - Agente di controllo
- Agente di controllo come entità passiva
  - Risorsa protetta (p.es. un semaforo)
- Agente di controllo come entità attiva
  - *Server* (uno speciale processo)

Un modello di concorrenza - 3

Tipo Entità		Implementazione
Attiva		Processo
Reattiva	Risorsa protetta	Modulo con agente di controllo (attivo o passivo)
	Server	Processo
	Passiva	Modulo senza agente di controllo

Il progetto di un programma concorrente che usi questo modello comporta l'immediato riconoscimento di queste entità nel problema

Un modello di concorrenza - 4

- **L'implementazione di questo modello richiede fino a 3 categorie di primitive**
  - **Processo (entità attiva)**
  - **Agente di controllo passivo per risorse protette**
    - Basso livello di astrazione
    - Efficienza d'esecuzione
    - Inflessibilità
  - **Agente di controllo attivo**
    - Alto livello d'astrazione
    - Proliferazione di processi, con elevati costi di gestione e d'esecuzione
    - Flessibilità

Espressione di concorrenza - 1

- ***Coroutine***
  - **Una delle prime (e più rudimentali) modalità espressive di concorrenza a programma**
  - **L'alternanza d'esecuzione tra strutture concorrenti doveva essere esplicita tramite comando `resume`**
  - **Modella una tecnica di rappresentazione della simulazione discreta → SIMULA (1967)**
  - **Presente in un linguaggio concorrente "storico" Modula-2, ma inadeguata alla programmazione concorrente**

Espressione di concorrenza - 2

- **Dichiarazione ed attivazione di processo**

Algol68, CSP, Occam

```
cobegin
  P1; P2; P3;
coend;
```

Attivazione esplicita Distinta dalla dichiarazione

Ada

```
procedure Main is
  task A;
  task B;
  …
  task A is … ;
  task B is … ;
begin
  …
end Main;
```

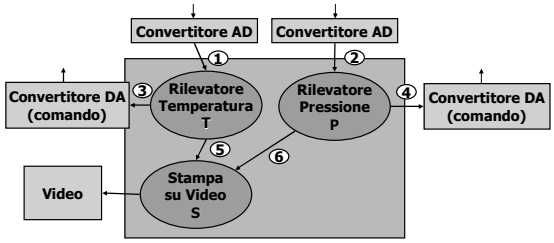
Dichiarazione

Implementazione

Attivazione implicita

A questo punto `Main`, `A` **e** `B` **sono 3 processi concorrenti**

Un esempio - 1



T e P devono mantenere temperatura e pressione del sistema entro limiti di sicurezza e stampare a video i valori correnti

Un esempio - 2

- **T e P sono entità attive**
- **S è una entità passiva (*server* o risorsa protetta)**
- **Vi sono almeno 3 possibili implementazioni**
  - **Completamente sequenziale → ignorando il parallelismo potenziale di T, P ed S**
  - **T, P ed S scritti in linguaggio sequenziale, ma trattati come processi distinti tramite chiamate al sistema operativo**
  - **L'intero sistema descritto in linguaggio concorrente**

Università degli Studi di Padova

Programmazione concorrente

Un esempio - 3

- **Vediamo le 3 possibili implementazioni …**
  - **Sequential_Controller**
  - **OS_Controller**
  - **Concurrent_Controller**

Università degli Studi di Padova

Programmazione concorrente

Un esempio - 4

- **Soluzione completamente sequenziale**
  - **Forza un ordinamento artificioso (p.es. prima controllo temperatura, poi controllo pressione) tra moduli indipendenti**
  - **Può ritardare o impedire del tutto l'esecuzione di azioni indipendenti ma programmate come successive**
  - **Non tiene conto di possibili differenze nel ciclo operativo di produzione dei dati (p.es. controllo temperatura 2 secondi, controllo pressione ogni 5 secondi)**
  - **Tenerne conto a programma comporterebbe gravi oneri di attesa attiva (*busy wait*)**

Università degli Studi di Padova

Programmazione concorrente

Un esempio - 5

- **Soluzione con primitive di sistema operativo**
  - **Assai migliore rispetto alla soluzione sequenziale**
    - Non comporta attese attive
    - Preserva la separazione logica tra moduli tra loro indipendenti
  - **Il controllo dell'esecuzione concorrente è demandato al sistema operativo**
    - Leggere il programma non ci aiuta a capirne il funzionamento: l'invocazione di servizi di sistema operativo ne ostacolo la comprensione
    - Il comportamento effettivo del programma a tempo di esecuzione dipende dall'implementazione del modulo OS

Università degli Studi di Padova

Programmazione concorrente

Un esempio - 6

- **Soluzione in linguaggio concorrente**
  - **L'intera logica dell'applicazione è inscritta nel programma**
    - Secondo regole garantite dal linguaggio di programmazione
  - **Se le operazioni di lettura di valori da dispositivi esterni (1 e 2) sono non bloccanti allora il codice di ciascun processo (*task*) di controllo può riflettere indipendentemente la proprio frequenza di operazione**
  - **La soluzione però non è ancora soddisfacente perché fa ipotesi troppo semplicistiche sulla risorsa S**
    - Attualmente modellata come entità passiva, il che non è realistico

Università degli Studi di Padova

Programmazione concorrente

Un primo raffronto

- **Fattori a favore della concorrenza espressa a linguaggio**
  - **Programmi più leggibili → maggiore manutenibilità**
  - **Indipendenza dal sistema operativo → maggiore portabilità ed idoneità all'uso in ambienti a risorse ristrette**
- **Fattori contrari all'espressione di concorrenza a linguaggio**
  - **Il linguaggio deve assumere uno specifico modello di concorrenza → perdita di generalità**
  - **La realizzazione del modello può confliggere con il supporto offerto dal sistema operativo sottostante → difficile implementazione**

Università degli Studi di Padova

Programmazione concorrente

La dimensione temporale - 1

- **Molti sistemi devono coordinare la propria esecuzione con la nozione del tempo insita nell'ambiente di esecuzione**
  - **Tipicamente concretizzata da (almeno) un orologio che approssima il trascorrere del "tempo"**
  - **Questo orologio diventa per il programma la sorgente del valore "tempo"**
  - **Varie scelte possono essere fatte dal linguaggio per rappresentare questo "tempo"**
    - Per esempio: ora del giorno espressa in secondi e frazioni nell'arco di 24 ore
    - Oppure: maggiore granularità (accuratezza), tempo monotono crescente

Università degli Studi di Padova

Programmazione concorrente

La dimensione temporale - 2

```
declare
   Start, Finish : Ada.Calendar.Time;
   Interval : Ada.Calendar.Duration := 2.5;
begin
   Start := Ada.Calendar.Clock;
   … -- actual activity
   Finish := Ada.Calendar.Clock;
   if Finish – Start > Interval then
      raise Overrun_Error;
   end if;
end;
```

Questa tecnica assume implicitamente un solo flusso di controllo!

```
Ada.Real_Time.Time;
Ada.Real_Time.Time_Span := Ada.Real_Time.To_Time_Span(2.5);
Ada.Real_Time.Time_Span := Ada.Real_Time.Milliseconds(2500);
```


Università degli Studi di Padova

Programmazione concorrente

La dimensione temporale - 3

- **L'orologio può anche essere usato a fini di sincronizzazione temporale**
  - **Sospensione relativa**

```
delay 10.0; -- valore di tipo Ada.Calendar.Duration
```

    - Dal tempo in cui viene presa in considerazione
    - Il linguaggio garantisce una durata della sospensione non inferiore alla richiesta
  - **Sospensione assoluta**

```
delay until A_Time; -- valore di tipo Ada.Real_Time.Time
```

    - Riferisce un valore dato tempo indipendente dall'esecuzione della richiesta
    - Il linguaggio garantisce un tempo di risveglio non precedente alla richiesta

Università degli Studi di Padova

Programmazione concorrente

La dimensione temporale - 4

```
Start := Clock;
First_Action;
delay until (Start + 10.0);
Second_Action;
```

A

```
Start := Clock;
First_Action;
delay (Start + 10.0 - Clock);
Second_Action;
```

B

A e B non hanno lo stesso effetto perché , in presenza di prerilascio, è azione interrompibile ed il valore assunto da `Clock` quando verrà valutato non è prevedibile a priori

Università degli Studi di Padova

Programmazione concorrente

La dimensione temporale - 5

Avendo accesso all'orologio ed usando sospensioni assolute possiamo facilmente programmare vere attività periodiche

```
with Ada.Real_Time; use Ada.Real_Time;
…
task body Periodic_Task is
   Suspension_Period : constant Time_Span := Millisecond(10_000);
   Next_Time : Time;
begin
   Next_Time := Clock + Suspension_Period;
   loop
      Periodic_Action;
      delay until Next_Time;
      Next_Time := Next_Time + Interval;
   end loop;
end Periodic_Task;
```


Università degli Studi di Padova

Programmazione concorrente

La dimensione temporale - 6

- **La regolarità temporale delle attività periodiche è affetta da 2 fattori di rischio**
  - **Deviazione locale (*local drift*)**
    La distanza effettiva che separa due successive invocazioni dell'attività
    - **Inevitabile**, può essere migliorata solo mediante migliore implementazione del linguaggio (granularità di accesso all'orologio)
  - **Deviazione cumulativa (*cumulative drift*)**
    L'effetto a catena causato dalla possibile varianza nel completamento delle attività precedenti
    - **Evitabile** tramite l'uso di sospensioni assolute, come nel riquadro precedente