Software Architectures

Concorrenza e Sincronizzazione

Mauro Migliardi Ph. D.

Definizione

- Un programma “ordinario” consiste di dichiarazioni e istruzioni che vengono eseguite sequenzialmente
- Un programma concorrente consiste di un insieme di programmi ordinari che vengono eseguiti in potenziale parallelismo

Programmazione Concorrente: serve davvero?

- Sovrapposizione I/O calcolo
  - Espressi come processi “paralleli”
- Multiprogrammazione
- Time Slicing
- Multitasking
  - Se completamente indipendenti => no problem
  - Altrimenti => comunicazione/sincronizzazione

Perche’ un’astrazione?

- Complessita’ ingestibile
  - Esplosione combinatoria
  - Non determinismo generalizzato
- Astrazione per modellare complessita’
  - Aumento lineare del numero di stati
  - Non determinismo isolato in specifiche parti del programma
  - Concorrente o parallelo non fa piu’ differenza

Astrazione Programmazione Concorrente

- Un processo di 3 moduli di 10 istruzioni
  - 10^3 = 1000 possibili stati
- Tre processi di un modulo di 10 istruzioni
  - Un solo programma per volta = 10 stati possibili
- Possibili situazioni critiche
  - Contesa
  - comunicazione

Astrazione Programmazione Concorrente

- Esecuzione simultanea/intermittente
- Azioni atomiche
- Processi sequenziali
- Scala dei tempi ininfluente
- ***Un programma concorrente deve essere corretto con qualsiasi possibile sequenza di istruzioni***
  - Facile dimostrare che e' sbagliato
  - Difficile dimostrare che e' corretto

Correctness

- ***Safety***
  - ***Non succede niente di male***
- ***Liveness***
  - ***Prima o poi succede qualcosa di buono***
- ***Partial Correctness***
  - ***Non termina mai con una soluzione sbagliata***
  - ***E' una safety property***
- ***Total Correctness***
  - ***Termina sempre e con la soluzione giusta***
  - ***E' la combinazione di una safety ed una liveness***

Scendiamo dal Pero

- Mutua esclusione

X = 0;

x = x + 1;	x= x + 1;
process(x);	process(x);

Che pid sono stati processati?

Due Possibili Sequenze

Load a, x

Inc a

Store x, a

Load b, x

Inc b

Store x, b

Due Possibili Sequenze

Load a, x

Load b, x

Inc b

Store x, b

Inc a

Store x, a

Mutua esclusione

- A software
  - Ci sono algoritmi che non vediamo qui
  - Complessa
  - Pesante dal punto di vista prestazionale
- Con hardware di support
  - Test and set
  - Swap

Primitive di supporto

- Semafori (Dijkstra 1968)
- Variabile intera SEMPRE POSITIVA
- Due primitive:
  - Wait (p nella notazione originale)
  - Signal (v nella notazione originale)

Definizione

- Wait:
  - If(S > 0)
    - { s = s –1; }
  - Else
    - <sospenditi>
- Signal
  - If(<esiste almeno un processo sospeso)
    - { <sveglia un processo> }
  - Else
    - s = s + 1;

Mutua Esclusione con Semafori

S = 1;

- While(true)
- {
  - <sezione non critica>
  - Wait(S);
  - <sezione critica>
  - Signal(S);
- }

Dimostrazione

- Invarianti dei Semafori
  - S >= 0;
  - S = S0 + #signal - #wait
- Dimostrazione
  1. Proviamo che #CS + S = 1 e' invariante
  2. Poiche' S>=0, quello di sopra implica #CS <=1
  3. #CS = #Wait - #Signal

     Quelli che sono entrati – quelli che sono usciti
  4. S = 1 + #Signal - #Wait l'invariante sopra
  5. S = 1 - #CS ho sostituito 3 in 4

Buone Notizie

- Unica dimostrazione del corso ;-)
  - Sort of…
- Sottigliezze semantiche dei semafori
  - Non le vediamo

Produttore consumatore

```
While(true) {
Wait(spazio);
Buffer[i] = <produci>;
i = (i + 1)%n;
Signal(roba);
}
```

```
While(true) {
Wait(roba);
<consuma>(Buffer[j]);
j = (j + 1)%n;
Signal(spazio);
}
```