Programmazione Concorrente

ITT MARCONI
TDP – 4SI

Sistema concorrente

- In informatica la **concorrenza** è una caratteristica dei sistemi di elaborazione nei quali può verificarsi che un **insieme di processi** o sottoprocessi (thread) computazionali sia **in esecuzione nello stesso istante**.
- L'esecuzione parallela può condurre a **interazione tra processi** quando viene coinvolta una **risorsa condivisa**.

Esempio dei filosofi

- Il problema dei filosofi a cena, un classico problema inerente concorrenza e condivisione di risorse
- Cinque filosofi siedono ad una tavola rotonda con un piatto di spaghetti davanti, una forchetta a destra e una forchetta a sinistra.

- cinque filosofi
- cinque piatti di spaghetti
- cinque forchette.

Problema dei cinque filosofi

- la vita di un filosofo consiste di periodi alterni di mangiare e pensare, e che ciascun filosofo abbia bisogno di due forchette per mangiare
- le forchette vengano prese una per volt
- dopo essere riuscito a prendere due forchette il filosofo mangia per un po', poi lascia le forchette e ricomincia a pensare.

→ PROBLEMI?

Deadlock e Starvation

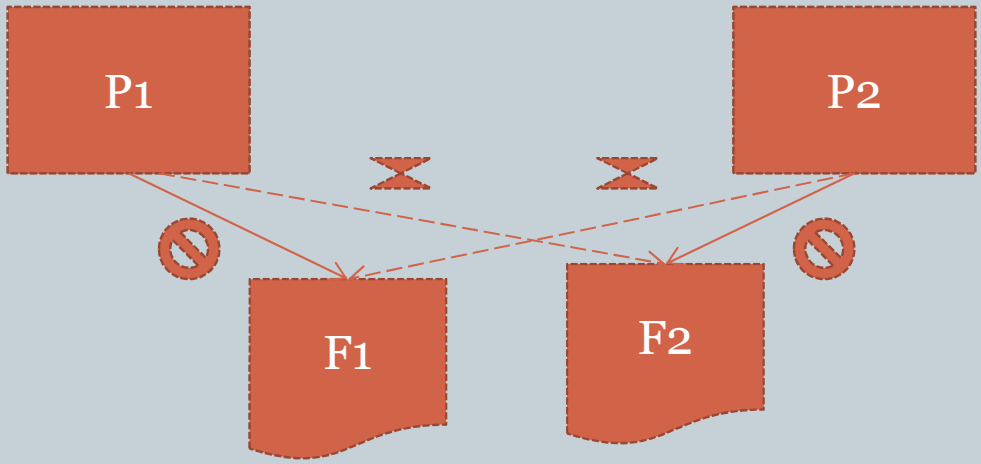
- Bisogna impedire lo stallo (deadlock) o la morte d'inedia (starvation).
- Il deadlock può verificarsi se ciascuno dei filosofi tiene in mano una forchetta senza mai riuscire a prendere l'altra. Il filosofo F_1 aspetta di prendere la forchetta che ha in mano il filosofo F_2, che aspetta la forchetta che ha in mano il filosofo F_3, e così via in un circolo vizioso.
- La situazione di starvation può verificarsi indipendentemente dal deadlock se uno dei filosofi non riesce mai a prendere entrambe le forchette.

Analogia con programmazione concorrente

- La presa di forchette è analoga al blocco di risorse limitate nella programmazione reale (concorrenza).
- Bloccare una risorsa è una tecnica comune per garantirne l'accesso da parte di un programma solo o di una sola porzione di programma alla volta.
- Se la risorsa richiesta da un programma è già stata bloccata, il programma aspetta fino a quando la risorsa si sblocca.
- Se il blocco coinvolge più di una risorsa, è possibile che in alcune circostanze si verifichi un deadlock.

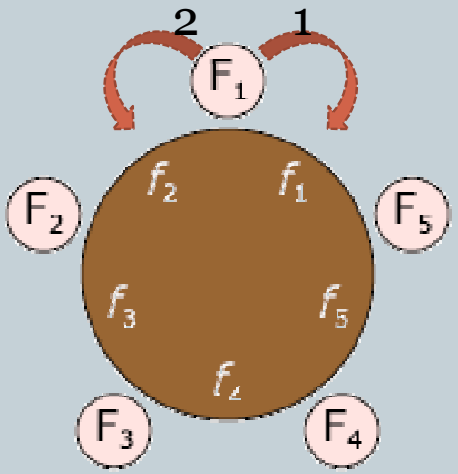
Esempio di deadlock

- Come esempio si consideri il caso di un programma che ha bisogno di due file da elaborare. Se due programmi di questo genere bloccano un file ciascuno, entrambi aspetteranno invano che l'altro programma sblocchi l'altro file.



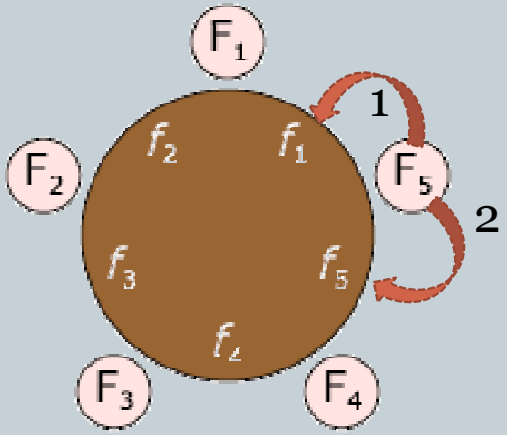
Soluzione

- Numerare le forchette ed esigere che vengano prese in ordine numerico crescente, analogamente ad un caso di allocazione gerarchica delle risorse. In questa soluzione i filosofi sono denominati F1, F2, F3, F4 e F5, mentre le forchette alla loro sinistra sono rispettivamente f1, f2, f3, f4 e f5.
- Il primo filosofo F1 dovrà prendere la prima forchetta f1 prima di poter prendere la seconda forchetta f2.



Cont..

- I filosofi F2, F3 e F4 si comporteranno in modo analogo, prendendo sempre la forchetta fi prima della forchetta fi+1.
- Rispettando l'ordine numerico ma invertendo l'ordine delle mani, il filosofo F5 prenderà prima la forchetta f1 e poi la forchetta f5.
- Si crea così un'**asimmetria** che **serve ad evitare i deadlock.**



Prevenire Starvation

- L'efficacia nella prevenzione della starvation dipende dal metodo di **mutua esclusione** utilizzato.
- Le implementazioni che fanno uso di *spinlock*, ovvero cicli di attesa, possono provocare starvation a causa di problemi di sincronia inerenti a tali metodi.
- Altri metodi di mutua esclusione che utilizzano **code di attesa** possono prevenire la starvation in modo più efficiente assicurando uguale accesso alle forchette da parte di tutti e due i filosofi adiacenti.

Accesso esclusivo

- Regole per consentire ad un n agenti (processi) di ottenere accesso esclusivo ad m risorse:

1. Tutti gli agenti devono richiedere accesso alle risorse in ordine crescente, prima cioè di richiedere l'accesso ad una risorsa di ordine maggiore, l'agente deve aver ottenuto l'accesso alle risorse di ordine minore di cui ha bisogno.
2. Prima di richiedere l'accesso a una risorsa di ordine minore, l'agente deve rilasciare le risorse di ordine maggiore alle quali sta accedendo.

Programmazione Concorrente e Distribuita

La programmazione concorrente nasce per gestire i ***Sistemi Concorrenti*** cioe' sistemi in grado di supportare piu' utenti (o programmi) ***contemporaneamente***

- ***Sistemi intrensecamente concorrenti***
  - Sistemi Real Time
  - Sistemi operativi
  - Gestione di basi di dati
- ***Applicazioni potenzialmente concorrenti***
  - ***Uso di algoritmi paralleli per computazioni:***
    - su grandi quantita' di dati
    - con grande mole di calcolo
    - vincoli di tempo reale

Programmazione Concorrente e Distribuita

Per ***sistema concorrente*** intendiamo:

- un sistema *software*
- implementato su *vari tipi di hardware*
- che *porta avanti* ***contemporaneamente*** una molteplicita' di attivita' diverse
- tra di loro *correlate*
  - possono *cooperare* ad un goal comune
  - possono *competere* per risorse condivise

Cosa cambia?

- Il concetto di **concorrenza** è contrapposto a quello di **sequenzialità**.
- In un sistema sequenziale i processi vengono eseguiti uno per volta e non si verifica alcuna forma di interazione tra essi durante l'esecuzione.

- Un programma sequenziale:

→ sequenza di dichiarazioni e istruzioni che verranno eseguite sequenzialmente

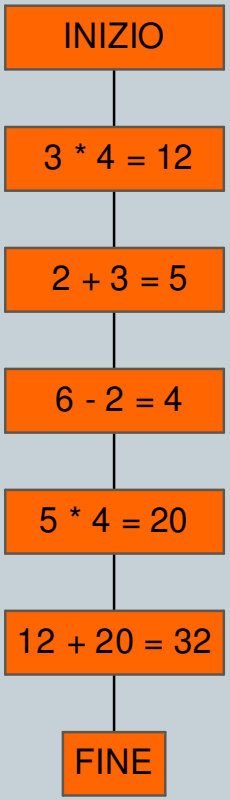
- Un programma concorrente:

→ un insieme di programmi sequenziali che verranno eseguiti in parallelo

Esempio: da sequenziale a concorrente

- Supponiamo di scrivere un programma che valuta delle espressioni.
- Normalmente scriveremmo un processo sequenziale, il cui grafo totalmente ordinato, cioè ogni nodo avrà un predecessore (eccetto il nodo iniziale) ed un successore (eccetto il nodo finale).

Grafo di precedenza ad ordinamento totale del processo che valuta l'espressione:
(3 * 4) + (2 + 3) * (6 - 2)



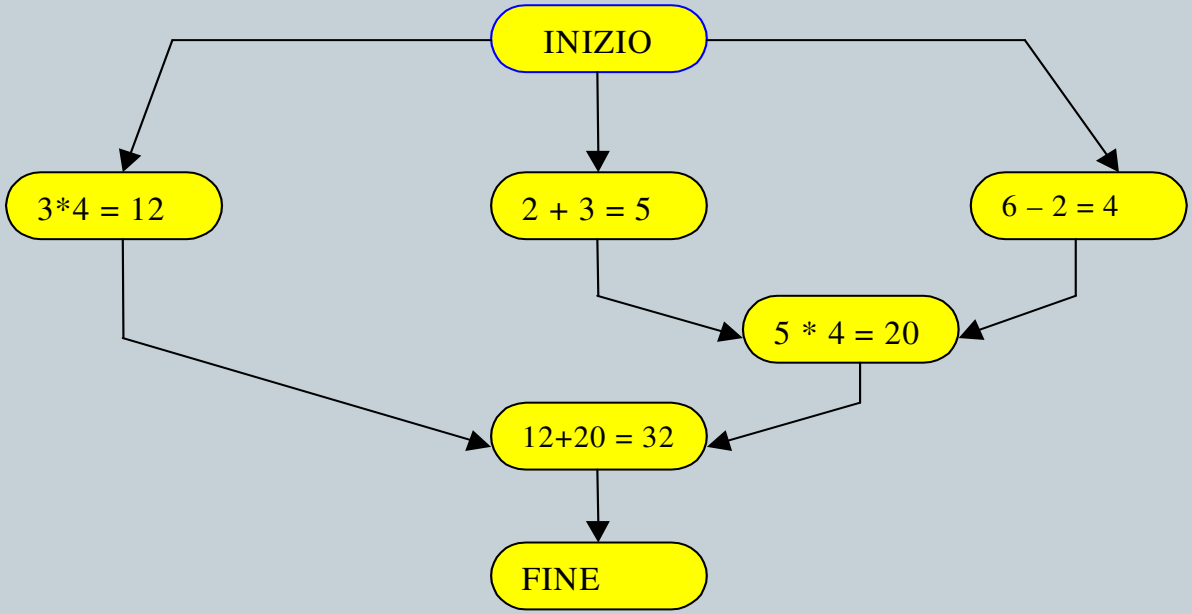
Individuare processi concorrenti

- L’ ordinamento totale del grafo è solo in parte dovuto alla natura del problema da risolvere, in parte è dovuto alla natura sequenziale del calcolatore.

- La natura del problema cioè impone di fatto solo un **ordinamento parziale** tra gli eventi.

Grafo di precedenza ad ordinamento parziale

- Ottengo dei risultati parziali in modo PARALLELO
- Li metto assieme seguendo un ordine
- Per ottenere un risultato totale



Programmazione Concorrente

Abbiamo visto che:

- alcuni problemi possono essere risolti mediante processi di calcolo non ***sequenziali*** *cioè* rappresentati da un *grafo ad ordinamento parziale*
- in questo caso il problema puo' essere risolto **da alcuni *moduli sequenziali* che lavorano *in parallelo***.
- *O*ccorre quindi *un* ***elaboratore parallelo***, in grado cioe' di eseguire un numero arbitrario di operazioni contemporaneamente.
- Occorre quindi *un* ***linguaggio di programmazione*** con il quale poter descrivere questi algoritmi non sequenziali.

Lo studio di questi linguaggi, dei loro compilatori e delle loro applicazioni prende il nome di ***Programmazione Concorrente***.