Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it

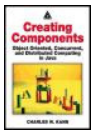
Argomenti

- Programmazione concorrente
- (Programmazione ad eventi)
- Programmazione distribuita

- Libri di testo:
  - un qualunque libro che copra gli argomenti trattati materiale disponibile tramite la piattaforma di e-learning di ateneo http://elearning.uninsubria.it

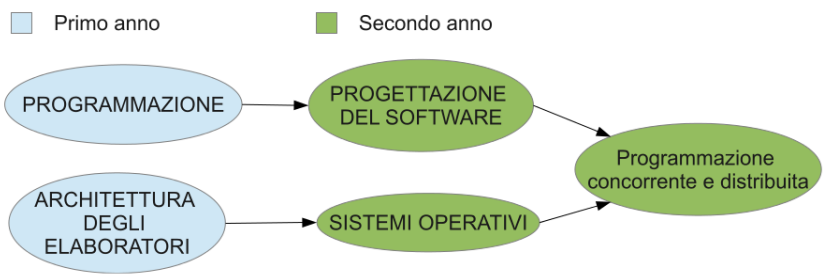
Riferimenti

- Gli argomenti trattati in questo corso sono tratti in parte dai seguenti libri:
- **Creating Components: Object Oriented, Concurrent, and Distributed Computing in Java**, by Charles W. Kann
- **Concurrent and Real-Time Programming in Java**, by A. Wellings
- **Concurrent and Distributed Programming in Java**, by Vijay K. Garg
- **Thinking in (Enterprise) Java**, by Bruce Eckel et. Al.,



Prerequisiti

- Padroneggiare la **programmazione Object Oriented in Java**
- Avere una buona conoscenza dei principi di funzionamento di un **Sistema Operativo**, in particolare i meccanismi di supporto alla concorrenza, ad esempio, processi, semafori, etc.
- Sapere cosa sono i **pattern** per la progettazione del software
- Conoscere **UML**



Ricevimento

- Su appuntamento
- Per email (luigi.lavazza@uninsubria.it)
  - per questioni puntuali
- Dove:
  - Dipartimento di Scienze Teoriche e Applicate
  - Via Mazzini, 5
- Il ricevimento è una opportunità
  - Usatela!

Organizzazione

- 8 crediti
  - 6 crediti di lezione (48 ore)
  - 2 crediti di esercitazione (24 ore)

Modalità di esame

- Prova scritta
  - Riguardante tutti gli argomenti trattati nel corso.
  - Svolta sui PC del laboratorio.
    - viene inoltre richiesto di rispondere a domande vertenti sugli argomenti affrontati.
    - viene chiesto di scrivere alcuni piccoli programmi
- Solo nel caso la prova scritta lasci dei dubbi sulla reale preparazione dello studente potrà svolgersi una prova orale.
  - esclusivamente a discrezione del docente
- Durante le lezioni e le esercitazioni potranno essere proposti quiz ed esercizi non obbligatori, che se risolti con successo, potranno fruttare un "bonus" che si aggiungerà al voto della prova scritta.

Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Domande ?

INTRODUZIONE AGLI ARGOMENTI DEL CORSO

Programma del corso

- **Programmazione concorrente**
  - Thread, scheduling, accesso sincronizzato a dati condivisi, comunicazione tra thread.
- **Programmazione distribuita**
  - Stream e serializzazione, socket, RMI.
- **Programmazione ad eventi**
  - Gli eventi come paradigma di comunicazione tra thread o come strumento per la realizzazione di GUI
- NOTA: il programma potrà subire aggiustamenti nel corso del semestre.

Programmazione Concorrente: motivazioni

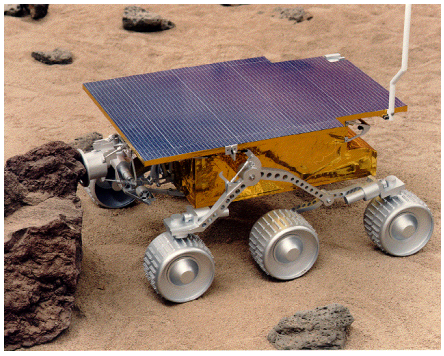
- Motivazioni principali per scrivere programmi concorrenti:
  - Ci sono tante cose da fare (contemporaneamente) e un processore è abbastanza potente da occuparsene
    - Concorrenza «virtuale»
    - Processi cooperanti
  - Per modellare il parallelismo del mondo reale
    - Alcuni programmi devono controllare e interfacciarsi con entità del mondo reale (robot, nastri trasportatori, ecc.) che sono intrinsecamente paralleli.
  - Quando si ha a disposizione più di un processore, per sfruttare il parallelismo reale

Sequenziale vs. concorrente

- Batch processing
  - Dati di input disponibili preventivamente
- Programmi interattivi
  - I/O "estemporaneo"
  - Ci sono periodi in cui il programma aspetta che l'utente fornisca un dato necessario per proseguire con l'elaborazione

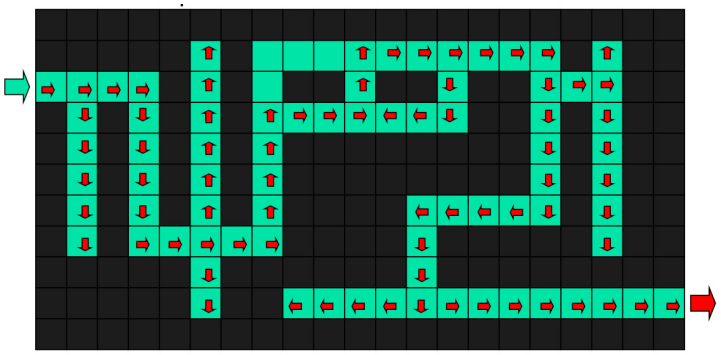
Parallelismo nel mondo reale

- Bisogna fare tante cose contemporaneamente:
  - Gestire il movimento
  - Comunicare
  - Leggere i sensori
  - Analizzare i campioni
  - Gestire l'energia
  - Ecc.
- NB: anche in casi più banali:
  - In un'automobile
  - In una lavatrice
  - ...



Sfruttare il parallelismo reale

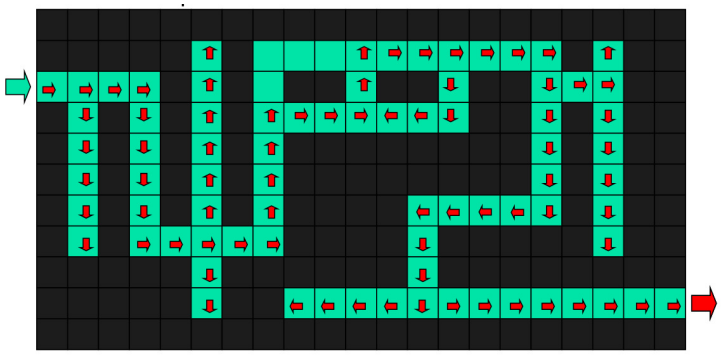
- Problema: trovare la strada in un labirinto



- Con un approccio sequenziale, ogni volta che trovo un bivio devo esplorare *prima* un'alternative e *poi* l'altra.

Sfruttare il parallelismo reale

- Problema: trovare la strada in un labirinto



- Con un molti processori, ogni volta che trovo un bivio uso un processore per esplorare un'alternativa e un secondo processore per esplorare l'altra.

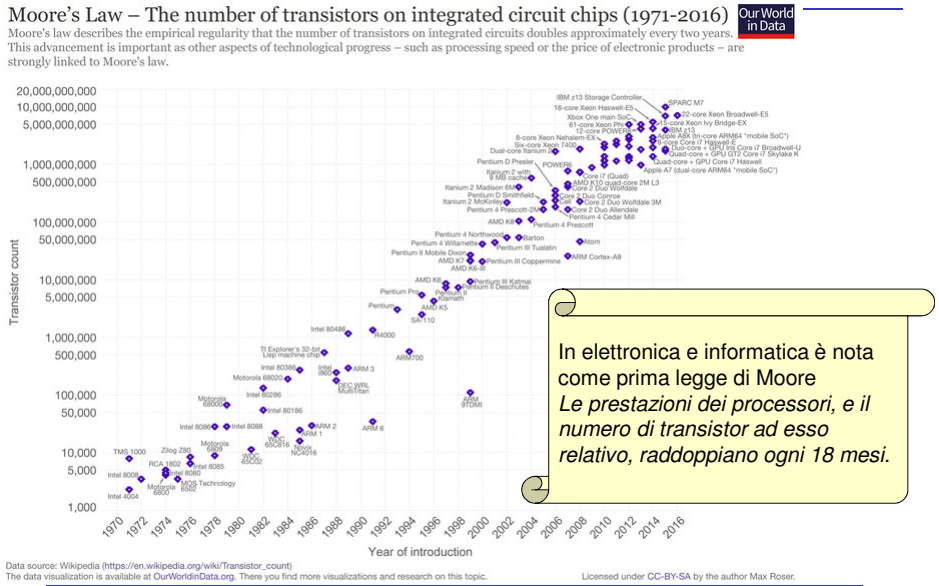
Motivazioni

- Velocità
  - Se si desidera un programma veloce, conviene spezzarlo in modo da poter eseguire ogni pezzo su un processore separato.
  - La concorrenza è uno strumento fondamentale per la programmazione multiprocessore.
- Gestione del design
  - Alcuni tipi di problemi sono difficili da risolvere senza l'uso della concorrenza.
    - Es., simulazione e controllo.
  - In questi casi la progettazione del programma si semplificata notevolmente.

Velocità

- Cercare i numeri primi richiede una grande massa di calcoli.
- Soluzione: distribuirla su tutti i computer aderenti al progetto di ricerca
  - Es. PrimeGrid (https://www.primegrid.com/)

Legge di Moore



In elettronica e informatica è nota come prima legge di Moore
Le prestazioni dei processori, e il numero di transistor ad esso relativo, raddoppiano ogni 18 mesi.

Legge di Moore

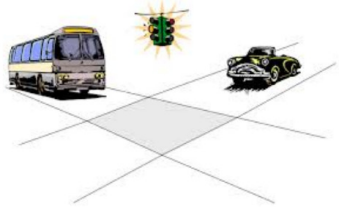
- I limiti della prima legge di Moore starebbero solo nel raggiungimento dei limiti fisici imposti per la riduzione delle dimensioni dei transistor
- Tali limiti sarebbero peraltro già stati raggiunti con la generazione dei processori Pentium
- l'unico modo possibile e praticabile per aumentare le prestazioni di calcolo ed elaborazione dati è rappresentato dalla tecnologia multicore ovvero dall'accoppiamento in parallelo di più processori
- Per rendere i programmi più veloci, dovrete imparare a sfruttare i processori in più, o meglio imparare la programmazione concorrente

Programmazione Concorrente: criticità e problemi

- Processi concorrenti e cooperanti devono coordinare le loro azioni

comunicando

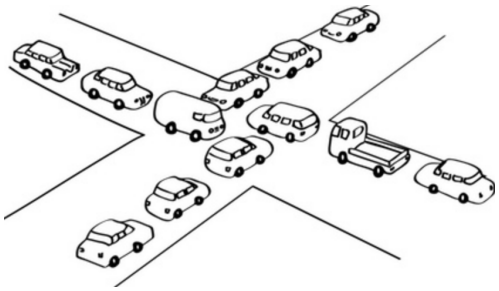
sincronizzandosi



- Queste azioni possono portare a nuove condizioni di errore.

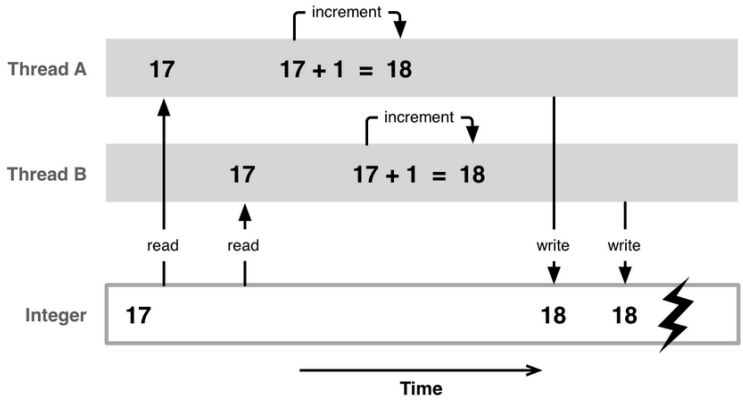
Programmazione Concorrente: criticità e problemi

- Deadlock
  - si verifica quando ciascuna attività concorrente è in attesa del risultato di un'altra attività per poter eseguire un'operazione.



Programmazione Concorrente: criticità e problemi

- Race condition
  - Si verifica quando due o più attività simultanee tentano di aggiornare lo stesso oggetto: questo può corrompere i dati dell'oggetto



Programmazione Concorrente: criticità e problemi

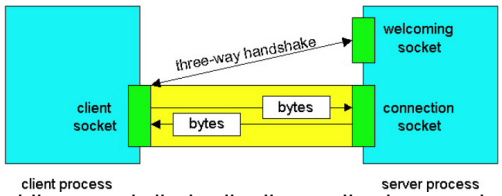
- Starvation
  - Ha luogo accadere quando a una o più attività simultanee vengono continuamente negate risorse come risultato delle azioni degli altri.

Comportamento desiderato dei programmi concorrenti

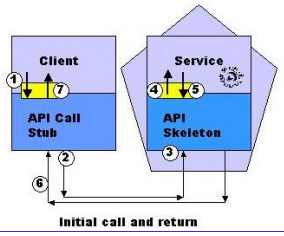
- Il comportamento desiderato di un programma concorrente è caratterizzato da due proprietà:
  - **safety** - le attività concorrenti non devono interferire tra di loro con il rischio di corrompere i dati o generare altre conseguenze sgradite
  - **liveness** - tutte le attività concorrenti devono essere in grado di fare progressi con le loro attività e non rimanere bloccati in deadlock o starvation.
- Es.: passaggio a livello
  - Safety: deve essere chiuso quando passa il treno
  - Liveness: deve essere aperto quando possibile (un passaggio a livello sempre chiuso è assolutamente sicuro, ma poco utile...)

Programmazione in ambiente Distribuito

- Un programma distribuito ha componenti che girano contemporaneamente su più computer.
- La programmazione distribuita si è evoluta da meccanismi semplici quali i socket



- a tecniche che sfruttano i linguaggi di alto livello quali ad esempio Java Remote Method Invocation (RMI)



Strumenti Java

- Libreria **`java.net`**
  - classi **`Socket`**, **`ServerSocket`**, **`DatagramSocket`**, **`DatagramPacket`**
- Serializzazione
  - scrivere/leggere un oggetto di una qualsiasi classe (che implementi l'interfaccia **`java.io.Serializable`**)
- Caricamento dinamico delle classi
  - possibilità di caricare una nuova classe scaricata dalla rete dinamicamente (es. le applet)
- RMI (Remote Method Invocation)

L'organizzazione Client-Server

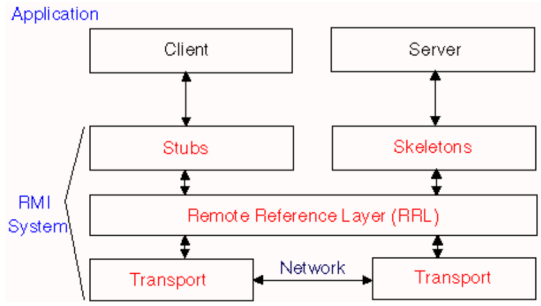
- **Server**: Entità che mette a disposizione delle risorse o dei servizi
- **Client**: Entità che richiede risorse e servizi al server
- Le richieste e le risposte seguono un protocollo di comunicazione comune, utilizzato sia al client che al server

- Esempio: il Web
  - Il server web mette a disposizione le pagine HTML (ed altri dati multimediali).
  - Il client (browser web) richiede le pagine HTML e le visualizza.
  - Il protocollo di comunicazione è l'HTTP

RMI (Remote Method Invocation)

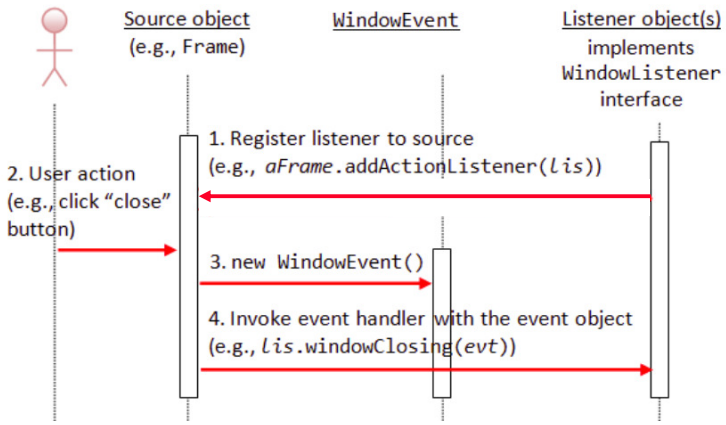
- Implementa l'RPC (Remote Procedure Call) nel contesto object-oriented:
  - Invece di una procedura remota si può invocare un metodo di un oggetto remoto
- Il client può invocare un metodo su un oggetto remoto in modo trasparente
  - Allo stesso modo dell'invocazione su un oggetto locale

Tutte le procedure per la realizzazione dell'RMI sono gestite automaticamente da Java



Eventi come Paradigma di Comunicazione

- I thread possono comunicare attraverso il meccanismo degli eventi



Conclusioni

- Abbiamo visto una panoramica sugli argomenti che saranno trattati in questo corso
- Alcuni di questi argomenti potranno essere trattati in modo molto più approfondito di altri
- Le lezioni ed esercitazioni che seguiranno affronteranno nel dettaglio questi argomenti.