PCD Module 1.1

Concurrent Programming - Overview

Concurrent programming

- **Concorrenza**: proprietà dei sistemi in cui diversi processi computazionali vengono **eseguiti nello stesso istante**, potenzialmente **interagendo** tra di loro.
- **Programmazione concorrente**: costruire programmi in cui più attività si *sovrappongono* nel tempo e *interagiscono* in qualche modo.
- **Programmi concorrenti**: insieme finito di programmi sequenziali (**processi**) eseguibili in parallelo (*processi paralleli*)
- **Tipologie di programmazione**
    - *Concorrente*: v. sopra. Non necessariamente i programmi vengono eseguiti in processori fisici separati. Focus sull'organizzazione, livello logico/astratto/programmazione.
    - *Parallela*: esecuzione sovrapposta su processori fisici separati. Focus sulle performance, livello fisico.
    - *Distribuita*: processori distribuiti su una rete, nessuna memoria condivisa.
- Concurrent vs. Parallel, pg. 6
- **Paradigmi**
    - *Programmazione multi-threaded*: stato condiviso, meccanismi di sync (sem_t, monitor)
    - *Programmazione basata sui messaggi*: no stati condivisi, interazione tramite scambio messaggi, sync/async
    - *Programmazione guidata dagli eventi*: flusso del programma determinato dagli eventi (azioni, sensori, messaggi da altre app)
    - *Programmazione asincrona*: progettare programmi con azioni e richieste async (never blocking dogma, future mechanisms & callbacks)
    - *Programmazione reattiva*: il flusso è progettato attorno a flussi di dati e alla propagazione dei cambiamenti

Architetture parallele

- **Multi-core**: più core sullo stesso chip, condividendo RAM ed eventualmente livelli di cache
- **Core eterogenei**: processore standard + processori *attached* specializzati (es. CPU e GPU, CUDA)
- **Supercomputer**: labs + grandi aziende, grande numero di processori e architetture diverse.
- **Clusters/grid**: composto da più computer interconnessi (es. server Apple). Memoria e risorse non condivise, comunicazione tramite scambio messaggi.
- **Cloud computing**: potenza di calcolo fornita come servizio in rete: risorse, software e informazione condivise. SAAS, PAAS e IAAS; cloud private/pubbliche, esempi (EC2, Azure, AppEngine).

Tassonomia di Flynn

Categorizzazione dei sistemi basata sul numero dei flussi di informazioni e dei flussi dati:

- **SISD**: modello Von Neumann, singolo processore.
- **SIMD**: singolo flusso di istruzioni trasmesse concorrentemente a più processori, ognuno con il proprio flusso dati.
- *MISD*: teorico.
- **MIMD**: ogni processore ha il suo flusso di informazioni operante su un flusso dati.
    - *a memoria condivisa*: tutti i processi condividono un singolo spazio degli indirizzi e comunicano su variabili condivise
        - *SMP - architettura simmetrica multiprocesso*: condivisione connessione a memoria condivisa + accesso alla stessa velocità
        - *NUMA - accesso alla memoria non uniforme*: memoria condivisa, alcuni blocchi di memoria possono essere fisicamente più vicinamente associati a certi processori
    - *a memoria distribuita*: ogni processo ha il proprio spazio degli indirizzi e comunica con gli altri tramite scambio messaggi
        - *MPP - processori massivamente paralleli*: infrastrutture altamente specializzate e accoppiate, scalabili (HPC)
        - *Clusters*
        - *Grid*: sistemi che usano risorse eterogenee distribuite tramite LAN/WAN senza un punto comune di amministrazione.

Motivazioni

- *Miglioramento delle performance*: aumento produttività e reattività. Misura: **speedup** ($S = \frac{T_1}{T_N}$)
    - **Legge di Amdahl**: massimo speedup nella parallelizzazione di un sistema $S = \frac{1}{1-P+\frac{P}{N}}$.
    - significa che serializzazione/sequenzializzazione sono un danno alle performance, ma a volte necessarie per la correttezza --> sforzi e compromessi
    - senza dimenticarsi dell'efficienza, $E = \frac{S}{P}$
    - *collo di bottiglia*: la memoria. Solo 1 operazione di memoria per volta --> importanza della cache
- *Progettazione e astrazione*: definizione livello di astrazione dei programmi che interagiscono con l'ambiente.
    - Concorrenza come strumento per la progettazione e la costruzione.
    - Entrambi influenzano l'intero spettro ingegneristico (modellazione, progettazione, implementazione, verifica, testing)

Gergo

- **Processo**: un programma sequenziale in esecuzione, unità base di un sistema concorrente, singolo thread logico. Unità di lavoro, task, concetto astratto/generale
    - *indipendenza dalla velocità*: l'esecuzione è completamente asincrona, non si possono fare previsioni sulla velocità --> **nondeterminismo**.
- **Interazione**: ogni programma concorrente è basato su processi multipli che hanno bisogno di interagire, in qualche modo

- *Cooperazione*: interazioni previste e ricercate, parte della semantica del programma concorrente
    - *Comunicazione*: realizzazione flusso di informazioni tra processi, tipicamente realizzato in termini di messaggi
    - *Sincronizzazione*: definizione esplicita di relazioni temporali/dipendenze tra processi e azioni. Introduzione supporto allo scambio di segnali temporali.
- *Competizione*: interazioni previste e necessarie, ma non ricercate. Necessità di coordinamento dell'accesso da processi multipli a risorse condivise.
    - *Mutex*: regolazione accesso a risorse condivise da processi distinti
    - *Sezioni critiche* : regolazione esecuzione concorrente di blocchi di azioni da processi distinti
        - Differenze
        - Esempio del latte + soluzione "note nel frigo"
- *Interferenza*: interazione inaspettate e non necessarie, producono cattivi effetti solo con determinati valori. **heisenbugs**
    - *Race conditions*: 2+ processi accedono e aggiornano concorrentemente risorse condivise. Il risultato del singolo aggiornamento dipende dall'ordine di accesso. Causato da cattiva gestione delle interazioni previste e presenza di interazioni spurie disattese nel problema.
- *Situazioni critiche*
    - *Deadlock*: 2+ azioni aspettano l'altra per finire, e nessuno lo fa (rilascio di risorse condivise bloccate)
    - *Starvation*: processo bloccato in attesa infinita
    - *Livelock* : simile a deadlock, ma lo stato cambia costantemente in accordo con l'altro

Storia

- Motivazione originaria: *sviluppare OS affidabili*
- 1961: nascita multiprogrammazione (interrupt), ASM
- 1965-1979: concetti fondamentali per PCD
- 1970s: primi libri su PCD

Linguaggi, meccanismi e astrazioni concorrenti

- Per eseguire un programma concorrente, serve una **macchina concorrente**
    - che fornisce:
        - supporto all'esecuzione di programmi concorrenti
        - processori virtuali
        - **meccanismi base**
- Meccanismi base:
    - **multiprogrammazione**: meccanismi che rendono possibile creare nuovi processori virtuali e allocarci processori fisici a basso livello tramite scheduling.

    - **sincronizzazione e comunicazione**
        - *modello a condivisione di memoria* tra processori virtuali (es. multithreading)
        - *modello a scambio di messaggi*: nessuna memoria fisica, scambio di messaggi tra processori

- Per descrivere un programma concorrente, serve un **linguaggio concorrente**
    - programmi organizzati come set di processi sequenziali eseguiti concorrentemente sui processori virtuali
    - costrutti base per specificare
        - **concorrenza** (processi multipli)
        - **interazione dei processi** (sync, comunicazione, mutex)

Linguaggi: approcci progettuali

- linguaggio sequenziale + libreria con primitive concorrenti (es. C + PThread)
- linguaggio progettato per la concorrenza (es. OCCAM, ADA, Erlang)
- approccio ibrido
    - paradigmi sequenziali estesi con supporto nativo per la concorrenza (es. Java, Scala)
    - librerie e pattern basati su meccanismi base (es. `java.util.concurrent`)

Notazioni base e costrutti

fork/join ('60/'70)

- **fork**: comportamento simile a invocazione procedure, con differenza che un nuovo processo è creato e attivato per eseguire la procedura. Input: procedura da eseguire, output: ID del processo creato. Biforcazione del flusso del programma: il processo figlio è eseguito async.
- **join**: intercetta processi creati con fork che sono terminati e sincronizza il flusso di controllo.
- pro: generale e flessibile
- contro: basso livello di astrazione, programmi difficili da leggere, nessuna rappresentazione esplicita del processo di astrazione

cobegin/coend ('60/'70)

Concorrenza "a blocchi": le istruzioni all'interno di un blocco **cobegin/coend** sono eseguite in parallelo, terminando solo quando tutti i componenti (processi) hanno finito.

- pro: disciplina più forte nello strutturare un programma concorrente rispetto a fork/join. Programmi più leggibili.
- contro: meno flessibilità di fork/join. Non c'è astrazione esplicita che incapsula il processo.

Linguaggi con supporto di prima classe per i processi

Processi come "entità di prima classe" del linguaggio, "moduli" per organizzare il programma e il sistema. Incapsulazione esplicita del flusso di controllo. Esempi: Concurrent Pascal, OCCAM, ADA, Erlang

Concorrenza nei linguaggi popolari

- I linguaggi popolari (C, Java, ecc.) supportano la creazione e l'esecuzione dei processsi tramite librerie.
- Supporto per *programmazione multithreading*: thread come implementazione di processo lightweight - non da

confondere con i processi di sistema.
- Esempi: multithreaded programming in Java, Pthread in C/C++

Programmazione multithreaded in Java

- Primo linguaggio popolare a fornire supporto nativo a programmazione concorrente, approccio conservativo.
- Processo implementato come **thread**, con mapping diretto sui thread dell'OS.
- Supporto esteso dal 2005 tramite *Java Concurrency Utility*

Threads in Java

- Modello
  - singolo flusso di controllo che condivide la memoria con tutti gli altri thread (stack privato, heap in comune)
  - ogni programma Java contiene almeno 1 thread
  - altri thread possono essere creati e attivati dinamicamente a runtime

- Definizione
  - oggetti che estendono la classe Thread fornita in `java.lang.Thread`

- Esecuzione
  - gli oggetti possono essere istanziati e generati invocando il metodo `start()`

Programmazione multithreaded in C/C++ + Pthreads

- Fornisce primitive nbasilari per programmazione multithreaded in C/C++
- Processo implementato come thread; corpo del processo specificato per mezzo di procedure
- Lo standard definisce interfacce e specifiche, non l'implementazione - la quale dipende dagli OS.

Funzioni

- Interfaccia definita in `pthread.h`
- Tipi di dato:
  - `pthread_t` - thread identifier data type
  - `pthread_attr_t` - data structure for thread attributes

- Funzioni principali:
  - creazione del thread (fork): `pthread_create(pthread_t* tid, pthread_attr_t* attr, void* (*func)(void*), void* arg)` e `pthread_attr_init(pthread_attr_t*)`
  - terminazione del thread: `pthread_exit(int)`
  - congiunzione: `int pthread_join(pthread_t thread, void ...)`

Oltre ai thread: uno scenario ricco

- Framework basati sui **task** (es. Java Executor)
- **Programmazione asincrona** (modelli event-loop, AsyncTasks)
- **Attori** (OOP concorrente, variante ad oggetti attivi)
- **Scambio di messaggi sincrono** (processi + s.m.s.)

- **Programmazione reattiva** (es. programmazione reattiva funzionale, estensioni reattive - Rx)
- **Calcolo parallelo** (GPGPU Programming - OpenCL, Lambda Architecture/Paralellism for BigData - MapReduce)

Modello ad attori

- Scambio messaggi asincrono tra oggetti reattivi chiamati attori
  - tutto è un attore, con ID unico e unica mailbox
  - ogni interazione ha luogo come scambio messaggi asincrono

- Poche primitive: **send**, **create**, **become**
- Sviluppato tra gli anni 70 e 90
- Diversi framework: Akka, Erlang, HTML5 Web Workers, DART Isolates

Concorrenza logica vs. fisica

I paradigmi di programmazione concorrenti moderni promuovono una visuale logica sulla concorrenza, ad alto livello. Si possono così avere migliaia di attori in esecuzione sulla stessa macchina.

Oggetti attivi

Concorrenza + OO. Oggetti attivi + passivi, creazione implicita dei thread + meccanismi di sincronizzazione.