UNICA
UNIVERSITÀ
DEGLI STUDI
DI CAGLIARI

sAIfer Lab
Joint lab on Safety and Security of AI

Processi Asincroni e Parallelismo

Maura Pintor
maura.pintor@unica.it

Gestione di utenti multipli e concorrenza

Oltre all'evoluzione delle comunicazioni e dei software dei web server, c'è stato un grosso cambiamento anche nel **numnero di connessioni** ricevute dai web server

Nel caso di connessioni multiple, ci sono due caratteristiche da tenere sotto controllo:

- **latenza**, ovvero il tempo totale di attesa tra richiesta e risposta
- **throughput**, ovvero quanti byte al secondo sono trasmessi

Per limitare questi ritardi di risposta, si utilizzano tecniche per evitare le *busy waiting*, ovvero i tempi in cui il server non può accettare richieste perché sta processando le richieste degli altri utenti

Gestione di utenti multipli e concorrenza

La normale esecuzione in Python è sincrona: i comandi vengono eseguiti in ordine di arrivo e il comando successivo non inizia finché non è iniziato il precedente

- esecuzione bloccante, non fa altro fino a che non termina l'esecuzione

Nel caso della gestione di utenti multipli, il metodo asincrono è molto meglio

- le richieste vengono ricevute e prese in carico con alta priorità, ma non bloccano la CPU
- vengono processate con priorità minore (le operazioni di lettura/scrittura disco hanno più latenza delle operazioni in RAM
- la risposta sarà inviata in seguito all'utente

Esempi di utilizzo dei processi asincroni

- Operazioni di I/O intensive: Lettura/scrittura su database o file system
- Chiamate API esterne: Riduzione della latenza nelle richieste HTTP
- Elaborazione in background: Generazione di report, invio di email, processamento di immagini
- Streaming di dati: WebSocket, tecnologia che consente una comunicazione bidirezionale e persistente tra client e server, rendendoli ideali per gli aggiornamenti in tempo reale

L’utilizzo di framework

Per gestire la concorrenza, la gestione di utenti multipli, e tante altre cose, non è raccomandabile scrivere tutto il codice partendo da “zero”

Solitamente si sceglie un framework, ovvero un isieme di software e funzionalità che sono pensate per lavorare assieme per lo sviluppo accelerato

Nella pratica, tra i vantaggi ci sono:

- performance migliori, alcune funzionalità sono ottimizzate e fanno girare codice compilato per svolgere delle funzioni predefinite
- sviluppo più veloce, perché consente il riutilizzo di grossi pezzi di codice già scritti
- migliore qualità del codice e minore probabilità di bug
- gestione di alto livello, non dobbiamo implementare troppe cose di basso livello (per esempio la gestione asincrona)

Come vengono implementati i processi asincroni nei web server

Programmazione basata su Callback

- Funzioni che vengono eseguite una volta completata un'operazione
- possono portare a problemi di "callback hell" (quando si annidano molte funzioni di callback all'interno di altre, rendendo il codice difficile da leggere e mantenere)

Promises e Async/Await

- Promises: Struttura che rappresenta un'operazione asincrona completata o fallita
- Async/Await: Semplifica la scrittura del codice asincrono, rendendolo più leggibile

Utilizzo di code: job Queue e Task Scheduling

- Esempi: Celery (Python), Sidekiq (Ruby), Bull (Node.js)
- Utilizzati per gestire compiti in background in modo efficiente

WebSockets e Event-Driven Programming

- WebSocket permette una comunicazione bidirezionale in tempo reale tra client e server
- Modelli event-driven come Node.js utilizzano il loop degli eventi per gestire richieste asincrone

Web applications e componenti di connessione con i web server

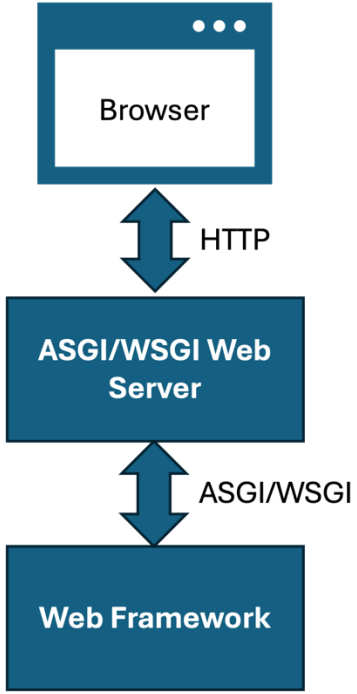
Per connettere il **web server** con l'**applicaizone web** che vedrà l'utente, ci serve un'interfaccia che metta in comunicazione questi. due pezzi

Questa interfaccia prende il nome di Interfaccia Gateway e consente di separare l'applicazione dal framework usato

Interfaccia Gateway

- Traduzione di protocolli
- Routing e smistamento delle richieste
- Sicurezza e autenticazione
- Bilanciamento del carico

Questo significa che cambiando il framework, si può scrivere la stessa applicazione senza modificarne l'aspetto e il modo in cui gli utenti ci devono interagire



Web Server Gateway Interface (WSGI)

Standard per l'interfacciamento tra web server e applicazioni **Python**

Caratteristiche:

- Definisce una interfaccia standard tra server web e framework Python
- Asincrono e sincrono
- Utilizzato da framework come Flask e Django

Limitazioni:

- Gestione sequenziale delle richieste
- Performance ridotte per applicazioni ad alto carico

Asynchronous Server Gateway Interface (ASGI)

Evoluzione moderna di WSGI per **applicazioni asincrone**

Vantaggi principali:

- Supporto nativo per async/await
- Gestione contemporanea di più richieste
- Migliori performance per applicazioni I/O intensive
- Compatibile con WebSocket (connessione bidirezionale full-duplex, ogni parte può inviare e ricevere dati nello stesso momento) e server push (server invia dati al client senza richiesta esplicita)
- Framework principali: **FastAPI**, Starlette

Migliore di WSGI perché:

- Maggiore scalabilità
- Gestione efficiente delle connessioni e prestazioni superiori in ambienti ad alto carico
- Supporto a protocolli moderni

Parallelismo, concorrenza, e supporto asincono

Nel calcolo *parallelo*, un task si può distribuire in sotto-task tra più CPU dedicate

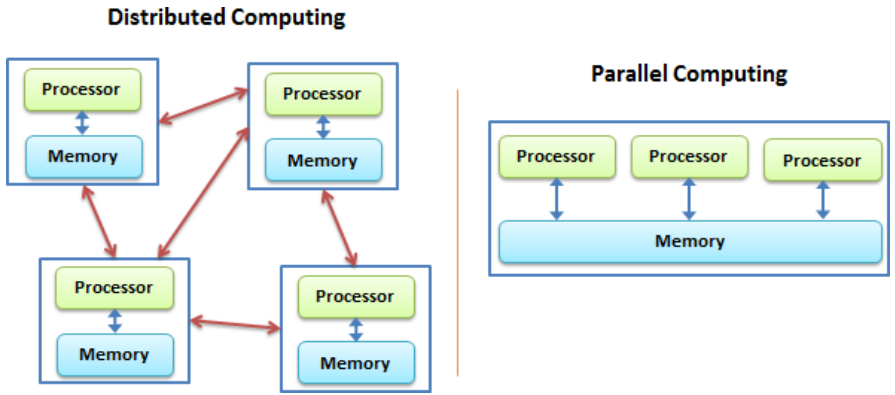
Nel calcolo *concorrente*, ogni CPU alterna diversi task

- alcuni task impiegano più tempo
- vogliamo evitare il più possibile i tempi di attesa
- le applicazioni web hanno tanti di questi task lunghi
- come possiamo ottimizzare le tempistiche?

Calcolo parallelo e distribuito

Quando l'applicazione diventa complicata, o quando si prevedono molti utenti, è consigliato spezzettarla in più task che possono essere svolti indipendentemente

- in diverse CPU su una sola macchina (calcolo parallelo)
- o addirittura su più macchine (calcolo distribuito)

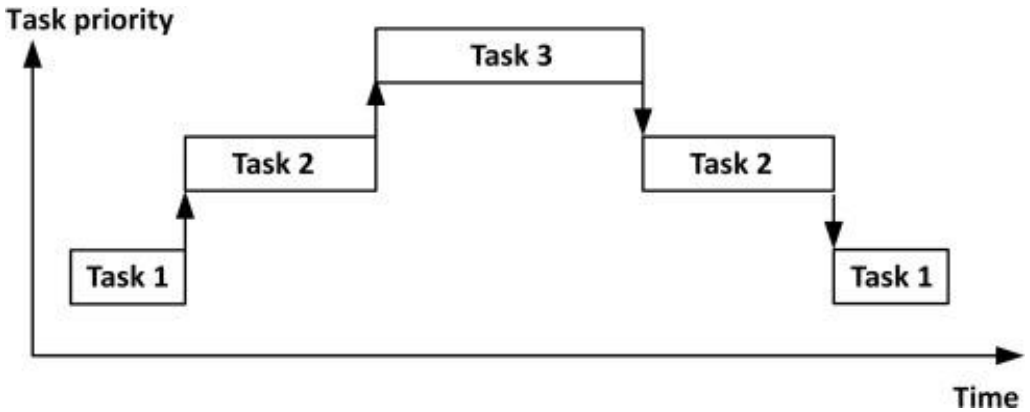


I processi nel sistema operativo

Nel sistema operativo le risorse (memoria, CPU, device, scheda di rete, ...) vengono "prenotate"

Ogni programma viene eseguito in uno o più processi che utilizzano queste risorse

- molti sistemi usano l'allocazione "**interrompibile**" (preemptive)
- i processi hanno priorità dinamiche
- processi a bassa priorità possono essere **sospesi** a favore di processi ad alta priorità



I thread del sistema operativo

Processi:

- Istanze di programmi in esecuzione
- Memoria separata
- Maggiore isolamento
- Overhead di creazione più alto
- Comunicazione inter-processo complessa

Thread:

- Unità di esecuzione all'interno di un processo
- Condividono memoria e risorse
- Comunicazione più veloce
- Creazione più leggera

Esempio di thread asincroni

```
import asyncio
import random

async def tell_joke_part(part_number):
    parts = ["Why", "did", "the", "chicken", "cross", "the", "road"]
    await asyncio.sleep(random.random())
    print(parts[part_number])
    return parts[part_number]

async def main():
    jokes = [tell_joke_part(i) for i in range(7)]
    results = await asyncio.gather(*jokes)
    print("Async Joke:", ' '.join(results))

await main()
```


https://colab.research.google.com/drive/1M47lj4BPoZevKW5DqmXeElqarpXLBjja?usp=sharing

Esempio di thread asincroni

Le keyword async e await servono a gestire codice asincrono in Python:

async
- Definisce funzioni che possono essere interrotte
- Permettono esecuzione non bloccante
- Restituiscono oggetti coroutine

await
- Sospende esecuzione di funzione asincrona
- Attende completamento di operazione asincrona
- Utilizzabile solo dentro funzioni async*

*Il Colab notebook è esso stesso una async, per questo motivo usiamo la `await main()` alla fine.
Di solito si usa direttamente `asyncio.run(main())`

Thread multipli nei web server

Web Server Multithread

- Gestiscono richieste multiple simultaneamente
- Ogni richiesta HTTP come thread separato
- Alta concorrenza
- Efficienza nell'I/O
- Utilizzo ottimale risorse CPU
- Scalabilità orizzontale (aggiunta di più macchine / nodi)

WSGI: Thread sequenziali

ASGI: Thread asincroni non bloccanti

Importante: L'uso di async e await non velocizza il codice di per sé. Anzi, potrebbe rallentarlo leggermente a causa dell'overhead di configurazione asincrona. Il principale vantaggio è evitare attese lunghe durante operazioni di I/O.

Asynchronous JavaScript And XML (AJAX)

AJAX è un meccanismo per aggiornare le pagine web in modo asincrono, tramite lo scambio dati con un web server

- aggiorna solo una parte della pagina, senza ricaricarla per intero

AJAX non è un linguaggio di programmazione, ma una semplice combinazione di due elementi:

- la richiesta XMLHTTP del browser (che richiede dati dal web server)
- JavaScript e HTML (per mostrare i dati agli utenti)

Nonostante il nome, non richiede l'uso esclusivo di XML, ma può essere anche usato il plain text o il JSON

Come funziona AJAX

1. AJAX è attivato da un evento (per esempio un pulsante premuto dall'utente)
2. JavaScript crea la richiesta e la invia al server
3. il server risponde alla richiesta e la rimanda al browser
4. La risposta è letta tramite JavaScript, che aggiorna la pagina