

Multithreading

Capitoli 4 -- Silberschatz

Multithreading: estensione del concetto di processo

- Alla base del concetto di thread sta la constatazione che la definizione di **processo** è basata su due aspetti:
 - Possesso delle risorse
 - Esecuzione
- I due aspetti sono indipendenti e come tali possono essere gestiti dal SO
 - L'elemento che viene eseguito è il **thread**
 - L'elemento che possiede le risorse è il **processo**
- Il termine multithreading è utilizzato per descrivere la situazione in cui ad un processo sono associati più thread

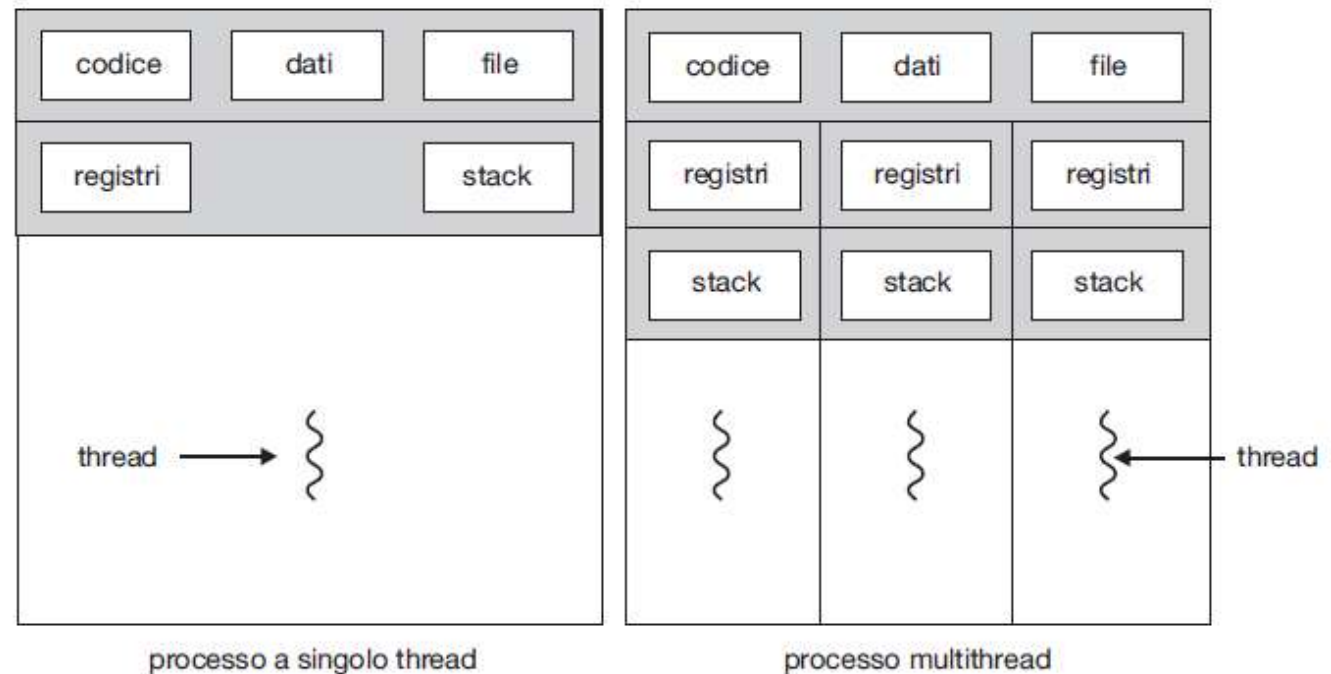
Multithreading: esempi

La maggior parte delle applicazioni attuali sono multithread

- **Web browser** potrebbe avere:
 - un thread per la rappresentazione sullo schermo immagini e testo,
 - un thread per reperire i dati dalla rete.
- **Word processor** potrebbe avere:
 - un thread per ciascun documento aperto (i thread sono tutti uguali ma lavorano su dati diversi)
 - Relativamente ad ognuno è anche possibile avere...
 - Un thread per la rappresentazione dei dati su schermo
 - Un thread per la lettura dei dati immessi da tastiera
 - Un thread per la correzione ortografica e grammaticale
 - Un thread per il salvataggio periodico su disco

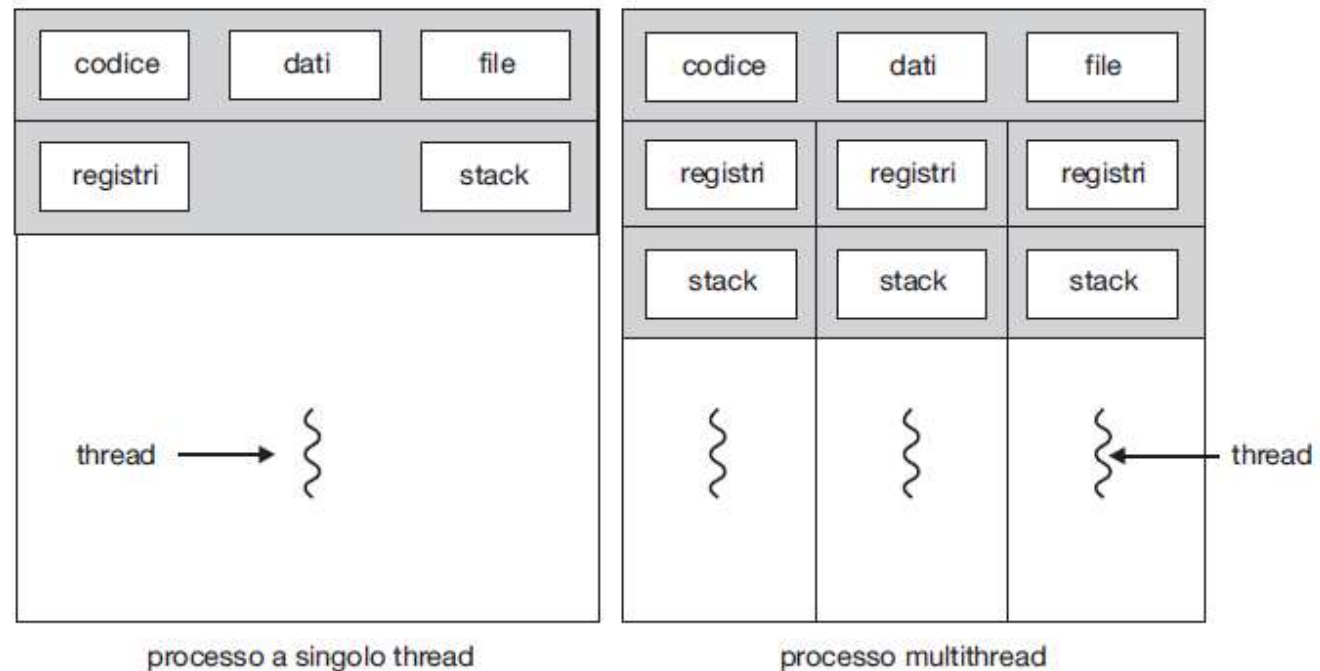
Multithreading

- Tutti i thread
 - condividono le risorse del processo,
 - risiedono nello stesso spazio di indirizzamento ed
 - hanno accesso agli stessi dati



Multithreading

- Ad ogni thread è però associato
 - proprio **descrittore**
 - un program counter,
 - uno stato di esecuzione,
 - uno spazio di memoria per le variabili locali,
 - uno stack,
 - un contesto



Multithreading

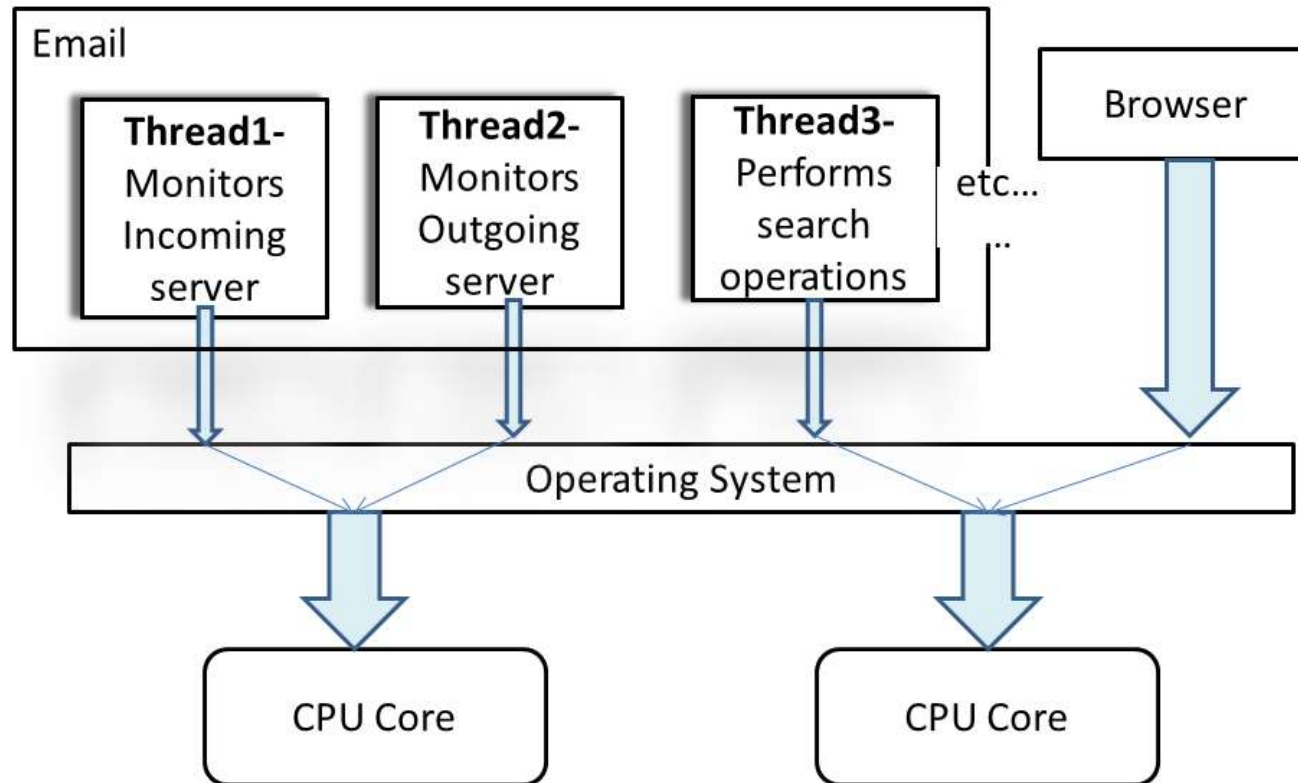
- Le informazioni associate al thread sono poche
 - le operazioni di cambio di contesto, di creazione e terminazione sono più semplici

Multithreading

- Esistono applicazioni che, presentando un intrinseco grado di parallelismo, possono essere decomposte in attività che procedono concorrentemente, condividendo un insieme di dati e risorse comuni
- **Esempio:** Applicazioni in tempo reale per il controllo di impianti fisici, in cui si individuano attività quali:
 - Controllo dei singoli dispositivi di I/O dedicati a raccogliere i dati del processo fisico
 - Invio di comandi verso l'impianto
 - Le diverse attività devono frequentemente accedere a strutture dati comuni, che rappresentano lo stato complessivo del sistema da controllare

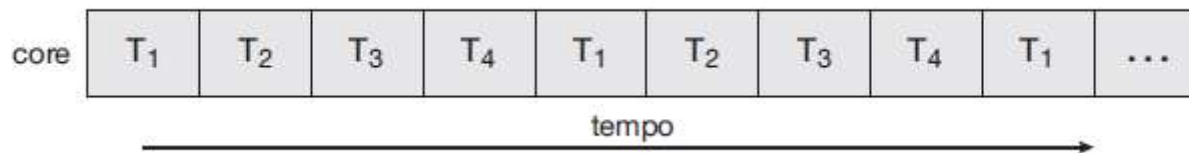
Multithreading

- Il multithreading si adatta perfettamente alla programmazione nei sistemi multicore

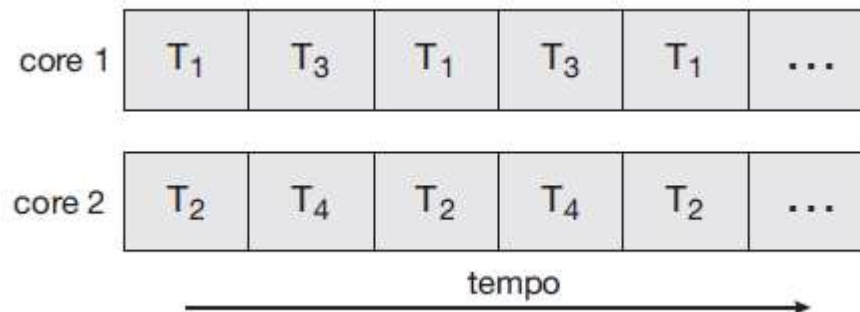


concorrenza VS parallelismo

Esecuzione concorrente su un sistema single core



Parallelismo nei sistemi multicore



Supporto del SO ai thread

- Supporto **user-level**, tramite librerie di funzioni (API) per gestire n thread in esecuzione
- Supporto **kernel-level**, tramite una tabella dei thread del sistema

Thread a livello utente

- Sono gestiti come uno strato separato sopra il nucleo del sistema operativo
 - Il kernel non ne è a conoscenza
- Sono realizzati tramite librerie di funzioni per la creazione, lo scheduling e la gestione dei thread, senza alcun intervento diretto del nucleo
 - **Vantaggio**: può essere implementato un pacchetto di thread anche su SO che non supportano i thread
 - **Svantaggio**: una chiamata di sistema bloccante da parte di un thread bloccherebbe tutti gli altri thread (il kernel blocca il processo)
- **POSIX Pthreads** è la libreria per la realizzazione di thread utente in sistemi UNIX-like
- **Windows threads** per sistemi Windows e **Java threads** per la JVM

Thread a livello kernel

- Sono gestiti direttamente dal SO: il nucleo si occupa di creazione, scheduling, sincronizzazione e cancellazione dei thread nel suo spazio di indirizzi
 - Quando un thread vuole creare o rimuovere un thread fa la relativa chiamata di sistema
 - Vantaggio: quando un thread si blocca, il kernel può eseguire altri thread (anche dello stesso processo)
- Supportati da tutti i sistemi operativi attuali
 - Windows
 - Solaris
 - Linux
 - Tru64 UNIX
 - Mac OS X

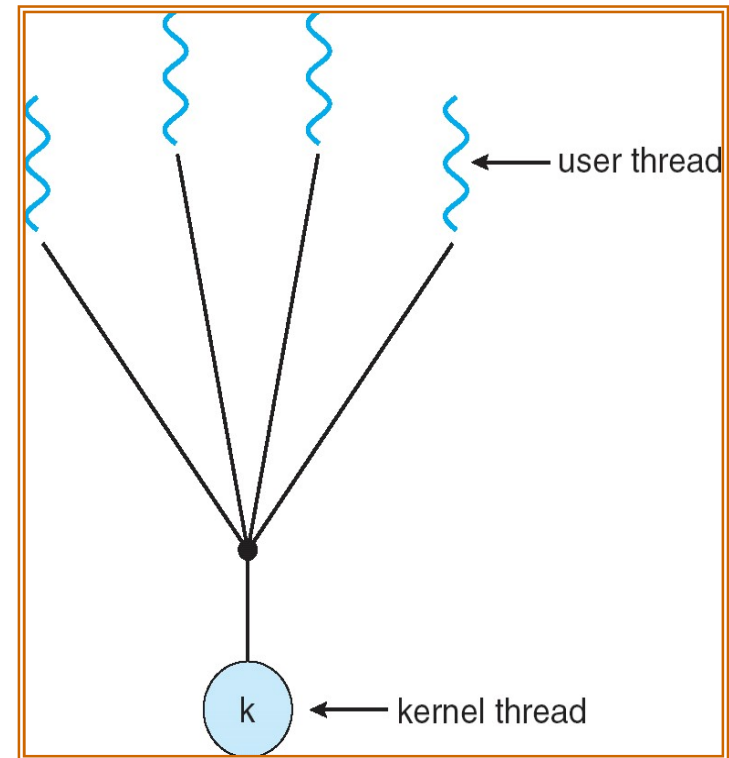
Modelli di programmazione multithread

Deve esistere una relazione tra i thread a livello utente ed i thread a livello kernel

- Modello multi-a-uno (M:1)
- Modello uno-a-uno (1:1)
- Modello multi-a-molti (M:M)

Modello multi-a-uno (M:1)

- Molti thread a livello utente vanno a corrispondere ad un unico thread a livello kernel
 - Il kernel “vede” una sola traccia di esecuzione
- In altre parole: i thread sono implementati a livello di applicazione, il loro scheduler non fa parte del SO, che continua ad avere solo la visibilità del processo



Modello multi-a-uno (M:1)

- Vantaggi

- Gestione efficiente dei thread nello spazio utente (scheduling poco oneroso)
- Non richiede un kernel multithread per poter essere implementato

- Svantaggi

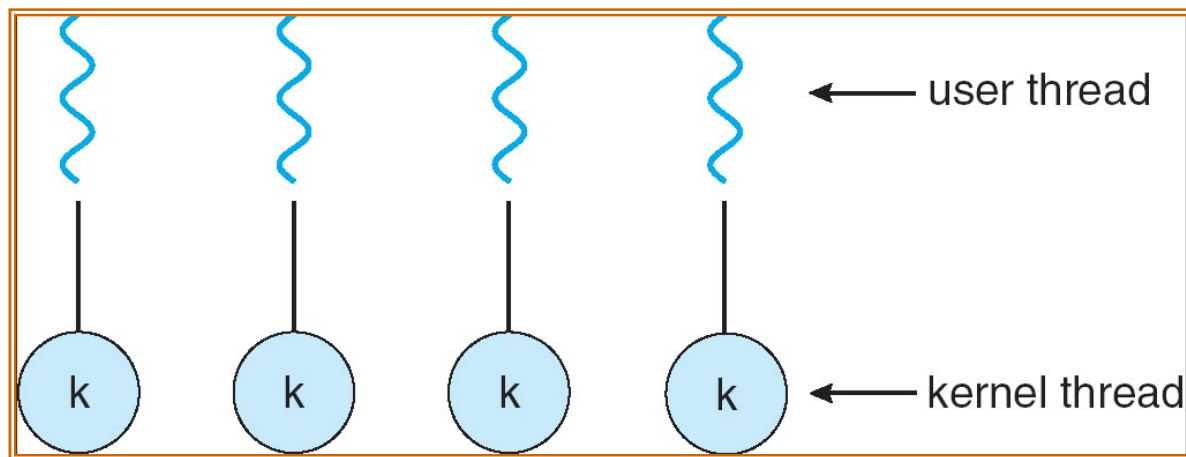
- L'intero processo rimane bloccato se un thread invoca una chiamata di sistema di tipo bloccante
- I thread sono legati allo stesso processo a livello kernel e non possono essere eseguiti su processori fisici distinti

- Attualmente, pochi SO implementano questo modello

- Solaris Green Threads
- GNU Portable Threads

Modello uno-a-uno (1:1)

- Ciascun thread a livello utente corrisponde ad un thread a livello kernel
 - Il kernel “vede” una traccia di esecuzione distinta per ogni thread
 - I thread vengono gestiti dallo scheduler del kernel (come se fossero processi)



Modello uno-a-uno (1:1)

- Vantaggi

- Scheduling molto efficiente
- Se un thread effettua una chiamata bloccante, gli altri thread possono proseguire nella loro esecuzione
- I thread possono essere eseguiti su processori fisici distinti

- Svantaggi

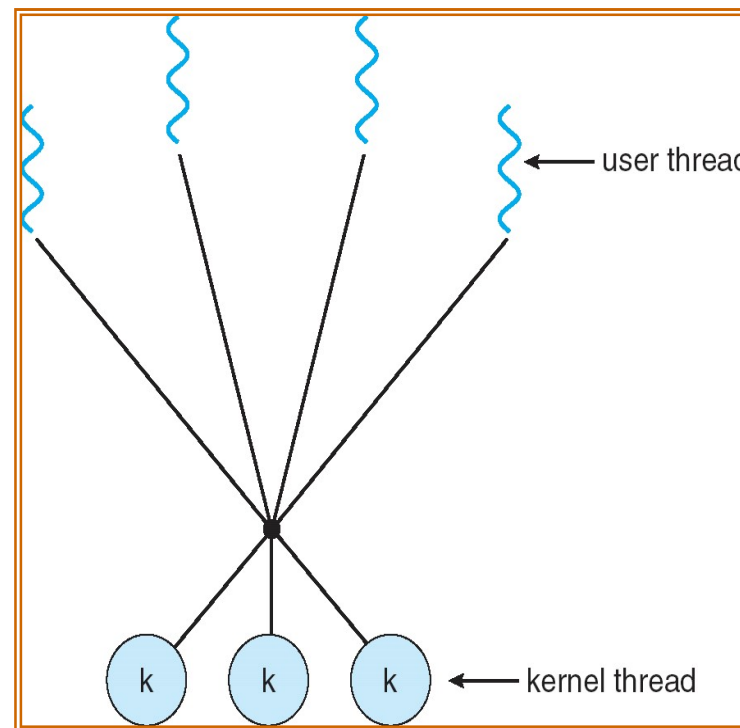
- Possibile inefficienza per il carico di lavoro dovuto alla creazione di molti thread a livello kernel
- Richiede un kernel multithread per poter essere implementato

- Esempi

- Windows 95/98/NT/2000/XP e versioni attuali
- Linux, Solaris (versione 9 e successive)

Modello multi-a-molti (M:M)

- Si mettono in corrispondenza più thread a livello utente con un numero minore o uguale di thread a livello kernel
- In altre parole, il sistema dispone di un pool di thread -*worker* - ognuno dei quali viene assegnato di volta in volta ad un thread utente



Modello multi-a-molti (M:M)

- Vantaggi

- Possibilità di creare tanti thread a livello utente quanti sono necessari per la particolare applicazione (e sulla particolare architettura) ed i corrispondenti thread a livello kernel possono essere eseguiti in parallelo sulle architetture multiprocessore
- Se un thread invoca una chiamata di sistema bloccante, il kernel può fare eseguire un altro thread

- Svantaggi

- Difficoltà nel definire la dimensione del pool di worker e le modalità di cooperazione tra i due scheduler

- Esempi:

- Windows con il ThreadFiber package
- Solaris (versioni precedenti alla 9)