Lezione 7.1 [07/10/22]

Capitolo 4: Thread

- Introduzione
- Programmazione multicore
- Modelli di supporto al multithreading
- Librerie dei thread
- Threading implicito
- Problematiche di programmazione multithread
- Esempi di Sistemi Operativi

Thread

In alcune situazioni una singola applicazione deve poter gestire molti compiti simili tra loro

In altre, una singola applicazione può dover gestire più compiti diversi, a volte eseguibili concorrentemente

Una possibile soluzione è quella della creazione di più processi tradizionali

Nel modello a processi, l'attivazione di un processo o il cambio di contesto sono operazioni molto complesse che richiedono ingenti quantità di tempo per essere portate a termine

Tuttavia a volte l'attività richiesta ha una vita relativamente breve rispetto a questi

- Ad es. l'invio di una pagina html da parte di un server web: applicazione troppo leggera per motivare un nuovo processo

Possibile soluzione: thread

Un **thread** è talvolta chiamato **processo leggero (lightweight process)**

Condivide con gli altri thread che appartengono allo stesso processo la **sezione del codice**, la **sezione dei dati** e altre risorse di sistema, come i **file aperti** e i **segnali**

Essi sono parte di un processo (un processo è un entità unica)

Ciascun thread rappresenta una diversa linea di esecuzione del processo

Processi tradizionali = singolo thread

Processi multithread = più thread

Molti kernel sono ormai multithread

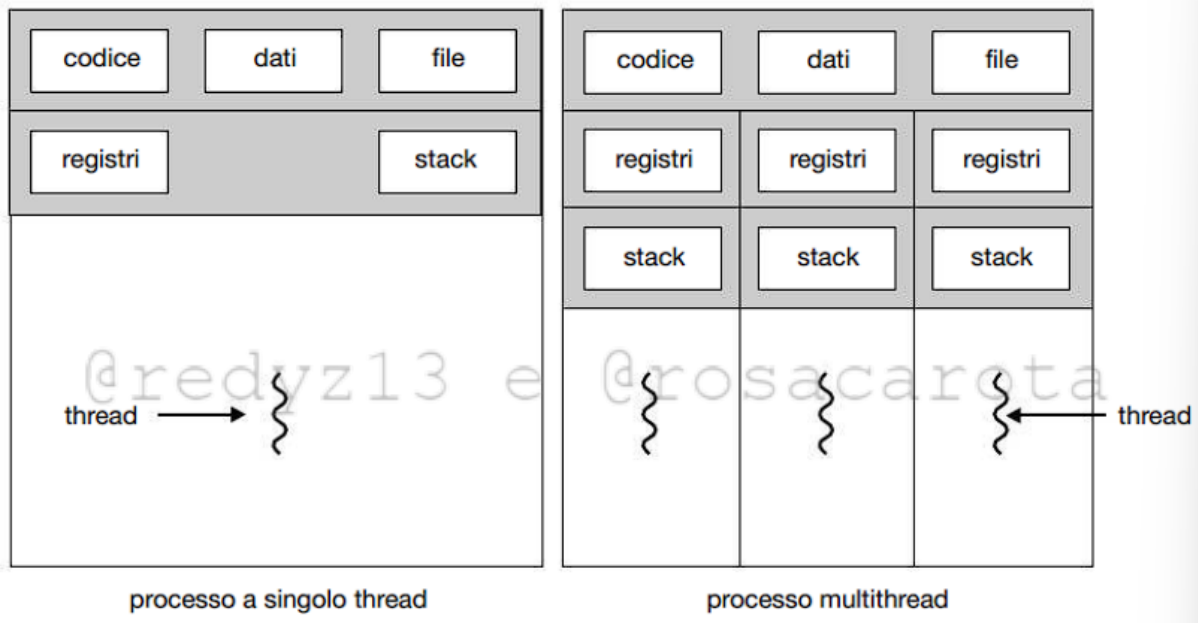
- Con i singoli thread dedicati a servizi specifici come la gestione dei dispositivi periferici o delle interruzioni

Molti programmi per i moderni PC sono predisposti per essere eseguiti da processi multithread

Un'applicazione, solitamente, è codificata come un processo a sé stante comprendente più thread di controllo

Il multithreading è la capacità di un S.O. di supportare thread di esecuzione multipli per ogni processo

Processi a singolo thread e multithread:



Il modello a thread

Le idee di base dietro il modello a thread sono:

- Permettere la definizione di attività "leggere" (lightweight processes - LWP) con costo di attivazione e terminazione limitato
- Possibilità di condividere lo stesso spazio di indirzzamento

Ogni processo racchiude più flussi di controllo (thread) che condividono le aree di testo e dei dati

Un thread è l'unità di base d'uso della CPU e comprende:

- Identificatore di thread
- Program counter
- Insieme di registri

- Stack

Condivide con altri thread che appartengono allo stesso processo:

- Sezione codice
- Sezione dati
- Risorse di sistema
    - Ad es. file aperti e segnali

Un processo tradizionale, chiamato anche **processo pesante (heavyweight process)**, è composto da un solo thread

Un processo multithread è in grado di svolgere più compiti in modo concorrente

Modello dei processi a thread singolo

In un modello a thread singolo la rappresentazione di un processo contiene il suo PCB e il suo spazio di indirizzamento utente, lo stack utente e lo stack del kernel

Modello multithread dei processi

In un ambiente multithreading ogni processo ha associato:

- Un solo PCB
- Un solo spazio di indirizzamento utente

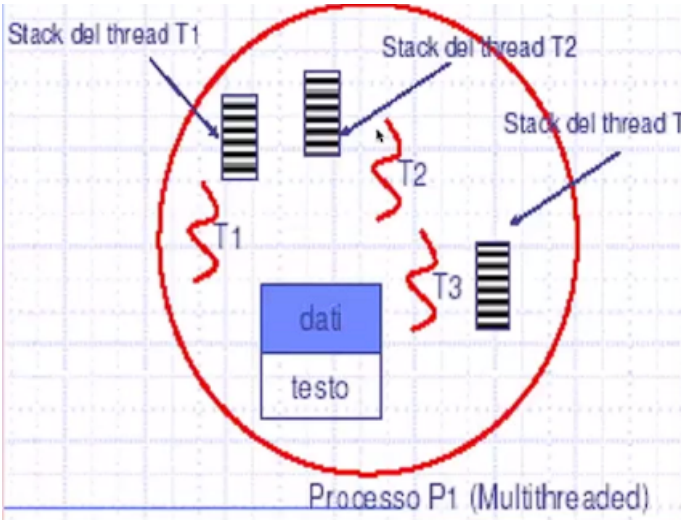
Ma ogni thread ha un proprio:

- Stack
- Un blocco di controllo privato contenente l'immagine dei registri
- Una priorità
- Altre informazioni relative al thread

Tutti i thread componenti un processo:

- Condividono lo stato e le risorse del processo
- Risiedono nello spazio di indirizzamento
- Hanno accesso agli stessi dati

Esempio:



T1 legge i caratteri da tastiera e li visualizza

T2 formatta il testo

T3 memorizza periodicamente sul disco

Vantaggi

La programmazione multithread offre numerosi vantaggi classificabili rispetto a 4 fattori principali

- **Tempo di risposta:**
  - Rendere multithread un'applicazione interattiva può permettere a un programma di continuare la sua esecuzione, anche se una parte di esso è bloccata o sta eseguendo un'operazione particolarmente lunga, riducendo il tempo di risposta all'utente. Per esempio, si consideri quello che succede quando un utente fa clic su un pulsante che provoca l'esecuzione di un'operazione che richieda diverso tempo. Un'applicazione a thread singolo resterebbe bloccata fino al completamento dell'operazione. Al contrario, se l'operazione viene eseguita in un thread separato, l'applicazione rimane attiva per l'utente
- **Condivisione delle risorse:**
  - I processi possono condividere risorse soltanto attraverso tecniche come la memoria condivisa e lo scambio di messaggi. Queste tecniche devono essere esplicitamente messe in atto dal programmatore. Tuttavia, i thread condividono per default la memoria e le risorse del processo al quale appartengono. Il vantaggio della condivisione del codice e dei dati consiste nel fatto che un'applicazione può avere molti thread di attività diverse, tutti nello stesso spazio d'indirizzi
- **Economia:**
  - Assegnare memoria e risorse per la creazione di nuovi processi è costoso; poiché i thread condividono le risorse del processo cui

appartengono, è molto più conveniente creare thread e gestirne i cambi di contesto. È difficile misurare empiricamente la differenza nell'overhead richiesto per creare e gestire un processo invece che un thread, tuttavia la creazione e la gestione dei processi richiedono in generale molto più tempo. In Solaris, per esempio, la creazione di un processo richiede un tempo circa trenta volte maggiore di quello necessario per la creazione di un thread, un cambio di contesto per un processo richiede un tempo pari a circa cinque volte quello necessario per un thread

- **Scalabilità:**
  - I vantaggi della programmazione multithread sono ancora maggiori nelle architetture multiprocessore, dove i thread si possono eseguire in parallelo su distinti core di elaborazione. Invece un processo con un singolo thread può funzionare solo su un processore, indipendentemente da quanti ve ne siano a disposizione. Il multithreading su una macchina con più processori incremeneta il parallelismo

Svantaggi

Il modello a thread presenta ovviamente anche alcuni svantaggi

Maggiore complessità di progettazione e programmazione:

- I processi devono essere "pensati" paralleli
- Difficile sincronizzazione tra i thread

Il modello è inadatto per situazioni in cui i dati devono essere protetti

La condivisione delle risorse accentua il pericolo di interferenze