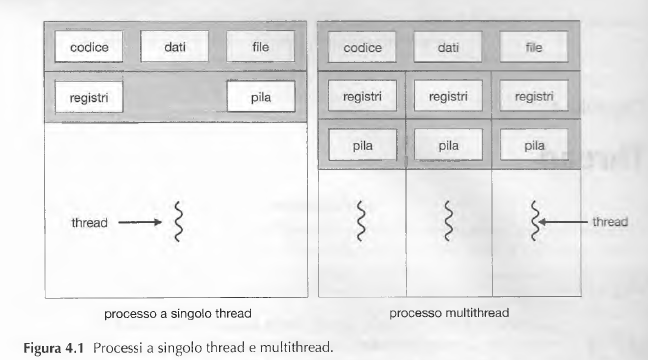
CAPITOLO 4: I THREAD

## 1.INTRODUZIONE

Un thread è l’unita di base d’uso della CPU e comprende un identificatore di thread (ID), un contatore di programma, un insieme di registri e una pila (Stack). Condivide con gli altri thread che appartengono allo stesso **processo la sezione del codice, la sezione dei dati e altre risorse del sistema.** Un processo tradizionale, chiamato anche processo pesante, è composto da un solo thread. Un processo multithread (processo leggero) è composto da più thread. Molti programmi per i moderni pc sono predisposti per essere eseguiti da processi multithread. Ad esempio, una pagina web può assegnare un thread per la richiesta dei dati, uno per la visualizzazione a video, ecc.



I thread hanno anche il ruolo primario nei sistemi che impiegano le RPC (remote procedure call), si tratta di un sistema che permette la comunicazione tra processi, fornendo un meccanismo di comunicazione simile alle normali chiamate di sistema. Molti kernel di sistemi operativi sono multithread.

**Multithreading** è la capacità di un sistema di supportare thread di esecuzione multiple per ogni processo, in questo ambiente ogni processo ha assegnato:

* Un solo PCB.
* Un solo spazio di indirizzamento utente

Ma ogni **thread** ha:

* Stack.
* Un blocco di controllo privato contentente l’immagine dei registri.
* Una priorità.
* Altre info relative ai thread.

## Vantaggi

I vantaggi della programmazione multithread si possono classificare secondo 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 una operazione particolarmente lunga.
* **Condivisione delle risorse:** i processi normalmente condividono risorse attraverso sistemi a memoria condivisa o a scambio di messaggi; i thread condividono di serie la memoria e le risorse del processo cui appartengono. Un’applicazione può avere molti thread di attività diverse, tutti nello stesso spazio degli indirizzi.
* **Economia:** assegnare memoria e risorse per la creazione di nuovi processi è costoso: poiché i thread condividono le risorse del processo cuoi appartengono, è molto più vantaggioso creare thread e gestirne i cambi di contesto
* **Uso di sistemi multiprocessore:** i vantaggi della programmazione multithread aumentano notevolmente nelle architetture multiprocessore dove i thread si possono eseguire in parallelo

## Svantaggi:

* **Maggiore complessità di progettazione e programmazione**, i program thread devono essere pensati e programmati in parallelo ed è difficile sincronizzarli.
* **Sicurezza:** Modello inadatto per situazioni in cui i dati devono essere protetti in quanto i thread condividono risorse, il che accentua il pericolo di interferenza.
* **//fino a dove chiede il prof carpentieri.**

## Una tendenza recente nel progetto dell'architettura dei sistemi consiste nel montare diverse unità di calcolo (core) su un unico processore(processore multi-core); ogni unità appare al sistema operativo come un processore separato.

es) si consideri un'applicazione con 4 thread : in un sistema con una singola unità di calcolo "esecuzione concorrente" significa solo che l'esecuzione dei thread è stratificata nel tempo(la CPU può eseguire al massimo un processo alla volta).

Su un sistema multi-core invece "esecuzione concorrente" vuol dire che i thread possono essere eseguiti in parallelo(il sistema può assegnare thread diversi a diverse unità di calcolo).

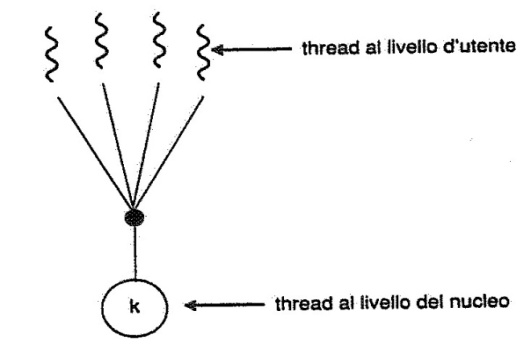
La tendenza verso i sistemi multi-core ha messo sotto-pressione sia i progettisti di SO , sia i programmatori di applicazioni , affinchè entrambi utilizzino al meglio unità di calcolo multiple.

## 2.MODELLI DI PROGRAMMAZIONE MULTITHREAD

I thread possono essere distinti in thread a livello utente e thread a livello kernel: i primi sono gestiti senza l’aiuto del kernel, i secondi sono gestiti direttamene dal SO.

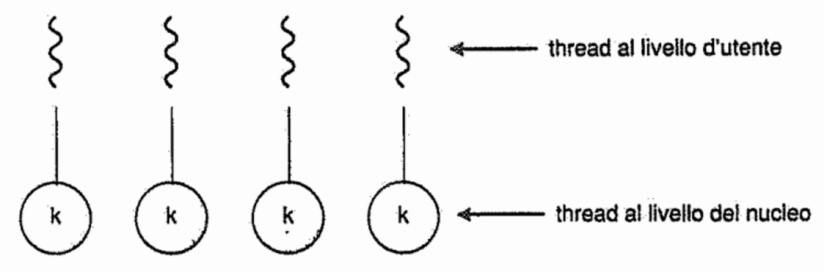
## 

## Modelli da molti a uno



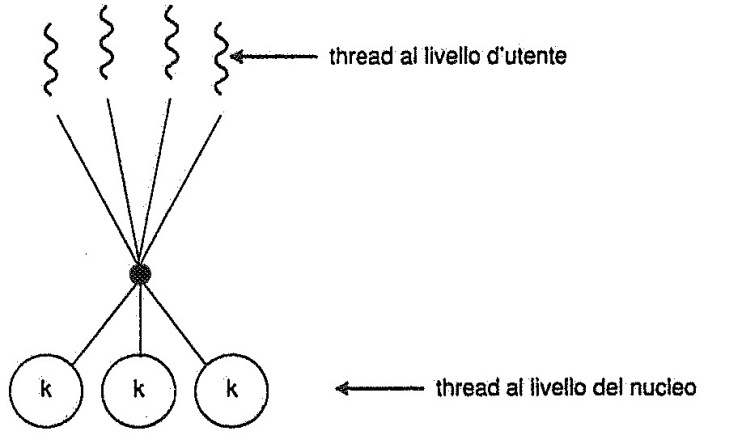
Il modello da molti a uno fa corrispondere molti thread a livello utente a un singolo thread a livello kernel. Poiché si svolge nello spazio utente, la gestione dei thread risulta efficiente, ma l’intero processo rimane bloccato se un thread invoca una chiamata di sistema di tipo bloccante. Inoltre poiché un solo thread alla volta puoi accedere al kernel, è impossibile eseguire thread multipli in parallelo in sistemi multiprocessore.

## Modello da uno a uno



Il modello da uno a uno mette in corrispondenza ciascun thread a livello utente con un thread a livello kernel. Questo modello offre un grado di concorrenza maggiore rispetto al precedente, poiché anche se un thread invoca una chiamata di sistema bloccante, è possibile eseguire un altro thread. L’unico svantaggio di questo modello è che la creazione di ogni thread a livello utente comporta la creazione del corrispondente thread a livello kernel.

## Modello da molti a molti



Il modello da molti a molti mette in corrispondenza di più thread a livello utente con un numero minore o uguale di thread a livello kernel; quest’ultimo può essere specifico per un certa applicazione o per un particolare calcolatore. Nonostante il modello da molti a uno permette di creare tanti thread a livello utente quanti ne desiderino, non viene garantita una concorrenza reale , in quanto il meccanismo di scheduling del kernel può scegliere un solo thread alla volta. Il modello da uno a uno permette una maggiore concorrenza ma i programmatori devono stare attenti a non creare molti thread all'interno di una applicazione. Il modello da molti a molti non presenta nessuno di questi difetti. I programmatori possono creare liberamente i thread che ritengono necessari e i corrispondenti thread a livello kernel si possono eseguire in parallelo nelle architetture multiprocessore. Inoltre se un thread impiega una chiamata di sistema bloccante, il kernel può fare in modo che si esegua un altro thread.

-Una diffusa variante del modello molti a molti mantiene la corrispondenza tra più thread con un numero minore o uguale di thread del kernel , ma permette anche di vincolare un thread utente a un singolo thread a livello kernel (MODELLO A DUE LIVELLI).