I thread

- Generalità.
- Modelli multithread.
- Problematiche relative ai thread.
- Pthread.

I thread

- Il **Thread** è un flusso di controllo relativo ad un dato processo.

Molti sistemi operativi moderni consentono ad un processo di contenere più flussi di controllo.

- *Generazione di thread* **vs** *generazione di processi figli*.
  - La creazione di processi è un meccanismo dispendioso in termini di tempo e risorse necessarie.
  - Quando le funzionalità da eseguire sono sostanzialmente analoghe ma semplicemente multiplate non ha senso ridondare l'intero apparato a corredo di un processo.
    - È il caso ad esempio di un demone HTTP che deve servire migliaia di client per restituire risorse complesse (immagini, audio, video).
  - Generalmente è più opportuno replicare semplicemente i flussi di controllo di un medesimo processo.
    - È il caso dei server RPC ed RMI che sono tipicamente multithread.

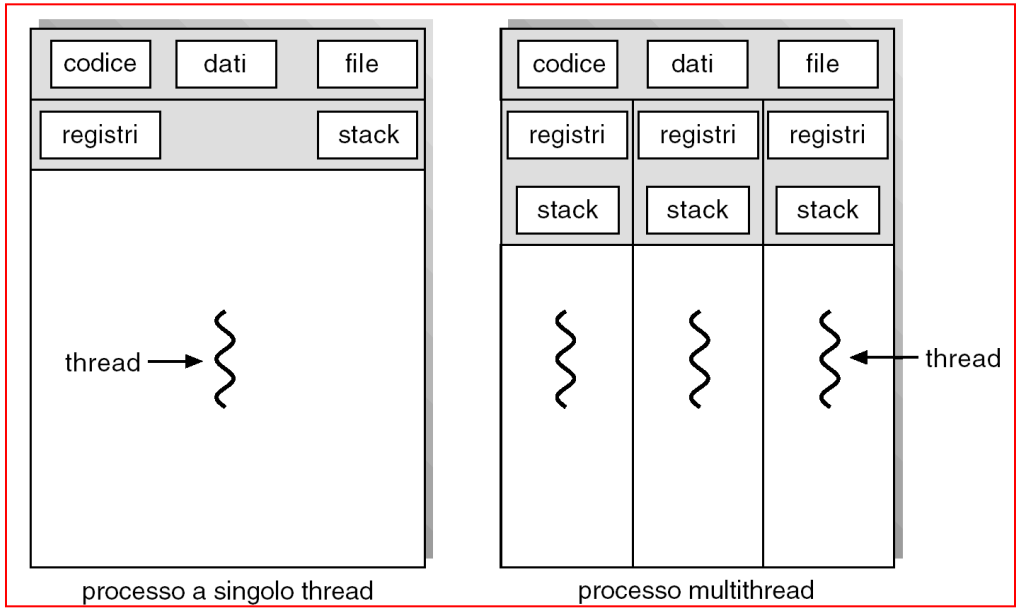
Single e multithreading

- **Thread** = unità atomica di utilizzo della CPU.

Comprende:

- Identificatore del thread;
- Program counter;
- Register set;
- Stack.

- Due o più thread relativi al medesimo processo condividono:
  - Sezione di codice e di dati;
  - Risorse (file aperti).



Vantaggi

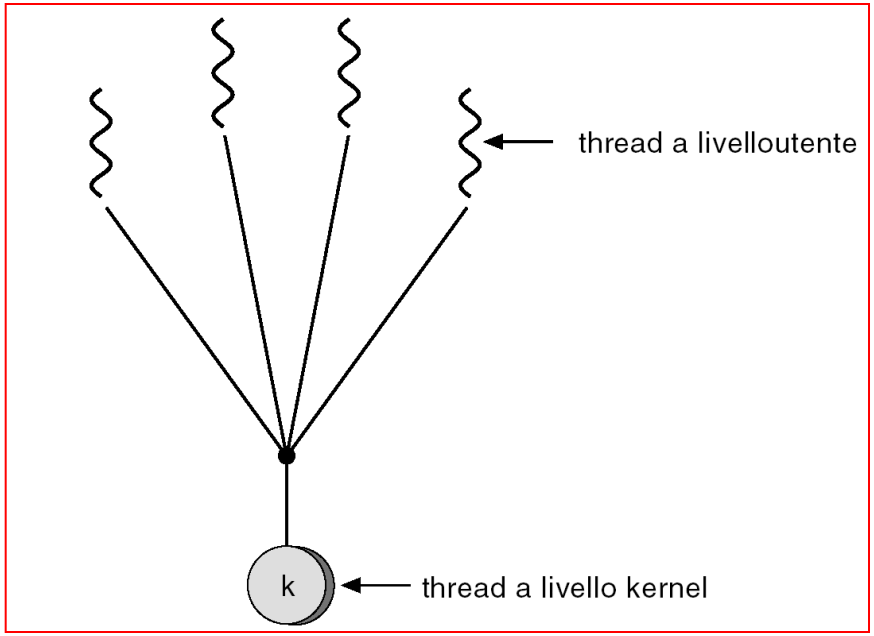
- Prontezza di risposta.
  - Migliore interazione con l'utente nel caso di applicazioni interattive.
- Condivisione di risorse.
  - I thread condividono memoria e risorse del processo al quale appartengono e questo rende possibile avere un elevato numero di thread all'interno dello stesso spazio di indirizzi.
- Economia.
  - È più economico cambiare il contesto di thread piuttosto che fra processi dato che essi condividono memoria e risorse:
    - in Solaris
      - $T_{\text{creazione processo}} \approx 30\ T_{\text{creazione thread}}$
      - $T_{\text{context switching processo}} \approx 5\ T_{\text{context switching thread}}$
- Utilizzo di architetture multiprocessore.
  - I benefici del multithreading aumentano fortemente su macchine multi-CPU (i thread vengono eseguiti con un parallelismo reale).

Thread user/kernel space

- La gestione dei thread è effettuata a **livello utente** attraverso una opportuna libreria.
  - Essa fornisce al programmatore le API per la creazione, scheduling e gestione dei thread.
- Le librerie di thread a livello utente possono essere implementate secondo due approcci:
  - Libreria completamente residente all'interno dello *user space* senza supporto da parte del kernel.
    - Invocare una funzione di libreria si traduce in una chiamata di funzione locale.
  - *Libreria a livello kernel.*
    - Invocare una funzione delle API della libreria comporta una chiamata di sistema.
- Tre principali librerie di thread:
  - POSIX Pthread (user/kernel thread).
  - Java thread (user/kernel thread).
  - Win32 thread (kernel thread).

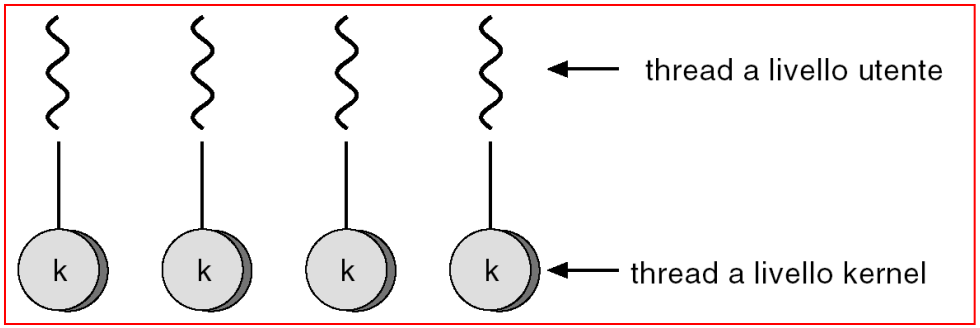
Programmazione Multi-Thread
Modello molti-a-uno

- Il modello molti-a-uno riunisce molti thread di livello utente in un unico kernel thread associato al processo in esecuzione
- La gestione è effettuata a livello utente e quindi è efficiente.
- L'intero processo si blocca se un thread esegue una chiamata di sistema bloccante.
- Non è possibile eseguire thread multipli in parallelo su più processori in quanto un solo thread per volta può accedere al kernel.



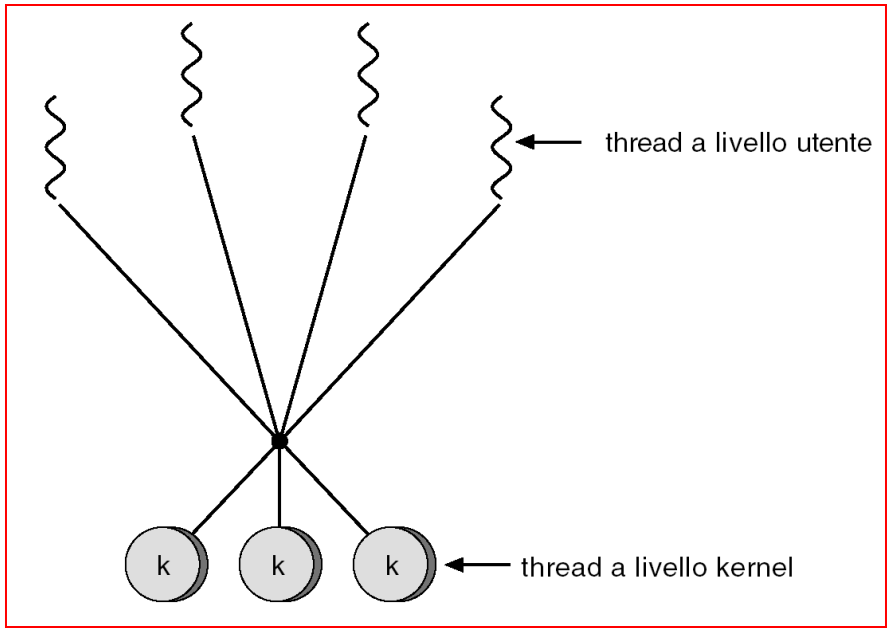
Modello uno-a-uno

- Il modello uno-a-uno mappa ciascun thread utente in un kernel thread.
- Aumenta il livello di concorrenza possibile:
  - Un thread può essere in esecuzione mentre un altro esegue una chiamata di sistema bloccante.
  - Più thread possono essere eseguiti in parallelo su più processori.
- Obbliga alla mappatura di uno user thread su un kernel thread.
  - L'overhead della creazione di un kernel thread si ripercuote sull'applicazione.
    - In genere è posto un limite al multithreading(per non appensatire lo scheduler)
- Esempi:
  - Windows NT/XP/2000
  - Linux
  - Solaris 9.



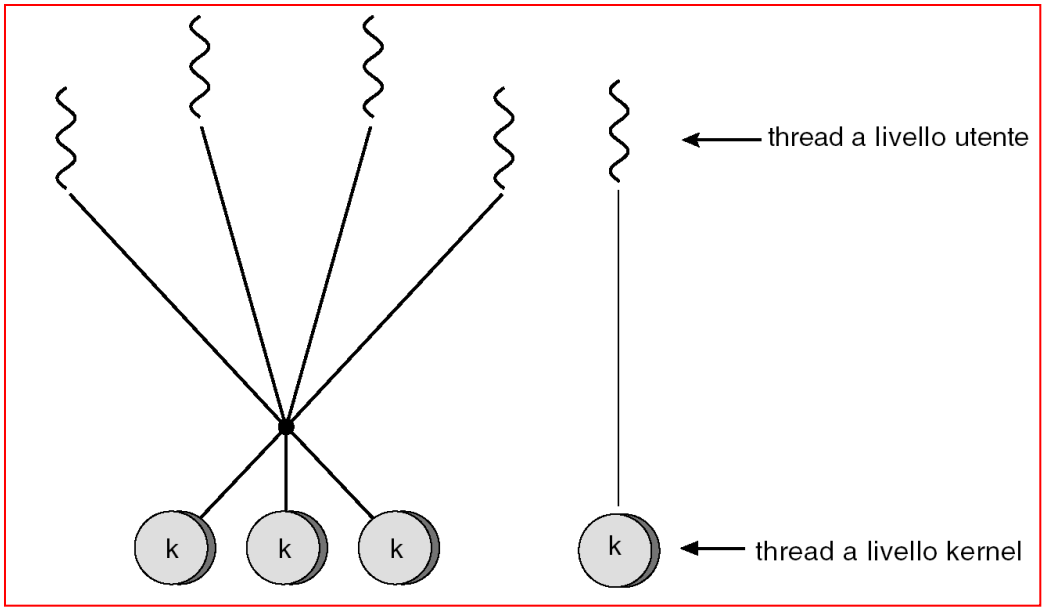
Modello molti-a-molti

- Permette di aggregare molti thread (n) a livello utente verso un numero più piccolo o equivalente di kernel thread (m).
- Permette al sistema operativo di creare un numero sufficiente di kernel thread in base alle caratteristiche dell'architettura.
- Non soffre di alcuna delle limitazioni dei precedenti modelli.
- Esempio: Windows NT/2000 con il pacchetto *ThreadFiber*.



Modello a due livelli

- Variante del modello molti-a-molti che permette anche di associare un thread di livello utente ad un kernel thread.
- Esempi:
  - True64 UNIX
  - Solaris 9
  - IRIX



fork() ed exec()

- Le chiamate di sistema **fork()** ed **exec()** hanno, nel caso dei thread, una semantica differente rispetto a quella dei processi.
- La chiamata exec() ha un funzionamento pressoché analogo:
  - si procede a rimpiazzare l'intero processo inclusi tutti i suoi thread.
- Se un thread in un programma chiama una fork() si potrebbe a seconda della specifica applicazione:
  - duplicare tutti i thread
  - duplicare solo il thread che invoca la fork().
- Quale delle due versioni di fork() è da usare?
  - Se la chiamata è seguita da una exec(), non è necessario duplicare l'intero thread set dal momento che il programma specificato come parametro della exec() rimpiazzerà l'intero processo.
  - In caso di fork() singola è opportuno che si duplichi l'intero thread set.

Cancellazione di thread

- È l'atto di terminare un thread prima che abbia completato la propria esecuzione.
- La cancellazione di un **thread target** può avvenire in due differenti scenari:
  - **Cancellazione asincrona:** un thread termina immediatamente il thread target.
  - **Cancellazione differita:** il thread target può periodicamente effettuare un controllo di terminazione (punto di check) avendo così l'opportunità di terminare in modo ordinario.
- La cancellazione asincrona comporta problemi quando:
  - in fase di cancellazione un thread sta modificando dati condivisi con altri thread;
  - le risorse assegnate ad un thread cancellato non vengano integralmente restituite.
    - In tal caso il sistema operativo potrebbe reclamarle senza esito.

Gestione dei segnali (1/2)

- Nei sistemi UNIX per notificare ad un processo che si è verificato un particolare evento si usa un **segnale**.
- Un segnale può essere ricevuto in modo sincrono o asincrono a seconda della sua origine e dell'evento che lo ha generato.
- La **gestione dei segnali** può essere affidata al:
  1. Gestore di default dello specifco segnale (eseguito dal kernel);
  2. Gestore definito dall'utente.
- Trattare un **segnale** può significare:
  1. ignorarlo;
  2. condurre ad un intervento stabilito dal signal manager;
  3. condurre alla terminazione del programma.

Gestione dei segnali (2/2)

- La gestione di segnali è più complessa nel caso di multithreading.
- Riguardo al **signal delivery** esistono differenti possibilità:
  - Consegnare il segnale al thread cui il segnale viene applicato.
  - Consegnare il segnale ad ogni thread del processo.
  - Consegnare il segnale al alcuni thread del processo.
  - Designare un thread specifico che riceva tutti i segnali per conto del processo.
- Il metodo adoperato in genere dipende dal tipo di segnale.
  - I segnali sincroni devono essere consegnati al thread che ha causato l'evento.
  - Nel caso di segnali asincroni la situazione è più complessa:
    - Alcuni segnali asincroni dovrebbero essere inviati a tutti i thread (come p.es nel caso di terminazione di un processo).
    - In taluni casi UNIX permette di lasciare al thread la decisione sui segnali da accettare e su quelli da bloccare: un segnale asincrono verrà consegnato solo al primo thread che non lo blocca.
- In Windows i segnali vengono emulati attraverso le Asyncronous Procedure Call (APC).

Gruppi di thread

- Nei server multithread:
  - si potrebbero esaurire le risorse di sistema (specialmente tempo di CPU e memoria) se non viene posto limite alla creazione di thread concorrenti;
  - prima che la richiesta possa essere servita occorre attendere un tempo imprecisato in attesa della creazione di un nuovo thread.
- Come soluzione si può pensare di creare numerosi thread all'avvio del processo e metterli dentro ad un pool dove restano in attesa di lavoro.
- Quando un server multithread riceve una richiesta, si limiterà a risvegliare uno dei thread disponibili all'interno del pool.
- Espletata la richiesta il thread rientra nel pool in sleeping.
- Se il pool non ha thread disponibili, il server resterà in attesa.
- Vantaggi:
  - Servire la richiesta all'interno di un thread esistente è tipicamente più veloce che attendere la creazione di un thread.
  - Un pool di thread limita il numero di thread esistenti in contemporanea.
- Le dimensioni dei pool di thread possono essere fissate in modo euristico oppure dinamicamente in base ai pattern di carico(n di richieste)

LWP-LightWeight Process

- Il modello molti-a-molti ed il modello a due livelli richiedono una comunicazione fra kernel e libreria di thread per mantenere un appropriato numero di kernel thread allocati all'applicazione.
- Si adopera una struttura intermedia tra user thread e kernel thread denominata LightWeight Process (LWP).
  - Un LWP appare alla libreria dei thread utente come un **processore virtuale** sul quale schedulare l'esecuzione.
    - Una applicazione CPU-bound su un sistema monoprocessore implica che un solo thread per volta possa essere eseguito, quindi per essa sarà sufficiente un unico LWP per thread.
    - Una applicazione I/O-bound tipicamente richiede un LWP per ciascuna chiamata di sistema bloccante.

Attivazione dello schedulatore

- Il kernel mette a disposizione un set di LWP (processori virtuali) per ciascuna applicazione . Essa può schedulare su ciascuno di essi un user thread.
- L'**attivazione dello schedulatore** fornisce **upcall** (chiamate al thread) – un meccanismo di comunicazione dal kernel alla libreria dei thread.
- La libreria di thread esegue un **upcall manager** per processore virtuale. Esso provvede alla gestione degli eventi diretti al thread.
- Quando un kernel thread sta per bloccarsi viene scatenato un evento che attiva una upcall.
  - Il kernel informa l'applicazione che uno specifico thread sta per bloccarsi e alloca un nuovo LWP per l'applicazione. Questa esegue un upcall manager su di esso.
  - L'upcall manager del thread in fase di blocco ne salva lo stato e rilascia il processore virtuale. Poi provvede a schedulare un nuovo thread eleggibile per il LWP appena rilasciato.
- Al verificarsi dell'evento per cui il thread era bloccato, il kernel fa una upcall alla libreria dei thread segnalando l'eleggibilità per l'esecuzione del thread fermo.
  - Il gestore di upcall per questo evento richiede un processore virtuale e vi schedula l'esecuzione del thread sbloccato.
- L'attivazione dello schedulatore permette ad una applicazione di mantenere il corretto numero di kernel thread.

Pthread

- Standard POSIX (IEEE 1003.1c).
- API per la creazione e la sincronizzazione dei thread.
- Fornisce una specifica per il comportamento della libreria dei thread.
- I progettisti di sistemi operativi possono implementare la specifica nel modo che desiderano.
- Frequente nei sistemi operativi di UNIX (Solaris, Linux, Mac OS X).
- Le API di Pthread possono essere considerate come librerie di thread a livello utente.
  - Non esiste alcuna relazione fra un thread creato usando le API di Pthread e un thread del kernel associato.