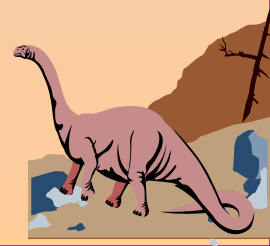




Capitolo 4: Thread

- Introduzione
- Modelli di programmazione multithread
- Librerie dei thread
- Questioni 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 vita relativamente breve rispetto a questi tempi.
 - ☞ Ad es. l'invio di una pagina html da parte di un server web: applicazione troppo leggera per motivare un nuovo processo.
- Possibile soluzione: threads.





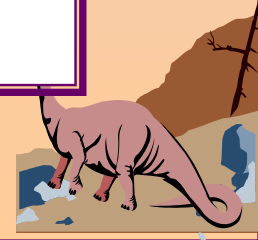
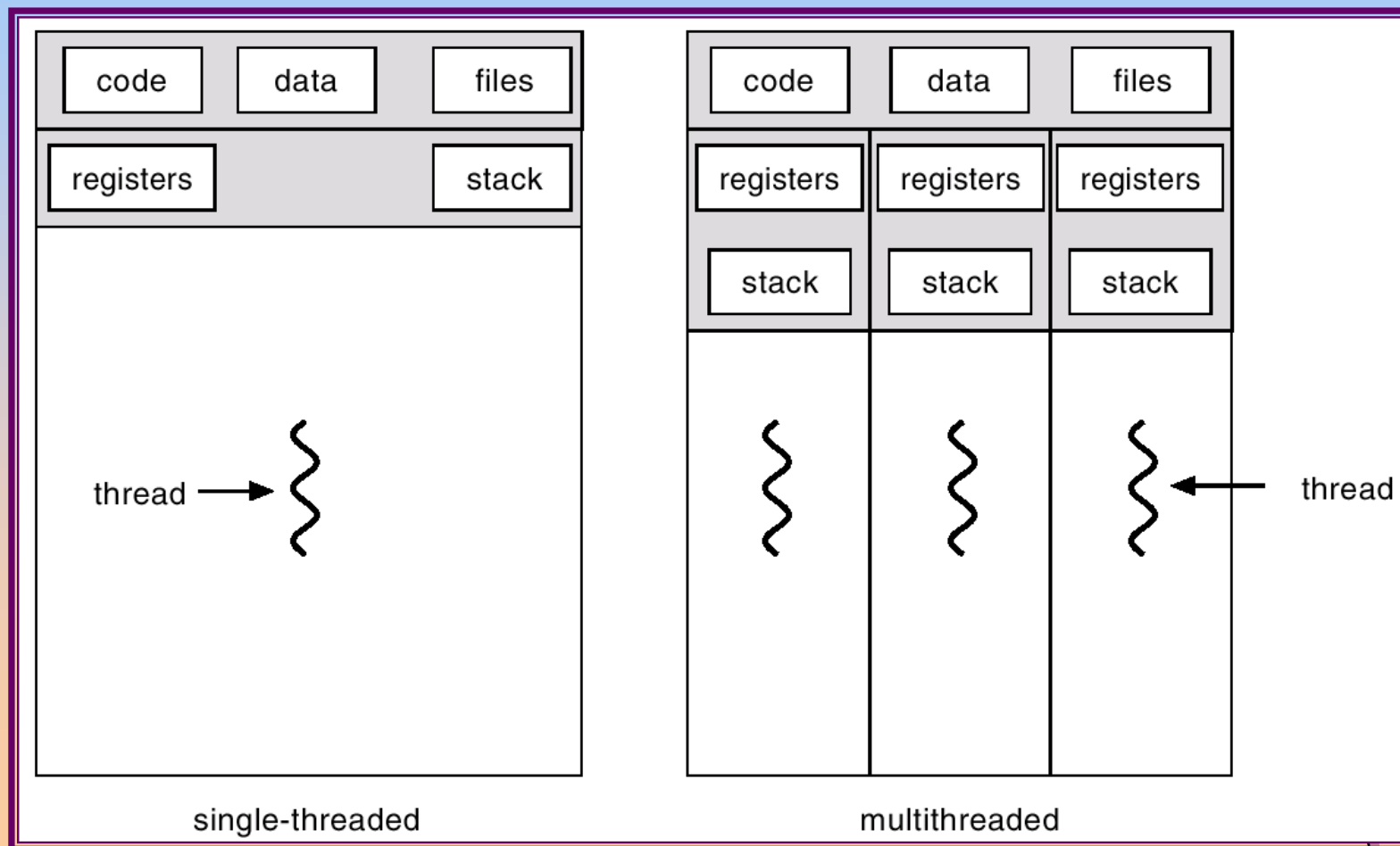
Thread (II)

- 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.
- 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 sè stante comprendente più thread di controllo.
- Il multithreading è la capacità di un sistema operativo di supportare thread di esecuzione multipli per ogni processo.





Processi a singolo thread e multithread





Il modello a thread

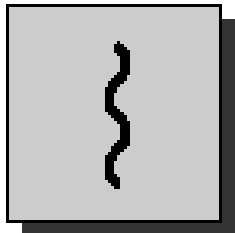
- Le idee di base dietro il modello a thread sono:
 - ☞ Permettere la definizione di attività “leggere” (lighthweight processes--LWP) con costo di attivazione e terminazione limitato.
 - ☞ Possibilità di condividere lo stesso spazio di indirizzamento.
- Ogni processo racchiude più flussi di controllo (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 gli altri thread che appartengono allo stesso processo:
 - ☞ Sezione codice.
 - ☞ Sezione dati.
 - ☞ Risorse di sistema.
 - 📄 Ad es.: file aperti.



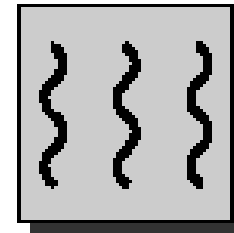
Thread e Processi: quattro possibili scenari

MS-DOS

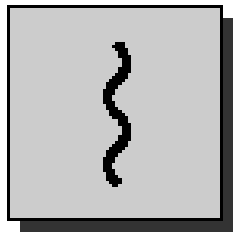
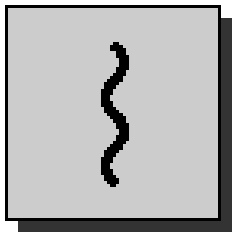
Motore
Runtim
e
java



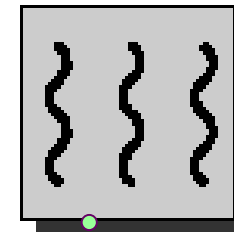
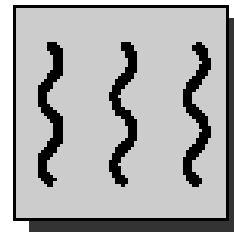
one process
one thread



one process
multiple threads



multiple processes
one thread per process



multiple processes
multiple threads per process

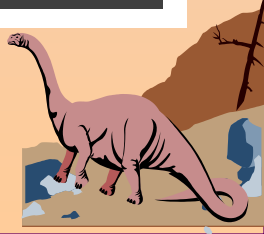
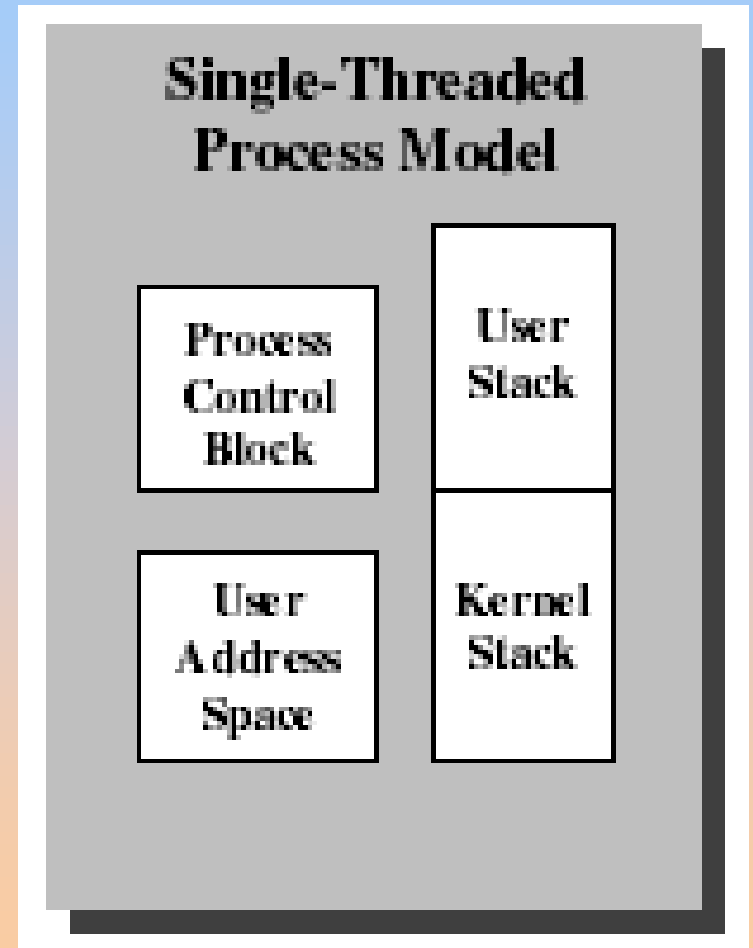
UNIX

WindowsN
T
Solaris
Mach



Modello dei processi a thread singolo

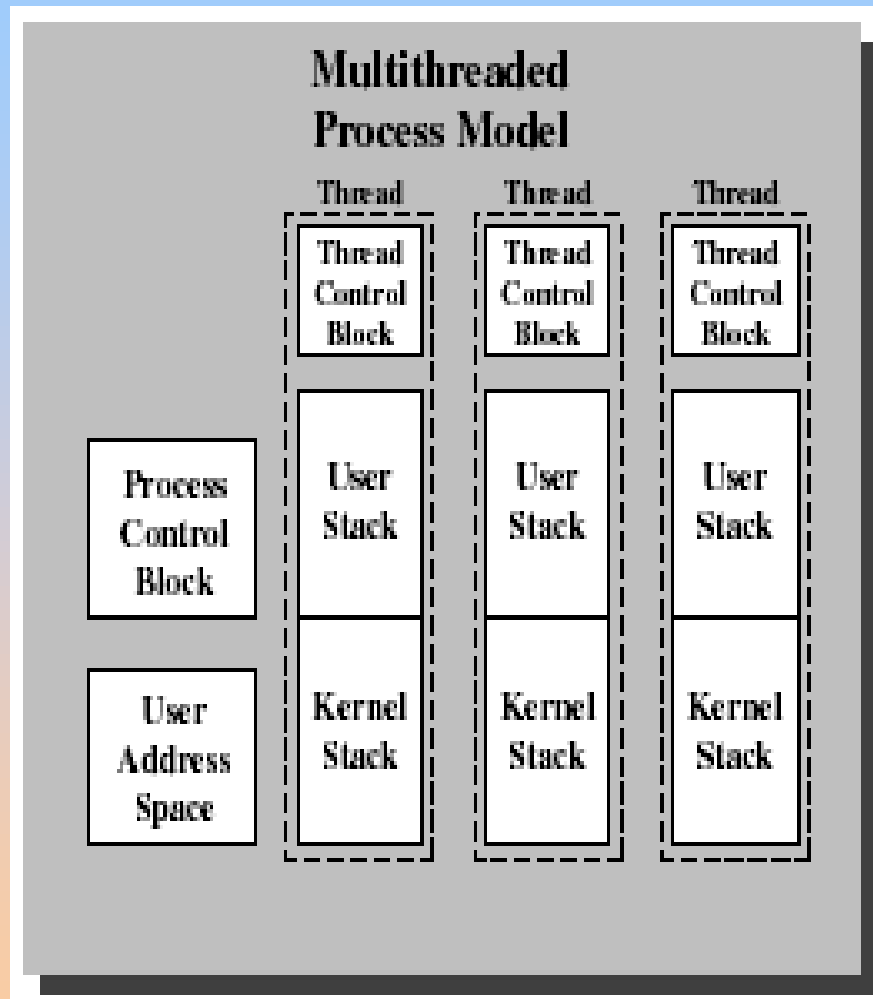
- In un modello a thread singolo la rappresentazione di un processo contiene il suo PCB e il suo spazio indirizzamento utente, lo stack utente e lo stack del kernel





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

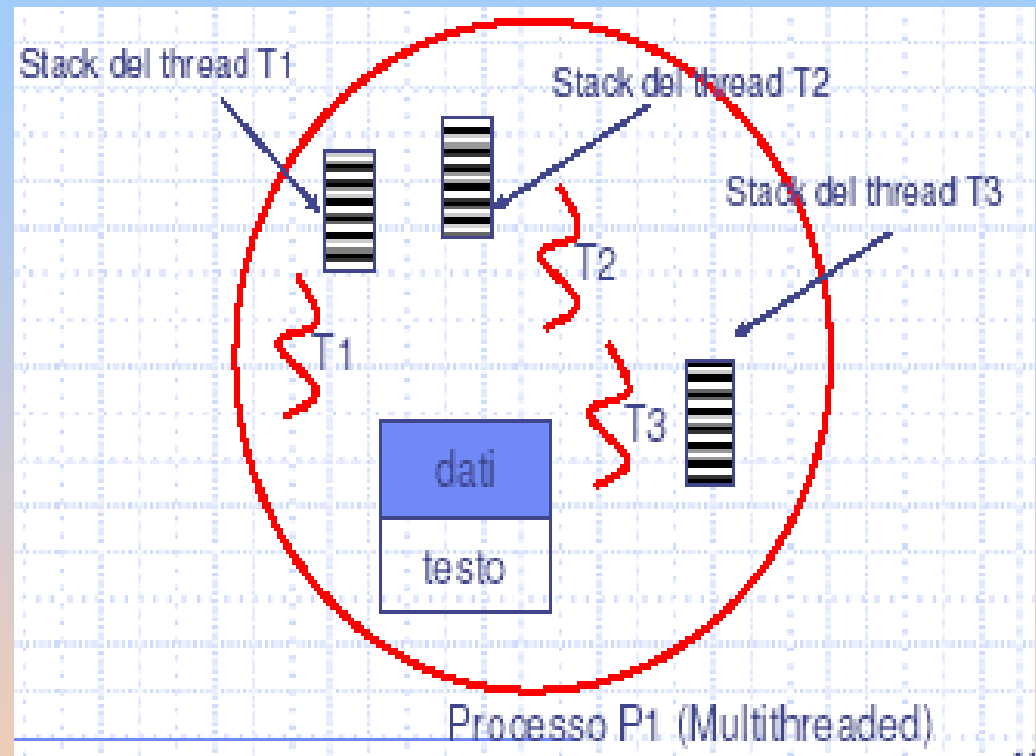




Esempio: struttura processo multithread

■ Tutti i thread componenti un processo:

- ☞ condividono lo stato e le risorse del processo
- ☞ risiedono nello spazio di indirizzamento
- ☞ hanno accesso agli stessi dati



- **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,*
- ☞ *condivisione delle risorse,*
- ☞ *economia,*
- ☞ *uso di più unità di elaborazione.*

- **Tempo di risposta:**

- ☞ l'esecuzione può continuare anche se parte del programma è bloccata o sta eseguendo un'operazione particolarmente lunga.
- ☞ Es.: Programma di consultazione web
 - 📄 Un thread carica un'immagine
 - 📄 Un thread permette l'interazione con l'utente





Vantaggi (II)

■ **Condivisione delle risorse:**

☞ I thread normalmente condividono la memoria e le risorse del processo cui appartengono.

■ Possiamo avere quindi più thread di attività diverse nello stesso spazio indirizzi.

■ Questa condivisione rende più conveniente creare thread e gestire cambiamenti di contesto piuttosto che creare nuovi processi

■ **Economia:**

☞ Assegnare memoria e risorse per la creazione di nuovi processi è costoso, così i cambi di contesti.

■ Con i thread queste operazioni sono alleggerite.

■ **Uso di più unità di elaborazione:**

☞ I thread di uno stesso processo possono essere eseguiti in parallelo.





Svantaggi

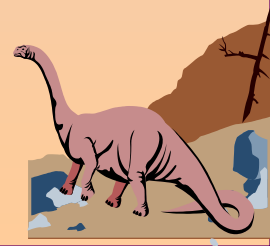
- Il modello a thread presenta ovviamente anche alcuni svantaggi.
- Maggiore complessità di progettazione e programmazione:
 - ☞ I processi devono essere “pensati” paralleli.
 - ☞ Difficile sincronizzazione tra i thread.
- Il modello è inadatto per situazioni in cui i dati devono essere protetti.
- La condivisione delle risorse accentua il pericolo di interferenze.





Thread a livello utente

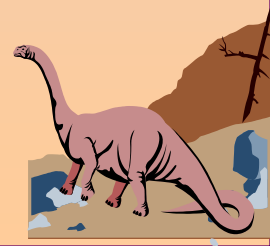
- Gestiti come uno strato separato al di sopra del nucleo del S.O..
- Realizzati attraverso una libreria di funzioni senza alcun intervento diretto del kernel.
- Tutto il lavoro di gestione dei thread viene effettuato dall'applicazione
- Creazione, scheduling, etc. avvengono nello spazio utente senza alcun intervento diretto del kernel
 - ☞ normalmente quindi sono operazioni veloci.
- Il kernel non è necessariamente conscio della presenza dei thread.
- Se il nucleo di S.O. è a singolo thread e da uno dei thread utenti viene richiesta un'operazione bloccante (ad es. I/O), allora tutto il processo utente deve essere bloccato.
- Esempi:
 - ☞ POSIX *Pthreads*
 - ☞ Mach *C-threads*
 - ☞ Solaris *threads*





Thread a livello del kernel

- Sono gestiti direttamente dal S.O.,
 - ☞ che si occupa della creazione, scheduling e gestione dello spazio di indirizzi.
- Non c'è codice di gestione dei thread ma un'API per la componente del kernel che gestisce i thread
- Sono più lenti da creare e gestire di quelli a livello utente.
- Tuttavia se un thread esegue un'operazione bloccante questa non blocca gli altri thread del processo.
- Se sono disponibili più CPU questi thread possono essere anche eseguiti in parallelo.
- Esempi:
 - ☞ Windows 95/98/NT/2000
 - ☞ Solaris
 - ☞ Tru64 UNIX
 - ☞ BeOS
 - ☞ Linux





Vantaggi/Svantaggi: Livello utente

■ Vantaggi:

- ☞ Lo switching non coinvolge il kernel è quindi non ci sono cambiamenti della modalità di esecuzione.
- ☞ Maggiore libertà nella scelta dell'algoritmo di scheduling che può anche essere personalizzato.
- ☞ Siccome le chiamate possono essere raccolte in una libreria, c'è una maggiore portabilità tra S.O..

■ Svantaggi:

- ☞ una chiamata al kernel può bloccare tutti i thread di un processo,
 - 📄 indipendentemente dal fatto che in realtà solo uno dei suoi thread ha causato la chiamata bloccante.
- ☞ In sistemi SMP, due processori non risulteranno mai associati a due thread del medesimo processo.





Vantaggi/Svantaggi: Livello Kernel

■ Vantaggi:

- ☞ il kernel può eseguire più thread dello stesso processo anche su più processori.
- ☞ il kernel stesso può essere scritto multithread.

■ Svantaggi:

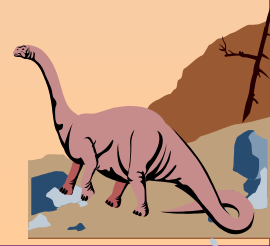
- ☞ lo switching coinvolge chiamate al kernel e questo, soprattutto in sistemi con molteplici modalità, comporta un costo.
- ☞ l'algoritmo di scheduling è meno personalizzabile e meno portabile.





Modelli di programmazione multithread

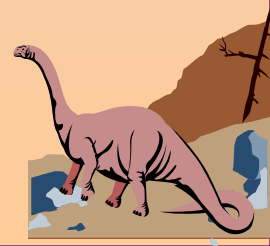
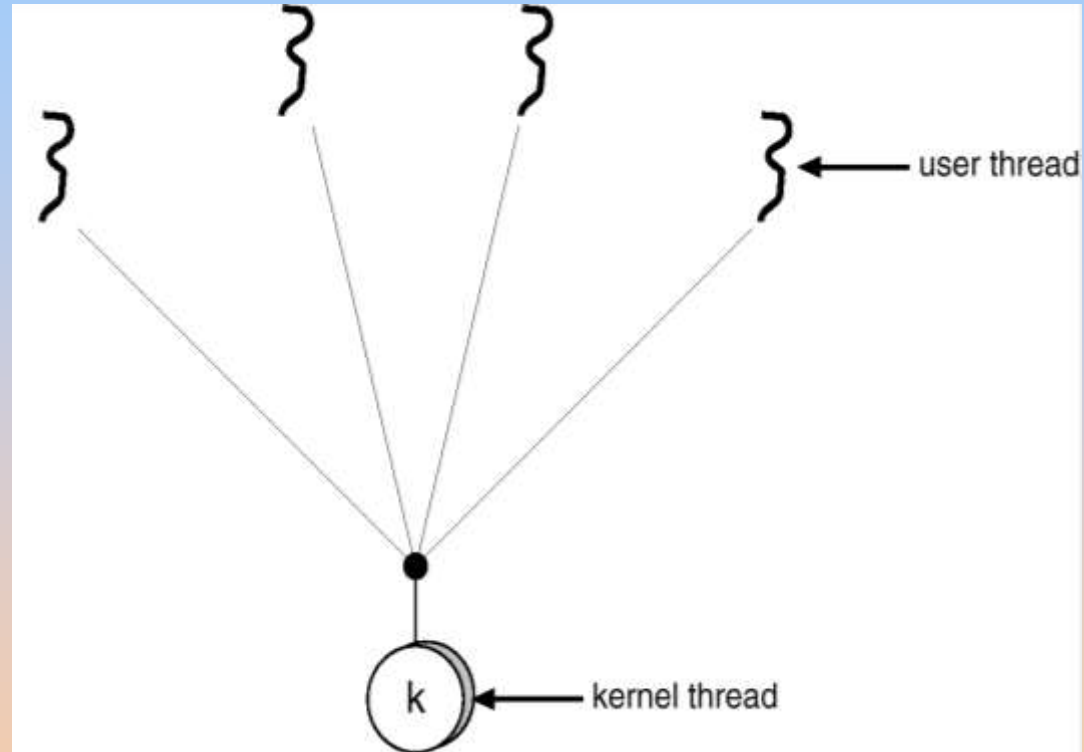
- Alcuni S.O. implementano sia thread di sistema che thread di utente.
- Questo genera differenti modelli di gestione dei thread:
 - ☞ Molti -a-Uno
 - ☞ Uno-ad-Uno
 - ☞ Molti -a-Molti





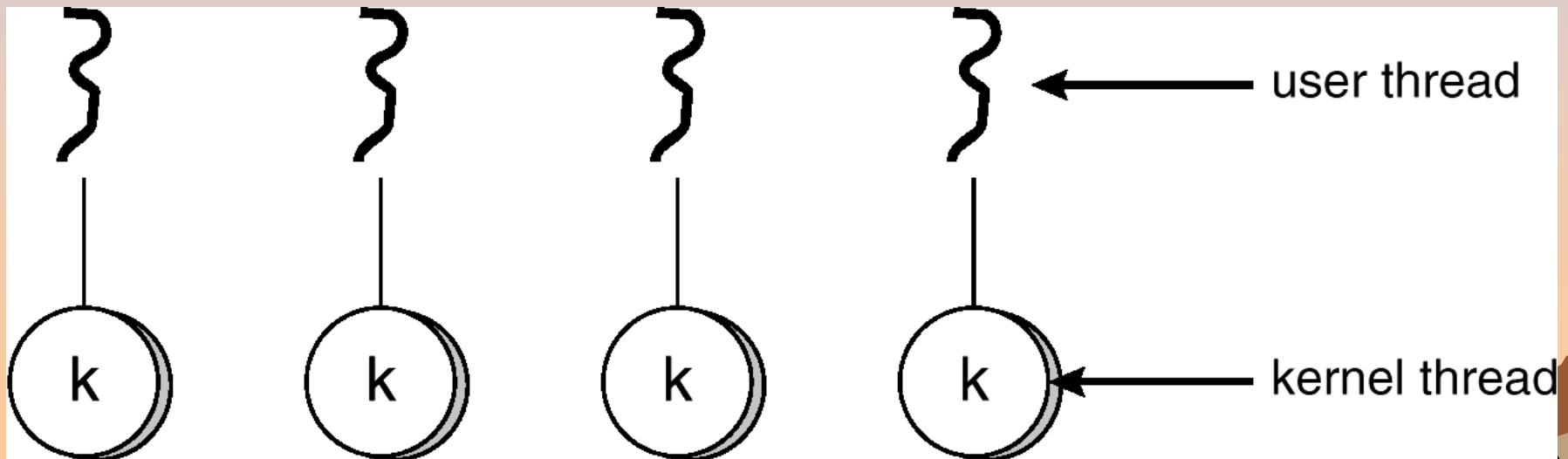
Modello da molti a uno

- Fa corrispondere molti thread al livello d'utente ad un singolo thread al livello del nucleo.
- La gestione è efficiente poichè si svolge nello spazio utente.
- L'intero processo, però, rimane bloccato se un thread invoca una chiamata di tipo bloccante.
- Impossibilità di parallelismo.
- Esempio: Solaris 2.



Modello da uno a uno

- Mette in corrispondenza ciascun thread del livello utente con un thread del livello kernel.
- Offre un alto grado di concorrenza: se un thread invoca una chiamata bloccante è possibile eseguire un altro thread.
- Svantaggio: la creazione di ogni thread al livello utente comporta la creazione del corrispondente thread al livello del kernel.
 - ☞ Maggiore carico (thread kernel più pesanti da gestire, si cerca quindi di limitare il numero di thread per processo).
- Esempio: Windows 95/98.





Modello da molti a molti

- Mette in corrispondenza più thread del livello utente con un numero minore o uguale di thread del livello kernel.
- Permette al sistema operativo di creare un numero sufficiente thread kernel.
- Se un thread invoca una chiamata bloccante il kernel può fare in modo che si esegua un altro thread.
- Esempi: Solaris 2 e Windows NT/2000.

