Lezione 4 – I Thread

ITIS Da Vinci - A.S. 2018/19

Classe 4 Tecnologia e Progettazione

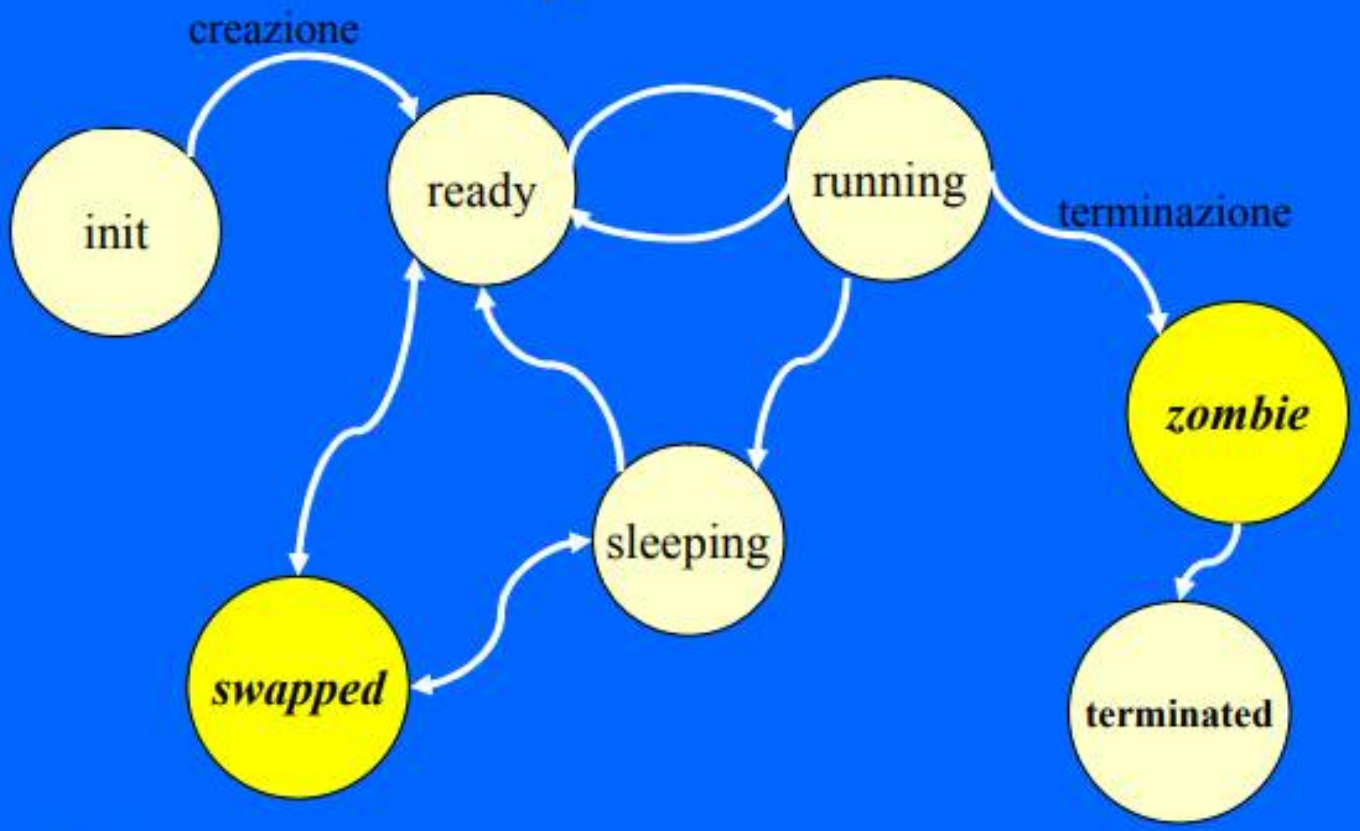
Prof. Maurizio Masetta

Stati di un processo

Stati di un processo

Transizioni di stato:

Esempio: Unix/Linux



Zombie: il processo è terminato, ma è in attesa che il padre ne rilevi lo stato di terminazione.

Swapped: il processo (o parte di esso) è temporaneamente trasferito in memoria secondaria.

Cos’è un Thread ?

- *Un thread (o processo leggero) è una sequenza di istruzioni di un programma in corso di esecuzione.*

Praticamente il flusso esecutivo di un processo viene scomposto in più flussi concorrenti.

I Threads di uno stesso processo condividono l'area dati e codice.

È, pertanto , necessaria una sincronizzazione nell'accesso ai dati globali.

Vantaggio : il cambio di contesto tra threads è più veloce che tra processi .

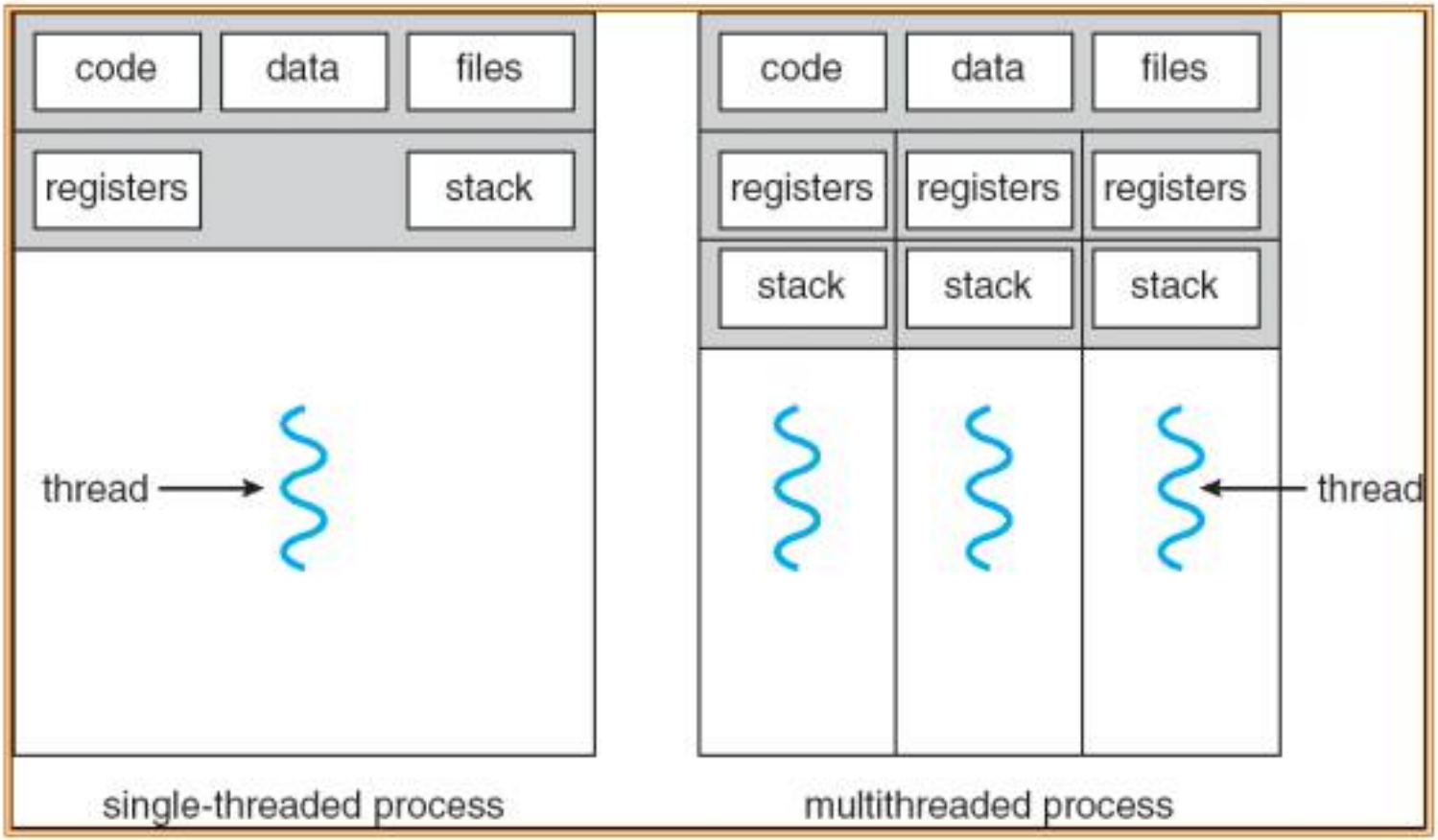
Concetto di Thread

- Anche chiamati ***lightweight process*** perché possiedono un contesto più snello rispetto ai processi
- **Flusso di esecuzione indipendente**
  - **interno ad un processo**
  - condivide lo spazio di indirizzamento con gli altri thread del processo
  - Rappresentato da un *thread control block* (TCB) che punta al PCB del processo contenitore

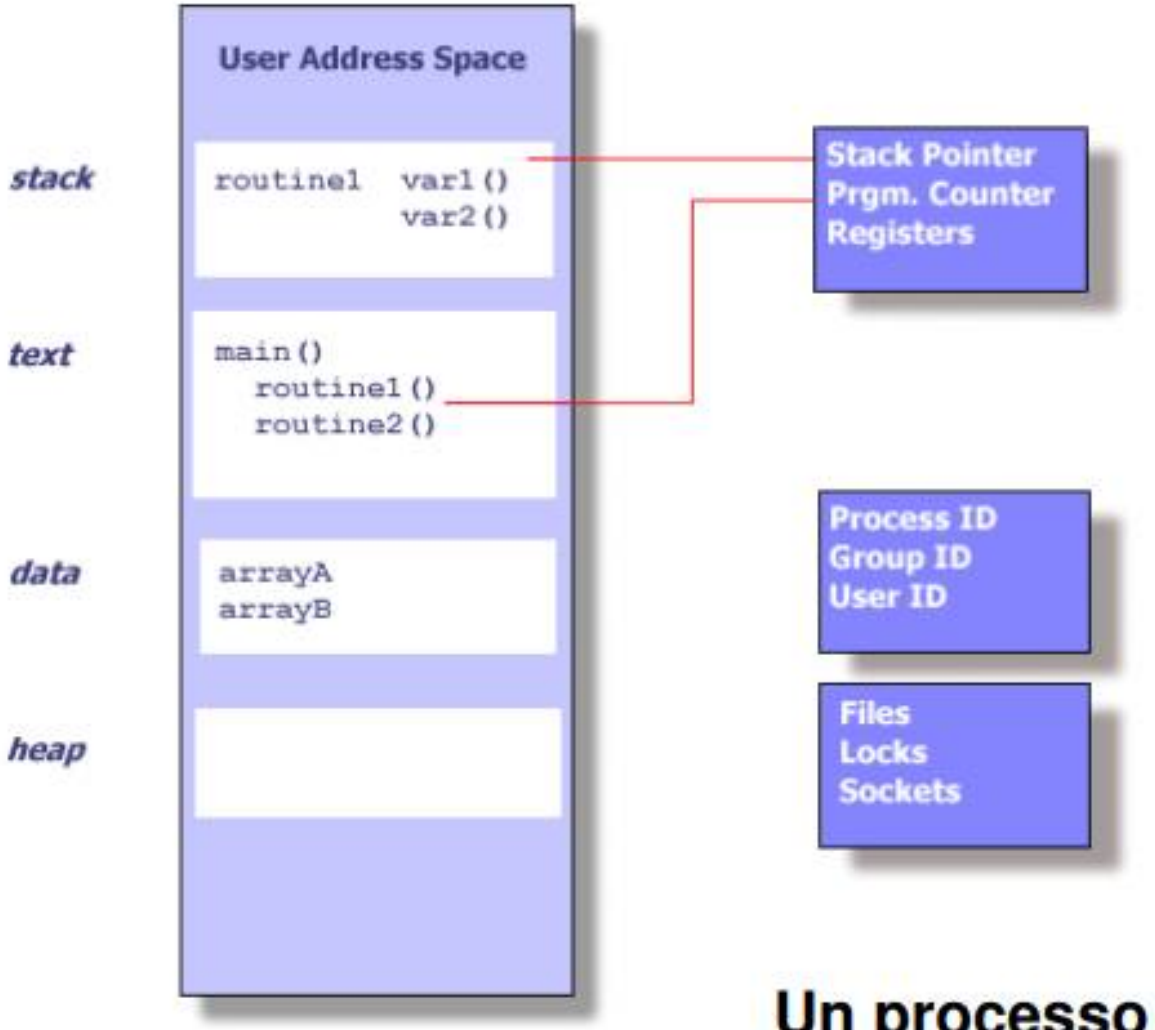
Confronto tra processi e thread

	processi	Thread
Creazione distruzione	Allocazione, copia e deallocazione di grandi quantità di memoria	Richiedono solo la creazione di uno stack per il thread
Errore	non può danneggiare altri processi	può danneggiare altri thread e l'intero processo a cui appartiene
Codice	Un processo può modificare il proprio codice mediante il cambiamento di eseguibile	il codice è fissato e presente nella sezione text del processo cui appartiene
Condivisione	E' onerosa e deve essere implementata dal programmatore	E' automatica perchè tutti i thread condividono la memoria del processo cui appartengono
Mutua esclusione	E' garantita automaticamente dall'isolamento dei vari processi	Deve essere realizzata dal programmatore mediante semafori ecc.
Prestazioni	limitate dall'overhead di gestione	elevate
Concorrenza	Limitata dalla difficoltà di comunicazione	elevate

Processi a singolo thread e multithread

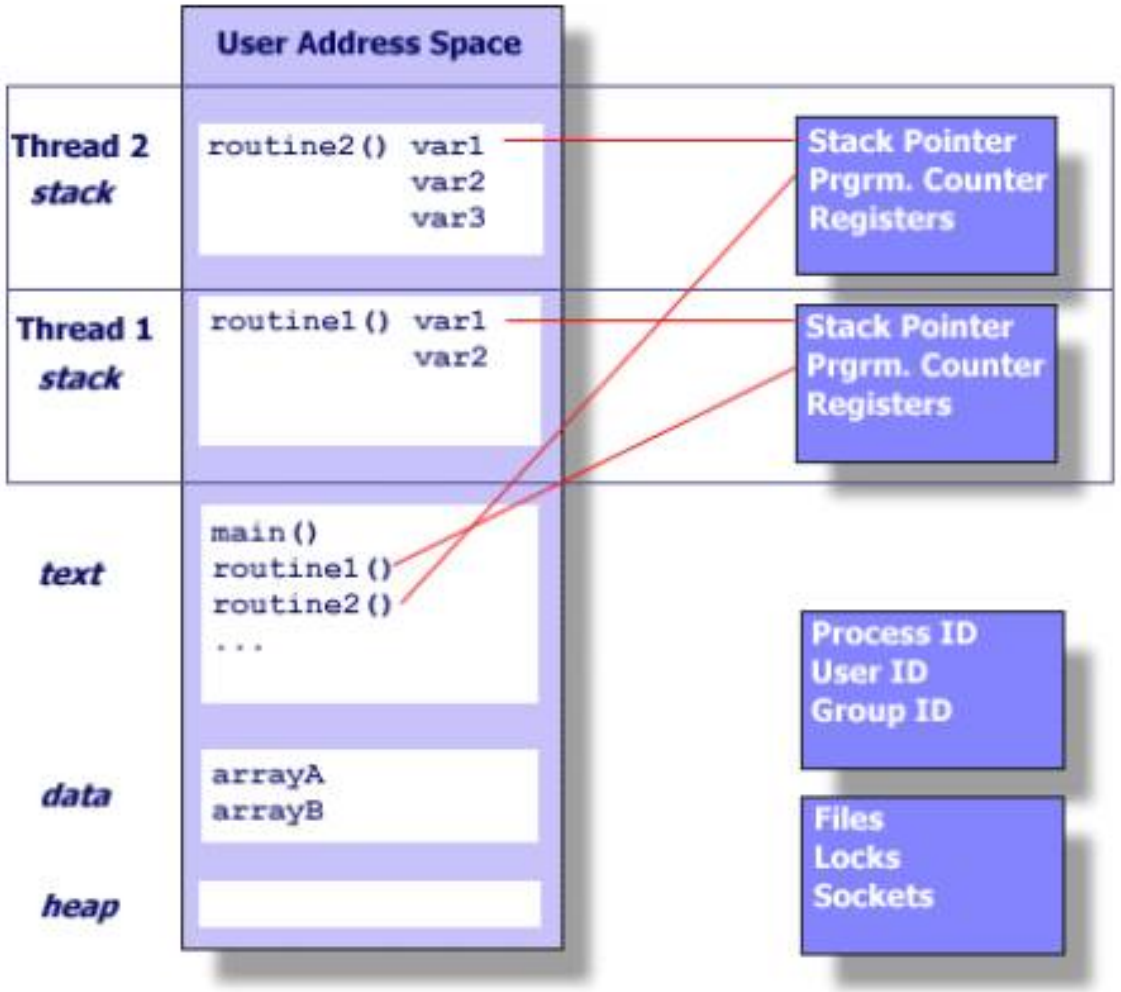


In Unix: Thread e Processi (1)



Un processo

In Unix: Thread e Processi (2)



Un thread all'interno di un processo

Contesto: Thread vs Processi

- **Contesto di un Thread**
  - stato della computazione (registri, stack, PC…)
  - attributi (schedulazione, priorità)
  - descrittore di thread (tid, priorità, segnali pendenti, …)
  - memoria privata (TSD)
- **Contesto di un processo**; tutto quello che è nel contesto di un thread, ed inoltre:
  - spazio di memoria
  - risorse private
    (con le corrispondenti tabelle dei descrittori)

Vantaggi dei Thread

- **Semplicità di comunicazione** inter-thread
  - Condivisione di risorse più naturale
  - Programmazione più semplice

- **Efficienza**
  - Creazione e distruzione rapidi
  - Context switch e quindi scheduling più veloci
  - Il più veloce dei meccanismi di comunicazione
  - Utilizzo di architetture multiprocessore

Context Switch

In informatica la **Context Switch** o *commutazione di contesto* è una particolare operazione del sistema operativo che cambia il processo correntemente in esecuzione su una CPU.

Questo avviene all'occorrenza di una qualsiasi interruzione dovuta allo scheduler, ma anche a interruzioni dovute a errori di altri processi o segnali; viene effettuato per salvare tutte le informazioni necessarie al riavvio successivo del processo.

Permette a più processi di condividere una stessa CPU, ed è utile quindi sia nei sistemi monoprocessore, perché consente di eseguire più programmi contemporaneamente, sia nell'ambito del calcolo parallelo, perché consente un migliore bilanciamento del carico.

Svantaggi dei Thread

- **Difficoltà di ottenere risorse private**
  - per ottenere memoria privata all'interno di un thread esistono appositi meccanismi
- **Pericolo di interferenza**
  - la condivisione delle risorse accentua il pericolo di interferenza
  - gli accessi concorrenti devono essere sincronizzati di modo da evitare interferenze (thread safeness)

Complementarietà Thread / Processi

- Processo
  - "Unità di allocazione delle risorse"
- Thread
  - flusso di esecuzione indipendente ma esistente nel contesto di un processo
- Quali utilizzare?

  dipende, una scelta da valutare di caso in caso

Modello di Thread: User vs Kernel

- **User-Level:** thread all'interno del processo utente gestiti da una libreria specifica (da un supporto run-time)
  - il kernel non è a conoscenza dell'esistenza dei thread
  - lo switch non richiede chiamate al kernel

- **Kernel-Level**: gestione dei thread affidata al kernel tramite chiamate di sistema
  - gestione integrata processi e thread dal kernel
  - lo switch è provocato da chiamate al kernel

User Level

Vengono implementati grazie ad apposite librerie (thread package) che contengono funzioni per creare, terminare, sincronizzare, effettuare lo scheduling ed il cambio di contesto (il nucleo gli ignora e gestisce solamente i processi)

VANTAGGI:

- Efficienza di gestione (tempi di switching molto ridotti non richiedendo chiamate al kernel)
- Possono essere implementati su qualsiasi S.O.(alto grado di portabilità tra macchine e sistemi diversi)

SVANTAGGI:

- Se un thread effettua una system call (ad es, per motivi di I/O) oltre a sospendere se stesso provoca la sospensione del processo che lo ha generato (quindi anche di tutti gli altri thread)
- Non è possibile sfruttare il parallelismo fisico in architetture multiprocessore(interni ad un processo e quindi assegnati ad un unico processore)

Thread nello spazio utente

- La gestione dei thread è effettuata a livello utente attraverso una libreria

- Tre principali **librerie di thread**:
  - *POSIX Pthreads* (in verità, sia a livello utente che a livello kernel)
  - *Win32 thread* (a livello kernel)
  - *Java thread* (implementata in POSIX, in Win32 ed altre)

Kernel Level (Linux,Unix e Windows)

La gestione è affidata al kernel tramite chiamate di sistema, vengono gestiti come tutti gli altri processi,schedulati, sospesi, risvegliati, gli vengono assegnate risorse di sistema ecc.

VANTAGGI:

- se un thread si sospende continuano ad evolversi altri thread generati dallo stesso processo (sono tra loro schedulati in modo autonomo)
- In architetture multiprocessor si può sfruttare al massimo il parallelismo fisico

SVANTAGGI:

- Tempi lunghi impiegati dal kernel per il cambio di contesto, reso ancora più complesso in quanto deve gestire sia processi che thread

Thread nello spazio kernel

- Supportati dai kernel

- Esempi:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

Vantaggi/Svantaggi: User Level

- Vantaggi:
  - Lo switch non coinvolge il kernel, e quindi **non ci sono cambiamenti della modalità di esecuzione**
  - Maggiore **libertà nella scelta dell'algoritmo di scheduling** che può anche essere personalizzato
  - Poichè le chiamate possono essere raccolte in una libreria, c'è **maggiore portabilità tra SO**
- 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** a multiprocessore simmetrico (**SMP**) **due processori non risulteranno mai associati a due thread del medesimo processo**

Vantaggi/Svantaggi: Kernel Level

- Vantaggi:
  - il kernel può **eseguire più thread** dello stesso processo anche su più processori
  - **il kernel stesso può essere scritto multithread**

- Svantaggi:
  - **lo switch coinvolge chiamate al kernel** e questo comporta un costo
  - **l'algoritmo di scheduling è meno facilmente personalizzabile**
  - **meno portabile**

Soluzione mista (Solaris)

Permette di creare dei thread a livello utente che devono essere preventivamente definiti a livello di kernel e lasciano all'utente le politiche di scheduling e di sincronizzazione.

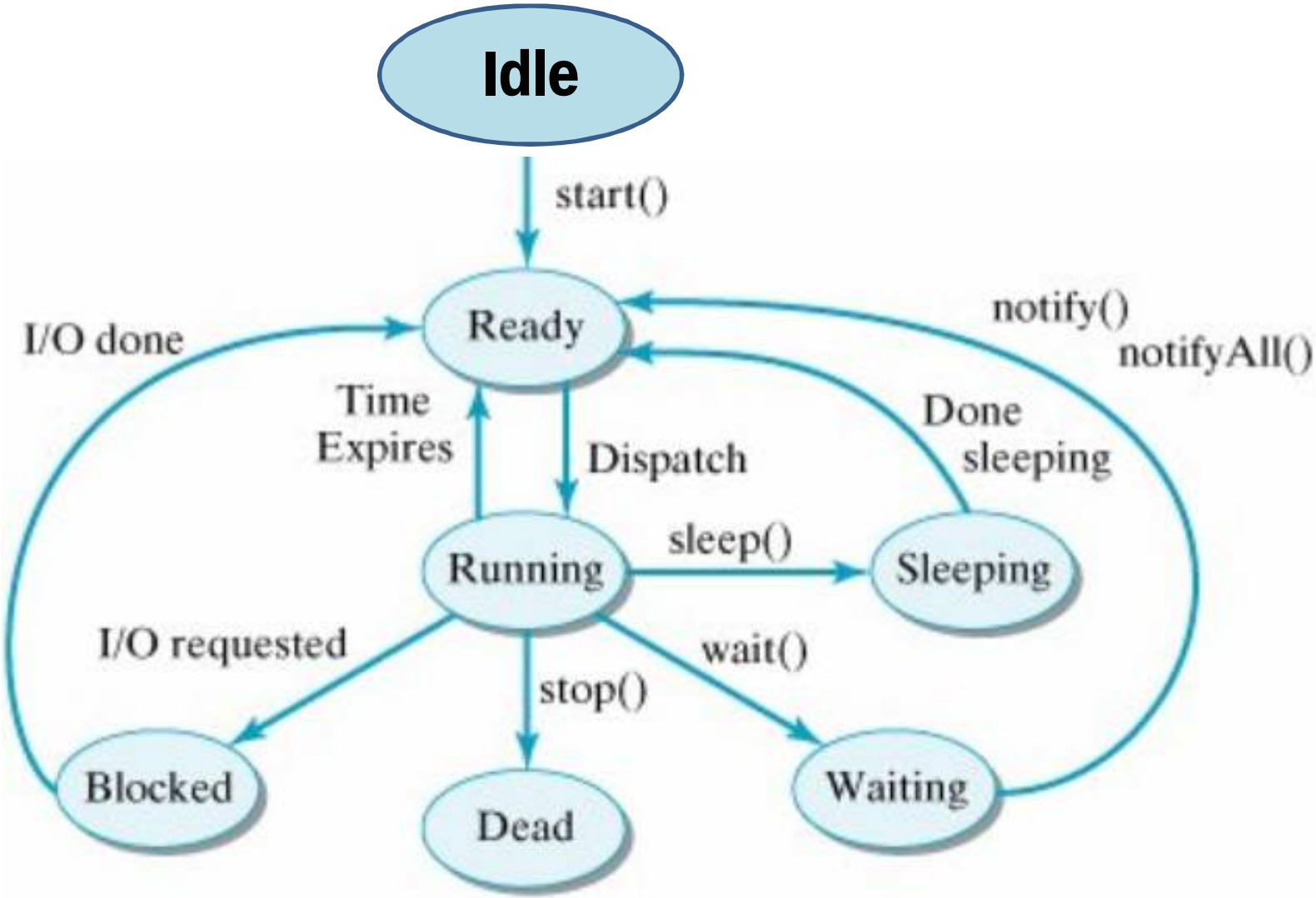
VANTAGGI:

- se un thread si sospende questo non blocca necessariamente altri thread generati dallo stesso processo.
- Thread della stessa applicazione possono essere eseguiti in parallelo su processori diversi.

Pthread

- Uno **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)

Stati di un thread



Gli stati dei Thread

- **Idle**: prima di essere avviato
- **Dead**: termina le sue istruzioni
- **Blocked**: in attesa di richieste o di completamento di operazioni di tipo I/O
- **Sleeping**: sospeso per un periodo
- **Waiting**: in attesa di un evento (*ad esempio l'attesa di dati dal disco*)
- **Running**: in esecuzione
- **Ready**: pronto per l'esecuzione ma attende che si renda disponibile la **CPU**

Gli stati dei Thread

Quando un **Thread** si blocca per una richiesta I/O e passa a **Blocked** il sistema potrebbe decidere si eseguire un altro Thread dello stesso processo o di altri pronti (dipende dal tipo di implementazione).

- Es1. Nei Thread Java implementato a livello utente il sistema non vede altri Thread dello stesso processo e quindi ne esegue uno di un altro processo.
- Es2. Nei Thread C implementati a livello Kernel il sistema può gestire lo «scheduling» a livello Thread seguendo una politica definita dal S.O.

Utilizzo dei thread

- Esecuzioni di programmi in «foreground» e in «background» per esempio la gestione dell'I/O da parte dell'utente in «foreground» il thread esegue dei calcoli in «background»
- Comuni sono gli aggiornamenti di calcolo di formule matematiche in Excel o la reimpaginazione, il controllo ortografico in Word
- Nella RAM le operazioni di «garbage collection» vengono svolti da Thread così come il backup automatico

garbage collection: modalità automatica di gestione della memoria, mediante la quale un sistema operativo, o un compilatore e un modulo di run-time, liberano porzioni di memoria non più utilizzate dalle applicazioni