

# **Processi, Thread, IPC & Scheduling**

## **Sistemi Operativi (M-Z)**

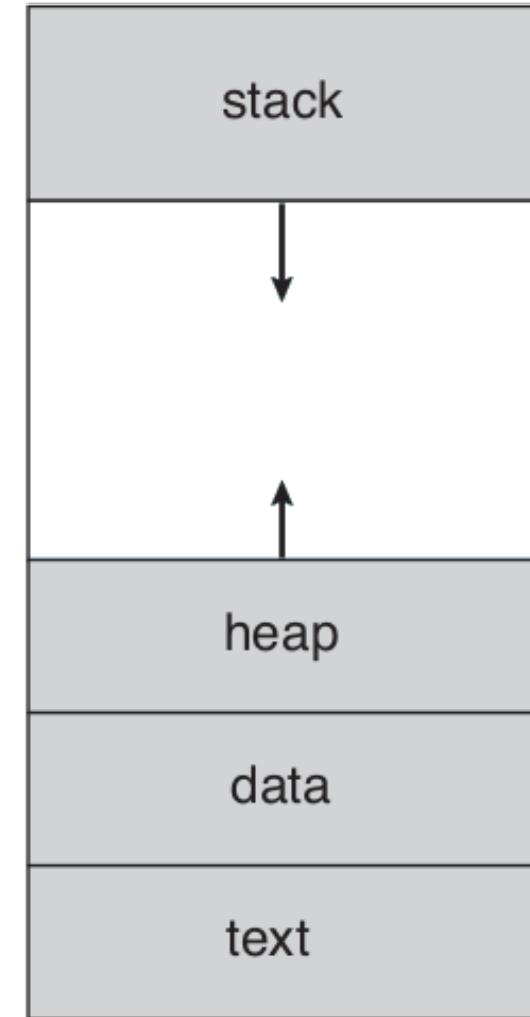
C.d.L. in Informatica  
(laurea triennale)  
A.A. 2020-2021

**Prof. Mario F. Pavone**

Dipartimento di Matematica e Informatica  
Università degli Studi di Catania  
[mario.pavone@unict.it](mailto:mario.pavone@unict.it)  
[mpavone@dmi.unict.it](mailto:mpavone@dmi.unict.it)

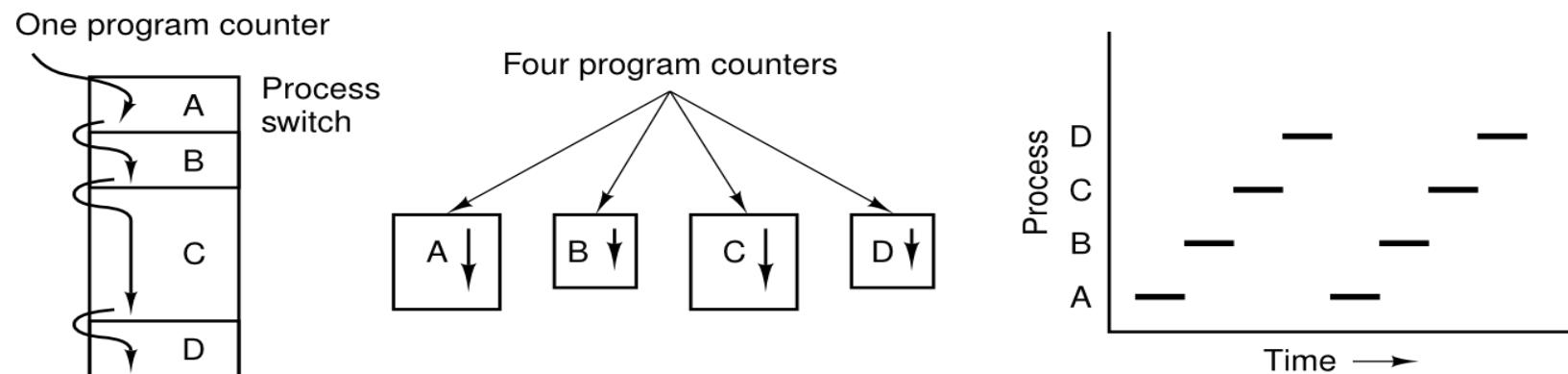
# Processo

- **Definizione:** una istanza di esecuzione di un programma.
- Ad ogni processi è associato il suo **spazio degli indirizzi**:
  - codice eseguibile;
  - dati del programma;
  - stack;
  - copia dei **registri** della CPU;
  - **file aperti**;
  - **allarmi** pendenti;
  - processi imparentati.
- Tutte le informazioni relative al processo devono essere salvate (es. file aperti)
- **Tabella dei processi** con un **Process Control Block** (PCB) per ogni processo.



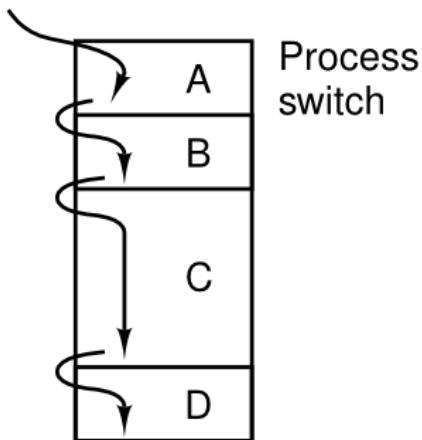
# Modello dei processi

- **Multiprogrammazione e pseudo-parallelismo.**
- È più semplice ragionare pensando a **processi sequenziali** con una **CPU virtuale** dedicata.

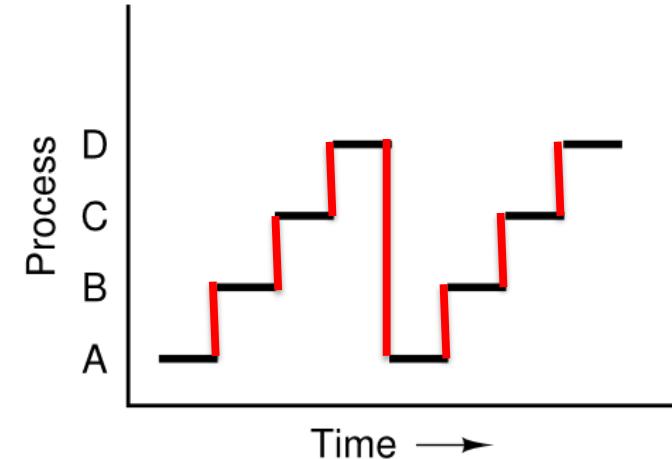
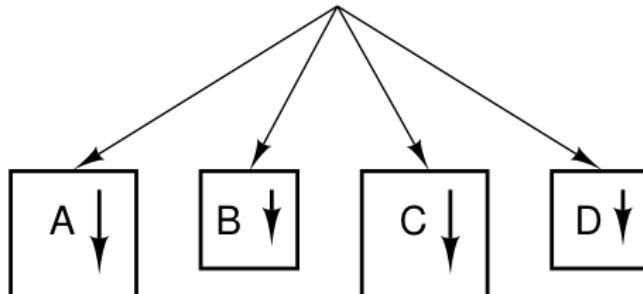


# Modello dei processi

One program counter



Four program counters



- Esiste un solo PC fisico:
  - PC logico viene caricato nel PC fisico
  - STOP => PC fisico viene memorizzato nel PC logico

# Modello dei

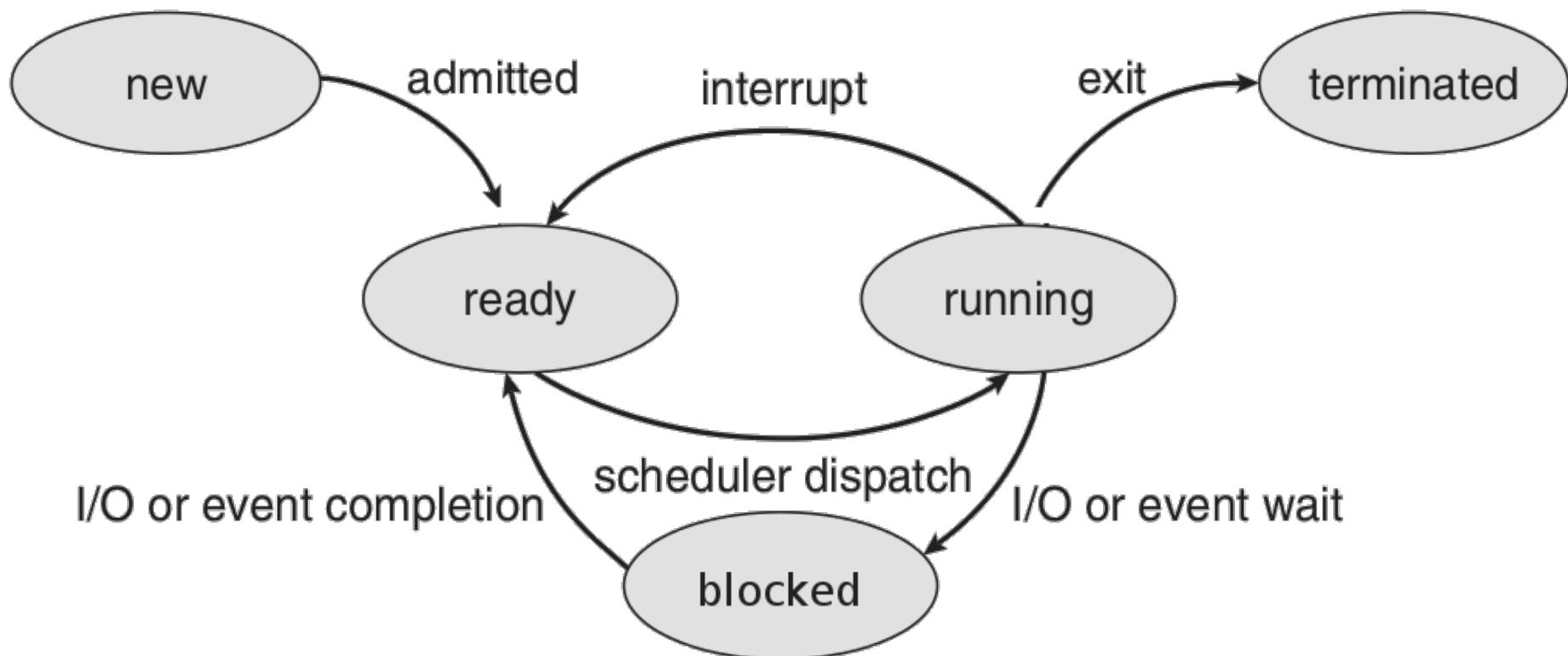
Multiprogrammazione e pseudo-parallelismo.

NOTA: due processi che eseguono lo stesso programma sono comunque distinti



# Stato di un processo

- 3 stati principali (+ 2 addizionali);
- transizioni.



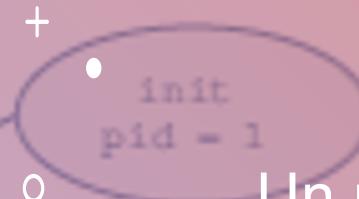
# Creazione dei processi

- **Creazione di un processo:**
  - in fase di inizializzazione del sistema;
  - processi attivi e in background (daemon)
  - esecuzione chiamate di sistema di creazione processo
  - richiesta dell'utente
- Creare nuovo processo da un altro processo:
  - **sdoppiamento del padre:** fork e exec (UNIX);
  - **nuovo processo per nuovo programma:** CreateProcess (Win32).
- NOTA: dopo la creazione del nuovo processo il padre e il figlio hanno il proprio spazio degli indirizzi.
  - una modifica nello spazio di indirizzo di uno non è visibile all'altro

# terminazione dei processi

- **uscita normale** (volontario): exit (UNIX), ExitProcess (Win32);
- **uscita su errore** (volontario);
- **errore critico** (involontario): alcuni sono gestibili, altri no;
- **terminato da un altro processo** (involontario): kill (UNIX), TerminateProcess (Win32).

# Gerarchia dei Processi

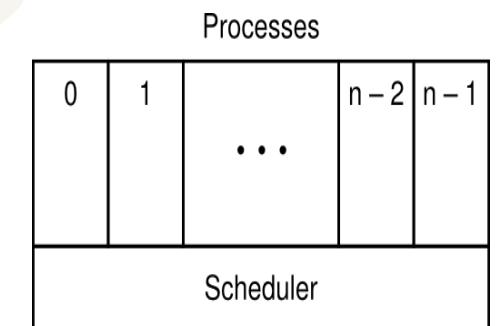
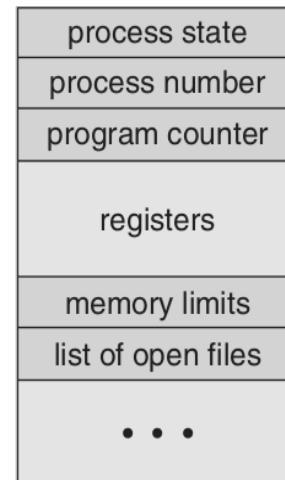


- Un processo può crearne un'altro
- processo padre e processo figlio
- processo figlio può a sua volta crearne altri
- **UNIX:** process group
  - processo speciale *init*
- **In Windows:** NO
  - Tutti i processi sono uguali
  - handle: token per il processo padre

# Tabella dei processi

- SO mantiene una **Tabella dei processi**;
- **Process Control Block** (PCB);
- Contiene tutte le informazioni importanti sullo stato del processo
- **Scheduler**
- NOTA: i campi della tabella dipendono dal SO

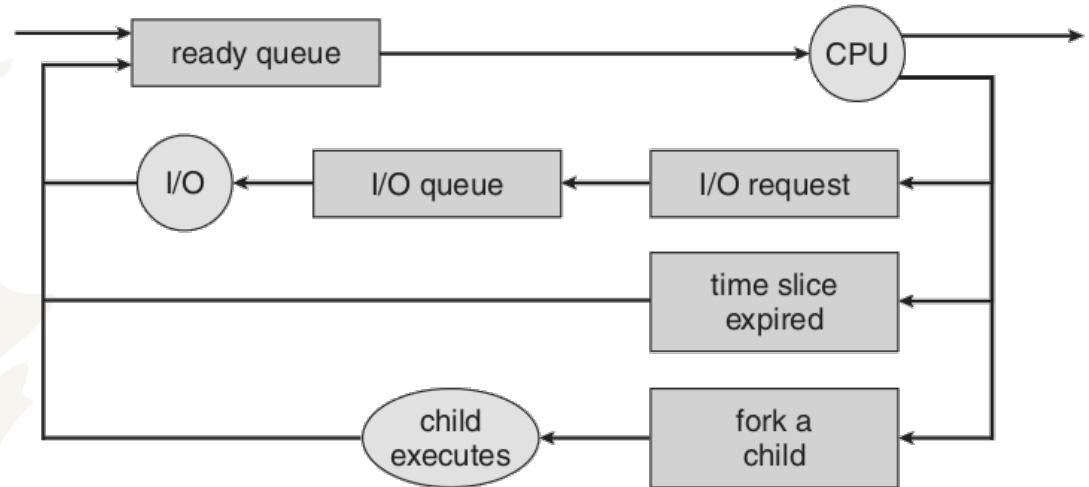
Process management	Memory management	File management
Registers Program counter Program status word Stack pointer Process state Priority Scheduling parameters Process ID Parent process Process group Signals Time when process started CPU time used Children's CPU time Time of next alarm	Pointer to text segment info Pointer to data segment info Pointer to stack segment info	Root directory Working directory File descriptors User ID Group ID



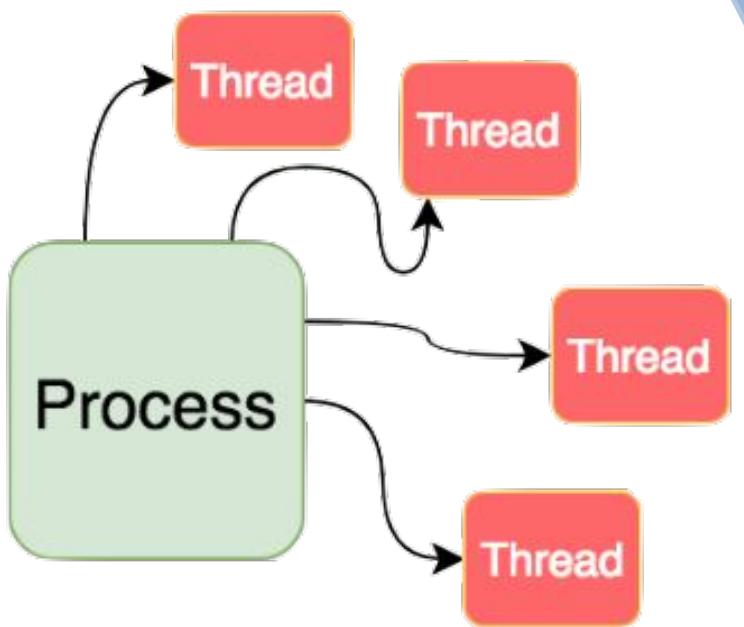
# Tabella dei processi

- **Gestione** degli **interrupt** per il passaggio di processo
- **vettore di interrupt:** indirizzo alla **procedura di servizio dell'interrupt:**
  - salvataggio nello stack del PC e del PSW nello stack attuale;
  - caricamento dal vettore degli interrupt l'indirizzo della procedura associata;
  - salvataggio registri e impostazione di un nuovo stack;
  - esecuzione procedura di servizio per l'interrupt;
  - interrogazione dello scheduler per sapere con quale processo proseguire;
  - ripristino dal PCB dello stato di tale processo (registri, mappa memoria);
  - ripresa nel processo corrente.

# Code e accodamento



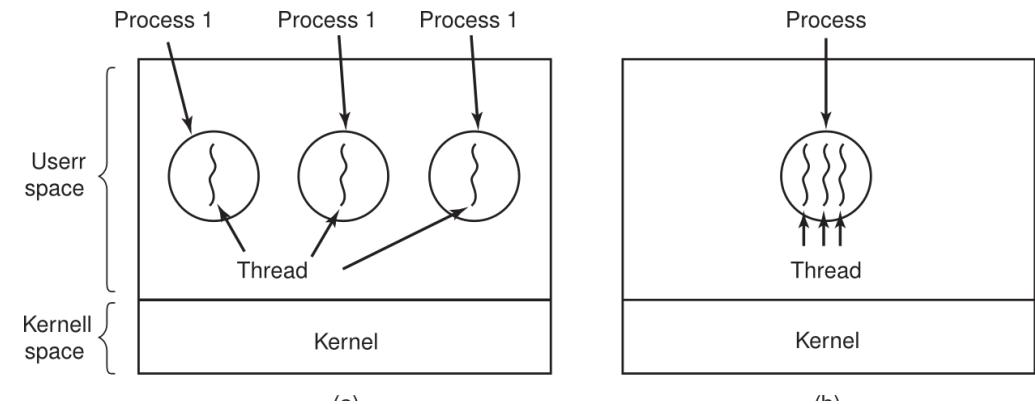
- **Coda dei processi pronti e code dei dispositivi;**
  - strutture collegate sui PCB;
- **Diagramma di accodamento:**



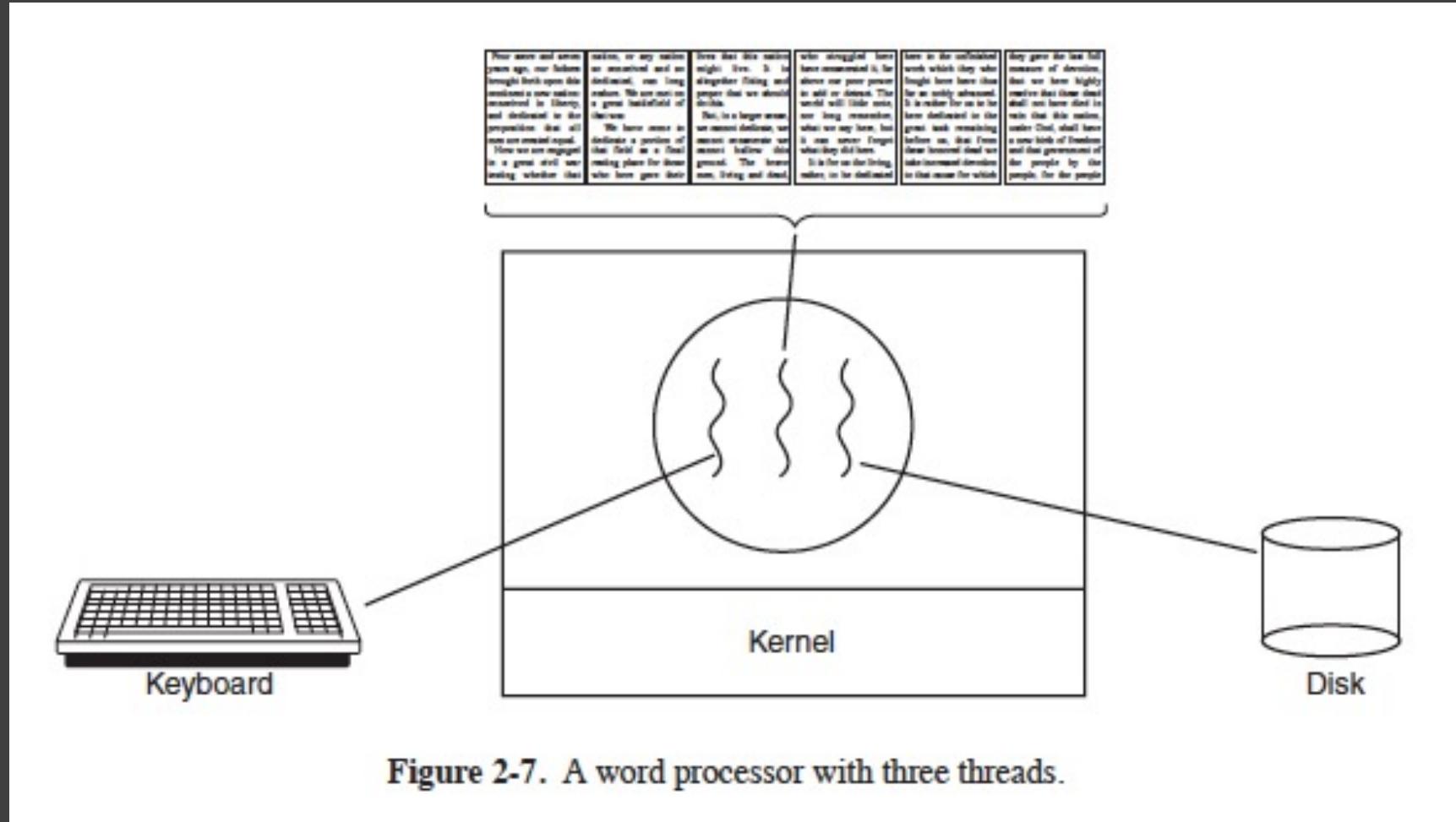
# I Thread



# Thread



- Modello dei processi: entità indipendenti che **raggruppano risorse** e con un **flusso di esecuzione**;
- può essere utile far condividere a più flussi di esecuzione lo stesso spazio di indirizzi: **thread**;
- quando può essere utile?
  - esempi: web-browser, videoscrittura, web-server, ...



**Figure 2-7.** A word processor with three threads.

**Thread: an example**

## WORD PROCESSOR

# Thread: an example

## WEB SERVER PROCESS

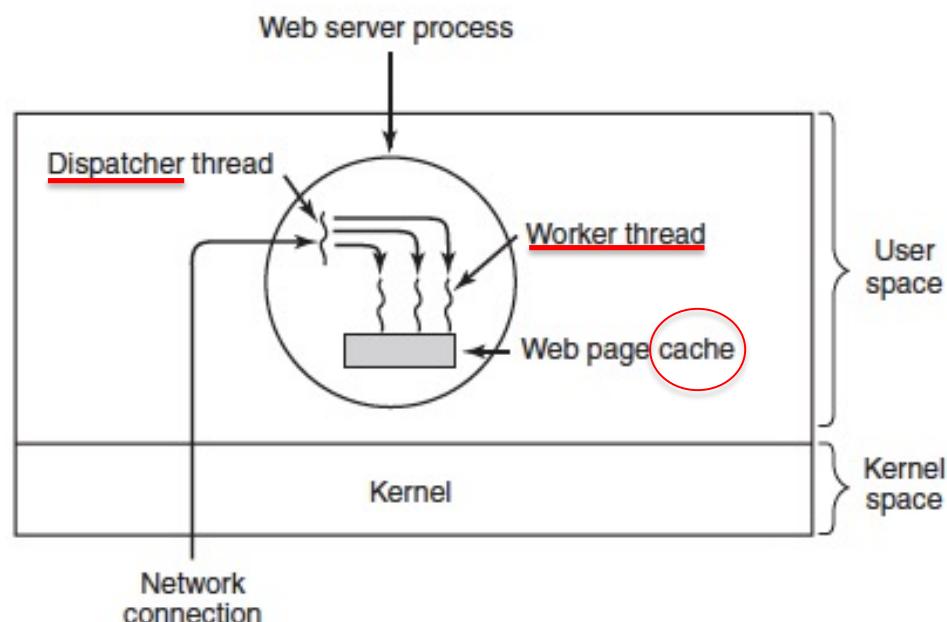
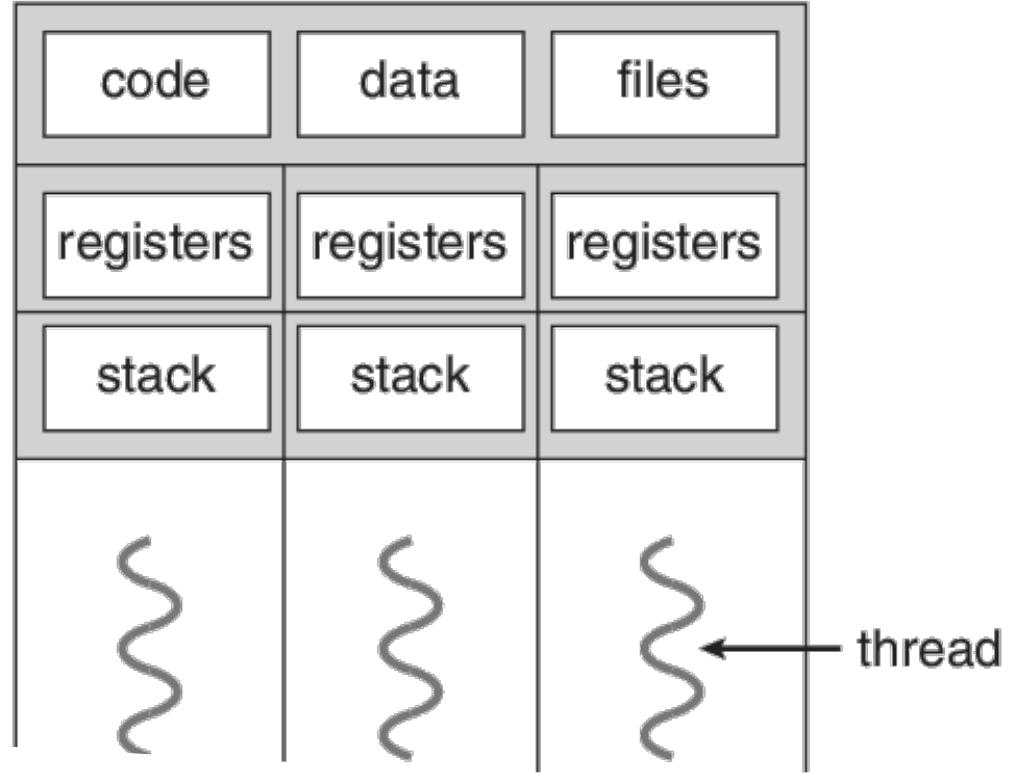
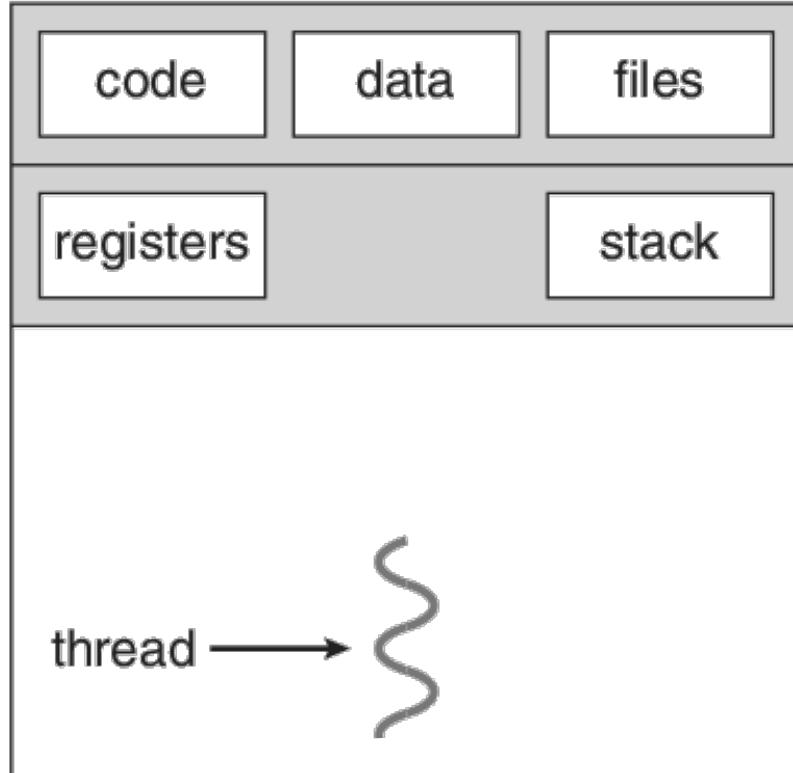


Figure 2-8. A multithreaded Web server.

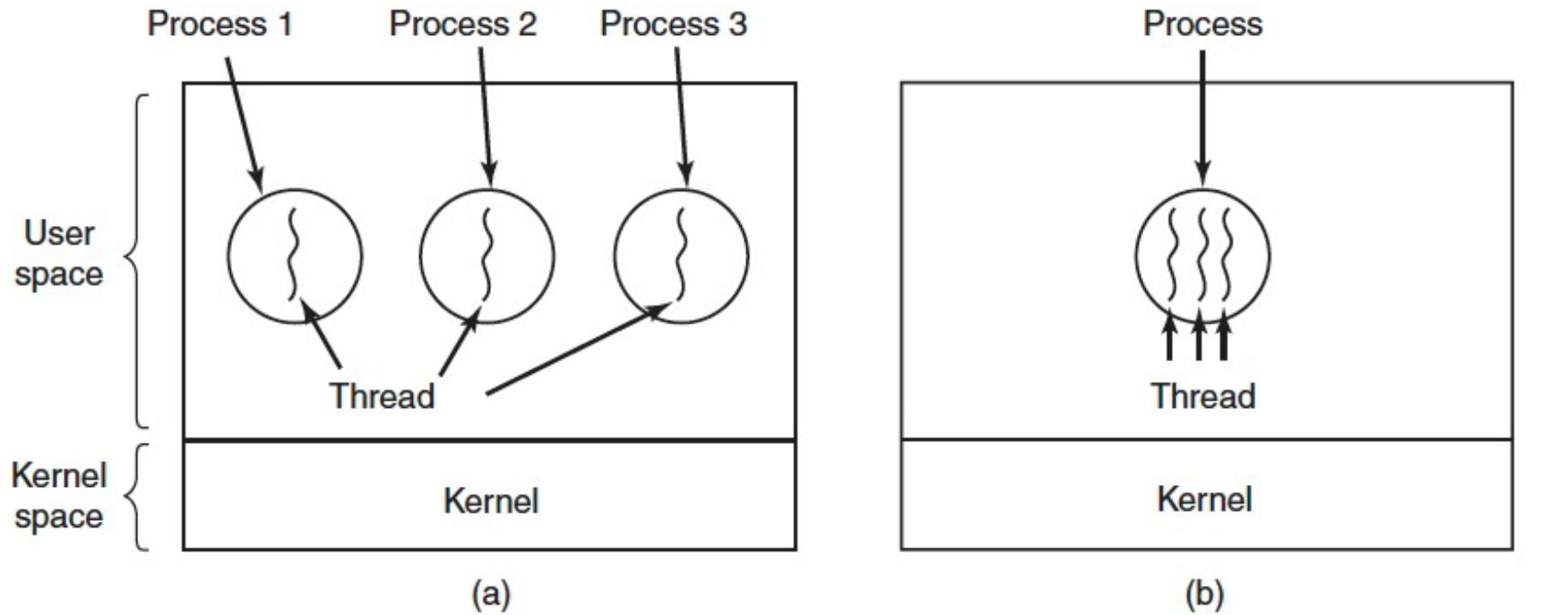
NOTA: il *dispatcher* è un ciclo infinito che acquisisce richieste di lavoro e le passa ad un *worker thread*



# Il Modello Thread

- Un thread è **caratterizzato** da:
  - PC, registri, stack, stato;
  - condivide tutto il resto;
    - non protezione di memoria.
- **scheduling** dei thread;
- **cambio di contesto** più veloce;

*Sistemi Multithreading*

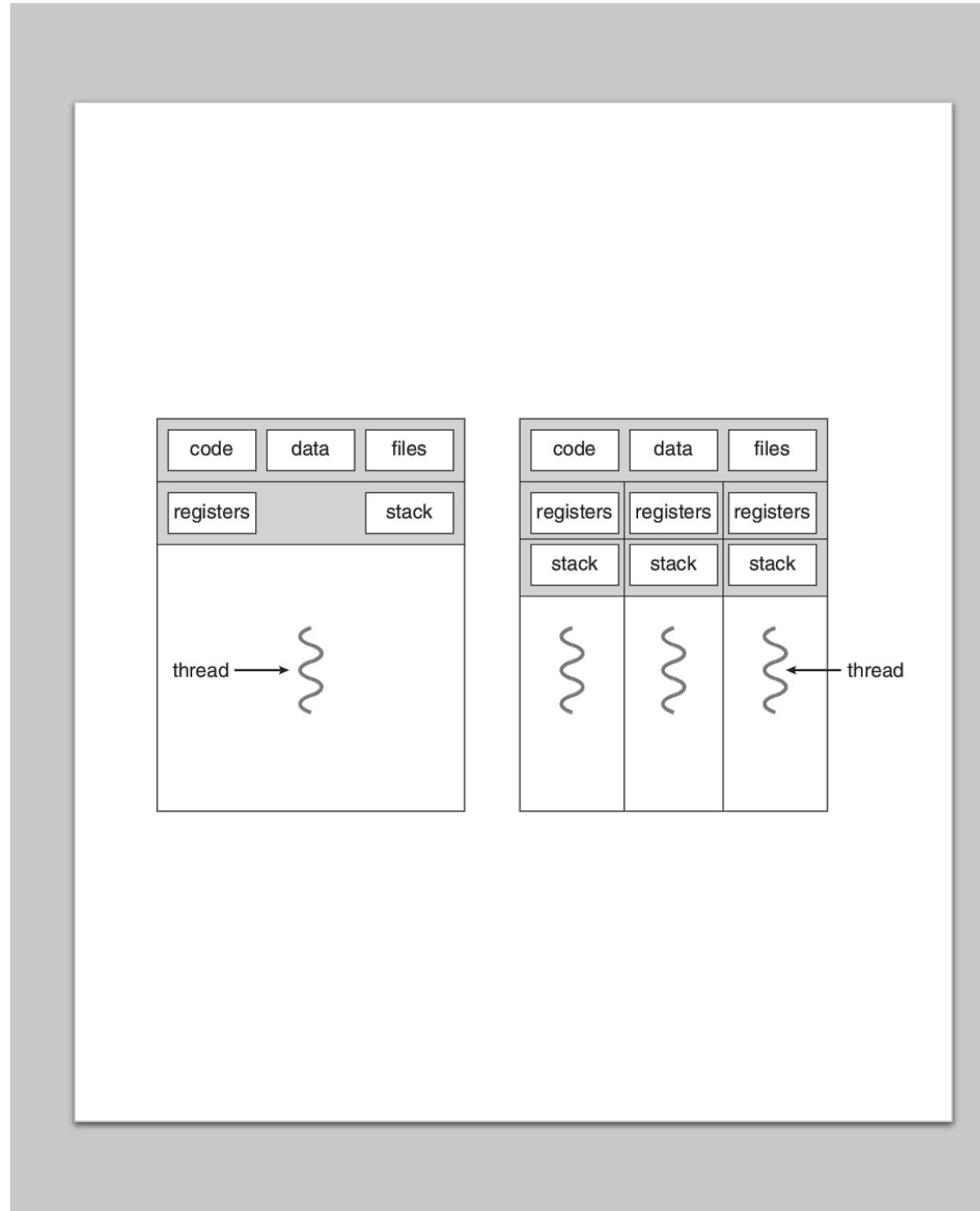


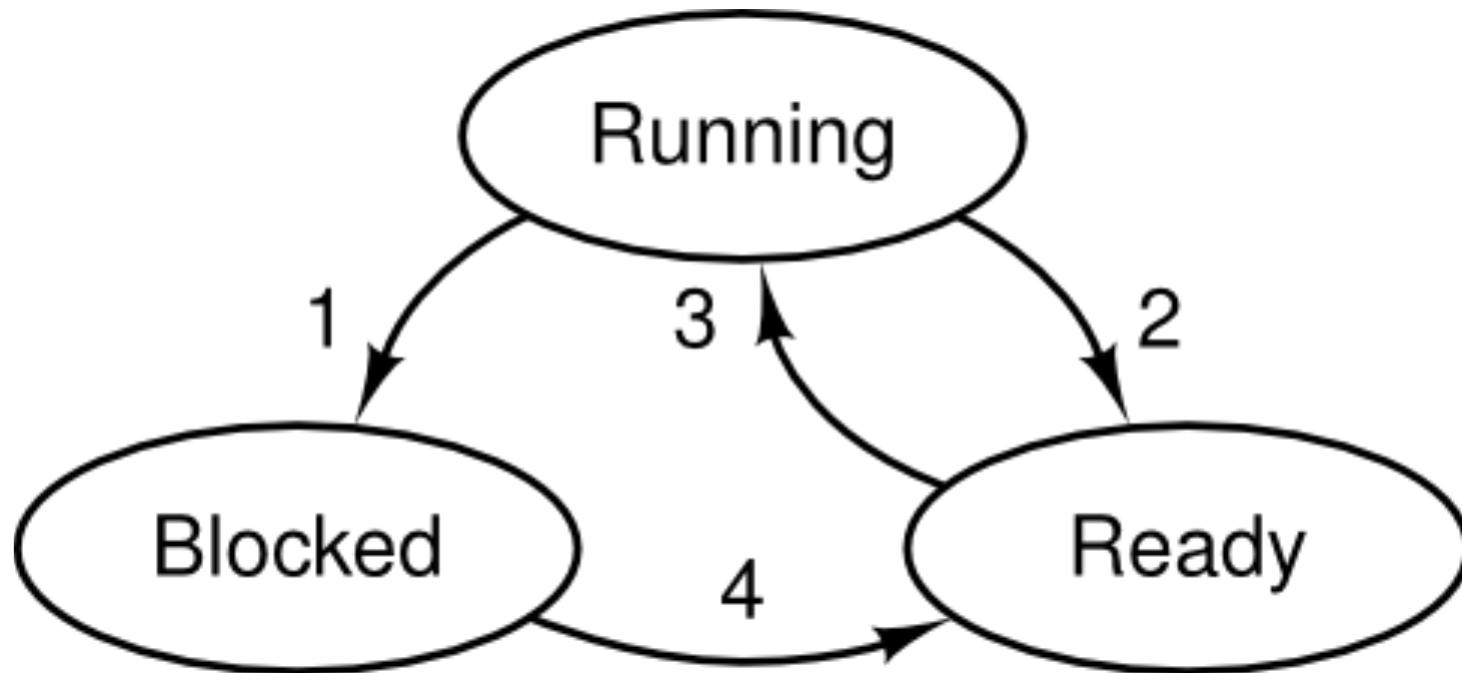
# Thread Vs. Process

- Fig. (a): Ogni processo lavora in spazi degli indirizzi diversi
- Fig. (b): tutti I thread condividono lo stesso spazio degli indirizzi

# Thread

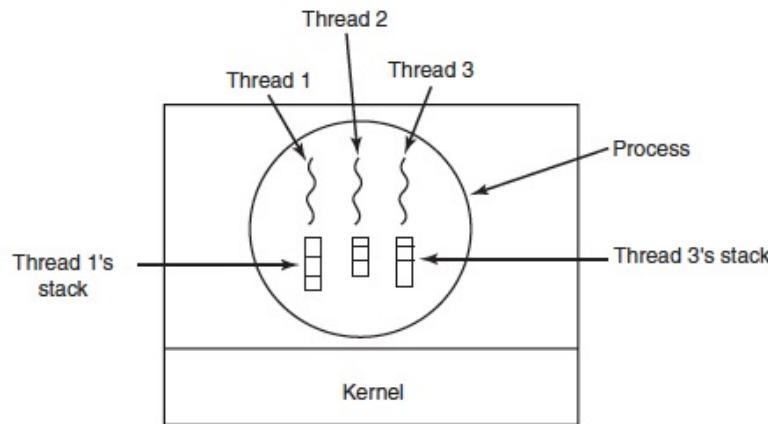
- Thread diversi nello stesso processo non sono indipendenti
- Condivisione e **No protezione di memoria**
  - un thread può leggere, scrivere o cancellare lo stack di un altro thread
- Stesso user -> Cooperano -> non entrano in conflitto
- Thread apre file => è visibile ad ogni thread nel processo
- Unità di gestione delle risorse => Processo





# Thread

- Anche un thread può trovarsi in uno dei seguenti stati
- Transizioni di stato => analogo a quello dei processi.



#### Per-process Items

- Address space
- Global variables
- Open files
- Child processes
- Pending alarms
- Signals and signal handlers
- Accounting information

#### Per-thread Items

- Program counter
- Registers
- Stack
- State

# Thread

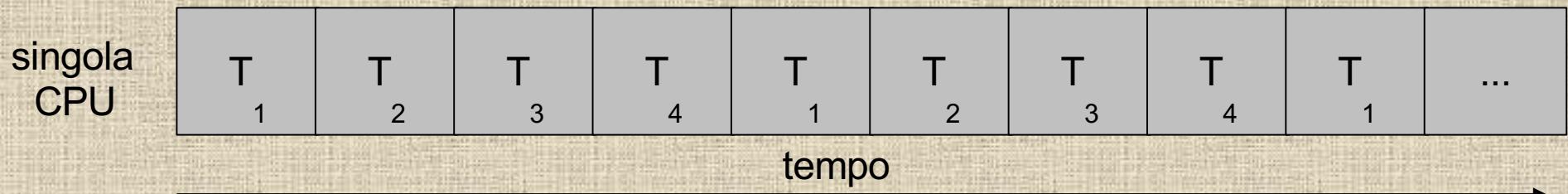
- Ogni Thread ha un proprio Stack
- **storia di esecuzione diversa**

# Operazioni sui Thread

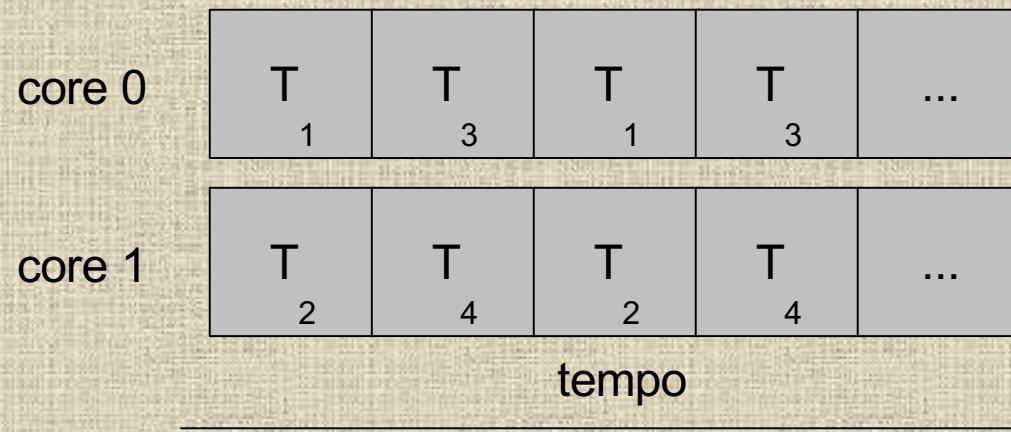
- Nei sistemi Multithreading i processi iniziano con un singolo thread
- **operazioni** tipiche sui thread:
  - **thread\_create**: un thread ne crea un altro;
  - **thread\_exit**: il thread chiamante termina;
  - **thread\_join**: un thread si sincronizza con la fine di un altro thread;
  - **thread\_yield**: il thread chiamante rilascia **volontariamente** la CPU
    - **no clock interrupt**

# Programmazione multicore

- I thread permettono una **migliore scalabilità** con core con hypertreading e soprattutto con sistemi multicore;
- con un sistema single-core abbiamo una esecuzione interleaved;



- su un sistema multi-core abbiamo parallelismo puro.



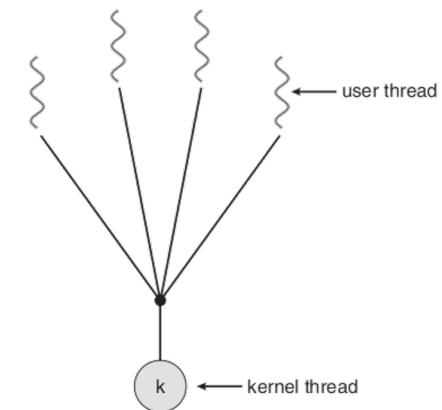
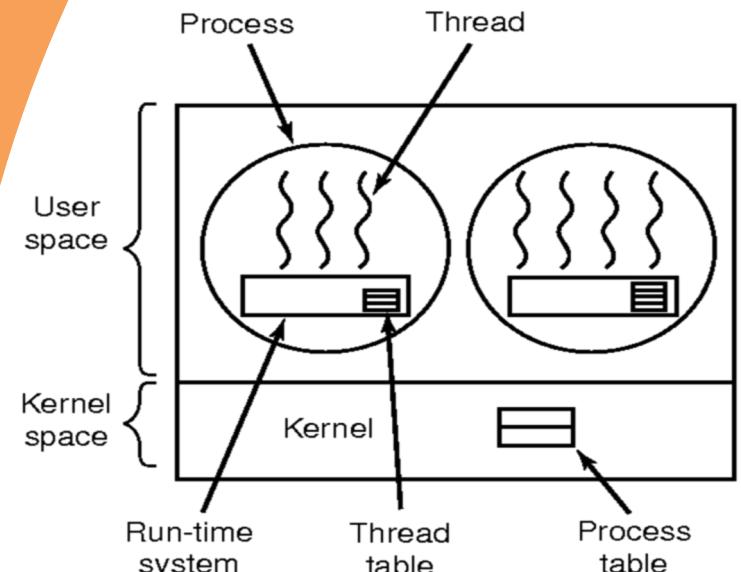


# Programmazione multicore

- Progettare programmi che sfruttino le moderne architetture multicore non è banale;
- **principi base:**
  - **separazione dei task;**
    - trovare aree per attività separate e simultanee
  - **bilanciamento;**
    - assicurarsi che eseguano lo stesso lavoro di uguale valore
  - **suddivisione dei dati;**
    - dividere i dati per essere eseguiti su core separati
  - **dipendenze dei dati;**
    - esecuzione delle attività sia sincronizzata
  - **test e debugging**

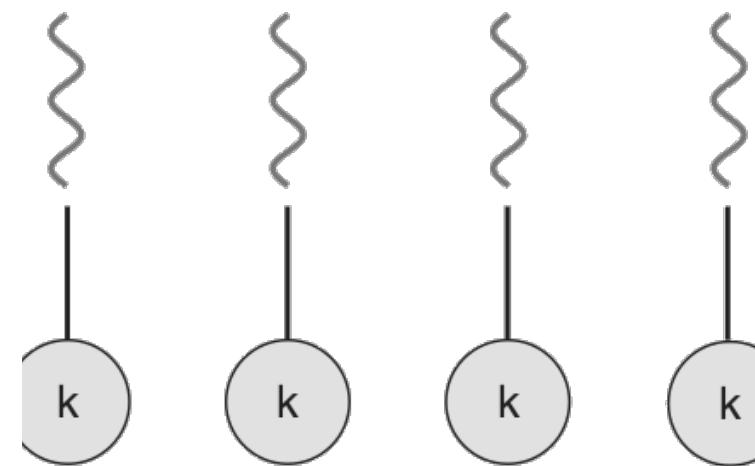
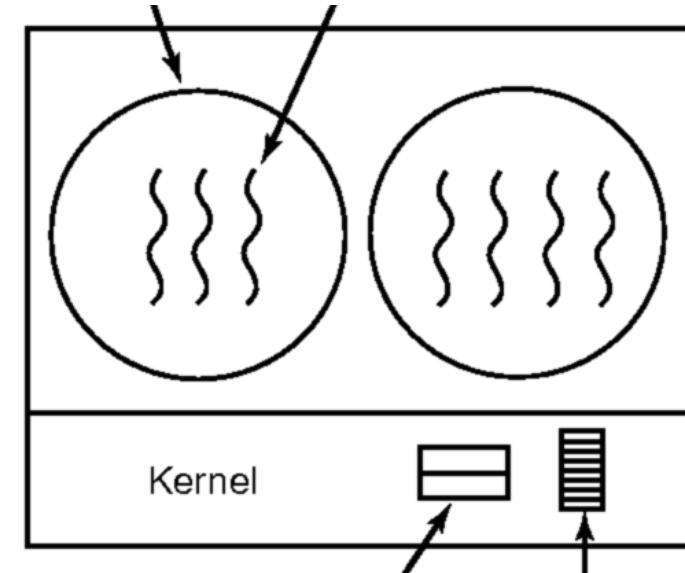
# Thread a livello utente

- Detto anche "**modello 1-a-molti**";
- utile se non c'è supporto da parte del kernel ai thread;
- una **libreria** che implementa un **sistema run-time** che gestisce una **tabella dei thread** del processo.
- **Pro:**
  - il dispatching non richiede trap nel kernel;
  - scheduling personalizzato;
- **Contro:**
  - chiamate bloccanti (select, page-fault);
  - possibilità di non rilascio della CPU.



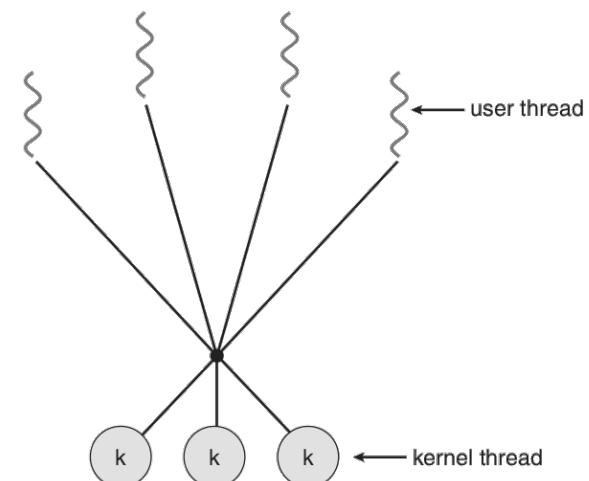
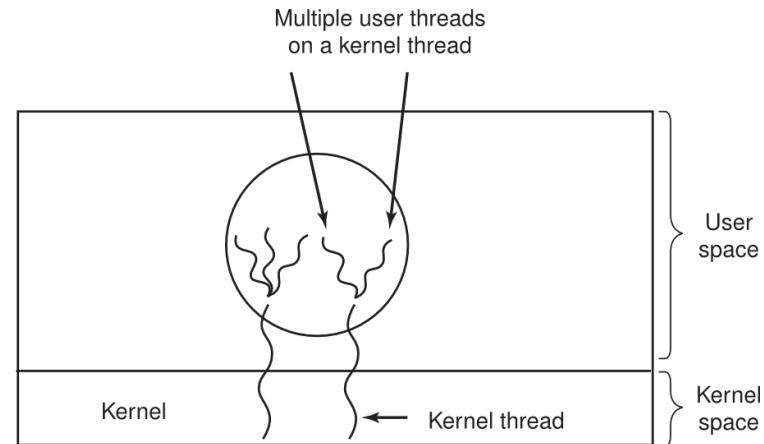
# || Thread a livello kernel

- Detto anche "**modello 1-a-1**";
- richiede il supporto specifico dal kernel (praticamente tutti i moderni SO);
- **unica tabella dei thread** del kernel;
- **Pro:**
  - un thread su chiamata bloccante non intralcia gli altri;
- **Contro:**
  - cambio di contesto più lento (richiede trap);
  - creazione e distruzione più costose (numero di thread kernel tipicamente limitato, possibile riciclo).



# Modello ibrido

- Detto anche "**molti-a-molti**";
- prende il meglio degli altri due;
- prevede un certo numero di **thread del kernel**;
- ognuno di essi viene assegnato ad un certo numero di **thread utente** (eventualmente uno);
- **assegnazione decisa** dal programmatore.



# I thread nei nostri sistemi operativi

- Quasi tutti i sistemi operativi supportano i **thread a livello kernel**;
  - Windows, Linux, Solaris, Mac OS X,...
- Supporto ai **thread utente** attraverso apposite librerie:
  - *green threads* su Solaris;
  - *GNU portable thread* su UNIX;
  - *fiber* su Win32.
- **Librerie di accesso ai thread** (a prescindere dal modello):
  - *Pthreads* di POSIX (Solaris, Linux, Mac OS X, anche Windows);
    - una specifica da implementare sui vari sistemi;
  - threads Win32;
  - thread in Java;
    - wrapper sulle API sottostanti.