POLITECNICO DI MILANO





Thread

Laboratorio Software 2008-2009
C. Brandolese

Introduzione

Un processo viene diviso in due componenti:

- Risorse allocate al processo
- Contesto di esecuzione

Un contesto di esecuzione

- È associato ad un thread

Un processo

- Deve avere almeno un thread
- Può avere più di un thread

I thread

- Condividono tutte le risorse del processo a cui appartengono
- Hanno stati di esecuzione simili ai processi

Introduzione

Thread

- Anche detti Light-Weight Process o LWP
- Condivide lo spazio di indirizzamento e altre informazioni di natura globale con il processo cui appartiene
- Registri, stack, maschere dei segnali ed altre informazioni specifiche sono locali ad ogni thread

I threads possono essere gestiti

- Dal sistema operativo
- Dall'utente

Esempi

- Win32 threads
- C-threads
- Pthreads

Informazioni locali e globali

Informazioni relative al processo

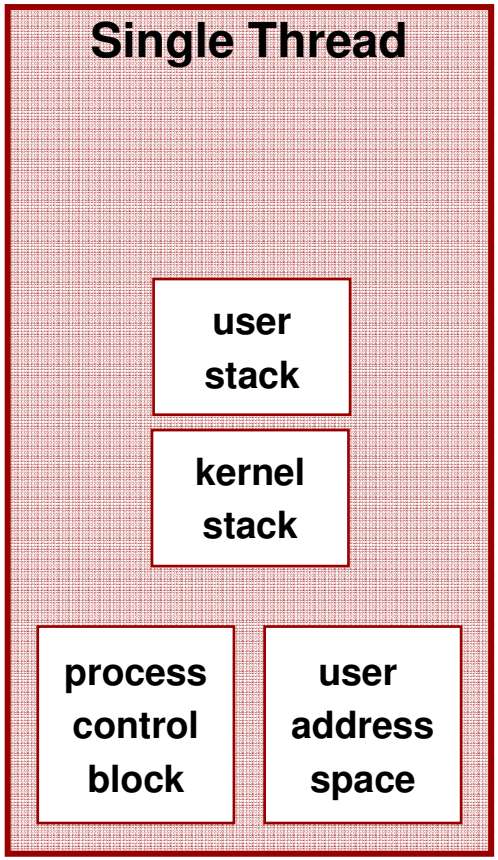
- Spazio degli indirizzi
- Variabili globali
- File aperti
- Processi figli
- Timer
- Segnali
- Semafori
- Informazioni di accounting

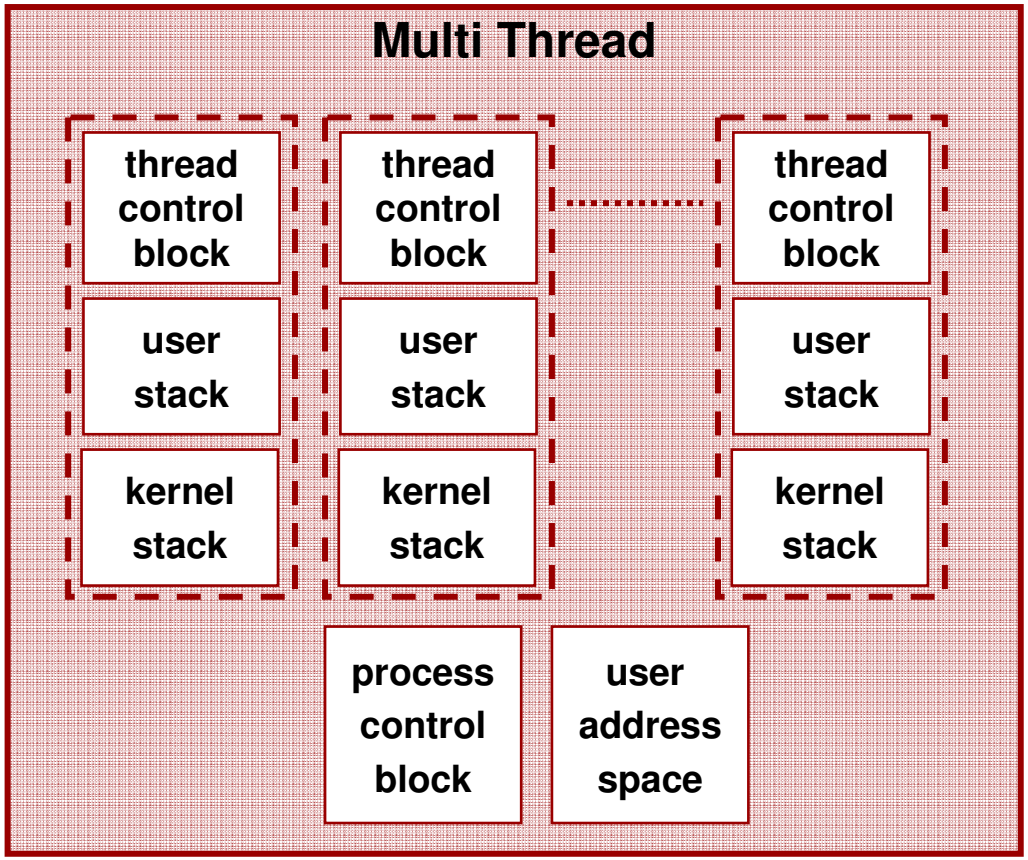
Informazioni relative al thread

- Program counter
- Stack
- Insieme dei registri
- Thread figli
- Stato
- Maschere dei segnali

Intrduzione

Modelli single-thread e multi-thread





Vantaggi del modello a trhaed

Il modello a thread è divenuto dominante per varie ragioni

Sviluppo del software

- Applicazioni complesse possono essere espresse
  - Mediante thread paralleli
  - In modo molto naturale
- Parallelismo a grana più fine

Prestazioni

- Il modello scala meglio verso sistemi multiprocessore
- Lo spazio di memoria condiviso riduce l'overhead per IPC

Gestione

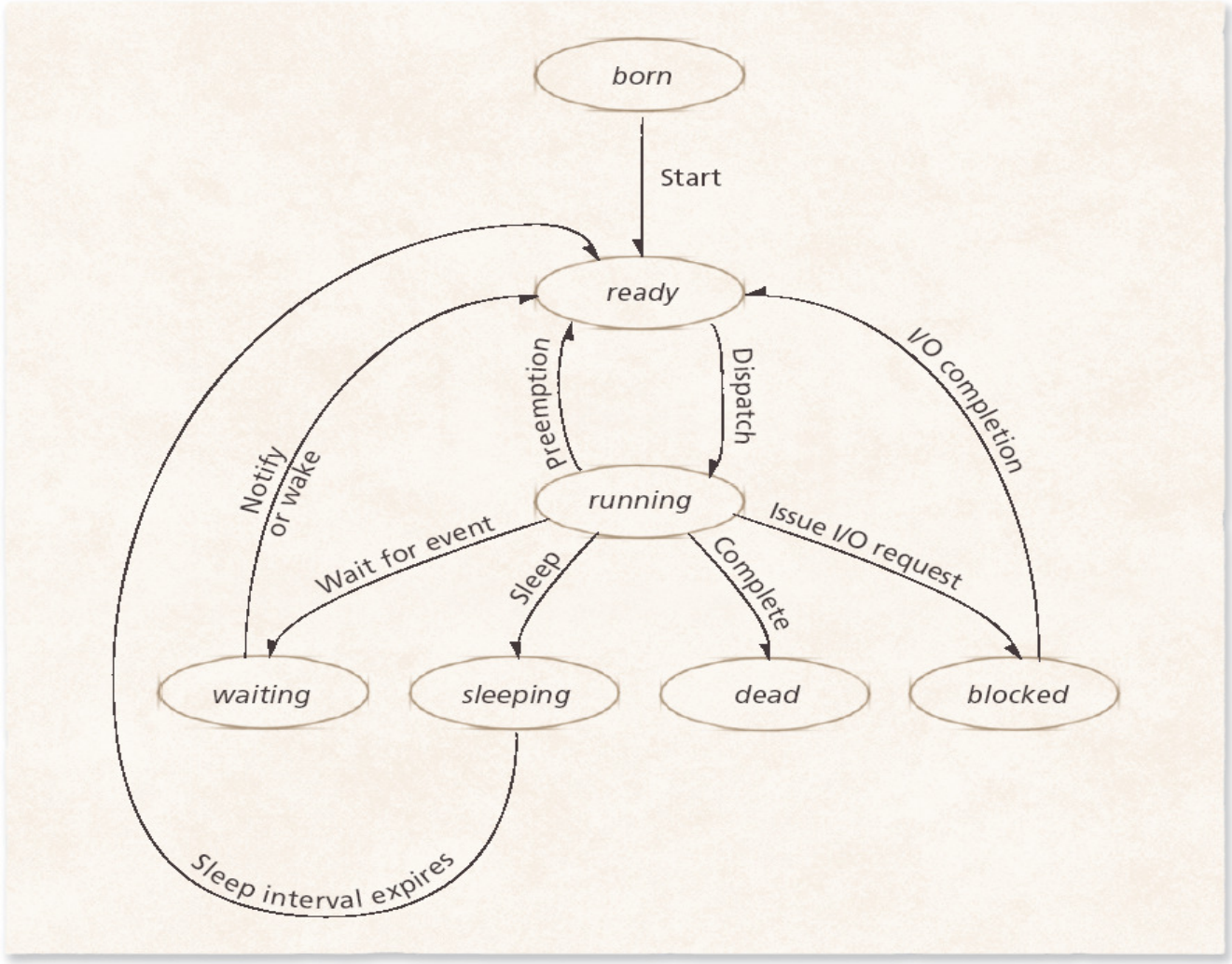
- Creazione e distruzione molto veloci
  - Il sistema operativo non deve allocare le risorse condivise con il processo
- Cambio di contesto molto veloce

Stati

Come un processo, anche un thread ha uno stato

- Born
  - Appena il thread viene creato
- Ready
  - Pronto per l'esecuzione
- Running
  - In esecuzione
- Dead
  - Terminato
- Blocked
  - Bloccato in attesa di I/O
- Waiting
  - Bloccato in attesa di un evento
- Sleeping
  - Sospeso per un certo intervallo di tempo

Stati e transizioni



Operazioni

Operazioni comuni a thread e processi

- Creazione - create
- Uscita - exit
- Sospensione - suspend
- Ripresa - resume
- Attesa - sleep
- Risveglio - wake

Operazioni specifiche dei thread

- Eliminazione - cancel
  - Richiede la terminazione di un thread
  - Non garantisce che ciò avvenga poiché i thread possono mascherare il segnale di cancellazione
- Join
  - Un thread si pone in attesa della terminazione di altri thread
  - Il thread che invoca il joining si sospende

Modelli di threading

Si possono avere tre modelli di threading

User-level thread

- Creati in user space
- Gestiti mediante librerie
- Non usano istruzioni privilegiate o accedere al kernel direttamente
- Un contesto di esecutizone unico per tutti i thread

Kernel-level thread

- Contesto di esecuzione singolo per ogni thread

Combinazione di user-level e kernel-level thread

- Cerca di superare le limitazioni dei due modelli
- Mapping generico tra thread e contesti di esecuzione

User-level thread

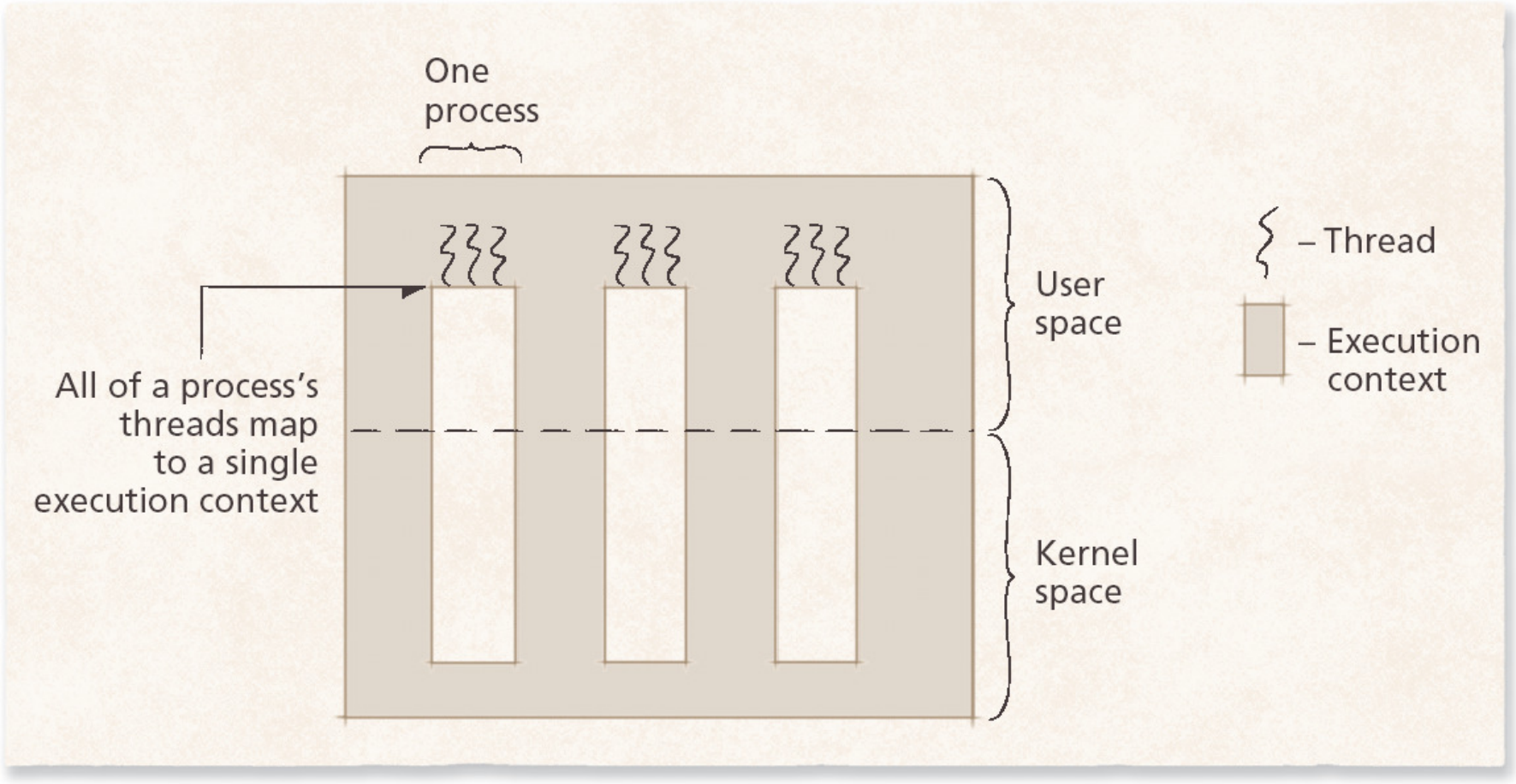
Mapping molti-a-uno

- Il sistema operativo mappa tutti i thread di un processo mutithreaded in un singolo contesto di esecuzione
- Vantaggi
  - Mediante le librerie utente si può gestire esplicitamente lo scheduling dei vari thread per ottimizzare le prestazioni
  - La sincronizzazione avviene al di fuori del kernel e pertanto non richiede contex switch
  - Maggiore portabilità
- Svantaggi
  - Il kernel vede un processo multithreaded come una singola entità
  - Può ridurre le prestazioni
  - Non può essere schedulanto su più processori

Implementazioni

- POSIX Pthreads, MAC C-Threads, Solaris Threads, …

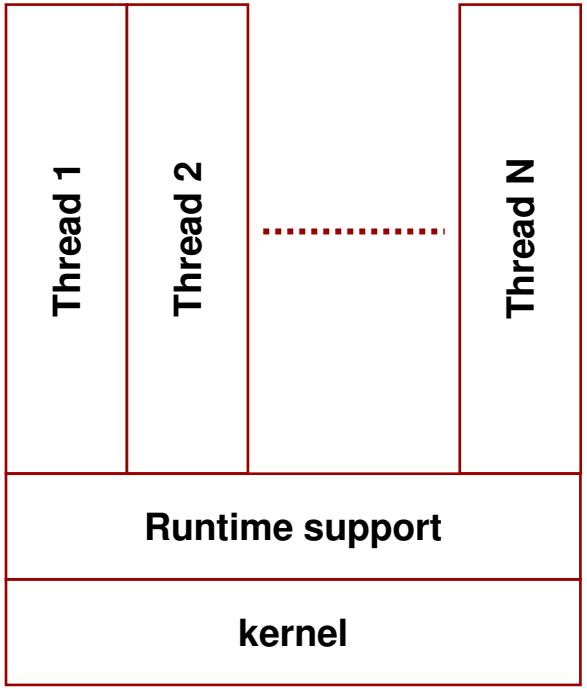
User-level thread



User-level thread

Package

- Fornisce uno strato di supporto runtime
  - Al di sopra il kernel
  - Si occupa dello scheduling dei thread attivi di un processo
- Vantaggi
  - Permette di aggiungere i thread ad un sistema operativo che non li preveda
  - Permette di avere un algoritmo di scheduling personalizzato
- Svantaggi
  - Un thread rimane attivo fino a che non si sospende volontariamente
  - Difficile o impossibile sfruttamento multiprocessing
  - Il blocco di un thread può bloccare l'intero processo



Kernel-level thread

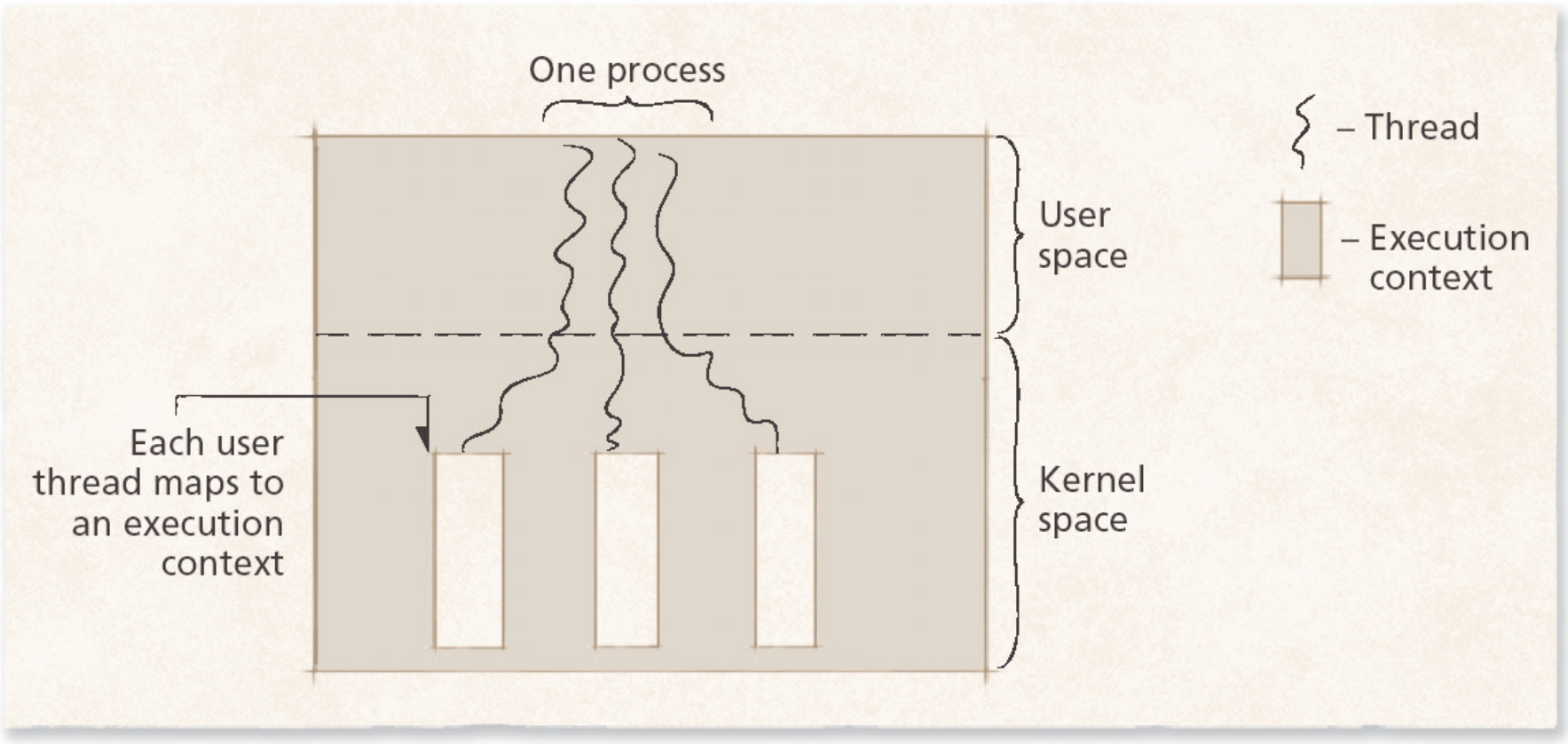
Mapping uno-a-uno

- Ogni thread ha un contesto di esecuzione specifico in kernel space
- Ogni processo in user-space è composto da più thread
  - Non sempre una buona soluzione per applicazioni multithreaded
- Vantaggi
  - Aumento della scalabilità
  - Miglioramento dell'interattività
  - Spesso si ha anche un miglioramento del throughput
- Svantaggi
  - Maggiore overhead docuto al cambio di contesto
  - Riduzione della portabilità a causa di API specifiche dei vari sistemi opertivi

Implementazioni

- Win 95/98/NT, Solaris, Digital UNIX, …

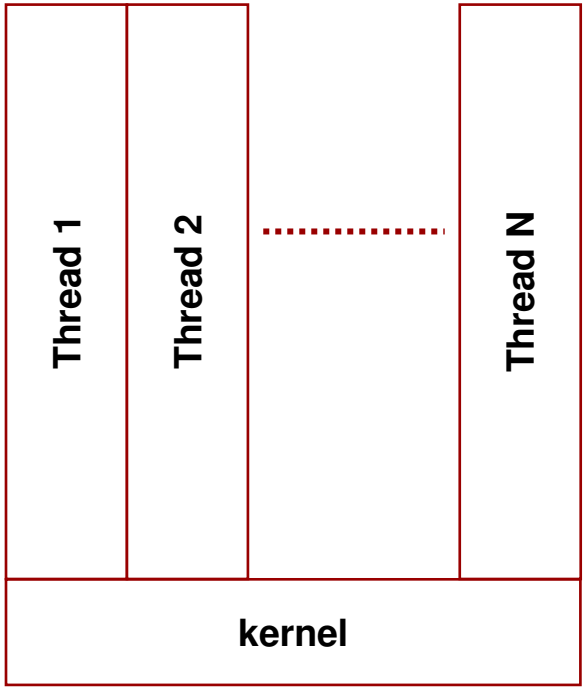
Kernel-level thread



Kernel-level thread

Package

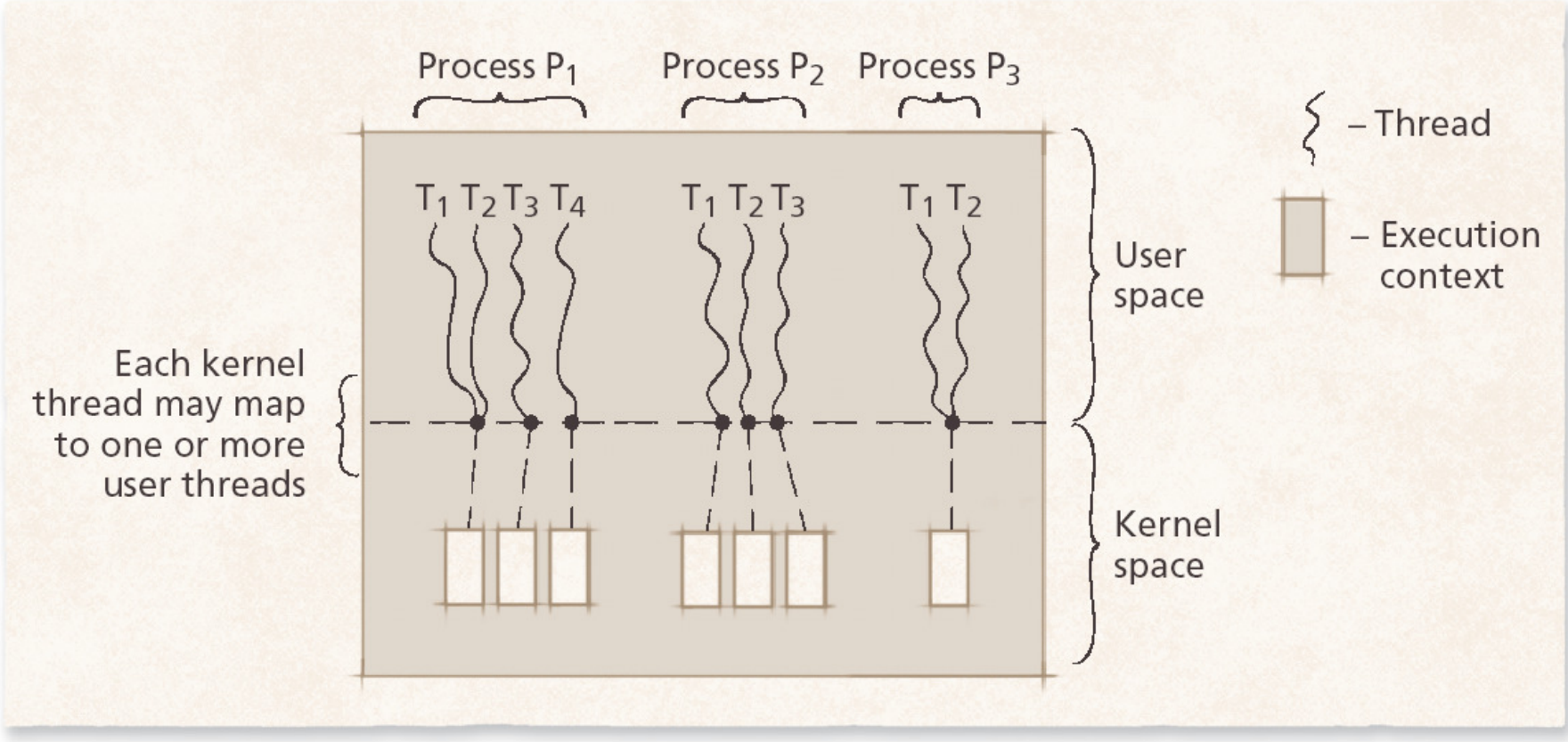
- Thread gestiti direttamente dal kernel
  - Assegna a ciascun thread un quanto di tempo
  - Alla fine seleziona un altro thread da eseguire
- Vantaggi
  - Su una chiamata di sistema bloccante, viene sospeso solo un thread e non tutto il processo a cui appartiene
- Svantaggi
  - La gestione dei thread va effettuata con chiamate di sistema
  - Si ha maggiore overhead rispetto a user thread package



User/Kernel-level thread

Mapping molti-a-molti

- Si tratta di una mediazione tra di due modelli visti
- Ha l'obiettivo di superare le limitazioni dei due modelli



Organizzazione – Team model

Utile per partizionare un'applicazione

- Si individuano alcune attività da separare
- Si crea un thread per ogni attività individuata

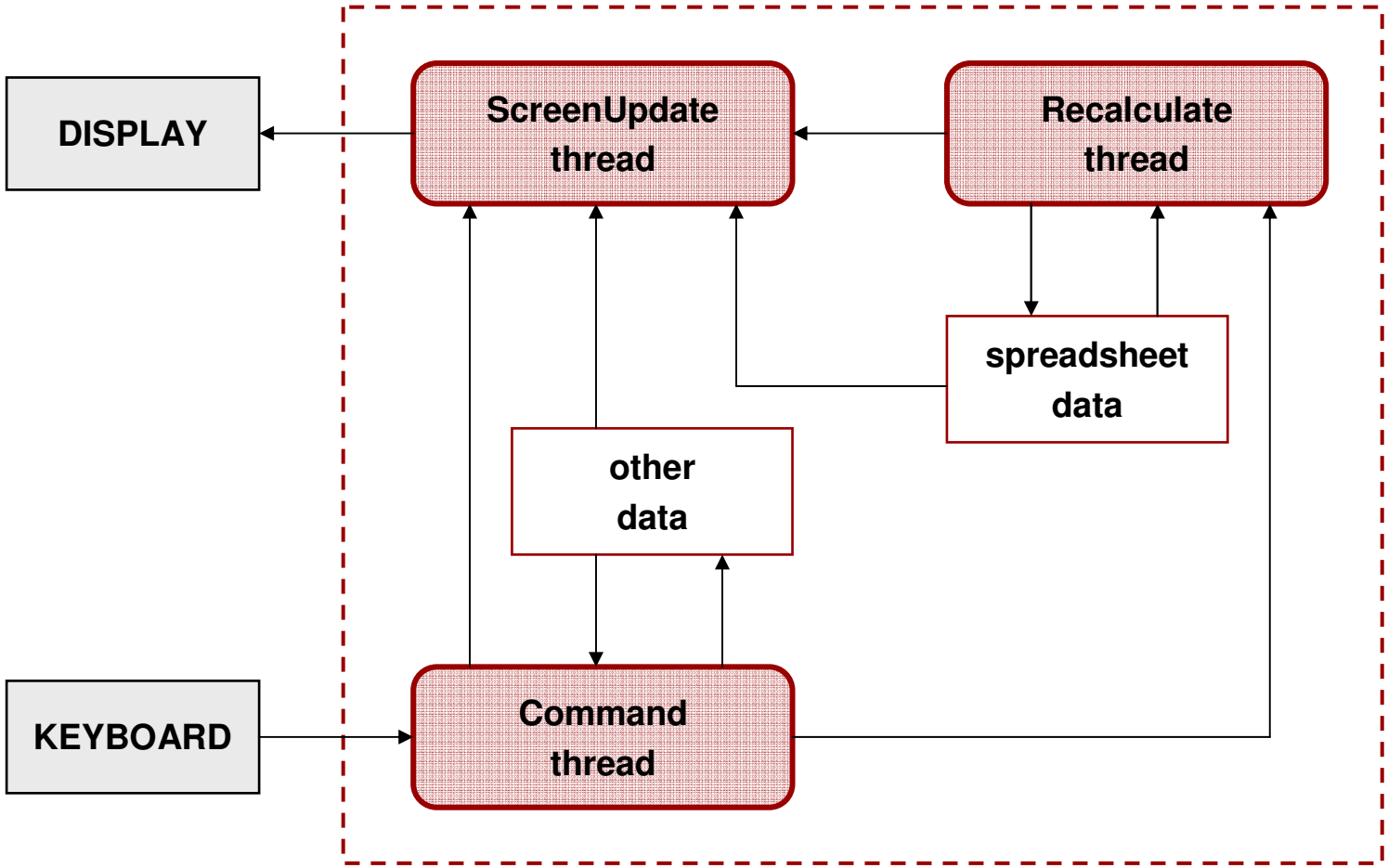
Vantaggi

- Maggior reattività agli input
- Semplificazione del processo di implementazione
  - Sviluppo effettuato da un gruppo di programmatori

Esempio

- Gestione di un foglio elettronico

Organizzazione – Team model



Organizzazione – Dispatcher model

Esiste un thread particolare

- Detto dispatcher
- Riceve richieste di servizio e le invia ai thread che le eseguono
- Simile al team model

Thread

- Uno per ogni richiesta di servizio
- Eventualmente si prevede un numero massimo

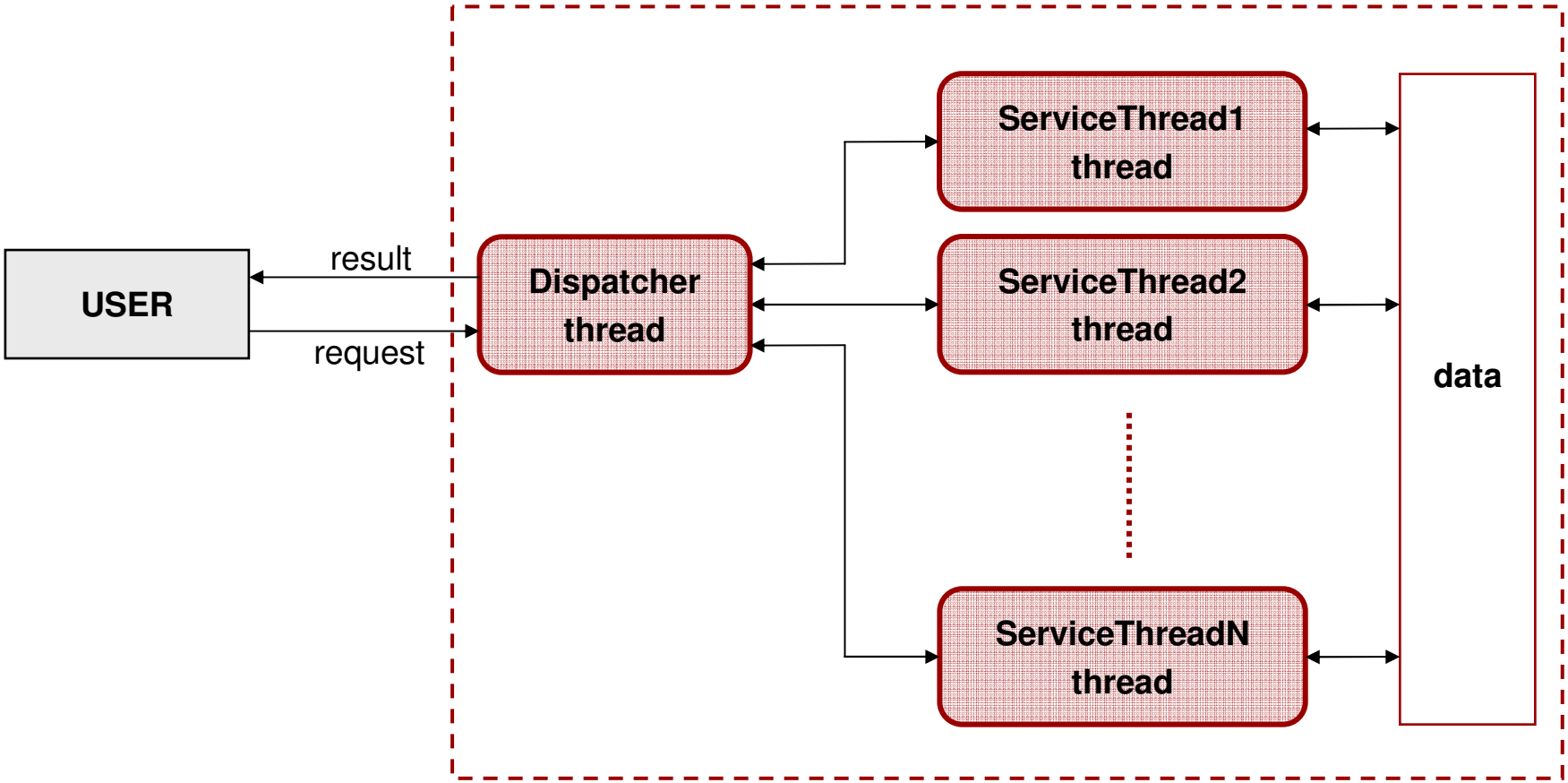
Vantaggi

- Utile per replicare attività

Esempio

- File server

Organizzazione – Dispatcher model



Organizzazione – Pipeline model

Schema semplice

- Partizionamento delle attività su base temporale
- Utile per gestire una catena di algoritmi
  - Ogni algoritmo ha in ingresso l'uscita di un altro algoritmo

Vantaggi

- Sistema maggiormente asincrono e più efficiente
- Se un thread deve attendere un evento (es. I/O da disco)
  - Gli altri thread possono proseguire la computazion

Esempio

- Script di shell di UNIX

Organizzazione – Pipeline model