

# Thread

Sistemi Operativi

Antonino Staiano

Email: [antonino.staiano@uniparthenope.it](mailto:antonino.staiano@uniparthenope.it)

# Perché i Thread

- I processi concorrenti velocizzano l'esecuzione delle applicazioni, ma
  - ...
  - I context switch introducono un elevato overhead
- Overhead
  - Esecuzione
    - Salvataggio dello stato della CPU del processo in esecuzione
    - Caricamento dello stato della CPU del nuovo processo
  - Uso risorse
    - Commutazione del contesto del processo
      - Informazioni sulle risorse allocate al processo
      - Informazione sull'interazione con altri processi

# Perché i Thread (cont.)

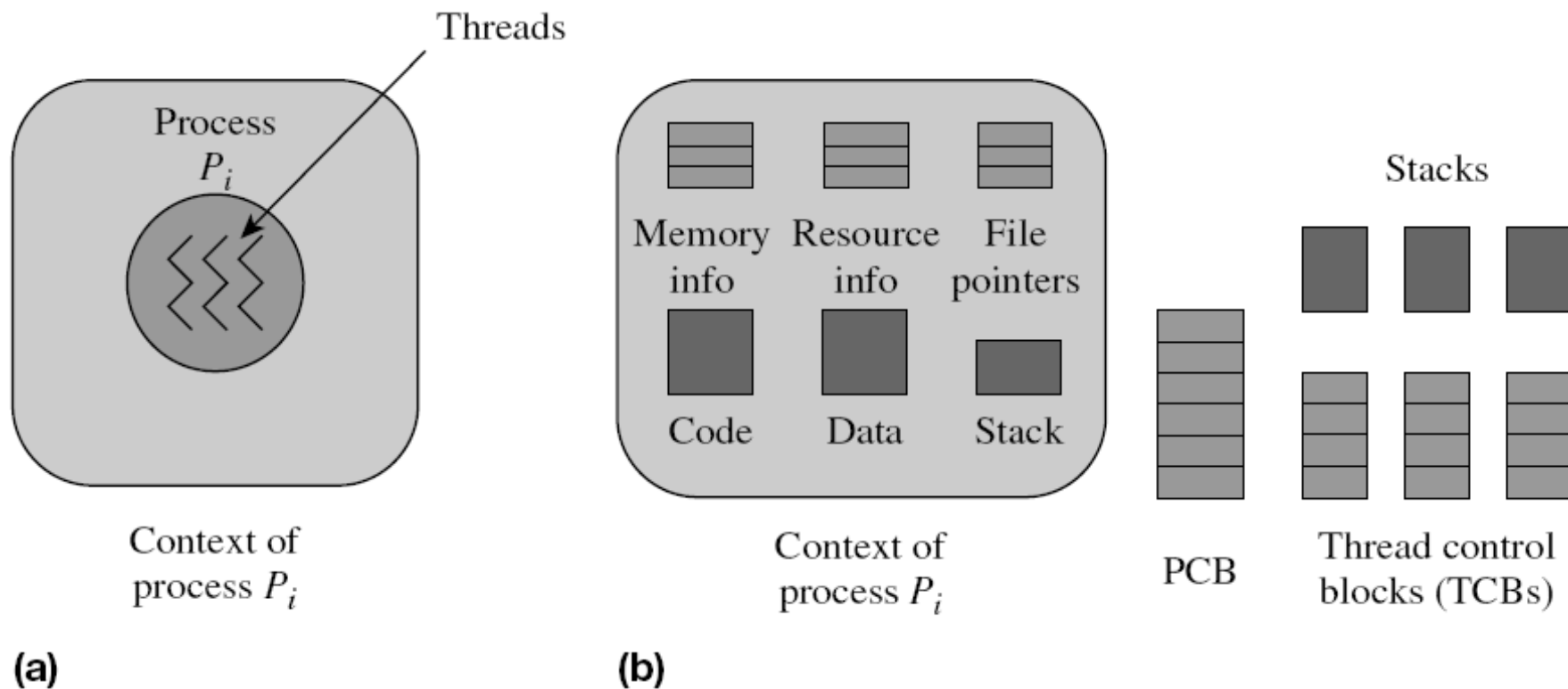
- Supponiamo di avere un processo  $P$  con due processi figli  $P_i$  e  $P_j$ 
  - $P_i$  e  $P_j$  ereditano il contesto del processo  $P$
  - Se  $P_i$  e  $P_j$  non hanno allocato alcuna risorsa, il loro contesto è identico
  - Differiscono solo per stato di CPU e stack
- Il context switch tra  $P_i$  e  $P_j$  coinvolge molte informazioni ridondanti
- I thread sfruttano tale considerazione
  - Esecuzione di un programma che usa le risorse di un processo
- I thread suddividono lo stato del processo in due parti
  - Stato delle risorse, associato al processo
  - Stato dell'esecuzione, associato ad ogni thread
- Solo gli stati di esecuzione devono essere scambiati nella commutazione tra thread
- Lo stato delle risorse è condiviso

# Thread

**Thread:** *esecuzione di un programma che usa le risorse di un processo*

- Un thread è un modello alternativo di esecuzione di un programma
- Un processo crea un thread attraverso una chiamata di sistema
- Il thread lavora all'interno nel contesto del processo
- L'uso di thread suddivide di fatto lo stato di un processo in due parti
  - Lo stato della risorsa resta con il processo
  - Lo stato della CPU è associato con il thread
- La commutazione tra thread comporta un minor overhead rispetto alla commutazione tra processi

## Thread (cont.)



**Thread nel processo  $P_i$ : (a) concetto; (b) implementazione.**

# Thread Control Block

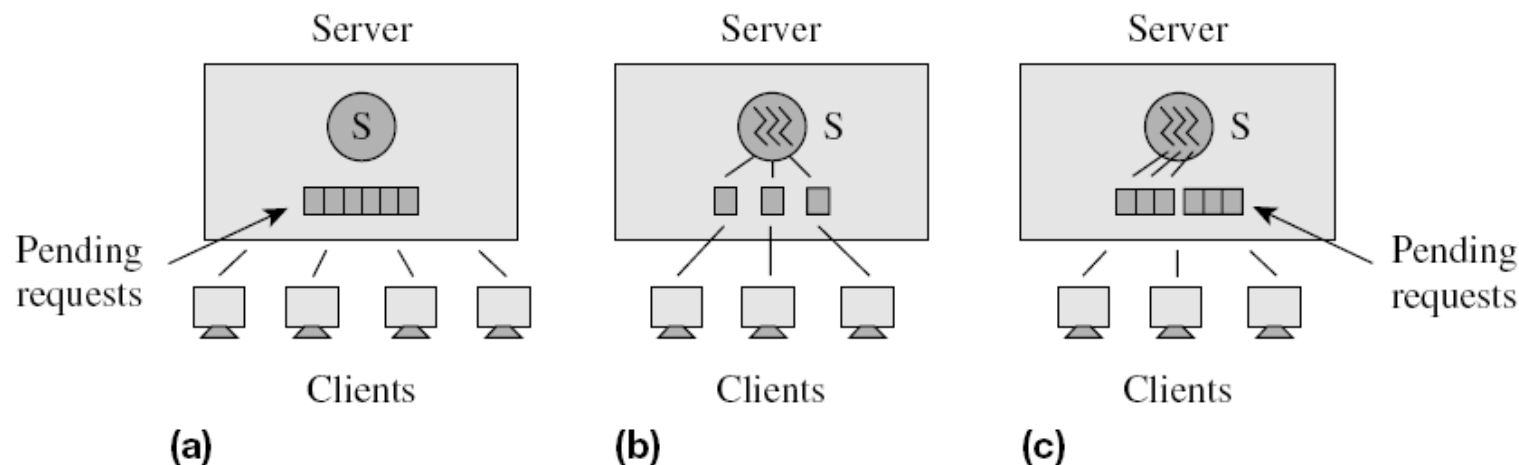
- Informazione per la schedulazione dei thread: *thread id, priorità, stato*
- Stato della CPU: PSW e GPR
- Puntatore al PCB del processo all'interno del quale è stato creato
- Puntatore al TCB per creare la lista necessaria alla schedulazione

# Stato di un Thread e Transizioni di Stato

- I Thread hanno stato e transizioni di stato dei processi
- Un thread appena creato è impostato allo stato *ready*
  - Il processo all'interno del quale si trova ha già le risorse necessarie
- Il Thread transisce nello stato *running* quando è sottoposto al dispatching
- Entra nello stato *blocked* a causa dei requisiti di sincronizzazione del processo

# Vantaggi Thread vs Processi

Advantage	Explanation
Lower overhead of creation and switching	Thread state consists only of the state of a computation. Resource allocation state and communication state are not a part of the thread state, so creation of threads and switching between them incurs a lower overhead.
More efficient communication	Threads of a process can communicate with one another through shared data, thus avoiding the overhead of system calls for communication.
Simplification of design	Use of threads can simplify design and coding of applications that service requests concurrently.



Uso dei thread per strutturare un server: (a) server che usa codice sequenziale; (b) server multi-thread; (c) server che usa un pool di thread



# Codifica per Usare i Thread

- Usare librerie thread safe per assicurare la correttezza della condivisione dei dati
- Gestione dei segnali: quale thread dovrebbe gestire un segnale?
  - La scelta può essere fatta dal kernel o dall'applicazione
    - Un segnale connesso ad un'eccezione hardware è gestito dal thread stesso
    - Gli altri segnali sono inviati ad un qualsiasi thread del processo
      - Idealmente il thread a priorità maggiore dovrebbe occuparsene

# I Thread in C: Thread Posix

- Lo standard ANSI/IEEE Portable Operating System Interface (POSIX) definisce l'API ***pthread***
  - Per l'utilizzo nei programmi C
  - Fornisce 60 routine che eseguono quanto segue
    - Gestione dei thread
    - Assistenza per la condivisione dei dati – mutua esclusione
    - Assistenza per la sincronizzazione – variabili di condizione
  - Un pthread è creato attraverso la chiamata  
`pthread_create(< data structure >, < attributes >, < start routine >, < arguments >)`
  - La sincronizzazione genitore-figlio avviene attraverso `pthread_join`
  - Un thread termina invocando `pthread_exit`

# Esempio di codice con pthread

```
#include <pthread.h>
#include <stdio.h>
int size, buffer[100], no_of_samples_in_buffer;
int main()
{
    pthread_t id1, id2, id3;
    pthread_mutex_t buf_mutex, condition_mutex;
    pthread_cond_t buf_full, buf_empty;
    pthread_create(&id1, NULL, move_to_buffer, NULL);
    pthread_create(&id2, NULL, write_into_file, NULL);
    pthread_create(&id3, NULL, analysis, NULL);
    pthread_join(id1, NULL);
    pthread_join(id2, NULL);
    pthread_join(id3, NULL);
    pthread_exit(0);
}

void *move_to_buffer()
{
    /* Repeat until all samples are received */
    /* If no space in buffer, wait on buf_full */
    /* Use buf_mutex to access the buffer, increment no. of samples */
    /* Signal buf_empty */
    pthread_exit(0);
}

void *write_into_file()
{
    /* Repeat until all samples are written into the file */
    /* If no data in buffer, wait on buf_empty */
    /* Use buf_mutex to access the buffer, decrement no. of samples */
    /* Signal buf_full */
    pthread_exit(0);
}

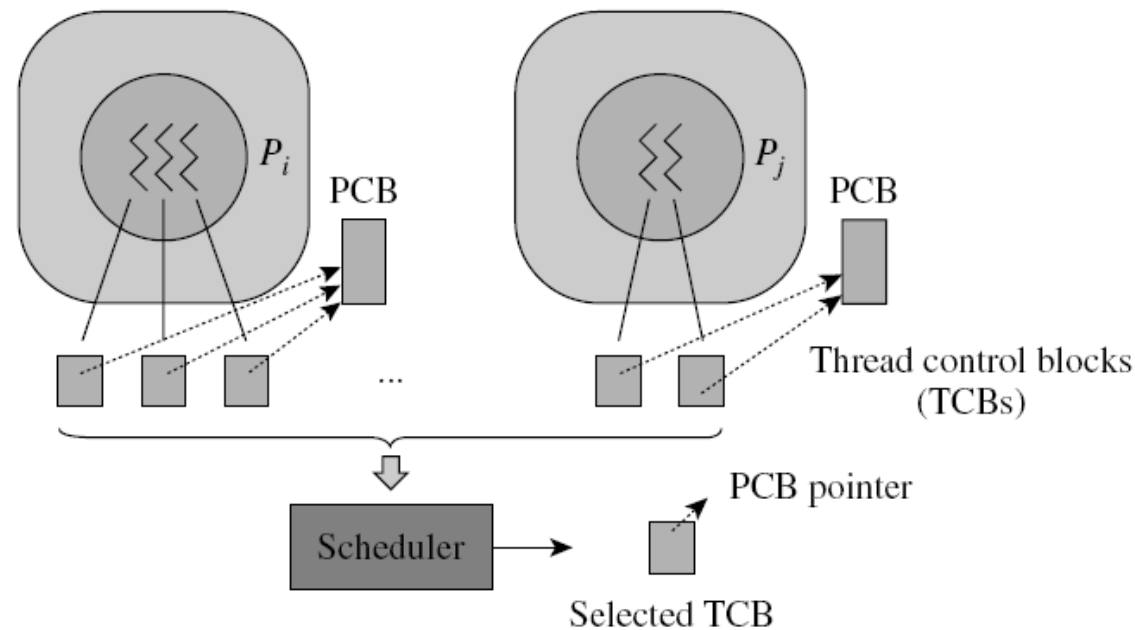
void *analysis()
{
    /* Repeat until all samples are analyzed */
    /* Read a sample from the buffer and analyze it */
    pthread_exit(0);
}
```

# Thread di livello kernel, utente e ibridi

- Thread di livello kernel
  - I thread sono gestiti dal kernel
- Thread di livello utente
  - I thread sono gestiti dalla libreria dei thread
- Thread ibridi
  - Combinazione di thread di livello kernel ed utente

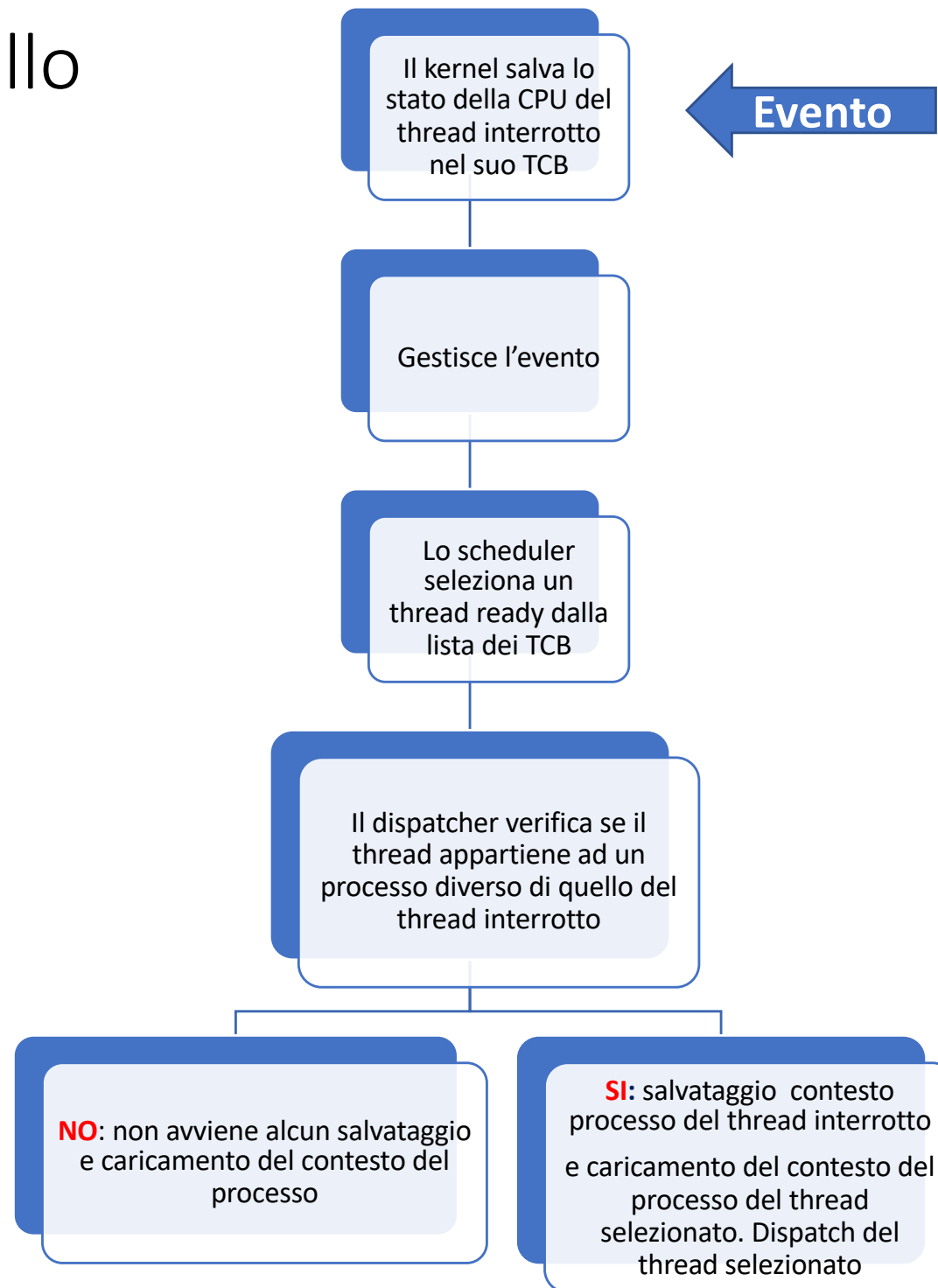
# Thread di livello kernel

- Un thread di livello kernel è come un processo con una minore quantità di informazioni di stato
- La commutazione tra thread dello stesso processo incorre in overhead per la gestione dell'evento



Scheduling di thread di livello kernel

# Thread di Livello Kernel



# Thread di Livello Kernel: Pro e Contro

- **Pro**

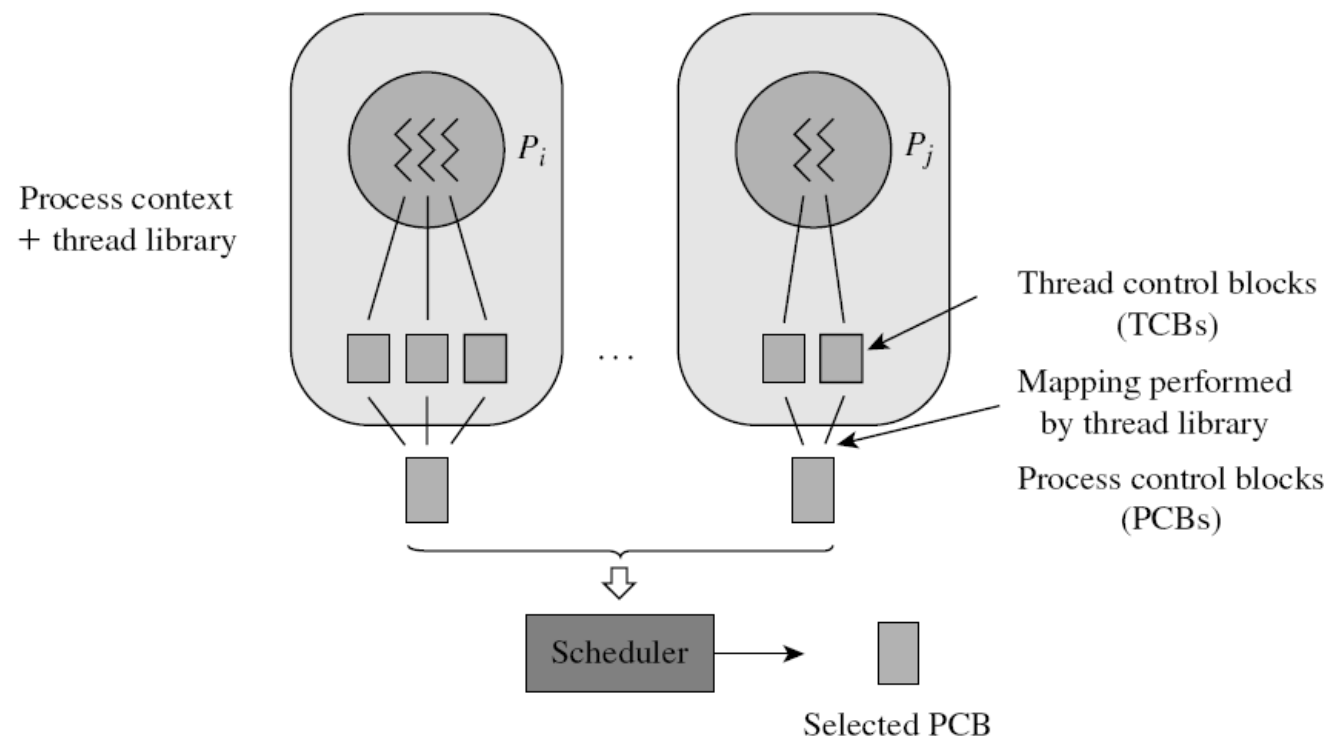
- Thread simili ai processi (con meno informazioni) -> conveniente per i programmatori
- In ambienti multi CPU consentono il parallelismo
  - Più thread di un processo schedulati contemporaneamente

- **Contro**

- La commutazione tra thread è fatta dal kernel quando è gestito un evento
  - Overhead di gestione evento anche se il thread interrotto ed il thread schedulato sono dello stesso processo
  - Limita il risparmio di overhead nella commutazione tra thread

# Thread di Livello Utente

- Commutazione veloce dei thread poiché non è coinvolto il kernel
- Il blocco di un thread blocca tutti i thread del processo
- Thread di un processo: nessuna concorrenza o parallelismo

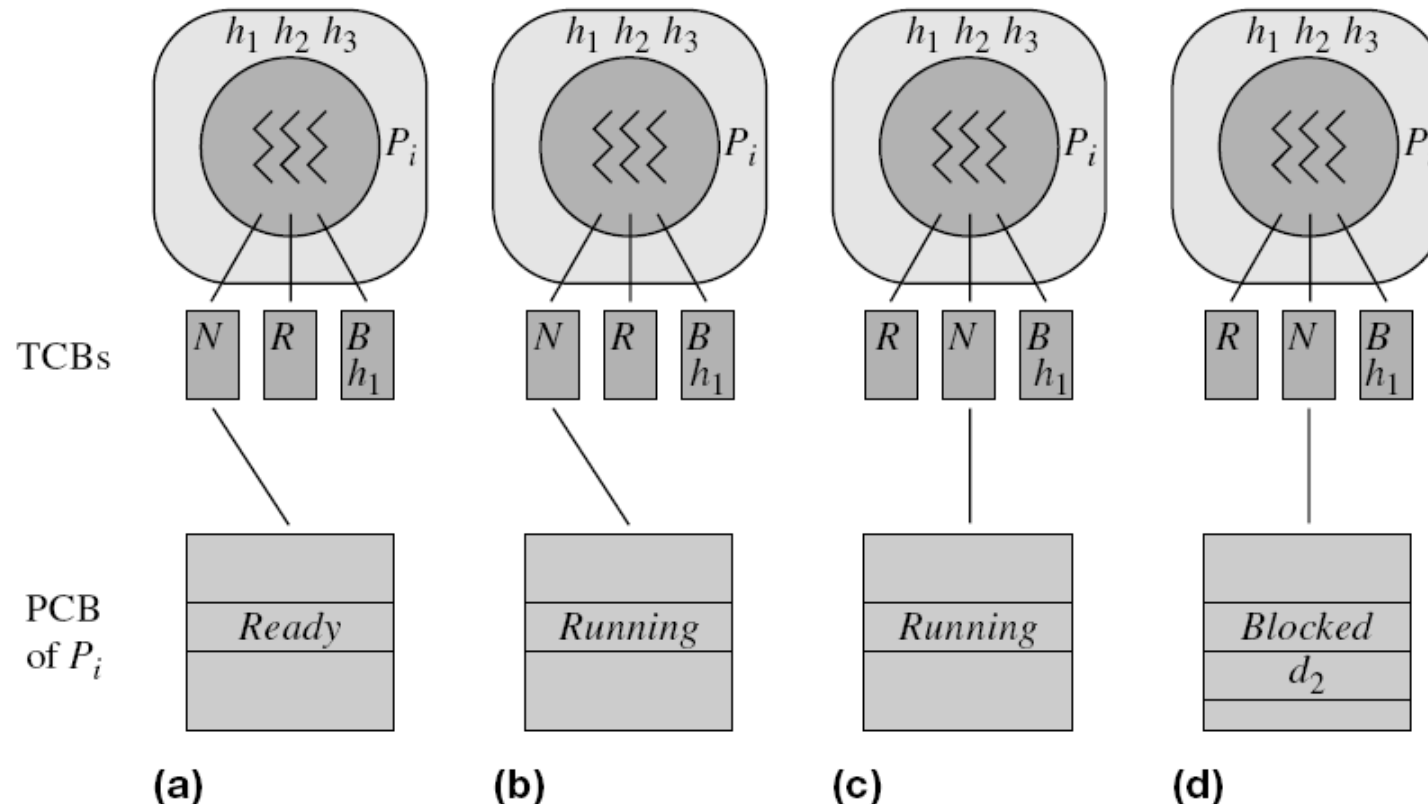


Scheduling di thread di livello utente



# Scheduling di thread di livello utente

- La libreria di thread mantiene lo stato del thread ed esegue la commutazione



Azioni della libreria di thread (**N**, **R**, **B** indicano running, ready e blocked)

# Thread di Livello utente: Pro e Contro

- **Pro**

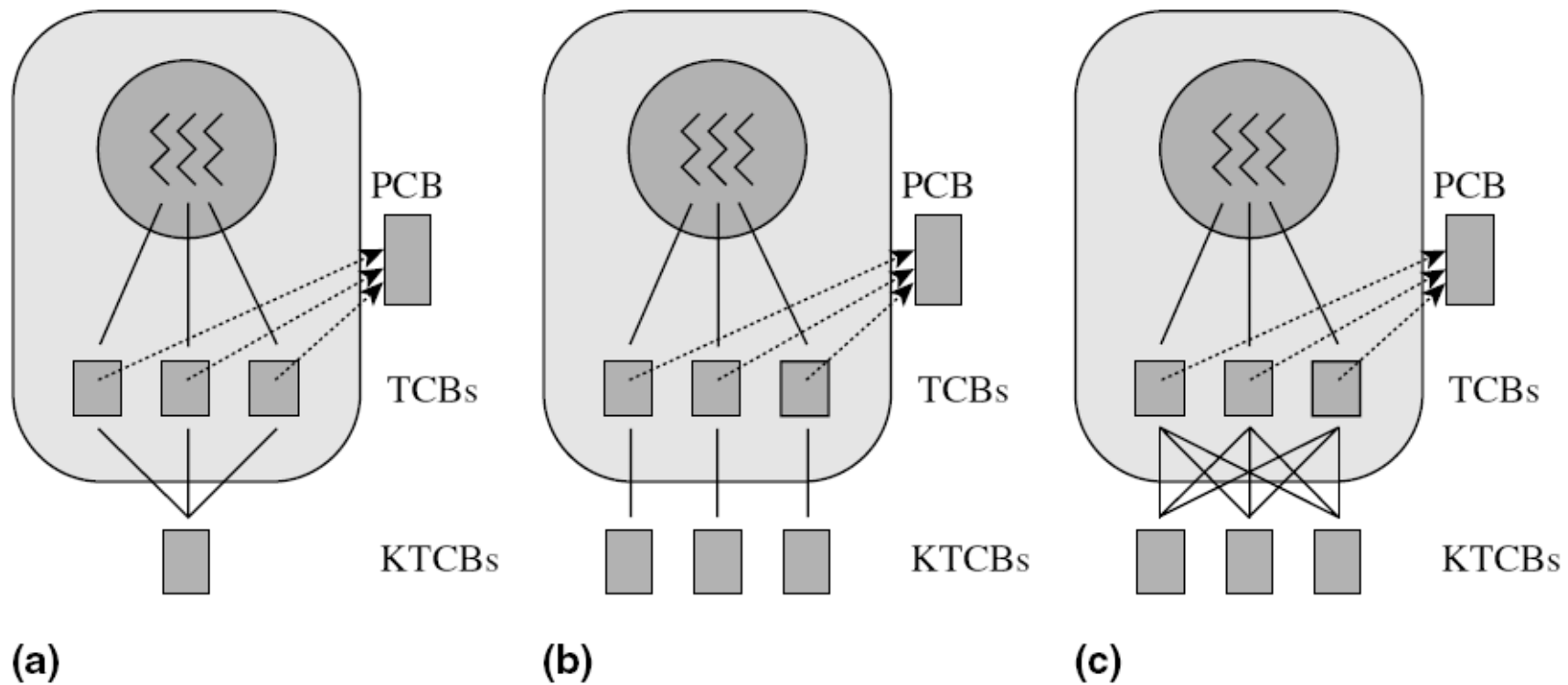
- **Schedulazione e sincronizzazione gestita dalla libreria che evita le system call per la sincro tra thread -> la commutazione tra thread meno onerosa rispetto ai thread di livello kernel**
  - **Consente anche ad un processo di scegliere una strategia di scheduling adatta alla sua natura**

- **Contro**

- **Il kernel non sa della distinzione tra thread e processo -> un thread bloccato su una system call fa sì che il kernel blocchi l'intero processo (tutti i thread del processo bloccati)**
- **Al più un thread alla volta può essere operativo**
  - **I thread di livello utente non forniscono parallelismo**
  - **La concorrenza è seriamente compromessa se un thread fa una system call che porta ad un blocco**

# Modello di Thread Ibridi

- Può fornire una combinazione di parallelismo e basso overhead



(a) Multi-a-uno; (b) uno-a-uno; (c) multi-a-molti

# Casi di Studio di Processi e Thread

- Processi in Unix
- Processi e thread in Linux
- Thread in Solaris
- Processi e thread in Windows

# Processi in Unix

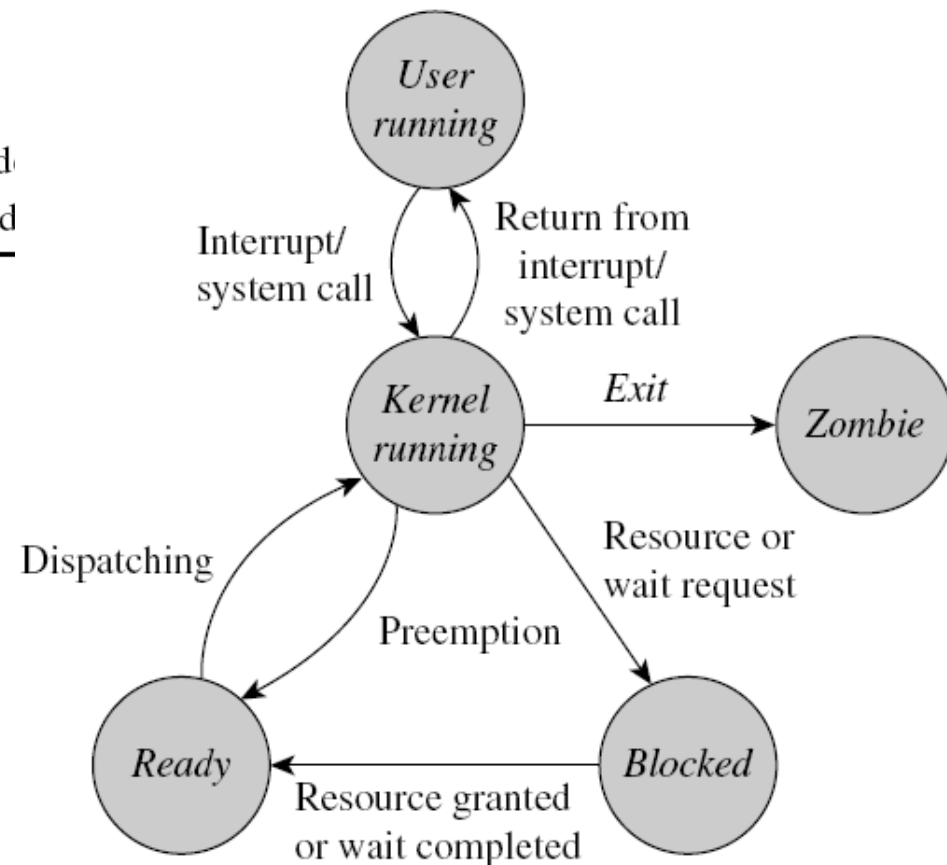
- Il processo esegue codice kernel in corrispondenza di un interrupt o di una chiamata di sistema, quindi ci sono uno stato di esecuzione kernel ed uno stato di esecuzione utente
- Un processo  $P_i$  può aspettare la terminazione di un processo figlio mediante la chiamata di sistema *wait*

```
main()
{
    int saved_status;
    for (i=0; i<3; i++)
    {
        if (fork()==0)
        { /* code for child processes */
            ...
            exit();
        }
    }
    while (wait(&saved_status) != -1);
    /* loop till all child processes terminate */
}
```

## Creazione terminazione di un processo in Unix

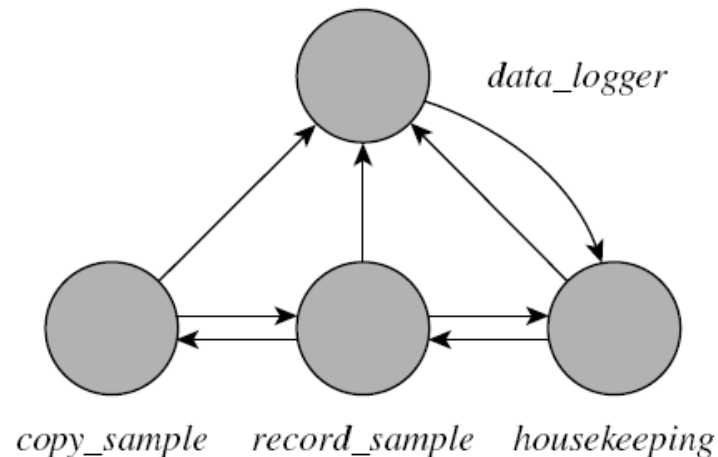
# Processi in Unix (cont.)

Signal	Description
SIGCHLD	Child process died or suspended
SIGFPE	Arithmetic fault
SIGILL	Illegal instruction
SIGINT	Tty interrupt (Control-C)
SIGKILL	Kill process
SIGSEGV	Segmentation fault
SIGSYS	Invalid system call
SIGXCPU	CPU time limit is exceeded
SIGXFSZ	File size limit is exceeded



# Processi e Thread in Linux

- Stati di processo: Task\_running, Task\_interruptible, Task\_uninterruptible, Task\_stopped e Task\_zombie
- Le informazioni sui processi genitori e figli o thread sono memorizzate in una `task_struct`



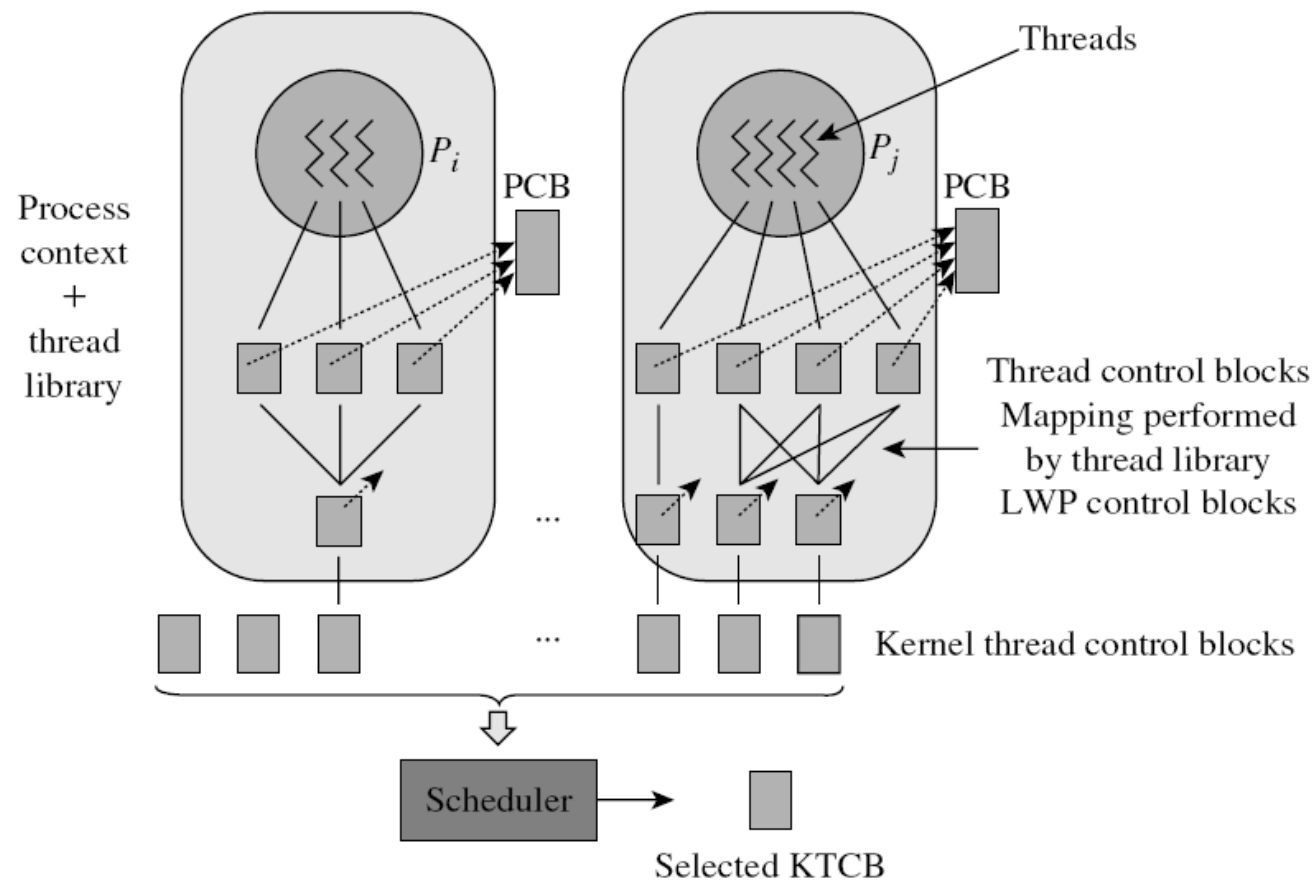
- Linux 2.6 supporta thread a livello kernel

# Thread in Solaris

- Tre tipi di entità governano la concorrenza ed il parallelismo in un processo:
  - Thread utente
  - Processi leggeri (lightweight – LWP)
    - Fornisce parallelismo in un processo
    - I thread utente sono mappati nei LWP
  - Thread kernel
- Ha supportato due differenti modelli di thread
  - Modello MxN fino a Solaris 8
  - Modello 1:1 da Solaris 8 in poi
- Fornisce le attivazione dello scheduler per evitare il blocco dei thread e per notificare gli eventi



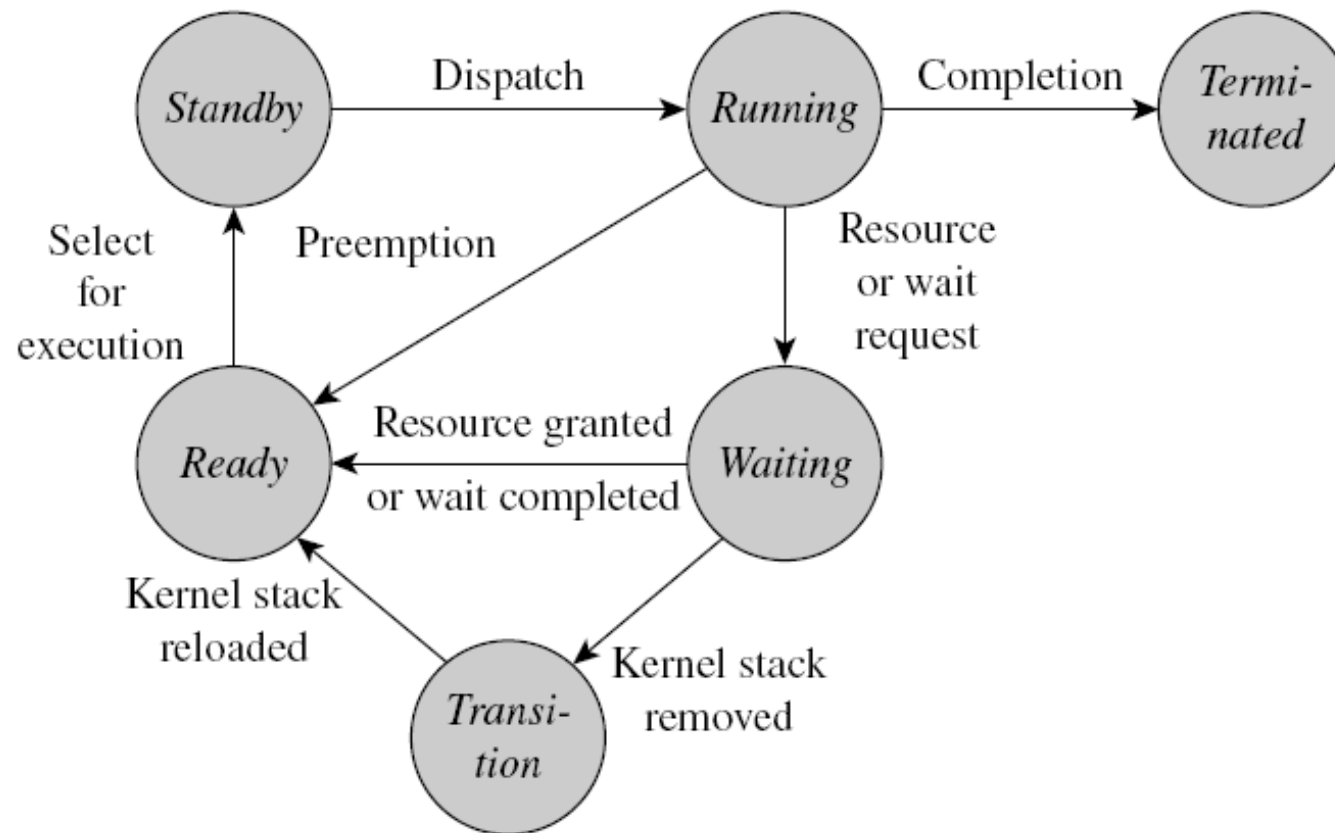
# Thread in Solaris (cont.)



# Processi e Thread in Windows

- Ogni processo contiene almeno un thread
- Usa tre blocchi di controllo per processo
  - Blocco di processo esecutivo: id del processo, un blocco di processo kernel e indirizzo blocco ambiente di processo
  - Blocco di processo kernel: stato del processo, indirizzi KTB
  - Blocco ambiente di processo: informazioni su codice e heap
- Usa tre blocchi di thread per thread
  - **Blocco di thread esecutivo** che contiene il puntatore al **blocco del thread kernel** e al blocco di processo esecutivo
  - Blocco del thread kernel: stack, stato e **blocco di ambiente kernel**

# Processi e Thread in Windows (cont.)



# Riepilogo

- L'esecuzione di un programma può essere velocizzata mediante parallelismo e/o concorrenza
- Un processo è un modello di esecuzione di un programma
  - Può creare altri processi facendo richieste al SO attraverso le chiamate di sistema
    - Ciascuno di tali processi è chiamato processo figlio
    - Fornisce parallelismo o concorrenza
- Il SO fornisce mezzi per la sincronizzazione dei processi
- Il SO alloca le risorse ad un processo e memorizza le informazioni su di esse nel contesto del processo

# Riepilogo (cont.)

- Per controllare il funzionamento del processo, il SO usa la nozione di stato del processo
  - Pronto, esecuzione, bloccato, terminato e sospeso
- Il SO mantiene le informazioni riguardanti ogni processo in un *process control block* (PCB)
  - Stati del processo e della CPU associati con il processo
  - Lo scheduler seleziona un processo pronto e il dispatcher commuta la CPU al processo selezionato mediante le informazioni trovate nel contesto del processo e nel PCB

# Riepilogo (cont.)

- Un thread è un modello di esecuzione alternativo per un programma
  - L'overhead della commutazione tra thread è molto minore dell'overhead nella commutazione tra processi
- Tre modelli di thread
  - Thread di livello kernel
  - Thread di livello utente
  - Thread ibridi
- I modelli di thread hanno diverse implicazioni per overhead di commutazione, concorrenza e parallelismo