

MODELLI DI SUPPORTO AL MULTI THREAD ➤ 4.3

I thread possono essere di

- livello utente: la gestione ricade ai gestori dell'utente.
- livello Kernel: sono gestiti direttamente dall'S.O.

Librerie di gestione sono Posix threads (Linux), Windows threads e Solaris threads.

In ogni caso deve esistere una relazione fra i livelli

MANY TO ONE 4.3.1

Più user-level threads fanno riferimento ad un solo system-level thread.

- Vantaggi: efficienza per gestione nel livello utente del multi-thread.
- Svantaggi: I thread si bloccano se si invocano sgs. call bloccanti.

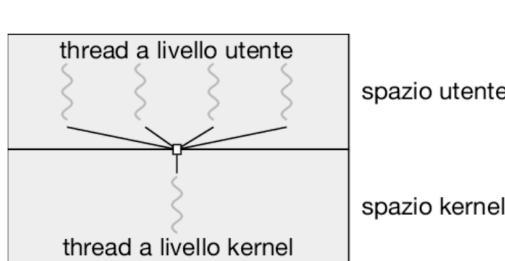


Figura 4.7 Modello da molti a uno.

ONE TO ONE 4.3.2

Associa ad ogni user-level thread un system thread dedicato.

- Vantaggi:
 - Ogni thread può chiamare sys calls bloccanti senza piantare tutto.
 - Si può realizzare il parallelismo.
- Svantaggi:
 - I S.O. mettono un limite al numero dei Kernel-level threads per evitare sovraccarichi.
 - Il programmatore deve fare attenzione a non creare troppi

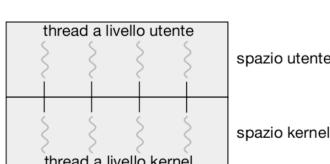


Figura 4.8 Modello da uno a uno.

Many to Many 4.3.3

Dati N user-level threads, ne assegna un numero minore o uguale di kernel-level threads.

- vantaggi: gli stessi dei one to one con l'leggierità
 - Non ci si deve preoccupare dei troppi thread perché l'S.O. pensa ad allocarre
- svantaggi: - L'implementazione può essere complicata.
 - Il numero di thread eseguibili in parallelo non è più tanto importante per i core che ci sono oggi.

Il num. di kernel-level threads può essere deciso in base al sistema o in base al programma in esecuzione.

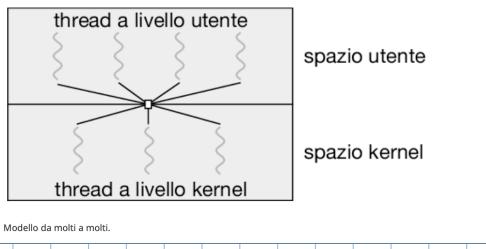


Figura 4.9 Modello da molti a molti.

Two Level Model 4.3.3

Variante del many-to-many con la possibilità di fare un binding di un user-level thread con uno system.

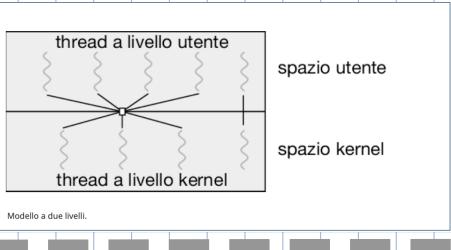


Figura 4.10 Modello a due livelli.

LIBRERIE DEL THREAD > 4.4

Le librerie dei thread forniscono delle API per la gestione.

Possono essere implementate in:

- user-level: Tutto il codice sta nello spazio utente.
- Kernel-level: si appoggia a funzionalità del S.O.
l'uso della libreria generalmente implica system calls.

Le strategie di creazione dei Threads possono essere:

- Asincrone: Il thread padre non aspetta i figli.
Di solito non scambiano dati
- Sincrone: Il thread padre aspetta che tutti terminino
 - fork join Di solito per scambio dati

Le librerie di gestione "comuni" sono

- PThreads (Linux): È una specifica per implementare
- Windows Threads :
- Java Threads: Sotto banco la JVM usa Pthreads e windows threads.

IMPLICIT THREADING > 4.5

È una metodologia che prevede di lasciare al compilatore e alle librerie di runtime la gestione effettiva dei threads. Lo sviluppatore si occupa solo di dichiarare le parti parallelizzabili.

THREAD POOLS

Prevede di creare N thread appena il processo viene avviato. Eventuali attività vengono prese in carico dal 1° task disponibile che appena termina torna in attesa nel pool.

OPEN MP

Set di direttive per C e C++ per definire aree di codice parallelizzabile.

GRANO GENERAL DISPATCH

Set di direttive per Mac OS e iOS. Prevede l'uso di una coda di task da eseguire.

Una volta staccato, viene assegnato al 1° thread del thread pool disponibile.

La coda di task può essere

```
#include <omp.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    /* sequential code */

    #pragma omp parallel
    {
        printf("I am a parallel region.");
    }

    /* sequential code */
}

return 0;
```

- **serial**: coda FIFO in cui si può prendere solo l'elem alla volta → per processo

- **concorrente**: sempre FIFO ma si possono prendere più elem

uso di `fork()`; ed `exec()`; IN MULTI-THREAD ➤ 4.6.1

La chiamata `fork()`; in un processo multi-thread può avere 2 comportamenti distinti:

- Il nuovo processo copia tutti i thread
- // // // copia solo il thread chiamante

La chiamata `exec()`; invece sostituisce il processo corrente **includendo** tutti i tasks.

GESTIONE DEI SEGNALI ➤ 4.6.2

I segnali servono per comunicare un evento ad un processo. Ogni segnale va gestito.

I segnali possono essere:

- Sincroni: come segfault o divisione per 0.
(intervi) vengono inviati al processo corrente.
- Asincroni: vengono dall'esterno

I segnali vengono gestiti tramite gestori dei segnali che possono essere del Kernel (Default) o dell'utente (può sovrascrivere default).

Delivery dei segnali

Se il processo è single threaded, il segnale viene mandato direttamente al processo.

Se è multi-threaded si può mandare:

- a tutti • solo al chiamante • a specifici thread
- Ad 1 thread che raccolge i segnali
viene deciso dal tipo di segnale

THREADS CANCELLATION ➤ 4.6.3

è l'operazione che consente di interrompere un thread prima della terminazione.

La cancellazione può essere:

- Asincrona: Il thread bersaglio viene cancellato subito.
Può essere difficile la gestione se il thread sta gestendo risorse perché non è detto che riesca a liberarsene.
- Differito: Il thread bersaglio controlla periodicamente se deve terminare o meno per terminare ordinatamente per poter de-allocare risorse

GESTIONE CON PTHREADS

Si usa la chiamata `pthread_cancel()`; e il tipo di cancellazione dipende dallo stato del thread

Modalità	Stato	Tipo
Off	Disabilitato	-
Differito	Abilitato	Differito
Asincrona	Abilitato	Asincrono

```
while (1) {
    /* fai qualche lavoro per un po' di tempo */
    ...
    /* verifica se vi sia una richiesta di cancellazione */
    pthread_testcancel();
}
```

Viene anche invocata `pthread_testcancel()` per verificare in maniera differita seguivi di cancellazione. Se presente viene invocato un cleanup-handler per rilasciare le risorse

THREAD LOCAL STORAGE ➤ (TLS) 4.6.4

Consente ai thread di avere dati privati oltre a quelli dello stack. Sono simili alle variabili static.

Sono utili quando non si ha controllo sulla loro creazione

```
static _thread int threadID;
```

SCHEDULER ACTIVATION ➤ 4.6.5

È una metodologia di gestione necessaria per modelli HIR e two level.

Prevede di creare un layer di processore virtuale o LWP che viene assegnato a ogni kernel-level thread.

Gli user-level thread farà lo scheduling sui LWP

UPCALL

Procedura in cui il Kernel informa allo user-level thread un evento tramite il LWP.

