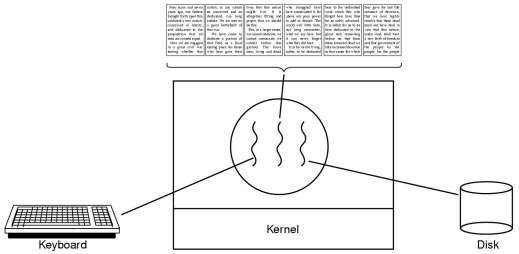
Lezione 5:

I THREADS

- Cosa e' un thread
- Motivazioni per l'uso dei threads
- Similitudini e differenze con i processi
- I vari livelli di supporto per i threads
- Il ciclo di vita dei threads
- Esempi di implementazioni.

Problema: come e' fatto un word processor



Molte azioni contemporanee e indipendenti (input, tabulazione, correzione ortografica, memorizzazione,..) sullo stesso insieme di dati !!

Soluzioni

1: piu' processi che comunicano attraverso una memoria condivisa
- ovrhead per la creazione di numerosi processi
- overhead per la gestione della memoria comune
- Condividere molti dati

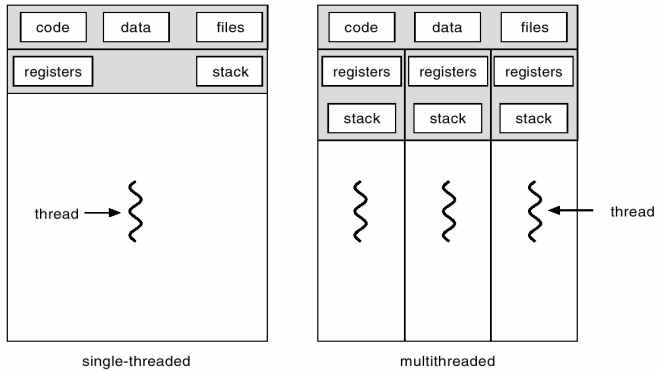
2: un solo processo con "stringhe di esecuzioni" indipendenti che operano sullo stesso spazio di indirizzamento

Stringhe di esecuzione
=
threads

Definizione di Threads

- Nei moderni S.O. un ***thread*** (o *lightweight process,* **LWP**) è spesso l'unità di base di utilizzo della CPU e consiste di:
  - Program counter
  - Insieme dei registri
  - Spazio dello stack
- Un thread condivide con i thread ad esso associati:
  - Spazio di indirizzamento (non c'e' protezione !!)
  - Dati globali
  - File aperti
  - Gestori dei segnali
- L'insieme dei thread e dell'ambiente da essi condiviso è chiamato *task*.
- Un processo tradizionale, o *heavyweight,* corrisponde ad un task con un solo thread.

Processi a thread singolo e multithread



Vantaggi dei threads

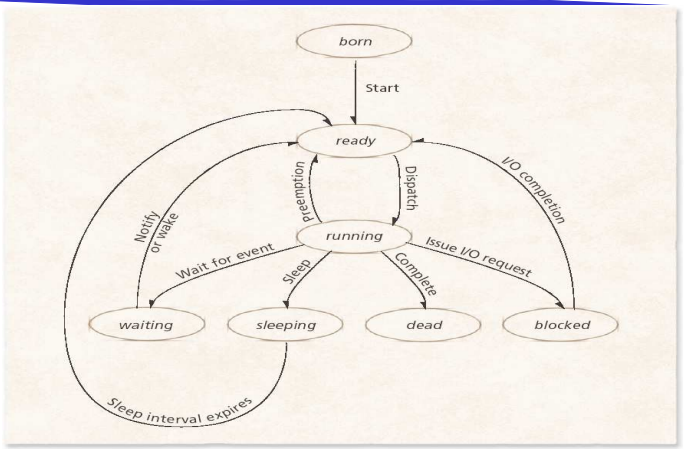
- In un task multithread, mentre un thread è blocccato in attesa, un secondo thread nello stesso task può essere in esecuzione.
  - La cooperazione di più thread nello stesso job fornisce un maggior throughput.
  - Applicazioni che richiedono la condivisione di un buffer (es. produttore-consumatore) traggono beneficio dall'impiego di thread.
- Possono essere gestiti dal sistema operativo o da una applicazione utente
- E' un modo per condividere risorse
- Realizzano una forma di parallelismo
- Hanno un overhead molto inferiore a quelli dei processi

Ciclo di vita di un thread

- Analogamente ad un processo i threads sono schedulati da uno scheduler e attraversano varie fasi:
  - *nuovo*
  - *pronto*
  - *esecuzione*
  - *terminato*
  - *bloccato*
  - *In attesa*
  - *dormiente*

... ma come sempre dipende dall'implementazione

Esempio di ciclo di vita dei threads



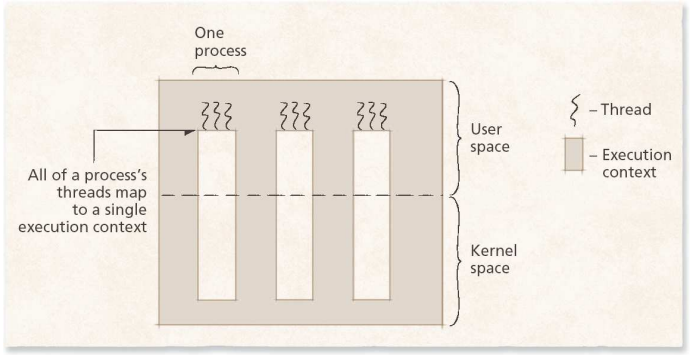
Alcune differenze con i processi

- Operazioni sui threads non corrispondono precisamente alle operazioni sui processi
  - Cancel
    - Indica che un thread deve essere terminato, ma non c'e' garanzia che il thread termina effettivamente (il thread puo' mascherare un segnale di cancellazione)
  - Join
    - Puo' esistere un thread primario che genera gli altri threads (windows XP)
    - quando questo termina fa terminare tutti gli altri thread
    - Il thread primario tipicamente crea i threads secondari e aspetta la loro conclusione

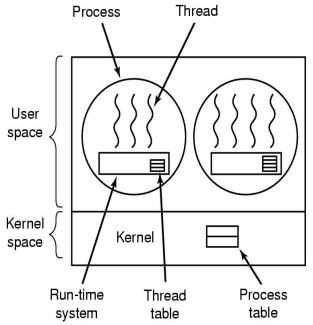
Modelli di implementazioni

- Quasi tutti i sistemi operativi implementano i threads in uno dei seguenti modi
  - User-level threads (modello "molti a uno")
    - i threads sono creati e gestiti nello spazio utenti da specifiche librerie che non possono eseguire istruzioni privilegiate o accedere alle primitive del kernel direttamente
  - Kernel-level threads (modello "uno a uno")
    - i threads sono creati e gestiti direttamente dal kernel
    - In questo caso i thread rappresentano l' "unita' di esecuzione" della CPU
  - Combinazione ibrida delle due modalita' precedenti
    - Modello "molti a molti"

User-level Threads



Implementazione del modello ULT



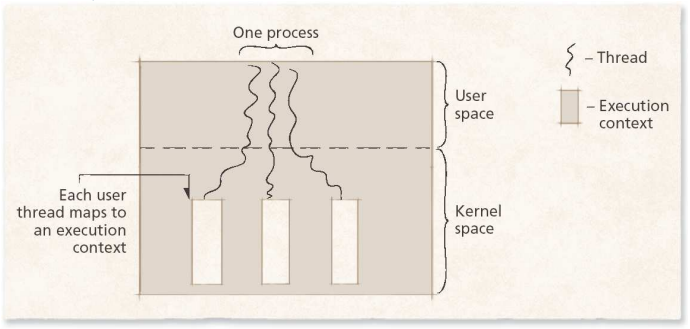
Il kernel mantiene la propria process table e i relativi PCB

Ogni processo ha una propria thread table con relative strutture dati per la descrizione dei thread analoga (ma di dimensioni ridotte) ai PCB

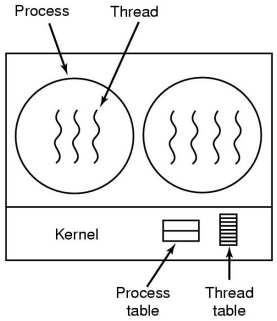
Thread a livello utente (ULT)

- Vantaggi:
  - Il cambio di contesto fra thread non richiede privilegi in modalità kernel (risparmia il sovraccarico del doppio cambiamento di modalità).
  - Lo scheduling può essere diverso per applicazioni diverse.
  - Gli **ULT** (*User Level Thread*) possono essere eseguiti su qualunque SO senza cambiare il kernel sottostante. La libreria dei thread è un insieme di utilità a livello di applicazione.
- Svantaggi:
  - In caso di system call bloccanti, quando un thread esegue una chiamata di sistema, viene bloccato tutto il processo.
  - Un'applicazione multithread non può sfruttare il multiprocessing: in un dato istante un solo thread per processo è in esecuzione.

Kernel-level Threads



Implementazione del modello KLT



Il kernel mantiene
- la process table
- i relativi PCB
- una thread table
- le relative strutture dati

Thread a livello kernel (KLT)

- Vantaggi:
  - Può schedulare simultaneamente più thread (predisposizione al parallelismo su multiprocessori)
  - Miglioramento della scalabilita' e dell'interattivita'
  - Se un thread di un processo è bloccato il kernel può schedulare un altro thread dello stesso processo.
- Svantaggi:
  - Il trasferimento del controllo fra thread dello stesso processo richiede il passaggio in modalità kernel: l'aumento di prestazioni è meno rilevante rispetto all'approccio ULT.
  - Sovraccarico del kernel che potrebbe gestire migliaia di threads (e delle relative strutture dati)

Combinazione ibrida

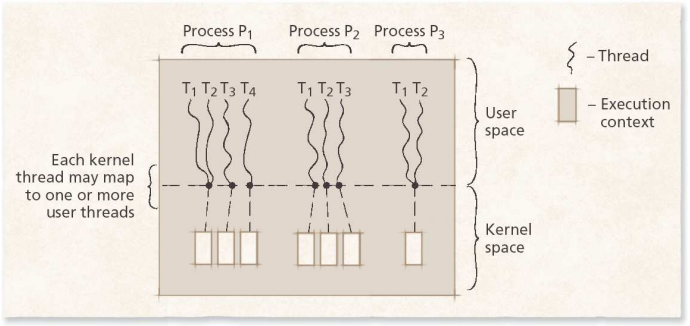
- Cerca di combinare i vantaggi dei due precedenti approcci
- Numero dei threads utente e threads kernel non uguale
- Puo' ridurre l'overhread dell'approccio uno-a-uno

- Realizzazione:
  - Un insieme di thread permanenti (worker threads) vengono creati dal kernel e formano il "threads pool"
  - Ogni nuovo thread utente e' eseguito da un worker thread

- Ottimizzazione
  - possibilita' di specificare il numero dei worker threads in base al carico del sistema
  - Schedulazione dei thread direttamente nel kernel

- Svantaggi
  - Complicazione per il sistema operativo
  - Difficolta' a determinare il numero di worker thread

Combinazione ibrida



Diverse scelte

- user level threads (modello molti-a-uno)
  - vecchi sistemi operativi che non supportano KLT

- kernel level threads (modello uno-a-uno)
  - Windows NT/2000, OS/2, linux

- modello ibrido
  - Solaris 2 , Windows NT/2000 con package *ThreadFiber*

Alcuni problemi:

Che succede se un thread genera un processo?

Due possibilita':
- il nuovo processo contiene il duplicato di tutti i threads
- il nuovo processo contiene il duplicato del solo thread che ha generato il processo

Dipende dai sistemi (alcuni permettono entrambe le possibilita')

Che succede se un thread invoca la funzione exec?

In generale viene sostituita tutta l'area testo, inclusi tutti i threads

Alcuni problemi: cancellazione

Se un thread acquisisce l'uso esclusivo di una variabile, e nel frattempo viene cancellato da un altro thread, la risorsa non viene piu' rilasciata

Due tipi di cancellazione:
- asincrona
  - il thread termina subito, efficiente ma pericolosa
- differita
  - il thread rilascia tutte le risorse prima di terminare, sicura ma meno efficiente

- Di solito sono presenti entrambe le possibilita'
- compito dell'utente scegliere la piu' opportuna

Alcuni problemi: la gestione dei segnali

- i segnali sono stati introdotti con le prime versioni di UNIX, quando non esistevano i threads, ma solo i processi
- I segnali sono quindi diretti a processi, non a thread.
- Se il processo ha solo un thread non ci sono problemi

- Come avviene la gestione dei segnali in un processo multithreading?

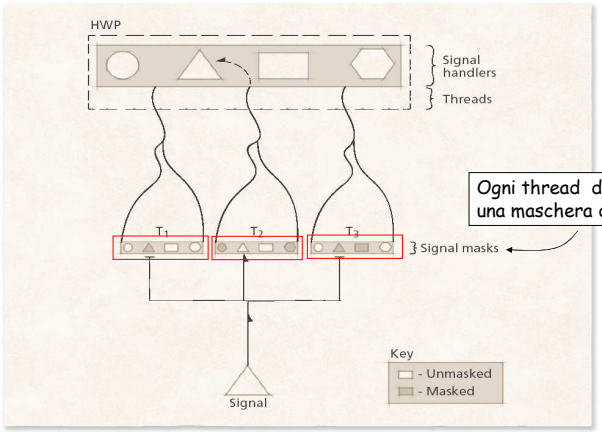
Gestione dei segnali

Due tipi di segnali
- sincroni:
  - Conseguenza dell'esecuzione del programma (es: div per 0)
  - Puo' essere ricevuto dal thread correntemente in esecuzione
- asincrono
  - Conseguenza di un evento esterno al programma in esecuzione (es: completamento di I/O)
  - La libreria di gestione dei Thread deve determinare a quale thread il segnale deve essere consegnato

SOLUZIONE

I threads possono mascherare (filtrare) i segnali

Gestione dei segnali



Threads POSIX (Pthreads)

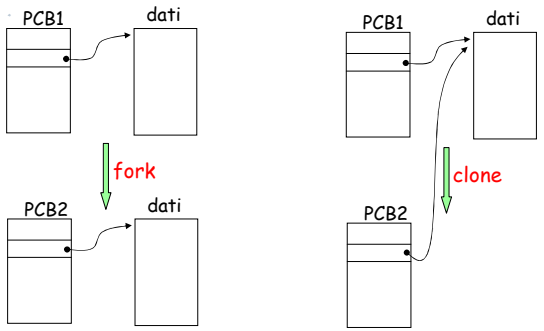
- Uno standard POSIX (IEEE 1003.1c) per la creazione e sincronizzazione dei threads
- Definizione delle API
- Threads conformi allo standard POSIX sono chiamati Pthreads
- Lo standard POSIX stabilisce che registri dei processori, stack e signal mask sono individuali per ogni thread
- Lo standard specifica come il sistema operativo dovrebbe gestire i segnali ai Pthtrads i specifica differenti metodi di cancellazione (asincrona, ritardata, …)
- Permette di definire politiche dischedulingе priorita'
- Alla base di numerose librerie di supporto per vari sistemi

Threads Linux

- Linux utilizza lo stesso tipo di struttura dati per descrivere processi e threads (Process Control Block)
- Linux chiama entrambi *tasks*.
- Modello KLT (uno a uno)
- Per la creazione dei threads, Linux fornisce una versione modificata della chiamata di sistema di fork( ) chiamata clone( ).
  - `clone` accetta argomenti per specificare quali risorse devono essere condivise con il task figlio

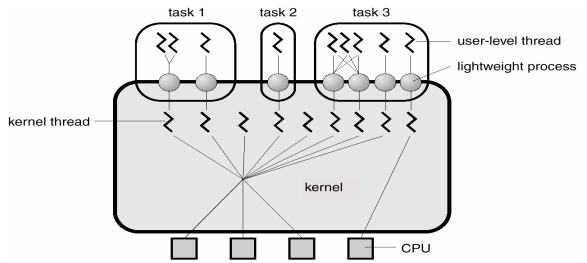
Threads Linux

Il PCB di Linux non contiene direttamente i dati,
ma un puntatore ai dati del processo



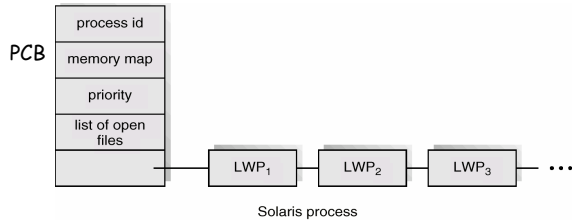
Threads Solaris 2

- Implementazione ibrida (molti a molti)
- tra i threads del kernel e quelli utente c'e' un livello intermedio: i "processi leggeri"
  - ad ogni thread del nucleo e' associato un proc leggero
  - ogni proc leggero puo' gestire piu' thread utente
  - un processo tradizionale e' composto da piu' proc leggeri



Threads Solaris 2

- lo scheduler agisce direttamente sui threads del nucleo
- il numero di processi leggeri varia dinamicamente in maniera da non avere mai processi bloccati
- al termine del processo i proc leggeri non vengono immediatamente cancellati ma si cerca di "riciclarli" per altri processi
- i descrittori di processi contengono un puntatore all'elenco dei threads del kernel associati al processo



Threads Windows XP

- Modello ibrido
- Effettiva unita' di esecuzione per il processore
  - Esegue un pezzo del programma in un contesto di esecuzione del processo usando risorse del processo
  - Il contesto di esecuzione contiene
    - stack
    - Stato dei registri della CPU
    - Attributi dei threads
- threads Windows XP possono creare altri threads chiamate fibre
  - Le fibre sono schedulate per l'esecuzione dal thread che lo ha creato
- Windows XP fornisce ad ogni processo un insieme di worker threads che eseguono funzioni specificate dall'utente

Threads Java

- Java e' uno dei pochi linguaggi di programmazione che prevedono esplicitamente l'uso dei threads (gli altri linguaggi usano librerie)
- esiste sempre almeno un thread dal quale vengono generati gli altri threads

- la JVM nasconde i dettagli del S.O. sottostante, e la sua implementazione determina il tipo di modello (uno-a-uno, molti-a-uno,...)
  - Linux, Windows adotta uno-a-uno ➔ ogni thread Java corrisponde a un thread del nucleo
  - Solaris adotta molti-a-molti ➔ un gruppo di thread Java corrisponde a un gruppo di threads del nucleo