Thread P.66-74

venerdì 7 aprile 2023 11:41

Nozioni sui Thread

Ogni processo puo' essere strutturato come un insieme di thread, ciascuno caratterizzato da una propria traccia di esecuzione

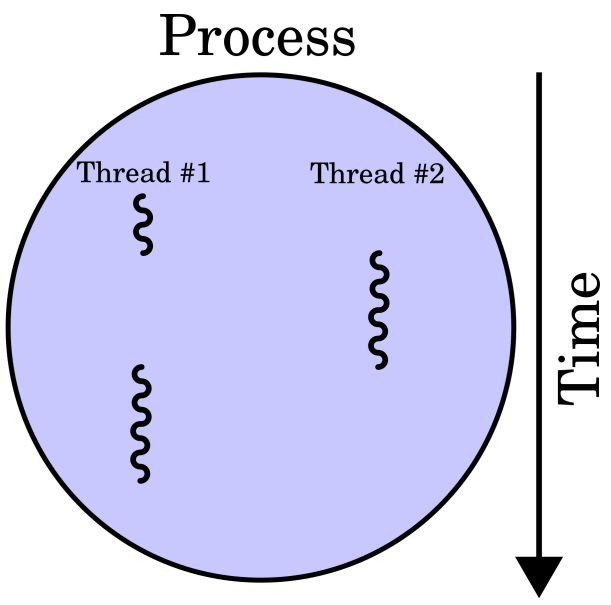
La specifica implementazione dei thread e dei processi dipende dal sistema operativo, ma in generale un thread è contenuto all'interno di un processo e diversi thread contenuti nello stesso processo condividono alcune risorse, lo spazio d'indirizzamento del processo, *mentre processi differenti non condividono le loro risorse*.

Il vantaggio principale dei Thread è nelle prestazioni: operazioni come creazione, terminazione e cambio tra due thread di un processo richiedono meno tempo rispetto alla creazione, terminazione e cambio di processi.

Come i processi anche i thread hanno uno stato di esecuzione e possono sincronizzarsi tra loro. Gli stati di un thread sono ready, running e blocked.

Ci sono quattro operazioni di base associate ai cambiamenti di stato di un thread.

- **Creazione:** quando un processo viene creato, si crea anche un thread. Successivamente un thread può creare un altro thread a cui deve fornire il puntatore delle istruzioni e gli argomenti: vengono creati un contesto per i registri e gli stack, e il nuovo thread è messo nella coda dei *ready*.
- **Blocco:** quando un thread deve aspettare un particolare evento entra in stato *blocked* (salvando i registri utente, il program counter e lo stack pointer)
- **Sblocco:** quando si verifica l'evento per cui il processo era stato posto in stato blocked, il thread passa allo stato ready.
- **Terminazione:** quando un thread completa il suo compito, il suo contesto per i registri e i suoi stack vengono deallocati.



Multithreading

Il **multithreading** indica il supporto hardware da parte di un processore di eseguire più thread. Questa tecnica si distingue da quella alla base dei sistemi multiprocessore per il fatto che i singoli thread condividono lo stesso spazio d'indirizzamento.

Il *multithreading* migliora le prestazioni dei programmi solamente quando questi sono stati sviluppati suddividendo il carico di lavoro su più thread che possono essere eseguiti in parallelo.

Mentre i sistemi multiprocessore sono dotati di più unità di calcolo indipendenti per le quali l'esecuzione è effettivamente parallela, un sistema multithread invece è dotato di una singola unità di calcolo che si cerca di utilizzare al meglio eseguendo più thread nella stessa unità di calcolo. Le due tecniche sono complementari: a volte i sistemi multiprocessore implementano anche il multithreading per migliorare le prestazioni complessive del sistema.

- l'unita' base “proprietaria” di risorse resta il processo in senso classico

Esempi di sistemi Multithreading sono: NT/2000/…., Solaris/Linux MacOS….

Ogni sistema operativo è Multithreading

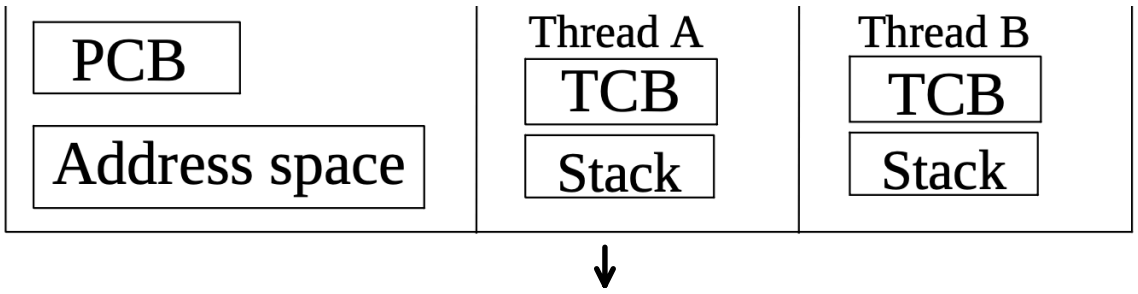
Le tracce lavorano all’interno dello stesso address space.

- Più thread condividono le stesse risorse come cache o translation lookaside buffer e quindi possono interferire a vicenda rallentandosi.
- Le prestazioni dei singoli thread non migliorano, ma anzi possono degradare all'aumento dei thread concorrenti.
- Il supporto hardware del multithreading e dei sistemi multiprocessori richiede anche un contributo del software, i programmi e i sistemi operativi devono essere adattati per gestire questa nuova possibilità.

La presenza di due o più thread che accedono contemporaneamente agli stessi dati può portare a risultati imprevisti e indesiderati. Infatti, senza l'applicazione di particolari tecniche di programmazione, non è possibile prevedere in maniera deterministica, al momento dell'esecuzione, quando verrà eseguito quel thread specifico: la loro progressione dipende infatti dalle priorità decise dallo scheduler del sistema operativo e non dal programmatore.

Più thread infatti possono accedere ad una stessa variabile e modificarne il contenuto o valore. Sebbene questo non accada nello stesso momento perché l'accesso ad una variabile, che di fatto è memorizzata in memoria RAM, è intrinsecamente limitata ad un'unità al massimo, può accadere che un thread modifichi il valore di una variabile, mentre un altro thread necessita del vecchio valore memorizzato in essa. Si ricorre pertanto all'uso di tecniche di sincronizzazione come la mutua esclusione per risolvere il problema.

Modello multithreading



- L’address space del processo e’ comunque pienamente accessibile a tutti i thread di quel processo, ad esempio tramite puntatori
- Questo pone importanti problemi di consistenza delle informazioni e quindi di sincronizzazione delle attivita’ dei thread

Variabili per thread – Thread Local Storage (TLS)

- Sono variabili globali di cui esistera’ a tempo di esecuzione una istanza per ogni thread che viene attivato
- Il singolo thread potra’ accedere alla sua istanza semplicemente riferendo il nome della variabile (come una tradizionale variabile globale)
- L’accesso puo’ avvenire anche tramite puntatore

Le informazioni dei thread devono sopravvivere, perché se sono locali alle funzioni, quando esse ritornano il controllo, perdiamo i riferimenti.

- I costrutti per il TSL in ambienti i sviluppo communi sono:

  `__declspec(thread)` in Visual-Studio

  `__thread` in gcc/ld

Windows - attributi principali di oggetti thread

- ID del thread
- Contesto del thread
- Priorita’ base del thread (legata a quella di processo)
- Priorita’ dinamica del thread
- Affinita’ di processore (insieme di processori utili per l’esecuzione)
- Tempo di esecuzione
- Stato di uscita

Windows - servizi principali di oggetti thread

- Creazione di thread
- Apertura di thread
- Richiesta/modifica di informazioni di thread
- Terminazione di thread
- Lettura di contesto
- Modifica di contesto
- Blocco

L'archiviazione thread-local (TLS, Thread-Local Storage) è il meccanismo attraverso il quale ogni thread in un dato processo multithread alloca lo spazio di archiviazione per i dati specifici dei thread. Nei programmi multithread standard, i dati vengono condivisi da tutti i thread di un determinato processo, mentre l'archiviazione thread-local è il meccanismo che consente di allocare i dati per singoli thread.

- Il Thread Local Storage (TLS) permette a ciascun thread di possedere una propria copia dei dati (non necessariamente tutti i dati del processo di origine)

- Diversamente dalle variabili locali (che sono visibili solo durante una singola chiamata di funzione), i dati TLS sono visibili attraverso tutte le chiamate
- Simili ai dati dichiarati **`static`** in C
  - I dati TLS sono però unici per ogni thread