2 - Thread

Thread

Nei sistemi operativi tradizionali, ogni processo ha un singolo thread di controllo. Tuttavia, in molte situazioni è utile avere più thread di controllo nello stesso spazio di indirizzi, permettendo un'esecuzione quasi parallela. La multithread execution consente a un processo di avere N thread in esecuzione.

Perché consentire più thread per processo?

I thread sono miniprocessi leggeri, detti lightweight processes, poiché non hanno un proprio PID o spazio di indirizzamento. Offrono un parallelismo efficiente in termini di spazio e tempo, poiché non devono essere gestiti dallo scheduler ma dal processo stesso.

> Esempio: Web Server
> Un web server che crea un nuovo processo per ogni richiesta client avrebbe difficoltà a gestire milioni di richieste simultanee. Utilizzando thread invece di processi, ogni richiesta è gestita da un thread, molto più leggero di un processo e gestito direttamente dal web server. Questo permette al server di rispondere a N richieste durante il tempo allocato alla CPU.

Sincronizzazione e Condivisione della Memoria

I thread condividono la stessa area di memoria del processo che li ha generati, permettendo lettura e scrittura sulla memoria condivisa. Tuttavia, ciò comporta problemi di sincronizzazione: se un thread modifica una variabile mentre un altro la legge, possono verificarsi inconsistenze.

Informazioni Personali di un Thread

Ogni thread ha:

- Stack
- Registri

- Memoria
- Stato
  I thread possono chiamare qualsiasi chiamata di sistema supportata dal sistema operativo per conto del processo a cui appartengono.

Thread in POSIX

- pthread_create: crea un nuovo thread.
- pthread_exit: termina il thread chiamante.
- pthread_join: attende l'uscita di uno specifico thread.
- pthread_yield: rilascia la CPU per consentire l'esecuzione di un altro thread.
- pthread_attr_init: crea e inizializza la struttura di attributi di un thread.
- pthread_attr_destroy: rimuove la struttura di attributi di un thread.

Implementazione dei Thread nello Spazio Utente (JAVA)

Pro:

- Gestiti dal kernel come processi a singolo thread.
- Compatibili con sistemi operativi che non supportano direttamente i thread, gestiti tramite una libreria.
- I processi che utilizzano i thread a livello utente hanno una tabella dei thread che mantiene le informazioni del thread
- Procedure per il salvataggio del thread nello spazio utente per non effettuare trap e cambi di contesto
- Permettono la personalizzazione dell'algoritmo di scheduling e maggiore scalabilità.

Contro:

- Problemi con chiamate di sistema bloccanti, che fermano tutti i thread nel processo.
- Errori di pagina possono bloccare l'intero processo.
- Mancanza di interrupt del clock per scheduling round-robin.
- Meno adatti per applicazioni dove i thread si bloccano frequentemente, come web server multithread.

Implementazione dei Thread nello Spazio Kernel

Pro:

- Il kernel gestisce i thread, eliminando la necessità di un sistema run-time per processo.
- Le chiamate di sistema bloccanti vengono implementate come tali.
- Alcuni sistemi riciclano i thread per ridurre i costi.
- Richiede cautela per evitare errori nella programmazione con thread.

Problemi aperti:

- Conflitti causati da thread che sovrascrivono dati cruciali.
- Wrappers possono evitare conflitti ma limitano il parallelismo.
- La gestione dei segnali è complessa, con segnali specifici per thread e altri no.