LEZIONE 5

Thread (letteralmente significa filo)

**Idea generale:** Sono i piccoli percorsi di esecuzione. Un programma è un’entità passiva che quando vien e mandata in esecuzione diventa uno o più processi, ovvero un’entità attiva; un processo poi per essere eseguito diventa uno o più thread.

Quindi un thread è un’unità elementare di elaborazione. Si ha un punto di start da cui originano questi thread, i quali si evolvono nel tempo finchè tali processi non terminano.

**Thread Linux e Unix**

Immagine che contiene tastiera

Descrizione generata automaticamenteI thread vanno a realizzare un vero e proprio **albero di thread** con radice root

Processi e thread

* **Processo (o task):** programma in esecuzione, con il suo spazio di indirizzamento ed il suo stato. Un processo è ad esempio un’istanza di word o chrome. Esso è detto anche **processo pesante**, in riferimento al contesto che si porta dietro (ovvero la PCB la sua carta d’identità che contiene tutte le informazioni che lo riguardano).
* **Thread:** **singolo flusso sequenziale di controllo all’interno di un processo. Un processo può contenere più thread.** Tutti i thread di un processo condividono lo stesso spazio di indirizzamento. Detto anche processo leggero, perché ha un contesto semplice (per costruzione i thread condividono lo spazio del processo per esempio il codice di esecuzione è lo stesso).

Parallelismo

* **Sistema parallelo:** architettura in cui sono presenti più unità di elaborazione (CPU) sulle quali sono in esecuzione processi e thread. In ogni istante di tempo, si può avere più di un processo o più di un thread fisicamente in esecuzione.
* **Sistemi monoprocessore time-sliced:** vi è una sola unità di elaborazione. **Il parallelismo di processi thread viene simulato (concorrenza)**allocando a ciascuna frazione del tempo di CPU (time-slice). Allo scadere di ogni unità di tempo, il SO opera un cambio di contesto (**context swich).** i thread sono processi leggeri perché il cambio di contesto è veloce.

Preemption

Quando si lancia un processo o un thread bisogna fare una considerazione, ovvero, questa esecuzione è prelazionabile o no? Teoricamente se si ragiona usando la logica del processo o del thread, essi non hanno alcun interesse ad auto interrompersi; di conseguenza cedono le risorse solo quando hanno finito. Se arriva un processo che ha maggior priorità che succede? Bisogna considerare quello detto prima, cioè se il sistema è preemptive o no.

* **Sistema non preemptive:** il cambio di contesto avviene quando il processo o thread interrompe la propria esecuzione o **volontariamente**, o perché in attesa di un evento (input, output); cioè il processo no può essere interrotto in alcun modo.
* **Sistema preemptive**: allo scadere del time-slice il processo o thread viene forzatamente interrotto e viene operato il cambio di contesto; ovvero può essere interrotto se arriva un processo con priorità più elevata.

Motivazioni dei thread

E’ spesso necessario dividere il programma anche in “sotto-compiti” indipendenti.

* Un programma può avere diverse funzioni concorrenti:
* **Operazioni ripetute nel tempo** ad intervalli regolari (es. animazioni)
* Esecuzioni di compiti laboriosi **senza bloccare la GUI del programma (**es. se si apre una pagina web anche se questa non è del tutto caricata, si ha comunque la possibilità di poter leggere la pagina e man mano che si legge, si può notare come si formano le immagini dato che ogni immagine vioene caricata da un thread specifico)
* Attesa di messaggi da un altro programma
* Attesa di input da tastiera o dalla rete
* Es. si pensi ai compiti svolti da un web browser
* L’alternativa al multithreading è il polling. Il polling, oltre che scomodo da implementare, consuma molte risorse di CPU. E’ quindi estremamente inefficiente

Definizioni thread e multithread

* Un thread è un singolo flusso di controllo sequenziale all’interno di un programma.
* Ogni thread ha un inizio, una fine, una sequenza e ad ogni istante un solo punto di esecuzione
* Thread= lightweight process: utilizza il contesto del processo
* Execution context: il contesto di un singolo thread prevede inoltre, ad esempio, un proprio stack ed un proprio Program Counter

**Esempio**

Immagine che contiene segnale

Descrizione generata automaticamente **Spiegazione** il client fa una richiesta al server, il quale genera un thread; nonostante ciò però continua ad essere in ascolto di altre richieste sia da parte dello stesso client che da parte di altri client

**Processi a singolo thread e multithread**

**Immagine che contiene screenshot

Descrizione generata automaticamente**

Vantaggi

* **Tempo di risposta**: Esempio: un browser web multithread potrebbe permettere l’interazione con un utente mediante un thread mentre un’immagine viene caricata da un altro thread.
* **Condivisione delle risorse**

Esempio stesso spazio di indirizzi

* **Economia:** assegnare memoria e risorse per la creazione di nuovi processi è costoso… un thread è ‘leggero’ e richiede meno assegnazioni di risorse.
* **Uso di più unità d’elaborazione:** i thread possono essere eseguiti in parallelo

**Svantaggio:** l’uso sconsiderato può facilmente indurre in situazioni incresciose, in quanto è molto complesso ed un minimo errore può essere dannoso.

Thread a livello d’utente

* Essi sono gestiti come uno strato separato sopra il nucleo del SO, e **realizzati tramite una libreria di funzioni** per la creazione, lo scheduling e la gestione, **senza alcun intervento del nucleo.**
* Esempi:
* libreria POSIX pthread
* libreria C-thread del sistema Mach
* UI-thread del sistema Solaris 2

Thread a livello Kernel

* Gestiti direttamente da SO
* Esempi:
* Windows 95/NT/2000
* Solaris
* Tru64 UNIX
* BeOS
* Linux

**Differenze tra thread livello utente e kernel:** i thread livello utente sono più semplici perché sono gestiti direttamente dall’utente; quelli kernel invece è il SO che deve in qualche modo gestire i thread, la situazione diventa molto più complicata.

Modelli multithread (relazioni tra thread a livello utente e thread livello kernel)

* **Modello da molti a uno**
* Fa corrispondere molti thread a livello utente ad un singolo thread al livello kernel
* Immagine che contiene acqua, sedendo, uomo, sciando

  Descrizione generata automaticamenteUsato su sistemi che non gestiscono i thread al livello kernel.

**Spiegazione**

Lo schema di elaborazione da seguire in questo caso è un qualsiasi algoritmo di scheduling, dato che ci sono più thread a lv utente che devono corrispondere ad un unico flusso di elaborazione

* Immagine che contiene orologio, lotto, acqua, rosso

  Descrizione generata automaticamente**Modello 1 a 1**
* Mette in corrispondenza ciascun thread del livello d’utente con un thread del livello kernel
* Esempio:

-windows 95/98/NT/2000

-OS/2

* **Modello molti a molti**

**Immagine che contiene fotografia, tenendo, acqua, largo

Descrizione generata automaticamente**-Mette in corrispondenza più thread del livello d’utente con un numero **minore o uguale** di thread del livello kernel

-Consente ai programmi di creare liberamente i thread che ritengono necessari

-Solaris

-Windows NT/2000 con il pacchetto ThreadFiber

**NOTA:** 1 a molti non c’è nello schema in quanto non è pensabile avere più thread a livello kernel che utente

Pthread

* Lo standard POSIX (IEEE 1003.1c) che definisce l’API per la creazione e la sincronizzazione dei thread

**POSIX: p**ortable **o**perating **s**ystem **i**nterfacefor Uni**x**

* **Non si tratta di una realizzazione ma di una definizione del comportamento dei thread;** i progettisti di SO possono realizzare le API così definite come meglio credono.
* Comuni nei sistemi UNIX

Funzionano cosi: in un programma multithread in linguaggio c che impiega le API Pthread: il programma esegue il primo thread (preogramma padre) e poi il thread figlio.

Thread nel linguaggio java

* I thread nel linguaggio java possono essere creati
* Creando una nuova classe derivata dalla classe Thread
* Sovrascrivendo il metodo run di quella classe
* I thread nel linguaggio java sono gestiti dalla macchina virtuale (JVN)

**Questioni di programmazione Multithread**

1. **Chiamate del sistema fork () ed exit ()**

In un programma multithread la semantica delle chiamate di sistema fork() ed exit() cambia se un thread in un programma invoca la chiamata di sistema fork(), il nuovo processo potrebbe, in generale, contenere un duplicato di tutti i thread oppure del solo thread invocante.

1. **Cancellazione**

Permette di terminare un thread prima che completi il suo compito

* Per es. in una ricerca concorrente in un database il primo thread che trova il risultato dovrebbe comportare la cancellazione degli altri
* Per es. la terminazione di un web browser comporta le terminazioni dei caricamenti di pagina.

1. **Gestione dei segnali**

* Riguarda la gestione e coordinazione delle occorrenze di eventi che avvengono nel sistema

1. **Gruppi di thread**

* un numero illimitato di thread potrebbe esaurire le risorse del sistema come la CPU o la memoria. Si cerca di creare dei gruppi di thread il cui scopo è quello di snellirne la gestione

1. **Dati specifici dei thread**

* in particolari circoastanze, ogni thread può necessitare di una copia èprivata di certi dati