Corso di Informatica

Modulo T5

1–I thread

Prerequisiti

- Concetto di processo
- Interazione diretta
- Interazione indiretta
- Mutua esclusione
- Produttore-consumatore
- Schedulatore

Introduzione

Generalmente, un programma viene eseguito in modo sequenziale: la **CPU** esegue, una dopo l'altra, le istruzioni del programma, dall'inizio alla fine.

In alcune situazioni, potrebbe essere interessante o utile studiare la possibilità che vengano eseguiti contemporaneamente:

- più programmi
- più parti indipendenti di un stesso programma.

Queste situazioni sono descritte in una particolare tecnica di programmazione detta **programmazione concorrente**.

La programmazione concorrente

Questo tipo di programmazione studia le tecniche e gli strumenti necessari a far eseguire simultaneamente più applicazioni (o più attività da parte di una stessa applicazione), comportamento tipico dei **sistemi multiprogrammati**.

La multiprogrammazione si presenta, ad esempio quando:

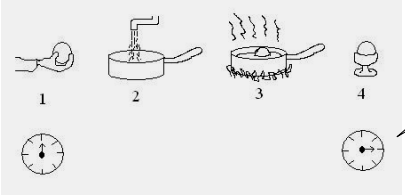
- più utenti possono accedere contemporaneamente ad un sistema informatico;
- un solo utente può eseguire più programmi simultaneamente
- si può scomporre l'esecuzione di un singolo programma in più attività concorrenti.

Per comprendere bene questo argomento è necessario richiamare alcuni concetti di base dei sistemi operativi.

Processo

Iniziamo con un esempio: quando si prepara un piatto, si eseguono una ad una le istruzioni di una ricetta.

La preparazione del piatto è un esempio di **processo** che nasce quando si esegue nel tempo una sequenza di azioni.



Il tempo è la variabile fisica che interviene nell'avanzamento di un processo

Processo e programma

Quando un **programma** va in esecuzione, si genera il corrispondente **processo**. Ad ogni **istruzione** del programma, corrisponde una specifica **azione** del processo.

Un programma è una sequenza di istruzioni, registrate su disco (concetto statico).

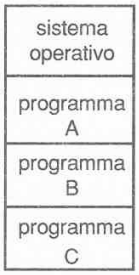
Un processo (concetto dinamico) è una sequenza di azioni svolte in **RAM**, durante l'esecuzione di un programma.

In informatica un programma in esecuzione è un esempio di processo.

Concetto statico	Concetto dinamico
Programma	Processo
Istruzione	Azione

Multiprogrammazione

La **multiprogrammazione** (*multitasking*) consiste nell'avere contemporaneamente più programmi presenti in **RAM**. Un algoritmo del sistema operativo, detto **schedulatore**, sottopone, secondo criteri e strategie opportuni, i rispettivi processi alla CPU per il loro avanzamento.



Lo spazio di memoria riservato:

- ai dati (area dati, stack e heap)
- alle istruzioni

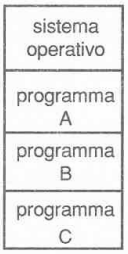
e assegnato a ciascun processo si dice **spazio di indirizzamento del processo.**

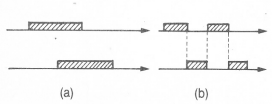
Tutti gli attuali sistemi operativi funzionano in multiprogrammazione.

Ogni processo ha il proprio spazio di indirizzamento.

Multiprocessing

Se ci sono più processori, i programmi presenti in **RAM** possono essere eseguiti contemporaneamente, e in questo caso si parla di **multiprocessing** o **multielaborazione**, altrimenti lo faranno a turno (**context switching**).





(a) *Multiprocessing* effettivo (**overlapping**)

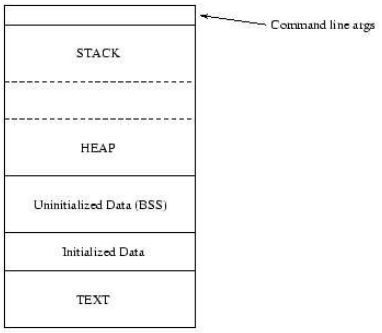
(b) *Multiprocessing* simulato (**interleaving**)

Caratteristiche di un processo

Un processo è individuato da alcune caratteristiche che possono variare da istante a istante, durante la sua evoluzione.

Lo **stato di un processo** è individuato in genere da:

- pid (*process identifier*)
- spazio di indirizzamento
  - area dati, stack e heap
  - area codice
- program counter
- registro di stato



Processi figli

Un processo può inoltre creare **processi figli**, il cui avanzamento avviene in parallelo con il processo padre (programmazione multiprocesso).

Ricordiamo che un processo figlio erediterà tutte le caratteristiche del processo padre (dati e codice) ma:

- agisce su uno spazio di indirizzamento proprio;
- opera su copie delle risorse necessarie.

Interazione tra processi

Sebbene si possa pensare che i processi avanzino sempre in modo indipendente l'uno dall'altro, tuttavia può succedere (molto spesso) che due o più processi si trovino a interagine, per esempio per l'uso di una risorsa.

Si ha, quindi:

- **interazione indiretta** (tipica del problema della mutua esclusione)
- **interazione diretta** (tipica del problema del produttore-consumatore)

Allo studio dettagliato di questi tipi di interazione è dedicata un'Unità successiva.

Vantaggi e svantaggi

VANTAGGIO: La programmazione multiprocesso permette l'esistenza dei sistemi **multitasking**.

SVANTAGGI: La creazione di nuovi processi porta a due problemi principali:

- un numero elevato di processi in esecuzione può causare un elevato sovraccarico nella macchina (eccesso di *context switching*)
- la condivisione dei dati è intrinsecamente complessa e i processi possono comunicare solo mediante meccanismi appositi:
  - memoria condivisa
  - pipe
  - semafori
  - messaggi

Possibili tecniche **IPC** (*Interprocess Communication*)

I *thread*

In certi casi, può essere utile che uno o più flussi di istruzioni, all'interno di uno stesso processo, vengano eseguiti in parallelo, condividendo le risorse del processo padre e il suo spazio di indirizzamento.

Questa opportunità è rappresentata da uno strumento che prende il nome di **thread.**

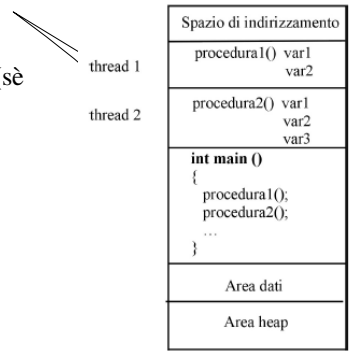
> A differenza dei processi, quando un *thread* viene creato, condivide il suo spazio di memoria con tutti gli altri *thread* che fanno parte del processo (codice, dati risorse, memoria).

Il *multithreading* è simile ad un *multitasking* all'interno dello stesso processo.

I *thread*

Un processo ha sempre almeno un **thread** (sè stesso), ma in alcuni casi un processo può avere più *thread* che vengono eseguiti in parallelo.

Un *thread* è un singolo flusso di istruzioni facente parte di un processo, che lo **schedulatore** esegue in concorrenza con il processo.



> Spesso i *thread* sono detti **processi leggeri** proprio in virtù del fatto che si tratta di particolari sottoprocessi non aventi un proprio spazio di indirizzamento, in quanto usano quello del processo padre.

Analogie con il mondo reale

Analogia 1: un *thread* può essere visivamente pargonato ad una fune composta da vari fili attorcigliati: se la fune è il processo in esecuzione, allora i singoli fili che la compongono sono i *thread*.

Analogia 2: un'altra analogia per descrivere il concetto di *thread*, è quella di immaginare più cuochi in una cucina, che consultano le istruzione di uno stesso libro di cucina, non necessariamente nella stessa pagina.

Caratteristiche di un *thread*

Un *thread*, per poter avanzare in parallelo con altri, deve possedere alcuni propri attributi:

- pid
- area stack
- program counter
- registro di stato

Il valore di questi attributi individua il *thread* che può essere visto come una procedura che evolve in parallelo con altre.

Thread e processi

Fra i tradizionali processi e i *thread* ci sono alcune differenze:

1. i processi sono in genere indipendenti e comunicano tramite appositi meccanismi offerti dal sistema stesso, mentre *thread* sono sottoinsiemi di uno stesso processo.
2. nello stesso processo i *thread* rappresentano più flussi di esecuzione, ognuno dei quali svolge un proprio compito.
3. i *thread* condividono le risorse del processo:
   - variabili globali
   - CPU
   - memoria heap
   - tabella dei descrittori dei file

   e quindi c'è una completa visibilità dei dati

Thread e processi

4. i processi hanno un proprio spazio di indirizzamento, mentr i *thread* hanno uno spazio di indirizzamento condiviso che consente quindi una elevata velocità di comunicazione tra essi.
5. il passaggio da un *thread* ad un altro (*context switching*), poiché l'ambiente è in gran parte comune, avviene rapidamente.
6. poiché un *thread* è una parte di un processo, la sua creazione o una sua nuova attivazione viene effettuata in tempi ridottissimi a costi minimi.

> Se, ad esempio, un *thread* cambia il valore di una variabile, tutti gli altri *thread* vedranno il cambiamento.
>
> Analogamente, se un dato *thread* chiude un file, gli altri *thread* non potranno più accedervi.

Esempi di *thread*

Un esempio di applicazione che fa uso di più *thread* è un browser web, che usa un *thread* distinto per scaricare ogni immagine di cui è composta una pagina web.

Un altro esempio è costituito dai processi *server*, spesso chiamati *servizi* o *daemon*, che sono in grado di rispondere contemporaneamente alle richieste provenienti da più utenti (*client*)

Un programma esegue certe operazioni oltre a rispondere agli input dell'utente. Invece di controllare periodicamente se l'utente ha inserito dei dati in input, un *thread1* esegue le operazioni principali, mentre *thread2* gestisce l'input dell'utente (nel caso questo input non coinvolga l'elaborazione di *thread1*).

Esempi di *thread*

Un web server, come Apache, deve essere sempre in ascolto di nuove richieste e contemporaneamente eseguire quelle ricevute.

Con la programmazione tradizionale non è possibile ricevere nuove richieste fino a quando l'ultima richiesta ricevuta non sia stata servita. Come è immaginabile molte delle richieste entranti andrebbero perse.

Se però utilizziamo due *thread*, possiamo fare in modo che uno si dedichi a ricevere e accodare le richieste, e l'altro a servirle. Ovviamente ciò può realmente avvenire solo su macchine dotate di più processori, ma viene ben simulato da ormai qualsiasi sistema operativo.

Ottimizzazione delle operazioni

Un sistema operativo in grado di gestire la programmazione *multithreading* ottimizza l'uso delle risorse, in particolare basti pensare ai tempi morti presenti nelle seguenti situazioni:

- dal punto di vista del processore, il tempo che intercorre fra due pressioni di tasti è enorme.
- le operazioni di lettura/scrittura su un hard disk sono lentissime se confrontate con la velocità di esecuzione del processore
- l'accesso alla rete e la navigazione su web, per quanto veloce, presenta spesso tempi morti.

In tutti questi casi, il sistema operativo sfrutta i tempi morti di CPU per eseguire altre operazioni come i *thread*

Argomenti

- La programmazione concorrente
- Processo
- Processo e programma
- Multiprogrammazione
- Multiprocessing
- Caratteristiche di un processo
- Processi figli
- Interazione tra processi
- Vantaggi e svantaggi
- I *thread*
- Analogie con il mondo reale
- Caratteristiche di un *thread*
- *Thread* e processi
- Esempi di *thread*
- Ottimizzazione delle operazioni

Altre fonti di informazione

- P.Camagni, R.Nicolassy – Java: interfacce grafiche e programmazione concorrente, ed. Hoepli Informatica
- **http**://**java**.sun.com/docs/books/tutorial/essential/threads/index.html