Processo (informatica)

Da Wikipedia, l'enciclopedia libera.

In Informatica per **processo** si intende un'istanza di un programma in esecuzione in modo sequenziale. Più precisamente è un'attività controllata da un programma che si svolge su un processore in genere sotto la gestione o supervisione del rispettivo sistema operativo.

Indice

Programmi e processi

Un programma è costituito dal codice oggetto generato dalla compilazione del codice sorgente, ed è normalmente salvato sotto forma di uno o più file. Esso è un'entità statica, che rimane immutata durante l'esecuzione.

Il processo è l'entità utilizzata dal sistema operativo per rappresentare una specifica esecuzione di un programma. Esso è quindi un'entità dinamica, che dipende dai dati che vengono elaborati, e dalle operazioni eseguite su di essi. Il processo è quindi caratterizzato, oltre che dal codice eseguibile, dall'insieme di tutte le informazioni che ne definiscono lo stato, come il contenuto della memoria indirizzata, i thread, i descrittori dei file e delle periferiche in uso.

L'uso dell'astrazione dall'hardware è necessario al sistema operativo per realizzare la multiprogrammazione.

Processi e thread

Il concetto di processo è associato, ma comunque distinto da quello di thread (abbreviazione di **thread of execution**, filo dell'esecuzione) con cui si intende invece l'unità granulare in cui un processo può essere suddiviso (sottoprocesso) e che può essere eseguito a divisione di tempo o in parallelo ad altri thread da parte del processore. In altre parole, un thread è una parte del processo che viene eseguita in maniera concorrente ed indipendente internamente allo stato generale del processo stesso. Il termine inglese rende bene l'idea, in quanto si rifà visivamente al concetto di fune composta da vari fili attorcigliati: se la fune è il processo in esecuzione, allora i

singoli fili che la compongono sono i thread.

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

Una differenza sostanziale fra thread e processi consiste nel modo con cui essi condividono le risorse: mentre i processi sono di solito fra loro indipendenti, utilizzando diverse aree di memoria ed interagendo soltanto mediante appositi meccanismi di comunicazione messi a disposizione dal sistema, al contrario i thread di un processo tipicamente condividono le medesime informazioni di stato, la memoria ed altre risorse di sistema.

L'altra differenza sostanziale è insita nel meccanismo di attivazione: la creazione di un nuovo processo è sempre onerosa per il sistema, in quanto devono essere allocate ovvero assegnate risorse necessarie alla sua esecuzione (allocazione di memoria, riferimenti alle periferiche, e così via, operazioni tipicamente onerose); il thread invece è parte di un processo e quindi una sua nuova attivazione viene effettuata in tempi ridottissimi a costi minimi.

Le definizioni sono le seguenti:

- Il *processo* è l'oggetto del sistema operativo a cui sono assegnate tutte le risorse di sistema per l'esecuzione di un programma, tranne la CPU.
- Il *thread* è l'oggetto del sistema operativo o dell'applicazione a cui è assegnata la CPU per l'esecuzione.

In un sistema che non supporta i thread, se si vuole eseguire contemporaneamente più volte lo stesso programma, è necessario creare più processi basati sullo stesso programma. Tale tecnica funziona, ma è dispendiosa di risorse, sia perché ogni processo deve allocare le proprie risorse, sia perché per comunicare tra i vari processi è necessario eseguire delle relativamente lente chiamate di sistema, sia perché la commutazione di contesto tra thread dello stesso processo è più veloce che tra thread di processi distinti.

Avendo più thread nello stesso processo, si può ottenere lo stesso risultato allocando una sola volta le risorse necessarie, e scambiando i dati tra i thread tramite la memoria del processo, che è accessibile a tutti i suoi thread.

Un esempio di applicazione che può far uso di più thread è un browser Web, che usa un thread distinto per scaricare ogni immagine in una pagina Web che contiene più immagini.

Un altro esempio è costituito dai processi server, spesso chiamati *servizi* o *daemon*, che possono rispondere contemporaneamente alle richieste provenienti da più utenti.

In un sistema multiprocessore (SMP), si possono avere miglioramenti prestazionali, grazie al parallelismo fisico dei thread. Tuttavia, l'applicazione deve essere progettata in modo tale che essa suddivida tra i thread il carico di elaborazione. Tale progettazione è difficile e soggetta a errori, e il programma risultante, se eseguito su un sistema monoprocessore, potrebbe essere più lento di uno con un solo thread; pertanto oggi sono ancora pochi i software che usano i thread per sfruttare i sistemi SMP.

Supporto del sistema operativo

I sistemi operativi si classificano nel seguente modo in base al supporto che offrono a processi e thread:

- *Monotasking*: non sono supportati né processi né thread; si può lanciare un solo programma per volta.
- *Multitasking cooperativo*: sono supportati i processi, ma non i thread, e ogni processo mantiene la CPU finché non la rilascia spontaneamente.
- *Multitasking preventivo*: sono supportati i processi, ma non i thread, e ogni processo mantiene la CPU finché non la rilascia spontaneamente o finché il sistema operativo sospende il processo per passare la CPU a un altro processo.
- *Multithreaded*: sono supportati sia i processi, sia i thread.
- *Multitasking embedded*: sono supportati i thread, ma non processi, nel senso che si può eseguire un solo programma per volta, ma questo programma può avere più thread (solitamente detti task).

Si noti inoltre che la mancanza di supporto ai thread da parte del sistema operativo non impedisce la programmazione parallela. Infatti il parallelismo tra thread può essere simulato da librerie di programmazione o anche dal supporto run-time del linguaggio di programmazione. In tal senso si parla di "thread del kernel" per indicare un thread gestito dal sistema operativo, e di "thread utente" per indicare un thread gestito da una libreria applicativa. Per esempio, alcune versioni di Unix non supportano i thread, per cui si ricorre ai thread utente, altri (per esempio Linux) supportano direttamente i thread a livello del kernel.

Strutture dati relative a processi e thread

Nel sistema operativo, ciascun processo è identificato da un numero, detto **PID** (**P**rocess **ID**entifier) oppure "process handle".

Ad un processo sono associate le seguenti strutture dati:

- Uno o più segmenti di codice.
- Uno o più segmenti di memoria dati.
- I descrittori di eventuali risorse in uso (file, finestre, periferiche, ecc.)
- Uno o più thread.

L'insieme di tali informazioni è raccolto o indicizzato da una struttura, unica per ogni processo, detta **process control block** (abbreviata in **PCB**). A loro volta, tutti i PCB sono elencati in una struttura detta **process table**.

Se il sistema operativo gestisce i thread, anche ciascun thread è identificato da un numero, detto **TID** (**T**hread **ID**entifier) oppure "thread handle".

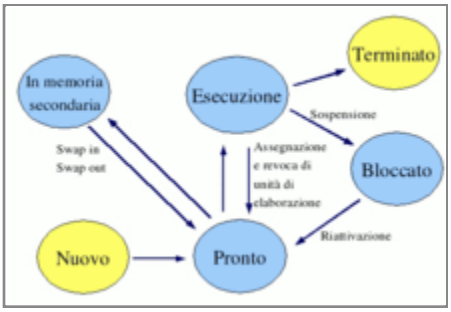
Ad un thread sono associate le seguenti strutture dati:

- Lo stack delle chiamate di funzione.
- I registri del processore, tra cui il program counter.

Se il sistema operativo non gestisce i thread, le suddette informazioni fanno parte dello stato del processo.

Stati di un processo

In un sistema operativo multitasking, ci sono più processi contemporaneamente in esecuzione. Di questi, al massimo un numero pari al numero di processori può avere effettivamente il controllo di un processore in un dato istante. I diversi processi possono quindi utilizzare il processore per un periodo limitato di tempo cioè a divisione di tempo, per questo motivo i processi vengono interrotti, messi in pausa e richiamati secondo i noti algoritmi di schedulazione, dando l'impressione all'utente di un processamento parallelo di questi.



Stati di un thread.

Gli stati in cui un processo si può trovare sono:

- **esecuzione** (**running**): il processo ha il controllo di un processore;
- **pronto** (**ready**): il processo è pronto ad essere eseguito, ed è in attesa che lo scheduler lo metta in esecuzione;
- **in attesa** o *sospeso* o *bloccato* (**suspended** o **blocked**): il processo ha eseguito una chiamata di sistema ed è fermo in attesa del risultato;

Con commutazione di contesto (*Context switch*) si indica il meccanismo tramite il quale un processo in esecuzione viene fermato (perché ha eseguito una chiamata di sistema o perché lo scheduler ha deciso di eseguire un altro processo), e un altro pronto viene messo in esecuzione.

Genesi di processi e thread

Al bootstrap del sistema operativo ci sono in esecuzione uno o più processi creati dal sistema operativo stesso. Durante l'avvio del sistema, in base alla configurazione, possono essere creati numerosi processi. Durante la normale operatività, in base alle richieste degli utenti, possono essere creati nuovi processi e altri possono terminare.

Quando il sistema operativo inizia l'esecuzione di un programma, crea un processo dotato di un solo thread. Durante l'esecuzione di tale thread, detto thread principale, il codice può creare altri thread o altri processi con apposite chiamate di sistema.

La creazione di un processo differisce tra i vari sistemi operativi. In ambiente Windows, si usa la chiamata di sistema "CreateProcess", con cui si specifica il nome del file contenente il programma eseguibile; tale file viene caricato in memoria ed eseguito. In ambiente Unix, si usa la chiamata di sistema "fork" per creare un nuovo processo figlio identico al chiamante (processo padre) eccetto che per il valore reso dalla chiamata stessa; e si usa poi la chiamata "exec" in uno dei due per caricare il codice eseguibile di un nuovo programma nel processo corrente, e mandarlo in esecuzione.

La creazione di un thread invece, è più uniforme. Infatti, sia la chiamata di sistema

"CreateThread" di Windows che la chiamata di sistema "thr_create" di Solaris (una variante di Unix) richiedono il passaggio dell'indirizzo di una routine, e della dimensione del nuovo stack, oltre ad altri parametri. La chiamata di sistema fa sì che venga eseguito il corpo della routine specificata, concorrentemente con il codice che il chiamante sta eseguendo.

Nei sistemi Unix-like, quando un processo termina, il sistema operativo provvede a liberarne le risorse occupate, tranne il PID e il Process Control Block (PCB). Il PCB rimane nella tabella dei processi (process table) per permettere al processo padre del processo terminato la lettura dell'exit status tramite la chiamata di sistema `wait()`. In seguito a tale chiamata anche il PID e il PCB vengono liberati per poter essere riutilizzati da altri processi. Fin quando non avviene tale chiamata il processo rimane nello stato di zombie.

Quando un processo termina prima di un proprio processo figlio, quest'ultimo diviene un cosiddetto processo orfano e, nei sistemi Unix-like, viene automaticamente adottato dal processo di sistema speciale init.

Bibliografia

- *Architettura dei Sistemi di Elaborazione, volume 1* (F. Baiardi, A. Tomasi e Marco Vanneschi, 1988, Franco Angeli Edizioni, ISBN 882042746X). Fondamenti, firmware, architetture parallele.
- *Architettura dei Sistemi di Elaborazione, volume 2* (F. Baiardi, A. Tomasi, Marco Vanneschi, 1987, Franco Angeli Edizioni, ISBN 882042746X) Sistemi operativi, multiprocessore e distribuiti.
- A.Tanenbaum, *I moderni sistemi operativi (seconda edizione)*, Jackson Milano, 2002, pp.66-144
- A.Tanenbaum, *I moderni sistemi operativi (terza edizione)*, Pearson Paravia, 2009

Voci correlate

- Kernel
- Multitasking
- Multithreading
- Oggetto (informatica)
- Sistema operativo
- Scheduler
- Task manager
- Thread (informatica)
- Unlocker

 **Portale Informatica**: accedi alle voci di Wikipedia che trattano di informatica

Categoria: Kernel

- Questa pagina è stata modificata per l'ultima volta il 28 apr 2014 alle 17:11.