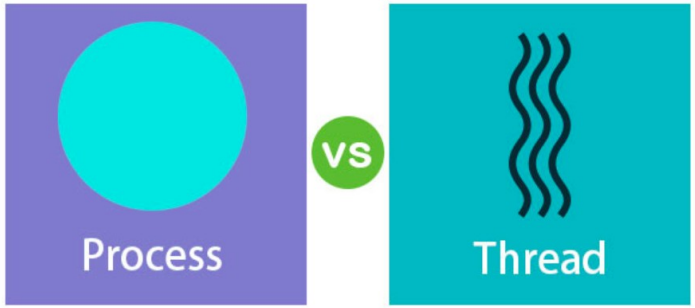
Processi VS Thread

Sistemi e Reti – A.S. 2022/2023
Prof. Verga Pierangelo
Prof.ssa Dalbesio Manuela

Processo o Thread?

Thread o Processo: quale usare? I sistemi operativi multi-tasking, in grado di gestire la concorrenza tra più task in esecuzione sullo stesso hardware, sono sempre più diffusi e trovano applicazione anche nei dispositivi con cui interagiamo tutti i giorni, come cellulari e smartphone. Parlando di multi-tasking, si usano spesso i termini **processo** e **thread**, ma qual è la differenza tra i due e quando usare l'uno o l'altro?



Processo

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.

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.

Processo

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.

PCB

I processi sono rappresentati dal PCB che contiene le informazioni seguenti:

- Stato del processo
- Dati identificativi (del processo, dell'utente)
- Program counter
- Registri della CPU
- Informazioni per lo scheduling della CPU
- Informazioni per la gestione della memoria
- Informazioni di utilizzo risorse
  - tempo di CPU, memoria, file. . .
  - eventuali limiti (quota)
- Stato dei segnali (per la comunicazione interprocess)

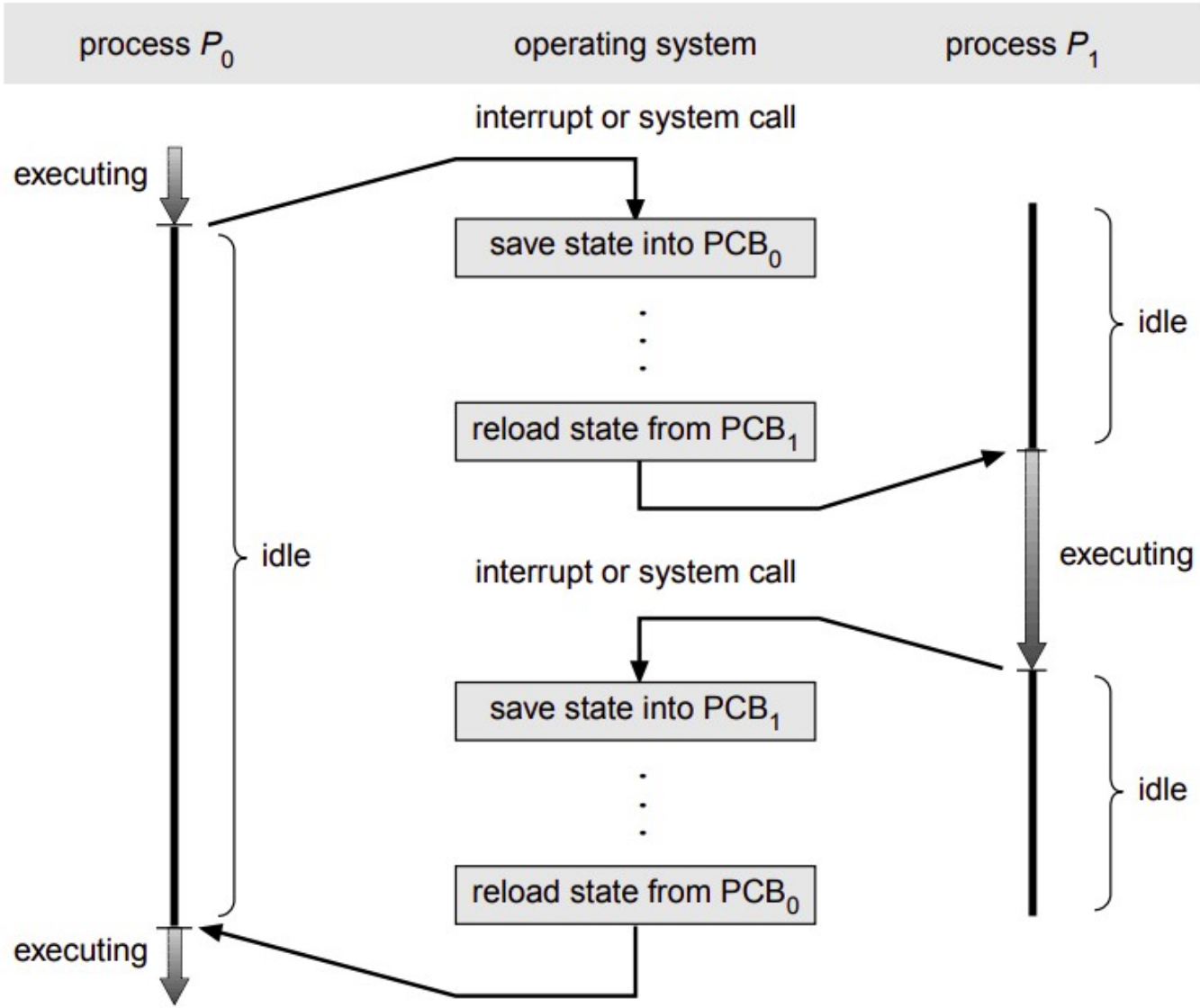
Processo

Un processo, durante la sua esistenza può trovarsi in vari stati:

Context-switch

Quando la CPU passa ad un altro processo, il sistema deve salvare lo stato del vecchio processo e caricare quello del nuovo processo.

- Lo stato del processo viene salvato nel Process Control Block (PCB)
- Il tempo di context-switch porta un certo overhead; il sistema non fa un lavoro utile mentre passa di contesto
- Può essere un collo di bottiglia per sistemi operativi ad alto parallelismo (migliaia - decine di migliaia di thread).


process P_0
operating system
process P_1
interrupt or system call
executing
save state into PCB_0
idle
reload state from PCB_1
executing
interrupt or system call
save state into PCB_1
idle
reload state from PCB_0
executing
idle

Context-switch

Le operazioni di scheduling, che determinano il passaggio da un processo ad un altro, sono onerose per il Sistema Operativo e dipendono essenzialmente da:

- **Frequenza** di cambio di contesto
- Dimensione **PCB**
- **Costo** dei trasferimenti da/verso la memoria

Thread

Un thread (o processo **leggero**) è un’unità di esecuzione che **condivide codice** e **dati** con altri thread ad esso associati.

I **thread** possono essere paragonati a dei processi "**light**", nel senso che offrono vantaggi analoghi a quelli dei processi, senza però richiedere le tecniche di comunicazione tipiche dei processi. I **thread** permettono di dividere il flusso di controllo principale di un programma in più flussi di controllo in esecuzione concorrente.

Thread

Ogni processo ha un proprio spazio di indirizzamento e delle risorse proprie; ne consegue che la comunicazione tra parti di codice in esecuzione su processi differenti può avvenire solo attraverso appositi meccanismi di gestione della concorrenza: pipe, code fifo, mailbox, aree di memoria condivise, ecc. I thread, viceversa, permettono di creare parti di programma in esecuzione concorrente, in cui ogni parte può accedere allo stesso spazio di indirizzamento, variabili, e costanti.

Thread

Per **thread** si intende un blocco di codice eseguibile all'interno di un sistema operativo che ha la capacità di essere schedulato.

Un thread richiede molta meno informazione di un processo, e quindi comporta un minore overhead per il Sistema Operativo.

Thread

Ogni processo ha almeno un thread (il **thread primario**, o principale) ma in genere dispone di thread multipli, ciascuno dei quali viene eseguito in modo concorrente ed indipendente dagli altri; in questo caso si parla di processo **multi-thread**. Tutti i thread in esecuzione all'interno di un processo condividono lo **stesso spazio** di **indirizzamento** e le **altre risorse**: i thread non hanno quindi risorse proprie, ma hanno ovviamente un proprio program counter, stack, e registri di stato. Gli stack dei vari thread saranno tutti contenuti all'interno del segmento stack del processo, mentre il segmento dati del processo verrà condiviso tra i thread.

Thread

I thread possono inoltre creare altri thread, all'interno dello stesso processo, e possono anche eseguire suspend, resume, o terminare altri thread dello stesso processo. Nei sistemi multi-processore, i thread appartenenti ad uno stesso processo possono essere eseguiti su processori diversi, ma solo sui processori assegnati a quel processo. Nei sistemi mono-processore, i thread sono costretti a competere per l'uso della risorsa condivisa. Quando poi più thread sono in esecuzione concorrente all'interno dello stesso processo, il processo stesso viene detto multi-thread.

Processo o Thread?

La principale differenza è che ogni processo ha il proprio spazio di indirizzamento, mentre il thread no. Quando un processo crea altri processi figli, questi avranno ciascuno il proprio spazio di indirizzamento ed una copia del segmento dati: se pertanto un processo figlio modifica parte dei suoi dati, questo non avrà conseguenze sui dati del processo padre; se si vogliono pertanto scambiare informazioni tra i processi, occorre ricorrere a meccanismi di inter-process-communication (IPC). Per contro, i thread appartenenti ad uno stesso processo possono comunicare e scambiarsi informazioni semplicemente accedendo ai dati del processo padre.

Thread - vantaggi

I principali **vantaggi** dei thread sono:

- Context-switch più veloce e meno oneroso per il sistema
- Miglioramento delle performance dell'applicazione (mentre un thread è sospeso in attesa di eseguire un'operazione di I/O, un altro può eseguire altre operazioni)
- Struttura del programma semplificata: l'architettura di un'applicazione può essere decomposta in vari task e ciascun task assegnato ad un thread, con la sua priorità
- Non richiedono meccanismi espliciti di IPC: i thread comunicano utilizzando la memoria condivisa nello spazio di indirizzamento del processo
- Controllo completo sugli altri thread dello stesso processo (un processo può controllare solo i suoi processi figli)

Thread - svantaggi

Gli **svantaggi** dei thread sono essenzialmente i seguenti:

- Possono corrompere l'area dati del processo di appartenenza
- Anche se non richiedono meccansimi di IPC, necessitano comunque di un'opportuna sincronizzazione quando accedono concorrentemente alle stesse aree di memoria
- Un thread può terminare altri thread, e tra questi anche il main thread del processo!
- Scarsa riusabilità del codice di un thread
- Se mal gestiti, possono causare il blocco/terminazione del processo: i thread non sono isolati come i processi, per cui un fatal access eseguito da un thread può mandare in crash l'intero processo

Thread - utilizzo

I thread possono eseguire parti diverse di una stessa applicazione.

Esempio: web browser

- thread che scrive il testo sul video
- thread che ricerca dati sulla rete

Esempio: word processing

- thread che mostra i grafici sul video
- thread che legge i comandi inviati dall'utente
- thread che evidenzia gli errori di scrittura

I thread possono eseguire le stesse funzioni (o funzioni simili) di un'applicazione.

Thread - utilizzo

Esempio: web server

- thread che accetta le richieste e crea altri thread per servirle.
- thread che servono la richiesta (possono essere uguali o diversi a seconda del tipo di richiesta).

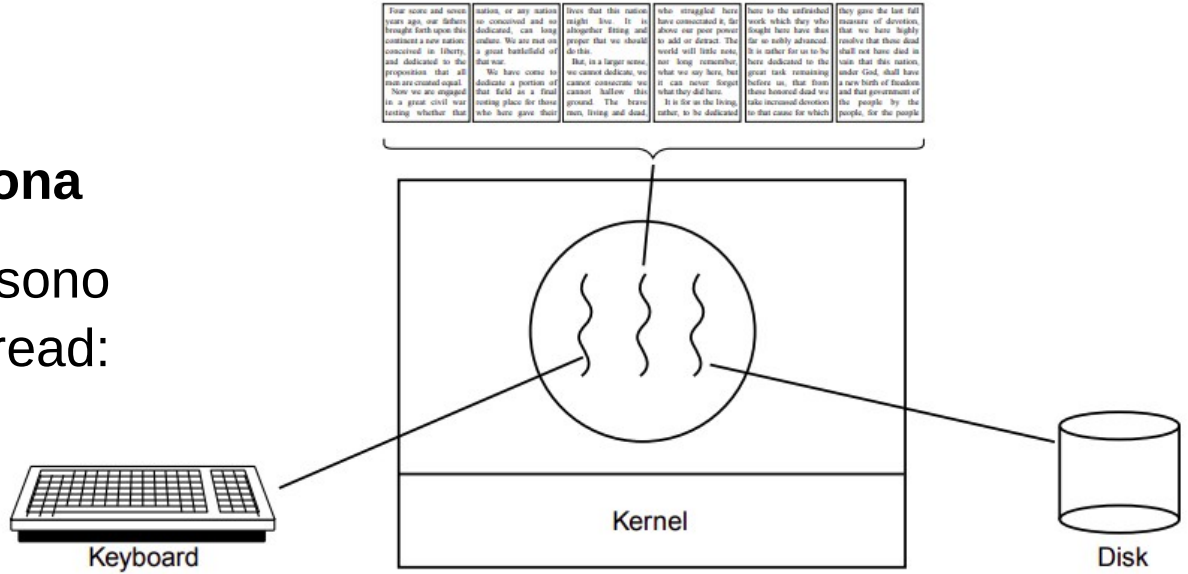
Esempio: lavoro foreground/background

Mentre un thread gestisce l'I/O con l'utente, altri thread operano sui dati in background. Spreadsheets (ricalcolo automatico), word processor (reimpaginazione, controllo ortografico...)

Thread - utilizzo

Esempio: elaborazione asincrona

Operazioni asincrone possono essere implementate come thread: salvataggio automatico.

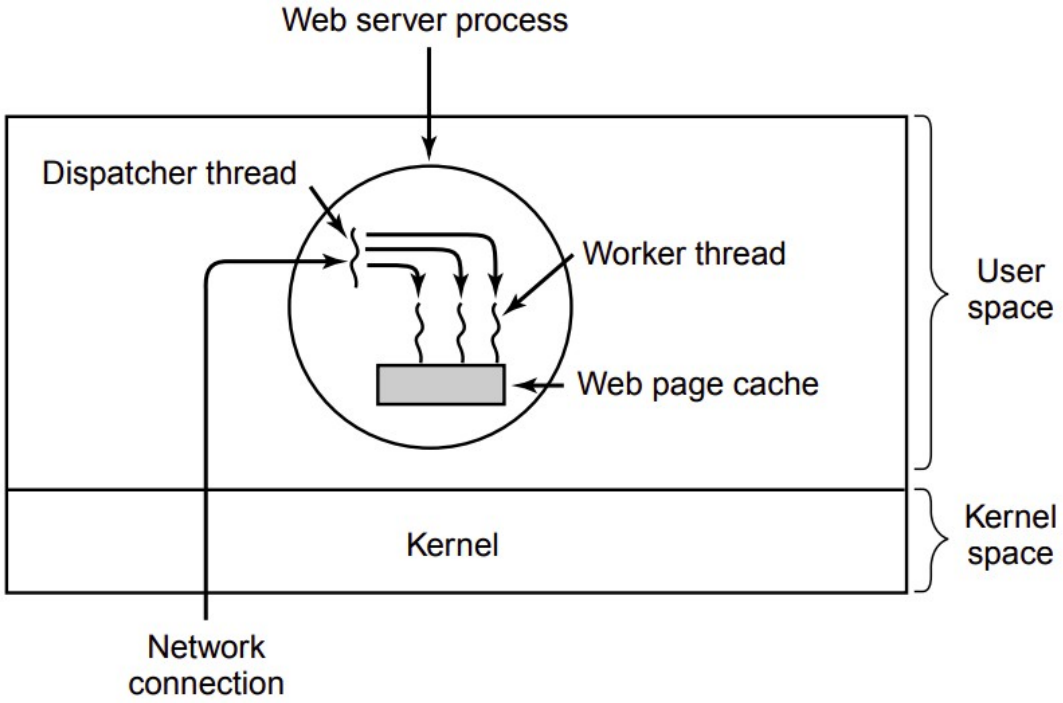


Thread - utilizzo

Esempio: Task intrinsecamente paralleli

Vengono implementati ed eseguiti più efficientemente con i thread.

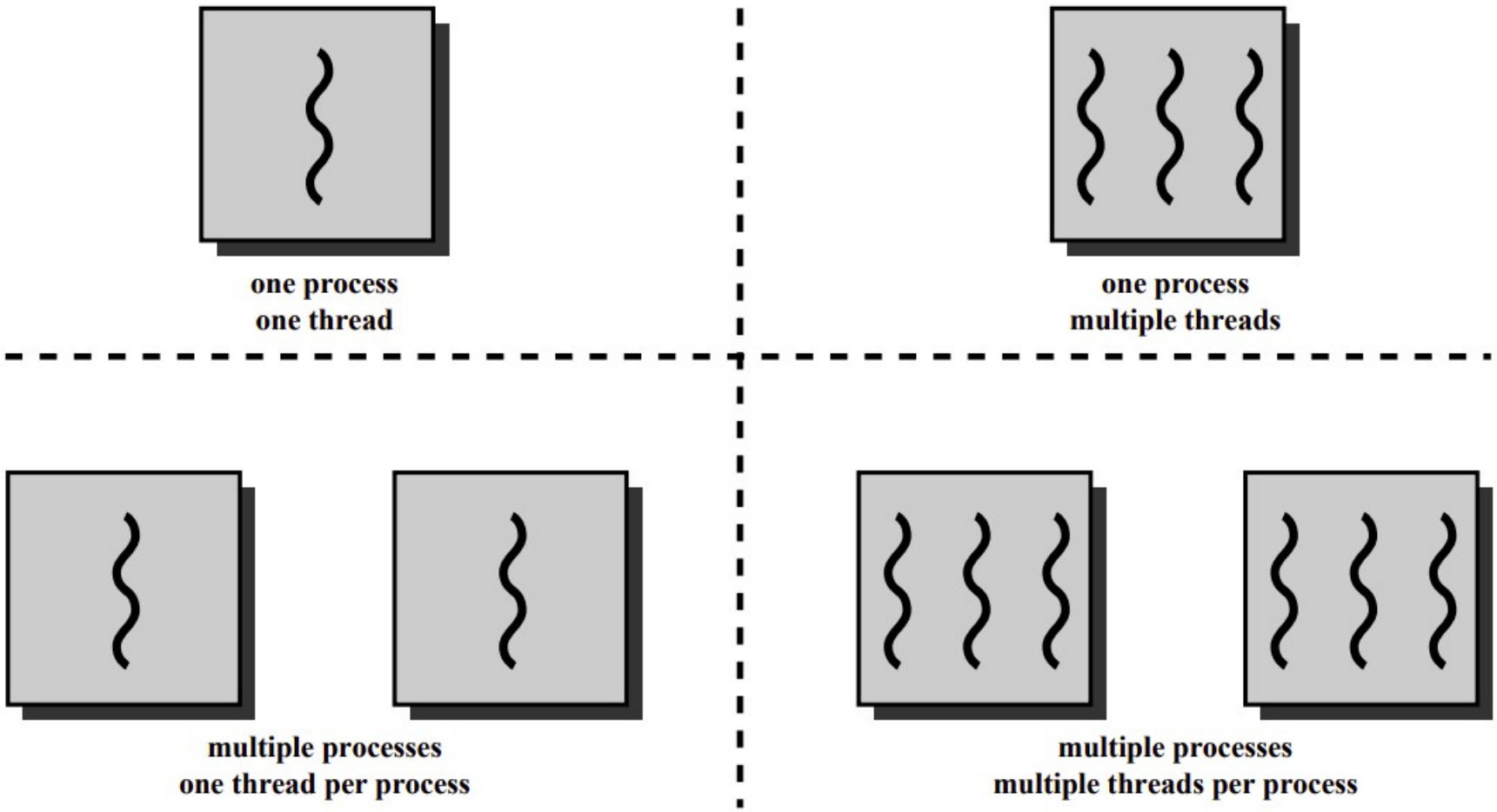
Es: file/http/dbms/ftp server...



Stati e operazioni sui thread

- Stati: running, ready, blocked.
- Operazioni sui thread:
  - creazione (spawn): un nuovo thread viene creato all'interno di un processo (thread_create), con un proprio punto d'inizio, stack…
  - blocco: un thread si ferma, e l'esecuzione passa ad un altro thread/processo. Può essere volontario (thread_yield) o su richiesta di un evento;
  - sblocco: quando avviene l'evento, il thread passa dallo stato "blocked" al "ready"
  - cancellazione: il thread chiede di essere cancellato (thread_exit); il suo stack e le copie dei registri vengono deallocati.
- Meccanismi per la sincronizzazione tra i thread (semafori, thread_wait): indispensabili per l'accesso concorrente ai dati in comune

Processi e Thread: quattro possibili scenari



Riferimenti

- https://person.dibris.unige.it/delzanno-giorgio/SO1/cap4.pdf
- https://vitolavecchia.altervista.org/caratteristiche-e-differenza-tra-processo-e-thread-in-informatica/
- https://it.emcelettronica.com/thread-o-processo-quale-usare
- http://lia.deis.unibo.it/Courses/sola0708-info/lucidi/2-ProcessiThread(2x).pdf