

Titolo

Daniele De Micheli

2019

Indice

I	Prima parte	1
1	Processi	1
1.1	Concetto di processo	1
1.1.1	Process Control Block	3
1.1.2	Threads	4
1.1.3	Scheduling dei processi	4
1.2	Operazioni sui processi	7
1.2.1	Creazione di processi	7

Parte I

Prima parte

1 Processi

1.1 Concetto di processo

I processi rappresenta la prima e più importante astrazione a livello software per un sistema operativo. Un SO esegue infatti un certo numero di programmi contemporaneamente; ogni programma rappresenta un **processo**, e questi processi vengono eseguiti in maniera sequenziale. Un processo è composto da diverse parti:

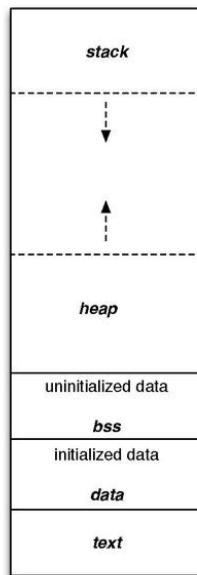
- Lo stato dei registri del processore, incluso il program counter.
- Il codice del programma (*text section*) - PID -.

- Lo **stack** delle chiamate, contenente parametri, variabili locali e indirizzo di ritorno (compreso lo *stack pointer*).
- La **data section**, contenente le variabili globali.
- Lo **heap**, contenente la memoria allocata dinamicamente durante l'esecuzione. Per esempio, in Java viene indicata con *New*, in C con *malloc*.
- Altre risorse acquisite (es. file aperti).

Un programma è un'entità passiva (file eseguibile su disco), un processo è un'entità attiva (è un programma in esecuzione). Un programma "diventa" un processo quando viene caricato nella memoria centrale. Esso può generare diversi processi:

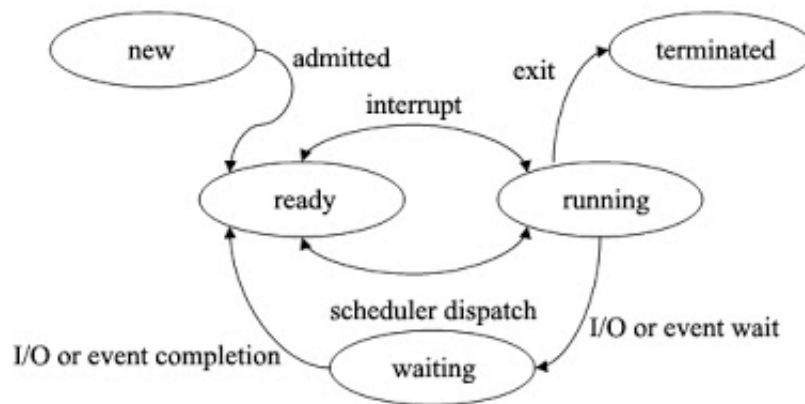
1. Molti utenti eseguono lo stesso programma
2. Uno stesso programma ...

La memoria di un processo è divisa tra stack e heap. Dopo lo heap c'è la sezione **data** (e in linux anche la sezione **bss**) e successivamente la sezione **text**.



Durante l'esecuzione un processo può trovarsi in diversi *stati*. Gli stati possibili sono:

- Nuovo (new): il processo è creato, ma non è ancora ammesso all'esecuzione.
- Pronto (ready): il processo può essere eseguito.
- In esecuzione (running): le sue istruzioni sono in esecuzione su un processore.
- In attesa (waiting): il processo non è in esecuzione perchè sta aspettando un evento (es. input utente..).
- Terminato (terminated): il processo ha terminato l'esecuzione.



1.1.1 Process Control Block

Detto anche "Task Control Block", contiene le informazioni relative ad un processo:

- Process state: ready, running...
- Program number (o PID): identifica il processo
- Program counter: contenuto del registro "istruzione successiva"
- Register: contenuto dei registri del processore
- Informazioni di scheduling: priorità, puntatori a code di scheduling..
- Informazioni relative alla gestione della memoria: memoria allocata al processo

- Informazioni di accounting: CPU utilizzata, tempo trascorso..
- Informazioni su I/O: dispositivi assegnati al processo, elenchi file aperti...

pointer	process state
process number	
program counter	
registers	
memory limits	
list of open files	
⋮	

1.1.2 Threads

Fino ad ora abbiamo assunto che un processo abbia un singolo flusso di esecuzione sequenziale. Supponiamo che si possano avere molti program counter per un singolo processo:

- 1.

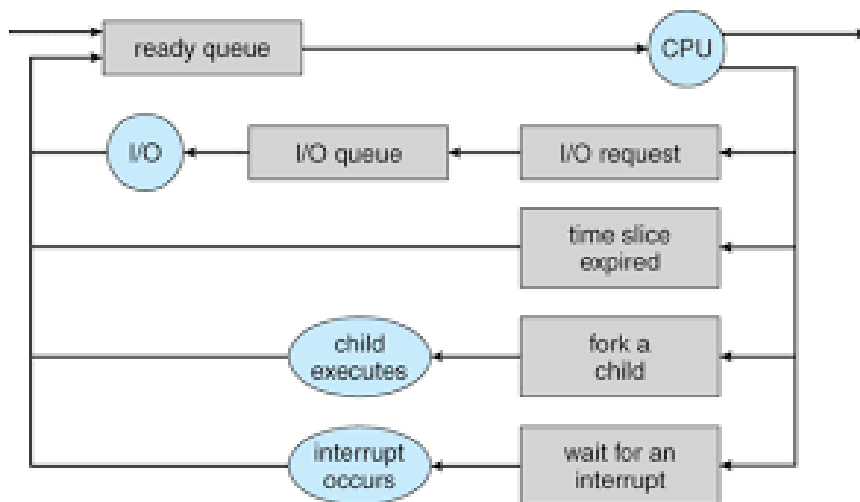
1.1.3 Scheduling dei processi

L'obiettivo dello scheduling dei processi è quello di massimizzare l'utilizzo della CPU. Una tecnica per fare questo è quella del *Time-sharing*:

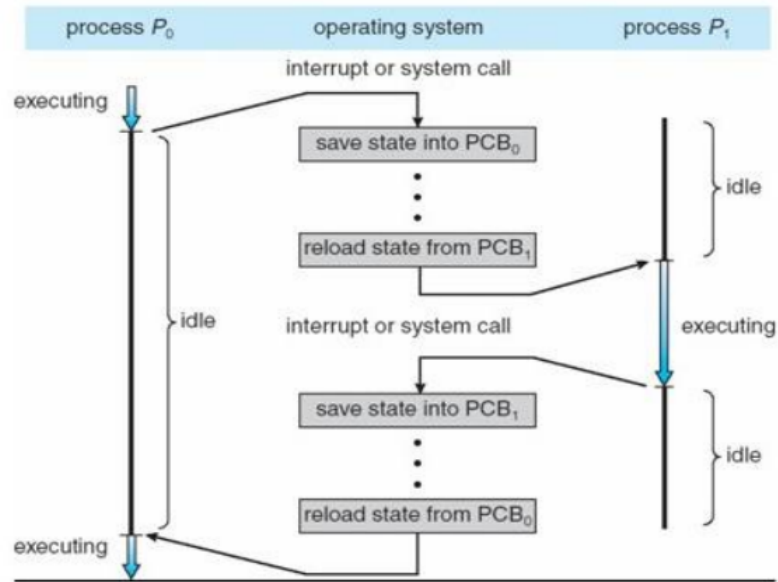
Lo scheduler dei processi sceglie il prossimo processo da eseguire tra quelli in stato ready. Ci sono diverse code di processi:

- Ready queue: processi residenti in memoria
- Wait queue: diverse code per i processi in attesa

Durante la loro vita i processi migrano tra una coda e l'altra.



Quando un SO decide che si deve cambiare processo, si ha la **commutazione di contesto** (o *context switch*). Quando la CPU passa ad eseguire un processo diverso, il sistema operativo deve salvare lo stato del processo precedente, e caricare lo stato salvato del processo da rieseguire attraverso un context-switch. Il PCB rappresenta il contesto di un processo. Il tempo necessario per il context switching è puro overhead: non viene eseguito alcun lavoro utile. Più è complesso l'SO, più è complesso cambiare processo per il context-switch.



Multitasking nei sistemi mobili Alcuni sistemi mobili (es. le prime versioni di iOS) permettevano solo ad un processo di essere in esecuzione. Da iOS4 è possibile avere un processo in esecuzione in **foreground** (ha lo schermo a disposizione) e un certo numero di processi in esecuzione in **background** (senza schermo), ma con dei limiti. Android ha molti meno limiti: i processi in background che vogliono effettuare delle elaborazioni devono creare opportuni *servizi*, che:

- non hanno interfaccia utente
- possono usare un ridotto contenuto di memoria
- possono continuare a funzionare anche quando l'app in background è sospesa

L'aumento di potenza dei sistemi mobili rende i loro OS sempre più simili a quelli non mobili.

1.2 Operazioni sui processi

1.2.1 Creazione di processi

Di solito nei sistemi operativi i processi sono organizzati in maniera gerarchica:

- un processo (padre) può creare diversi processi (figli) fino a creare un *albero di processi*.
- PORCODDIO PERCHECAZZO VA COSI VELOCE

Sistemi operativi diversi creano processi in modo diverso. Possono esistere diverse politiche di condivisione (padre e figlio condividono le risorse, solo alcune, nessuna), diverse politiche di creazione di spazio di indirizzi (il figlio è un duplicato del padre (stessa memoria e programma, oppure il figlio deve eseguire qualcos'altro) e ancora politiche di coordinazione padre/figli (il padre è sospeso finchè i figli non terminano, oppure eseguono in maniera concorrente).

Esempio: sistema UNIX

