Azzolini Riccardo 2019-10-23

Processi

1 Tipi di processi

In ambito UNIX, esistono tre tipi di processi:[1]

- **user process**;
- **daemon process**;
- **kernel process**.

1.1 User process

Uno user process è associato a un utente che lavora a un terminale. Questo tipo di processo esegue con la CPU in modalità user, quindi ha bisogno di usare le system call per richiedere i servizi del SO.

La maggior parte dei processi, tra cui la **shell** e tutti i programmi avviati dall'utente, appartengono a questa categoria.

Nota: Il termine "user process" indica che il processo è associato a un utente, e non semplicemente che esegue in modalità user: tutti gli user process eseguono in modalità user, ma *non tutti i processi che eseguono in modalità user sono user process*.

1.1.1 Shell

La shell è uno user process che è normalmente in waiting, in attesa dell'input dell'utente. Quando viene inserito un comando, la shell avvia il programma specificato. Supponendo, per esempio, che tale programma sia un eseguibile chiamato "a.out", la shell può eseguirlo in due modi diversi:

- Esecuzione classica (comando `a.out`):
  1. la shell fa una *fork*;
  2. il figlio fa la *exec* su a.out, mentre il padre (la shell) fa la *wait* per attendere la terminazione del figlio;

[1] Negli altri SO si ritrovano sostanzialmente gli stessi concetti, anche se con nomi diversi.

3. quando il figlio termina, il padre si mette in attesa di un altro comando da parte dell'utente.

- Esecuzione in background (comando `a.out &`): *fork* ed *exec* vengono fatte come per l'esecuzione classica, ma poi la shell non aspetta la terminazione del figlio (cioè non fa la *wait*), e invece si mette subito in attesa di un altro comando.

1.2 Daemon process

I daemon process sono processi che non sono associati a nessun utente. Essi:

- tipicamente non terminano;
- svolgono operazioni di routine, spesso vitali per il sistema, come ad esempio il controllo delle email, la gestione della coda di stampa, ecc.;
- solitamente passano una parte significativa della loro vita in stato di waiting;
- eseguono in modalità user, quindi devono usare le system call.

Un esempio è il processo *getty*, che:

1. gestisce il processo di login di un utente al terminale;
2. se questo va a buon fine, avvia la shell (mediante una *fork* e una *exec*) e si mette in attesa che essa termini (*wait*).

1.3 Kernel process

I kernel process sono daemon che eseguono in modalità kernel,[2] quindi:

- possono accedere alle procedure e alle strutture dati del kernel senza bisogno di invocare le system call;
- sono "potentissimi", in quanto possono controllare i propri parametri di scheduling, ecc.

Alcuni esempi sono:

- Lo **swapper process**, che nei sistemi con swapping viene svegliato a intervalli regolari per fare:
  - lo swap out (in memory → swapped) dei processi con maggiore anzianità in memoria;

[2] Ciò *non* significa che, quando la CPU è in modalità kernel, sia sempre in esecuzione un kernel process, perché esistono procedure del SO che non fanno parte di un processo, come ad esempio gli interrupt handler.

  - lo swap in (ready swapped → ready in memory) dei processi che sono da più tempo in stato ready swapped.

- Lo **stealer process**, che nei sistemi con paginazione della memoria viene svegliato a intervalli regolari e fa lo swap out delle pagine che recentemente non hanno avuto accessi (ma non fa swap in, perché le pagine vengono caricate su richiesta quando servono).

Al momento del boot, deve essere eseguito del codice di inizializzazione che "costruisca" almeno un processo, dal quale possono poi discendere tutti gli altri. Per esempio, nel caso di UNIX System V:

1. al momento del boot viene creato il **processo 0**, che è lo swapper (e quindi è un kernel process);
2. lo swapper crea il **processo 1**, detto **init**, che si cambia il bit PM a 0, e diventa così un daemon "normale" (non kernel);
3. tutti gli altri processi vengono creati dallo swapper e da init:
   - lo swapper crea altri processi kernel (come per esempio lo stealer, nei sistemi con paginazione), e, nei sistemi con swapping, si occupa anche di quest'ultimo (come descritto sopra);
   - il processo init crea altri daemon (tra cui un *getty* per ogni terminale) e usa la *wait* per mettersi in stato di waiting finché questi non terminano.

Nei sistemi moderni, che usano la paginazione invece dello swapping, il processo swapper esiste ancora, ma non ha più la sua funzione originale (serve solo per il boot e la creazione dei kernel process), e ha mantenuto lo stesso nome solo per tradizione.

2 Terminazione dei processi

La **terminazione** di un processo è effettuata sempre dal SO, ma può avvenire per vari motivi:

- Il processo stesso può richiedere di terminare mediante un'apposita system call (ad esempio, **exit** in UNIX e **ExitProcess** in APIWin32).
- La terminazione può essere forzata dall'interrupt handler delle eccezioni (ad esempio, in seguito a una violazione di protezione di memoria).
- Un processo può richiedere di farne terminare altri (ad esempio, con la system call **kill** in UNIX, o con **TerminateProcess** in APIWin32). Tale operazione va a buon fine solo quando il "killer" ha il diritto di eseguire l'operazione. Ad esempio, in UNIX, il padre può sempre uccidere il figlio.

Quando un processo termina:

- *rilascia tutte le risorse*: CPU (non è più schedulabile), memoria, file, ecc.;
- i suoi eventuali figli possono essere o meno terminati, a seconda del sistema operativo (sia in Windows che in UNIX i figli non vengono uccisi, e in particolare, in UNIX, diventano figli del processo *init*);
- la distruzione del PCB non è sempre immediata (ad esempio, in UNIX, il PCB rimane in stato *zombie* finché non è cancellato dal padre).

Osservazione: Non tutti i processi terminano. Ad esempio, i daemon sono solitamente pensati per non terminare mai, cioè per rimanere in esecuzione fintanto che la macchina è accesa.