Azzolini Riccardo 2019-10-09

Processi

1 Processo

Definizione: Un **processo** è un'*esecuzione di un programma*, o, detto in modo diverso, un'*istanza di un programma in esecuzione*.

Osservazioni:

- Il programma è un'entità passiva, che non esegue nessuna azione di per sé.
- L'esecuzione del programma, chiamata appunto processo, concretizza le azioni specificate nel programma.
- Le entità che vengono schedulate dal SO sono i processi, non i programmi.
- Nei SO con multiprogrammazione, più processi possono coesistere ed eseguire "contemporaneamente".
- Si possono avere *più processi* relativi al *medesimo programma*, cioè più esecuzioni, anche "in contemporanea", del programma. Ognuna di queste non deve interferire con le altre, quindi ogni processo deve poter accedere solo alle proprie aree di memoria, ecc.

2 Componenti di un processo

- Le aree di testo, dati e stack,
- le risorse logiche (es. file) e fisiche (es. dispositivi),
- lo stato della CPU, cioè il contenuto dei registri (generali e di controllo)

sono *associati al processo*, non al programma. Si può quindi dare un'altra definizione di processo.

Definizione: Un **processo** consiste delle seguenti componenti:

- **stato della CPU**;
- **area di testo**;
- **area dati**;

- **area di stack**;
- **risorse logiche e fisiche** assegnate al processo.

Osservazione: Se si esegue uno stesso programma più volte, tutti i processi risultanti hanno aree di testo uguali, perché l'area di testo di un processo non è modificabile durante l'esecuzione.

3 Parallelismo e concorrenza

Definizione:

- Due eventi sono **paralleli** se occorrono nello *stesso momento*. Due attività sono parallele se vengono portate avanti contemporaneamente.
- La **concorrenza** è l'*illusione del parallelismo*. Due attività sono concorrenti se si ha l'illusione che vengano eseguite in parallelo, mentre, in realtà, in ogni singolo istante viene eseguita solo una di esse.

Su un sistema con una singola CPU, il SO può assegnarla "a turno" ai vari processi (**interleaving**), realizzando così l'**esecuzione concorrente** di tali processi: il parallelismo è solo simulato, perché in ogni istante la CPU può essere usata al massimo da un solo processo. Se, però, è presente un DMA controller, possono avvenire in parallelo l'esecuzione di un processo e un'operazione di I/O richiesta da un altro processo, ma questo non è comunque un parallelismo che riguarda la CPU e l'"avanzamento" dei processi.

3.1 Velocità di avanzamento dei processi

- La velocità di avanzamento di un processo (la velocità con la quale "aumenta il suo Program Counter", cioè vengono eseguite le sue istruzioni), *non è uniforme nel tempo*, perché in alcuni momenti esso dispone della CPU, mentre in altri no (e quindi non avanza).
- La velocità di avanzamento di un processo dipende dal numero e dal comportamento (CPU-bound o I/O-bound) di tutti i processi presenti, quindi *non è riproducibile*: esecuzioni diverse dello stesso programma con gli stessi dati in input possono richiedere tempi diversi.
- Anche la velocità di avanzamento relativa dei processi (cioè come avanzano l'uno rispetto all'altro) non è riproducibile.

3.2 Sistemi con più CPU

Se sono disponibili più CPU, il parallelismo può essere reale, perché più processi possono eseguire contemporaneamente su CPU diverse. Si distingue tra due casi:

multiprocessing: più processi possono eseguire su una macchina dotata di più CPU;

distributive processing: più processi possono eseguire su più macchine distribuite e indipendenti.

Comunque, tale parallelismo è limitato dal numero di CPU: se il numero di processi è maggiore, si ha ancora concorrenza.

Questi sistemi risultano più efficienti, ma richiedono SO più complessi.

4 Stati di un processo

In ogni istante, ogni processo si trova in un certo **stato**, che indica l'attività che tale processo sta svolgendo. Tutti i SO devono considerare almeno tre stati diversi:[1]

Running: il processo è in esecuzione sulla CPU. In un sistema con un singolo processore, ci può essere al massimo un processo running alla volta, oppure nessuno, quando viene eseguito il codice del SO, come ad esempio un interrupt handler. Per convenzione, però, quando un processo viene interrotto da un interrupt, esso si considera running anche durante l'esecuzione dell'handler e dello scheduler, finché quest'ultimo non seleziona eventualmente un processo diverso da eseguire.

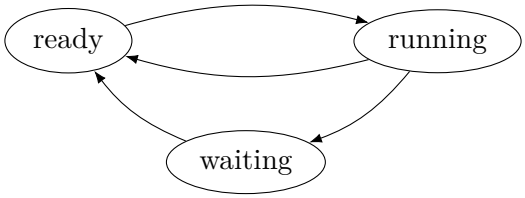
Ready: il processo non è in esecuzione, ma sarebbe in grado di eseguire se ottenesse la CPU. Ciò avviene quando il numero di processi che potrebbero eseguire è maggiore del numero di CPU disponibili.

Waiting: il processo non è in esecuzione, e *non* sarebbe in grado di eseguire se ottenesse la CPU, ma acquisirà la capacità di eseguire se e quando si verificherà un determinato evento (ad esempio, il completamento di un'operazione di I/O), che il processo sta quindi aspettando. Questo stato è chiamato anche *blocked* o *sleeping*.

4.1 Transizioni

Ci sono 4 possibili transizione da stato a stato, determinate da eventi che avvengono nel sistema.

[1] Alcuni SO prevedono stati aggiuntivi, e/o suddividono alcuni di questi tre stati in più sottostati.



- ready → running: lo scheduler seleziona il processo per l'esecuzione, e assegna a esso la CPU. Se ci sono più processi ready, la scelta viene effettuata in base alla politica di scheduling. Il processo non ha modo di forzare lo scheduler, cioè non può richiedere di essere schedulato in un certo momento, o con priorità rispetto ad altri processi.
- running → ready: viene sottratta la CPU al processo, nonostante esso sia in grado di continuare a eseguire, per assegnarla a un altro. Ciò avviene:
  - quando il processo subisce una *preemption*, perché è diventato ready un processo con priorità maggiore;
  - in un sistema con timesharing, quando *scade il time slice* del processo.

  In entrambi i casi, il processo subisce l'evento passivamente, senza poter fare nulla per opporsi.
- running → waiting: il processo si trova impossibilitato a eseguire, e si blocca in attesa di un *evento sbloccante*. Ciò avviene, ad esempio, quando il processo richiede un'operazione di I/O: in tal caso, esso non può sfruttare la CPU finché l'operazione non è completata, e questa viene effettuata dal DMA mentre la CPU esegue altri processi.
- waiting → ready: si verifica l'evento sbloccante atteso dal processo. A questo punto, in base alla politica di scheduling, agli altri processi presenti, ecc., potrebbe capitare che il processo venga immediatamente schedulato, cioè che la transizione waiting → ready sia seguita subito da ready → running, ma non è detto. Ciò succede, ad esempio, se lo scheduler è basato sulle priorità e il processo diventato ready ha priorità maggiore di quello running, che allora subisce una preemption, perdendo la CPU e andando quindi in ready.