Azzolini Riccardo 2019-09-19

Breve storia dei Sistemi Operativi

1 Generazione 3: circuiti integrati

In questa generazione, i calcolatori vengono ancora usati per calcoli scientifici, ma sono anche impiegati per l'elaborazione di dati commerciali (che in genere richiede meno CPU ma più capacità di I/O).

Vengono introdotti:

- dispositivi: hard disk e terminali (un terminale è costituito da tastiera e video);
- software: editor di testo e linguaggi ad alto livello (C, script di shell, ecc.).

I SO di questa generazione sono **interattivi**, con **multiprogrammazione** o **timesharing**.

2 Multiprogrammazione

In un SO che supporta la multiprogrammazione, la user area viene partizionata, in modo da poter caricare in memoria più job contemporaneamente.[1]

Quando il job in esecuzione effettua un'operazione di I/O,[2] la CPU viene assegnata a un altro job, evitando così che rimanga inutilizzata.

Perché ciò sia possibile, l'hardware deve consentire a CPU e dispositivi di lavorare in parallelo.

[1] Complessivamente, la memoria si suddivide in una partizione per il sistema (system area), una partizione per ciascun job, ed eventualmente una parte di spazio libero.

[2] Alcuni esempi di operazioni di I/O sono la lettura/scrittura di dati sull'hard disk, la lettura di dati da tastiera, e la visualizzazione di dati sullo schermo. In generale, le operazioni di I/O richiedono molto più tempo rispetto ai calcoli e agli accessi alla RAM.

2.1 CPU scheduling

Quando un job esegue un'operazione di I/O, il SO deve determinare a quale altro job assegnare la CPU, scegliendo tra quelli caricati in memoria e non impegnati in attività di I/O.

A tale scopo, è necessario stabilire una **politica di scheduling**, in base alla quale vengono effettuate tali scelte.

2.2 Gestione della memoria

Ogni job deve poter accedere ai propri dati, ma non a quelli degli altri.

Un esempio di tecnica banale per la protezione della memoria prevede che la CPU sia dotata di due appositi registri:

- **Lower Bound Register** (**LBR**);
- **Upper Bound Register** (**UBR**).

Prima di eseguire un'istruzione di accesso alla memoria, la CPU controlla che l'indirizzo interessato sia compreso tra gli indirizzi contenuti in LBR e UBR: se non lo è, si genera un segnale hardware che viene usato per fermare l'esecuzione del programma.

Le istruzioni per la modifica dei valori di LBR e UBR sono privilegiate (disponibili solo in modalità kernel), dato che tali registri devono poter essere gestiti solo dal SO:

- durante l'esecuzione di un job, essi sono impostati in modo da consentire solo l'accesso alla partizione di memoria assegnata a tale job;
- quando cambia il job in esecuzione, il SO modifica i registri, impostando i valori corrispondenti all'area di memoria del nuovo job.

2.3 Gestione dei dispositivi

Ogni job deve poter usare i dispositivi di I/O, senza interferire sull'uso degli stessi da parte degli altri job.

Ci sono due principali modalità per la gestione dei dispositivi:

Partitioning: I dispositivi vengono assegnati staticamente a ciascun job (ad esempio, un job potrebbe avere una porzione riservata del disco). Questo metodo è semplice, ma porta a un uso poco efficiente dei dispositivi.

Pooling: I dispositivi vengono assegnati dinamicamente ai job. Con questo metodo si ha un uso più efficiente dei dispositivi, ma il SO è più complesso.

2.4 Requisiti hardware

- L'hardware deve consentire alla CPU e ai dispositivi di I/O di lavorare contemporaneamente. Ciò è possibile sfruttando il **DMA** (**Direct Memory Access**), che permette ai dispositivi di accedere alla RAM senza "passare" dalla CPU.
- Quando un dispositivo di I/O termina di eseguire un'operazione, il job che l'ha richiesta deve essere reinserito nell'elenco dei job schedulabili. Per fare ciò si ricorre alla tecnica dell'**interrupt**: si sospende temporaneamente il normale funzionamento della CPU per promuovere il job da "non schedulabile" a "schedulabile".

3 Throughput e classificazione dei programmi

Si definisce **throughput** il rapporto tra il numero di programmi eseguiti e il tempo necessario a eseguirli.

Un programma si dice:

- **CPU-bound** se fa uso intenso della CPU, ma effettua poco I/O;
- **I/O-bound** se, invece, effettua tanto I/O e usa poco la CPU.

3.1 Ottimizzazione del throughput

Per ottimizzare il throughput,[3] lo scheduler dovrebbe privilegiare i programmi I/O-bound rispetto a quelli CPU-bound, così che siano frequenti gli usi concorrenti di CPU e I/O.

A tale scopo, in un sistema multiprogrammato si possono introdurre i concetti di:

program priority: il SO assegna a ogni programma una priorità, che viene poi considerata in fase di scheduling per scegliere a quale di essi assegnare la CPU;

preemption: in alcuni casi, la CPU viene sottratta forzatamente al programma in esecuzione, anche se esso non esegue operazioni di I/O, in modo da poter essere assegnata a un altro (in genere, uno con priorità maggiore).

Così, per migliorare il throughput:

- si assegna ai programmi CPU-bound una priorità inferiore rispetto a quelli IO-bound;

[3] Non è detto che la priorità di un sistema operativo sia massimizzare il throughput, ma lo era quando è stata introdotta la multiprogrammazione, perché al tempo i calcolatori erano ancora piuttosto costosi, e dovevano quindi essere sfruttati il più possibile. In generale, obiettivi/esigenze diversi richiedono politiche di scheduling diverse.

- quando un programma I/O-bound termina un'operazione di I/O, se la CPU è assegnata a un programma CPU-bound, quest'ultimo subisce la preemption.

4 Spooling

Con l'introduzione degli hard disk, la necessità di macchine dedicate alla preparazione dei batch e alla stampa dei risultati viene eliminata grazie al meccanismo dello **spooling** (dall'acronimo SPOOL: Simultaneous Peripheral Operation On Line):

- i programmi possono essere caricati su disco;
- appena un job termina e libera una partizione di memoria, il SO la assegna a uno dei job su disco, che allora può essere eseguito;
- per migliorare il throughput, conviene avere in memoria sia programmi CPU-bound che programmi I/O-bound;
- anche i dispositivi di output (come ad esempio le stampanti) possono essere gestiti in modo analogo.