Azzolini Riccardo 2019-09-18

Breve storia dei Sistemi Operativi

1 Importanza

Sapere la storia dei SO è utile perché certe scelte/caratteristiche/funzionalità dei sistemi attuali sono più facili da capire conoscendo:

- il contesto storico nel quale sono state introdotte;
- quanto accadeva prima che fossero introdotte;
- perché siano state introdotte.

2 Evoluzione dei SO

L'evoluzione dei SO è stata influenzata dai progressi dell'hardware, ma al tempo stesso ha anche guidato tale progresso: da nuove esigenze del software sono spesso nate nuove esigenze a livello hardware.

Risulta quindi utile studiare la storia dei SO considerando quattro generazioni di computer:

1. 1945–1955: *tubi a vuoto*;
2. 1955–1965: *transistor*;
3. 1965–1980: *circuiti integrati*;
4. dal 1980: *circuiti integrati LSI/VLSI*, *PC*.

3 Generazione 1: tubi a vuoto

In questo periodo, i calcolatori vengono utilizzati per svolgere calcoli scientifici, e sono *progettati, costruiti, e programmati da uno stesso gruppo di persone*.

Questi calcolatori *non hanno sistemi operativi*: siccome i programmatori conoscono i dettagli dell'architettura, possono programmare in linguaggio macchina e interagire direttamente con l'hardware, non esiste l'esigenza di una macchina virtuale più facile da utilizzare.

Non essendoci un SO, può essere eseguito un solo programma alla volta, e questo:

- ha a disposizione tutte le risorse fisiche;
- deve crearsi le risorse logiche;
- deve controllare tutti i dispositivi fisici, senza intermediari.

4 Generazione 2: transistor

Le macchine di questa generazione sono ancora impiegate prevalentemente per calcoli scientifici, ma vengono prodotte in serie (seppure in pochi esemplari) e vendute. Si distinguono quindi i ruoli di progettista, costruttore, operatore, manutentore, e programmatore (che è l'utente finale). Di conseguenza, i programmatori non conoscono più i dettagli dell'hardware, e allora, per semplificare il loro lavoro, vengono introdotti i linguaggi *assembly* e *Fortran*, che permettono di non dover effettuare lo sviluppo direttamente in linguaggio macchina.

I dispositivi utilizzati per l'input e l'output sono prima i *lettori di schede perforate* e le *stampanti*, e poi anche i *nastri magnetici*.

Per un periodo iniziale, questi calcolatori non sono dotati di SO, mentre in seguito vengono introdotti i **sistemi batch** (detti anche *monitor residenti* o *batch monitor*).

5 Sistema batch

Prima dell'avvento dei SO, per eseguire un programma è necessario:

1. incidere il programma sorgente su schede perforate;
2. usare il lettore di schede per caricare in memoria il programma (e, se quest'ultimo è in assembly/Fortran, caricare anche l'assembler/compilatore);
3. se il programma è in assembly/Fortran, eseguire l'assembler/compilatore per ottenere l'eseguibile;
4. eseguire il programma eseguibile;
5. raccogliere l'output dalla stampante.

Un sistema batch, invece, permette di caricare più programmi alla volta ed eseguirli automaticamente uno dopo l'altro. Ciascun programma (scritto su schede perforate) è detto **job**, e un gruppo di programmi da eseguire in sequenza, memorizzato su un nastro magnetico, si chiama **batch**.

Alcune schede, presenti in ciascun job, contengono delle *indicazioni al SO* ("carica il compilatore", "esegui", ecc.), che vengono trattate dal **command processor** del SO.

Per eseguire dei programmi con un sistema batch:

1. Un batch di job viene caricato su un nastro magnetico usando una macchina dedicata (meno potente, e quindi più economica, rispetto al calcolatore principale).
2. Il SO esegue i job del batch, *uno alla volta*, caricandoli dal nastro e scrivendo i risultati su un altro nastro. Esso si avvale anche di un *nastro di sistema*, che contiene ad esempio l'assembler e il compilatore Fortran.
3. I risultati presenti sul nastro di output vengono stampati mediante un'altra macchina dedicata.

5.1 Funzioni svolte dal SO

Un sistema batch svolge, seppure in modo banale, funzioni che si ritrovano anche nei SO moderni:

Scheduling: L'assegnazione della CPU ai processi è determinata implicitamente dall'ordine dei job nel batch.

Gestione della memoria: All'avvio del sistema, la memoria viene suddivisa in due zone:

- la **system area**, riservata al SO;
- la **user area**, che ospita il singolo job in esecuzione.

Questa suddivisione è presente anche nei sistemi attuali (ma la user area contiene invece porzioni di più programmi contemporaneamente).

Protezione: Se un job richiede più schede di quante ne siano effettivamente fornite, le restanti non devono essere "rubate" al job seguente. Inoltre, i job non devono poter accedere alla system area.

Interazione con l'utente: Il SO esegue i comandi inviati al command processor.

5.2 Vantaggi e svantaggi

Con l'introduzione dei sistemi batch, diminuiscono i tempi di inutilizzo della CPU: alla fine di un job, il SO carica immediatamente il successivo dal nastro, evitando lo "spreco" di tempo dovuto al caricamento manuale di un job da schede perforate. Ciò è importante perché i calcolatori di questo periodo sono molto costosi, e perciò devono essere sfruttati al massimo.

In compenso, per il singolo utente, il *tempo di turn-around* (tra la sottomissione del job e la restituzione dei risultati) può aumentare, perché un batch non viene preparato finché non si ha un numero sufficiente di job.

Si tratta quindi di un caso in cui lo sfruttamento efficiente delle risorse viene privilegiato rispetto alla user convenience.