Azzolini Riccardo 2019-09-18

Introduzione ai Sistemi Operativi

1 Funzioni del sistema operativo

Alcune tra le principali funzioni del sistema operativo (SO) sono:

- consentire la **multiprogrammazione**, cioè l'esecuzione di più di un programmi contemporaneamente;
- realizzare la **memoria virtuale**, che permette di eseguire più programmi di quanti ce ne starebbero in RAM;
- gestire la corrispondenza tra file (che non esistono direttamente sull'hardware) e byte sull'hard disk;
- realizzare l'ambiente multiutente, assicurandosi che ciascun utente possa accedere solo ai propri file e programmi.

2 Modalità operative dell'hardware

L'hardware ha due modalità operative, **user** e **kernel**. La modalità attiva in un dato istante è determinata dal valore di un bit presente nel registro **PSW** (**Program Status Word**) della CPU.

Alcune istruzioni del linguaggio macchina, dette *privilegiate*, possono essere eseguite *solo* in modalità kernel.

Il sistema operativo esegue in modalità kernel, mentre la shell o l'interfaccia grafica (GUI) e i programmi applicativi vengono eseguiti in modalità utente. Per questo, si può definire il SO come il software che esegue in modalità kernel, ma questa definizione, per quanto precisa, dà poche informazioni.

Il passaggio tra modalità user e kernel avviene continuamente, ogni volta che le applicazioni si avvalgono delle funzionalità del sistema operativo.

Osservazione: Siccome la GUI esegue in modalità user, *non fa parte del SO*, nonostante sia compresa in quasi tutte le distribuzioni di sistemi operativi per PC (comunque, essa non è indispensabile: per interagire con il SO è sufficiente una shell / prompt dei comandi). Analogamente, tali distribuzioni solitamente includono anche altri programmi che vengono eseguiti in modalità user e hanno lo scopo di aiutare il SO a svolgere determinati compiti (ad esempio, il cambio della password).

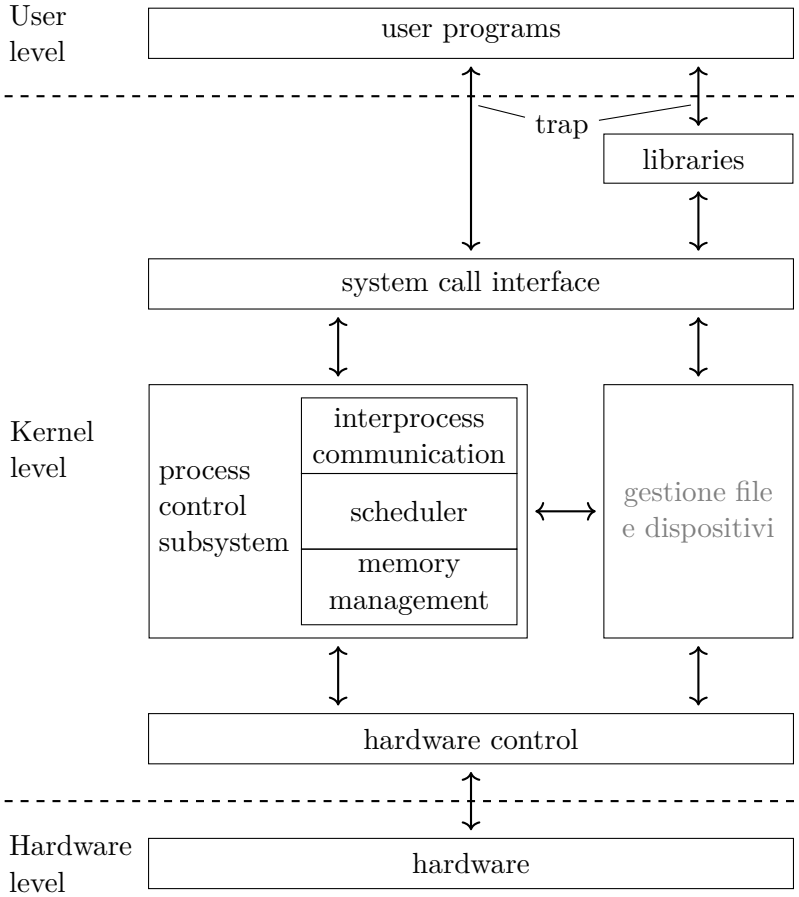
3 Esempio: UNIX System V

Nonostante sia un sistema datato (1986), UNIX System V svolge compiti che devono essere svolti anche dai sistemi attuali, quindi la sua struttura costituisce un buon esempio.

Il sistema è costituito da quattro livelli:

1. hardware;
2. kernel (SO);
3. programmi user di utilità, venduti insieme al SO;
4. altri programmi (compresi quelli installati o scritti dagli utenti), che usano i servizi offerti da quelli del livello 3.

3.1 Struttura del kernel



Hardware control: comunica con l'hardware, mediante:

- la lettura e scrittura dei *registri del controllore*;
- la gestione degli *interrupt*.

Process control subsystem: gestisce l'assegnazione della CPU (**scheduling**) e della memoria (**memory management**) ai processi, e permette la comunicazione tra di essi (**interprocess communication**).

System call interface: permette ai programmi di chiedere l'intervento del SO.

Libraries: facilitano l'invocazione delle system call.

Trap: istruzioni macchina usate per implementare le system call.

4 Definizione più completa di Sistema Operativo

Come definizione alternativa, un SO è un programma, opportunamente strutturato (si veda ad esempio la struttura di UNIX System V), che:

- controlla l'esecuzione dei programmi applicativi (cioè quelli che vengono eseguiti in modalità user), garantendo loro l'accesso a **risorse fisiche** (CPU, memoria, dispositivi) e **logiche** (come ad esempio i file);
- fornisce ai programmi applicativi un'interfaccia verso l'hardware, che permette di interagire con esso senza doverne conoscere i dettagli architetturali.

Osservazione: Il SO è software che controlla l'esecuzione del software e fornisce al software un'interfaccia verso l'hardware. Perché ciò sia possibile, esso deve per forza essere software particolare, e infatti è l'unico eseguito in modalità kernel.

5 Macchina virtuale e gestore di risorse

Il SO realizza una **macchina virtuale**:

- ogni programma applicativo "vede" una propria macchina, dotata di risorse fisiche e logiche;
- tale macchina è più facile da programmare rispetto a quella fisica costituita dall'hardware.

Il SO può quindi essere visto come:

- un **gestore di risorse**, fisiche e logiche, che mette a disposizione degli applicativi (visione top-down: si parte da un livello in cui le applicazioni hanno a disposizione risorse virtuali, per poi scendere al livello delle risorse fisiche);

- il realizzatore di una **macchina estesa**, messa a disposizione degli applicativi (visione bottom-up: si ragiona in termini di macchine via-via più evolute e facili da programmare).

5.1 SO come gestore di risorse

Come gestore di risorse, il SO deve fornire a ogni programma:

- una **CPU virtuale**, assegnando la CPU fisica a turno ai vari programmi;
- una **memoria virtuale**, che permetta al programma di accedere ai propri dati, ma non a quelli delle altre applicazioni;
- vari **dispositivi virtuali**, per consentire la condivisione dei dispositivi fisici (es. stampanti) senza interferenze tra programmi;
- le **risorse logiche** (es. file) di cui ha bisogno.

Quindi, in generale, il SO deve consentire ai programmi di *condividere le risorse*, garantendo:

efficienza: tutti i programmi devono essere eseguiti in un tempo ragionevole;

protezione: non devono esserci interferenze tra programmi.

5.2 SO come macchina estesa

Come macchina estesa, il SO fornisce un'**API** (**Application Programming Interface**) che:

- nasconde ai programmatori i dettagli dell'hardware, perciò è più facile da usare rispetto all'interazione diretta con la macchina fisica;
- favorisce la *portabilità dei programmi*, perché diverse configurazioni hardware richiedono modifiche all'implementazione dell'API, ma non all'API stessa, e quindi neanche ai programmi che la utilizzano.

6 Obiettivi del SO

Mentre svolge il proprio compito, il SO deve perseguire due obiettivi:

- *Uso efficiente* del sistema di computazione: le risorse vengono monitorate e assegnate secondo opportuni criteri che consentono di sfruttarle al meglio. Quest'attività di ottimizzazione, però, introduce un **overhead**, cioè consuma le risorse stesse, quindi bisogna trovare un equilibrio.[1]
- *User convenience*:
  - possibilità di eseguire programmi e usare il file system;
  - ottenere risposte veloci;
  - facilità d'uso attraverso interfacce amichevoli;
  - disponibilità di interfacce ad hoc per esperti.

Questi obiettivi sono spesso in competizione tra loro: ad esempio, una GUI è più amichevole rispetto a una shell, ma usa più risorse, sottraendole ai programmi applicativi.

[1] Un SO ideale non userebbe risorse, e al tempo stesso permetterebbe il funzionamento ottimale dei programmi, ma ciò è impossibile.