Sistemi Operativi

Alexej Nardovičk

September 27, 2022

1 Contenuti del corso

- **Funzione** e **struttura** di un Sistema Operativo
- **Sistemi a processi**: proprioetà di processi e thread
- Gestione dell'**unità centrale** - Alg.s di scheduling
- Gestione della **memoria centrale**
- Gestione del **file system**
- Gestione della **memoria secondaria** e i dispositivi di I/O
- **Casi di studio** - Unix/Linux. Linee generali di Windows e MacOS.

2 Introduzione

2.1 Cos'è un Sistema Operativo?

- macchina estesa
- gestore di risorse

2.2 Storia dei S.O.:

2 Storia dei Sistemi Operativi

2.1 I generazione – 1945 - 1955

La prima generazione degli elaboratori elettronici è caratterizzata dall'utilizzo della tecnologia delle valvole e delle tavole di commutazione (*plug boards*), insieme ad una programmazione avvenuta direttamente in linguaggio macchina.
Alcuni antecedenti di queste macchine furono la *Macchina analitica* del matematico Charles Babbage (1837 ca.) e il lavoro di Lady Ada Lovelace, considerata la prima programmatrice della storia.
La programmazione in questa epoca avveniva senza alcun sistema operativo di riferimento, bensì direttamente in linguaggio macchina o cablando manualmente i circuiti del calcolatore.
Gli anni '40 videro la nascita di alcuni dei primi calcolatori *general purpose*, (*i.e.* macchine senza un unico programma di esecuzione ma programmabili per eseguire calcoli generici), tra i quali il *MARK I* della Harvard University costruito da IBM, l'*ENIAC* (*Electronic Numerical Integrator and Calculator*) e successivamente l'*EDVAC*, (*Electronic Discrete Variable Automatic Computer*), la prima macchina basata sulla tecnologia dei transistor che segue l'architettura di von Neumann.
Queste macchine erano molto ingombranti fisicamente, e offrivano una potenza di calcolo minima rispetto a quello che ci offrono oggigiorno i Personal Computer: erano noti infatti i problemi relativi all'affidabilità, alla complessità d'uso e alla lentezza, dovuti tra le altre cose ad una mancata distinzione tra i ruoli di costruzione, progettazione, programmazione e manutenzione delle macchine.
La memorizzazione di dati e programmi avveniva prima attraverso schede perforate, successivamente tramite transistor.

2.2 II generazione – 1955 - 1965

A partire da metà anni '50 prese piede lo sviluppo di linguaggi di programmazione ad alto livello, tra i quali ricordiamo sicuramente il linguaggio assembly e Fortran. La memorizzazione di dati e programmi avveniva ancora tramite schede perforate e transistor, e i calcolatori erano caratterizzati dalla cosiddetta **monoprogrammazione**: i programmi utente memorizzati su schede perforate e in attesa di esecuzione venivano caricati ed eseguiti nelle macchine **uno alla volta**. Questi programmi o insiemi di programmi prendono il nome di **job**, e la loro esecuzione avveniva in differita attraverso un operatore che eseguiva dei **comandi *batch***.
Questa generazione vede anche la distinzione tra i seguenti ruoli:

- Costruttori
- Programmatori (programmavano in *high level language*)
- Operatori (gli effettivi utilizzatori dei calcolatori; gestivano le schede di programma, le schede JCL (Job Control Language) e l'I/O)

In questi anni cominciano a nascere i primi sistemi operativi per semplificare l'uso dei calcolatori, che funzionavano appunto secondo il concetto di *batch*:

1) I programmi e i dati venivano caricati in sequenza su un nastro
2) Questi nastri venivano spostati sul computer ed eseguiti a lotti (da qui *batch*)
3) L'output veniva stampato su nastro
4) Altri computer più piccoli si occupavano della lettura/scritta da/verso nastro

Figure 1: Struttura tipica di un job - Fortran

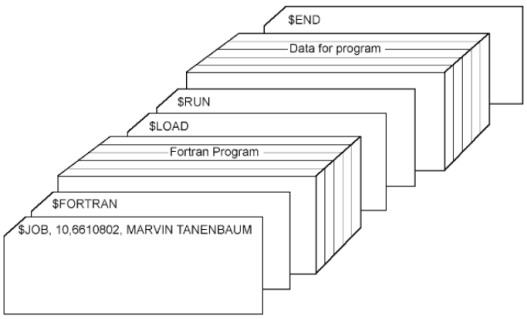
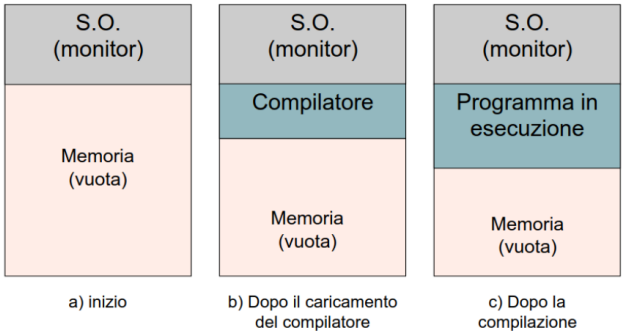


Figure 2: Organizzazione della memoria durante l'esecuzione di un sistema *batch*



2.3 III generazione – 1965 - 1980

La terza generazione di calcolatori conserva dagli anni precedenti l'utilizzo di sistemi di elaborazione *batch*, tuttavia con una grande novità: quella della **multiprogrammazione**: secondo questo principio, un job può usare il processore, mentre altri job usano le unità periferiche di cui hanno bisogno. Questo portò ad una partizione della memoria, assegnandone le varie parti a diversi job, in favore di un'esecuzione contemporanea di diversi job e una potenza di calcolo decisamente superiore.
Per gestire questo *pool* di job da eseguire, nacque la necessità di algoritmi di **scheduling** della CPU: algoritmi efficienti grazie ai quali il processore sa come, quando e quanto allocare risorse ai programmi che più ne hanno bisogno in un determinato momento. Vennero quindi progettati i primi ***scheduler***, software del sistema operativo adibiti a questo compito.
In questi anni si progettarono nuovi linguaggi di programmazione ad alto livello (*e.g.* C) e si diffuse l'uso di editor testuali e grafici.
Un'altra novità tecnologia di questa generazione è lo sviluppo dei **circuiti integrati**, che permisero una riduzione dei costi ottenendo anche una maggior velocità, e la nascita di sistemi operativi più complessi.
Tra le macchine degne di nota ricordiamo la famiglia di elaboratori *System/360* di produzione IBM e finalizzata all'uso per *mainframe* e i minicalcolatori PDP, caratterizzati dai costi contenuti e dalle prestazioni limitate.

Studiamo più nel dettaglio alcuni concetti della multiprogrammazione:

- **SPOOL** (*Simultaneous Peripheral Operation On Line*): tecnologia che permette l'esecuzione di operazioni concorrenti e l'esecuzione di job parallela alle operazioni I/O e al trasferimento dati.
- **Sistemi timesharing**: variante della multiprogrammazione, sviluppata

per supportare molti utenti interattivi simultanei ai terminali. Questa tecnologia prevedeva di dividere il tempo della CPU in **quanti di tempo**, di lunghezza arbitraria, al termine dei quali il job viene interrotto e si assegna la CPU al job successivo (*prelazione*).
Questo concetto comportava dunque:

- Cambi di contesto frequenti (**context switch**, che avvengono tuttora nelle architetture moderne)
- Attenzione alla protezione e sicurezza dei dati e della memoria (un job non può poter accedere ai dati di un altro job)
- Principio di sviluppo della tecnologia della **memoria virtuale**
- Tempo di risposta ridotto a minuti/secondi

Per quanto questo concetto possa sembrare estremamente efficiente, bisogna selezionare la dimensione del quanto di tempo con una certa cognizione di causa: un quanto troppo breve causerebbe un overhead troppo grande dovuto alla maggior frequenza di cambi di contesto, mentre un quanto troppo lungo rallenterebbe job di minore entità in favore di job più complessi.

Alcuni esempi di tecnologie di timesharing introdotte in questo periodo furono il sistema **CTSS** (*Compatible Time Sharing System*) da parte dell'MIT, evoluto poi nel **MULTICS** (*Multiplexed Information and Computer Service*), in cui venne introdotto il concetto di **processo**. Questi sistemi, insieme al **TSS** (*Time Sharing System*, su cui si basa *Unix*, di cui parleremo in seguito) e al **CP/CMS** (*Control Program/Cambridge Monitor System*) includono il concetto di **memoria virtuale**, per dare ai processi l'illusione di avere maggior memoria di quella fisicamente disponibile.
Nacquero anche i sistemi **real-time**, atti a fornire una risposta entro un dato periodo di tempo limitato.

Abbiamo già menzionato *Unix*, sistema operativo derivato da CTSS e MULTICS che implementa le tecnologie di timesharing multiprogrammati che abbiamo visto finora. Nel 1974 *Unix* fece registrare una doppia licenza, commerciale e libera, e venne pubblicato il suo codice sorgente: per rendere compatibili le diverse versioni che si vennero a creare di questo sistema operativo, fu introdotto lo standard *POSIX*, appartenente allo standard *IEEE*, dal momento che disponendo del codice sorgente ognuno aveva la possibilità di creare una nuova versione del sistema operativo. Questa possibilità fu anche favorita dallo sviluppo dei primi semplici Personal Computer, costituiti da una tecnologia a microprocessori.

Infine, in questa generazione si iniziò a sviluppare anche la tecnologia Internet e i protocolli TCP/IP da parte del Dipartimento della Difesa americano, usati in ambienti militari e accademici per una massiccia e veloce trasmissione dei crescenti volumi di dati.

3 Cos'è un sistema operativo

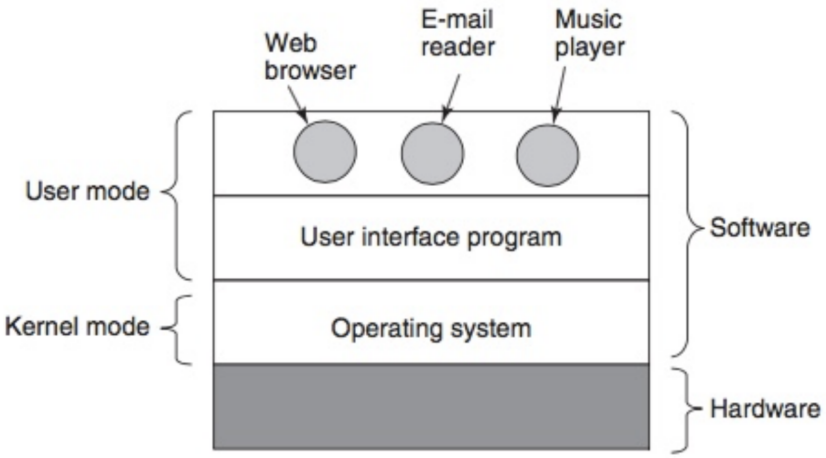
Un S.O. è un software che controlla l'hardware.
Un S.O. è un programma che gestisce e controlla l'esecuzione di un insieme di applicazioni, agisce come interfaccia tra le applicazioni e l'hardware del calcolatore e gestisce le risorse hardware.
Il S.O. è (prevalentemente) eseguito con il processore in modalità kernel Le applicazioni sono eseguite principalmente in modalità utente. Un sistema operativo ha applicazioni separate dall'hardware utilizzato:
- livello software
- gestione software e hardware per produrre i risultati desiderati

Un Sistema operativo innanzitutto è un gestore di risorse:
• Risorse harware;

- processore
- periferiche I/O
- memoria
- periferiche di comunicazione
- ecc.

• Applicazioni software

componenti di un sistema operativo:



4 Sistemi timesharing

- Variante della multiprogrammazione, fu sviluppato per supportare più utenti interattivi nello stesso momento sul terminale
- Il tempo di CPU è suddiviso in **quanti di tempo**
- Al termine del quanto il job viene interrotto e si assegna la CPU al job successivo (prelazione)
- Il tempo di risposta viene ridotto a minuti o secondi

Progettazione dei sistemi timesharing:
- Gestione del processore: scheduling
- Gestione della memoria: memoria virtuale
- Protezione delle risorse (memoria, filesystem, ...)