01 - Introduzione Sistemi Operativi

Cosa si intende per sistema operativo?

E' un programma che agisce come intermediario tra l'utente e l'hardware del computer:

- Fornisce un ambiente di sviluppo e di esecuzione per i programmi applicativi
- Fornisce una visione astratta dell'HW
- gestisce efficientemente le risorse del sistema di calcolo

Il sistema operativo puo' quindi essere visto come interfaccia tra cio' che offre l'hardware e il livello applicativo. Quello che deve fare il sistema operativo, tra gli altri compiti, e' quello di mappare le risorse hardware fisiche con delle risorse logiche, che offrono come funzionalita' la possibilita' di essere piu' facili da gestire e fornire a livello applicativo.
In particolare:

- La capacita' di processamento della CPU viene offerta tramite l'astrazione dei **processi**
- L'accesso ai dischi fisici viene fornito attraverso l'astrazione del **file system**
- L'accesso alla memoria, in particolare la memoria virtuale, viene offerta grazie al concetto di memoria virtuale.

Nel sistema operativo, l'utente e' l'utilizzatore dei programmi applicativi.
Possiamo quindi vedere l'architettura hardware e software di un elaboratore come un'architettura a strati in cui al centro troviamo l'hardware e le risorse fisiche, il sistema operativo che ah come compito di fornire una visione astratta e facilitata alle risorse dell'hardware, i programmi applicativi che grazie al sistema operativo e all'astrazione che offre sono in grado di accedere e utilizzare le risorse hardware e infine gli utenti che accedono ai programmi applicativi.

Un sistema operativo e' un programma che gestisce risorse del sistema di calcolo in modo corretto ed efficiente e le alloca ai programmi/utenti che ne necessitano. E' quindi un programma che ha come un risultato di innalzare il livello di astrazione con cui le risorse vengono sfruttate.

I sistemi operativo possono differire rispetto a molteplici aspetti:

- **Struttura**: come e' organizzato un SO (monoblocco o modulare);
- **Condivisione**: quali risorse vengono condivise tra utenti e programmi e in che modo;
- **Efficienza**: come massimizzare l'utilizzo delle risorse disponibili e anche che cosa massimizzare;
- **Affidabilita'**: come reagisce un SO a malfunzionamenti (SW/HW)
- **Estendibilita'**: e' possibile aggiungere funzionalita' al sistema?;
- **Protezione e sicurezza**: SO deve impedire interferenze tra programmi/utenti diversi;
- **Conformita' Standard**: portabilita', estendibilita', apertura.

Evoluzione SO

- Prima generazione (anni '50):
  - Linguaggio macchina
  - dati su schede perforate
- Seconda generazione ('55-'65):
  - sistemi batch semplici
  - linguaggio di alto livello (fortran)
  - input mediante schede perforate
  - aggregazione di programmi **lotti** (batch) con esigenze simili

Un sistema batch e' essenzialmente un insieme di programmi (job) che possono eseguire in modo sequenziale. L'esecuzione termina quando l'ultimo dei job e' arrivato a terminazione.
In questo caso il sistema operativo viene chiamato **monitor**, ovvero offre funzionalita' di trasferimento e controllo da un job all'altro. Si parla quindi di batch semplici.

Caratteristiche dei sistemi batch semplici:

- **SO residente in memoria (monitor)**;
- **assenza di interazione tra utente e job**;
- **scarsa efficienza**: durante l'I/O del job corrente, la CPU rimane inattiva (lentezza dei dispositivi I/O meccanici). Questo perche' in memoria centrale veniva caricato al piu' un solo job.

A causa della scarsa efficienza dei sistemi batch semplici e per migliorare l'utilizzo della CPU, e' stato introdotto il meccanismo di **Spooling (Simultaneous Peripheral Operation On Line)**. Tale meccanismo permetteva di caricare su disco i programmi e i dati quando la CPU era ancora in utilizzo per altri job, era quindi possibile sovrapporre l'uso della CPU per un job con la parte I/O dei job successivi.

Problemi:

- finche' il job corrente non e' terminato, il successivo non può iniziare l'esecuzione;
- se un job si sospende in attesa di un evento, la CPU rimane inattiva
- non c'e' iterazione con l'utente.

Sistemi batch multiprogrammati

Per ovviare a questi problemi si e' passati a sistemi batch multiprogrammati, in questo caso abbiamo sempre un pool di job che possono eseguire, ma in questo caso contemporaneamente e i job che possono eseguire sono tutti presenti su disco. In questo caso l'SO evolve e ha due compiti principali che nei sistemi batch semplici non aveva:

- SO seleziona un sottoinsieme di job appartenenti al pool da caricare in memoria centrale;
- mentre un job e' in attesa di un evento, il sistema operativo assegna CPU a un altro job. Si ha quindi una riduzione dei tempi di esecuzione dei job.

E' quindi l'SO in grado di portare avanti l'esevuzione di diversi job contemporaneamente, in ogni caso un solo job alla volta puo' usare la CPU e quindi e' in effettiva esecuzione. Mentre molteplici job possono essere pronti ad essere eseguiti e quindi attendono che il sistema operativo li assegni alla CPU.
Il sistema operativo non e' piu' un monitor come nei sistemi batch semplici ma si evolve verso un sistema operativo di tipo **scheduling**, ovvero ha il compito di scegliere quali dei job pronti per l'esecuzione deve essere effettivamente messo in esecuzione, e deve anche supportare tutte le funzionalita' necessare per fermare un job, selezionarne un altro e metterlo in memoria centrale.

Le scelte importanti che l'SO deve fare sono due:

- quali job caricare in memoria centrale: **scheduling del job** (long-term scheduling)
- a quale job assegnare la CPU: **scheduling della CPU** (short-term scheduling)

In memoria centrale, ad ogni istante, possono essere caricati piu' job. Dato che nella memoria centrale c'e' il sistema operativo, e' necessario che il sistema operativo assicuri che ciascun job possa intervenire solo nello spazio di memoria ad esso associato. Serve quindi protezione.

Sistemi time-sharing (Multics, 1965)

Nascono dalla necessita' di:

- **interattivita'** con l'utente, l'utente e' quindi in grado di interagire con i job tramite il sistema operativo, non solo all'inizio e alla fine del job, ma anche durante l'esecuzione del job;
  Per garantire un'accettabile velocita' di "reazione" alle richieste dei singoli utenti, SO **interrompe l'esecuzione** di ogni job dopo un intervallo di tempo prefissato (**quanto di tempo** o **time slice**), assegnango la CPU a un altro job.
- **multi-utenza**: piu' utenti interagiscono contemporaneamente con SO. Il sistema presenta ad ogni utente una **machina virtuale completamente dedicata** in termini di
  - utilizzo della CPU
  - utilizzo delle risorse, ad es. file system

I sistemi time-sharing sono sistemi in cui:

- attivita' della **CPU e' dedicata a job diversi** che si alternano **ciclicamente** nell'uso della risorsa
- frequenza di commutazione della CPU e' tale da fornire l'illusione ai vari utenti di una macchina completamente dedicata (**macchina virtuale**)

Cambio di contesto (context switch): operazione di trasferimento del controllo da un job al successivo --> costo aggiuntivo (overhead)

Requisiti del time-sharing:

- **Gestione/protezione** della memoria:
  - trasferimento memoria-disco
  - **separazione degli spazi** assegnati ai diversi job
  - molteplicita' job + limitatezza della memoria
  - a ogni job e' assegnata una cella di memoria virtuale (esso puo' accedere solo alla memoria a lui assegnata)
- **Scheduling CPU:** il sistema operativo deve poter schedulare l'assegnamento della CPU ai job anche in relazione al tempo di esecuzione di ciascun job
- **Sincronizzazione/comunicazione** tra job (job diversi possono avere necessita' di scambiarsi informazioni tra di loro o di accedere a risorse condivise):
  - interazione
  - prevenzione/trattamento di blocchi critici (**deadlock**)
- **Interattività'**: accesso on-line al file system per permettere agli utenti di accedere semplicemente a codice e dati

Esempi di SO attuali:

- **MSDOS**: monoprogrammato, monoutente
- **Windows 95/98**, molti SO attuali per dispositivi portabili (**Symbian, PalmOS**): multprogrammato (time sharing), tipicamente monoutente
- **Windows NT/2000/XP**: multiprogrammato, "multiutente"
- **MacOSX**: multiprogrammato, multiutente
- **UNIX/LINUX**: multiprogrammato, multiutente

Hardware di un sistema di elaborazione

Funzionamento a interruzioni:

- le varie componenti (HW e SW) sono in grado di interagire col sistema operativo tramite degli **interrupt (interruzioni asincrone)**
- ogni interruzione e' causata da un **evento**, ad es:
  - richiesta di servizio al SO
  - completamento di I/O
  - accesso non consentito alla memoria

- ad ogni interruzione e' associata una **routine di servizio (handler)** per la **gestione dell'evento**

Le interruzioni possono essere di due tipi:

- **Interruzioni Hardware**: dispositivi inviano segnali per richiedere l'esecuzione di servizi di SO
- **Interruzioni Software**: i programmi in esecuzione sono in grado di generare interruzioni SW
  - quando tentano l'esecuzione di **operazioni non lecite** (ad es. divisione per 0): **trap**
  - quando richiedono l'esecuzione di servizi al SO - **system call**

Gestione delle interruzioni

Alla ricezione di un'interruzione, SO:

1. interrompe la sua esecuzione --> **salvataggio dello stato** in memoria (locazione fissa, stack di sistema, ...)
2. attiva la **routine di servizio all'interruzione** (handler)
3. **ripristina lo stato** salvato

Per individuare la routine di servizio, SO puo' utilizzare un **vettore delle interruzioni**, vettore che metta in associazione ciascun possibile interrupt HW/SW con un'opportuna routine di servizio che sia in grado di gestire quel particolare interrupt.

Come avviene l'I/O in un sistema di elaborazione?

Avviene tramite dei controller, che fungono da interfaccia tra l'hardware e il bus d sistema che permette di mettere in comunicazione diverse periferiche, memoria centrale e CPU. Ogni controller e' dotato di:

- **un buffer** (con la funzione di memorizzare temporaneamente le informazioni da leggere o scrivere)
- alcuni **registri speciali**, ove memorizzare le specifiche delle operazioni di I/O da eseguire e in quale area di memoria leggere o scrivere

Quando un job richiede un'operazione di I/O (ad esempio, lettura da un dispositivo):

- CPU scrive nei registri speciali del dispositivo le specifiche dell'operazione da eseguire
- controller esamina i registri e provvede a trasferire i dati richiesti dal dispositivo al buffer
- invio di interrupt alla CPU (complemento del trasferimento)
- CPU esegue l'operazione di I/O tramite la routine di servizio (trasferimento dal buffer del controller alla memoria centrale)

2 tipi di I/O:

- **Sincrono**: il job viene sospeso finche' l'operazione di I/O non viene terminata
- **Asincrono**: il sistema restituisce immediatamente il controllo al job (molto piu' efficiente ma anche molto piu' complesso)
  - e' necessario predisporre delle funzionalita' di blocco in attesa di completamento dell'I/O

    - e' possibile avere piu' I/O pendenti
        - tabella di stato dei dispositivi

Direct Memory Access

Il trasferimento tra memoria e dispositivo viene effettuato direttamente, **senza intervento della CPU**, questo rende piu' efficiente il trasferimento di grandi quantita' di dati.

Introduzione di un dispositivo HW per controllare I/O: **DMA controller**.

- **driver di dispositivo**: componente del SO che
    - copia nei registri del DMA controller i dati relativi al trasferimento da effettuare
    - invia comando di richiesta al DMA controller
- **Interrupt** della CPU (inviato dal DMA controller) solo alla fine del trasferimento dispositivo --> memoria, usualmente di grandi quantita' di dati.

Protezione HW degli accessi a risorse

Nei sistemi che prevedono multiprogrammazione e multiutenza e' necessario adottare dei sistemi di protezione che si basano anche su meccanismi HW.
Dato che le risorse, l'accesso ai dispositivi, l'accesso alla memoria e l'accesso alla CPU sono condivisi tra i diversi job in esecuzione, e' necessario che l'accesso a tali risorse condivise sia mediato tramite opportuni meccanismi che ne impediscano l'accesso illecito da parte di programmi e utenti.
Ad esempio: accesso a locazioni esterne allo spazio di indirizzamento del programma.

Protezione della memoria

In un sistema **multiprogrammato** o **time-sharing**, ogni job ha un suo spazio di indirizzi:

- e' necessario impedire al programma in esecuzione di accedere al aree di memoria estere al proprio spazio (as esempio de SO oppure di altri job)
- se fosse consentito, un programma potrebbe modificare codice e dati di altri programmi, o ancor peggio, del SO

Per garantire protezione, molte architetture di CPU prevedono un duplice modo di funzionamento (**dual mode**)

- **user** mode
- **kernel** mode (supervisor, monitor code)

Realizzazione: l'architettura hardware della CPU prevede un bit di modo

- kernel: 0
- user: 1

Dual mode

Istruzioni privilegiate: sono quelle piu' pericolose e possono essere eseguite soltanto se il sistema si trova in **kernel mode**

- accesso a dispositivi I/O (dischi, schede di rete, ...)
- gestione della memoria (accesso a strutture dati di sistema per controllo e accesso alla memoria, ...)
- istruzioni di **shutdown** (arresto del sistema)
- ...

L'intero sistema operativo esegue in modalita' kernel in quanto deve poter accedere alle risorse hardware a disposizione.
Ogni programma utente invece esegue in **user** mode:

- quando un programma utente tenta l'esecuzione di una istruzione privilegiata, viene generato un **trap**
- se necessita di **operazioni privilegiate** --> chiamata a system call

La system call serve per ottenere l'esecuzione di istruzioni privilegiate, un programma i utente deve chiamare una system call:

- invio di un'interruzione software al SO
- salvataggio dello stato (PC, registri, bit di modo, ...) del programma chiamante e trasferimento del controllo a SO
- SO esegue in modo **kernel** l'operazione richiesta
- al termine dell'operazione, il controllo ritorna al programma chiamante (ritorno al modo **user**)

Introduzione all'organizzazione dei Sistemi Operativi

Le principali componenti di un SO sono:

- gestione dei processi
- gestione della memoria centrale
- gestione della memoria secondaria e file system
- gestione dell'I/O
- protezione e sicurezza
- interfaccia utente/programmatore

Processi

Processo = programma in esecuzione.
Il programma e' un'entita' passiva (un insieme di byte contenente le istruzioni che dovranno essere eseguite).
Il processo, invece, e' un entita' attiva, ovvero e' l'unita' di lavoro/esecuzione all'interno del sistema.
Ogni attivita' del SO e' rappresentata da un processo. Essenzialmente, un processo e' l'istanza di un programma che viene messo in esecuzione.

Processo = programma + contesto di esecuzione (PC, registri, ...)

Gestione dei Processi

In un sistema multiprogrammato piu' processo possono essere simultaneamente presenti nel sistema.
Il compito cruciale del SO e' quindi quello di:

- **creazione/terminazione** dei processi
- **sospensione/ripristino** dei processi
- **sincronizzazione/comunicazione** dei processi
- **gestione del blocco critico (deadlock)** di processi

Gestione della memoria centrale

Dal punto di vista hardware il sistema di elaborazione e' equipaggiato con **un unico spazio di memoria** accessibile direttamente da CPU e dispositivi.
il compito cruciale del SO e' quindi quello di:

- **separare** gli spazi di indirizzi associati ai processi
- **allocare/deallocare** memoria ai processi
- **memoria virtuale** - gestione spazi logici di indirizzi di dimensioni complessivamente superiori allo spazio fisico
- realizzare i collegamenti (**binding**) tra memoria logica e memoria fisica

Gestione dei dispositivi I/O

La gestione dei dispositivi I/O rappresenta una parte di SO:

- interfaccia tra programmi e dispositivi, ovvero e' il SO che interagisce interamente coi controller dei dispositivi e col DMA
- per ogni dispositivo esiste all'interno del SO un **device driver**
    - routine pr l'interazione con un particolare dispositivo
    - contiene una conoscenza specifica sul dispositivo (ad es., routine di gestione delle interruzioni)

Gestione della memoria secondaria

Tra tutti i dispositivi, la memoria secondaria riveste un ruolo particolarmente importata es. gestione del file system, ma ha anche altre funzionalita':

- allocazione/deallocazione di spazio
- gestione dello spazio libero
- scheduling delle operazioni su disco

Di solito la gestione dei file usa i meccanismi di gestione della memoria secondaria. La gestione della memoria secondaria e' indipendente dalla gestione dei file.

Gestione del file system

Ogni sistema di elaborazione dispone di uno o piu' dispositivi per la memorizzazione persistente delle informazioni (memoria secondaria).
Compito di SO --> Fornire una visione logica uniforme della memoria secondaria (indipendente dal tipo e dal numero dei dispositivi):

- realizzare il concetto astratto di **file**, come unita' di memorizzazione logica
- fornire una struttura astratta per l'organizzazione dei file (**direttorio)**

inoltre il sistema operativo si deve anche occupare di:

- creazione/cancellazione di file e directory
- manipolazione di file/directory
- associazione tra file e dispositivi di memorizzazione secondaria

Spesso file, directory e dispositivi I/O vengono presentati a utenti/programmi in modo uniforme

Protezione e sicurezza

In un sistema multiprogrammato, piu' entita' (processi o utenti) possono utilizzare le risorse del sistema contemporaneamente: necessita' di protezione.

Protezione: controllo dell'accesso alle risorse del sistema da parte dei processi (e utenti) mediante **autorizzazioni** e **modalita' di accesso**.

Sicurezza: se il sistema appartiene a una rete, la sicurezza misura l'affidabilita' del sistema nei confronti di accessi (attacchi) dal mondo esterno.

Risorse da proteggere:

- memoria
- processi
- file
- dispositivi

Interfaccia utente

Il sistema operativo deve offrire la possibilita' agli utenti di interfacciarsi con il SO e quindi alle risorse che il SO gestisce

- **interpretazione comandi (shell)**: l'interazione avviene mediante una linea di comando
- **interfaccia grafica** (graphical user interface, **GUI**): l'interazione avviene mediante interazione mouse-elementi grafici su desktop; di solito e' organizzata in finestre.

Interfaccia programmatore

L'interfaccia del SO offre anche un'interfaccia per i programmatori, ovvero SO offre un insieme di system call con cui il programmatore puo' richiedere di interagire con le risorse condivise da parte del SO.

- mediante la system call il **processo richiede a SO** l'esecuzione di un servizio
- la system call esegue **istruzioni privilegiate**: passaggio da modo user a modo kernel

Classi di system call:

- gestione dei processi
- gestione di file e dispositivi (spesso trattati in modo omogeneo)
- gestione informazioni di sistema
- comunicazione/sincronizzazione tra processi

Programma di sistema = programma che chiama system call

Struttura e organizzazione di SO

Sistema operativo = insieme di componenti

- gestione dei processi
- gestione della memoria centrale
- gestione dei file
- gestione dell'I/O
- gestione della memoria secondaria
- protezione e sicurezza
- interfaccia utente/programmatore

Sulla base di questa osservazione bisogna chiedersi come e' possibile organizzare una struttura di un sistema operativo.
Esistono 3 macro approcci:

- struttura monolitica
- struttura modulare: stratificazione

- microkernel

SO monolitici

SO e' costituito da un unico modulo contenente un insieme di procedure, che realizzano le varie componenti. L'interazione tra le componenti avviene mediante il meccanismo di chiamata a procedura. (ne sono esempi, MS-DOS e le prime versioni di LINUX)

Principale vantaggio: basso costo di interazione tra le componenti;
Svantaggio: SO e' un sistema complesso e presenta gli stessi requisiti delle applicazioni **in-the-large**;
Soluzione: organizzazione modulare.

SO modulari

Le varie componenti del SO vengono organizzate in moduli caratterizzati da interfacce ben definite.

Storicamente, il primo sistema stratificato a livelli e' il sistema operativo THE di Dijkstra (1968) ed e' costituito da vari livelli sovrapposti. Ogni livello realizza un'insieme di funzionalita':

- ogni livello realizza un insieme di funzionalita' che vengono offerte al livello superiore mediante un'interfaccia;
- ogni livello utilizza le funzionalita' offerte dal livello sottostante, per realizzare altre funzionalita'

Il sistema operativo THE e' composto da 5 livelli

- livello 5: programmi di utente
- livello 4: buffering dei dispositivi di I/O
- livello 3: driver della console
- livello 2: gestione della memoria
- livello 1: scheduling della CPU
- livello 0: hardware

Vantaggi:

- Astrazione: ogni livello e' un oggetto astratto, che fornisce ai livelli superiori una visione astratta del sistema (macchina virtuale), limitta alle astrazioni presente nell'interfaccia
- Modularita': relazioni tra livelli sono chiaramente esplicitate dalle interfacce e possibilita' di sviluppo, verifica, modifica in modo indipendente dagli altri livelli.

Svantaggi:

- Organizzazione gerarchica tra le componenti: non sempre e' possibile --> difficolta' di realizzazione
- scarsa efficienza (costo di attraversamento dei livelli)

Soluzione --> limitare il numero dei livelli (struttura intermedia tra una struttura a 5 livelli e una struttura monolitica).

Questo avviene tramite la realizzazione di un sistema operativo con un nucleo (modo kernel), che e' la pare del sistema operativo che esegue in modo privilegiato. Quindi il nucleo (kernel) ha come compito di eseguire utte le operazioni privilegiate.

- E' la parte piu' interna di SO che si interfaccia direttamente con l'hardware della macchina
- Le funzioni realizzate all'interno del nucleo variano a seconda del particolare SO

Per un sistema multiprogrammato a divisione di tempo, il nucleo deve, almeno:

- gestire il salvataggio/ripristino dei contesti (context-switching)
- realizzare lo scheduling della CPU
- gestire le interruzioni
- realizzare il meccanismo di chiamata system call

SO a microkernel

La struttura del nucleo e' ridotta a poche funzionalita' di base:

- gestione della CPU
- gestione della memoria
- gestione dei meccanismi di comunicazione I/O

Il resto del SO e' mappato su processi utente

Caratteristiche:

- sono piu' affidabili (separazione tra componenti)
- possibilita' di estensioni e personalizzazioni
- scarsa efficienza (molte chiamate a system call)

MS-DOS

E' stato progettato per avere **minimo footprint**

- non diviso in moduli
- sebbene abbia una qualche struttura, interfacce e livelli di funzionalita' sono ben separati

UNIX

Dati i limiti delle risorse hardware del tempo, originariamente UNIX sceglie di avere una strutturazione limitata. Consiste di due parti separabili

- programmi di sistema
- kernel
  - costituito da tutto cio' che e' sotto l'interfaccia delle system-call interface e sopra hardware fisico
  - fornisce funzionalita' di file system, CPU scheduling, gestione memoria, ...; molte funzionalita' tutte allo stesso livello

Principi di progettazione e vantaggi:

- progetto snello, pulito e modulare
- scritto in linguaggio di alto livello (C)
- disponibilita' codice sorgente
- potenti primitive di SO su una piattaforma a basso prezzo
- progettato per essere time-sharing
- user interface semplice (Shell), anche sostituibile
- file system con direttori organizzati ad albero
- concetto unificante di file, come sequenza non strutturata di byte
- supporto semplice a processi multipli e concorrenza
- supporto ampio allo sviluppo di programmi applicativi e/o di sistema