

INTRODUZIONE AI SISTEMI OPERATIVI

Premessa (questa parte non verrà chiesta all'orale, quindi è la parte più importante!)

Concetti da sapere:

- SW significa softwaree
HW significa hardware
SO significa Sistema operativo
- differenza fra software di base e software applicativo (o applicazione)
 - software di base: consente il funzionamento del computer
 - applicazione: consente ad un utente l'utilizzo del computer per svolgere una specifica attività
- processo=programma in esecuzione, ossia le cui istruzioni binarie sono eseguite dal processore (per ora)
- multitasking: esecuzione di più processi contemporaneamente.

Cominciamo con un esempio

Avevamo suddiviso i software in

- software applicativo (applicazioni: word, videogiochi, IDE, browser, riproduttore musica)
- software di base (sistema operativo e driver).

Riprendiamo un esempio con la macchina di Von Neumann che consente di mostrare che i software applicativi, alla fine, devono “diventare” istruzioni binarie che vengono eseguite dall'hardware. **Alla fine tutte le istruzioni sono eseguite dall' hardware.**

Consideriamo come sw applicativo il file eseguibile di un nostro programma in C “somma fra due numeri”

Il cuore del programma da noi scritto è il seguente:

```
a=5;  
b=2;  
c=a+b;
```

quando viene compilato, il nostro programma viene tradotto in una sequenza di istruzioni binarie. Quando il programma viene eseguito, questa sequenza di istruzioni viene “caricata” nella RAM ed eseguita dal processore.

Cerchiamo di capire come sono fatte queste istruzioni binarie e come fa il processore ad eseguirle.

Per capire come sono fatte le istruzioni è utile ricorrere all'assembly della macchina di Von Neumann, anche se il compilatore non traduce in assembly ma direttamente in binario. Per noi è comodo usare l'assembly perché (essendo il linguaggio più a basso livello) ha una corrispondenza 1 a 1 con il codice binario generato.

In assembly il nostro programma diventerebbe:

```
load #5
sto 128      (ipotizzo che x sia la locazione di memoria con indirizzo 128 per comodità)
load #2
sto 129      (y è la locazione con indirizzo 129 e così via...)
add 128
sto 130      (z= locazione con indirizzo 130)
```

Ma in realtà, fisicamente, cosa viene scritto nelle celle della ram quando eseguiamo il file *somma fra due numeri.exe*?

Nella ram vengono caricate le istruzioni in forma binaria. Ipotizziamo per comodità che le istruzioni vengano caricate a partire dall'indirizzo 0 della RAM.

Come sono fatte in binario le istruzioni?

Per capirlo dobbiamo sapere che ogni istruzione (load, add, sto, ecc..) ha un equivalente codice binario che dipende dal processore utilizzato, tale codice si chiama codice operativo (*op code*). Nel caso del nostro processore Macchina di Von Neumann gli op.code si possono trovare sugli appunti della macchina di VN. Ogni istruzione avrà quindi, in binario, la seguente forma all'interno di una cella della RAM:

Cella di memoria RAM:

op code (add, sto,...)	operando (valore o indirizzo a cui è rivolta l'operazione)
------------------------	--

Ecco gli op code della macchina di VN:

```
load ind      4
lod #num      20
sto ind       5
add ind       0
.....
```

Sono i valori che il decoder della UC decodificherà- Il decoder infatti riceverà in ingresso i bit dell'op code ed attiverà in uscita i segnali elettrici che consentiranno ai dispositivi hw di eseguire l'operazione relativa (load, sto, ecc..)

Supponiamo che il bus dati ed il bus indirizzi siano a 32 bit, quindi ipotizziamo che le istruzioni siano a 32 bit (16 per l'op code e 16 per l'operando) in tal caso il nostro programma apparirà nella RAM nel seguente modo

ISTRUZIONI

0	00000000000000100	00000000000000101
1	00000000000000101	0000000010000000
2	00000000000000100	0000000000000010
3	00000000000000101	0000000010000001
4	00000000000000000	0000000010000000
5	00000000000000101	0000000010000010

DATI

128 (x)	00000000000000000	00000000000000101
129 (y)	00000000000000000	0000000000000010
130 (z)	00000000000000000	0000000000000111

Si può osservare con il simulatore della macchina di VN la fase di fetch-execute almeno della prima istruzione:

- Il PC contiene il valore 0
 - l'istruzione contenuta all'indirizzo 0 viene caricata nell' IR
 - Il PC viene incrementato e diventa 1
- FETCH**
-
- Il decoder decodifica l'istruzione (è una LOAD! su un operando che vale 5)
 - L'istruzione viene eseguita (si attivano i circuiti elettrici che portano il valore 5 nell' accu)
- EXECUTE**

Perché abbiamo visto tutto questo? Perché è importante vedere, almeno una volta, come il sw e l'hw interagiscono fra loro, ossia come il sw viene, alla fine, eseguito dall'hw. Il nostro esempio è semplice, ma questo è ciò che accade per qualunque istruzione di qualunque sw applicativo, un videogioco, word, un browser.... e non solo per i software applicativi, infatti, se ci pensiamo bene, anche il SISTEMA OPERATIVO (che d'ora in poi chiameremo SO) è un sw, quindi anche il SO non sarà altro che una serie di istruzioni binarie caricate nella RAM. Le principali istruzioni del SO (**quelle che chiameremo poi "kernel"**) vengono caricate nella RAM all'avvio del computer. La procedura di avvio di un computer è chiamata **bootstrap** e ha come risultato finale il caricamento delle istruzioni del SO dalla memoria di massa (SSD) alla memoria centrale.

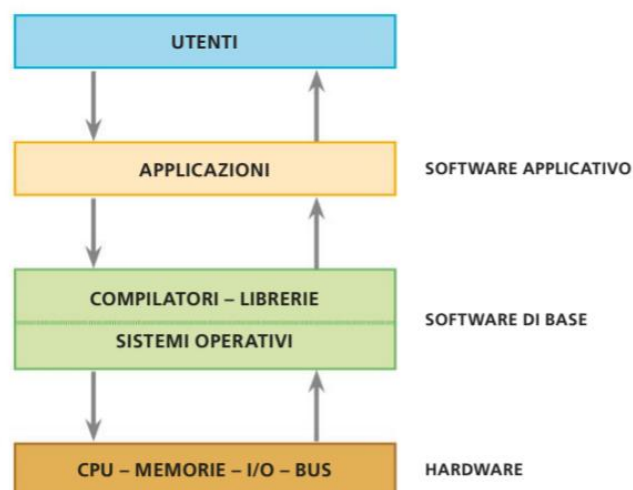
E' quindi importante capire che nella RAM ci saranno le istruzioni ed i dati NON SOLO del nostro sw applicativo "Somma fra due numeri.exe" ma di tutti i software applicativi e di sistema in esecuzione sul computer (SO, driver, browser, word,...).

D'ora in poi chiameremo **processo** ciascun software in esecuzione in un qualsiasi istante nel computer.

CHE COSA E' IL SISTEMA OPERATIVO (SO)? (CAP 1.1)

Per dare un'idea di cosa sia il SO possiamo utilizzare una similitudine. Quando una persona guida un'automobile conosce alcuni comandi (accendere l'auto, accelerare, frenare, girare il volante..) utilizzati attraverso un'interfaccia costituita da pulsanti, volante, pedali ecc.. che gli permettono di utilizzare l'automobile senza conoscere il funzionamento dell'hardware sottostante (motore a scoppio, servosterzo ecc..). Allo stesso modo un'utente utilizza un computer eseguendo dei comandi (cliccare su icone, doppio cliccare, ctrl+x, trascinare...) che gli permettono di utilizzare il computer senza sapere come funziona l'hardware sottostante (CPU, Memoria ecc..). Il Sistema Operativo è un insieme di sw che si pone fra l'utente che "comanda" l'esecuzione di determinate operazioni e l'hardware che, di fatto, "esegue" tali operazioni.

Quando gli utenti utilizzano delle applicazioni, i software applicativi "chiedono" le risorse hardware e software del computer al SO. Il SO si occupa di "fornire" alle applicazioni le risorse richieste. Le risorse sono costituite dall' hardware del computer.



Come abbiamo visto nell'esempio della MVN, quando viene eseguita un'applicazione sono necessarie numerose operazioni a livello hardware:

- Le istruzioni devono essere copiate dalla memoria di massa nella RAM
- La CPU deve eseguire (ciclo fetch-decode.execute) ogni istruzione
- i dati devono essere inseriti nella RAM, ad esempio dalla tastiera, e i risultati mostrati, ad esempio sul monitor.

Inoltre il multitasking prevede che vi siano più applicazioni in esecuzione contemporaneamente, ma se la CPU è solo una, essa, in ogni istante, può eseguire le istruzioni di **una sola** di queste applicazioni.

Il SO si occupa di gestire le risorse in modo che tutte le applicazioni che ne hanno bisogno vi possano accedere, il tutto nella maniera più efficiente possibile.

Si può dire che il SO fornisce **una rappresentazione astratta** dell'HW ai processi poiché il SO non permette alle applicazioni in esecuzione di accedere direttamente all'HW (CPU, MEMORIA, dispositivi di I/O). Infatti il SO "nasconde" l'HW alle applicazioni. Quando le applicazioni hanno bisogno di risorse HW le "chiedono" al SO il quale gliele fornisce ma verificando prima che non ci siano altri processi che le devono usare, verificando che l'applicazione può accedere a tali risorse, insomma verificando le opportune condizioni che consentano al processo di accedervi.

Il sistema operativo si occuperà di gestire le risorse mostrando e fornendo ai processi, solo le componenti di cui essi hanno bisogno.

COSA FA IL SO?

Riassumendo (poi vedremo i dettagli):

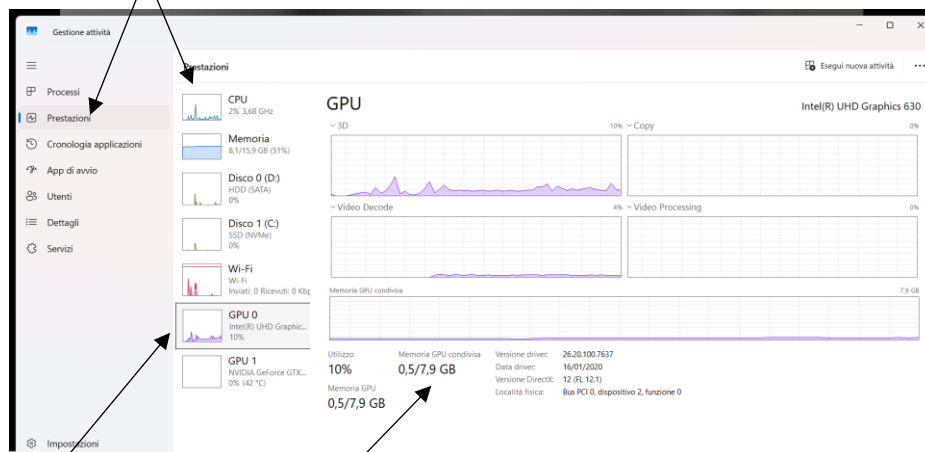
1. Gestione delle risorse HW (monitor , stampante ecc)
2. Gestione delle risorse SW (aprire file, chiudere file, leggere, scrivere file...)
3. Gestione degli utenti (solo gli utenti che ne hanno il permesso devono poter svolgere determinate operazioni su determinati file)
4. Gestione della sicurezza (Solo determinate applicazioni devono poter accedere a determinate zone della memoria, cancellare dati, creare dati, ecc.)
5. Gestione dell'interfaccia utente.

LA GESTIONE DELLE ATTIVITA' (CAP 1.2)

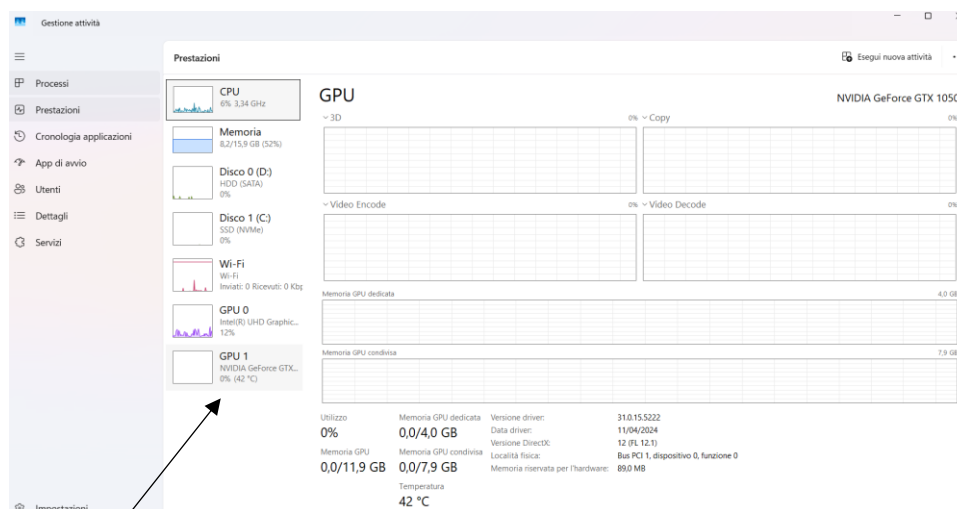
Per vedere quali sono le risorse hardware, quali sono i processi in esecuzione, quali risorse utilizza ciascun processo, e per interrompere eventuali processi, nel SO Windows si può utilizzare l'app "Gestione delle attività".

Vediamo alcune cose interessanti:

Le varie risorse HW



GPU integrata nel processore.
Usa parte della RAM del PC
Al max 8 GB dei miei 16 GB



GPU della scheda video. Viene utilizzata solo per attività specifiche (giochi, rendering, calcoli) solo quando la GPU 0 non è sufficiente. Questa GPU ha una propria RAM (4GB)

Quante risorse utilizza ciascun processo. Si noti che ogni istanza di un'applicazione è un processo. Ad esempio ogni pagina del Browser Google

Nome	Stato	7% CPU	45% Memoria	0% Disco	0% Rete
Applicazioni (7)					
Esplora risorse		0%	62,8 MB	0 MB/s	0 Mbps
Gestione attività		1,2%	76,1 MB	0 MB/s	0 Mbps
Google Chrome (9)		0%	49,8 MB	0 MB/s	0 Mbps
Google Chrome		0%	0,3 MB	0 MB/s	0 Mbps
Google Chrome		0%	5,0 MB	0 MB/s	0 Mbps
Google Chrome		0%	3,0 MB	0 MB/s	0 Mbps
Google Chrome		0%	1,1 MB	0 MB/s	0 Mbps
Google Chrome		0%	21,6 MB	0 MB/s	0 Mbps
Google Chrome	Modalità ...	0%	6,2 MB	0 MB/s	0 Mbps
Google Chrome	Modalità ...	0%	3,9 MB	0 MB/s	0 Mbps
Google Chrome	Modalità ...	0%	6,9 MB	0 MB/s	0 Mbps
Google Chrome	Modalità ...	0%	1,9 MB	0 MB/s	0 Mbps
HUB Young (6)		0%	173,7 MB	0,1 MB/s	0,1 Mbps
Microsoft Word (2)		0%	146,5 MB	0 MB/s	0 Mbps
Strumento di cattura		0%	76,1 MB	0 MB/s	0 Mbps
VLC media player (32 bit)		0%	19,8 MB	0 MB/s	0 Mbps
Processi in background (78)					
64-bit Synaptics Pointing Enha...		0%	0,1 MB	0 MB/s	0 Mbps
Acrobat Collaboration Synchr...		0%	8,1 MB	0 MB/s	0 Mbps
Acrobat Update Service (32 bit)		0%	0,7 MB	0 MB/s	0 Mbps
Activity Licensing Service		0%	0,5 MB	0 MB/s	0 Mbps

I processi si dividono in Applicazioni (quelli che prevedono interazione con l'utente) e "Background" (quelli necessari al SO e avviati dalle applicazioni senza che l'intervento da parte dell'utente, ad esempio antivirus ecc.)

DUE COMPONENTI FONDAMENTALI: IL KERNEL E LA SHELL

Il SO non è un semplice software ma un insieme di software, ognuno dei quali ha uno scopo specifico e fa parte di uno specifico MODULO. Il SO occupa molto spazio, perciò, non sempre tutti i moduli sono presenti contemporaneamente nella memoria centrale. Per lasciare una certa quantità di RAM libera per essere utilizzata dalle applicazioni, i moduli del SO possono essere caricati nella RAM solo nel momento in cui servono.

CAP 2.1 IL KERNEL

Il modulo “più importante” è il KERNEL. Kernel significa NUCLEO. Il kernel è la parte di SO che viene caricato nella memoria centrale all'avvio del computer, durante il bootstrap, e rimane sempre presente nella memoria centrale. Si dice che è il più importante perché nessuna applicazione può funzionare senza il kernel del SO.

Il kernel infatti si occupa della gestione dell'HW, quindi qualsiasi applicazione per utilizzare l'HW che ne esegue le istruzioni deve rivolgersi al kernel.

Il kernel si occupa dei seguenti compiti:

- gestione della CPU. I computer moderni funzionano in multitasking, se la CPU è solo una, il parallelismo è solo apparente; infatti, in ogni istante la CPU può eseguire le istruzioni di un solo processo. Il kernel decide a quale processo assegnare la CPU e per quanto tempo.
- gestione delle interruzioni dell'esecuzione delle istruzioni (**interrupt**) quando si presentano (ad esempio: assegnamento della CPU ad un altro processo, richiesta di un'operazione di I/O, errore di programma)
- gestione della memoria centrale. Decide quale parte della memoria centrale assegnare a ciascun processo in esecuzione in multitasking e carica il codice binario del processo nella memoria.
- controllo il corretto funzionamento delle periferiche hardware per mezzo dei driver.
- modifica dello stato di un processo (ready, wait, run, ...vedremo più avanti cosa significa)
- gestione lo scambio di messaggi fra processi che devono comunicare fra loro (ad esempio fra un word processor e uno spooler di stampa, i messaggi sono: “stampa questo file”, “OK ho stampato”.)

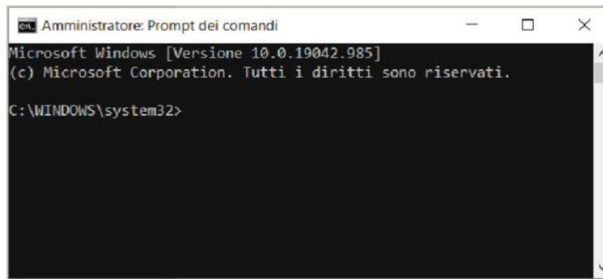
Il kernel è la parte di SO a diretto contatto con l'hardware, la parte più “interna” del SO (nocciolo).

CAP 2.2 LA SHELL

Se il kernel è il nocciolo, la shell è il guscio (shell=guscio, conchiglia) del SO. Essa infatti è costituita dai software del SO che gli utenti utilizzano per interagire con il SO. La shell dunque è la parte più “esterna” del SO operativo. Viene chiamata anche “**Interprete dei comandi**” o “**Interfaccia utente**”.

Le shell sono generalmente di 4 tipi:

- A linea di comando (come il prompt di MS DOS, Power Shell di Windows, Bash di Linux).



- A menu (come il BIOS).



- GUI (Graphic User Interface) come il desktop.



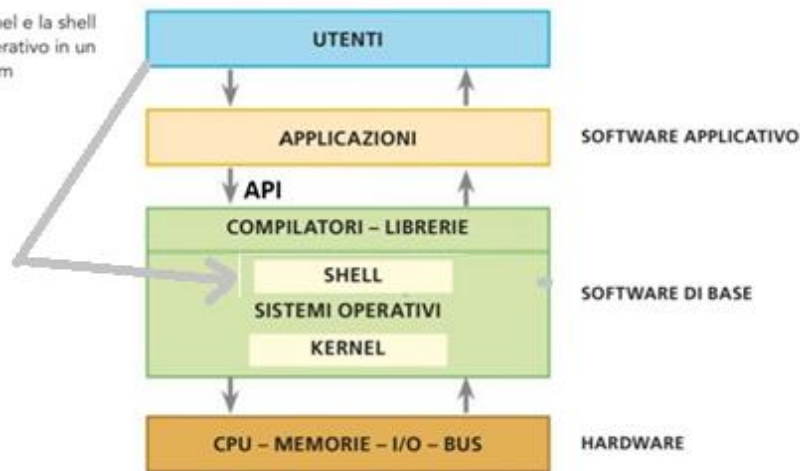
Mac OS Mojave

- A comando vocale. Come quello di Amazon Alexa.

La shell è lo strumento con il quale l'utente "invia un comando" al computer (ad es. sposta questo file).

Le applicazioni invece, quando devono "chiedere" delle risorse HW al computer (ad esempio "memorizza questi dati su un file"), utilizzano delle funzioni del SO che fanno parte di librerie chiamate **API (Application Program Interface)**.

FIGURA 3 Il kernel e la shell del sistema operativo in un computer system



FISSA E APPLICA LE CONOSCENZE

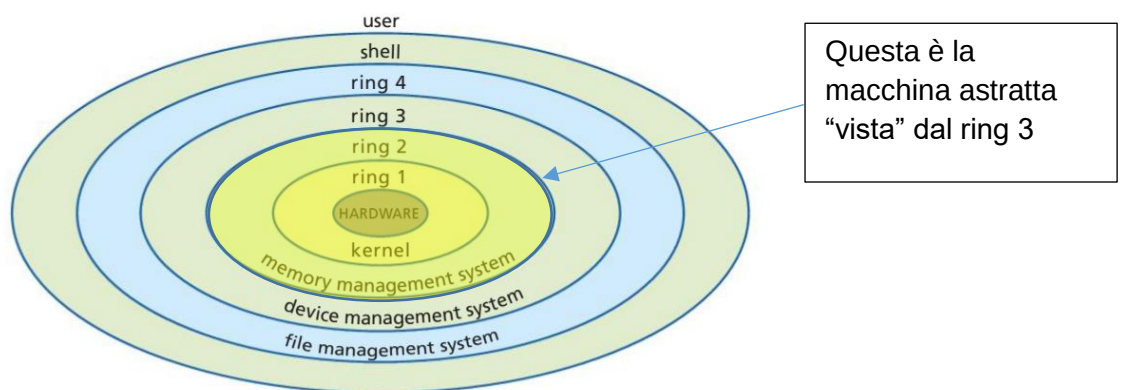
1. Che cos'è il kernel?
2. Quali sono i compiti che il kernel svolge per i processi in esecuzione?
3. Che cosa caratterizza le shell a linea di comando?
4. Descrivi gli elementi caratteristici di una GUI. Su quale metafora si basano?

IL MODELLO A RING CON USER MODE E KERNEL MODE Par.3

Il modello a ring (par. 3.1)

Il SO non è un semplice software, è un sistema complesso costituito da più software. Per semplificare la progettazione e la realizzazione, ciascuno dei software del sistema operativo fa parte di un modulo.

I moduli sono 5 e, per comprenderne il funzionamento, vengono sempre rappresentati con un modello chiamato modello a ring o modello a cipolla:



Due moduli li abbiamo già visti, il più interno (kernel) e il più esterno (shell). Il kernel è il modulo che sta a diretto contatto con l'hardware, la shell è il modulo con il quale interagisce l'utente.

Ciascuno degli altri tre moduli si occupa di attività specifiche che vedremo in seguito.

Per ora si sappia che i moduli sono, in ordine dal kernel verso la shell

- Il gestore della memoria (memory management system)
- Il gestore dei dispositivi (device management system)
- Il gestore del file system (file management system o semplicemente file system).

Perché il SO viene rappresentato in questo modo?

Perché rappresenta le seguenti caratteristiche con cui viene realizzato un SO:

- I software di ciascun ring "vedono" una macchina astratta definita dal ring sottostante.
- La macchina astratta sottostante fornisce i propri servizi al ring soprastante
- Ciascun ring può chiedere dei servizi alla macchina astratta sottostante. Facciamo un esempio: il modulo "file System" contiene il software che consente di creare un nuovo file. Per creare un nuovo file chiederà alla macchina astratta sottostante di eseguire **fisicamente** questa operazione nell' SSD e quindi di individuare in quali celle della memoria di massa scrivere i bit che costituiscono il file da creare. Questa operazione richiederà delle operazioni del kernel, infatti è solo il kernel che può agire direttamente sull' HW ma questo è irrilevante per il modulo "file System", infatti il modulo "file system" deve solamente chiedere il servizio "scrivi un file con questi dati sull' SSD" alla macchina astratta sottostante.

Quali sono i vantaggi di questa organizzazione a ring?

1. **Facilità nella progettazione del sistema operativo.** I livelli più esterni, infatti, non conoscono il funzionamento dei livelli più interni, ne invocano i servizi attraverso delle funzioni che i vari livelli mettono a disposizione del livello ad essi più esterno.

2. **Facilità di manutenzione.** Un'eventuale modifica di un modulo software che si trova in un determinato livello non richiede che la modifica coinvolga l'intero sistema operativo. E' sufficiente modificare quel modulo software. Ad esempio quando viene sostituita una scheda video, il suo driver viene sostituito nel kernel. Gli altri moduli del SO non devono essere modificati. (I driver delle periferiche fanno generalmente parte del kernel, anche se questo non è vero per tutti i sistemi operativi e per tutte le periferiche).

User mode e kernel mode (par. 3.2)

Gli insiemi di istruzioni che svolgono un determinato compito sono chiamati anche **routine**. User mode e kernel mode sono due modi con le quali possono essere eseguite le **routine** da parte della CPU.

Kernel mode: è una modalità protetta, nel senso che la CPU non può essere interrotta da altri processi quando esegue delle routine in kernel mode. E' la modalità che ha privilegio più alto nell'utilizzo dell'HW. Il software che fa parte del kernel del SO viene sempre eseguito in kernel mode. Anzi, ora che sappiamo cosa significa kernel mode, possiamo definire il kernel del SO nel seguente modo **“il kernel è l'insieme dei componenti software del SO eseguiti in kernel mode”**.

User mode (modalità utente) è il modo con il quale vengono eseguite dalla CPU le routine dei software applicativi e alcune routine del SO (non quelle del kernel), tali istruzioni hanno un livello di privilegio minore nell'utilizzo del HW. Quando vengono eseguite routine in user mode, la CPU può essere interrotta per eseguire altre istruzioni con privilegi maggiore (quelle del kernel). Ad esempio, quando viene eseguito il codice binario di un software applicativo che svolge una routine matematica (user mode), la CPU può essere interrotta per eseguire delle istruzioni in kernel mode. Al termine delle istruzioni in kernel mode, la CPU riprenderà l'esecuzione della routine matematica dal punto in cui era stata interrotta.

Come fa la CPU a sapere se una routine deve essere eseguita in kernel mode o in user mode? La CPU contiene un flag che viene posto a 0 (kernel mode) o a 1 (user mode) da parte del SO.

Le system call e le API del sistema operativo

Come già detto, le applicazioni per essere eseguite hanno bisogno di servizi che “chiedono” al Sistema Operativo, ad esempio possono chiedere di accedere ad un file o trasmettere i dati ad un dispositivo di output. Le applicazioni, per chiedere questi servizi utilizzano delle funzioni swl SO chiamate **system call (chiamate di sistema)**. **Le system call vengono sempre eseguite dalla CPU in kernel mode.**

Per facilitare il lavoro di chi scrive le applicazioni, il SO mette a disposizione dei programmatori delle funzioni che sono “più facili” da utilizzare rispetto alle system call, tali funzioni sono raccolte in librerie chiamate **API (Application Program Interface)**. Le funzioni delle API vengono eseguite in user mode e alcune di esse, nel loro codice, invocano delle system call, che vengono eseguite in kernel mode. Ad esempio: quando un programmatore utilizza la funzione printf(“CIAO”) in C, questa funzione dovrà chiedere al SO la risorsa “monitor” per mostrare sullo schermo la scritta CIAO. La funzione printf() fa parte delle librerie standard del C, è una funzione in linguaggio C che invoca nel suo codice in C una funzione del SO che fa parte delle API (nel caso del SO windows invoca la

funzione WriteConsole()). Tale funzione delle API , richiama una routine del SO eseguita in kernel mode per l'accesso all'hardware.

Sistemi operativi microkernel, monolitici, ibridi

L'approccio modulare e a ring vale in generale in tutti i SO. Vi sono però molte differenze fra i diversi SO in base a quali funzioni dei vari moduli vengono eseguite in kernel mode e in quali in user mode. In base a questo, i SO si classificano nel seguente modo.

SO microkernel: il kernel contiene solo le funzionalità essenziali della gestione del processore e della memoria, tutto il resto è eseguito in user mode. Hanno prestazioni inferiori agli altri SO poiché ogni operazione che richiede l'utilizzo dell'hw viene eseguita con una system call. Sono molto stabili poiché eventuali crash nei moduli non causano la necessità di riavviare il SO ma solamente il modulo in questione

SO monolitici: tutti i moduli fanno parte del kernel e quindi possono accedere direttamente all'hardware. Il SO è dunque un unico file binario che contiene tutti i moduli. Le performance sono molto elevate ma il rischio di instabilità è molto alto poiché un malfunzionamento in un qualsiasi modulo (ad esempio nel driver della penna USB) può causare il crash dell'intero SO e la necessità di ricompilare l'intero SO. Linux è monolitico, è possibile selezionare quali moduli aggiungere, tali moduli entrano a far parte del kernel.

SO ibridi, sono una via di mezzo fra microkernel e monolitico. Alcune funzionalità dei moduli sono eseguite in kernel mode per migliorare le performance, altre in user mode per garantire la stabilità. Sono più difficili da progettare. (Microsoft, MacOS, Android sono SO Ibridi).

FISSA E APPLICA LE CONOSCENZE

- | | |
|--|--|
| 1. Da cosa è costituita una macchina astratta? | 4. Quali sono le funzioni del file management? |
| 2. Quali sono le funzioni del memory management? | 5. Che differenza c'è tra user mode e kernel mode? |
| 3. Quali sono le funzioni del device management? | 6. Che cosa sono le system call? |

LE TIPOLOGIE DI SO in base a come elaborano i processi. Par.4

Si possono distinguere diverse tipologie di SO in base a come essi **elaborano i processi**.

Le tipologie di SO sono le seguenti:

- ad elaborazione centralizzata o distribuita
- elaborazione a lotti (batch) o interattiva
- monoprogrammazione o multiprogrammazione
- elaborazione in time-sharing o in real-time

Elaborazione centralizzata o distribuita

Nei sistemi **informatici ad elaborazione centralizzata** è presente un solo elaboratore molto potente chiamato **mainframe** al quale gli utenti si connettono attraverso terminali detti "**terminali non intelligenti**" poiché non muniti di capacità di calcolo (senza CPU). I dati, le applicazioni, la capacità di calcolo (CPU, GPU) sono presenti solamente nel mainframe. I terminali sono semplicemente dei dispositivi di I/O (mouse, tastiera, monitor, senza CPU e memoria di massa) che consentono di svolgere operazioni di I/O sul mainframe. (nei SO attuali, in realtà, anche i terminali possono essere dei PC). Il nome mainframe deriva dai grandi computer degli anni '60 del 1900 (frame=armadio). In seguito alla grande diffusione dei personal computer degli anni '80 e '90, e in seguito alla diffusione delle reti di computer e di internet, i sistemi centralizzati sembravano essere caduti in disuso, tuttavia sono ancora presenti dove si ha la necessità di elaborazione di grandi quantità di dati da parte di molti utenti, ad esempio banche, assicurazioni, enti pubblici, poiché i mainframe garantiscono elevate prestazioni in termini di capacità di calcolo, sicurezza, affidabilità. I mainframe sono solitamente modulari, ossia costituiti da più moduli, in modo da poter bilanciare su più moduli la potenza di calcolo. Il costo di un mainframe può raggiungere le centinaia di migliaia di euro.

I sistemi ad elaborazione distribuita.

Il problema del modello centralizzato dei sistemi informatici consiste nel fatto che se il mainframe si guasta, il sistema informatico non può funzionare (il sistema è poco robusto).

Per ovviare a questo problema e grazie all'evoluzione delle reti informatiche (LAN , WAN, Internet) i sistemi informatici si sono evoluti negli ultimi anni trasformandosi sempre più in sistemi ad elaborazione distribuita detti anche "**sistemi distribuiti**".

Un sistema distribuito è un sistema informatico in cui la capacità di calcolo (le CPU e GPU), l'hardware, i dati e le applicazioni sono distribuiti su diversi computer **collegati fra loro come nodi di una rete**, che cooperano per svolgere un compito e si presentano all'utente come un unico sistema informatico coerente. **Coerente** significa che, se più utenti modificano lo stesso dato, comunque quel dati avrà, in ogni istante un solo valore valido. Gli utenti utilizzano i servizi del sistema distribuito collegandosi ad esso attraverso delle reti informatiche e l'utilizzo delle risorse fornite dai sistemi distribuiti avviene in **maniera trasparente** per l'utente. Trasparente significa che l'utente, non si rende conto di operare su un sistema distribuito, infatti utilizza i servizi come se essi fossero presenti in locale sul proprio terminale. Diversamente dai sistemi centralizzati, gli utenti utilizzano terminali intelligenti, quindi dotati di una propria CPU, memoria, applicazioni che possono interagire con i sistemi distribuiti a cui sono collegati (per esempio scaricare un file da google drive in locale).

Ciascun utente collegato ad un sistema distribuito può utilizzare diversi servizi come se fossero collegati in locale, ad esempio:

- servizi hardware (una stampante di rete)

- potenza computazionale (CPU e GPU)
- Spazio di memorizzazione (storage) (ad esempio Google Drive)
- Applicazione (ad esempio i sw di Google drive)

I vantaggi di un sistema distribuito:

- Affidabilità: se un nodo si guasta, gli altri nodi possono farsi carico del suo lavoro.
- Scalabilità: se i nodi non sono sufficienti a garantire le prestazioni richieste il sistema informatico può ingrandirsi aggiungendo nuovi nodi.
- Chi lo utilizza non deve installare aggiornamenti.

Lo svantaggio principale è:

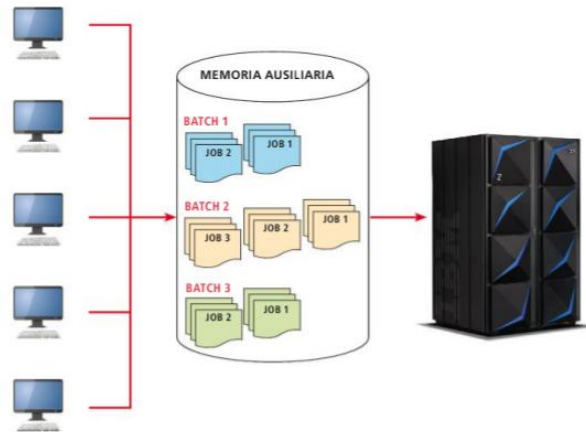
- Sicurezza: i dati sono distribuiti su diversi computer in una rete, quindi non sono in un unico computer nei locali del proprietario dei dati. Il proprietario dei dati non è a conoscenza di dove siano fisicamente i propri dati. Ci sono quindi più computer da proteggere rispetto ad un solo computer locale (sistema centralizzato).
- Dipendenza dalla rete.
- Maggiore complessità di gestione da parte di chi fornisce il servizio.

Esempi di sistemi di distribuiti sono:

- la web mail (ad esempio Gmail) perché i dati sono distribuiti su più computer (server), e vi sono server diversi che si occupano di compiti diversi per garantire il servizio (autenticazione, memorizzazione delle email, invio e ricezione delle email)
- Lo spazio di memorizzazione su Google Drive perché i dati sono distribuiti su più server e l'utente può gestire i propri file agendo come se fossero in locale.
- Giochi online multiplayer perché parti del gioco sono gestiti in locale (rendering grafico) e altri gestiti su più server (gestione degli account, stato del gioco)
- Sistemi di calcolo distribuito, che si dividono in
 - **cluster computing**, un insieme di pc **simili**, che si trovano nello stesso luogo , collegati da reti veloci , che si distribuiscono i compiti per eseguire **calcoli complessi** (ad esempio previsioni meteo, rendering grafici complessi, machine learning)
 - **grid computing**, sistemi distribuiti costituiti da diversi nodi eterogenei che si suddividono calcoli mettendo a disposizione del sistema la propria potenza di calcolo per risolvere **un problema comune**. Un esempio di Grid computing è GIMPS in cui diversi utenti volontari mettono a disposizione il proprio PC per svolgere dei “compiti di calcolo” assegnati e poi trasmettere i risultati ad un server. Lo scopo dei calcoli è trovare numeri primi di Mersenne, ossia del tipo $n=2^p-1$ dove P è a sua volta numero primo. Questi numeri sono molto utili nella crittografia.

Elaborazione a lotti (batch) o interattiva

Elaborazione a lotti: è tipica dei sistemi centralizzati, tutti i dati e i programmi necessari all'elaborazione vengono inseriti con i terminali in una memoria ausiliaria. Terminato l'inserimento di dati e programmi il computer centrale effettua l'elaborazione. Al termine dell'elaborazione il computer centrale dispone dei risultati che i terminali possono leggere.



Durante l'elaborazione non c'è alcuna interazione con l'utente, le operazioni di I/O vengono svolte solamente con i dati precedentemente caricati nella memoria ausiliaria.

Le elaborazioni da svolgere vengono chiamate **job** e vengono accorpate in gruppi chiamati **batch** (lotti).

Un esempio di elaborazione batch potrebbe essere la produzione della busta paga di un dipendente di un'azienda: durante tutto il mese vengono inseriti dati (ora ingresso, ora uscita, permessi, ferie ecc.). Al termine del mese tutti i dati sono inseriti e viene elaborata la busta paga.

In questi sistemi la CPU è utilizzata con grande efficienza poiché non viene mai interrotta durante l'elaborazione per svolgere operazioni di I/O.

Elaborazione interattiva: il sistema di elaborazione riceve continuamente dati di input, li elabora e fornisce risultati in output appena i dati sono disponibili. Ad esempio la videoscrittura di un testo con word o un videogame. In questo caso la CPU oltre ad elaborare i dati deve continuamente gestire le operazioni di I/O.

I sistemi Operativi moderni consentono entrambi i tipi di elaborazione e il funzionamento a lotti o interattivo dipende dall'applicazione utilizzata.

Osservazione: nel SO DOS il termine batch si riferisce anche ai file con estensione .bat. Sono file di testo che contengono una sequenza di comandi MS DOS. Una volta avviato il file, la shell esegue i comandi presenti nel file di testo in sequenza.

Esempio file prova.bat:

```
MD prova
CD prova
ECHO Ciao Ciao>miofile.txt
ECHO Scrivo in append >>miofile.txt
ECHO Lao Lao>> miofile.txt
```

Elaborazione monoprogrammazione o multiprogrammazione

Un processo è un programma in esecuzione. Per attivare un processo è necessario che il codice binario del programma da eseguire venga caricato nella memoria centrale (RAM) perché le istruzioni, per essere eseguite dalla CPU devono trovarsi nella memoria centrale.

Un SO che funziona in **monoprogrammazione** consente il caricamento nella MC di un solo programma alla volta. La CPU esegue le istruzioni di quel programma. Quando “quel” processo, per evolvere, necessita di operazioni di I/O per ottenere/comunicare dati, la CPU non fa nulla, attende che l’operazione di I/O si concluda e poi riprende l’esecuzione delle istruzioni (è l’esecuzione delle istruzioni da parte della CPU che fa evolvere il processo). Solamente quando il processo è terminato il SO carica un altro programma (oppure di nuovo lo stesso programma) nella MC ed inizia l’esecuzione di un altro processo in monoprogrammazione.

Un esempio sono i SO dei controlli numerici delle macchine utensili (un solo programma per volta viene eseguito).



Un SO che funziona in **multiprogrammazione** è in grado di caricare nella MC più processi. La CPU fa evolvere uno dei processi alla volta ma, nel momento in cui il processo necessita di un’operazione di I/O e quindi non può evolvere perché è in attesa di leggere o comunicare i dati, la CPU anziché rimanere inattiva, viene assegnata dal SO ad un altro processo e fa evolvere quest’altro processo. Nel frattempo le operazioni di I/O del primo processo vengono svolte da un apposito dispositivo HW chiamato “**elaboratore di canale**” (è un vero e proprio processore che si occupa dell’I/O, attualmente poco utilizzato e sostituito dal meccanismo del **DMA** della MC. In questo modo la CPU è utilizzata in maniera più efficiente poiché anziché rimanere inattiva quando un processo necessita di operazioni di I/O, viene utilizzata per far evolvere altri processi. Questo è ciò che accade con i moderni SO **multitasking** in cui, infatti, vengono eseguiti più processi (task, compiti) contemporaneamente.



FIGURA 15 Utilizzo della CPU in monoprogrammazione

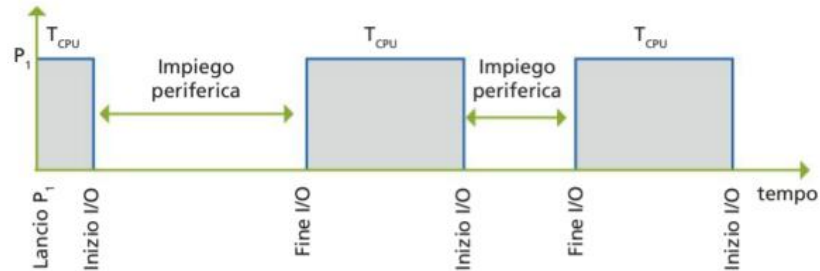
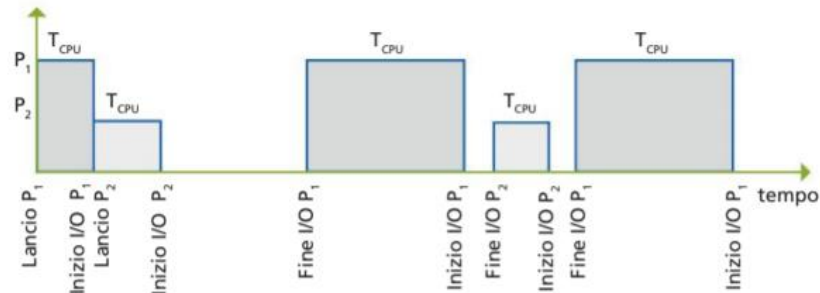


FIGURA 16 Utilizzo della CPU in multiprogrammazione



Elaborazione in time sharing o in real time

I SO time sharing sono quelli in cui vi è multiprogrammazione e ogni processo può utilizzare la CPU solo per un preciso intervallo di tempo (chiamato **time slice**), al termine del time slice il SO “assegna” la CPU ad un altro processo.

Perché ciò avviene? Per capire si guardi la precedente figura 16. Se il processo P1 non avesse mai bisogno di svolgere operazioni di I/O, la CPU sarebbe sempre “assegnata” ad esso e il processo P2 non potrebbe mai evolvere. Per evitare questa situazione, si è introdotto nei moderni SO il time sharing, in questo modo, dopo un certo time slice, il processore viene assegnato a P2 anche se P1 non è terminato e non deve svolgere operazioni di I/O. In questo modo tutti i processi in esecuzione, avranno a disposizione per un certo periodo la CPU per poter evolvere.

Nei SO che operano in time sharing quindi, un processo utilizza la CPU fino a che accade una di queste tre cose:

- Il processo termina
- Il processo richiede un'operazione di I/O
- Il processo termina il proprio time slice, allora interviene il SO ad assegnare la CPU ad un altro processo (operazione chiamata **context switching=cambio di contesto**). In seguito, prima o poi, il processore verrà nuovamente assegnato a questo processo

Quando i cambi di contesto sono troppo frequenti e quindi la CPU deve occuparsi più delle operazioni del SO che dell'evoluzione dei processi le prestazioni del computer degradano notevolmente, tale fenomeno è indicato con il termine **System Overhead**.

Con il termine **real time** si indica invece una modalità di esecuzione dei processi che richiede che un processo risponda alle richieste entro un certo intervallo di tempo chiamato **deadline**.

I sistemi operativi si dividono in **real time stretto (Hard Real-Time)**, e **real time largo (Soft Real-Time)**.

Nei sistemi real time stretti vengono eseguite applicazioni in cui la violazione del vincolo temporale (deadline) può compromettere l'incolumità di persone o cose, tali sistemi sono presenti generalmente in ambienti industriali. Ad esempio il sistema di frenata ABS o il sistema di controllo di un velivolo, o i controllori di reazioni chimiche o nucleari.

I sistemi real time larghi sono quelli in cui il mancato rispetto dei vincoli temporali da parte delle applicazioni in esecuzione è tollerato, tali violazioni degradano le prestazioni del sistema ma non il fallimento, ad esempio lo streaming, nel quale un rallentamento della quantità di dati scaricata peggiora la qualità dell'immagine ma la riproduzione comunque continua anche se con qualità peggiore.

I SISTEMI OPERATIVI IN COMMERCIO. Par.5

I SO si possono classificare in base ai dispositivi sui quali sono installati, in tre tipologie: SO desktop, SO di rete, SO per dispositivi mobili.

SO desktop

I SO desktop hanno le seguenti caratteristiche:

- vengono utilizzati da un solo utente per volta
- sono multitasking nel senso che consentono l'esecuzione di più applicazioni utente contemporaneamente.
- consentono di creare piccole reti in cui un computer può condividere alcune delle proprie risorse (cartelle, stampanti) con altri computer della garantendo un livello di sicurezza limitato

Il primo SO realizzato è stato UNIX, ideato nel 1969 nei famosi laboratori Bell della società americana AT&T, dove è nato anche il linguaggio C. UNIX viene considerato il capostipite dei SO moderni.

I principali SO desktop in commercio sono

- **Windows** della società Microsoft. I SO Windows possono essere installati su hardware prodotti da vari produttori, ad esempio su processori AMD, INTEL ecc. La società Microsoft è stata fondata da Bill Gates e Paul Allen nel 1975.
- **macOS** della società Apple. I SO MacOS possono essere installati solamente su hardware proprietario della società Apple. I fondatori sono stati Steve Jobs e Wozniak nel 1976.
- **GNU/Linux (perchè Lunux è solo il kernel)**, è un SO Open Source, ciò significa che il codice con cui è realizzato è reso disponibile a tutti in modo che chiunque possa modificarlo e distribuirne nuove versioni, per questo ne esistono diverse versioni, generalmente distribuite gratuitamente, chiamate distro, ad esempio Debian, Red Hat, Ubuntu. Questo sistema operativo è stato sviluppato a partire da un progetto di uno studente Finlandese di

nome Linus Torvalds il quale, nel 1991 ha messo a disposizione della comunità scientifica la prima versione del kernel aprendo il progetto al contributo volontario di tutta la comunità informatica. Il principio alla base della realizzazione di Linux è quello di una comunità che collabora per la realizzazione di un prodotto disponibile a tutti gratuitamente.

Poiché MacOS e Apple derivano in qualche modo da UNIX e riprendono delle sue caratteristiche, vengono indicati con il termine di UNIX Like. Windows non è UNIX Like.

SO di rete

I SO di rete vengono utilizzati per la gestione di reti di computer e sono chiamati anche NOS (Network Operating System). In una rete locale (LAN) vi sono computer che forniscono servizi, chiamati server, e computer che richiedono ed utilizzano tali servizi, chiamati client. I **NOS** sono installati sui server della rete, e consentono di gestire:

- spazi di archiviazione condivisi fra computer della rete
- hw condiviso (stampanti)
- utenti della rete e relativi permessi
- sistemi di archiviazione di dati ridondanti come raid e backup
- applicazioni server utilizzate dai client (ad esempio software gestionale ERP in un'azienda)

Esempi di NOS sono Windows Server (Windows), macOS Server (Apple), Red Hat Enterprise Linux, Ubuntu Server.

Sui server possono essere comunque installati applicativi utente in caso di necessità

SO per dispositivi mobili

Sono SO adatti per essere installati su smartphone e tablet. Consentono di eseguire programmi chiamati App. I principali sono due:

Android: è il più diffuso, è open source UNIX Like realizzato da Google, tutti i maggiori produttori di dispositivi mobili ad eccezione di Apple lo installano di default. L'offerta di App ampia ma non sempre di grande qualità.

iOS: è installato sui dispositivi mobili di Apple, iPhone e iPad. Il fatto che Apple controlli la distribuzione delle app e che sia il produttore sia dell'hw che del sw garantisce che il funzionamento del sw non abbia problemi di compatibilità e abbia ottime prestazioni. Lo svantaggio è il costo generalmente più elevato rispetto ai dispositivi Android.

LE PRESTAZIONI DEL SISTEMA OPERATIVO. Par 6

Un computer è costituito da HW e SW. La CPU è l'elemento hardware che deve eseguire le istruzioni di tutti i processi in esecuzione, sia i processi dei software applicativi sia il processo "sistema operativo". Un SO sarà tanto più efficiente quanto, a parità di HW, riesce ad far sì che vengano eseguiti il maggior numero di processi applicativi.

Alcuni parametri che vengono utilizzati per valutare l'efficienza di un SO sono i seguenti:

Throughput: indica il numero di lavori utenti (task, processi, job, thread) che vengono completati nell'unità di tempo. Si esprime processi al secondo oppure task al secondo.

Per calcolarlo si misurano i processi conclusi, ad esempio, in un'ora e si calcola:

$$\text{throughput} = N \text{ processi} / (T_{\text{totale}}) = N/3600.$$

Tournaround Time (TT): indica il tempo trascorso fra l'avvio di un processo e la sua terminazione.

$$TT = F - A \quad \text{dove } F = \text{istante di terminazione e } A = \text{istante d'inizio}$$

Dati N processi si può calcolare il TT medio, chiamato **ATT(AverageTournaroundTime)**

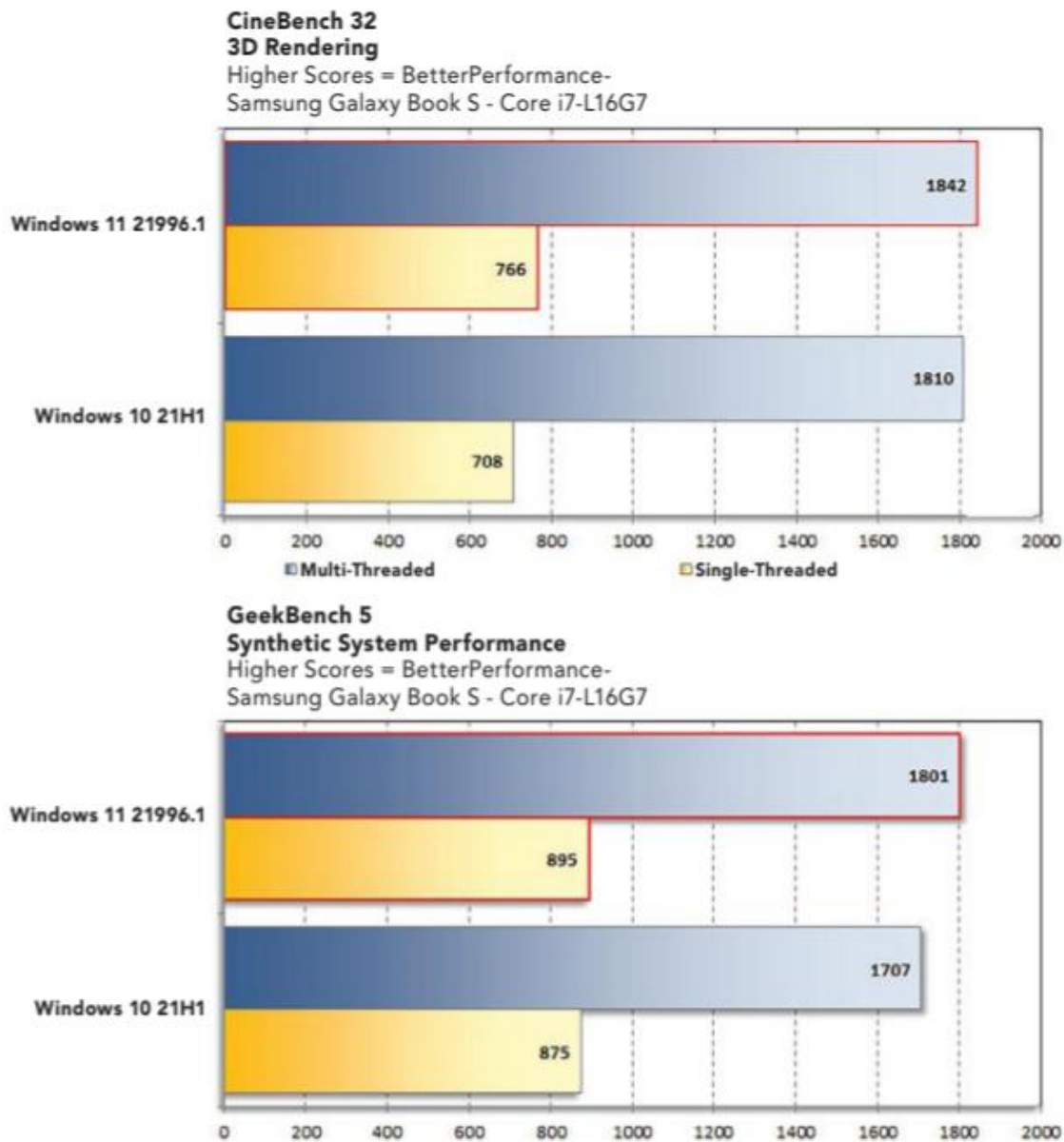
$$\sum_{i=1}^N TT_i / N$$

Benchmark: sono dei software specifici che, svolgendo opportune operazioni, consentono di valutare le prestazioni di un sistema informatico nel suo complesso, sia delle componenti HW sia del SO. Eseguendo benchmark su computer che hanno le stesse caratteristiche HW con SO diversi si possono valutare le prestazioni di un SO (con quello specifico HW).

Esistono benchmark diversi che misurano le prestazioni dei SO per diverse tipologie di operazioni permettendo di confrontare diversi SO nelle specifiche attività.

Ad esempio nelle seguenti immagini, due SO (Windows 10 e Windows 11) vengono confrontati con due diversi benchmark per valutare le loro prestazioni in due attività diverse:

Il primo grafico mostra il comportamento per la specifica attività di rendering 3D, Il secondo è un benchmark che valuta le generiche le prestazioni del SO eseguendo più processi.



In pratica i benchmark eseguono specifiche operazioni misurando i tempi di risposta del sistema informatico.

I benchmark sono software la cui validità e terzietà è riconosciuta dai produttori di HW e SW. Vengono utilizzati sia dai progettisti in fase di sviluppo dei sistemi informatici sia dagli utenti per confrontare i sistemi informatici prima dell'acquisto.

Ad esempio un benchmark molto noto è GeekBench (gratuito). Esistono benchmark per valutare le prestazioni dell'hardware nei videogame. Il termine benchmark è un termine generico utilizzato anche in ambiti diversi dall'informatica.