CAPITOLO 1

INTRODUZIONE AI SISTEMI OPERATIVI

- Sistemi Operativi - 8CFU – 72H

  Prof. Simone Tini.

  simone.tini@uninsubria.it

- Mercoledì ore 14.00-18.00, aula ???
- Giovedì ore 8.30-10.30, aula ???

- Propedeuticità: Programmazione e Architetture.

Testi consigliati.

- A. Tanenbaum: I Moderni Sistemi Operativi – 3 ed. Pearson – McGraw Hill.
- A. Tanenbaum: Modern Operating Systems – 3 ed.
- A. Silberschatz, G. Gagne, G.P. Baer: Sistemi Operativi – concetti ed esempi. Paerson – McGraw Hill.
- D.M. Dhamdhere: Operating Systems – A concept based approach. McGraw Hill.
- A. Ancillotti, M. Boari, A. Ciampolini, G. Lipari: Sistemi Operativi. McGraw Hill.
- Maurice J. Bach: The Design of the Unix Operating System. Prentice Hall. 3 copie in biblioteca.

Alcune considerazioni “ovvie”:

- Sul PC posso eseguire 14 programmi *contemporaneamente* (e.g.: 3 word, 3 explorer, 2 excel, 1 outlook, 1 nero, 4 JVM).

- Sul mio PC, che ha una RAM da 2 Gb, posso eseguire un programma da 1,2 Gb.

- Gli applicativi (Word, Photoshop, Nero, …) possono creare, leggere, modificare e cancellare file, memorizzati nell’hard disk.

- Sul mio PC posso avere 15 utenti, ognuno dei quali “vede” i propri file e programmi.

Alcune considerazioni “ovvie”. Sono così “ovvie”?

- Sul PC posso eseguire 14 programmi *contemporaneamente* (e.g.: 3 word, 3 explorer, 2 excel, 1 outlook, 1 nero, 4 JVM).
  Come è possibile? Ho una sola CPU, anche se dual core.

- Sul mio PC, che ha una RAM da 2 Gb, posso eseguire un programma che occupa 1,2 Gb.
  E se di programmi ne eseguo 8, per un totale di 9,6 Gb?

- Gli applicativi (Word, Photoshop, Nero, …) possono creare, leggere, modificare e cancellare file memorizzati nell’hard disk.
  What? L’hard disk non memorizza file, ma byte!

- Sul mio PC posso avere 15 utenti, ognuno dei quali “vede” i propri file e programmi.
  What? L’architettura del PC non prevede nulla di tutto ciò!

Alcune considerazioni "ovvie" Sono così "ovvie"? Certo!

- Sul PC posso eseguire 14 programmi *contemporaneamente* (e.g.: 3 word, 3 explorer, 2 excel, 1 outlook, 1 nero, 4 JVM).

  Come è possibile? Ho una sola CPU, anche se dual core.

  Si, ma il S.O. consente la ***multiprogrammazione***.
- Sul mio PC, che ha una RAM da 2 Gb, posso eseguire un programma che occupa 1,2 Gb.

  E se di programmi ne eseguo 8, per un totale di 9,6 Gb?

  Il S.O. realizza la ***memoria virtuale***.
- Gli applicativi (Word, Photoshop, Nero, …) e l'utente possono creare, leggere, modificare e cancellare file memorizzati nell'hard disk.

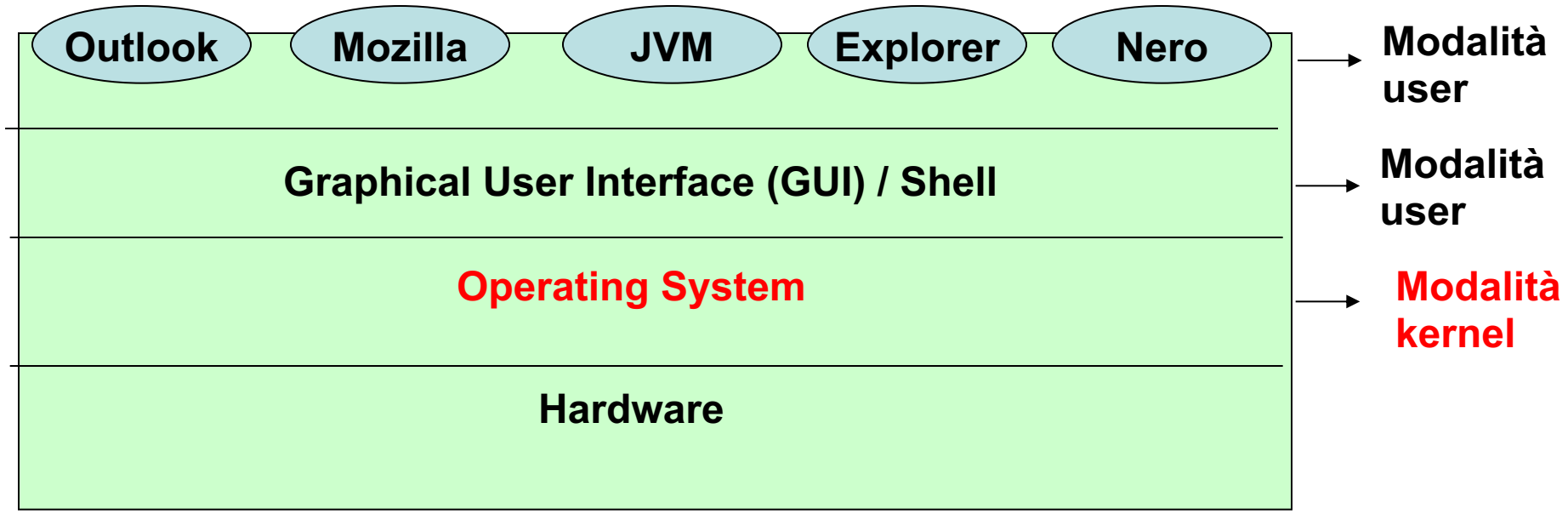
  What? L'hard disk non memorizza file, ma byte!

  Il S.O. gestisce le corrispondenze tra file e byte.
- Sul mio PC posso avere 15 utenti, ognuno dei quali "vede" i propri file e programmi.

  What? L'architettura del PC non prevede nulla di tutto ciò!

  Il S.O. realizza l'ambiente multiutente.

Cos'è il Sistema Operativo? Non esiste una risposta "ufficiale".



- L'hw ha 2 modalità operative: user e kernel, discriminate da un bit nel registro **PSW** (**P**rogram **S**tatus **W**ord) della CPU.

- In modalità user solo un sottoinsieme delle istruzioni macchina è disponibile.

- Il **S.O. esegue in modalità kernel**. Il resto in modalità user.

Potremmo dire che il S.O. è il software che esegue in modalità kernel. Ciò è vero, ma poco chiaro!

Alcune precisazioni:

- In questo momento io non sto interagendo direttamente con il S.O., ma con la **GUI** (**G**raphical **U**ser **I**nterface), la quale interagisce con il S.O..

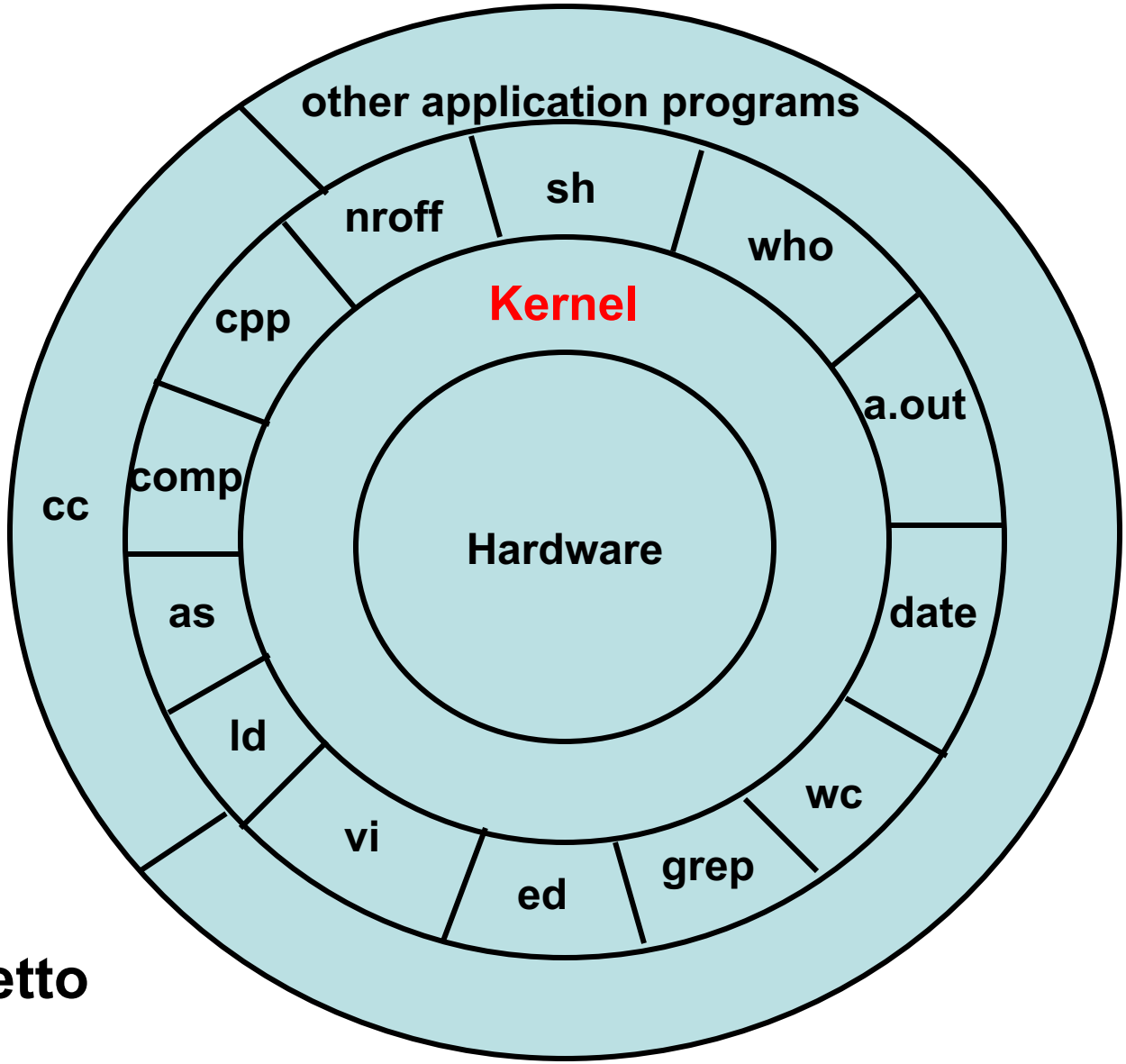
- La GUI esegue in modalità user e consente, ad esempio, all'utente di avviare/fermare altri programmi.

- La GUI non fa parte del S.O., anche se tutte le distribuzioni dei S.O. per PC comprendono una GUI.

- La GUI non è indispensabile: la shell/prompt dei comandi sarebbe sufficiente per interagire con il S.O..

- Spesso un sistema offre programmi che eseguono in modalità user e che aiutano il S.O. a svolgere determinati compiti (e.g. un programma per il cambio della password).

La visione di Unix V – 1986 (non abbiamo GUI, ma shell):



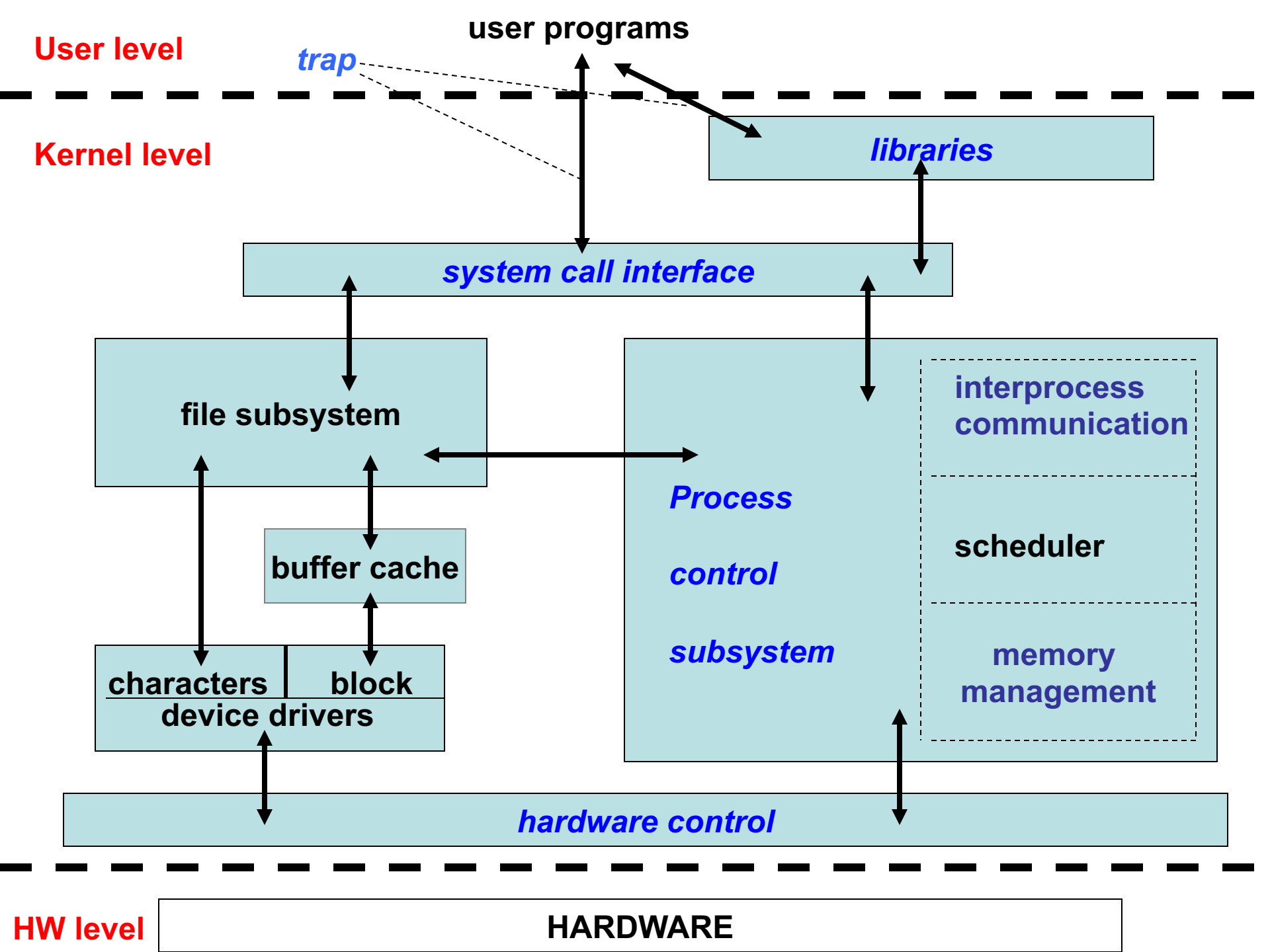
N.B.:
Il S.O. è detto Kernel

Vediamo la struttura del kernel di Unix System V nella prossima slide.

Questa slide va interpretata come un sovrainsieme del programma del corso, non si pretende che sia tutto già chiaro! In blu ciò che fa parte del corso.

N.B.: Il corso di S.O. non è un corso di Unix, però le funzioni che svolge Unix vengono svolte da tutti i S.O., ciò motiva il fatto che si possa interpretare la prossima slide per avere un'idea circa il programma del corso.

Osservazione: la struttura di Unix System V è *monolitica*.


User level
user programs
trap
Kernel level
libraries
system call interface
file subsystem
interprocess communication
Process control subsystem
scheduler
buffer cache
memory management
characters
block
device drivers
hardware control
HW level
HARDWARE

Commento alla slide precedente:

- Hardware control: comunicazione con l'hw mediante la lettura/scrittura dei registri del controllore (Cap. 3) e la gestione degli interrupt (Cap. 3).

- Device drivers: controllo dei dispositivi (a blocchi ed a caratteri) (argomento non trattato nel corso).

- Buffer cache: cache in memoria per i dispositivi a blocchi, quali, per esempio, il disco (argomento non trattato nel corso).

- File subsystem: realizzazione e gestione file (argomento non trattato nel corso). Osservazione: in Unix l'interfaccia verso i dispositivi è uguale all'interfaccia verso i file.

- Process control subsystem: ambiente a processi (Cap. 4-5):
  - scheduler: allocazione della CPU ai processi (argomento non trattato nel corso).
  - inter process communication: gestione comunicazioni tra processi (argomento parzialmente trattato nel corso – Cap. 6).
  - memory management: allocazione della memoria al S.O. ed ai processi (Cap 7).

- System calls: strumento usato dai programmi per chiedere l'intervento del S.O. (Cap. 3).

- Libreraries: servono per facilitare l'invocazione delle system call (Cap. 3).

- ***trap***: istruzioni macchina usate per implementare le system call (Cap. 3).

Torniamo a qualche slide fa. Cos'è il Sistema Operativo?

Un S.O. è un programma, opportunamente strutturato, che:

1. controlla l'esecuzione dei programmi applicativi, garantendo loro l'accesso a:
   - risorse fisiche (CPU, memoria, dispositivi)
   - risorse logiche (file, pipe, messaggi, semafori)

2. fornsice ai programmi applicativi un'interfaccia verso l'hardware, in modo che l'interazione con l'hardware non richieda la conoscenza dei dettagli architetturali.

Osservazione: in base ai punti 1 e 2, il S.O. è software che controlla l'esecuzione del software e fornisce al software un'interfaccia verso l'hardware!

E' chiaro che si tratta di software particolare.

Dalla definizione precedente emerge che il S.O. realizza una macchina virtuale, tale che:

1. Ogni applicativo "vede" una propria macchina, dotata di risorse fisiche e logiche.

2. Questa macchina è più facile da programmare rispetto a quella fisica.

In base a questi due punti, il S.O. può essere visto come:

1. Un gestore di risorse, fisiche e logiche, che vengono messe a disposizione degli applicativi.

2. Il realizzatore di una macchina estesa, messa a disposizione degli applicativi.

La prima è una visione top-down, la seconda bottom-up.

Come gestore di risorse, il S.O. deve, tra gli altri compiti:

- Assegnare ad ogni programma una CPU virtuale: la CPU fisica va assegnata "a turno" ai vari programmi.

- Assegnare ad ogni programma una memoria virtuale: ogni programma deve accedere al prorpio testo, ai propri dati, al proprio stack senza interferenze da parte di altri programmi.

- Assegnare ad ogni programma i propri dispositivi virtuali (stampanti, schede di rete,…): i dispositivi vanno condivisi tra i vari programmi evitando interferenze reciproche.

- Assegnare ad ogni programma le proprie risorse logiche.

Riassumendo, il S.O. deve consentire ai programmi di condividere le risorse, garantendo efficienza e protezione.

Come macchina estesa, il S.O.:

- Nasconde ai programmatori i dettagli dell'hardware, forndendo un'Application Program Interface (API) facile da usare.

- Favorisce la portabilità dei programmi. Per esempio, sostituire un CD drive richiede di modificare la realizzazione dell'API (installando il nuovo driver, non è necessario modificare tutto il S.O.), ma non l'API medesima. Ciò è trasparente per gli applicativi che interagiscono con la API. Considerazioni simili si applicano se facciamo girare il programma su una macchina diversa.

Obiettivi che il S.O. deve perseguire mentre realizza il proprio compito:

- Uso efficiente del sistema di computazione: le risorse vengono monitorate ed assegnate secondo criteri opportuni per sfruttarle al meglio.

  Questa attività introduce overhead: il S.O. consuma le risorse medesime.

  E' necessario trovare un equilibrio.

- User convenience: L'utente vorrebbe:
  - possibilità di eseguire programmi ed usare il file system
  - ottenere risposte veloci
  - facilità d'uso – interfacce amichevoli
  - strumenti per esperti – interfacce ad hoc

Non è detto che gli obiettivi siano compatibili, anzi sono in competizione tra loro.