Organizzazione e funzioni del SO

Sistema di protezione

- Un processo potrebbe tentare di modificare il programma o i dati di un altro processo o di parte del S.O. stesso.
- Protezione: politiche (cosa) e meccanismi (come) per controllare l'accesso di processi alle risorse del sistema di elaborazione
- Esempi:
  - l'hardware di indirizzamento della memoria assicura che un processo possa operare solo entro il proprio spazio di indirizzi.
  - l'I/O system impedisce l'accesso diretto ai dispositivi, etc.
- All'hardware é affidato il compito di *rilevazione* di errori, come op code illegali o riferimenti in memoria illegali, possibili effetti di errori di programmazione o di comportamenti deliberatamente intrusivi.
- Tali errori vengono segnalati e affidati alla gestione al S.O. tramite il meccanismo delle trap.
- In presenza di errore o di violazione della protezione il S.O. provvede a terminare il processo, segnala la terminazione anomala ed effettua un dump della memoria.

Sistema di protezione

- Ciascun processo opera in un *dominio di* protezione che specifica le risorse a cui il processo può accedere e le operazioni consentite.
- Diritto di accesso (coppia ordinata <risorsa, diritti>): abilitazione all'esecuzione di un'operazione sulla risorsa; un dominio di protezione è una collezione di diritti di accesso.
- Lista degli accessi di un oggetto: operazioni consentite da ciascun dominio.
- Lista delle *capabilities*: lista di oggetti e di operazioni consentite su tali oggetti riferita ad un dominio. Per eseguire una operazione su un oggetto il processo deve specificare la capability (indirizzo protetto mantenuto dal S.O.).

Esempio di capability

- Una *capability (a volte anche denominata key* – chiave) è un token di autorità immodificabile dai processi ma che può essere comunicato tra essi. Si tratta di un valore che è un riferimento a un oggetto e a un insieme associato (lista) di diritti di accesso. Un programma utente in un SO basato su *capability* deve necessariamente utilizzare una *capability* per accedere ad un oggetto
  Supponiamo che nella memoria di un processo ci sia la stringa

  "/etc/passwd" essa identifica univocamente un oggetto (file) nel sistema ma non specifica i diritti di accesso e quindi non è una capability

  Supponiamo che ci siano questi due valori:

  "/etc/passwd" **O_RDWR** essi identificano un oggetto e un insieme di diritti di accesso (lettura e scrittura in UNIX). Non si tratta tuttavia di una capability dato che il possesso da parte del processo di questi valori non indica se quell'accesso sia autorizzato dal SO
- Supponiamo che processo esegua la system call `open` con successo (cfr. lucidi UNIX) :

```
int fd = open("/etc/passwd", O_RDWR);
```

- La variabile **fd** ora contiene l'indice di un file descriptor nella tabella dei file aperti del processo. Questo file descriptor è una *capability* : infatti la sua esistenza nella tabella dei file aperti del processo è sufficiente per sapere che il processo ha veramente un accesso legittimo all'oggetto.
- Un aspetto fondamentale è che la tabella dei file aperti è mantenuta in memoria di kernel (associata al processo) e non può essere direttamente manipolata dal processo utente, come si vedrà nella parte UNIX del corso.

Protezione

- Il sistema di protezione richiede l'esistenza di *più modi di funzionamento della CPU:*
  - *supervisor mode* (detto anche system mode, monitor mode)
  - *user mode*
- Il passaggio dal modo user al modo supervisor avviene tramite *interruzione :*
  - esterna (asincrona)
  - interna (sincrona, trap), generata da una SVC (SuperVisor Call o System call).
- Il passaggio dal modo supervisor al modo user avviene tramite una istruzione speciale di *cambiamento di modo* eseguita dal S.O. prima della cessione del controllo ad un processo di utente.
- Istruzioni *privilegiate* (eseguite solo in system mode):
  - I/O
  - modifica dei registri che delimitano le partizioni logiche di memoria
  - manipolazione del sistema di interruzione
  - cambiamento di modo
  - halt
- Protezione della memoria (registri barriera, registri limite)
- Protezione della CPU (time limit nello scheduling round-robin)

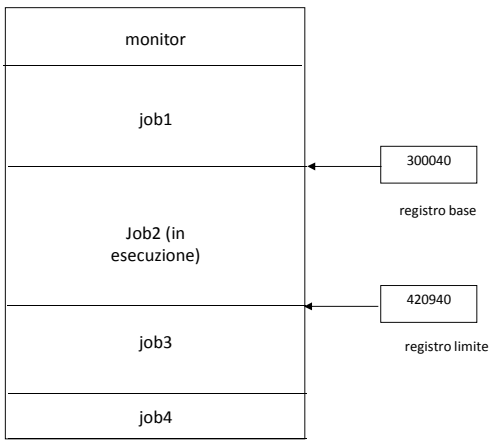
Protezione della memoria con registri limite

- Prima di mettere un processo in esecuzione, il S.O. ne confina lo spazio logico di memoria mediante registri limite:

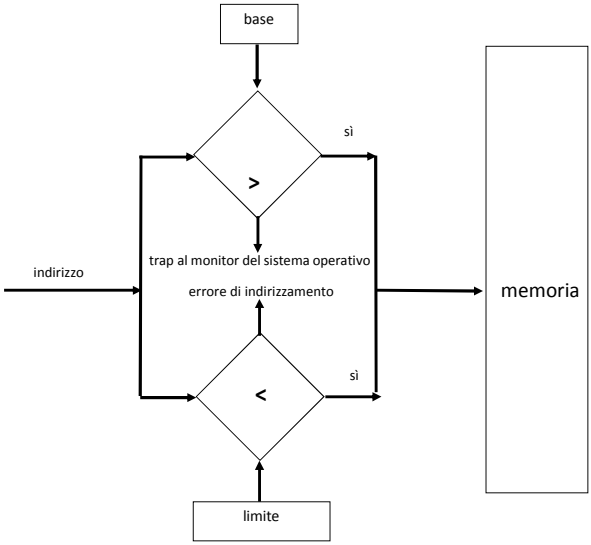
N.B.

E' un esempio elementare e obsoleto di tecnica di protezione della memoria

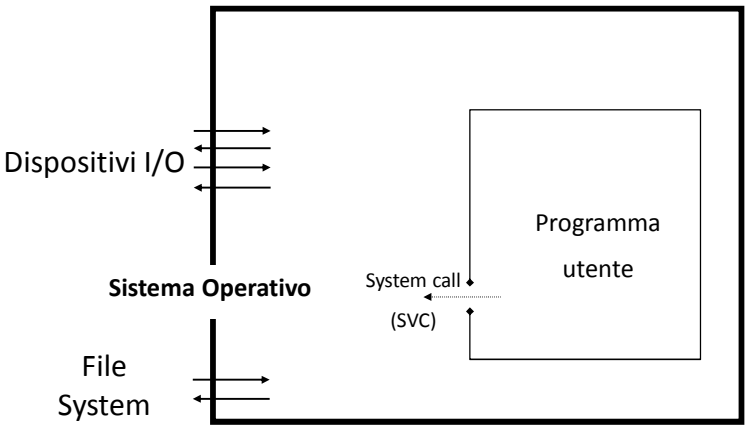
Le CPU moderne incorporano una MMU (memory management unit) che viene gestita dal S.O. per ottenere una protezione della memoria più fine (pagine da 4k) e flessibile all'interno della gestione della memoria virtuale



Protezione della memoria con registri limite



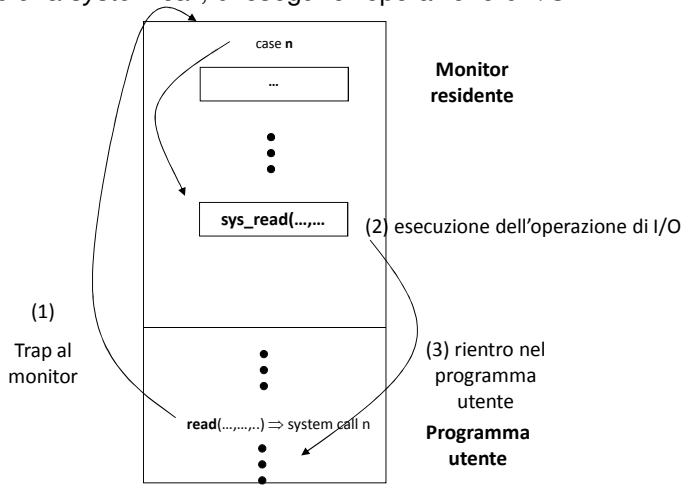
Confinamento del programma utente



Il programma utente non può accedere direttamente alle risorse del sistema ma solo attraverso l'intermediazione del S.O.

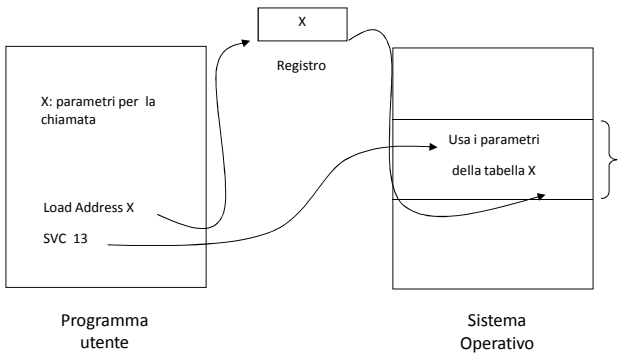
Esecuzione delle operazioni di I/O

- Le istruzioni di I/O sono privilegiate. Il programma utente richiede *al S.O., tramite una system call,* di eseguire l'operazione di I/O.



System call

- Tipicamente l'operando della istruzione di system call ne specifica il tipo (INT n), mentre il passaggio degli eventuali parametri avviene tramite registri o per indirizzo.
- Passaggio dei parametri mediante tabella:



System call

- Costituiscono *l'interfaccia tra un programma in esecuzione ed il S.O.*
- Istruzioni assembly, procedure chiamabili da linguaggi di alto livello. Nei linguaggi di alto livello sono tipicamente mascherate dal supporto a tempo di esecuzione all'interno di librerie.
- Categorie principali di system call:

  a) controllo dei processi e dei job
  b) manipolazione dei file e dei dispositivi
  c) gestione delle informazioni
  d) comunicazione

System call

a) controllo dei processi e dei job
- end, abort
- load, execute
- create process, terminate process
- get, set process attributes
- wait for time, wait for event, signal event
- allocate, free memory
- dump, trace

b) manipolazione dei file e dei dispositivi
- create, delete file
- open, close
- read, write, reposition file or device
- get, set file or device attributes
- request, release device

c) gestione delle informazioni
- get, set time or date
- get, set system data
- get, set attributes

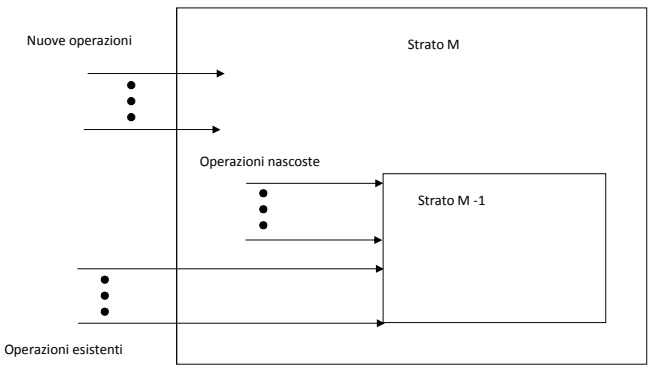
d) comunicazione
- create, delete communication connection
- open, close communication
- send, receive message

Programmi di sistema

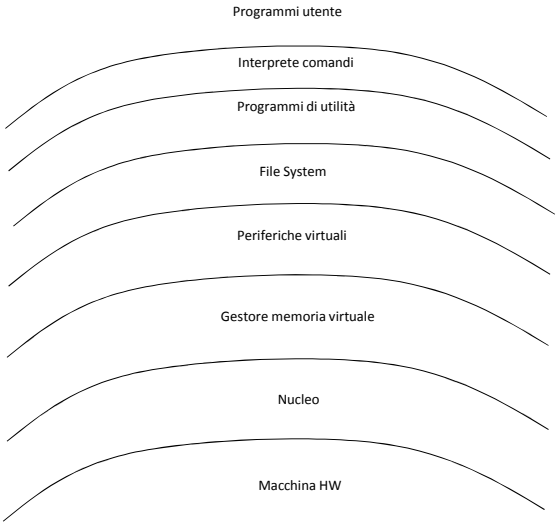
- Di varia natura, ad esempio in UNIX/Linux :
  - manipolazione dei file (editor, cp, mv, rm, mkdir, ...)
  - Informazioni di stato (date, time, who, df, ...)
  - sviluppo software ed esecuzione (traduttori, linker e loaders debuggers)
  - comunicazione (ssh, ftp, mail)
  - applicativi (spreadsheet, latex, ...)
  - *interprete dei comandi*: esegue i comandi realizzati come programmi di sistema speciali
- I programmi di sistema sono determinanti per la *visione d'utente* del S.O., mentre le system call ne riflettono la struttura interna.
- La progettazione della interfaccia con l'utente è indipendente dalla struttura interna del S.O.

Struttura del S.O.

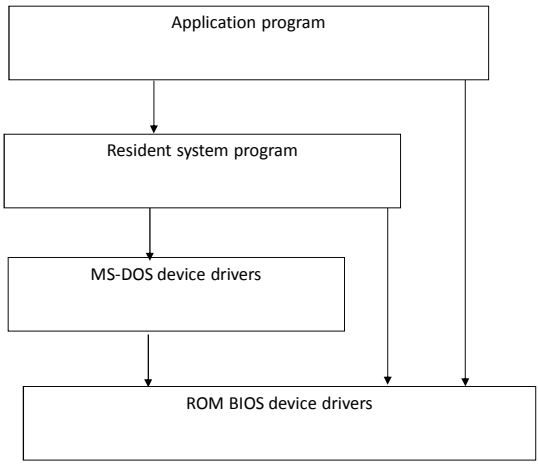
- Necessità di modularizzazione date le dimensioni
- Sistema a *livelli: il livello più interno è l'hw, quello più esterno l'interfaccia di utente*
- Affinché il livello L_j possa richiedere i servizi al livello L_{i-1} deve conoscerne una *specifica precisa, tuttavia l'implementazione di tali servizi* deve risultare totalmente nascosta.



La struttura "a cipolla"



La struttura a livelli di MS-DOS

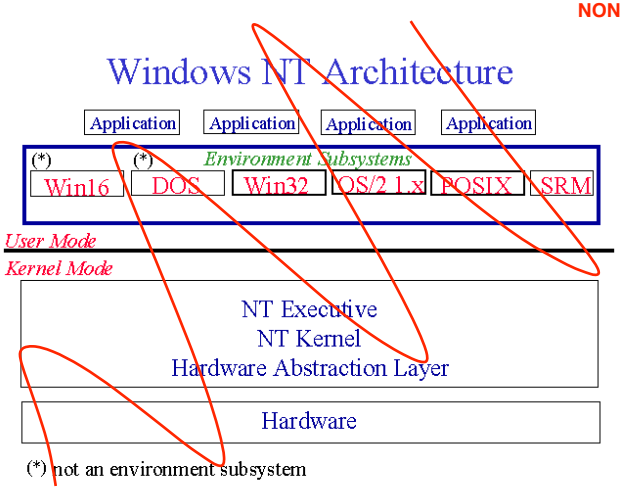


La struttura a livelli di UNIX

<table>
<tr><td colspan="3">(Users)</td></tr>
<tr><td colspan="3">Shells and commands
Compilers and interpreters
System libraries</td></tr>
<tr><td colspan="3">System-call interface to the kernel
Signals
Terminal handling
Character I/O system
Terminal drivers
File system
swapping
block I/O system
disk and tape drivers
CPU scheduling
Page replacement
Demand paging
Virtual memory
Kernel interface to the hardware</td></tr>
<tr><td>Terminal controllers
terminals</td><td>Device controllers
Disk and tapes</td><td>memory controllers
physical memory</td></tr>
</table>

La struttura a livelli di Windows NT

NON STUDIARE



Struttura del S.O.

- La stratificazione più opportuna può risultare non evidente; è dipendente dall'evoluzione tecnologica dell'hw.
- Sistema a *macchine virtuali* (VM IBM): usando lo scheduling della CPU e la tecnica della memoria virtuale, si possono creare macchine virtuali, una per ogni processo. Si consegue il massimo livello di protezione, a scapito dell'efficienza.
- Realizzazione in linguaggi ad alto livello (UNIX BSD4.3: 3% assembly, il resto in C)
- *Nucleo o kernel:* mette a disposizione le system call ai programmi di sistema ed applicativi.

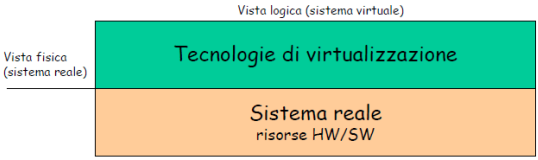
Macchine virtuali

- Macchine virtuali (VMWare, VirtualBox, xen, Java?, .NET?) sono la logica evoluzione dell'approccio a livelli.
  - Virtualizzano sia hardware che kernel del SO
- Creano l'illusione di processi multipli , ciascuno in esecuzione sul suo processore privato e con la propria memoria virtuale privata, messa a disposizione dal proprio kernel SO, che può essere diverso per processi diversi

(slide tratte da quelle della Prof.ssa Ciampolini – UNIBO)

Virtualizzazione

- Dato un sistema caratterizzato da un insieme di risorse (hardware e software), virtualizzare il sistema significa presentare all'utilizzatore una visione delle risorse del sistema diversa da quella reale.
- Ciò si ottiene introducendo un livello di indirezione tra la vista logica e quella fisica delle risorse.



- **Obiettivo**: disaccoppiare il comportamento delle risorse hardware e software di un sistema di elaborazione, così come viste dall'utente, dalla loro realizzazione fisica.

Esempi di virtualizzazione

Astrazione: in generale un oggetto astratto (risorsa virtuale) è la rappresentazione semplificata di un oggetto (risorsa fisica):
  - esibendo le proprietà significative per l'utilizzatore
  - nascondendo i dettagli realizzativi non necessari.
- Es: tipi di dato vs. rappresentazione binaria nella cella di memoria
- Il disaccoppiamento è realizzato dalle operazioni (interfaccia) con le quali è possibile utilizzare l'oggetto.

Linguaggi di Programmazione. La capacità di portare lo stesso programma (scritto in un linguaggio di alto livello) su architetture diverse è possibile grazie alla definizione di una macchina virtuale in grado di interpretare ed eseguire ogni istruzione del linguaggio, indipendentemente dall'architettura del sistema (S.O. e HW):
  - Interpreti (esempio Java Virtual Machine)
  - Compilatori

Virtualizzazione a livello di processo. I sistemi multitasking permettono la contemporanea esecuzione di più processi, ognuno dei quali dispone di una macchina virtuale (CPU, memoria, dispositivi) dedicata. La virtualizzazione è realizzata dal kernel del sistema operativo.

Sistemi Operativi per la Virtualizzazione

- La macchina fisica viene trasformata in n interfacce

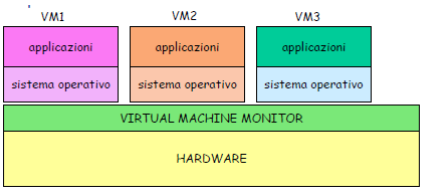
(macchine virtuali), ognuna delle quali è una replica della macchina fisica:

  - dotata di tutte le istruzioni del processore (sia privilegiate che non privilegiate)
  - dotata delle risorse del sistema (memoria, dispositivi di I/O).

- Su ogni macchina virtuale è possibile installare ed eseguire un sistema operativo (eventualmente diverso da macchina a macchina): **Virtual Machine Monitor**

Virtualizzazione di Sistema

- Una singola piattaforma hardware viene condivisa da più sistemi operativi, ognuno dei quali è installato su una diversa macchina virtuale.



- Il disaccoppiamento è realizzato da un componente chiamato Virtual Machine Monitor (VMM, o hypervisor) il cui compito è consentire la condivisione da parte di più macchine virtuali di una singola piattaforma hardware. Ogni macchina virtuale è costituita oltre che dall'applicazione che in essa viene eseguita, anche dal sistema operativo utilizzato.
- Il VMM è il mediatore unico nelle interazioni tra le macchine virtuali e l'hardware sottostante, che garantisce:
  - **isolamento** tra le VM
  - **stabilità** del sistema

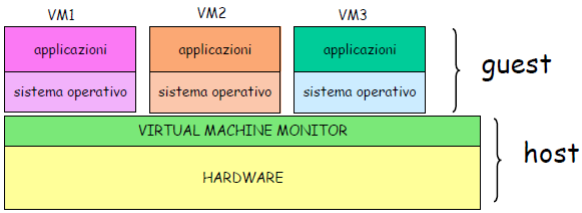
VMM di sistema vs. VMM ospitati

VMM di Sistema

- le funzionalità di virtualizzazione vengono integrate in un sistema operativo leggero, costituendo un unico sistema posto direttamente sopra l'hardware dell'elaboratore.
- E' necessario corredare il VMM di tutti i driver necessari per pilotare le periferiche.
- Esempi di VMM di sistema: Vmware ESX, xen, VirtualIron.

VMM di Sistema

- **Host**: piattaforma di base sulla quale si realizzano macchine virtuali. Comprende la macchina fisica, l'eventuale sistema operativo ed il VMM.
- **Guest**: la macchina virtuale. Comprende applicazioni e sistema operativo



VMM di Sistema

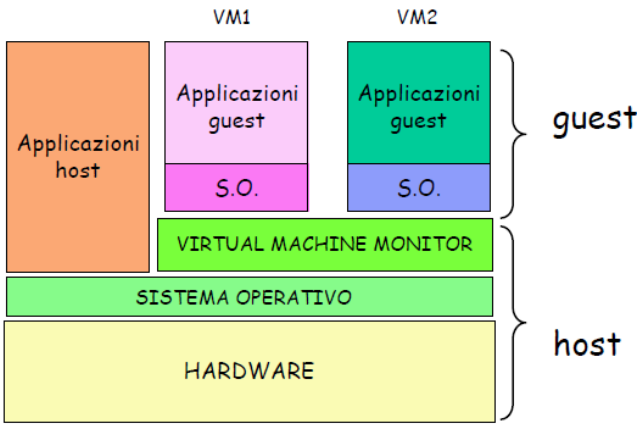
VMM ospitato

Il VMM viene installato come un'applicazione sopra un sistema operativo esistente, che opera nello spazio utente e accede l'hardware tramite le system call del S.O. su cui viene installato.

- Più semplice l'installazione (come un'applicazione).
- Può fare riferimento al S.O. sottostante per la gestione delle periferiche e può utilizzare altri servizi del S.O.(es. scheduling, gestione delle risorse.).
- Peggiore la performance.

- Prodotti: User Mode Linux, VMware Server/Player, Microsoft Virtual Server , Parallels

VMM ospitato



Emulazione vs Virtualizzazione

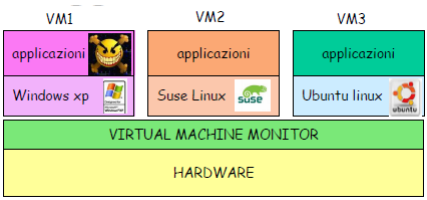
Emulazione

- eseguire applicazioni (o SO) compilate per un'architettura su di un'altra.
- uno strato software emula le funzionalità dell'architettura; il s.o. esegue sopra tale strato (a livello user). Le istruzioni macchina privilegiate e non privilegiate vengono emulate via SW. (Bochs, Qemu)

Virtualizzazione

- definizione di contesti di esecuzione multipli (macchine virtuali) su di un singolo processore, partizionando le risorse;

Vantaggi della virtualizzazione



- **Uso di più S.O. sulla stessa macchina fisica:** più ambienti di esecuzione (eterogenei) per lo stesso utente:
  - Legacy systems
  - Possibilità di esecuzione di applicazioni concepite per un particolare S.O.
- **Isolamento degli ambienti di esecuzione**: ogni macchina virtuale definisce un ambiente di esecuzione separato (sandbox) da quelli delle altre:
  - possibilità di effettuare testing di applicazioni preservando l'integrità degli altri ambienti e del VMM.
  - Sicurezza: eventuali attacchi da parte di malware o spyware sono confinati alla singola macchina virtuale

Vantaggi della virtualizzazione

- **Consolidamento HW**: possibilità di concentrare più macchine (ad es. server) su un'unica architettura HW per un utilizzo efficiente dell'hardware (es. server farm):
  - Abbattimento costi HW
  - Abbattimento costi amministrazione
- **Gestione facilitata delle macchine**: e` possibile effettuare in modo semplice:
  - la creazione di macchine virtuali (virtual appliances)
  - l'amministrazione di macchine virtuali (reboot, ricompilazione kernel, etc.)
  - migrazione a caldo di macchine virtuali tra macchine fisiche:
    - possibilità di manutenzione hw senza interrompere i servizi forniti dalle macchine virtuali
    - disaster recovery
    - workload balancing: alcuni prodotti prevedono anche meccanismi di migrazione automatica per far fronte in modo “autonomico” a situazioni di sbilanciamento

Nucleo di un S.O.

- Fornisce un meccanismo per la creazione e la distruzione dei processi
- Provvede allo scheduling della CPU, alla gestione della memoria e dei dispositivi di I/O
- Fornisce strumenti per la sincronizzazione dei processi
- Fornisce strumenti per la comunicazione tra processi

Struttura gerarchica del Sistema Operativo

- L0:bare machine
- L1:processor management (lower module) / scheduler
- L2:memory management
- L3:processor management (upper module) [messaggi, creazione/distruzione processi]
- L4:device management
- L5:information management