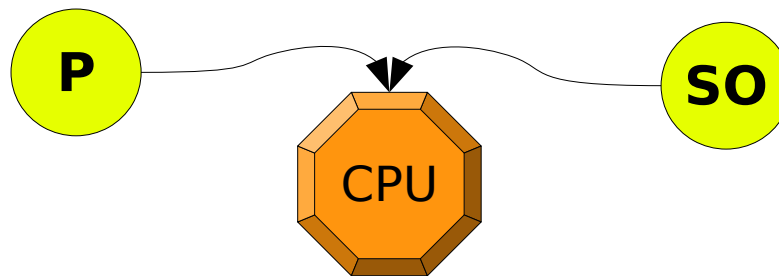


Dual Mode

Introduzione

- In ogni istante la CPU eseguirà istruzioni appartenenti o a un programma utente o al SO

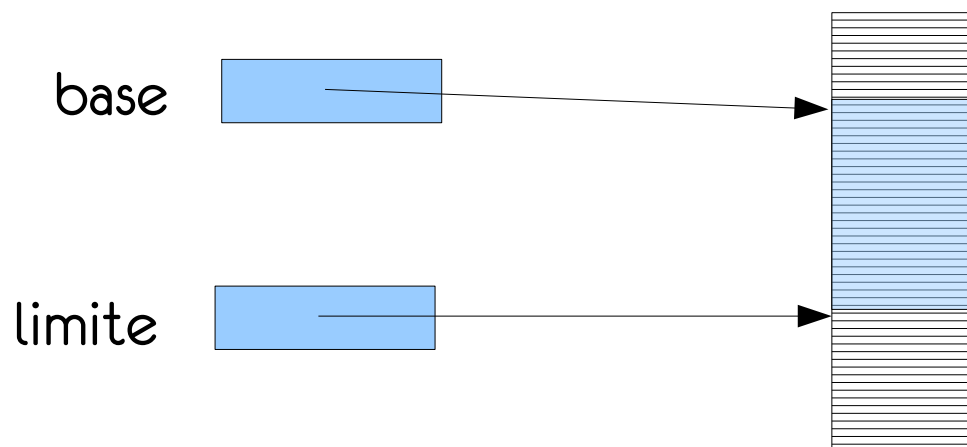


- la CPU ha il proprio **instruction set**
- Possiamo lasciare che tutte le istruzioni dell' instruction set siano invocate direttamente da qualsiasi processo o può essere problematico?

Introduzione

Esempio:

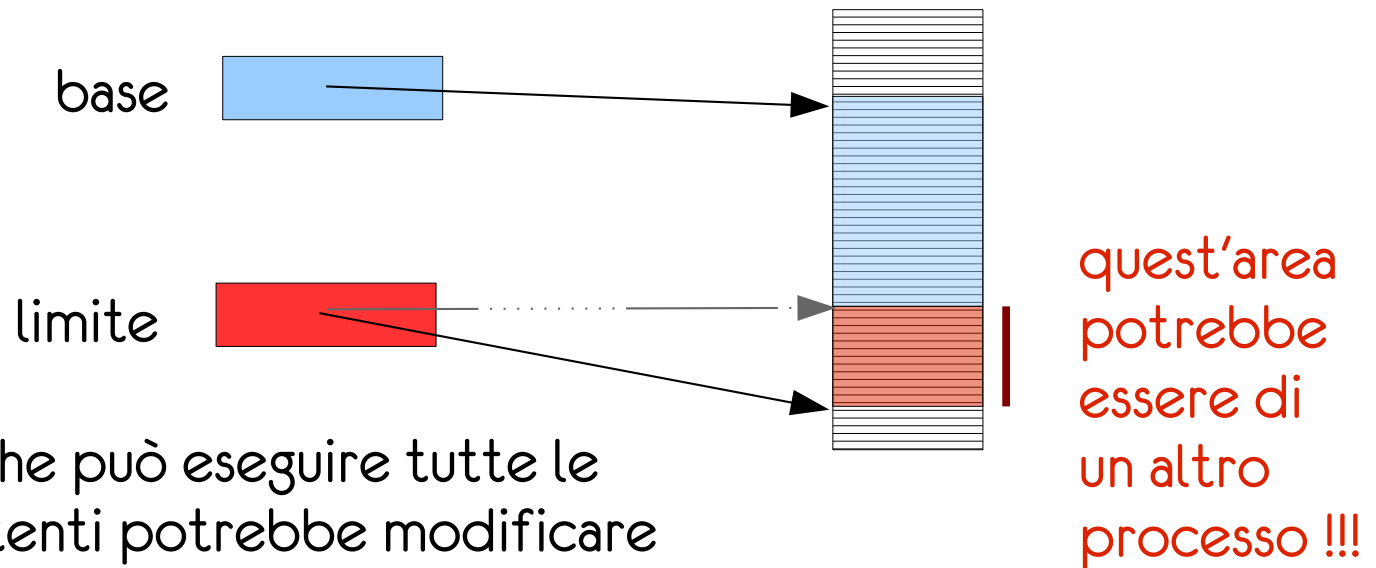
certi processori attuano una protezione della memoria basata su questo principio: **l'area di memoria modificabile da un processo è identificata dal contenuto di registri (base e limite),** che possono essere modificati solo tramite istruzioni privilegiate.



Introduzione

Esempio:

certi processori attuano una protezione della memoria basata su questo principio: **l'area di memoria modificabile da un processo è identificata dal contenuto di registri (base e limite)**, che possono essere modificati solo tramite istruzioni privilegiate.



un processo che può eseguire tutte le istruzioni esistenti potrebbe modificare arbitrariamente il valore di tali registri !!!

Introduzione

Impedire ai processi utenti di eseguire le istruzioni che consentono di cambiare i valori di tali registri significa **implementare una politica di protezione**

Esempio: evitare che un utente sovrascriva il SO

Un processo utente non può modificare l'area di memoria su cui scrivere perché non può eseguire le istruzioni necessarie a questo scopo

Modello a livelli e Dual mode

Il **modello a livelli** impone che certe istruzioni siano esclusive del sistema operativo:

- 1) Sono definite delle classi di processi (modalità di esecuzione)
- 2) Ogni modalità di esecuzione ha associato un sottoinsieme delle istruzioni del linguaggio macchina: le uniche eseguibili in quella modalità

Il **dual mode** in particolare definisce:

modalità utente **non** si ha accesso alle istruzioni di I/O

modalità kernel si ha accesso all'**intero** instruction set

Identifica la due modalità corrente il **bit di modalità**

Bit di modalità

bit di modalità vale:

= 0 se in modalità kernel (o supervisore)

= 1 se in modalità utente

Al bootstrap il bit di modalità è inizializzato a 0

Con la richiesta di esecuzione di processi utente, il bit di modalità cambierà valore, a seconda del processo che richiede l'esecuzione della prossima istruzione

Problema

Se in modalità utente non si possono eseguire istruzioni, per esempio, di I/O come possibile eseguire operazioni di input/output da un normale programma utente?

Soluzione

Un utente non può usare direttamente certe operazioni di basso livello ma può utilizzare all'interno dei propri programmi delle funzioni speciali che invocano un'esecuzione da parte del SO (**system call**).

Durante l'esecuzione di una system call il bit di modalità vale 0 (**modalità kernel**)

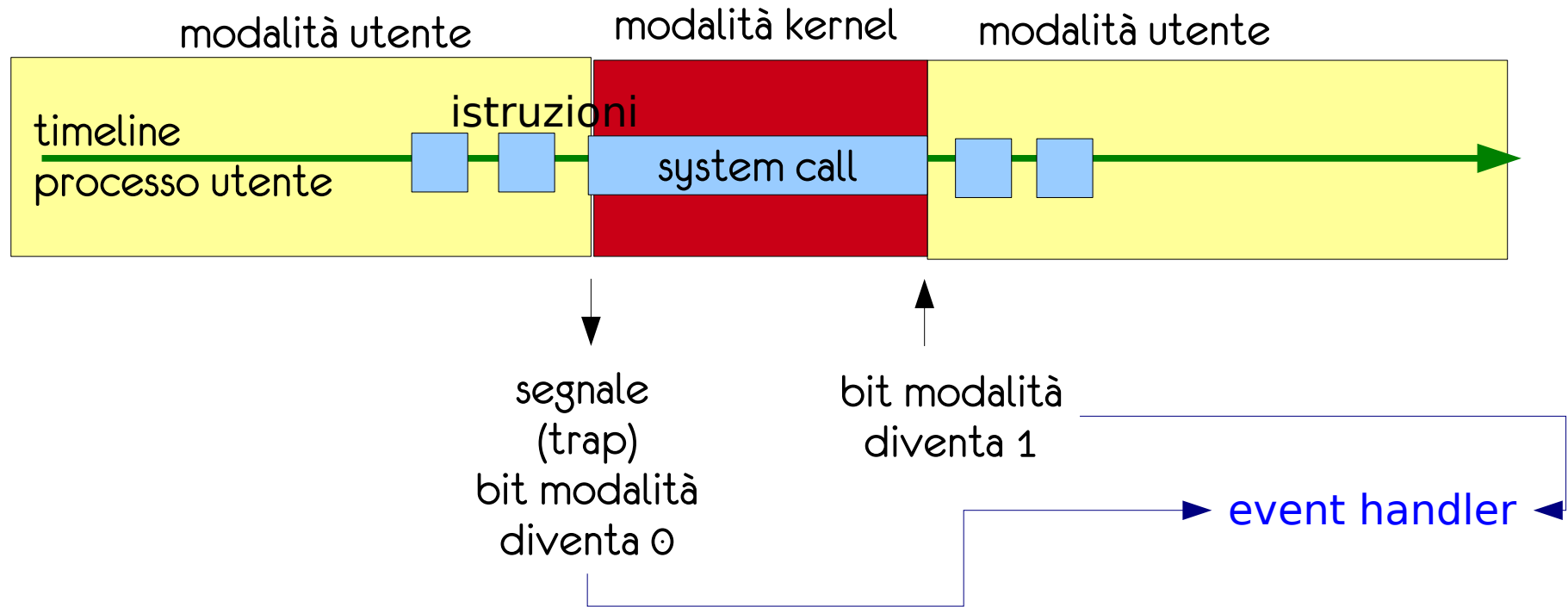
Dual mode: supporto HW

il modello a livelli è possibile solo se la CPU fornisce a livello HW un supporto adeguato

il bit di modalità deve essere fornito dall'HW

Nota: in generale si possono avere più di due modalità di esecuzione attuando meccanismi di protezione a grana più fine

System call e dual mode



system call: operazioni tramite le quali un programma utente richiede al SO di compiere per suo conto operazioni ad esso riservate.

System Call

- 1) Il compilatore C usa una libreria predefinita di funzioni aventi i nomi delle system call
 - 2) Le funzioni di libreria invocano un'istruzione che cambia la modalità di esecuzione a kernel facendo sì che il kernel avvii l'esecuzione del codice delle system call
 - 3) Questo cambiamento avviene tramite un'interruzione generata via software, detta "operating system trap"
- ...

System Call

4) Quindi la funzione di libreria esegue in user mode ma l'interfaccia per le system call avvia uno speciale event handler

5) La funzione di libreria passa al kernel un'identificatore univoco della system call richiesta in modo dipendente dalla macchina (es. Un parametro della trap inserito in un registro o sullo stack di esecuzione) che il SO utilizza per avviare l'handler giusto

6) Nella gestione il SO cerca il numero della system call in una tabella identificando così l'indirizzo della routine di gestione da eseguire

7)NB: gestione affine a quella degli interrupt

Avete mai usato una system call?

Avete mai usato una system call?

Cinque categorie di system call:

controllo dei processi

gestione dei file

gestione dei device (dispositivi)

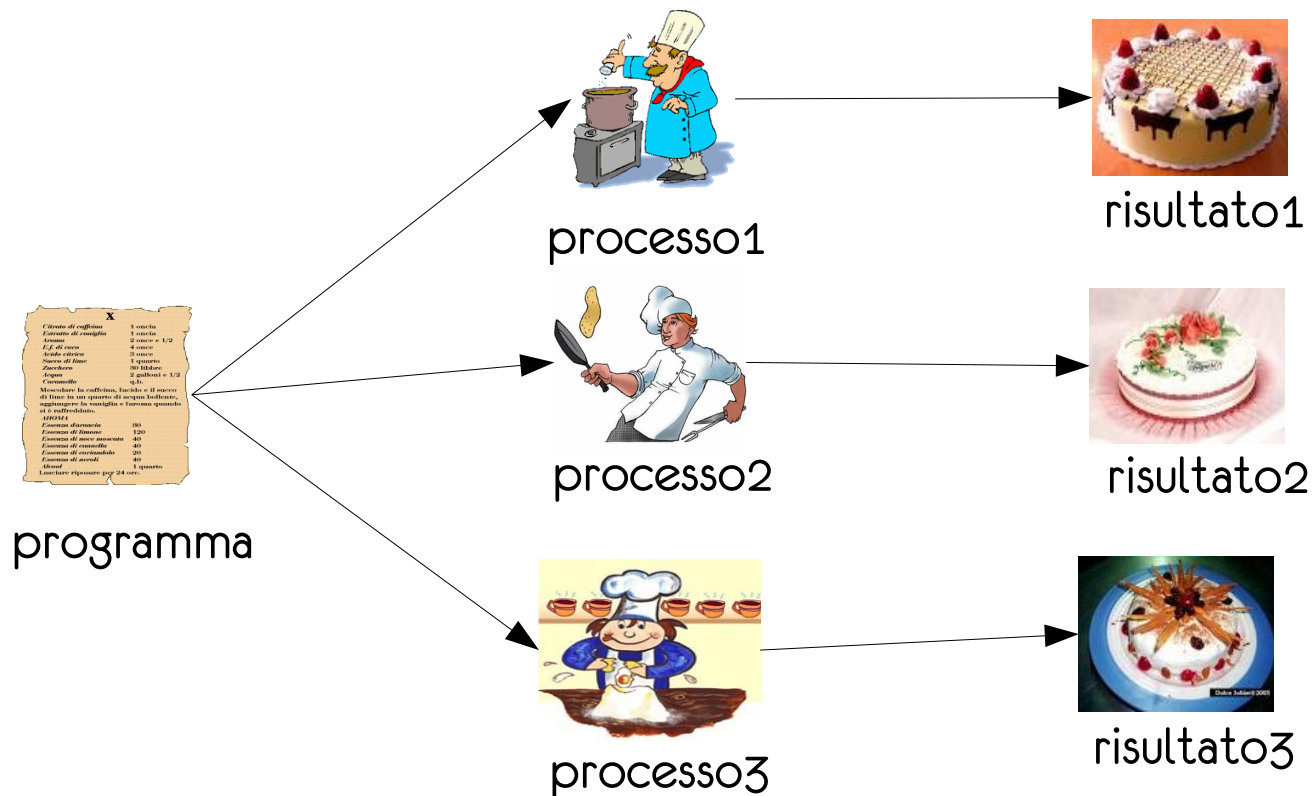
gestione delle informazioni

comunicazione

In UNIX sono definite oltre 300 system call

Cos'e` un processo?

un **programma**, è un file, un'entità passiva, non fa nulla
la sua esecuzione produce un'entità attiva, detta **processo**



possono condividere risorse
il cui uso va gestito

**NB: attenzione a non confondere
processo e processore!!! Ogni me-
tafora ha un limite**

Programmi e processi

- un processo adopera delle risorse: CPU, memoria, dispositivi, ...; alcune risorse sono condivise
- il **SO deve gestire i processi**, quindi deve:
 - poterli identificare
 - poterli creare e cancellare
 - fornire strumenti per la loro sincronizzazione e per la comunicazione
 - identificare e gestire situazioni di stallo (deadlock)
 - evitare starvation
- di ogni processo **viene mantenuta un'astrazione**
- **NOTA:** anche il SO esiste come processo!!!



Controllo dei processi

- la **terminazione** di un processo è gestita da una system call (perché ???)
- la terminazione anormale (**abort**) può causare la copiatura in un file dell'immagine del processo (**dump** o **core**), da ispezionare tramite un debugger
- in questo caso la CPU mette a disposizione una modalità di **esecuzione** detta **a passo singolo** che permette di tracciare istruzione per istruzione quanto eseguito
- un processo può richiedere il **caricamento di un programma** diverso da quello originario
- un processo può **generare un altro processo**
- ogni processo ha delle caratteristiche che lo definiscono. Sono disponibili system call per **(re)impostare tali caratteristiche o per leggerle**
- un processo può essere **sospeso** per un lasso di tempo o in attesa di un evento

Nota Bene

- il SO deve controllare gli altri processi e la CPU ma è un processo come tutti gli altri per quel che riguarda l'uso delle risorse, in particolare ha bisogno della CPU per lavorare ...
- se il SO non esegue non può effettuare le operazioni di gestione di sua competenza
- in un sistema multiprogrammato il SO “convive” con altri processi (processi utente), *dovrebbe avere uno stato privilegiato?* (non tanto a basso livello, dual mode, ma a livello di accesso alla CPU)
- **problema:** come può il SO riprendere il controllo della CPU, se questa è assegnata a un processo utente?
- **soluzione:** uso di **timer predefiniti**, causano un interrupt che restituisce il controllo al SO

Gestione dei file

- **creazione e cancellazione** di file
- **apertura e chiusura** di file
- **lettura**
- **scrittura**
- **riposizionamento** (reposition, effettua uno spostamento che non segue il normale criterio di accesso sequenziale)
- **spostamento** di un file nel file system
- **copiatura**
- **lettura/modifica delle caratteristiche** di un file

Gestione di device e informazioni

Dispositivi (es. CD, chiavi USB, nastri, ...)

request/release di un dispositivo

lettura/scrittura/riposizionamento

Informazioni, sono system call che tipicamente trasferiscono informazioni all'utente o a programmi dell'utente

ottenere la data e l'ora

ottenere informazioni sui processi

ottenere informazioni sui file

Comunicazione

- **Modello a scambio di messaggi**
 - identificare la macchina e il processo (get host id, get process id), tipicamente usate dal mittente e aggiunte al messaggio per consentire una eventuale risposta
 - apertura di una connessione da parte del mittente
 - chiusura di una connessione
 - accettazione di una connessione da parte del destinatario
 -
- **Modello a memoria condivisa (shared memory)**
 - allocazione di un'area di memoria condivisa
 - attach (aggancio) a un'area di memoria condivisa

Operazioni e chiamate di sistema

Supponiamo che un utente digiti in una shell il comando:

```
cp documento.odp destinazione/versione_finale.odp
```

che corrisponde a richiedere che il file documento.odp venga copiato in una certa cartella (destinazione) con il nome versione_finale.odp

Quante system call verranno eseguite per completare il comando?

Operazioni e chiamate di sistema

supponiamo che un utente digiti in una shell il comando:

```
cp documento.odp destinazione/versione_finale.odp
```

il codice corrispondente al comando cp effettua, in maniera trasparente all'utente una serie di chiamate di sistema facendo cambiare il valore del **bit di modalità** più volte:

apertura file sorgente

apertura file destinazione

ciclo di letture/scritture che implementano la copiatura

chiusura del file sorgente

chiusura del file destinazione

messaggio al terminale

terminazione del processo

queste system call sono eseguite anche quando l'utente interagisce tramite GUI, quel che cambia è il linguaggio di interazione che è grafico e non testuale

Ogni sys. call comporta un cambio di modalità (dual mode)

Quante system call usiamo?

Esempio

```
baroglio@esmeralda: ~/BACKUP/LAB_S0/Slide$ ls
2007_2008          memoriaVirtuale-1x3.pdf      slidesSis0p0607_processi_latest.odp
2008_2009          memoriaVirtuale.odp         slidesSis0p0607_processi.odp
deadlock.odp       memoriaVirtuale0ld.odp      slidesSis0p0809.odp
fileSystem.odp     memoriaVirtuale.pdf         slidesSis0p_intro_0607.odp
fileSystemRel.odp  memSecondaria-1x3.pdf      slidesS00809.tar.gz
memoria_harlock.odp memSecondaria.pdf
memoria.odp        slidesSis0p0607_processiEsme.odp
baroglio@esmeralda: ~/BACKUP/LAB_S0/Slide$
```

ls: elenca i file contenuti nella directory di lavoro

strace: comando che consente di listare le system call eseguite da un processo e i segnali gestiti dal medesimo

strace -cw: riassume in una tabella finale quanto intercettato

strace ls

The image is a screenshot of a Windows XP desktop environment. At the top, a KVM session window is open, titled "baroglio@esmeralda: /BACKUP/LAB_SO/Slide - Shell No. 5 - KvmSpole". The window's menu bar includes "Session", "Edit", "View", "Bookmarks", "Settings", and "Help". Below the menu bar, there are tabs for "Shell No. 3", "Shell", "Shell No. 2", "Shell No. 4", and "Shell No. 5". The active tab is "Shell No. 5", which displays a terminal window. The terminal shows the command "strace ls" and its output, which is a list of system calls and their return values. The calls include "execve", "brk", "access", "mmap2", "open", "fstat64", and "close". The output shows that the "ls" command is being executed on the file "/etc/ld.so.nohwcap", and that the file is not found (ENOENT). The terminal also shows the execution of the "ls" command on the file "/lib/tls/i686/cmov/librt.so.1", which is found. The desktop background is a dark, abstract image. The taskbar at the bottom of the screen shows several icons, including a folder, a document, and a network icon. The system clock in the bottom right corner shows "15:55" and "2009-01-12".

strace -wc ls

Microsecondo: 10^{-6}

Usecs/call: durata
in microsecondi della
singola call (con
arrotondamento)

%time: percentuale
di tempo usato
all'interno dell'esecuzione tracciata

Calls: numero di
invocazioni della
stessa system call

Errors: numero
di eventuali errori

syscall: system call

% time	seconds	usecs/call	calls	errors	syscall
22.96	0.000214	8	27		write
13.52	0.000126	7	19		mmap
12.02	0.000112	112	1		execve
8.80	0.000082	7	12		mprotect
7.83	0.000073	8	9		open
6.33	0.000059	5	11		close
5.36	0.000050	5	10		fstat
4.94	0.000046	7	7		read
4.61	0.000043	22	2		getdents
4.29	0.000040	6	7	7	access
1.93	0.000018	9	2	2	statfs
1.72	0.000016	5	3		brk
1.18	0.000011	11	1		munmap
1.07	0.000010	5	2		ioctl
0.86	0.000008	4	2		rt_sigaction
0.54	0.000005	5	1		rt_sigprocmask
0.54	0.000005	5	1		getrlimit
0.54	0.000005	5	1		arch_prctl
0.54	0.000005	5	1		set_robust_list
0.43	0.000004	4	1		set_tid_address
100.00	0.000932		120	9 total	

echo

```
baroglio@esmeralda: ~/BACKUP/LAB_S0/Slide$ echo "ciao"  
ciao  
baroglio@esmeralda: ~/BACKUP/LAB_S0/Slide$ █
```

echo visualizza a terminale delle scritte

strace echo

```
baroglio@esmeralda: ~/BACKUP/LAB_S0/Slide - Shell No. 5 - Kpmple
Session Edit View Bookmarks Settings Help
Shell No. 3 Shell Shell No. 2 Shell No. 4 Shell No. 5
baroglio@esmeralda: ~/BACKUP/LAB_S0/Slide$ echo "ciao"
ciao
baroglio@esmeralda: ~/BACKUP/LAB_S0/Slide$ strace echo "ciao"
execve("/bin/echo", ["echo", "ciao"], [/ * 32 vars */]) = 0
brk(0) = 0x804d000
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7fa1000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=67913, ...}) = 0
mmap2(NULL, 67913, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7f90000
close(3) = 0
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
open("/lib/tls/i686/cmov/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\260a\1"... , 512) = 512
fstat64(3, {st_mode=S_IFREG|0644, st_size=1339816, ...}) = 0
mmap2(NULL, 1349136, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb7e46000
mmap2(0xb7f8a000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x143) = 0xb7f8a000
mmap2(0xb7f8d000, 9744, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7f8d000
close(3) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7e45000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb7e456b0, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb7f8a000, 4096, PROT_READ) = 0
munmap(0xb7f90000, 67913) = 0
brk(0) = 0x804d000
brk(0x806e000) = 0x806e000
open("/usr/lib/locale/locale-archive", O_RDONLY|O_LARGEFILE) = -1 ENOENT (No such file or directory)
open("/usr/share/locale/locale.alias", O_RDONLY) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=2586, ...}) = 0
mmap2(
read(
```

strace -wc echo "ciao"

% time	seconds	usecs/call	calls	errors	syscall
31.20	0.000161	161	1		execve
17.05	0.000088	10	9		mmap
8.53	0.000044	11	4		mprotect
6.98	0.000036	36	1		write
6.98	0.000036	12	3		open
6.78	0.000035	7	5		close
6.01	0.000031	8	4		fstat
5.43	0.000028	9	3	3	access
4.84	0.000025	8	3		brk
3.10	0.000016	16	1		munmap
1.74	0.000009	9	1		read
1.36	0.000007	7	1		arch_prctl
100.00	0.000516		36	3 total	

strace -wc cp file1 file2

% time	seconds	usecs/call	calls	errors	syscall
51.75	0.002311	2311	1		execve
8.96	0.000400	16	25		mmap
6.85	0.000306	26	12		open
5.89	0.000263	16	16		mprotect
4.70	0.000210	14	15		close
4.50	0.000201	18	11		read
3.49	0.000156	13	12		fstat
3.31	0.000148	16	9	9	access
2.17	0.000097	32	3		brk
2.08	0.000093	93	1		write
1.50	0.000067	22	3	2	stat
1.23	0.000055	28	2		munmap
0.85	0.000038	19	2	2	statfs
0.52	0.000023	12	2		rt_sigaction
0.38	0.000017	17	1	1	lseek
0.34	0.000015	15	1		geteuid
0.27	0.000012	12	1		rt_sigprocmask
0.25	0.000011	11	1		getrlimit
0.25	0.000011	11	1		arch_prctl
0.25	0.000011	11	1		set_tid_address
0.25	0.000011	11	1		set_robust_list
0.22	0.000010	10	1		fadvise64
100.00	0.004466		122	14 total	

Linguaggi di programmazione e system call

I linguaggi di programmazione “nascondono” le system call all'interno di librerie, cioè collezioni strutturate di funzioni di supporto. I programmatori conoscono le funzioni a disposizione dalle descrizioni contenute in apposite interfacce dette API (application programming interface).

Fra le API più note:

POSIX (per Linux, Unix, Mac OS)

Win 32 (Windows)

API Java (per applicazioni eseguite da Java Virtual Machine)

Processi, RAM e file system

Processi e ram

- Abbiamo visto che di ogni processo viene mantenuta un'astrazione
- **Dove?** In RAM
- Cosa significa che un processo è in RAM?

Processi e ram

- Abbiamo visto che di ogni processo viene mantenuta un'astrazione
- **Dove?** In RAM
- Cosa significa che un processo è in RAM? Il codice che esegue, il suo stack, in suo heap sono contenuti in RAM
- **Alcuni problemi di rappresentazione:**
 - occorre associare porzioni di RAM ai vari processi
 - occorre un modo per sapere se e quanta RAM è libera qualora venga generato un nuovo processo

Gestione della memoria principale e secondaria

La memoria principale è limitata

il SO è responsabile delle seguenti attività:


- mantenere l'informazione relativa a chi sta usando quale parte di memoria e quale parte è invece libera
- assegnare/revocare lo spazio a seconda delle necessità
- decidere quali (parti di) processi vanno trasferiti in memoria centrale e quali vanno rimossi

il SO gestisce anche la memoria secondaria (hard disk)

- assegnazione dello spazio / scheduling del disco
- quando la quantità di memoria richiesta dai processi è troppo elevata non è possibile mantenerli tutti in RAM, occorre tenerne alcuni in memoria secondaria

Memoria e file system

- Gli utenti vedono la memoria organizzata in file
- **file**: unità logica di archiviazione, di norma organizzati in una struttura a **directory**
- Gli utenti usano nomi e cammini, il SO deve saper individuare i blocchi di memoria corrispondenti: per ogni file è necessario mantenere diverse proprietà, es. limitazioni all'accesso da parte di utenti diversi, tipo del file, data di creazione, ...



sistemi operativi e utenti

capitolo 2 del libro (VII ed.)

SO come ambiente di lavoro

Classi di servizi offerti 1/2

Interfaccia utente (user interface)

- **command-line**: comandi espressi come stringhe di caratteri
- **interfacce grafiche (GUI)**: sistema grafico a finestre con puntatore e menu

Esecuzione di programmi

- deve essere possibile far eseguire programmi
- i programmi possono richiedere l'accesso a file e adispositivi di I/O (es. monitor)
- deve essere possibile rilevare lo stato di terminazione (corretta o errata) di un programma

SO come ambiente di lavoro

Classi di servizi offerti 2/2

Comunicazione fra processi

- modello a memoria condivisa: i processi scrivono/leggono dati in/da un'area comune che deve essere gestita in modo da evitare inconsistenze
- modello a scambio di messaggi: il SO gestisce lo scambio di pacchetti di informazione fra i diversi processi

Protezione

- gli utenti possono desiderare di limitare l'accesso all'informazione di loro proprietà
- non tutti possono eseguire tutti i comandi, tutti gli applicativi, leggere/modificare qualsiasi file

Interfacce utente

- In molti SO (es. Unix, Windows XP) l'interfaccia utente è un **programma particolare** che viene avviato all'atto del login (autenticazione ed accesso alla macchina)
- **Command-line interface** o interprete di comandi o **shell**: esegue un ciclo infinito in cui attende un comando, lo esegue, torna in attesa. I comandi possono essere codificati all'interno dell'interprete stesso oppure possono corrispondere a nomi di programmi che vengono identificati dall'interprete, caricati in memoria e quindi eseguiti
- **Graphical User Interface** (GUI): vige la metafora del tavolo da lavoro (desktop), su cui sono posti oggetti selezionabili ed attivabili tramite puntatore



struttura di un sistema operativo

capitolo 2 del libro (VII ed.), da 2.6 in
poi

Come si implementa un SO?

- Originariamente scritti in linguaggio assembly di specifici processori
 - Dipendenza dall'hardware → poca portabilità
 - Esempio
MS-DOS era scritto nel linguaggio assembly del processore Intel 8088, quindi poteva essere usato solo su processori Intel
- Attualmente SO sono scritti in linguaggi di programmazione di alto livello
 - es. Unix, Linux e Windows XP sono scritti in C
- CPU attuali troppo complesse per riuscire a programmarle ancora in assembly direttamente

SCELTA DI UN LINGUAGGIO CHE SUPPORTI LA PORTABILITÀ

Passo 1: definire i criteri

- Per quale scopo e per quale tipo di uso e di utenza verrà progettato il SO?
- Messi in luce questi aspetti si passa alla **definizione dei criteri più opportuni**, cioè tutte le politiche di gestione di tutte le risorse

Criteri e meccanismi

- **criterio**: specifica un comportamento, **cosa** fare in una certa circostanza (**es. alle 20:00 accendi il riscaldamento**). Sono anche detti politiche.
- **meccanismo**: strumenti, anche software, neutri rispetto ai criteri (**es. istruzioni per avviare il riscaldamento**)
- è possibile implementare criteri diversi con gli stessi meccanismi, per esempio posso adoperare le stesse istruzioni di accensione del riscaldamento per definire le politiche:
 - alle 6:30 accendi il riscaldamento
 - quando la temperatura scende a 16.5 gradi accendi il riscaldamento

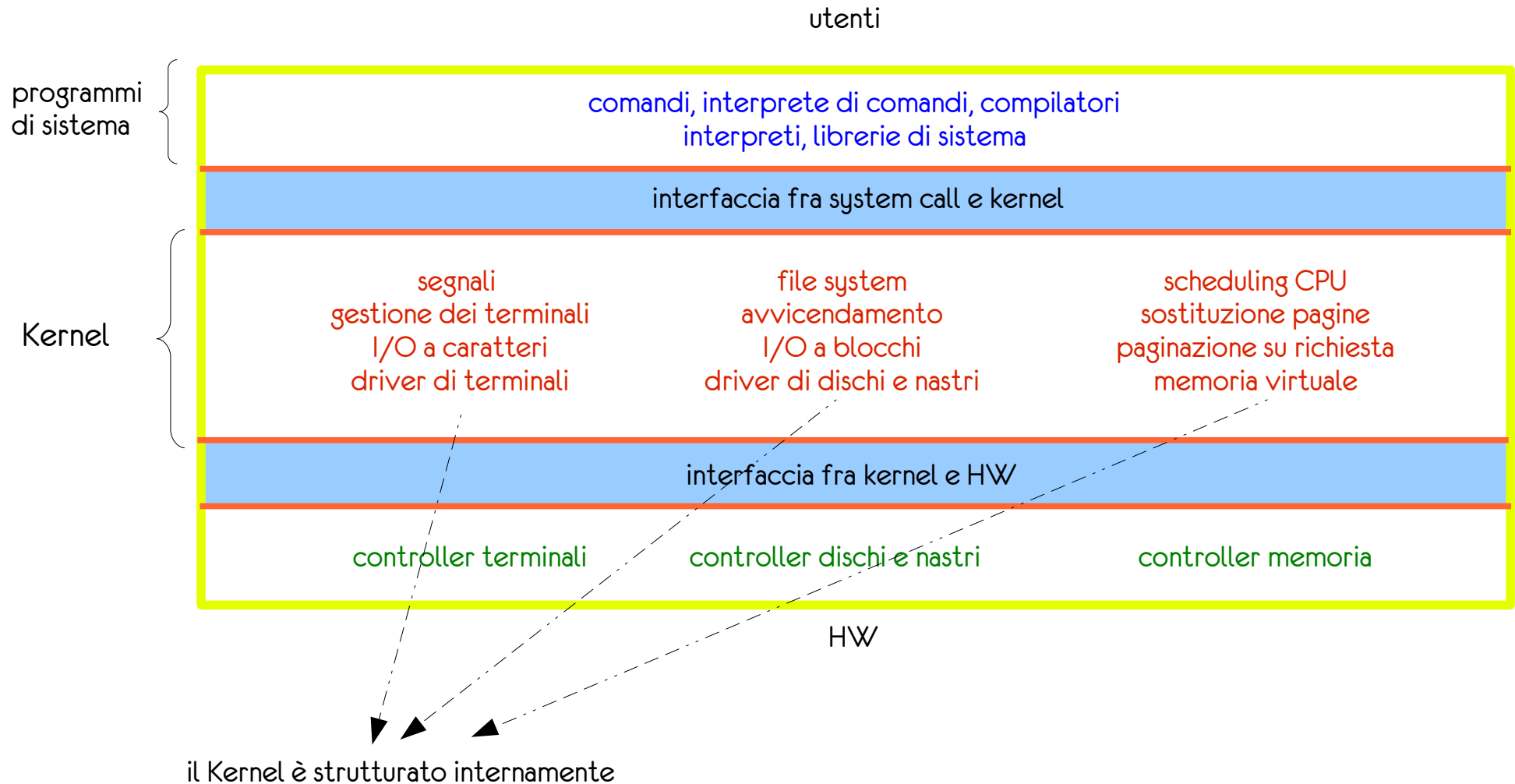
Struttura

- Come ogni programma complesso, **un SO deve essere ben strutturato** per poter essere mantenuto e aggiornato
- **Alcuni noti SO nati come sistemi piccoli e limitati e poi accresciutisi al di là delle previsioni, non avevano una chiara struttura in moduli, con una chiara definizione delle interfacce**
- **Alcuni limiti derivavano dall'HW** in uso. Per esempio l'Intel 8088 non prevedeva dual mode: gli applicativi potevano accedere direttamente all'I/O scrivendo direttamente su video e dischi ... per questo MS-DOS, scritto per l'8088, era **vulnerabile**

Tecniche di modularizzazione

- NB: la modularizzazione è possibile se e solo se supportata dall'hardware
- Tecniche:
 - stratificazione
 - microkernel
 - moduli

Struttura del primo Unix

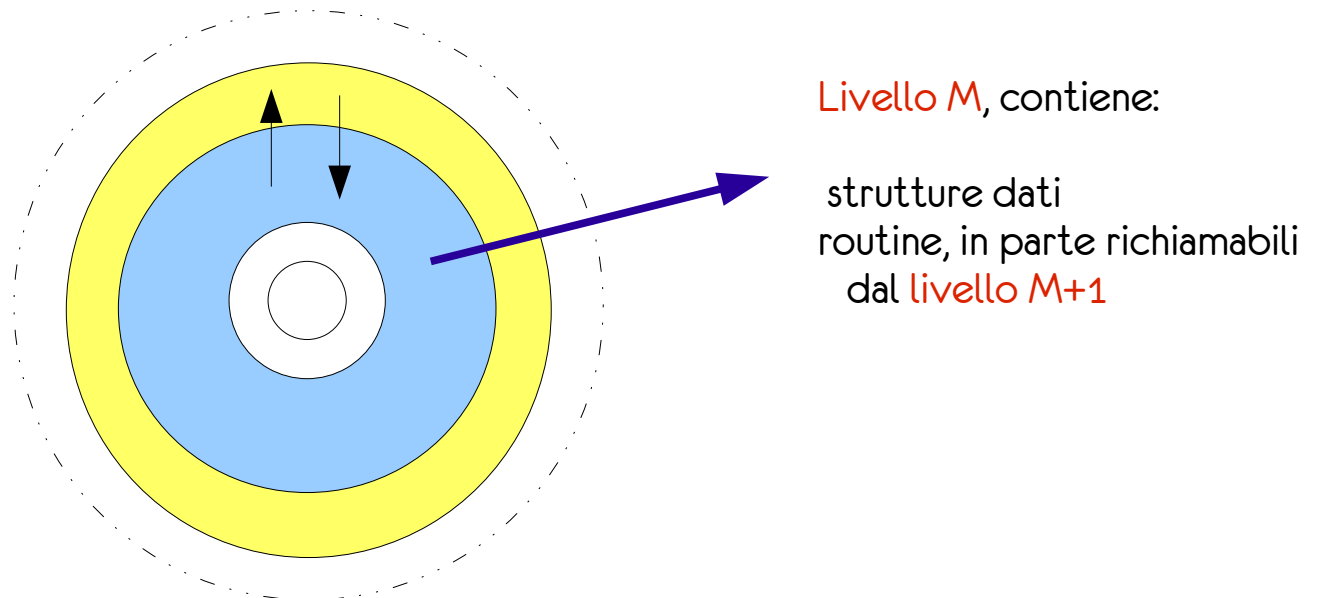


Stratificazione

Tecnica di stratificazione

Il sistema ha una struttura “a cipolla”, lo strato più interno (o strato 0) corrisponde all'HW, quello più esterno all'interfaccia utente

ogni strato realizza un **oggetto astratto, che incapsula dei dati e le operazioni che consentono di elaborare quei dati**, parte delle quali sono rese visibili/accessibili dall'esterno (appartengono all'interfaccia dello strato)



Tecnica di stratificazione

Vantaggi e svantaggi

semplicità di progettazione: per realizzare uno strato basta sapere quali funzionalità ha a disposizione, non importa sapere come sono realizzate

semplicità di debug e verifica del sistema: ogni strato usa solo funzionalità messe a disposizione dallo strato immediatamente inferiore, possiamo verificare gli strati uno per volta

difficoltà: definire in modo opportuno gli strati, quale funzionalità sta in quale strato?

minore efficienza in fase di esecuzione, ogni passaggio di stato comporta infatti la chiamata di una nuova funzione, da caricare, che può richiedere parametri, da caricare a loro volta ...

Stratificazione e macchine virtuali

- L'approccio a strati trova il suo vertice massimo nel concetto di **macchina virtuale**: uno strato software che duplica il comportamento di ogni componente hardware del computer
- A che pro? Alcuni scenari:
 - uso di SO diversi su uno stesso computer
 - verifica e debugging di applicativi su SO diversi
 - uso di SO diversi da parte di utenti diversi
 - ...
- Invece di installare un SO direttamente, lo si installa su di una macchina virtuale, che a sua volta lavora su un computer che ha già un proprio SO
- I SO installati sulla macchina virtuale sono detti “ospiti”

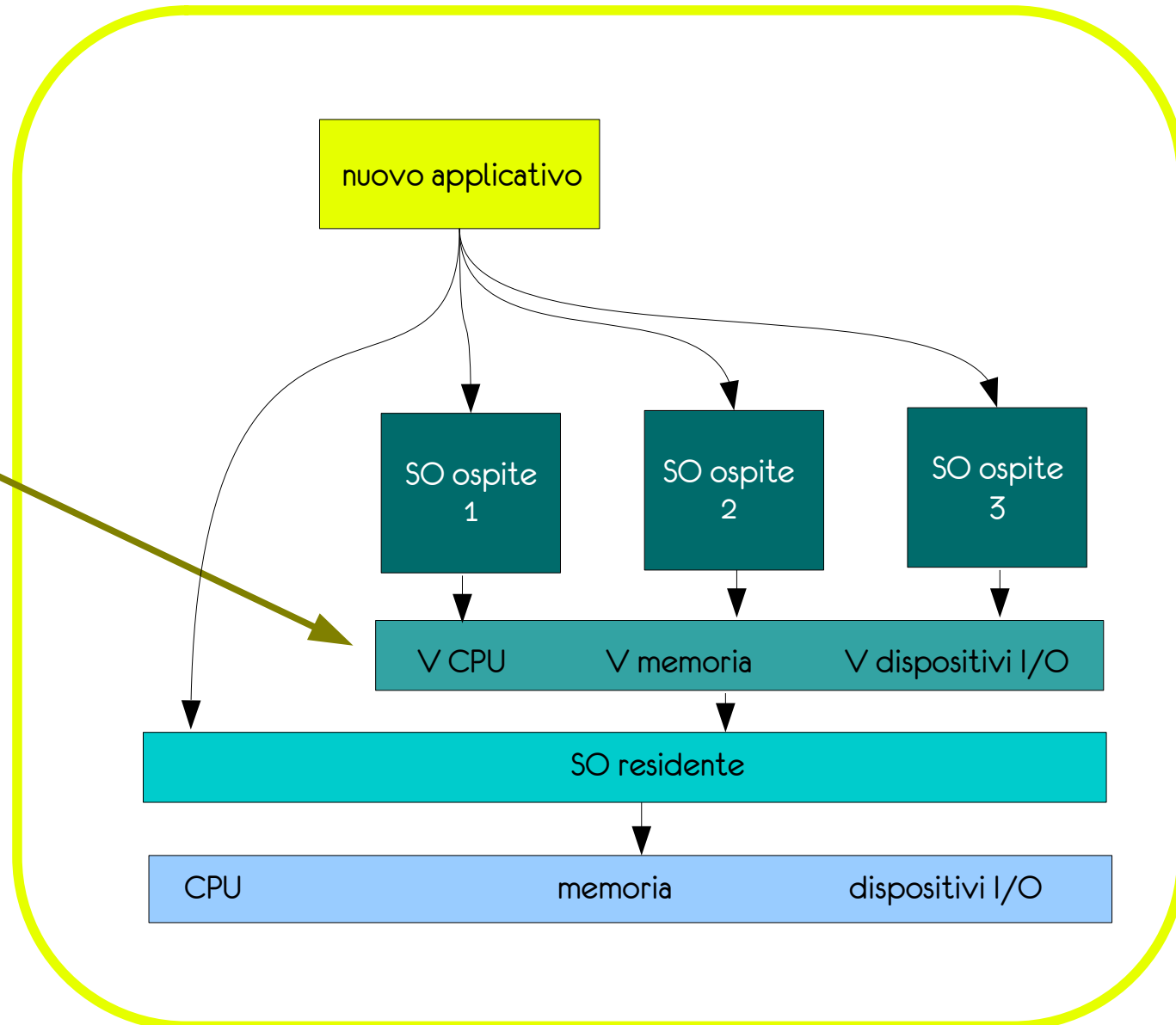
macchine virtuali - Esempi 2/2

Java Virtual Machine

CLR di .Net

Vmware

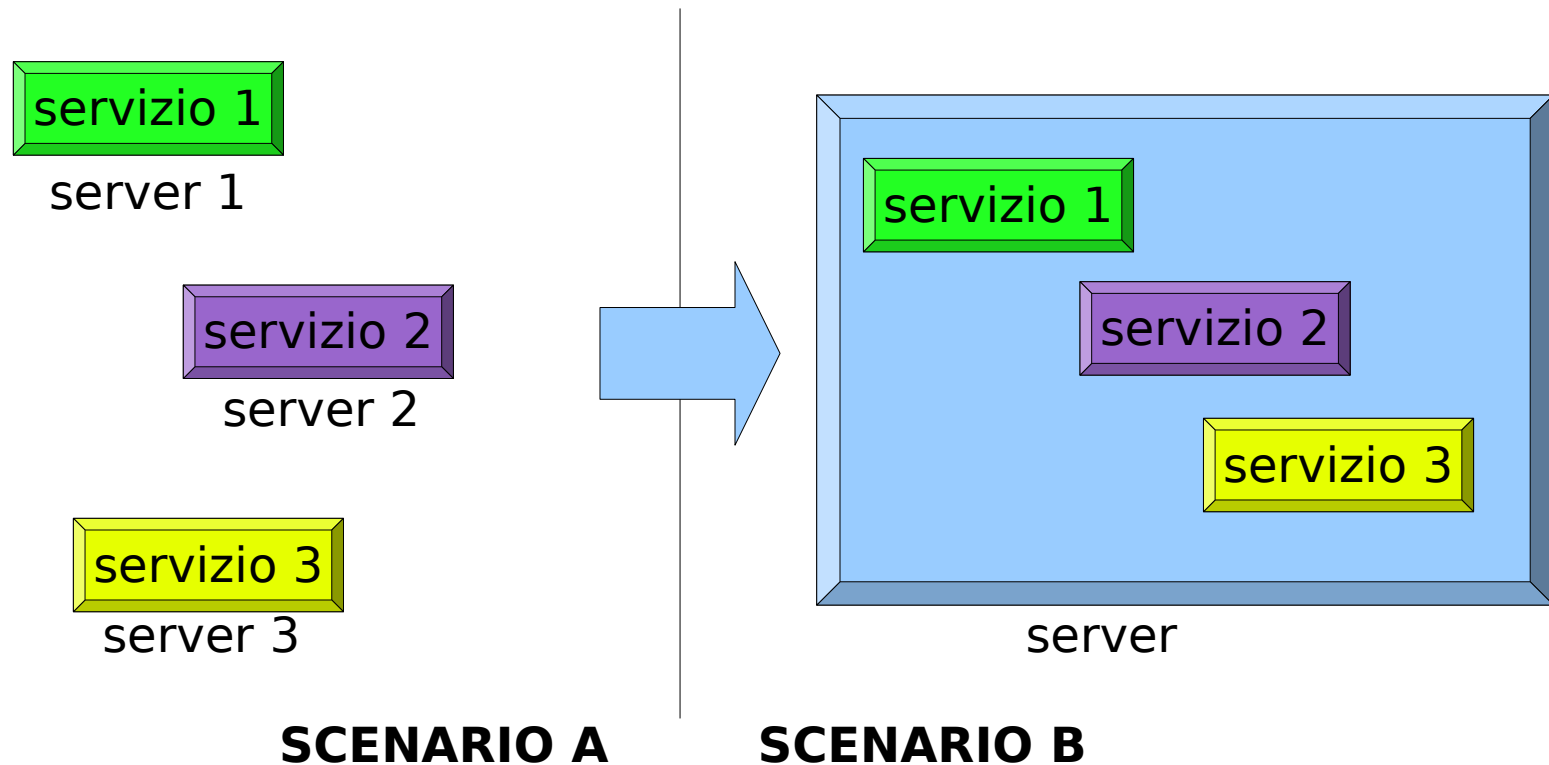
Hypervisor:
elemento di una
macchina virtuale che
fornisce un'astrazione
delle diverse risorse
hardware



Virtualizzazione delle piattaforme

tema attuale, non strettamente legato ai SO ma più all'erogazione di servizi

es. Server Consolidation



Pro/contro virtualizzazione

La riduzione del numero dei server comporta:

- riduzione di occupazione di spazio
- riduzione dei contratti di manutenzione
- riduzione del consumo energetico (funzionamento e refrigerazione)
- se permangono più server posso attuare politiche di bilanciamento del carico

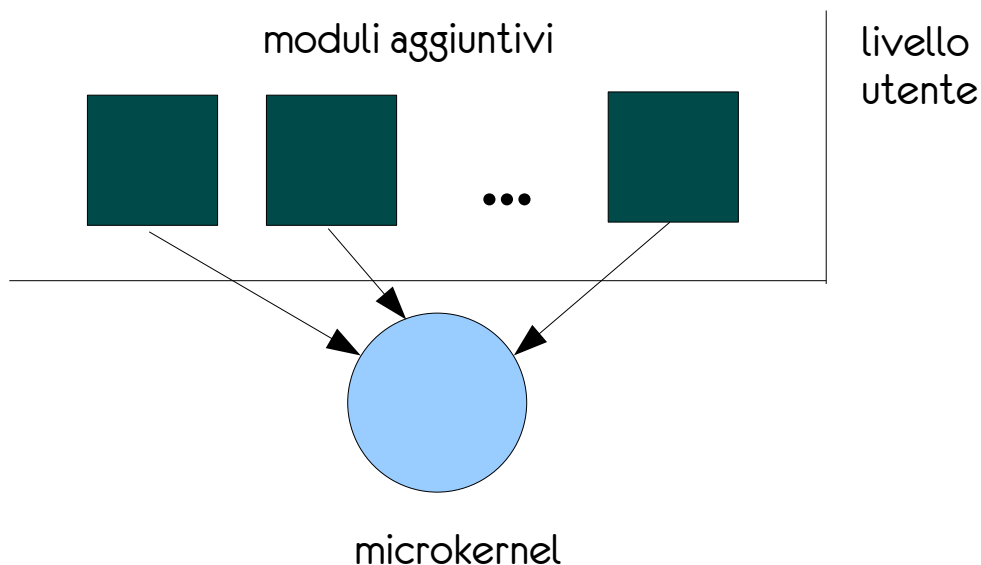
Ridurre il numero dei server ha dei vantaggi ma ridurre eccessivamente il numero dei server comporta una riduzione delle prestazioni

Tecnica a microkernel

Nata verso la metà degli anni '80 con la realizzazione del SO Mach, presso la Canegie Mellon University

L'idea è rimuovere dal kernel tutto ciò che non è essenziale, **spostandolo a livello di applicativo utente.**

I microkernel in genere contengono servizi minimi per la gestione di **processi**, **comunicazione** (a scambio di messaggi) e **memoria**



I moduli aggiuntivi (es. Il file server) forniscono servizi secondo un modello client-server

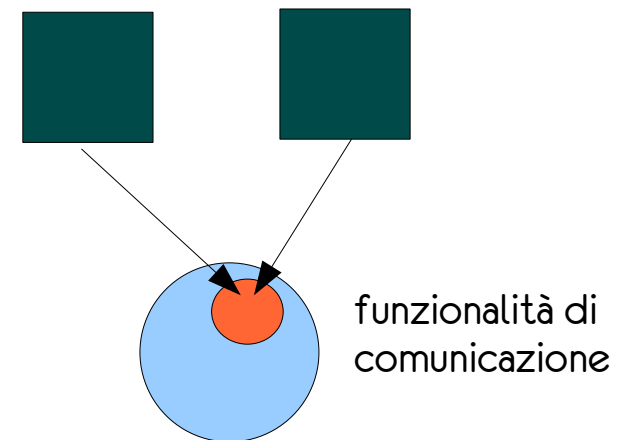
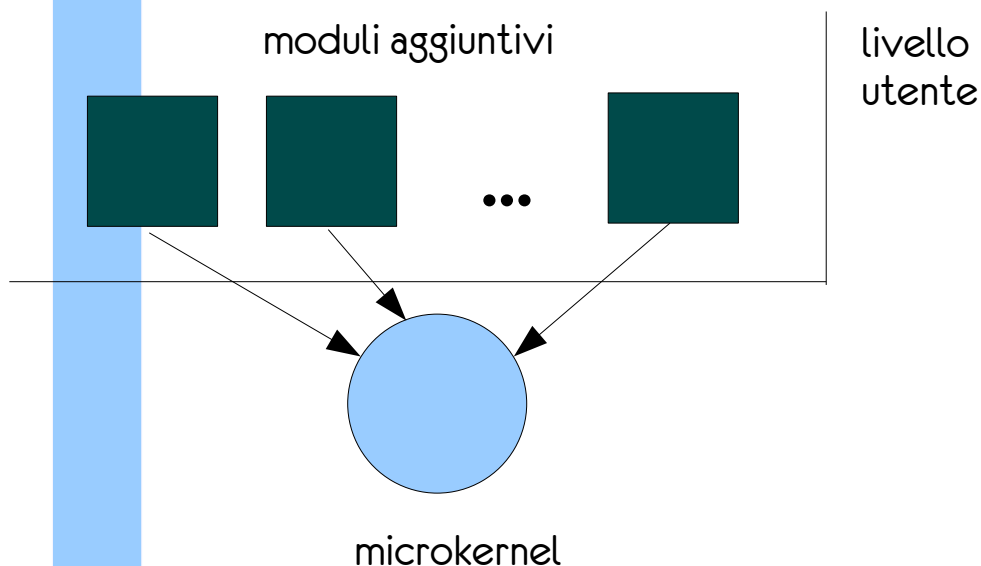
La comunicazione avviene tramite scambio di messaggi

Tecnica a microkernel

Nata verso la metà degli anni '80 con la realizzazione del SO Mach, presso la Canegie Mellon University

L'idea è rimuovere dal kernel tutto ciò che non è essenziale, **spostandolo a livello di applicativo utente.**

I microkernel in genere contengono servizi minimi per la gestione di **processi, comunicazione** (a scambio di messaggi) e **memoria**



Inefficienza prestazionale:
lo scambio di messaggi (richiesta/risposta)
appesantisce l'esecuzione col crescere del
numero dei processi

Tecnica a microkernel

vantaggi:

- **facilità di estensione**: i nuovi servizi vengono aggiunti senza modificare il kernel
- il SO risulta più **semplice da adattare** a nuove architetture
- **maggiore sicurezza**: un servizio compromesso non incide sull'affidabilità del kernel

svantaggi:

- **sovraccarichi** dati dall'esecuzione di processi utente con funzionalità di SO
- in particolare la comunicazione indiretta è un collo di bottiglia

Tecnica a moduli

- **Approccio attualmente considerato il migliore**, adottato da SO come Solaris, Linux, Mac OS
- **A un kernel minimale possono essere aggiunti moduli nuovi, in fase di avvio ma soprattutto anche di esecuzione:**
 - aggiunta moduli significa aggiunta di system call
 - significa anche aggiunta della capacità di gestire nuovo hardware (per esempio driver per dispositivi specifici)
- Come nei sistemi a microkernel, il kernel gestisce un nucleo ridotto all'osso di funzionalità essenziali, che però possono essere modificate run-time a seconda delle esigenze (per es. per gestire nuovi device o qualche protocollo di networking)
- Ogni modulo ha una propria interfaccia ben definita (come nell'approccio stratificato)

Tecnica a moduli

Differenze e vantaggi:

- a differenza dai sistemi stratificati, ogni modulo può usarne qualsiasi altro (flessibilità)
- a differenza dai microkernel, la comunicazione fra moduli è diretta (efficienza)

lsmod

per visualizzare I moduli caricati si può utilizzare il comando lsmod:

LSMOD(8)

lsmod

LSMOD(8)

NAME

lsmod - Show the status of modules in the Linux Kernel

SYNOPSIS

lsmod

DESCRIPTION

lsmod is a trivial program which nicely formats the contents of the /proc/modules, showing what kernel modules are currently loaded.

COPYRIGHT

This manual page originally Copyright 2002, Rusty Russell, IBM Corporation. Maintained by Jon Masters and others.

SEE ALSO

insmod(8), modprobe(8), modinfo(8)