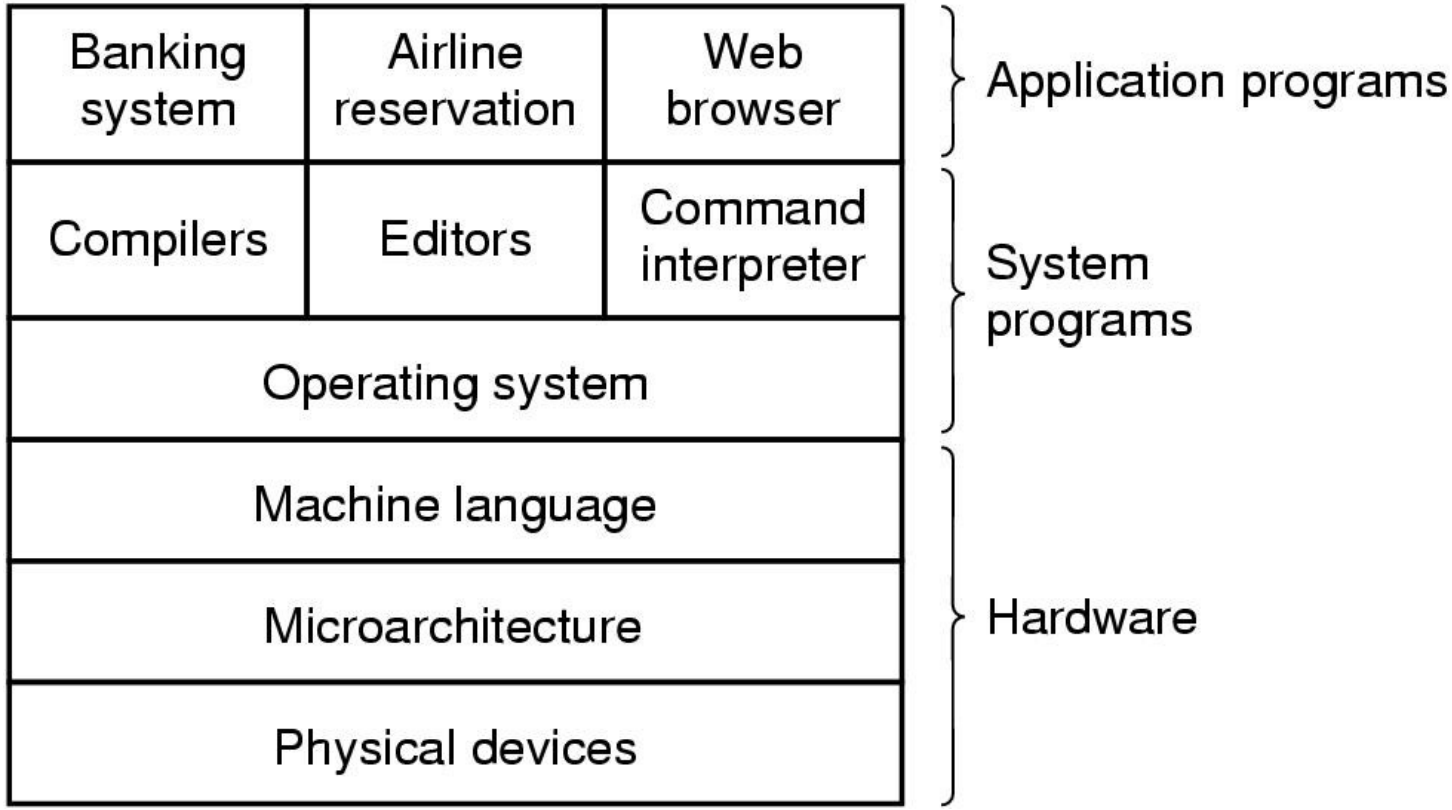
Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative
Istituto sistemi informativi e networking

SUPSI

Ambienti Operativi: Struttura dei sistemi operativi

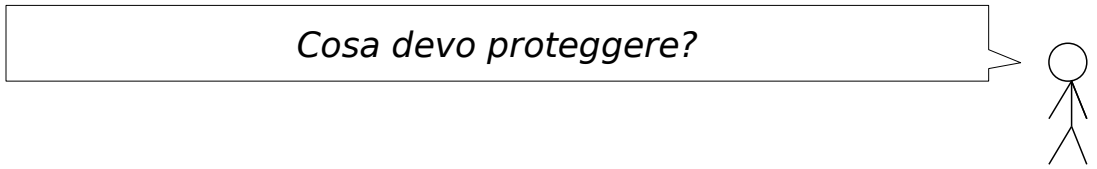
Amos Brocco, Ricercatore, DTI / ISIN

Organizzazione di un sistema operativo



Da A. Tanenbaum, "Modern Operating Systems", 2ª Edizione

Protezione



- **CPU**
  - Controllo del tempo CPU utilizzato dalle applicazioni
  - Evitare che un'applicazione prenda troppo tempo
- **Memoria**
  - Evitare che un processo possa “curiosare” nella memoria di un altro processo
  - Evitare che si possa modificare la memoria che non ci appartiene
- **Accesso alle periferiche**
  - Prevenire l'utilizzo di istruzioni I/O non valide

Parzialmente risolto con l'utilizzo di indirizzi virtuali e il meccanismo di traduzione degli indirizzi

Come proteggere?

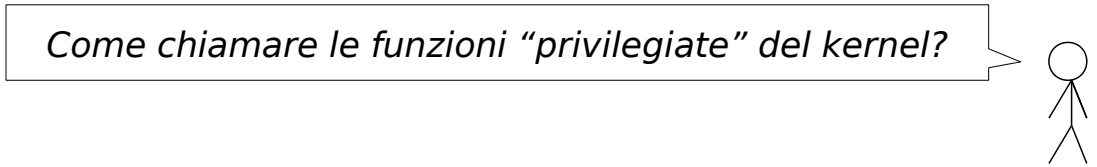
- Permettiamo l'accesso diretto alle periferiche e alla memoria tramite indirizzi fisici, e l'esecuzione di certe istruzioni solo a un'entità di cui ci possiamo fidare:

  il kernel

Il kernel: “un processo come un'altro”

- Il kernel non è altro che un processo che si contente l'utilizzo della CPU insieme agli altri processi
- A differenza dei processi utente, il kernel ha un accesso privilegiato a certe istruzioni del processore che gli altri processi non possono avere
- Il kernel fa da mediatore offrendo dei servizi per
  - l'esecuzione e terminazione dei processi
  - l'accesso a file e directory
  - l'accesso alla rete
  - l'accesso alle periferiche

System call



- **Una chiamata di sistema (system call) permette a un'applicazione di richiedere un “servizio” offerto dal kernel (“mediatore”)**
  - Gestione dei processi
    - fork, exec, exit, wait, kill...
  - Gestione della memoria
    - brk,...
  - Accesso al filesystem
    - open, read, write, close
  - Accesso alle periferiche
  - Comunicazione

- Numero di system call dipende dal kernel:
  - Linux ~ 320
  - POSIX ~130

"Vedere" le chiamate di sistema: strace

- Il comando **strace** su Linux ci permette di vedere quali chiamate di sistema vengono utilizzate da un processo

[X] bash

```
utente@host:~/Documenti$ strace -o ls.trace.txt ls
```


L'opzione -o mi permette di salvare l'output di strace in un file

Variabili di ambiente

strace: esempio 'ls'

[X] bash

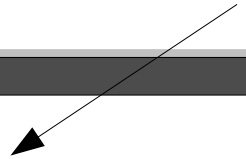
```
utente@host:~/Documenti$ strace -o ls.trace.txt ls
```

ls.trace.txt

```
execve("/bin/ls", ["ls"], [/* 52 vars */]) = 0
brk(0)                                  = 0x9c20000
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7893000
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=85514, ...}) = 0
mmap2(NULL, 85514, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb787e000
close(3)                                = 0
open("/lib/libselinux.so.1", 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\0p\343P\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=122436, ...}) = 0
...
open(".", O_RDONLY|O_NONBLOCK|O_LARGEFILE|O_DIRECTORY|O_CLOEXEC) = 3
fcntl64(3, F_GETFD)                     = 0x1 (flags FD_CLOEXEC)
getdents64(3, /* 140 entries */, 32768) = 4624
getdents64(3, /* 0 entries */, 32768)   = 0
close(3)                                = 0
...
write(1, "bbb.JpG  calendari-pap-tp.txt  D"..., 156) = 156
write(1, "bhello\t chello\t\t       dirmenu.s"..., 170) = 170
write(1, "bla\t comandi\t       dirmenu.sh~ "..., 140) = 140
write(1, "bla.sh\t curllogin.sh\t       Docu"..., 140) = 140
write(1, "booo\t curllogin.sh~\t       dodoc"..., 145) = 145
write(1, ",c\t dati.txt\t       dodocurl.py~"..., 136) = 136
close(1)                                = 0
```

strace: esempio 'bash'

Nella nuova shell



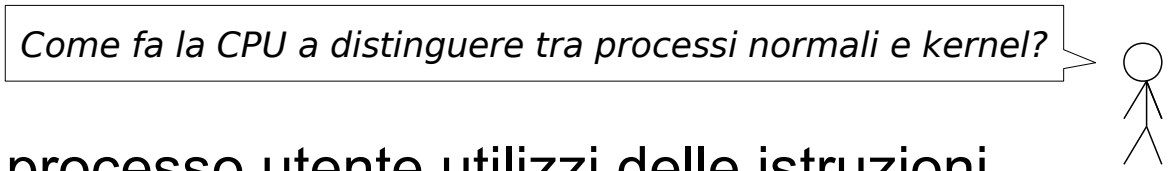
[X] bash

```
utente@host:~/Documenti$ strace -o bash.trace.txt bash
utente@host:~/Documenti$ echo "Hello world" 2> err.txt 1> out.txt
```

ls.trace.txt

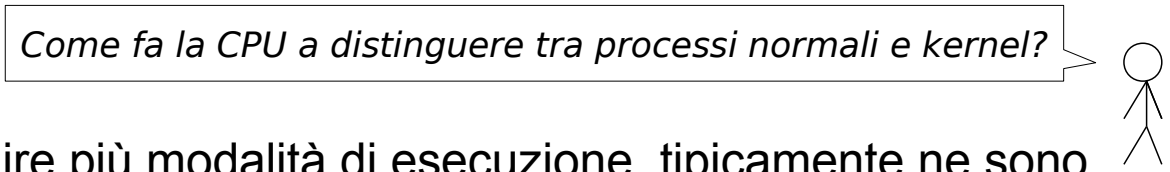
```
execve("/bin/bash", ["bash"], [/* 52 vars */]) = 0
...
open("err.txt", O_WRONLY|O_CREAT|O_TRUNC|O_LARGEFILE, 0666) = 3
...
dup2(3, 2)                              = 2
close(3)                                = 0
open("out.txt", O_WRONLY|O_CREAT|O_TRUNC|O_LARGEFILE, 0666) = 3
...
dup2(3, 1)                              = 1
close(3)                                = 0
write(1, "Hello world\n", 12)           = 12
...
```

Kernel mode / User mode



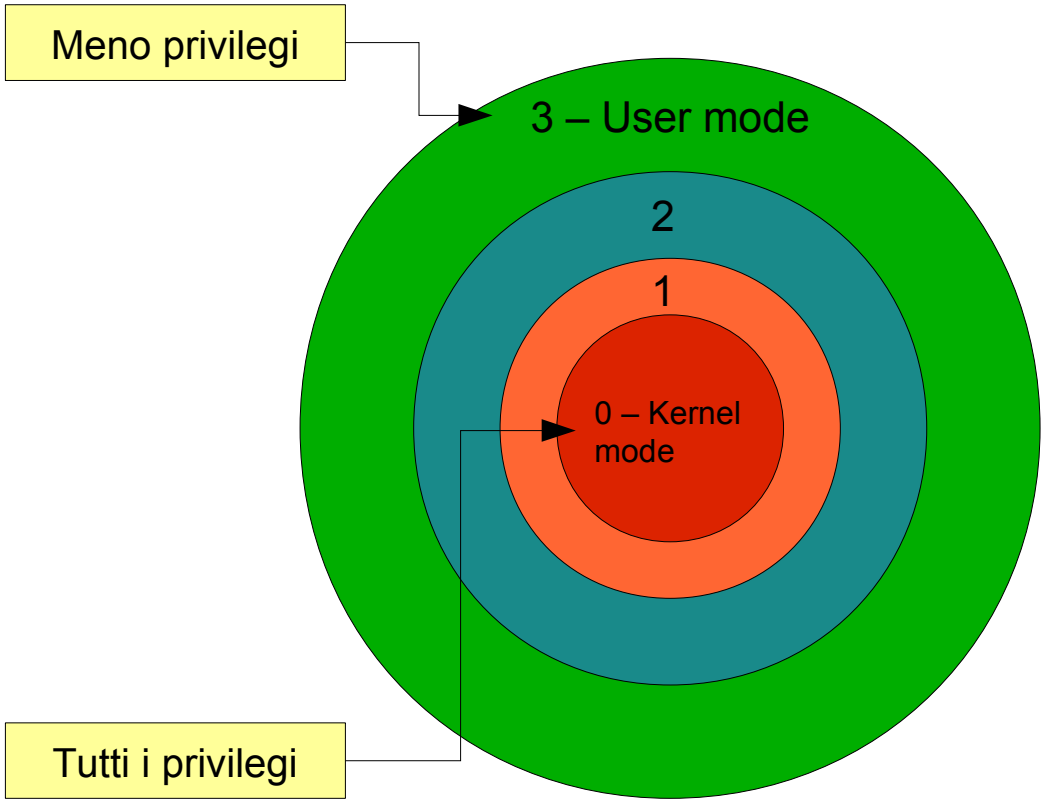
- Per impedire che un processo utente utilizzi delle istruzioni “privilegiate” per accedere alle periferiche o alla memoria dobbiamo disporre di un meccanismo per permettere alla CPU di distinguere tra processo kernel e non-kernel.

Kernel mode / User mode



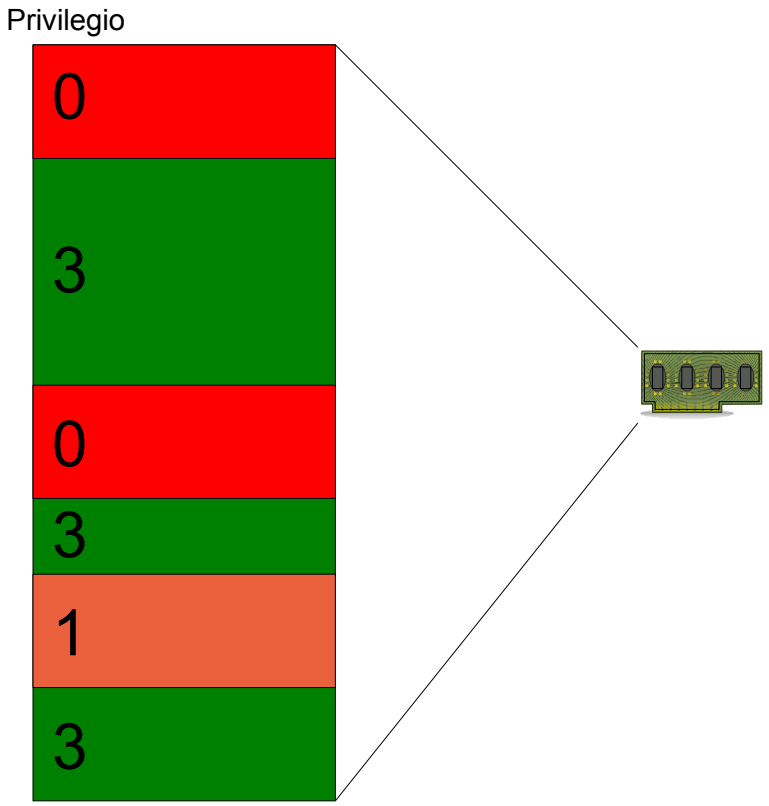
- La CPU permette di definire più modalità di esecuzione, tipicamente ne sono utilizzati due:
  - **Kernel mode** (modalità kernel): accesso completo all'hardware attraverso istruzioni privilegiate, gestione della memoria e degli interrupt
  - **User mode** (modalità utente): accesso limitato all'hardware, utilizzato dalle applicazioni

Schema a cipolla (anelli di protezione)



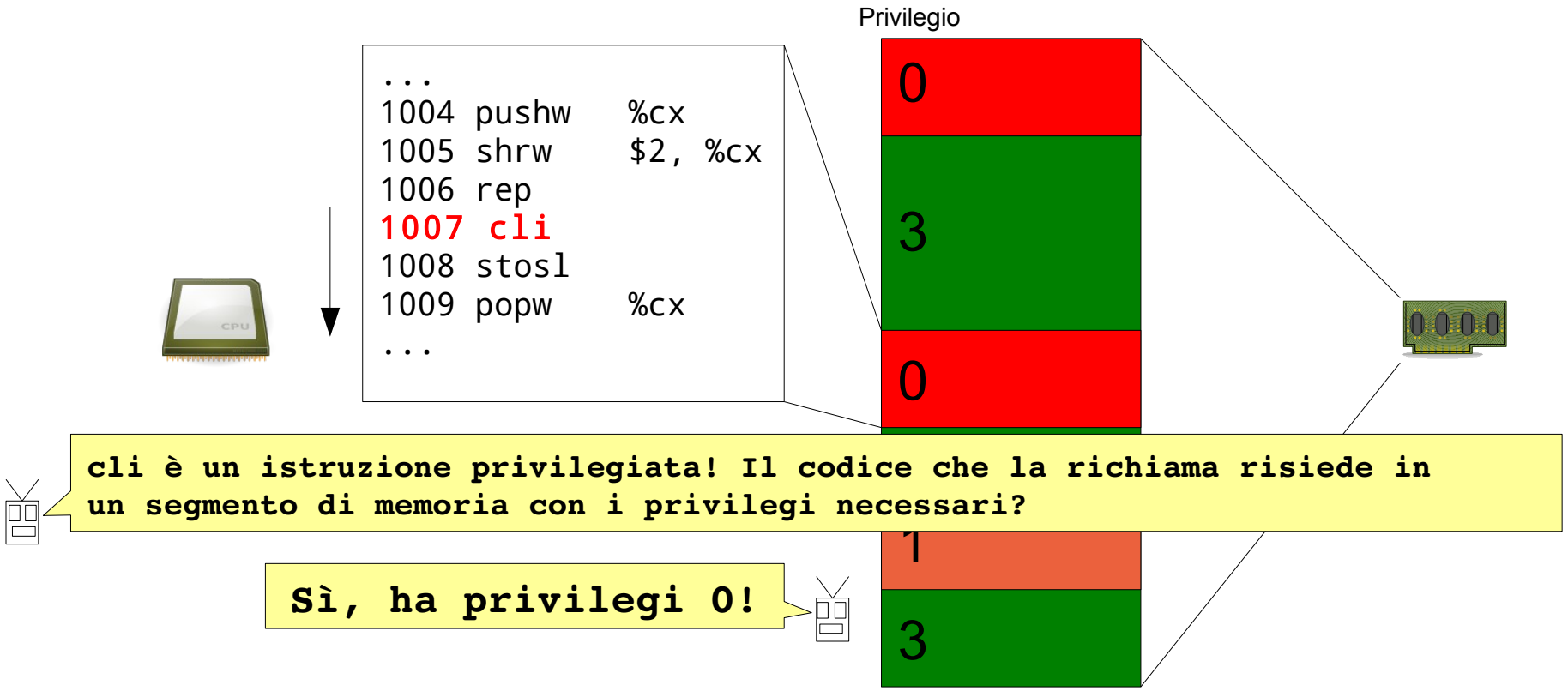
Protezione della memoria

- Abbiamo visto che la memoria è divisa in aree logiche, i segmenti
  - i segmenti possono contenere codice macchina o dati
  - possiamo definire il livello di privilegio di un processo assegnando un codice numerico ai segmenti di memoria utilizzate dal processo
  - solo il kernel può cambiare questi livelli



Protezione della memoria

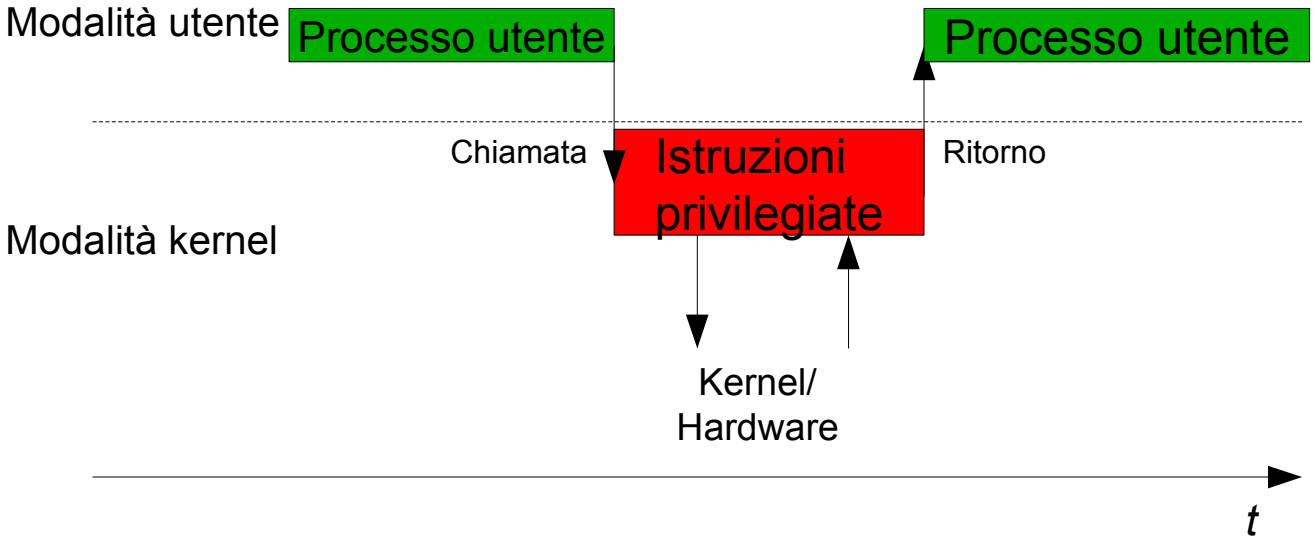
- Quando viene eseguita un'istruzione privilegiata il processore controlla che l'area di memoria abbia il codice dei privilegi necessario, altrimenti ho un errore di protezione



Spostarsi tra aree con privilegi diversi

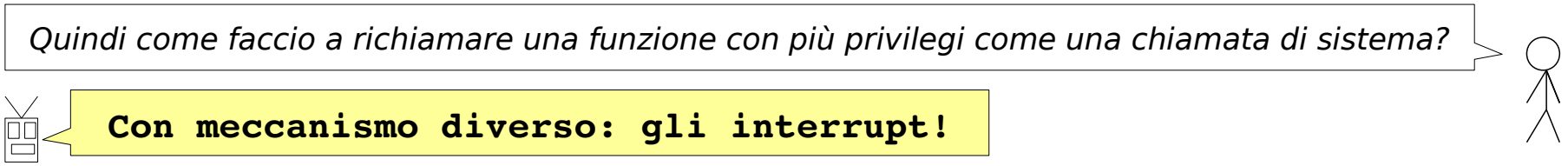
- Quando richiamo una funzione, effettuo un “salto” (jump) nel codice verso l'indirizzo di memoria che contiene la prima istruzione della funzione
- **Problema:** **Per ragioni di sicurezza** non è permesso “saltare” tra aree con privilegi diversi durante l'esecuzione per richiamare funzioni di sistema
  - Se fosse possibile saltare da una regione con privilegi più bassi a una con privilegi più alti il meccanismo di protezione della memoria non funzionerebbe
  - Allo stesso modo, se salto da una regione con privilegi più alti a una con privilegi più bassi, quando ritorno ho il problema inverso, ovvero passare da una regione con meno privilegi a una con più privilegi

Chiamata di sistema



Quindi come faccio per chiamare funzioni implementate dal kernel?

Interrupt



- Un **interrupt** (interruzione) permette di interrompere l'esecuzione di un processo, passare il controllo al sistema operativo, ed eseguire una routine di gestione predefinita
  - **Interrupt hardware (asincroni)**: quando una periferica vuole segnalare qualcosa al S.O
  - **Interrupt software (sincroni)**: quando un processo vuole segnalare qualcosa al S.O

Interrupt

- Quando viene eseguito un interrupt avviene un cambio di contesto (context switch):
  - Il controllo viene passato a un punto di accesso specifico del kernel (modalità privilegiata)
  - Il kernel esegue la funzione richiesta
  - Tramite un'istruzione posso ridare il controllo all'applicazione (modalità non-privilegiata)

- Visto che il punto di accesso è definito dal kernel, un processo utente non può eseguire istruzioni da un indirizzo arbitrario

/proc/interrupts

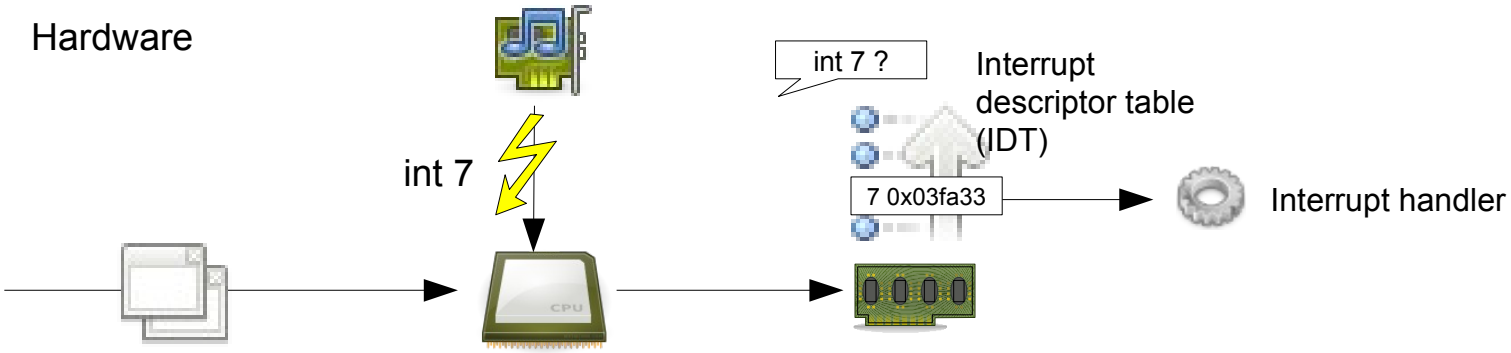
[X] bash

```
utente@host:~/Documenti$ cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:        131          0          0          0   IO-APIC-edge      timer
  1:         10          0          0          0   IO-APIC-edge      i8042
  4:          2          0          0          0   IO-APIC-edge
  7:          0          0          0          0   IO-APIC-edge      parport0
  8:          1          0          0          0   IO-APIC-edge      rtc0
  9:      37488      24161          0          0   IO-APIC-fasteoi   acpi
 12:        129          0          0          0   IO-APIC-edge      i8042
 16:     105002      17428          0          0   IO-APIC-fasteoi   ehci_hcd:usb1
 17:     418215      66538          0          0   IO-APIC-fasteoi   ehci_hcd:usb2, mmc0, hda_intel
 24:        300          0          0          0  HPET_MSI-edge      hpet2
 25:          0          0          0          0  HPET_MSI-edge      hpet3
 26:          0          0          0          0  HPET_MSI-edge      hpet4
 27:          0          0          0          0  HPET_MSI-edge      hpet5
 29:     108546         22    2012463   12211960   PCI-MSI-edge      i915
 30:      38254     930452          0          0   PCI-MSI-edge      ahci
 31:    1450962          0          0          0   PCI-MSI-edge      em1
 32:      31965       3310          0          0   PCI-MSI-edge      hda_intel
 33:    2587381          0          0          0   PCI-MSI-edge      iwlagn
NMI:          0          0          0          0   Non-maskable interrupts
LOC:    6945632    4580848    6065484    4045225   Local timer interrupts
SPU:          0          0          0          0   Spurious interrupts
PMI:          0          0          0          0   Performance monitoring interrupts
PND:          0          0          0          0   Performance pending work
RES:     156739      76288      84755     130210   Rescheduling interrupts
CAL:      61430      76493      63114      53259   Function call interrupts
```

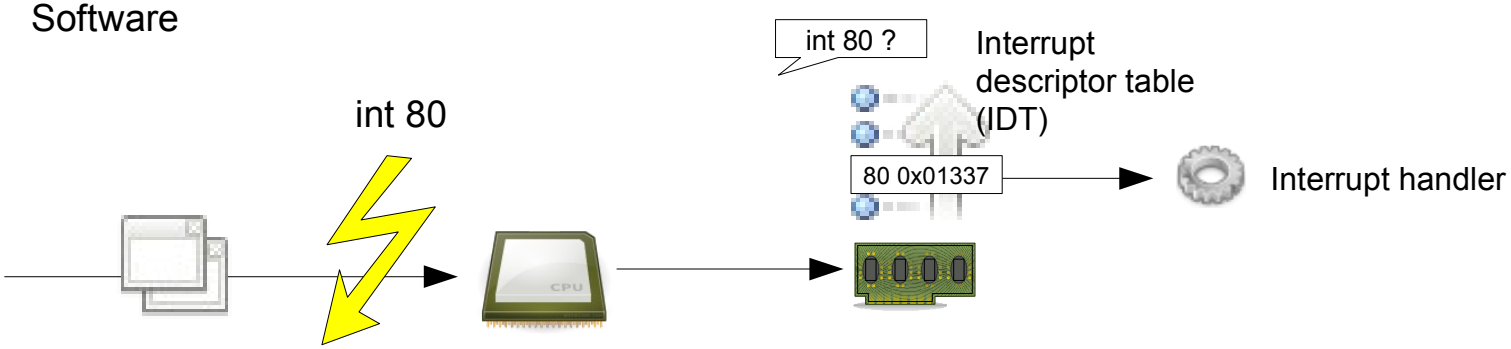
Interrupt software

Interrupt

- Hardware



- Software



System call con interrupt

- Il numero (identificatore) della funzione da chiamare è memorizzato in un registro (piccola memoria all'interno della CPU)
- I parametri possono essere memorizzati nei registri o sullo stack
  - Il kernel può accedere allo stack del processo utente
- Quando ritorno dalla system call il processo utente riprende la sua normale esecuzione

System call più veloci con SYSENTER/SYSEXIT

- Quando utilizzo un interrupt software (istruzione int) sono necessari **più accessi alla memoria** per ottenere l'indirizzo della routine di gestione degli interrupt (interrupt handler)

- **Soluzione più efficiente:**
  - A partire dal Pentium 2 è stato introdotto il “Fast System Call”
  - Idea: definiamo l'indirizzo dove si trova la routine per la gestione della chiamata in un registro che può essere modificato solo dal kernel
    - L'istruzione SYSENTER (SYSCALL su processori AMD) salta direttamente a questo indirizzo

Kernel mode / User mode

- Passare da modalità utente a modalità kernel è “più costoso” che chiamare una funzione nello stesso livello di protezione:
  - quindi perché non eseguire la maggior parte delle operazioni in modalità kernel?
    - sicurezza, robustezza
    - vale il “*principio del privilegio minimo*”
      - concedere ad ogni funzione del S.O. solo i privilegi di cui ha realmente bisogno

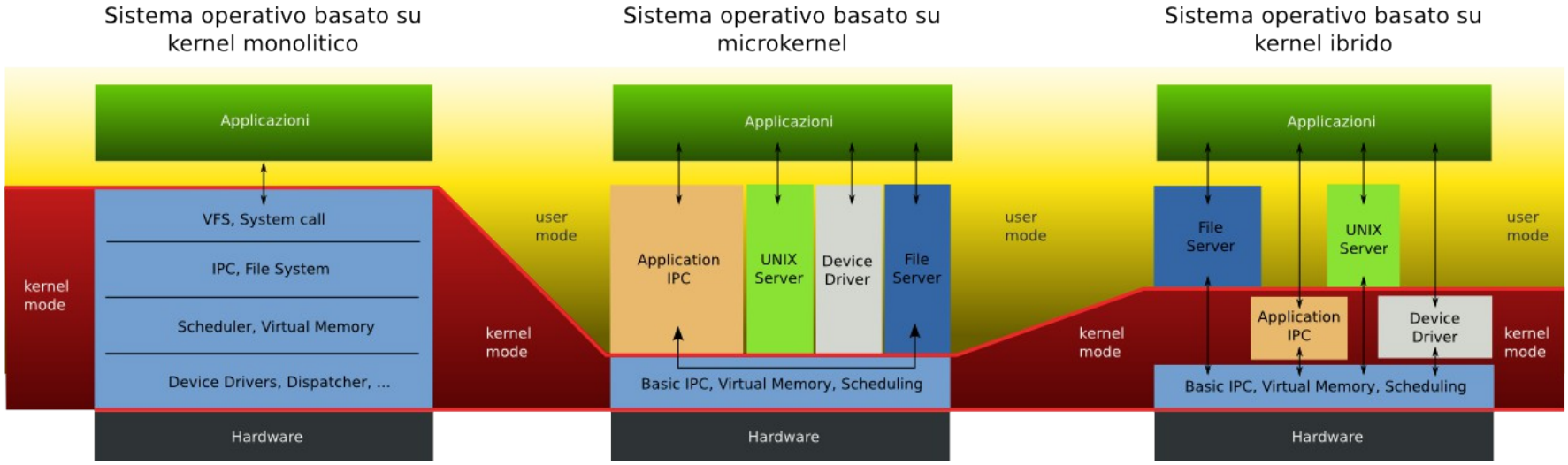
Struttura dei sistemi operativi

- Diverse tipologie di sistemi operativi possono definire diversamente quali funzionalità sono da eseguire in modalità “utente” e quali in modalità “kernel”
  - Non tutto *può* essere eseguito in modalità utente
  - Non tutto *deve* essere eseguito in modalità kernel

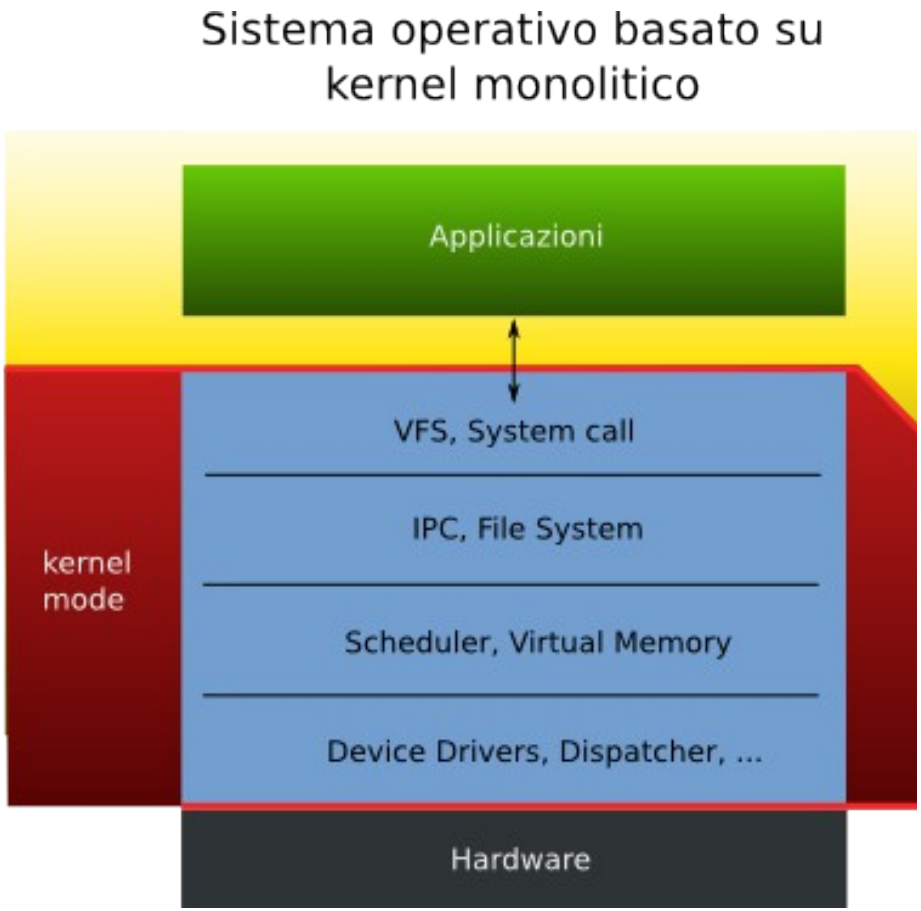
Tipologie di kernel

- **Sistemi micro-kernel**
  - Solo le funzionalità strettamente necessarie vengono eseguite in modalità kernel
  - Basato su server indipendenti eseguiti in modalità utente che comunicano tramite messaggi (più robusto)
- **Sistemi monolitici**
  - Tutte le funzionalità sono eseguite in modalità kernel
- **Sistemi ibridi**
  - Struttura modulare simile a un microkernel ma eseguita in modalità kernel

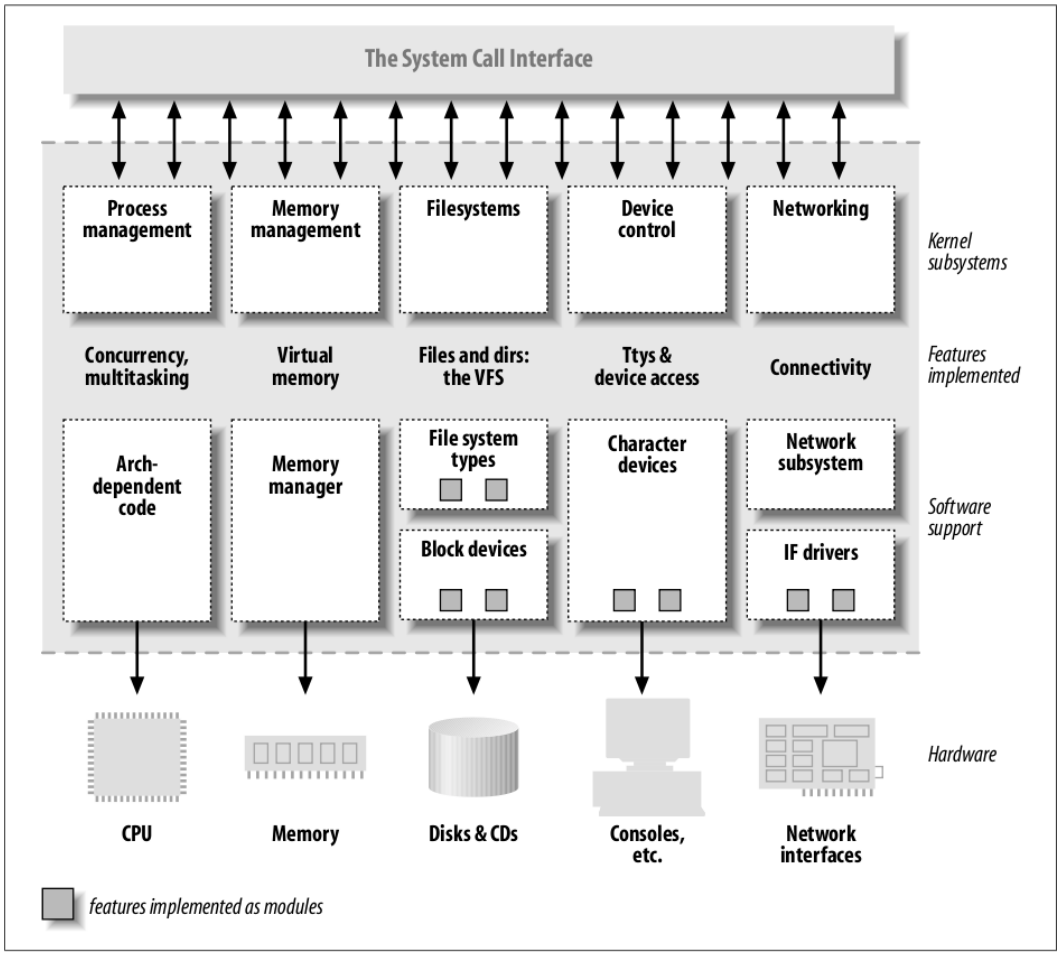
Tipologie di kernel



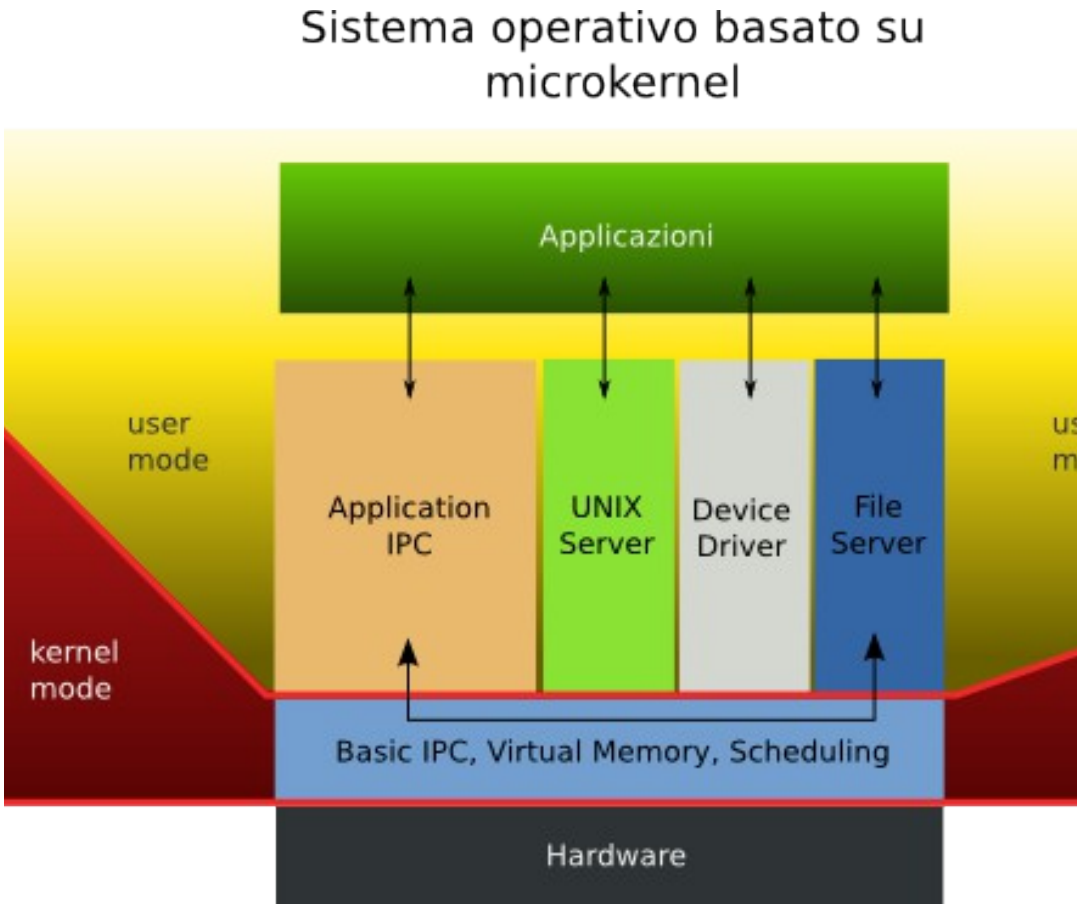
Kernel monolitico



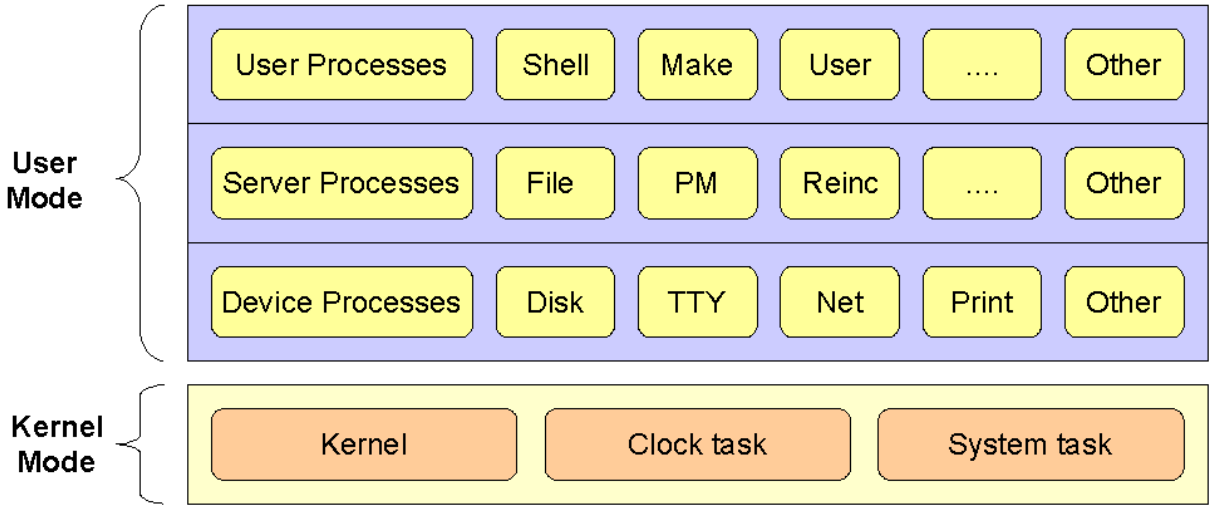
Esempio di sistema monolitico: Linux



Microkernel

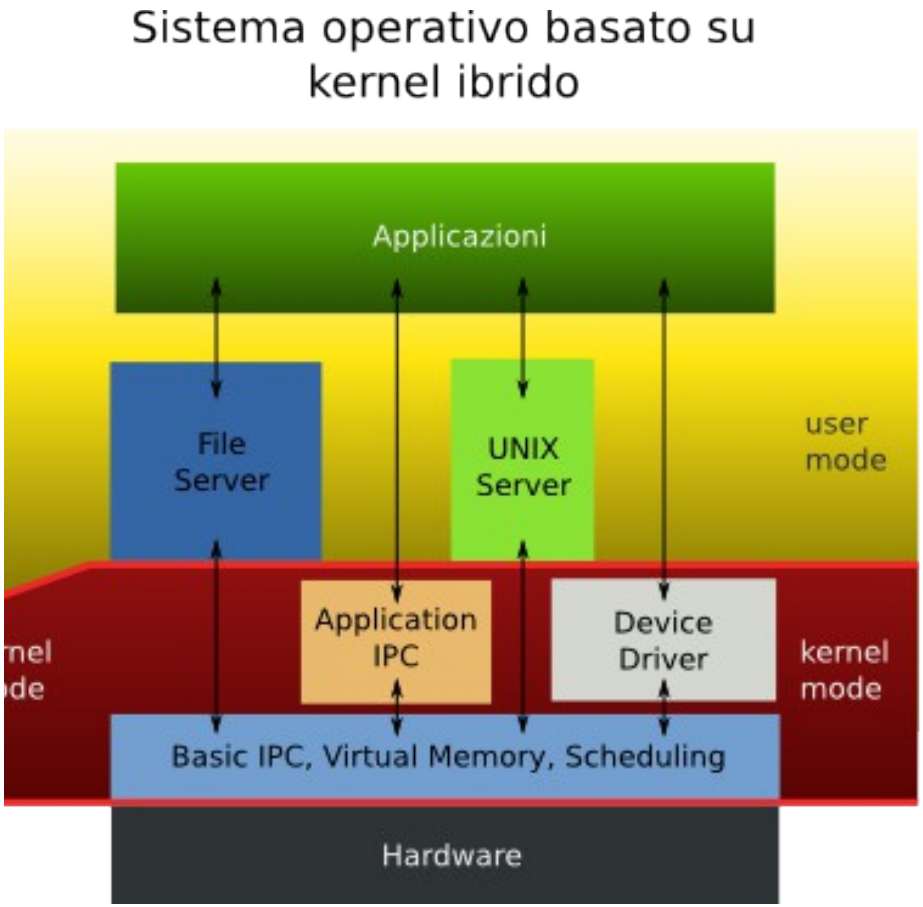


Esempio di sistema microkernel: Minix

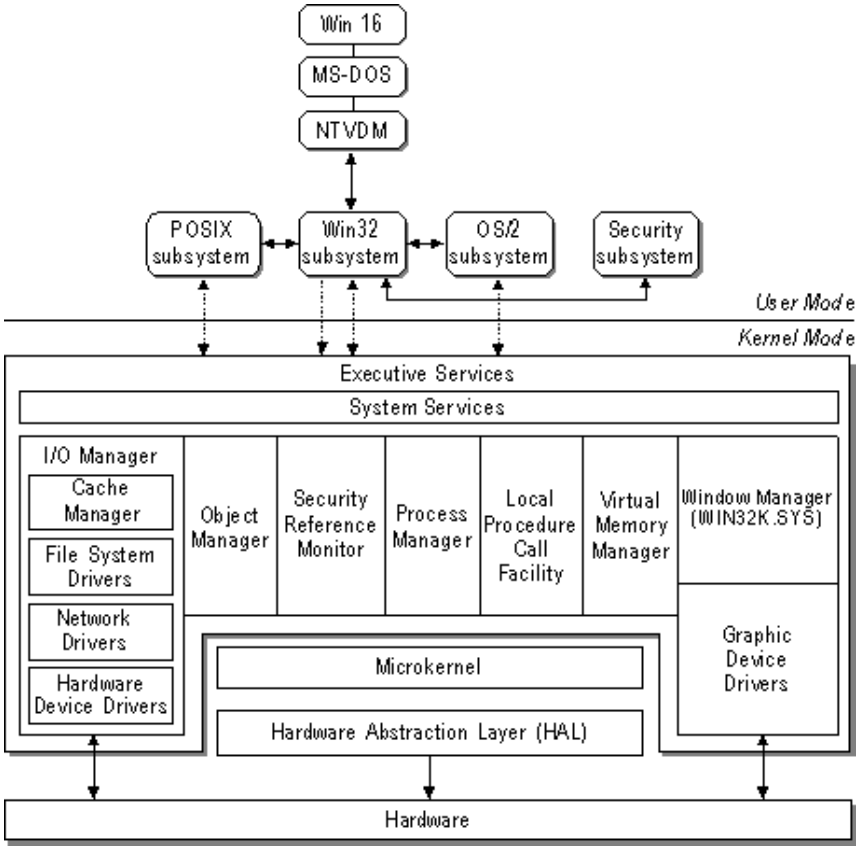


The MINIX 3 Microkernel Architecture

Kernel ibrido



Esempio di sistema ibrido: Windows NT



Virtualizzazione

- **Macchina virtuale (Virtual Machine)**
  - implementa un ambiente virtuale che emula una macchina fisica
- **System Virtual Machine**
  - emula un'architettura completa che permette, per es. l'esecuzione di sistemi operativi diversi sullo stesso sistema
- **Application Virtual Machine**
  - è un software che implementa un'astrazione che nasconde i dettagli dell'hardware e permette la programmazione indipendente dalla piattaforma (es. Java Virtual Machine, Common Language Runtime)

System virtual machine: motivazione

- Disponibilità di sistemi multi-processore e multi-core
  - Aumento delle performance
    - ma non sempre utilizzate al 100%
    - richiesta di servizi on-demand
  - Diminuzione dei prezzi
    - più performance a un prezzo minore rispetto al passato

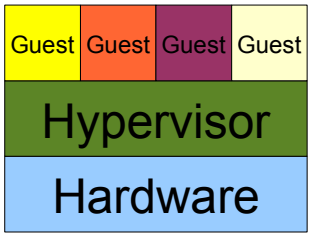
- Con la virtualizzazione è possibile creare più sistemi virtual su una sola macchina fisica, sfruttando più efficientemente le risorse disponibili

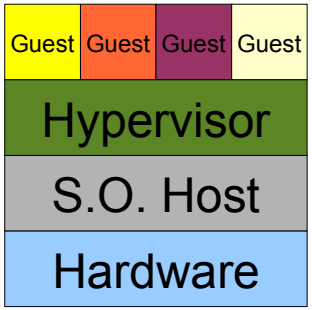
Benefici

- Possibilità di utilizzare **più sistemi operativi** contemporaneamente
- **Isolazione**
  - Sistemi con requisiti diversi
  - Sicurezza
- **Consolidazione**: riunire più servizi su un solo sistema
  - Utilizzo più efficace delle risorse
  - Riduzione dei costi per il mantenimento dell'hardware
- Mantenimento di applicazioni “legacy”
- On-demand computing
  - Allocazione dinamica delle risorse
  - Balanciamento del carico

Hypervisor

- Virtual Machine Monitor (VMM)
- Software che gestisce le virtual machines
  - Regola l'accesso all'hardware
  - “Sistema operativo per sistemi operativi”
- **Tipo 1 “bare metal” (VMWare ESX, Citrix XenServer,...)**
  - L'hypervisor viene eseguito direttamente sull'hardware e ha il controllo delle risorse
  - Non c'è un sistema operativo “host”
- **Tipo 2 “hosted” (VirtualBox, KVM, XEN,...)**
  - L'hypervisor viene eseguito sul sistema operativo principale “host”
  - Altri sistemi (detti “guest”) possono essere gestiti dal hypervisor





Requisiti per la virtualizzazione (Popek e Goldberg)

- **Equivalenza / Fedeltà**
  - Un programma che viene eseguito in un VMM deve mostrare un comportamento essenzialmente identico a quello mostrato quando esso viene eseguito direttamente sulla macchina

- **Controllo delle risorse / Sicurezza**
  - Il VMM deve avere il controllo completo delle risorse virtualizzate

- **Efficienza / Performance**
  - Una percentuale statisticamente significativa delle istruzioni macchina deve essere eseguita senza l'intervento del VMM