Sistemi operativi

1. Introduzione e concetti base

Mauro Fiorentini

Bibliografia

- A.S. Tanenbaum, *Modern Operating Systems*, Prentice-Hall, 1992.
- C. Ghezzi, M. Jazayeri, *Programming Language Concepts*, Wiley, 1982.
- Hennessy, John L. e Patterson, David A., *Computer Architecture*, IV ediz., Morgan Kaufmann, 2007.

Sommario del corso

1. Introduzione
2. Processi e thread
3. Deadlock
4. Gestione della memoria
5. Richiami hardware
6. Input/output
7. File sistem
8. Sicurezza
9. Reti

1. Introduzione

Cos'è un sistema operativo?

- Visione dall'alto: un insieme di funzioni che creano una macchina virtuale, più facile da usare e flessibile.
- Visione dal basso: un gestore di risorse.

Un sistema operativo come macchina virtuale

- Il sistema operativo “estende” le istruzioni macchina.
- Anziché occuparsi dei dettagli...
  - es.: decidi il settore, sposta il braccio del disco, attendi che il settore passi sotto la testina, leggi, verifica il CRC…
- … il programmatore vede comandi semplici e generali.
  - es.: read(file, buffer, size).

Un sistema operativo come gestore di risorse

- Alloca e gestisce le risorse (memoria, tempo di CPU, spazio disco).
- Evita interferenze tra programmi distinti in esecuzione simultanea.
- Garantisce protezione e sicurezza.
  - Evita che un programma, intenzionalmente o per errore, danneggi altri programmi o dati.
  - Impedisce accessi non autorizzati ai dati.

Componenti principali

- Gestione dei processi, comunicazione tra processi e scheduling.
- Gestione della memoria, anche virtuale.
- Gestione I/O.
- File system.

Concetti base

- Processo.
- Spooling.
- Multiprogrammazione.
- Timesharing.
- Context switch.

Processo

- Un processo è un programma in esecuzione.
- Una possibile definizione:

  “la sequenza di stati attraversata dall’elaboratore durante l’esecuzione di un programma”.

Spooling

- Deriva da Simultaneous Peripheral Operation On Line.
- Consiste nel rendere l'I/O parallelo ad altre operazioni.
- Inizialmente applicato alla stampa e al caricamento dei programmi, resi paralleli all'esecuzione di altri processi.
- Poi esteso a tutte le operazioni di I/O.

La multiprogrammazione

- Esecuzione simultanea di più programmi.
  - Il parallelismo è solamente simulato.
- Si possono sfruttare i tempi morti di I/O di un processo per eseguire gli altri.
  - Migliora l'utilizzo della CPU.
  - Se n processi restano ciascuno in attesa di I/O per una frazione p del tempo, la CPU è allocata in media per $1 - p^n$ del tempo.

La multiprogrammazione

Parallelismo simulato tra più processi

Timesharing

- Condivisione della CPU, attribuita ciclicamente ai processi.
  - Ogni processo riceve la CPU per tempi brevi, dai millisecondi ai decimi di secondo.
- Simula l'esecuzione parallela di più processi con una sola CPU.
  - Se i processi eseguono molto I/O, la simulazione è pressoché perfetta.

Tipi di processi

- Un processo si dice “CPU-bound”, se prevale l’utilizzo della CPU.
- Un processo si dice “I/O-bound”, se prevale l’I/O.
- La multiprogrammazione è efficace in presenza di un mix di tipi di processi.

Sospensione e context switch

- Si chiama context switch la cessione della CPU a un differente processo.
- Il processo sospeso deve poter ripartire dallo punto di interruzione, nelle stesse condizioni.
  - A parte il tempo trascorso, l'operazione **non deve essere rilevabile** dal processo.

Process table

- Area che contiene tutte le informazioni relative a un processo.
  - In particolare, la process table contiene per ogni processo:
    - l'immagine di tutti i registri accessibili;
    - la situazione delle risorse allocate (pagine di memoria, file aperti ecc.).
- Quando un sistema operativo sospende un processo, salva l'immagine dei registri nella process table.

System call

- Un processo può richiedere servizi del sistema operativo tramite **system call**: un insieme di primitive che costituiscono la macchina virtuale accessibile al programmatore.
- L'esecuzione di una system call sospende l'esecuzione del processo richiedente e cede il controllo al sistema operativo.

Callback

- Nei sistemi più moderni si usa moltissimo il meccanismo di callback.
  - Un processo notifica al sistema una funzione, da chiamare al verificarsi di un evento (es. pressione di un tasto, movimento del mouse etc.).
  - Quando l'evento si verifica, il sistema interrompe il processo e gli fa eseguire la funzione richiesta.
  - Al termine della funzione il controllo ritorna al sistema operativo, che fa ripartire il processo dal punto in cui era stato interrotto.

Creazione dei processi

- Al bootstrap di solito vengono creati alcuni processi, per consentire la comunicazione con gli utenti.
- Ogni processo può poi crearne altri (detti “figli”).
- Tipicamente un interprete dei comandi utente (“shell”) crea un processo per ogni comando utente.

Terminazione dei processi

- Un processo termina:
  - volontariamente, se completa la sua esecuzione;
  - perché ucciso da un altro;
  - ucciso dal sistema operativo, se commette un errore irrimediabile.

Componenti di un sistema operativo

- Process management: gestisce i processi.
  - creazione, distruzione, comunicazione, sincronizzazione, scheduling.
- Memory management: gestisce la memoria.
- Storage management: gestisce lo spazio disco.
- I/O management: gestisce le comunicazioni con le periferiche.
- File system: implementa file e directory.
  - Fornisce una visione ad alto livello dello storage system.

Altri possibili componenti

- Network management: permette le interazioni con altri calcolatori.
- Protection management: garantisce protezione e separazione tra gli utenti.

La storia dei sistemi operativi

- Si possono grossolanamente distinguere 4 generazioni:
  - Prima generazione, sistemi a valvole, 1945 – 1955;
  - Seconda generazione, sistemi a transistor, 1955 – 1965;
  - Terza generazione, circuiti integrati, 1965 – 1980;
  - Quarta generazione, circuiti VLSI, 1980 – 1990.

La prima generazione

- Problemi da risolvere: calcolo numerico scientifico.
- Linguaggi utilizzati: esclusivamente linguaggio macchina.
  - Verso la fine comparvero i primi assemblatori.
- Sistemi operativi pressoché inesistenti,
  - Al massimo utility generiche per I/O.

Seconda generazione (1)

- I calcolatori cominciano a essere venduti!
- Nascono le prime figure professionali.
  - Inizia la distinzione dei ruoli.
- Compaiono i primi linguaggi.
- Sistemi operativi a lotti, embrionali, nasce lo spooling.
  - I programmi da eseguire vengono copiati su nastro, per essere avviati all'esecuzione.
  - L'output viene generato su nastro e poi stampato tramite differenti macchine.

Seconda generazione (2)

- Si differenziano i mercati.
- Calcolo gestionale:
  - macchine indirizzate a byte;
  - istruzioni per trattamento caratteri e aritmetica decimale;
  - COBOL.
- Calcolo scientifico:
  - macchine con indirizzamento a word;
  - istruzioni per aritmetica floating;
  - FORTRAN.

Terza generazione

- Nasce il time-sharing (CTSS e MULTICS)
- I sistemi operativi divengono grandi e complessi.
  - Migliaia di programmatori, milioni di righe.

Quarta generazione

- Nascono reti e calcolatore personale.
  - Sistemi operativi capaci di gestire la rete e distribuiti.
  - Architetture client-server.

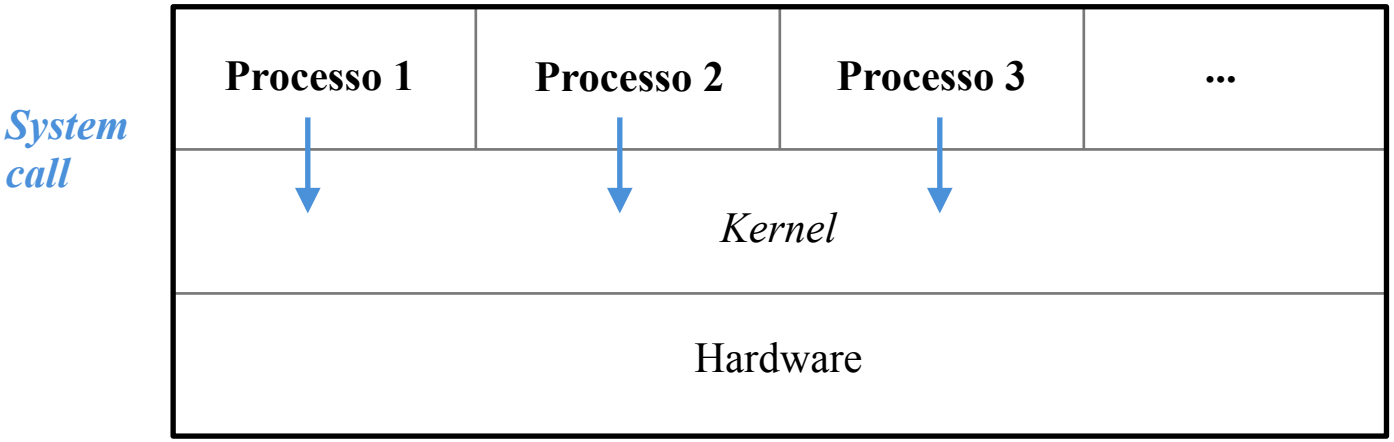
Strutture dei sistemi operativi

- Monolitico.
- A strati.
- Micro-kernel
- Client - server.
- A macchine virtuali.

Sistema operativo monolitico

- Totale assenza di strutture.
- Il sistema è un aggregato di procedure, che si chiamano liberamente tra loro.
- I processi accedono ai servizi tramite system call.
- L'architettura di impiego più frequente.
- Per sistemi operativi di grandi dimensioni è difficile gestire la complessità.

Sistema operativo monolitico



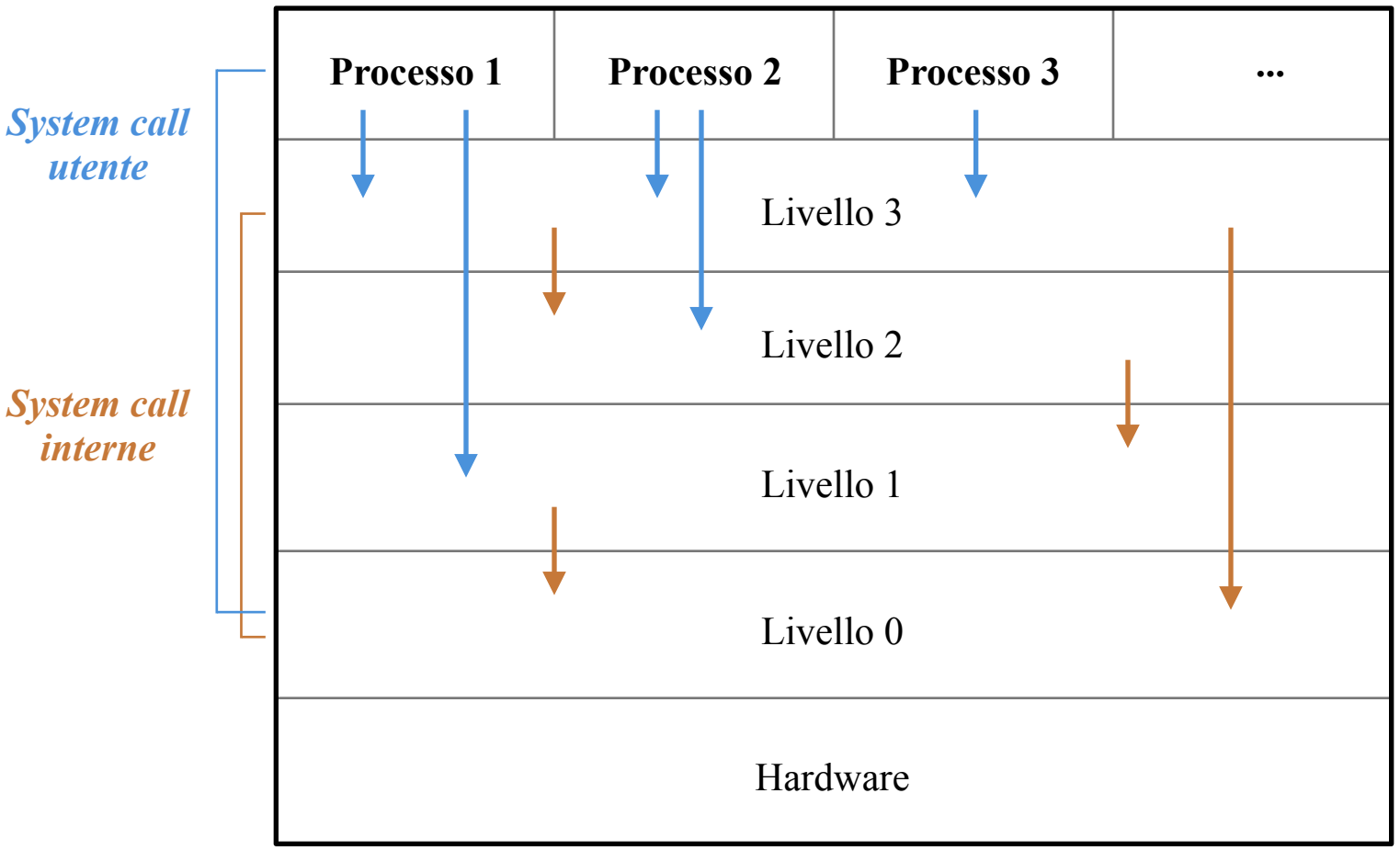
Svantaggi dei sistemi monolitici

- Le componenti interne sono più difficili da realizzare e provare di quelle esterne.
- E' più difficile garantirne la correttezza.
- Solo un processo per volta può eseguire il kernel.

Sistema operativo a strati

- I servizi offerti sono suddivisi in strati.
- Ogni strato è un aggregato di procedure, che si chiamano liberamente tra loro.
- Ogni strato accede ai servizi offerti dagli strati inferiori tramite system call.
  - Nessuna funzione può accedere ai servizi degli strati inferiori, se non tramite system call.
- I processi accedono ai servizi tramite system call ai vari livelli del sistema.

Sistema operativo a strati



Il primo sistema operativo a strati

- Scritto da E.W. Dijkstra in Olanda nel 1968, chiamato “THE” (Technische Hogeschool Eindhoven) per l’Electrologica X8 (32K parole a 27 bit); aveva 6 strati:

  5 operatore;
  4 utente;
  3 gestione I/O;
  2 comunicazione tra processi e operatore;
  1 gestione della memoria e del tamburo;
  0 allocazione del processore e paginazione.

MULTICS

- Gli strati si chiamano “anelli”.
- Ogni anello può accedere ai servizi degli anelli più interni solo con system call, gli argomenti delle quali vengono controllati.
- Il meccanismo degli anelli è supportato dall’hardware.
- Può essere esteso dall’utente verso l’alto.

Micro-kernel (1)

- Il cuore del sistema operativo rende disponibili i servizi base:
  - virtualizza l'hardware,
  - fornisce concorrenza, sincronizzazione e comunicazione tra processi.
- Architettura di base di Mach, Chorus e Windows NT.

Micro-kernel (2)

- L'idea chiave è che un sistema operativo fornisce:
  - multiprogrammazione,
  - un'interfaccia più comoda dell'hardware reale.
- In un sistema a micro-kernel queste funzioni vengono separate:
  - la multiprogrammazione è offerta dal kernel;
  - l'interfaccia semplificata dai sistemi del livello superiore.

Caratteristiche dei micro-kernel

- Le prestazioni peggiorano.
  - Vengono eseguite più chiamate al kernel e più transizioni di stato.
  - Vi è maggiore comunicazione tra le componenti.
- Servono macchine veloci perché l'overhead sia accettabile.
- Si possono sfruttare molto meglio i sistemi multiprocessore.

Sistemi client-server (1)

- I sistemi client-server esasperano la tendenza di estrarre funzionalità dal kernel, portandole a livello più alto e facendole diventare processi, spostando **tutte** le funzionalità in processi.
  - Il kernel fornisce solo il supporto di base alla multiprogrammazione, alla paginazione e alla comunicazione tra processi.
  - In questo modo il sistema diviene più semplice e le funzionalità offerte dai processi possono essere facilmente aggiunte, rimosse, sostituite, addirittura con la macchina in funzione.

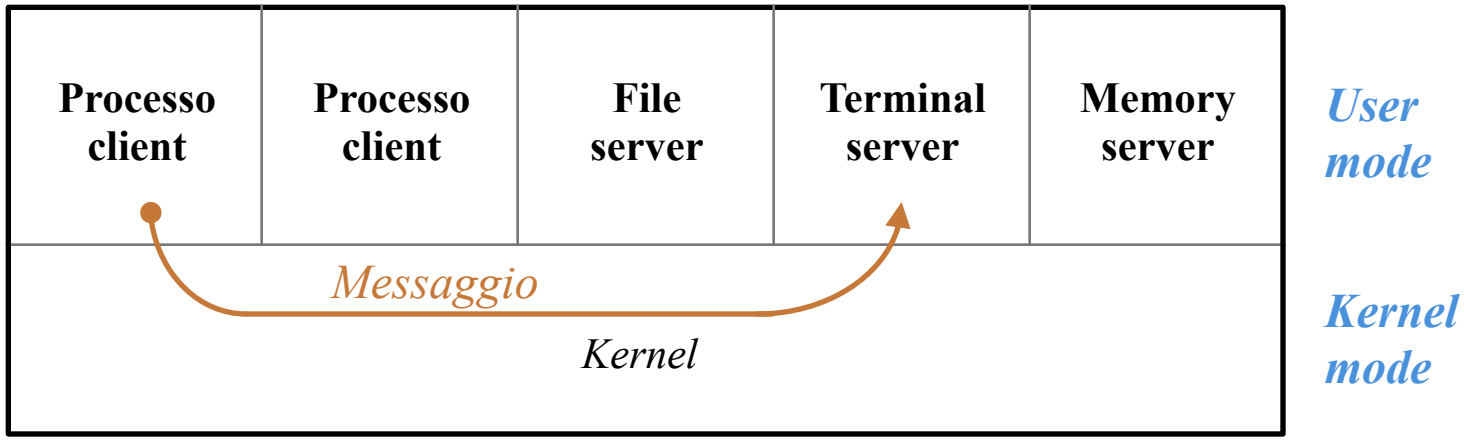
Sistemi client-server (2)

- I sistemi di questo tipo possono diventare facilmente sistemi distribuiti, in modo trasparente ai processi.
  - Il fatto che il processo che risponde a una richiesta sia sulla stessa macchina o meno diventa irrilevante.
  - Molte applicazioni sono costruite allo stesso modo.

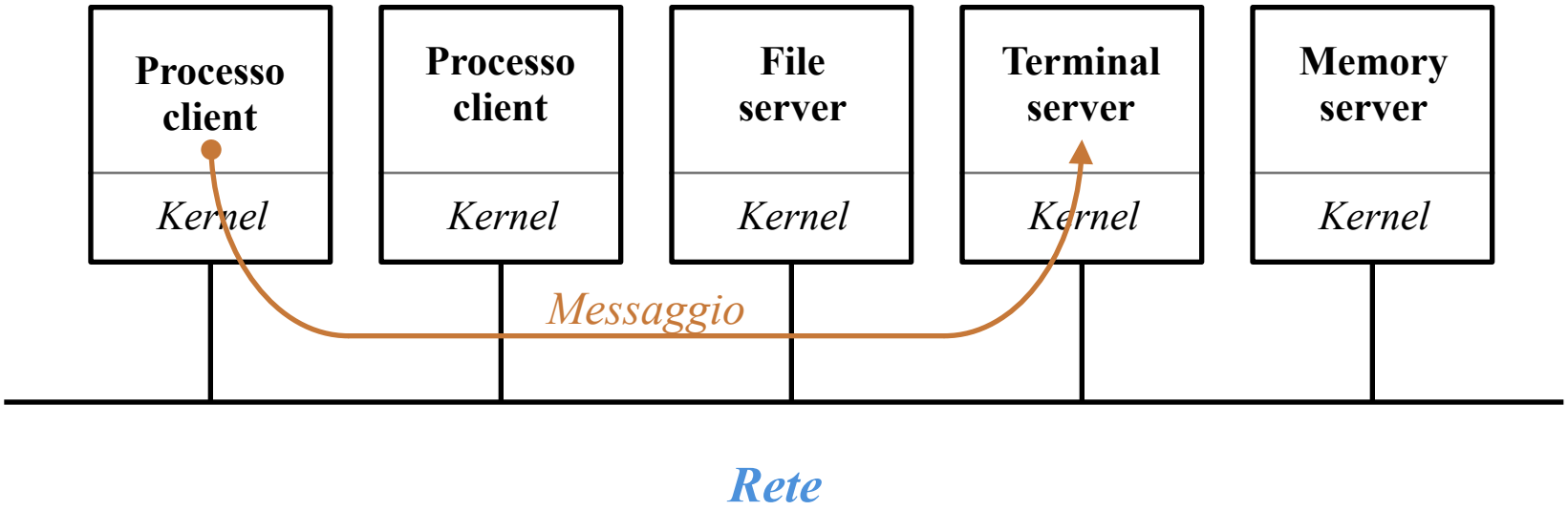
Sistemi client-server (3)

- Alcuni dei processi server devono girare in stato kernel.
  - Utilizzano istruzioni di I/O.
  - Devono poter scrivere nella memoria di altri processi.
- In alternativa il kernel può offrire primitive di basso livello per queste operazioni, lasciando tutti i server in stato utente.
  - L'implementazione è più complessa e lenta, ma più sicura.

Client-server monoprocessore



Client-server multiprocessore



Macchine virtuali

- Facilmente realizzabili su un sistema a micro-kernel.
- L'interfaccia visibile del sistema non è costituita da system call, ma da una copia esatta (parzialmente simulata) di una macchina.
- Sopra questo strato si scrive un sistema operativo, che si comporta come se fosse realmente padrone della macchina.

Vantaggi delle macchine virtuali

- Una macchina virtuale offre parecchi vantaggi:
  - è più facile isolarla, aumentando la sicurezza del sistema;
  - se il sistema operativo della macchina virtuale è poco affidabile, si limitano i danni che può fare;
  - Si può condividere un unico calcolatore tra più utenti, senza rischi.
  - Si possono facilmente replicare installazioni complesse, senza dover installare più volte lo stesso s/w.

Tipi di macchine virtuali

- Se la macchina virtuale è differente da quella che ospita il sistema operativo, è necessario scrivere un interprete, che la simula, eseguendone le istruzioni.
  - ES.: JVM.
- Se invece la macchina virtuale è una copia di quella ospitante, l'implementazione è di solito molto più semplice.
  - Il processore esegue normalmente le istruzioni.
  - Quando utilizza istruzioni privilegiate o accede ad aree di memoria riservate, un trap generato dall'h/w attiva funzioni di supporto s/w, che simulano l'operazione.

Architettura di una macchina virtuale

- Generalmente composta di due parti:
  - Un processo in stato utente che esegue la VM.
  - Un monitor in stato sistema, che intercetta tutti i trap generati dalla macchina virtuale ed esegue le azioni appropriate o li comunica al processo.

Supporto h/w per macchine virtuali (1)

- L'esecuzione diretta è però possibile solo se il comportamento di **tutte** le istruzioni che non generano un trap quando eseguite in stato utente è **identico** in modo system e utente.
- In particolare, non devono esistere istruzioni che:
  - hanno un comportamento in stato sistema e uno diverso (o non fanno nulla) in stato utente, senza generare trap (es. POPF e LAHF su Intel 80286);
  - permettono di leggere lo stato (utente o sistema) da stato utente.
- In pratica, deve essere impossibile per del codice capire se gira in una macchina reale o virtuale.

Supporto h/w per macchine virtuali (2)

- Se il set di istruzioni non è progettato per la virtualizzazione, questa può essere molto inefficiente.
- Alcune architetture sono nate per essere virtualizzabili, permettendo la virtualizzazione con minimo overhead.
  - Es.: IBM 370.
- Su alcune macchine è prevista una modalità apposita di esecuzione del codice, per facilitare la creazione di macchine virtuali e renderle più efficienti.
  - Es. Intel, AMD.

Supporto h/w per macchine virtuali (3)

- Una delle cose più complesse da gestire è la page table.
  - Il sistema operativo della macchina virtuale “crede” di gestire una page table, che però non è quella reale.
  - Bisogna intercettare tutti i tentativi di modificarla, modificando di conseguenza la page table reale (detta “shadow page table”).
  - Se l’h/w supporta un livello di indirettezza aggiuntivo, la shadow page table non è necessaria (es. IBM 370).

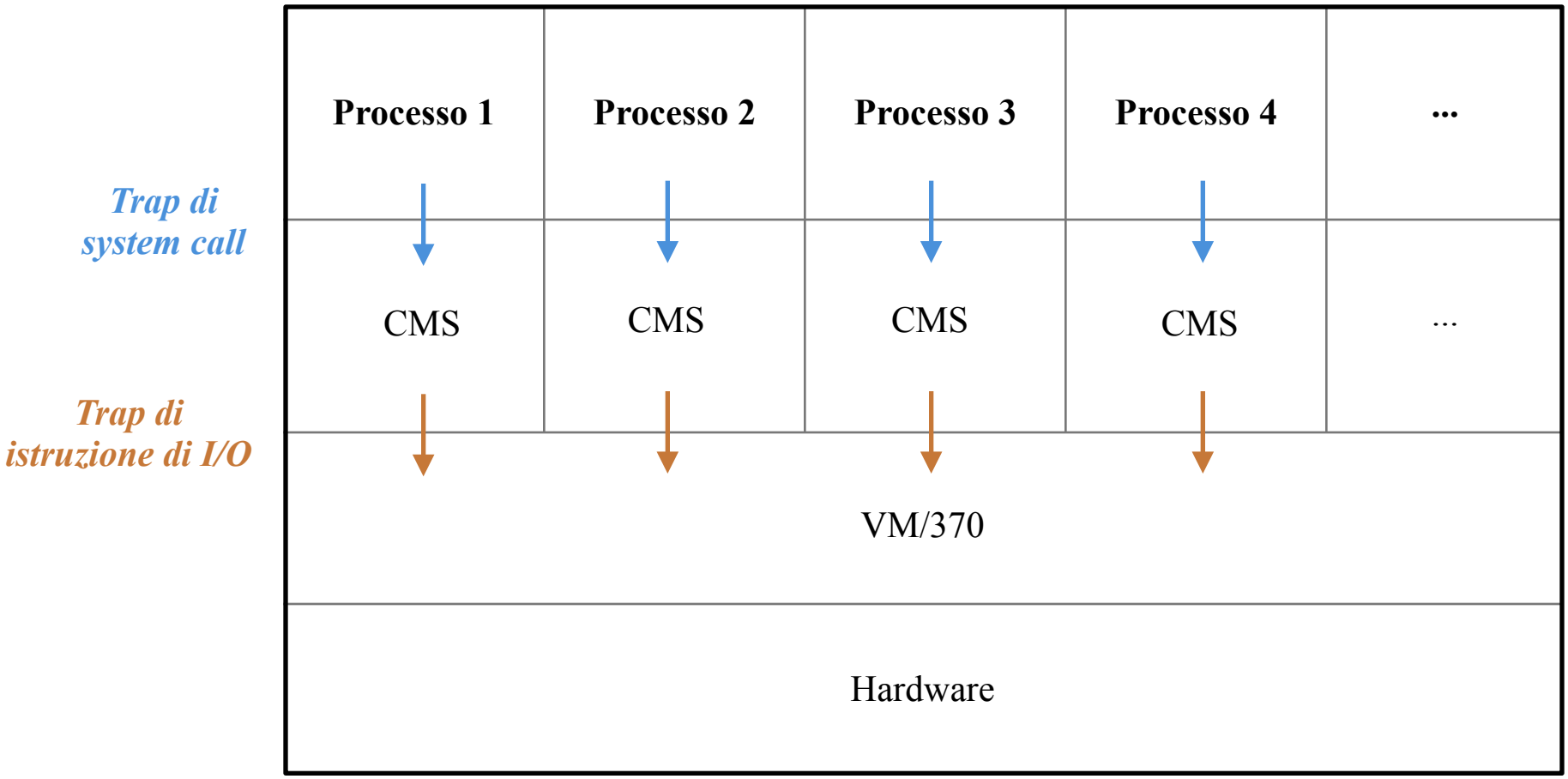
Macchine virtuali e I/O

- L'I/O di una macchina virtuale è molto complesso da gestire.
  - Anche per l'enorme varietà di dispositivi esistenti e dei driver relativi.
- Far gestire interamente l'I/O a un driver sulla macchina virtuale può essere molto inefficiente.
  - Bisogna "tradurre" ogni azione eseguita sul dispositivo virtuale in termini di azioni disponibili su quello reale.
- La soluzione normale è presentare alla macchina virtuale dispositivi "generici" molto semplici e lasciare al monitor il compito di gestire quelli reali.
  - Una partizione di un disco o un grosso file possono essere fatti vedere come se fossero un disco dedicato.

VM/370

- Sugli IBM 360 girava l'OS/360: un sistema batch.
- Il suo successore ufficiale in IBM era un sistema time-sharing: il TSS/360.
- 50 milioni di dollari dopo, il sistema era tanto lento e ingombrante che pochi utenti lo vollero.
- Il VM/370, sviluppato a Cambridge dalla IBM, ebbe invece grande successo.

Macchina virtuale VM 370



VM/370 (1)

- I processi accedono ai servizi di sistema operativo tramite system call.
- Il sistema operativo (CMS: Conversational Monitor System) offre tutti i servizi necessari agli utenti time-sharing.
- Quando il CMS tenta di eseguire I/O fisico o di accedere ad alcuni registri riservati, la CPU genera un trap, che cede il controllo al VM/370.

VM/370 (2)

- Il trap handler del VM/370 simula l'operazione e restituisce il controllo al CMS.
- Ogni istanza del CMS "crede" di avere una stampante, dei dischi, un lettore di schede, un terminale ecc.; in realtà molte unità sono simulate o condivise.
- Possono essere contemporaneamente utilizzati più sistemi di tipi diversi (batch, time-sharing, monitor transazionali).

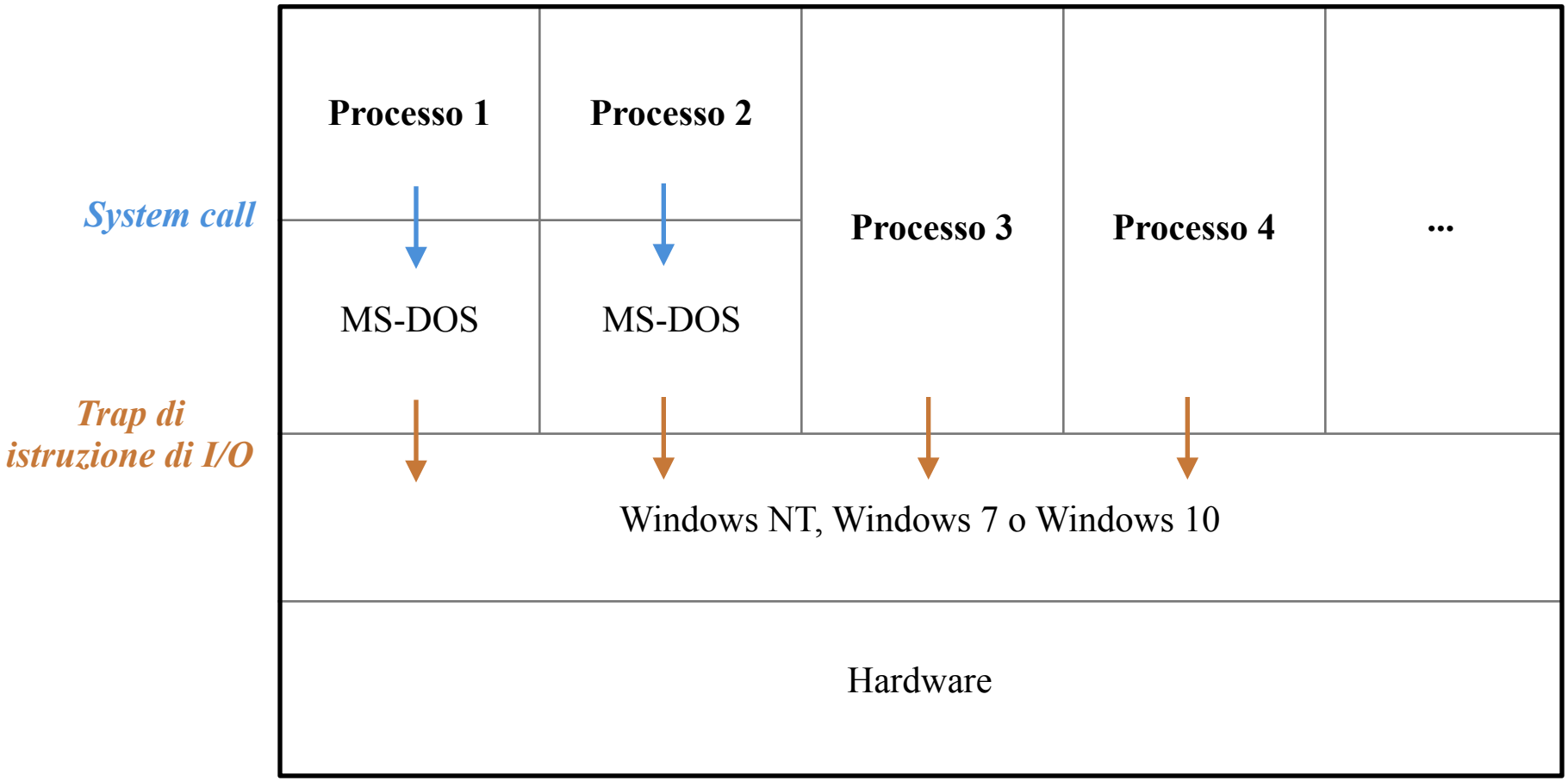
MS-DOS virtuale (1)

- Windows 95 e NT mettono a disposizione macchine virtuali MS-DOS.
  - L'hardware mette a disposizione alcune risorse (una modalità di funzionamento apposita della CPU), per facilitare l'implementazione.
  - Possono essere contemporaneamente eseguite più copie di MS-DOS.
  - Ogni istanza di MS-DOS con un programma in esecuzione viene vista come un processo da Windows.

MS-DOS virtuale (2)

- Le macchine virtuali coesistono con normali processi Windows, che richiedono servizi con comuni system call.
  - Sono solo una “feature” aggiuntiva, integrata nel sistema, non l’unico modo di accedere ai servizi del sistema.

Macchina virtuale MS-DOS



MS-DOS sotto Windows

- I processi eseguono MS-DOS, come se avessero il controllo della macchina.
  - Ogni istanza di MS-DOS “crede” di controllare le periferiche, che in realtà sono simulate (video) o condivise (tastiera, dischi).
- Quando tentano di eseguire I/O fisico o di accedere ad alcuni registri riservati, la CPU genera un trap, che cede il controllo a Windows.
- Il trap handler di Windows simula l’operazione e restituisce il controllo a MS-DOS.