Macchine virtuali

Pietro Braione

Reti e Sistemi Operativi – Anno accademico 2021-2022

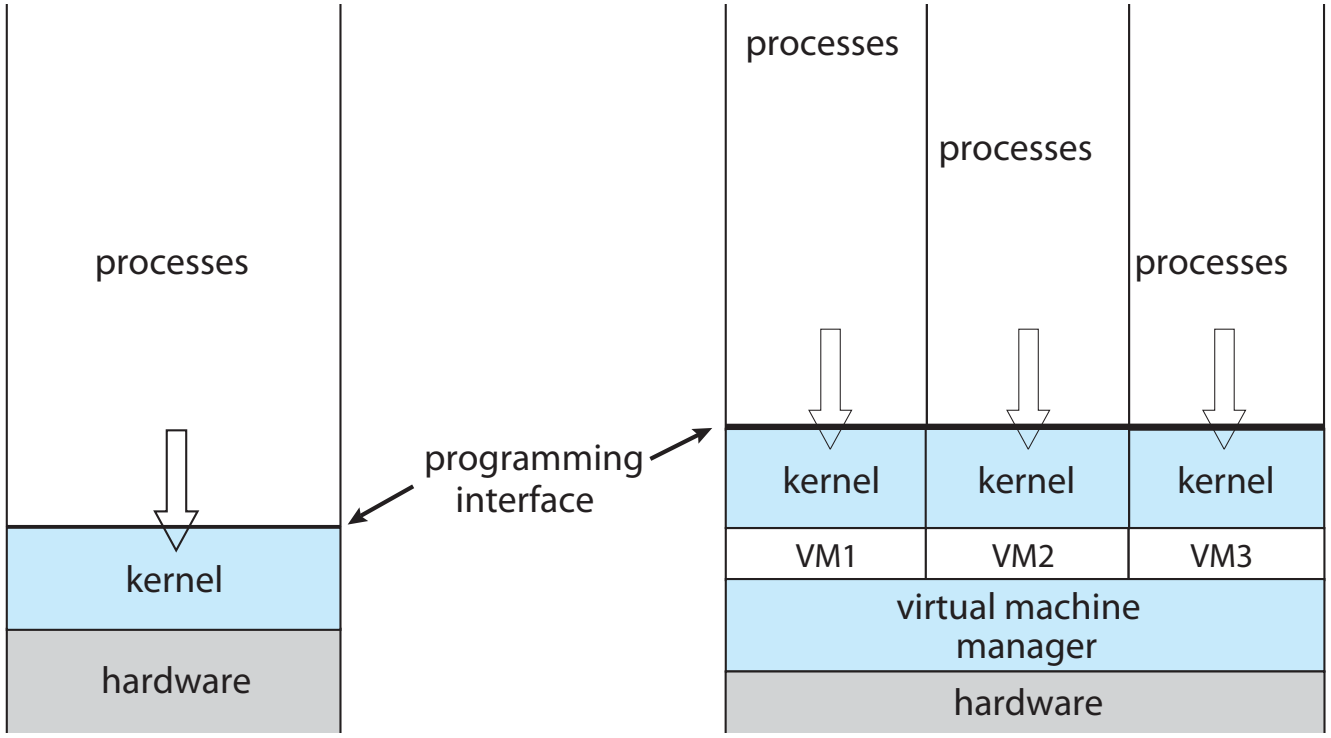
Obiettivi

- Presentare i vantaggi delle macchine virtuali
- Analizzare le diverse tecnologie di macchine virtuali
- Descrivere i metodi utilizzati per implementare la virtualizzazione

Introduzione

- Idea: astrarre l'hardware di un computer in diversi ambienti di esecuzione, in maniera che ogni ambiente abbia l'illusione di essere in esecuzione su un proprio computer privato
  - Concetto simile alla stratificazione in un sistema operativo
  - Ma il sistema virtuale creato (**macchina virtuale**, o VM) è esso stesso, di solito, in grado di eseguire un sistema operativo
- Componenti:
  - **Host**: il sistema hardware
  - **Virtual machine manager** (VMM), o **hypervisor**: crea ed esegue le macchine virtuali, le quali offrono un'interfaccia che è identica (o quasi, nel caso della paravirtualizzazione) a quella dell'host
  - **Guest**: il programma eseguito dalla macchina virtuale, di solito un sistema operativo
- In questo modo una singola macchina fisica può eseguire diversi sistemi operativi contemporaneamente

Modelli di sistema



Non virtualizzato

Virtualizzato (tipo 1)

Vantaggi e caratteristiche (1)

- L'host è protetto dalle VM, e le VM sono protette l'una dalle altre
  - Maggiore sicurezza e affidabilità
  - Condivisione di risorse attraverso unità disco virtuali condivise e comunicazione di rete attraverso interfacce di rete virtuali
- Possibilità di sospendere una VM
  - Possibilità di copiarla o spostarla su un altro host e farla riprendere dal punto di sospensione
  - Possibilità di creare istantanee (snapshot) di uno stato e di ripristinare lo stato quando necessario
- Possibilità di eseguire diversi sistemi operativi sullo stesso host
  - Consolidamento di più macchine a bassa utilizzazione su un unico host
  - Sviluppo e testing di una stessa applicazione su diversi ambienti

Vantaggi e caratteristiche (2)

- **Templating**: possibilità di creare un ambiente standard OS + applicazioni, e di distribuire tale ambiente a molti clienti, abbattendo i costi di installazione
- **Live migration**: possibilità di trasferire una VM da un host ad un altro senza interrompere il suo funzionamento o le connessioni di rete che questa ha attive
- Tutte queste caratteristiche permettono il paradigma del **cloud computing**, nel quale utilizzando opportune API un programma può chiedere ad un'infrastruttura cloud, che mette a disposizione le risorse hardware, di creare macchine virtuali per l'esecuzione di specifiche applicazioni

Virtual CPU

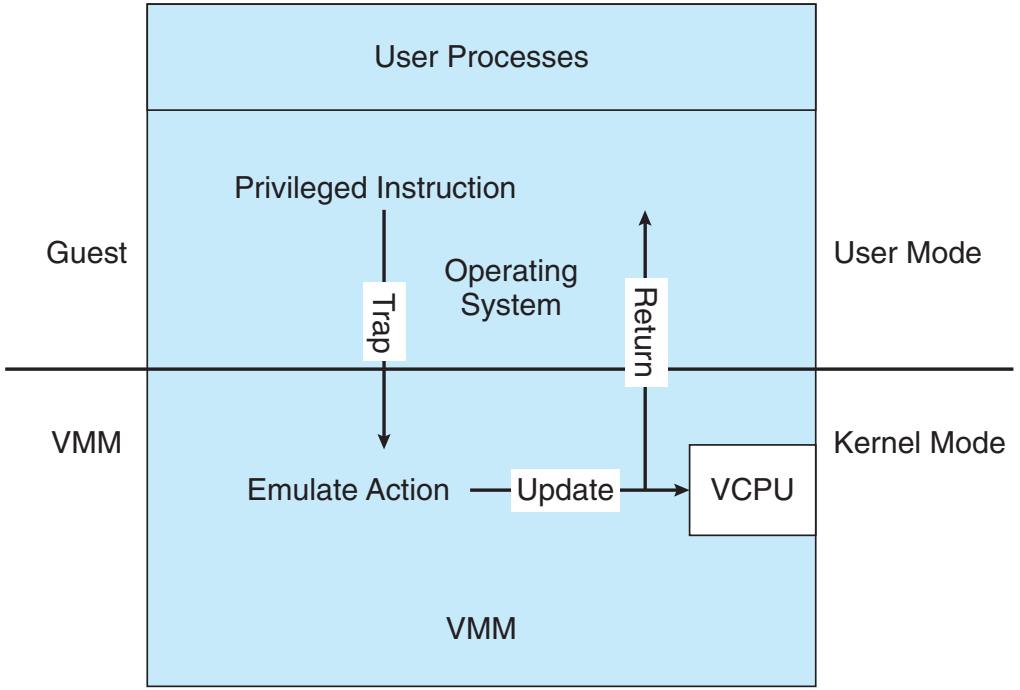
- Un concetto importante per gli hypervisor è quello di **virtual CPU** (VCPU)
- La VCPU è una rappresentazione dello stato in cui un certo guest ritiene che sia la CPU
- È quindi una sorta di contesto che l'hypervisor mantiene per ogni guest, simile al PCB per un sistema operativo
- Quando l'hypervisor assegna la CPU ad un guest utilizza le informazioni della sua VCPU per caricare il giusto contesto

Virtual CPU: modalità

- Problema: se l'hypervisor esegue in modalità kernel, il sistema operativo guest non può a sua volta eseguire in modalità kernel, ma deve per forza eseguire in modalità utente
- Allo stesso tempo, il sistema operativo guest deve avere un supporto dalla VCPU per isolare i processi utente
- Quindi la macchina virtuale deve avere una modalità **kernel virtuale** (per l'OS guest) e modalità **utente virtuale** (per le applicazioni che eseguono sull'OS guest)
- Entrambe queste modalità, in realtà, sono eseguite nella modalità utente (autentica) della CPU fisica
- Come sono implementate?

Trap-and-emulate

- Quando il sistema operativo guest esegue un'istruzione privilegiata viene generata una trap
- Tale trap viene intercettata dall'hypervisor che analizza l'istruzione privilegiata che ha generato la trap
- Quindi l'hypervisor emula la sua esecuzione modificando lo stato della VCPU (in maniera diversa a seconda se è in stato virtual user o virtual kernel)
- Infine l'hypervisor restituisce il controllo al guest



Trap-and-emulate: caratteristiche

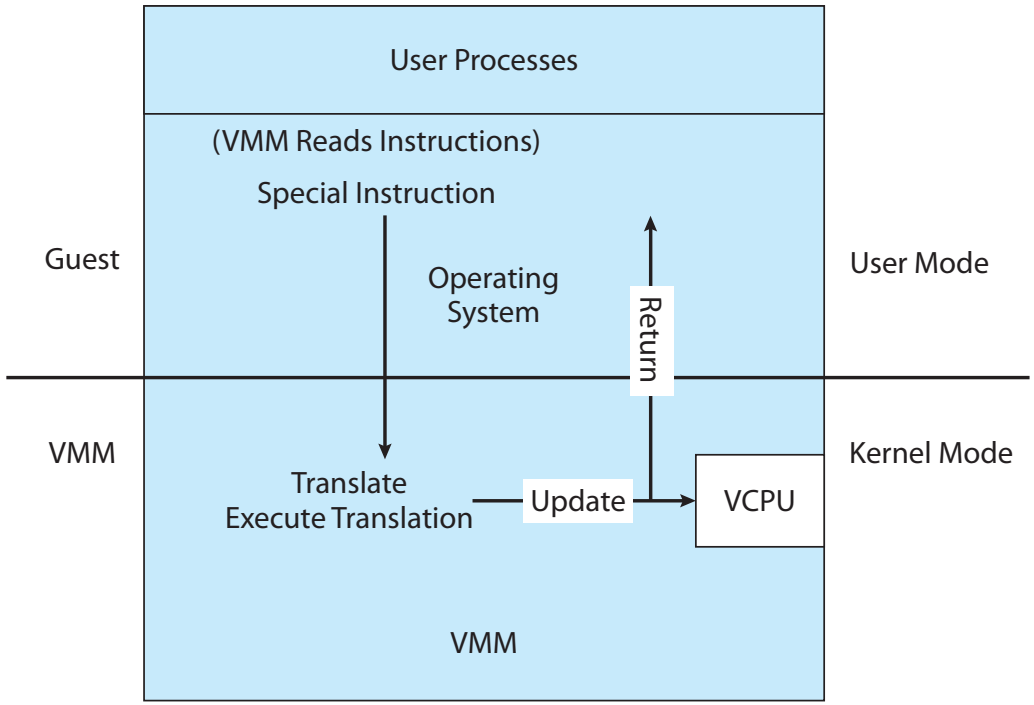
- Il codice utente può essere eseguito a velocità pressoché nativa
- Le istruzioni privilegiate del kernel (attenzione, solo quelle privilegiate!) sono eseguite molto lentamente, dal momento che ognuna genera una trap e va emulata singolarmente dall'hypervisor
- Processori moderni mettono a disposizione modalità aggiuntive oltre a user e kernel, per facilitare la gestione delle modalità virtual user e virtual kernel

Criteri di virtualizzazione trap-and-emulate

- In un classico lavoro del 1974, Popek e Goldberg hanno stabilito i criteri secondo i quali una architettura hardware è virtualizzabile in maniera trap-and-emulate
- Chiamiamo **istruzioni sensibili** quelle istruzioni macchina che modificano o sono influenzate dalla configurazione del processore (rilocazione, interruzioni, modalità…)
- Chiamiamo **istruzioni privilegiate** le istruzioni che, se eseguite in modalità utente, generano una trap
- Una architettura hardware è virtualizzabile in maniera trap-and-emulate solo se le istruzioni sensibili sono un sottoinsieme di quelle privilegiate
- Purtroppo i processori Intel x86 non rispettano tali criteri:
  - Alcune istruzioni sensibili eseguite in modalità utente non generano trap ma vengono semplicemente ignorate (esempio: POPF)
  - Altre istruzioni permettono di leggere uno stato sensibile in modalità utente (esempio: è possibile leggere il selettore di segmenti di codice in modalità utente e capire se si è in modalità utente o kernel)

Traduzione binaria

- È un modo per virtualizzare certe architetture hardware che non rispettano i criteri di Popek e Goldberg
- Se la VCPU è in modalità utente virtuale, il codice è eseguito nativamente sulla CPU fisica
- Se la VCPU è in modalità kernel virtuale, il VMM analizza dinamicamente le istruzioni che la VCPU deve eseguire, basandosi sul program counter
- Se tali istruzioni sono sensibili, vengono dinamicamente tradotte in opportune sequenze di istruzioni che cambiano correttamente lo stato della VCPU



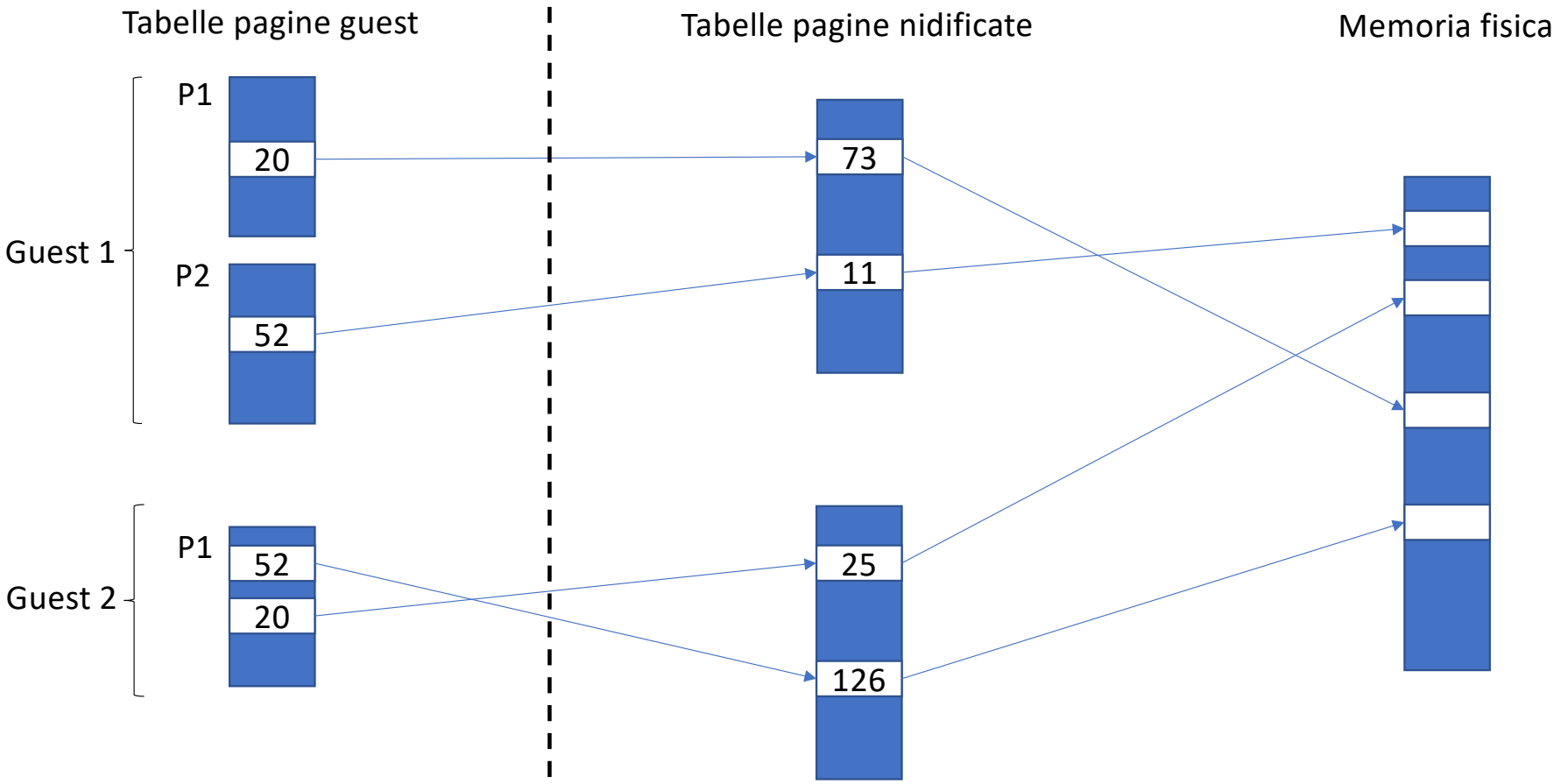
Traduzione binaria: caratteristiche

- VMWare è stato pioniere nell'ambito della traduzione binaria, tecnica introdotta per consentire la virtualizzazione dei processori Intel x86
- La performance di questo metodo è scarsa a meno che non venga introdotta una qualche forma di caching
- L'approccio «translate once» di VMWare memorizza la traduzione di blocchi di istruzioni in una cache, dalla quale la traduzione viene recuperata ogni volta che viene incontrato lo stesso blocco
- In tal modo si è ottenuta una performance con un overhead del solo 5% del tempo di esecuzione (benchmark: boot di Windows XP)
- Un confronto tra la performance di trap-and-emulate e traduzione binaria è comunque difficile: sulle architetture odierne le trap deteriorano le cache e i TLB, e quindi in certe situazioni la traduzione binaria con caching può diventare conveniente

Tabelle delle pagine nidificate

- Un altro problema è la virtualizzazione dell'accesso alla memoria: come si può permettere ai sistemi operativi guest di mantenere e modificare le tabelle delle pagine dei propri processi?
- Il metodo utilizzato, valido sia per la soluzione trap-and-execute che per la traduzione binaria, è quello delle **tabelle delle pagine nidificate**
- L'hypervisor mantiene per ogni guest una tabelle delle pagine nidificata che mappa i frame (virtuali) del guest sui frame (fisici) della macchina hardware
- Ad ogni cambio di contesto del guest, la tabella nidificata è usata per tradurre la tabella delle pagine del guest in maniera che riferisca direttamente i frame fisici, e tale tabella tradotta viene fornita alla CPU
- I cambiamenti ad una tabella delle pagine del guest devono essere riflessi sulla corrispondente tabella delle pagine nidificata e sulla tabella delle pagine della CPU: a tale scopo la memoria delle tabelle delle pagine del guest è impostata read-only e l'hypervisor intercetta la trap dovuta al tentativo di scrittura su di essa
- Problema: basse prestazioni (aumento dei TLB miss e flush, trap all'hypervisor), overhead in memoria; per tale motivo le CPU odierne offrono supporto specifico alla virtualizzazione del paging

Tabelle delle pagine nidificate: esempio



Ciclo di vita macchine virtuali

- L'hypervisor crea le macchine virtuali, in base ai parametri (numero di processori virtuali, memoria, spazio su disco, dispositivi di I/O virtuali…)
- Le risorse fisiche assegnate possono essere dedicate (es. spazio disco virtuale) oppure multiplexate (es. CPU e interfacce di rete virtuali)
  - Negli hypervisor di tipo 0 di solito tutte le risorse sono dedicate
  - Negli altri hypervisor possono anche essere multiplexate
- Quando una macchina virtuale non è più necessaria le sue risorse possono essere liberate ed assegnate ad altre macchine virtuali

Tipologie di hypervisor

- **Hypervisor di tipo 0**: implementati in hardware, forniscono il supporto per gestione e creazioni di macchine virtuali via firmware
  - IBM LPARs
  - Oracle LDOMs
- **Hypervisor di tipo 1**: implementati come strato software immediatamente al di sopra dell'hardware
  - VMWare ESX
  - Joyent SmartOS
  - Citrix XenServer
  - Microsoft Windows Sever con Hyper-V
  - RedHat Linux con KVM
- **Hypervisor di tipo 2**: applicazioni che eseguono su sistemi operativi standard
  - VMWare Workstation e Fusion
  - Oracle Virtualbox
  - Parallels Desktop

Hypervisor di tipo 0

- Una vecchia idea, che va sotto diversi nomi («domini», «partizioni»)
- Ogni guest ha delle risorse hardware dedicate
- Risulta difficile condividere le risorse di I/O: spesso si utilizza una partizione di controllo, che esegue dei demoni che gestiscono l'I/O condiviso
- A differenza degli altri tipi di hypervisor è possibile eseguire un altro hypervisor come guest (virtualizzazione nella virtualizzazione)
- Può essere completamente trasparente per il sistema operativo guest, oppure paravirtualizzato (es. per permettere la migrazione di risorse hardware a runtime)

Hypervisor di tipo 1

- Eseguono sulla macchina hardware al posto del sistema operativo
- Uno o due ordini di grandezza più semplici di un sistema operativo, e quindi molto più robusti, sono i «sistemi operativi dei data center»:
  - Usati per consolidare server e OS multipli su una stessa macchina hardware
  - Permettono di spostare un guest da una macchina all'altra rapidamente
  - Permettono di fare snapshot e di clonare un guest
- Eseguono in modalità kernel e implementano i device driver dei dispositivi della macchina hardware
- Un'altra tipologia di hypervisor di tipo 1 comprende sistemi operativi general-purpose, come Windows o Linux, con funzionalità hypervisor
- Non è possibile eseguire un hypervisor di tipo 1 come guest di un hypervisor di tipo 1

Hypervisor di tipo 2

- Meno interessanti per il nostro corso
- Sono normali processi che eseguono su un determinato sistema operativo (il sistema operativo **host**)
- Il sistema operativo host non fornisce alcun supporto all'implementazione degli hypervisor di tipo 2
- Tende ad avere una performance decisamente inferiore rispetto ad un hypervisor di tipo 0 o 1
- Utile per sperimentazione di sistemi operativi senza dover dedicare una macchina hardware apposita, o per lo sviluppo di software multipiattaforma

Varianti della virtualizzazione

- **Paravirtualizzazione**: richiede una modifica del guest perchè questo collabori con l'hypervisor
  - Xen
  - Microsoft Hyper-V
- **Emulazione**: permette di eseguire applicazioni per un certo ambiente hardware su un ambiente hardware diverso
  - Bochs
  - QEMU
  - Digital FX!32
  - Apple Rosetta e Rosetta 2
  - MAME
- **Virtualizzazione dell'ambiente di programmazione**: fornisce un ambiente virtuale ottimizzato per l'esecuzione di applicazioni in uno o un insieme di linguaggi
  - JVM
  - .NET CLR
- **Containers**: forniscono funzionalità simili alla virtualizzazione permettendo di segregare applicazioni che eseguono su un certo sistema operativo
  - Solaris Zones
  - BSD Jails
  - AIX WPARs
  - Linux LXD

Paravirtualizzazione

- Nella paravirtualizzazione l'hypervisor non replica esattamente l'hardware sottostante, e quindi richiede che il sistema operativo guest venga opportunamente modificato perché collabori con l'hypervisor
- Anziché utilizzare alcune funzionalità dell'hardware come le istruzioni sensibili, il guest deve effettuare opportune chiamate all'hypervisor, o **hypercall**
- La paravirtualizzazione permette di virtualizzare alcune architetture hardware che non rispettano i criteri di Popek e Goldberg, e di ottenere prestazioni più elevate
- Esempi di tecniche di paravirtualizzazione usate in Xen:
  - L'I/O è realizzato con buffer circolari di memoria condivisa tra guest e hypervisor
  - La paginazione non è realizzata con le tabelle delle pagine nidificate: il sistema operativo guest effettua direttamente il mapping alle pagine fisiche, ma aggiorna le tabelle delle pagine attraverso una opportuna hypercall che verifica che la pagina fisica appartenga effettivamente al guest
- Sempre meno necessaria con il migliorare del supporto alla virtualizzazione da parte dei nuovi processori

Emulazione

- L’emulazione riproduce un ambiente hardware diverso rispetto all’hardware sul quale è in esecuzione
- Più complicata della virtualizzazione
  - Richiede la traduzione di tutte le istruzioni della CPU guest in quelle della CPU host
  - Tecniche: interpretazione o traduzione binaria dinamica
- Il problema principale è la performance molto bassa
- Spesso usata per eseguire software per macchine hardware obsolete, la cui performance è ottenibile con la potenza di calcolo delle CPU attuali (esempio: retrogaming)

Virtualizzazione dell'ambiente di programmazione

- Si tratta di eseguire i programmi in un determinato linguaggio (o insieme di linguaggi) in un ambiente di esecuzione specifico per essi
- Non è virtualizzazione nel senso di «replica di hardware sottostante»
- Tecniche: interpretazione o traduzione binaria dinamica («just-in-time (JIT) compilation»)
- Notare la somiglianza con l'emulazione (può essere considerata come l'emulazione di un opportuno processore specifico per il linguaggio)

Containers

- Spesso le macchine virtuali servono solo ad isolare le applicazioni, facilitarne la gestione, il loro avvio, arresto e spostamento
- Quando le applicazioni sono scritte per un solo sistema operativo non è necessario replicare lo stesso sistema operativo su più macchine virtuali
- I containers, o zone, creano uno strato virtuale tra sistema operativo ed applicazioni
- Ogni container è un sistema operativo virtualizzato, che offre l'illusione di avere le proprie risorse di sistema dedicate (processi, account utente, indirizzi di rete…)
- Memoria e CPU sono divise tra i diversi container, ed ogni container ha il proprio scheduler per distribuire tali risorse tra i propri processi
- Vantaggi:
  - Molto più rapidi da avviare e arrestare rispetto alle macchine virtuali
  - Facili da gestire in maniera automatica (strumenti Docker e Kubernetes)
- Svantaggio: i containers sono robusti quanto il sistema operativo sul quale sono ospitati

Alcuni esperimenti interessanti all'UCSD

- (tratto dalle slides di Alex C. Snoeren per un corso all'UCSD)
- «Fork» di macchine virtuali con copy-on-write (Michael Vrable):
  - Permette di scalare il numero di macchine virtuali su una stessa macchina hardware (anche centinaia)
- Dilatazione del tempo (Diwaker Gupta):
  - L'hypervisor può controllare la frequenza degli interrupt del timer consegnati ai guest, e quindi può creare l'illusione di un tempo che scorra più lentamente o più velocemente del tempo reale
  - Possibile applicazione: rallentare i timer in maniera che gli altri dispositivi (dischi, rete…) appaiano molto più veloci della CPU virtuale, così da sperimentare come si comportano i sistemi con futuri dispositivi hardware veloci
- Cluster virtuali (Marvin McNett):
  - All'UCSD esiste un cluster di 200 nodi, e tutti i gruppi di ricerca vorrebbero usarlo
  - Se il cluster venisse assegnato ad un gruppo di ricerca per volta rimarrebbe inutilizzato per il 99% del tempo
  - Soluzione: dare a ogni gruppo di ricerca un cluster virtuale di 200 macchine virtuali, facendo migrare le macchine virtuali sulle macchine fisiche a seconda del carico del momento