Universitatea Politehnica din Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Master Ingineria Informației și a Sistemelor de Calcul

Sisteme de operare avansate

Mașini Virtuale – legătura cu arhitectura sistemului și importanța lor

Profesor Coordonator,
Conf. Dr. Ing. Stefan Stancescu

Masterand,
Theodor Cristian BUCULEI

2016 – 2017

Cuprins

1. Introducere
   1.1 Descrierea arhitecturii sistemului de calcul
   1.2 Necesitatea virtualizării
      1.2.1 Definirea noțiunii de mașină virtuală
2. Arhitectura unei masini virtuale
3. Principalele tipuri de mașini virtuale
   3.1 Mașini virtuale de process
      3.1.1 Multiprogramarea
      3.1.2 Emularea
      3.1.3 Optimizarea dinamică
      3.1.4 Mașini virtuale bazate pe limbaje de nivel înalt
   3.2 Masini virtuale de system
      3.2.1 Mașini virtuale clasice
      3.2.2 Mașini virtuale hostate
      3.2.3 Mașini virtuale ale întregului system
      3.2.4 Mașini virtuale co-proiectate
4. Tipuri de virtualizare a serverelor
   4.1 Virtualizarea întregului sistem – VMWare
      4.1.1 Implementarea unui VMWare
5. Aplicații ale mașinilor virtuale
   5.1 Loggerul de sistem – ReVirt
   5.2 Mate: mașini virtuale pentru rețele de senzori
6. Crearea unei mașini virtuale
7. Concluzii
8. Bibliografie

Mașini Virtuale – legătura cu arhitectura sistemului și importanța lor

1. *Introducere*

Conceptul de virtualizare poate fi aplicat nu numai pentru subsisteme, cum ar fi discurile, dar și asupra unei mașini. Pentru a implementa o mașină virtuală, dezvoltatorii au adăugat un nivel software mașinii reale pentru a suporta arhitectura dorită.[1]

Acestea oferă o abstractizare pentru sistemului fizic de bază al sistemului de operare oaspete care rulează pe el. În funcție de nivelul de abstractizare pe care un Virtual Machine Monitor (VMM)[1] îl oferă și dacă sistemele gazdă și oaspete utilizează aceeași ISA, putem să clasificăm mașinile virtuale în mai multe tipuri. Deoarece acestea oferă optimizările dorite cum ar fi flexibilitate software, protecție sporită și independență din punct de vedere hardware, pot fi folosite în arii de cercetare variate.[1]

1.1 *Descrierea arhitecturii sistemului de calcul*

Descrierea mașinilor virtuale este o discuție legată de arhitectura unui calculator în adevăratul sens al cuvântului. Deoarece implementarea unei mașini virtuale se bazează pe interfețe proiectate, o considerenta importantă în construcția unei VM este modul în care se implementează aceste interfețe.[1]

Arhitectura[1], relativ la sistemele informatice, se referă la o specificație formală a unei interfețe a sistemului, incluzând comportamentul logic al resurselor coordonate prin intermediul interfeței. Implementarea descrie adevărata comportare a interfeței.

Nivelele abstractizate corespund straturilor de implementare, fie hardware sau software, fiecare asociat cu propria interfață sau arhitectură.

Figura 1 arată câteva interfețe și straturi de implementare importante în cadrul unui sistem de calcul obișnuit.[1]

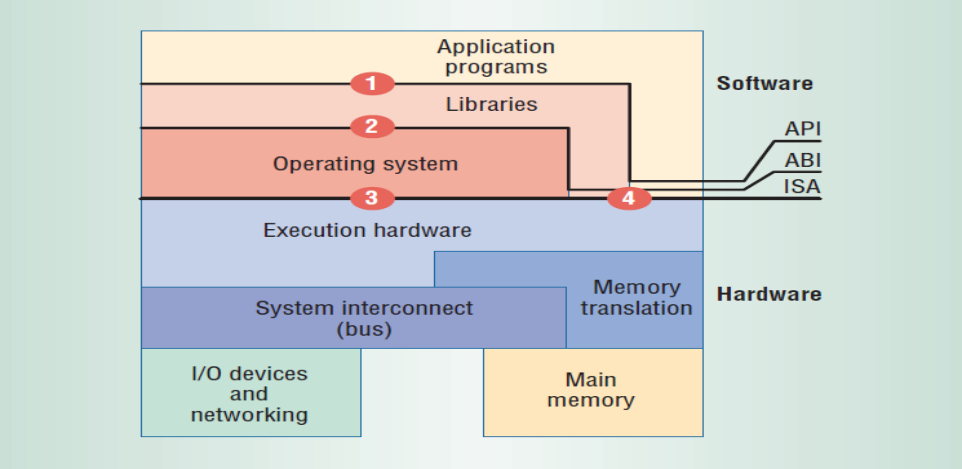

Figura 1 : Arhitectură unui sistem de calcul. Nivelele importante comunică vertical prin intermediul arhitecturii setului de instrucțiuni(ISA), interfeței binare (ABI) și a unei interfețe de programare a aplicațiilor (API).[3]

Trei dintre aceste interfețe situate pe sau în apropierea limitei HW/SW sunt extrem de importante pentru construcția mașinilor virtuale.[1]

ISA (Instruction Set Arhitecture)

Acesta arhitectură marchează clar delimitarea zonei hardware de cea software și constă în interfețele 3 și 4 din Figura 2.[1]

Interfața 4 reprezintă userul ISA și include acele aspecte vizibile pentru un program.[1]

Interfața 3 a sistemului ISA[1], este un superset al utilizatorului ISA și include doar aspectele vizibile pentru sistemul de operare responsabil pentru repartizarea resurselor hardware.[1]

ABI (Application binary interface)

Aceasta oferă un acces programat la resursele hardware și serviciile disponibile dintr-un sistem prin intermediul unui utilizator ISA (interfața 4) și interfeței de apel a sistemului (interfața 2).[1]

Nu include instrucțiunile sistemului, ci toate programele de aplicație interacționează indirect cu resursele hardware, invocând serviciiile sistemului de operare, prin intermediul interfeței de apel a sistemului.[1]

Apelurile de sistem pun la dispoziție o modalitate prin care sistemul de operare să realizeze operații în detrimentul unui utilizator de program după validarea autenticității și siguranței sale.[1]

API (Application programming interface)

Acesta oferă un program de acces la resurselor hardware și serviciile disponibile într-un sistem prin intermediul utilizatorului ISA (interfața 4) completate de apelurile unei librării cu un nivel de limbaj înalt (HLL).[1] Apelurile de sistem sunt în mod usual realizate prin librării. Folosind un API, aplicațiile software pot fi portate cu ușurință, prin recompilare, pe alte sisteme care suportă același API.[1]

1.2 Necesitatea virtualizării

Virtualizarea a devenit un instrument important în proiectarea unui sistem de calcul și mașinile virtuale sunt folosite într-o gamă mare de subdiscipline, variind de la sisteme de operare la limbaje de programare pentru diferitele arhitecturi de procesor.[1]

Eliberând dezvoltatorii și utilizatorii de problema interfețelor tradiționale și de cea a constrângerilor de resurse, mașinile virtuale îmbogățesc interoperabilitatea software și versatilitatea în funcție de platformă.[1]

Deoarece mașinile virtuale sunt produsul unor grupuri diverse cu obiective diferite, există doar câteva concepte comune.[1]

În ciuda complexității lor incredibile, sistemele de calcul există și continuă să evolueze, fiind concepute ca și ierarhii cu interfețe bine definite care separă nivelele de abstractizare.

Folosind aceste interfețe[1], dezvoltarea unor subsisteme independente este mult mai facilă, atât din punct de vedere software cât și hardware.[1]

Figura 2[1] descrie un exemplu de abstractizare aplicată unui disk. Sistemul de operare abstractizează detaliile de adresare ale hard disk-ului, spre exemplu, acesta este compus din partiții astfel încât disk-ul apare ca și un set de fișiere de mărime variabilă.[1] Programatorii de aplicații pot creea, scrie sau citi fișiere independent de construcția hard disk-ului și organizarea fizică.[1]

O arhitectură a setului de intructiuni (ISA) exemplifică avantajele interfețelor bine definite. Astfel de interfețe permit devoltarea unor subsisteme interactive.[1] Spre exemplu, dezvoltatorii Intel și AMD au dezvoltat microprocesoare care implementau această arhitectură pe 32 de biți (Intel IA -32), în timp ce dezvoltatorii Microsoft au scris software care este compilat pentru același set de instrucțiuni.[1]

Deorece ambele grupuri satisfac specificațiile ISA[1], software-ul poate fi folosit în mod corect pe orice calculator cu un microprocesor pe 32 de biți.

Din păcate, interfețele bine definite au de asemenea limitările lor.[1] Subsistemele și componenetele proiectate specificațiilor unei interfețe nu vor funcționa folosind altă interfață. Aceasta lipsă de interoperabilitate poate fi importantă, în mod special atunci când este avantajos să poți muta software la fel de ușor pe cât datele.[1]

Virtualizarea oferă o modalitate de a scăpa de această limitare. Virtualizand un sistem sau componentă, cum ar fi un procesor, memoria sau un dispozitiv de I/O, la un anumit nivel de abstractizare se pot ascunde interfața cât și resursele pe o interfață a unui sistem de bază, posibil diferit. [1]

În mod diferit față de abstractizare, virtualizarea nu simplifică sau ascunde detalii. Spre exemplu, în figura 2 b), virtualizarea transformă întregul disk în două disk-uri virtuale mai mici, fiecare având o structură internă.[1]

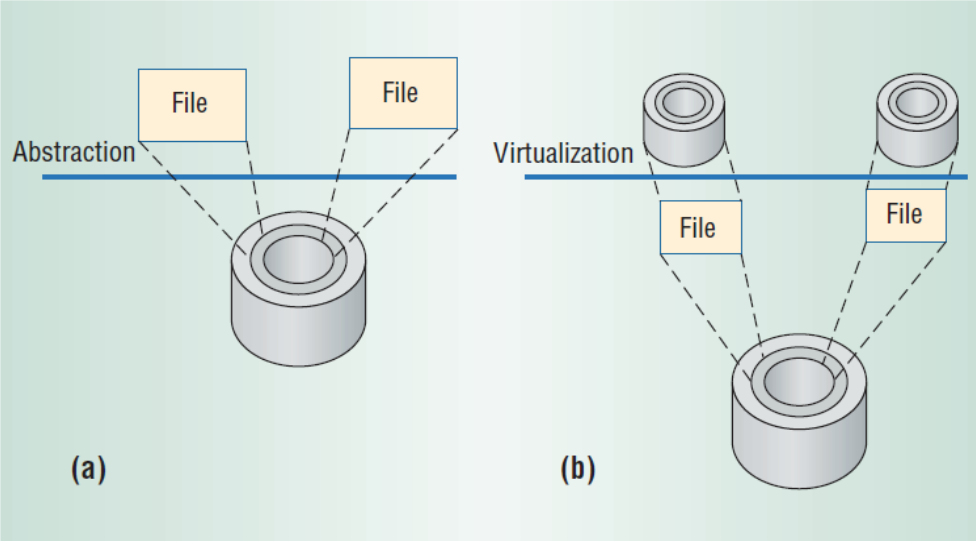

Figura 2: Abstractizarea și virtualizarea aplicate pe un spațiu de disk[1]
(a) Abstractizarea oferă o interfață simplificată resurselor de bază.
(b) Virtualizarea oferă o interfață diferită sau diferite resurse pe același nivel de abstractizare

Virtualizarea software folosește abstractizarea fișierelor ca o cale intermediară pentru a creea legătura între disk-ul real și cel virtual. Nivelul de detalii oferit de interfața disk-ului virtual (adresarea memoriei) nu este diferit față de cel pentru disk-ul real.[1]

1.2.1 Definirea noțiunii de mașină virtuală

Prin mașina virtuală se înțelege adăugarea unui nivel adițional unei platforme existente pentru a simula o platformă diferită sau, în alt caz, mai multe.[2]

Primele mașini virtuale au fost dezvoltate de IBM în anii 1960 și au devenit foarte populare în anii 70' . În acel moment, sistemele de calcul erau mari și expresive, astfel încât IBM a inventat conceptul de mașină virtuală ca o modalitate de a partiționa resursele mașinii între diferiți utilizatori. [2]

2. Arhitectura unei mașini virtuale

O mașină virtuală oferă o versiune izolată și protejată a sistemului fizic de bază. Este nevoie de un nou strat deasupra celui original pentru a abstractiza resursele fizice și a oferi o interfață pentru sistemul de operare care rulează. Acest nivel este numit Virtual Machine Monitor (VMM).[2]

VMM este partea esențială în cadrul implementării unei mașini virtuale, deoarece face translatarea dintre partea hardware și platforma virtualizată care stă la bază: oferind procesoare virtuale, memorie și dispozitive de intrare/ieșire virtualizate. Din moment ce toate mașinile virtuale folosesc aceeași parte hardware, VMM trebuie să ofere protecție portivita așa încât fiecare mașină virtuală este o replică izolată.[2]

Figura următoare descrie un model de mașină virtuală, unde VMM este situat între sistemul hardware și sistemul de operare. În mod obișnuit, platforma compusă din VMM și mașină, care oferă mediul mașinii virtuale, este numit gazdă (host), pe când sistemul de operare și aplicațiile care rulează pe el sunt numite oaspeți (guests). [2]

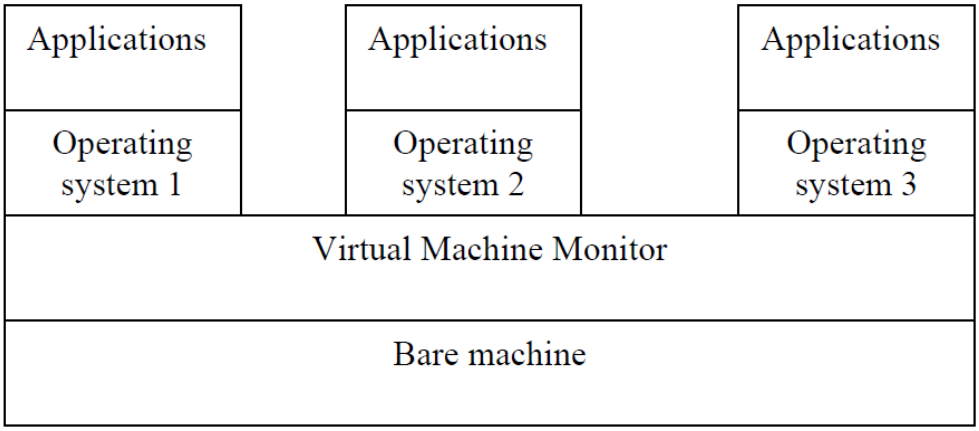

Figura 3 : Descrierea arhitecturii unei mașini virtuale simple[2]

De asemenea, interfețele abstracte pe care VMM le ofere pot fi diferite. Unele VMM realizează întreaga virtualizare a sistemului, ceea ce înseamnă că sistemul de operare oaspete nu are nevoie de schimbări pentru a rula pe un sistem hardware virtualizat, în timp ce alte VMM-uri nu pot vitualiza întregul sistemul și trebuie avut în vedere că este necesară schimbarea unor părți de cod pe sistemul de operare guest pentru a-l face potrivit pentru interfața abstractă.[2]

Robert Goldberg a rezumat documentarea cercetărilor asupra mașinilor virtuale din anii 60'-70' și de asemenea a enunțat principiile de implementare ale unei mașini virtuale.[2]

În lucrările lui, cel mai important scop al mașinilor virtual este de a rezolva portablitatea software, posibilitatea de a rula teste și a diagnostica programele.[2]

Deoarece arhitectura calculatoarelor din generația a 3-a nu puteau fi virtualizate direct, a fost concepută o modalitate software foarte dificilă. Unii cercetători au propus ca rezolvare a problemei arhitecturile virtualizabile care să suporte direct mașini virtuale, incluzând virtualizatorul hardware al lui Goldberg.[2]

În prezent, implementarea unei mașini virtuale necesită mult efort.[2]

3. Principalele tipuri de mașini virtuale

În prezent, există multe tipuri de mașini virtuale, acoperind multe zone de cercetare. În toate acestea, VMM este așezat diferit, având roluri diferite.[2]

J. E. Smith și Ravi Nair au propus o taxonomie sistematică a mașinilor virtuale pentru le clasifica și au realizat o diagramă care poate face distincția precisă între diferite tipuri.[2]

Din moment ce o mașină virtuală este un strat care abstractizează toate celelalte straturi de sub el și oferă o interfață stratului de deasupra lui, nivel pe care mașina face abstractizarea, acesta poate fi un bun criteriu de clasificare.[2]

Sunt două perspective din care putem privi o mașină virtuală. Există mașini de process și mașini virtuale de sistem.[2]

Din perspectiva unui proces, mașina este spațiul de adresare asignat memoriei, intrucțiunile și nivelul de regiștri ai utilizatorului. Un process nu are acces direct către disk , nici către alt mediu de stocare secundar sau resurse de I/O.[2] Poate accesa aceste resurse prin apeluri de sistem. În ceea ce privește întregul sistem, mașina virtuală oferă un mediu care suportă simultan procese multiple și alocă memorie fizică și resurse de intrare/ieșire proceselor.[2]

Așadar, bazat pe nivelul de abstactizare, avem mașini virtuale de process și mașini virtuale de sistem. Așa cum indică și numele, o mașină virtuală de process[2] poate suporta numai un process individual, pe când o mașină de sistem suportă mediul și un sistem de operare complet.

Pentru a stăpâni aceste două tipuri de mașini virtuale, detalierea a două interfețe standardizate este utilă: **ISA** și **ABI (Application Binary Interface).**[2]

Așa cum am mai spus, ISA este o parte a procesorului vizibilă programatorului și care include atât instrucțiunile utilizatorului cât și pe cele ale sistemului.[2]

ABI[2] include întregul set de instrucțiuni ale utilizatorului, precum și interfețele de sistem prin care aplicațiile pot accesa resursele hardware. Cu alte cuvinte, ABI separă procesele de restul sistemului,pe când interfața ISA separă partea hardware de restul.[2]

Având în vedere definițiile celor 2 interfețe, putem afirma că mașinile virtuale de proces oferă o interfață de tip ABI aplicațiilor iar cele de sistem oferă o interfață de tip ISA sistemului de operare și aplicațiilor care rulează pe el.[2]

În funcție de modul în care acestea suportă una din cele 2 interfețe și având în vedere dacă sistemele guest și host sunt de același tip (ISA), putem face o clasificare a mașinilor virtual în diferite tipuri.[2]

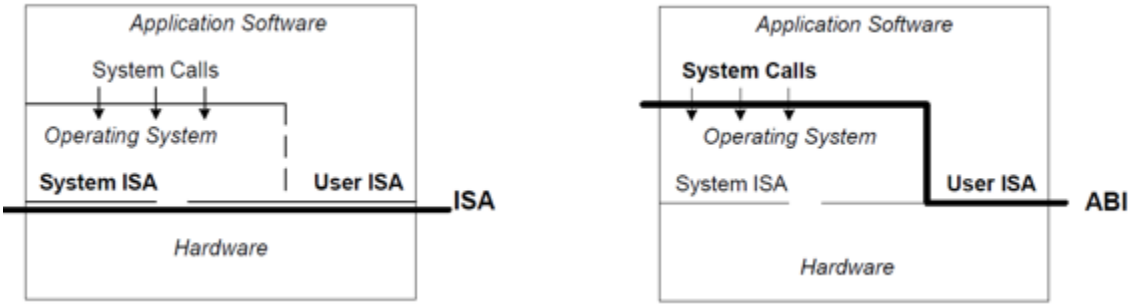

Figura 4 : Interfețe de system[3]

a) interfața ISA (Instruction Set Architecture) b) interfața ABI (Application Binary Interface)

3.1. Mașini virtuale de process

Acest tip de mașini virtuale oferă utilizatorului un mediu care dispune de o interfață ABI virtuală. În diferitele lor forme, mașinile virtuale de proces pot realiza emularea, replicarea și optimizarea.[2]

3.1.1. Multiprogramarea

Multiprogramarea[2] este o caracteristică standard în sistemele de operare moderne. Sistemul de operare oferă o interfață ABI reprodusă pentru fiecare proces iar fiecare proces se comportă ca și cum ar deține întreaga mașină. Astfel, aplicațiile executate în mod concurent rulează pe mașină virtuală de proces.[2]

În acest caz, host-ul și guest-ul se află în aceași interfață ISA și același sistem de operare.[2]

3.1.2. Emularea

Cea dea doua categorie de mașini virtuale de proces, sunt acelea care rulează programe binare compilate pe o sursă ISA, în timp ce partea hardware de bază este o altă interfață ISA.[2]

Mașina virtuală va trebui să emuleze execuția sursei ISA și cea mai simplă cale de a face asta este ca VMM să interpreteze fiecare instrucțiune a sursei prin executarea unor instrucțiuni ISA native. Metodă are performanțe slabe.[2] Așadar, translatarea binară este mai

frecvent utilizată, ce convertește instrucțiunile sursei în instrucțiuni native cu funcții echivalente iar după traslatarea blocului de instrucțiuni,acesta poate fi reutilizat în mod repetat.[2]

3.1.3. Optimizarea dinamică

În exemplele de mașini virtuale de mai sus, sursa și tinta ISA sunt diferite, astfel încât scopul mașinii virtual este să emuleze execuția. Unele tipuri sunt cu aceeași sursă și țintă de tip ISA și rolul acestora este să realizeze optimizări pe proces. Implementarea mașinilor virtuale pentru optimizări dinamice este foarte similară cu cea pentru emulare.[2]

3.1.4. Mașini virtuale bazate pe limbaje de nivel înalt

Ultimul tip de mașini virtuale de proces este cel mai frecvent recunoscut în mare parte datorită popularității Java. Scopul primelor 3 mașini virtuale, exceptând optimizatorul dinamic, este de a îmbunătății portabilitatea platformelor.[2] Dar implementarea lor are nevoie de efort pentru fiecare ISA, astfel că o altă modalitate este a traslata virtualizarea către un nivel mai înalt.

Două exemple foarte bune sunt mașinile Pascal și Java. Într-un sistem convențional, programele HLL sunt compilate în coduri abstracte și apoi generate în cod obiect specific interfeței sau sistemului de operare de către un generator de cod.[2]

Însă, pentru aceste mașini (Java/Pascal) codul care urmează să fie distribuit nu este codul obiect, ci codurile intermediare : cod P pentru Pascal și bytecode pentru Java. Pe fiecare sistem de operare sau interfață ISA există o mașină virtuală pentru a interpreta codurile intermediare pentru instrucțiunile host-ului specific platformei. Așadar acest tip de mașină virtuală de proces oferă independență de platformă . [2]

3.2. Mașini virtuale de sistem

Sunt în mod normal recunocute folosind termenul de mașini virtuale.[2]

3.2.1. Mașini virtuale clasice

Mașinile virtuale clasice sunt modelul original pentru mașinile de sistem. VMM[2] este așezat peste partea hardware și oferă replică hardware și managementul resurselor.[2] În cele mai multe cazuri, acest model oferă eficientă, însă VMM trebuie să se ocupe de driverele device-urilor și uilizatorii trebuie să instaleze VMM și guest-ul după ștergerea sistemului existent.[2]

3.2.2. Mașini virtuale hostate

Acestea construiesc un VMM pe un sistem de operare de tip host existent. De cele mai multe ori, implementarea acestor tipuri de mașini virtuale este mai puțin eficientă, datorită nivelului software adițional.[2]

Pentru ambele tipuri de mașini, cele clasice și cele hostate, interfața ISA a sistemului de operare de tip guest este aceeași ca și partea hardware pe care se bazează.[2]

3.2.3. *Mașini virtuale ale întregului sistem*

Există cazuri în care avem nevoie să rulăm sisteme de operare și aplicații pe o aceeași interfață ISA. În aceste cazuri, datorită interfețelor ISA[2] diferite, este necesară o emulare și traslatare completă a aplicației și intergului sistem de operare, astfel încât acestea se numesc mașini virtuale ale întregului sistem.[2]

3.2.4. *Mașini virtuale co-proiectate*

Cele 3 mașini virtuale descrise mai sus au o interfață ISA bine definită. Mașinile virtuale co-proiectate au ca scop îmbunătățirea performanței sau eficientizarea non-standardelor interfeței.[2]

Nu există implementări ISA native astfel încât nu este posibilă o execuție nativă.[2]

În mod uzual, VMM[2] utilizează un traslator binar pentru a converti instrucțiunile guest-ului în instrucțiuni native ISA. VMM funcționează ca o parte din implementarea hardware pentru a oferi sistemului de operare găzduit și aplicațiilor o platformă de mașini virtuală exact ca cea a unei platforme hardware native. Interfața ISA nativă este în totalitate ascunsă de către sisitemul de operare găzduit și de către software.[2]

Mai jos este prezentată o clasificare a mașinilor virtuale, cuprinzând toate categoriile de mai sus, bazate pe tipul de nivel al mașinii virtuale- ABI sau ISA –si evidențiind dacă host-ul și guest-ul mașinii virtuale au o aceeași ISA.[2]

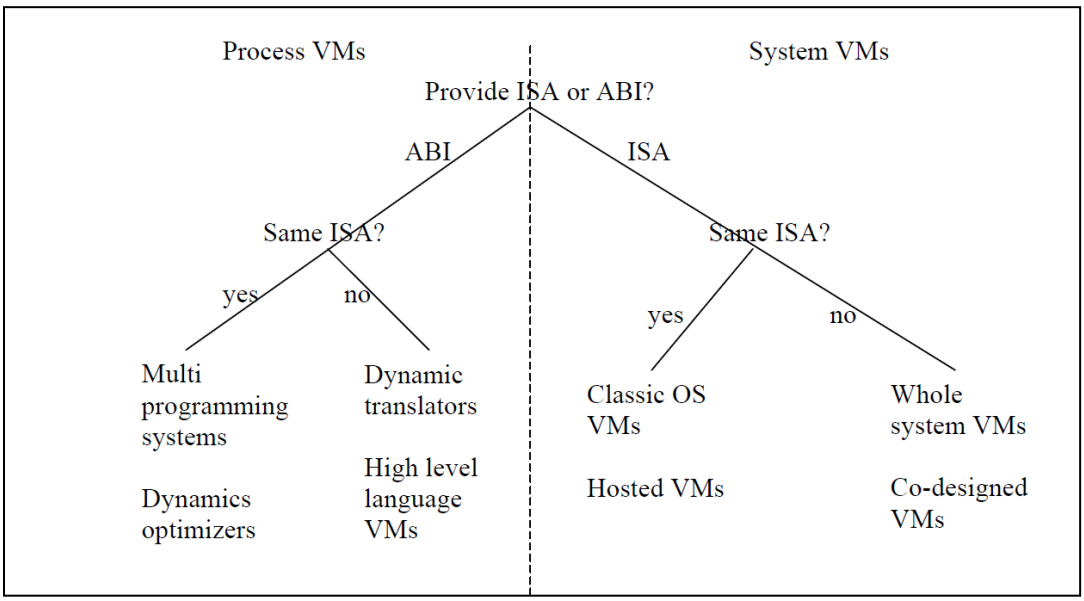

Figura 5 : Taxonomie a arhitecturii mașinilor virtuale[2]

4. Tipuri de virtualizare a serverelor

4.1. Virtualizarea întregului sistem – VMWare

Există diferite modalități de a implementa o mașină virtuală. Dintre acestea metode cele mai importante pot fi clasificată în 2 categorii: virtualizarea completă și paravirtualizarea.[2]

Virtualizarea completă este realizată astfel încât să ofere o formă nemodificată de abstractizare pentru sistemul fizic de bază și crează un sistem virtual complet în care sistemul de operare guest poate executa.[2] Scopul virtualizării complete este ca sistemul de operare găzduit și aplicațiile care rulează pe el să nu aibă schimbări atunci când sunt migrate pe mașina virtuală.[2]

Abstractizarea oferită de virtualizarea completă trebuie să fie aceeași că a părții hardware fizice astfel încât sistemul de operare găzduit și aplicațiile nu sunt conștiente de faptul că rulează pe o mașină virtuală și nu pe o mașină fizică.[2]

Astfel, unul dintre cele mai mari avantaje a virtualizării complete este că se poate porta orice sistem de operare existent pentru un sistem dat având în vedere o virtualizare completă a unei mașini virtuale fără cost adițional.[2]

Spre exemplu, dacă avem un VMWare[2] ca server, se pot rula fără probleme sisteme de operare și aplicații pe 32 de biți.

Însă, datorită cerințelor restrictive ale unui sistem fizic de bază complet, virtualizarea completă îndeplinește performanțe slabe.[2]

4.1.1. Implementarea unui VMWare

Există două tipuri de virtualizare completă în cazul soluțiilor de tip VMWare : **arhitectura hostata** și **cea bazată pe hipervizor**.[2]

Ambele sunt create pentru o arhitectură pe 32 de biți și suportă sisteme de operare cum ar fi Windows 2000, XP și Linux Redhat.[2] Mașina VMWare folosește abordarea hostata în care sistemul de operare guest este instalat și rulează pe nivelul cel mai de sus al sistemului de operare. Astfel, ea folosește sistemul de operare oaspete pentru a suporta diversele dispozitive hardware.[2]

Arhitectura hipervizată, în contrast cu cea prezentată mă sus, instalează un strat software numit hipervizor, exact deasupra nivelului hardware iar sistemul de operare guest rulează deasupra acestui nivel.[2]

Ca și exemplu reprezentativ al arhitecturii hipervizate este **VMWare ESX Server**.[2]

Abordarea hostată – VMWare Workstation

Există două mari avantaje în implementarea unei arhitecturi de acest tip:

- arhitectura deschisă a calculatoarelor, rezultată printr-o diversitate mare de dispozitive hardware care au nevoie de o mașină virtuală pentru a fi controlate.[2]

Această abordare poate extinde driverele dispozitivelor existente din sistemul de operare nemaifiind necesară portarea sutelor de drivere către driverele virtuale.[2]

- cea mai mare parte a utilizatorilor de calculator au un număr mare de aplicații software instalate și configurate adecvat în sistemul de operare existent.[2] Nu vor dori să renunțe la acestea atunci

când folosesc mașină virtuală. Exact această abordare, a mașinilor virtuale hostate, permite coexistența sistemului de operare și software-ului original alături de cele ale mașinii virtuale.[2]

Însă dezavantajele acestei implementări sunt vizibile. Deoarece sistemul de operare hostat are acces complet la partea hardware, chiar dacă VMM[2] are privilegii hardware și de sistem complete, nu poate realiza o programare cu drepturi depline.

Spre exemplu, mașina virtuală nu poate garanta pentru un anumit schimb al unității centrale de prelucrare, deoarece însuși VMM este programat de sistemul de operare gazdă.[2]
Pentru avea însă performanțe acceptabile, sistemul de operare găzduit trebuie să ruleze exact deasupra nivelului hardware fizic.[2] Astfel încât contextul traslatarii între sistemul de operare oaspete(mașină virtuală) și sistemul de operare gazdă este mai costisitor decât procesul în sine.[2]
Performanța dispozitivelor de intrare/ieșire devine astfel o problemă, deoarece operațiile de intrare/ieșire realizate de sistemul de operare oaspete trebuiesc să transmită driverelor dispozitivelor din sistemul de operare gazdă și contextul traslatarii este inevitabil prezent.[2]

Figura următoare ilustrează structura unui sistem de operare oaspete dintr-o mașină virtuală pe o arhitectură gazdă în cazul unei soluții de tip VMWare Workstation.[2]

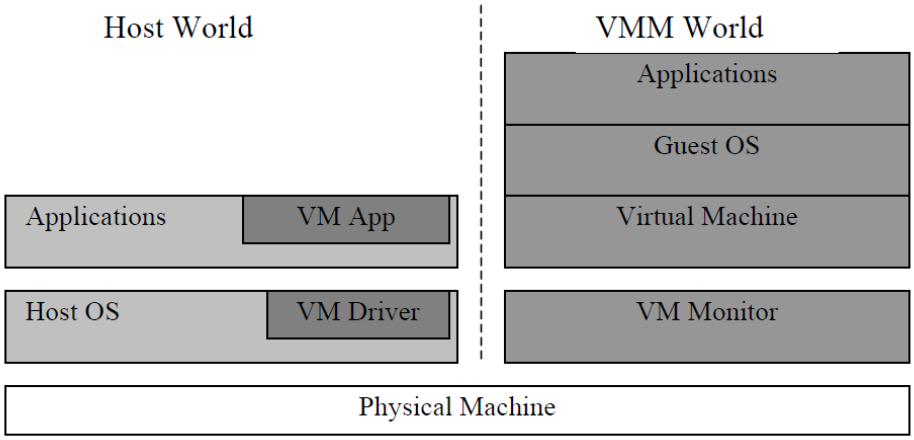

Figura 6 : Arhitectura mașinii VMWare Workstation[2]

Atunci când rulează, porțiunea dedicată aplicațiilor (VMApp)[2] utilizează un driver (VMDriver) încărcat în sistemul de operare gazdă pentru a creea o componenta de tip VMM privilegiată. Această componentă este integrată în kernelul sistemului de operare guest și rulează direct pe hardware-ul fizic. Procesorul face traslatarea între arhitectura sistemului de operare gazdă și cea a mașinii virtuale.[2]
Atât sistemul de operare cât și aplicațiile de tip guest rulează folosind privilegii de utilizator.[2]
Execuția task-urilor este o combinație între execuția directă și traslatarea binară.[2]
Marea majoritate a instrucțiunilor non-privileagiate rulează folosind direct hardware-ul fizic.
Instrucțiunile privilegiate sunt translatate către o altă secvența de intructiuni la rulare.[2]
Secvența translatată va asigura încapsularea și emularea aceluiași efect când vine vorba de executarea instrucțiunilor privilegiate în cadrul mașinii virtuale.[2]

Când sistemul de operare găzduit realizează operații de intrare/ieșire, acestea vor fi interceptate de VMM.[2]

Evitând accesarea hardware-ului fizic în mod direct, VMM va translata către sistemul de operare gazdă și va face apelul către VMApp[2] pentru a realize operația de intrare/ieșire folosind sistemul normal al apelurilor în detrimental mașinilor virtuale.

VMM poate prelua controlul sistemului de operare gazdă, la nevoie, astfel încât cel găzduit să poată trata apelurile întrerupte trimise de către hardware la finalul task-ului raspectiv.[2]

VMApp va comunica interceptând request-urile și răspunsurile de la și către mașina virtuală și sistemul de operare gazdă.[2]

Astfel, VMM va face abstactie de dispozitivele fizice de intrare/ieșire.[2]

Virtualizarea memoriei este un alt concept ce are legătură cu virtualizarea completă.[2]

Abordarea hipervizată – VMWare ESX Server

În mod diferit față de VMWare Workstation care este construită pe sistemul de operare existent, WMWare ESX Server rulează direct pe hardware-ul fizic.[2]

În ceea ce privește virtualizarea unității centrale de prelucrare,această abordare folosește aceeași tehnică ca și cealaltă: execuția directă însoțită de traslatarea binară dinamică.[2]

Tehnica este explicată în continuare:[2]

- noțiunea de mașină virtuală lucrează cu trei tipuri de adrese: adresele virtual care sunt adresele vizibile aplicației în cadrul sistemului de operare găzduit; adresele fizice care reprezintă spațiul de adrese liniar abstractizat de către VMM[2] și cu care lucrează sistemul de operare găzduit; adresele mașinii care reprezintă spațiul adreselor memoriei hardware accesate de către procesorul fizic.

Sistemul de operare găzduit crede că rulează direct pe hardware, care are spațiul de adrese liniar, începând de la 0. Pentru a realiza acest lucru, VMWare ESX Server[2] menține o mapare a structurii datelor de tip m pentru fiecare mașină virtuală pentru a face legătura între adresele fizice și adresele mașinii.

În locul accesării tabelei de paginare a sistemului de operare găzduit, procesul accesează o tabela de adresare ascunsă, care conține traslatarea adreselor virtuale în adrese ale mașinii.

ESX Server este responsabilă în actualizarea atât a maparii p cât și a tabelei ascunse pentru a le păstra sincronizate.[2]

Cel mai mare avantaj al acestei tabele de paginare ascunse este faptul că accesarea memoriei în sistemul de operare găzduit poate fi astfel executată direct în procesorul nativ deoarece traslatarea de la adresele virtuale către cele ale mașinii se face în buffer-ul de translatare (translation lookaside buffer-TLB). [2]

Supraîncărcarea este considerată unul dintre cele mai importante avantaje ale utilizării mașinilor virtuale.[2]

Conform acesteia, dimensiunea totală a memoriei configurate pentru toate mașinile virtuale depășește dimensiunea totală a memoriei fizice.[2]

Paginile de memorie pot fi deplasate către cele ale mașinii virtuale, potrivit configurării.[2]

Astfel, oferă o utilizare eficientă a resurselor de momorie limitate deoarece în marea majoritate a timpului, diferitele sisteme de operare găzduite vor avea nivele de cereri de memorie diferite.[2]

Performanța generală este îmbunătățită atunci când mai multă memorie îi este alocată sistemului de operare găzduit cu cea mai mare cerere.[2]

Problemă care poate să apăra este cum se pot găsi aceste pagini reclamate. Serverul ESX[2] decide să ofere sistemului de operare găzduit posibilitatea de a lua decizia bazată pe faptul că cele mai relevante informații despre cele mai utilizate pagini sunt cunoscute numai de sistemul de operare oaspete.

Un driver de dispozitiv de tip balon este instalat pe fiecare sistem de operare de tip guest.[2]

Atunci când serverul ESX are nevoie de memorie de la unul din sistemele de operare, interoghează acest driver de tip balon care trimite cereri către sistemul de operare găzduit.[2]

Bazat pe propriul algoritm de înlocuire, sistemul de operare guest va specifica cele mai utilizate pagini de memorie disk-ului virtual și paginile obținute de către driverul de tip balon vor fi trimise către serverul ESX, care va actualiza tabela de paginare ascunsă să pointeze către un alt sistem de operare găzduit ce are o mai mare cerere de memorie.[2]

Dispozitivul de tip balon va returna paginile către sistemul de operare. Astfel, sistemul de operare găzduit, menționat ultima dată, va avea mai multe pagini libere.[2]

Partajarea memoriei bazate pe conținut transparent

Tabela de paginare transparentă este o modalitate ușoară de a partaja memorie între diferitele mașini virtuale. Numerele mai multor pagini virtuale pot fi mapate pe un singur număr de pe mașină.[2]

Serverul ESX[2] utilizează o tehnică de partajare a memoriei într-un mod transparent. Prin trasparent se înțelege că mașinile virtuale nu știu de aceste pagini partajate iar toate acestea seamănă cu paginile private. Partajarea paginilor cu conținut bazat reprezintă faptul că VMm va partaja toate paginile care au același conținut. În mod evident, compararea fiecărei pagini cu o altă are o complexitate de ordinal n2.[2]

Serverul ESX ulitizeaza funcții hash pentru a reduce comparația întreagă a paginilor.

O funcție hash este executată pe fiecare pagină read-only, întâi pentru a sumariza conținutul paginii, apoi valoarea hash rezultată este folosită ca și cheie și introdusă într-o tabelă de tip hash globală.[2]

De fiecare dată când o pagină read-only este cerută, valoarea hash a acestei pagini este de asemenea calculată și actualizată în tabela globală.[2] Dacă această valoare există déjà în tabelă, se efectuează o comparație completă între noua pagină și setul de pagini înregistrate sub valoarea hash obținută.[2]

Dacă acestea coincid, pagina partajată este identificată, tabela de paginare trasparentă este actualizată să mapeze pagina partajată și de asemenea pagina transmisă este marcată ca și copy-on-write.[2]

Dacă paginile nu coincid, noua pagină va fi adăugată paginii setată în tabela globală.[2]

Chiar dacă acest concept este mai costisitor, se pot identifica pagini identice care nu pot fi detectate prin abordarea tradițională, eliberând astfel spațiul de memorie și datele operațiilor.[2]

Paravirtualizarea

Spre deosebire de **virtualizarea completă**, **paravirtualizarea** nu urmărește o abstractizare identică a sistemului fizic de bază pentru a diminua costurile de performanță care nu sunt necesare.[3]

Oferă fiecărei mașini virtuale o abstractizare care poate fi implementată eficient folosind hardware-ul dat. Deoarece această abstractizare este diferită de interfața hardware originală, sistemul de operare oaspete trebuie să fie modificat astfel încât să ruleze pe mașina virtuală.

În funcție de implementare, interfața ABI poate rămâne nemodificată și se pot rula aplicații pe sistemul de operare găzduit fără alte modificări.[3]

Recent, o atenție sporită este îndreptată către această metodă deoarece poate micșora costul de performanță spre un nivel cât mai mic și obține scoruri de performanță foarte apropiate ale sistemului de operare native.[2]

Una dintre mașinile care se bucură de success este **Xen**[2], dezvoltată de un grup de cercetători ai Universității din Cambridge. Unele centre industrial au început deja să ofere clienților lor servere bazate pe mașina virtual Xen pentru un prêt mai mic, însă performanțe similare.[2]

Procesorul mașinii Xen este unul pe 32 de biți, cea mai utilizată arhitectură din lume, dar de asemenea destul de scumpă pentru a implementa o virtualizare completă.[2]

Buferul de translatare este coordonat de partea hardware în locul celei software și nu este etichetat cu adresa descriptorului de spațiu.[2]

Pentru a evita plata altor costuri de performanță, Xen a ales să afișeze mașinii virtuale o nouă interfața și să modifice sistemul de operare gazdă, XenoLinux astfel încât să adopte această nouă interfață.[2]

Principiile de implementare ale mașinii Xen[2] sunt:

- suport pentru aplicații binare nemodificate: Chiar dacă sunt necesare anumite modificări ale sistemului de operare găzduit, interfața existentă standard trebue să fie aceeași.[2]
- suport pentru multi-aplicații complete ale sistemului de operare: Sistemul XenoLinux modificat poate rula multiple aplicații Linux standard complexe în mod concurent pe aceeași instanță a sistemului de operare găzduit.[2]
- paravirtualizarea[2]: Pentru a obține o izolare puternică a resurselor și performanță ridicată în același timp este destul de dificil, mai ales în ceea ce privește arhitectura pe 32 de biți. Paravirtualizarea este necesară pentru a depăși aceste dificultăți.

Nu ascunde complet resursele reale de sistemul de operare găzduit.[2] Această decizie este luată atât din punctual de vedere al performanței cât și al corectitudinii mașinii virtuale. În unele situații este indicat ca sistemul de operare găzduit să aibă acces atât la resursele virtuale cât și cele reale.[2]

Dispozitivele I/O

Pentru a oferi izolarea, demultiplexarea și o performanța înaltă în același timp, designerii Xen au trebuit să rezolve două aspecte importante: managementul resurselor și notificările legate de evenimente. [2]

Pentru prima provocare, transerul datelor dintre Xen[2] și sistemul de operare host, este implementat eficient și exploatând puterea memoriei virtuale, cum ar fi memoria partajată. Atunci când este nevoie de o operație de intrare/ieșire, sistemul de operare guest va creea un buffer din propria sa memorie rezervată.[2] Buferul va fi pasat către Xen prin hiper request-uri sincrone și Xen va valida inițial acest request, sper exemplu, va verifica dacă bufferul este în memoria rezervată a sistemului se operare guest, după care mașina Xen va remapa pagina pentru a fi accesibilă doar de către ea.[2] Operația DMA va fi executată de către controlerul DMA. Atunci când trasmiterea datelor se termină, un semnal de întrerupere va fi trimis de la device către Xen. După primirea acestui semnal, Xen va remapa buferul înapoi către spațiul de adresare al sistemul de operare guest.[2]

În ceea ce privește cea de-a doua provocare, a notificărilor legate de evenimente, Xen suportă un mecanism de transmitere a evenimentelor destul de simplu, care propagă notificările asincrone către guest.[2] Aceste notificări sunt făcute updatând un bitmap cu evenimentele în așteptare din kernelul sistemului de operare guest și opțional, Xen va chema evenimentul de callback înregistrat de către guest.[2]

Xen crează un descriptor I/O pentru fiecare guest care să se ocupe de request-urile și răspunsurile asincrone.[2] Descriptorul este creat în memoria rezervată a sistemului de operare guest, însă este accesibil din Xen.[2] El are o pereche de pointeri de creare și terminare a request-ului și o pereche de pointeri de creare și terminare a răspunsului.[2] Sistemul guest este responsabil de translatarea pointerului de creare a request-ului atunci când când se inițiază un nou request I/O și translatarea pointerului de terminare a răspunsului atunci când termină de procesat răspunsul.[2] Pe de altă parte, Xen va trimite pointerul de consumare a request-ului atunci când va începe să proceseze request-ul și va translata pointerul de creare a răspunsului atunci când va trimite răspunsul I/O înapoi.[2]

5. Aplicații ale mașinilor virtuale

În zilele noastre, sistemele informatice devin din ce în ce mai ieftine, astfel încât scopul principal al mașinilor virtuale de a partaja un sistem informatic scump nu mai este atât de important.[2]

Peter Chen și Brian Noble au propus ca sistemul de operare curent și structura software să fie înlocuită cu o nouă mașină virtuală cu un sistem de operare având o structură pe trei nivele și au argumentat că această structură este foarte utilă pentru anumite sisteme de cercetare cum ar fi logarea securizată, prevenirea și detectarea pătrunderii și migrarea mediului.[2]

Câteva dintre aceste aplicații vor fi prezentate în continuare:[2]

5.1. Loggerul de sistem – ReVirt

Logarea în sistem oferă primele informații pentru analizarea intruziunilor dupa ce au avut loc atacuri asupra sistemului. Însă, fiabilitatea nivelului de logare al sistemului de operare depinde de sistemul de operare în sine.[2]

Odată ce personajul negativ obține drepturi administrative, primul lucru pe care va încerca să-l facă este să-și ascundă urmele modificând sistemul de loguri utlizand aceste privilegii. Așadar, dacă un log urmează să fie folosit pentru o analiză a atacurilor, trebuie să fie protejat chiar și atunci când kernelul[2] este compromis. Cu alte cuvinte, sistemul de logare trebuie să fie realizat cu un nivel de privilegii chiar mai mare decât al kernelului. Un sistem de operare normal rulează pe hardware-ul de bază unde nu există un asemenea nivel de privilegii, ceea ce înseamnă că dacă se reușește pătrunderea în kernel, este foarte greu ca sistemul de logare să fie protejat de modificări. Însă dacă sistemul de operare rulează pe un monitor al mașinii virtuale, acesta din urmă ar putea fi locul perfect de realizare a task-ului sistemului de logare ca și când ar rula pe un nivel cu privilegii mai mari chiar și decât ale sistemului de operare găzduit.[2]

ReVirt este un asemenea sistem de logare ce rulează pe un VMM.[2] Loghează toate evenimentele non-deterministe dup ace sistemul este bootat. Având toate informațiile, un administrator de sistem poate relua întreaga istorie de execuție și află cum a fost spart și ce pagube au fost cauzate.[2]

ReVirt rulează pe UMLinux, un VMM care rulează un sistem de operare găzduit că și process singular pe un sistem gazdă.[2]

Toate apelurile și întreruperile de sistem ale celui găzduit sunt mapate către semnale variate din cel gazdă.[2]

VMM interceptează aceste semnale, loghează evenimentele corespunzătoare și apoi le transmite sistemului de operare găzduit. Atunci când răspunde, ReVirt[2] starteaza sistemul pornind de la starea inițial cunoscută și injectează evenimentele non-deterministe în momentul oportun pentru a dirija sistemul să evolueze exact în același mod ca și în rulările precedente.[2]

ReVirt rulează pe nivelul VMM și este aproape transparent sistemului de operare găzduit, exceptând faptul că una din instrucțiunile non-deterministe din sistemul de operare găzduit trebuie să fie înlocuită cu un apel de sistem. [2]

Deoarece VMM transmite toate interacțiile sistemului dintre sistemul de operare găzduit și cel gazdă, logarea în VMM este o modalitate ușoară de a înregistra suficiente informații pentru un răspuns complet.[2]

5.2. Mate: maşini virtuale pentru reţele de senzori

În mod diferit față de mașinile virtuale de sistem care încearcă să ofere o abstractizare peste partea hardware, mașinile virtuale de nivel înalt încearcă să ofere o interfață mai simplă și mai sigură nivelului de limbaj de programare.[2] Java și C# sunt cele două mari nume care fac parte din această categorie. Pe aceeași linie, există alte câteva proiecte de cercetare care dezvoltă aceeași idée în câteva medii informatice diferite.[2]

Mate, un interpretor de limbaj pentru o nouă interfață ISA abstractă, este realizat să îndeplinească cerințele special ale mediului de rețele de senzori.[2]

O rețea de senzori este compusă dintr-un dispozitiv informatics simplificat care are propriul processor, memorie ROM și RAM și comunică prin conexiune wireless.[2]

Pentru o rețea de senzori, energia este cea mai importantă resursă deoarece dispozitivul nu este reincrcabil odată instalat.[2] Iar energia necesară transmiterii unui singur byte prin wireless poate rula sute de instrucțiuni. Singura cale de a reprograma un astfel de dispozitiv este prin transfer wireless, prin intermediul unui parametru schimbat la instalarea unei noi binare.[2]

Provocarea este de a oferi capacitate de reprogramare cu un minim de rețea de transmisie.[2]

6. Virtualizarea resurselor I/O

În sistemele informatice, prin multele lor niveluri de abstractizare, noțiunea de mașină este o problemă de perspectivă.[3]

Din perspectiva unui proces, o mașină constă în spațiul de adresare al memoriei care a fost asignat procesului, alături de regiștrii nivelului userului și instrucțiunile care permit execuția codului aparținând procesului. Sistemul de intrare/ieșire, perceput de către proces, este mai degrabă abstract.[3] Diskurile și alte spații de stocare secundare apar ca și o colecție de fișiere pentru care procesul a accesat permisiuni. În mediul desktop, procesul poate interacționa cu userul printr-o fereastră (sau ferestre) pe care o creează printr-o interfață grafică.[3] Singura modalitate prin care procesul poate interacționa cu sistemul I/O al mașinii sale este prin apelurile sistemului de operare, fie direct, sau prin intermediul unor librării corespunzătoare procesului. Procesele sunt adesea tranzitorii (însă nu întotdeauna).[3] Ele sunt create, executate pentru o perioadă de timp și într-un final se încheie.[3]

Dintr-o perspectivă de nivel înalt, întregul sistem este suportat de o mașină existentă.[3] Mediul sistemului persistă de-a lungul timpului cu ocazionale reboot-uri atât timp cât procesele sunt create sau se termină.[3] Sistemul aloca memorie fizică și resurse I/O proceselor și permite acestora să interacționeze cu resursele lor prin intermediul sistemului de operare care este parte din sistem.[3]

7. Concluzii

Conceptul de mașină virtuală nu este nou. În anii 60' IBM a dezvoltat pentru prima oară o mașină virtuală pentru a partaja resursele mașinii utilizatorilor. Acesta a fost intotdeaua un subiect de cercetare interesant și recent se bucură de o popularitate destul de mare.

Partea esențială în cadrul unei mașini virtuale este VMM – (Virtual Machine Monitor) . El abstractizează resursele hardware-ului care stă la bază și oferă o replică protejată și izolată a sistemului fizic.

Există tipuri diferite de mașini virtuale în diferite zone de cercetare. În funcție de tipul interfeței pe care VMM o oferă, ABI sau ISA, putem distinge mașini virtuale de process și de sistem.

Pe lângă acestea, dacă sistemele de tip guest și host au același tip de interfață, putem clasifica mașinile virtuale în mai multe tipuri, stabilind o taxonomie completă.

În implementarea lor există două abordări importante: paravirtualizarea și virtualizarea completă.

Virtualizarea completă este folosită pentru a oferi o abstractizare identică a sistemului fizic astfel încât sistemul de operare guest și aplicațiile care rulează pe acesta nu au nevoie de modificări pentru a fi migrate pe mașina virtuală.

Paravirtualizarea, oferă o abstractizare care poate fi implementată eficient pe un anumit ansamblu hardware. Asta înseamnă că sistemul de operare guest trebuie să sufere modificări pentru a rula pe o mașină virtuală.

Astăzi, prețul resurselor hardware nu mai constituie o problemă, astfel încât motivul original pentru care au fost implementate mașinile virtuale nu mai este atât de important.

Cercetătorii sunt interesați de aplicațiile pe care mașinile virtuale le pot oferi, cum ar fi flexibilitatea, o protective mai mare și independent hardware.

Multe proiecte le utilizează pentru a obține migrarea mediilor de lucru sau interfețe reprogramabile în rețele de senzori.

În mod special mașina Xen, datorită performanței excelente, este targhetata în multe arii de cercetare.

Spre exemplu, proiectul Planetlab și-a propus să îmbunătățească sistemul de operare Planetlab astfel încât să poată suporta atât porțiuni de Vserver cât și Xen pentru a obține un management remote cât mai bun al nodurilor Planetlab. Așa încât putem afirma că mașinile virtuale au un mare potențial.

8. *Bibliografie*

[1]. The architecture of virtual machine, James E. Smith University of Wisconsin-Madison, Ravi Nair IBM T.J. WatsonResearch Center

[2]. Introduction to Virtual Machines, J.E. Smith and Ravi Nair, 2005 J. E. Smith and Ravi Nair, Virtual Machine Architectures, Implementations and Applications

[3]. An Overview of Virtual Machine Architectures, J. E. Smith and Ravi Nair

[4]. http://ro.linux-discovery.wikia.com/wiki/Crearea_unei_masini_virtuale

[5]. http://labs.cs.upt.ro/labs/lft/html/LFT10.htm