Facultatea de Electronica, Telecomuncatii si Tehnologia Informatiei

Virtualizarea procesorului si a memoriei intr-un hipervizor

Coordonator: Student:

Conf. dr. ing. Stefan Stancescu Petre Marian

Grupa 432 A

Universitatea Politehnica Bucuresti

**Cuprins**

1. Introducere pagina 3

Clasificare hipervizori

Conditii de virtualizare

Tipuri de virtualizare

2. Translatie binara pagina 6

3. Shadow page table pagina 10

4. Avantaje si dezavantaje fata de sisteme de operare native pagina 11

5. Bibliografie pagina 13

**Introducere**

Un hipervizor (sau virtual machine monitor – VMM) este un program ce creaza o platforma virtual pe un un singur sistem hardware, numit statie host, pe care ruleaza si sunt monitorizate mai multe sisteme de operare. Sistemele de operare create in masini virtuale(MV), se mai numesc si statii guest. [1]

Conceptul de virtualizare a fost creat in anii ’60 de IBM. In 1967 a aparut primul sistem de operare care a implementat virtualizarea completa, numit Control Program-40 (CP-40), ce permitea rularea mai multor instante a sistemului de operare Cambridge Monitor System (CMS). [2]

Gerald J. Popek si Robert P. Goldberg au clasificat [3] hipervizorii (1974) in:

1. Tip 1: Hipervizori nativi (sau bare-metal)

Acestia ruleaza direct pe hardware-ul host-ului fara un sistem de operare (SO) host. Este tipul preferat in majoritatea aplicatiilor. Exemple: Oracle VM, Microsoft Hyper-V, VMWare ESX si Xen.

2. Tip 2: Hipervizori hostati

Fata de cei de tip 1, acestia ruleaza pe un SO host. Acesta poate fi orice sistem de operare, cum ar fi Windows, Linux sau MacOS. Exemple: Oracle VM VirtualBox, VMWare Server and Workstation, Microsoft Virtual PC, KVM, QEMU si Parallels.

In figura 1 sunt ilustrate cele doua tipuri.

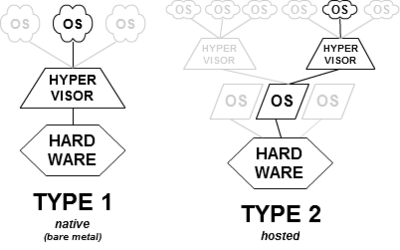


Figura 1 [4]

Hipervizorul are controlul total al resurselor sistemului si le partajeaza sistemelor de operare guest, dand impresia acestora ca au acces la hardware-ul propriu-zis.

Popek si Goldberg au considerat ca sunt trei conditii necesare ca un hipervizor sa indeplineasca pentru a suporta virtualizarea corespunzator: eficienta (un procent mare de instructiuni trebuie executate fara interventia hipervizorului), controlul resurselor (hipervizorul trebuie sa aiba control complet asupra resurselor virtulizate) si echivalenta (programul trebuie sa aiba aceeasi comportare ca pe hardware). In acest scop, ei au introdus o clasificare a instructiunilor: privilegiate (daca sunt executate in modul utlizator, are loc o “capcana”), sensibile (se executa diferit daca sunt executate in modul kernel, in comparatie cu modul utilizator) si neprivilegiate. O capcana are loc daca, de exemplu, o masina virtuala incearca sa modifice sau acceseze memoria unei alte masini virtuale. [3] [5]

Tinand cont de arhitectura hipervizorului, exista mai multe tipuri de virtualizari:

1. Virtualizare completa (full virtualization)

Reprezinta simularea completa a hardware-ului. Hipervizorul ofera aceleasi interfete hardware ca cele oferite de hardware-ul platformei simulate dand impresia SO guest ca acesta nu se afla intr-un mediu virtualizat. Virtualizarea completa este folosita des pentru a permite utilizarea aplicatiilor care ruleaza numai pe versiuni vechi ale unui anumit sistem de operare, ce nu are drivere pentru hardware-ul curent. Este foarte ineficienta, dar s-a putut mari eficienta prin diferite metode (ex. recompilare dinamica).

Pentru ca o masina virtual nu trebuia sa aiba posibilitatea de a altera alte masini virtuale sau hardware-ul, arhitectura x86 (cu 17 instructiuni sensibile, dar neprivilegiate) nu a suportat virtualizarea completa pana la aparitia Intel VT (2005) si AMD-V (2006), decat printr-o operatie numita translatie binara care inlocuia instructiunile cu probleme. [7]

1. Paravirtualizare (paravirtualization)

Are o interfata similara cu cea a sistemului fizic, dar nu este identica.

Kernel-ul SO guest este modificat special pentru a rula pe hipervizor, astfel incat toate apelurile la instructiunile privilegiate sunt inlocuite cu apeluri catre hipervizor. Aceasta metoda necesita portarea kernel-ului SO guest si a driverelor la para-API. Nu pot fi rulate versiuni nemodificate ale SO prin paravirtualizare. In general, ofera performanta mai buna fata de virtualizarea completa, fiindca SO guest a fost modificat in acest sens. [8]

3. Virtualizare hibrida (hybrid virtualization)

Reprezinta o imbinare a primelor doua metode, unde SO guest folosesc paravirtualizarea pentru anumite drivere hardware, si SO host foloseste virtualizarea completa pentru alte lucruri. Aceasta metoda ofera o performanta crescuta fara a fi nevoie ca SO guest sa fie paravirtualizate complet. De exemplu, SO guest poate folosi virtualizarea completa pentru instructiunile privilegiate din kernel si paravirtualizare pentru operatiile de intrare/iesire, folosind un driver special din SO guest. In acest fel, SO guest nu trebuie paravirtualizat complet (ceea ce in anumite cazuri nu e posibil), dar poate profita de unele imbunatatiri prezente la aceasta metoda de virtualizare. [6]

**Translatie binara**

Translatia binara este o metoda de a implementa virtualizarea completa a procesorului (CPU) prin simularea unui set de instructiuni al unei arhitecturi de alt set de instructiuni prezent in alta arhitectura.

Pe scurt, aceasta metoda este implementata in urmatorul mod: hipervizorul analizeaza si executa codul generat de statia guest, daca gaseste instructiuni sensibile (dar neprivilegiate), le translateaza in timp real in instructiuni privilegiate si le pune intr-un cache pentru a fi folosite ulterior. [9] [10]  
 Exista doua metode pentru a realiza translatia: static si dinamic.

Metoda statica translateaza tot codul unui fisier executabil in cod ce relueaza pe arhitectura curenta fara a rula codul inainte. Aceasta reprezinta o metoda dificila de realizat din diferite motive. De exemplu, codul obiect combina des instructiunile cu datele in acelasi spatiu de adrese, ceea ce face foarte dificila distingerea intre cele doua. Un exemplu ar fi tabela de salt creata la utilizarea instructiunii switch, compilata impreuna cu codul executabil. In tabela gasim adrese, nu cod si pentru a interpreta corespunzator este necesar cunoasterea detaliata a generatorului de cod al compilatorului si nu va functiona daca folosim alt compilator.

Cea dinamica reprezinta analiza si translatia codului in timpul executarii. Acest lucru implica necesitatea ca analiza si optimizarea (daca e realizata) sa aiba loc intr-un timp cat mai scurt. [11] [12]

In cazul arhitecturii x86, hipervizorul trebuie sa ia in considerare inele de protectie, pentru a realiza translatia. Acestea au scopul de a restrictiona accesul anumitor procese pentru a proteja date si pentru a imbunatati securitatea sistemului. Ring 0 este cel mai privilegiat, find cel care interactioneaza cel mai direct cu resursele fizice ale calculatorului si este locul unde ruleaza kernelul, iar ring 3 este cel cel cu cele mai putine privlegii si locul unde ruleaza de obicei aplicatiile, numit si modul utilizator.

Daca hipervizorul se afla pe cel mai privilegiat (ring 0), kernelul sistemelor de operare guest trebuie sa fie in inelele mai putin privilegiate. In majoritatea solutiilor de virtualizare se poate observa ca acesta e tinut in ring 1. [13]

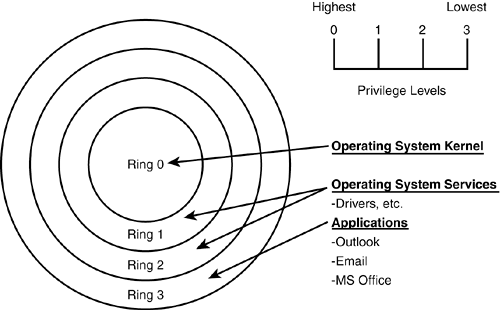


Figura 2 [14]

Translatia binara este realizata, in general, prin segmentarea codului in unitati mai mici, numite basic blocks (set de instructiuni care se termina cu o instructiune de salt, fara a avea una in interior). Rezultatul e cache-uit (deci nu mai trebuie translatate in viitor din nou) si este realizata o corespondenta intre codul sursa si codul destinatie (un map table) folosind de obicei tabele de dispersie (hash tables). In continuare este executat urmatorul basic block (daca a fost translatat) sau se incepe translatarea lui, ceea ce in final duce la executarea numai a codului translatat, cu exceptia codului care conecteaza basic blocks intre ele (de care se ocupa un program de administrare a intregului proces de translatie). Dupa cum putem observa, in acest caz, este necesara revenirea de fiecare data la programul de administrare, ceea ce afecteaza performanta. Pentru a rezolva asta, basic blocks pot fi “legate” intre ele (procedeu numit block-chaining). Acest lucru poate fi realizat prin inlocuirea instructiunilor de salt catre programul de administrare, cu un salt catre urmatorul basic block, folosind map table. Adresa urmatoarului basic block este determinata folosind adresa din codul sursa ca cheie in hash table petru a gasi adresa succesorului translat (daca chiar a fost translatat). In cazul in care nu a fost translatat, este introdus un cod ciot (stub) la sfarsit si dupa ce acest basic block va fi translatat, codul ciot va fi inlocuit de programul de administrare cu un salt direct catre codul translatat al succesorului. [12] [15.1]

In continuare vom arata modul in care poate fi structurat un translator de tip dinamic: executabilul este transmis catre un modul (numit front-end) care generareaza un cod intermerdiar in modul urmator: incarca programul in memoria virtuala, dezasambleaza codul, il analizeaza pana gaseste un salt (aceasta procedura este apelata recursiv pe codul la care se face salt). Aceasta reprezentare este apoi compilata sau emulata de translator pentru a genera cod masina (neoptimizat) pentru masina destinatie. Daca la un moment dat, translatorul determina ca e nevoie de o noua regiune din program, front-end-ul este apelat pentru a genera codul intermediar. Astfel fragmentele de cod care nu sunt necesare la o executie nu sunt translatate (Figura 3). [11]

In translatia dinamica este posibila o imbunatatire majora, numita recompilare dinamica, care permite imbunatatirea performantei prin recompilarea anumitor fragmente din codul unui program in timpul executiei. De exemplu, Intel a introdus incepand cu arhitectura Nehalem (AMD a facut-o in arhitectura Barcelona), instructiunea popcnt (population count sau distanta Hamming intre doua siruri de aceeasi lungime, unul avand toate valorile 0) in setul de instructiuni SSE4.2, care permite o executare de 2-3 ori mai rapida decat implementarea software obisnuita. Un hipervizor ar putea profita de acest lucru pentru a realiza un cod mai eficient. [11] [16] [17] [18]

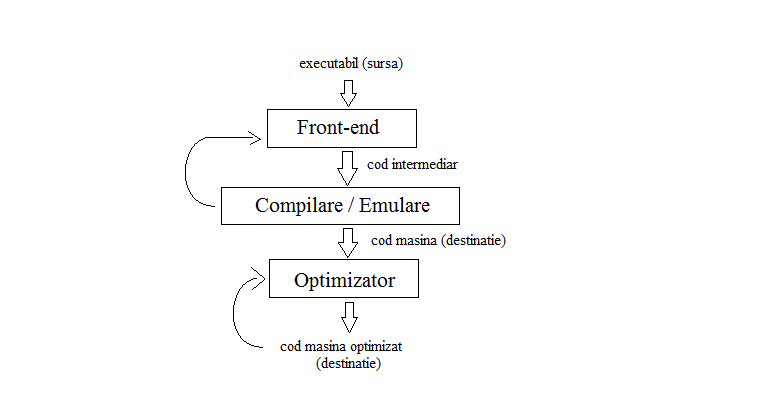


Figura 3 [11]

In codul de mai jos avem o translatie de la arhitectura x86 la PowerPC, exemplificata de Smith si Nair [15.1]. Avem doua basic block la care nu mai e necesara revenirea la programul de administrare pentru a realiza tranzitia de la primul la al doilea, daca deja au fost translatate. De retinut ca salturile indirecte tot trebuie sa fie procesate de programul de administrare pentru ca nu poate fi realizata asocierea in cazurile in care destinatia unui salt se schimba.

Arhitectura x86:

4FD0: addl %edx,(%eax) ; incarca valorile si le insumeaza

movl (%eax),%edx ; salveaza in memorie

sub %ebx,1 ; decrementeaza bucla

jz 51C8 ; salt daca s-a ajuns la finalul buclei

4FDC: add %eax,4 ; incrementeaza %eax

jmp 4FD0 ; salt inaintea buclei de mai sus

51C8: movl (%ecx),%edx ; salveaza ultima valoare a %edx

xorl %edx,%edx ; pune valorea 0 in %edx

jmp 6200 ; salt in alta parte

Translatie PowerPC:

9AC0: lwz r16,0(r4) ; incarca valorea din memorie

add r7,r7,r16 ; insumeaza

stw r7,0(r4) ; salveaza in memorie

addic. r5,r5,-1 ; decrementeaza bucla, seteaza cr0 (conditional register)

beq cr0,pc+12 ; salt daca s-a ajuns la capatul buclei

bl F000 ; salt si link(prin adaugarea *l* la instructiunea de salt *b*) catre programul de administrare

4FDC ; salveaza contorul (numaratorul) de program sursa in registrul link, un registrul care salveaza adresa de returnare cand o apelare de functie se sfarseste

9AE4: b 9C08 ; salt la urmatorul block (codul ciotul a fost inlocuit de un salt catre urmatorul basic blcok), in mod obisnuit aici s-ar fi realizat saltul catre programul de administrare

51C8 ; salveaza contorul de program sursa in registrul link

9C08: stw r7,0(r6) ; stocheaza valoarea din %edx

xor r7,r7,r7 ; pune valoarea 0 in %edx

bl F000 ; salt si link catre programul de adminstrare

6200 ; salveaza contorul de program sursa in registrul link

**Shadow page table**

Translatia binara se ocupa cu virtualizarea CPU, dar un PC mai are memorie si dispozitive de intrare/iesire, care la randul lor trebuie virtualizate pentru a permite folosirea lor in mai multe masini virtuale.

Pentru ca o MV nu trebuie sa aiba acces la memoria fizica (altfel ar putea afecta alte MV sau chiar hipervizorul, deci ar prelua controlul hardware-ului) hipervizorul trebuie sa se interpuna intre MV si memoria fizica.

O tabela de pagini este o structura de date folosita pentru a mapa adrese virtuale la adrese fizice (in RAM), dar in cazul unui hipervizor, pentru a determina adresa fizica a hardware-ului host, mai este necesara o mapare a memorie reale a SO guest. In acest caz, memoria reala este memoria care SO guest crede ca reprezinta memoria lui fizica. Pentru a crea aceasta iluzia, hipervizorul intercepteaza instructiunile care modifica memoria si le face sa indice catre o alta tabela de pagini, numita shadow page table (folosind registrul de control 3 (CR3) al procesorului la arhitectura x86). [21] [15.2]

Urmatorul exemplu arata cum se creaza shadow page tables:

- SO guest incearca sa adauge o noua intrare in tabela de pagini, dar nu are acces direct la hardware;

- pentru a realiza maparea, zona de memorie in care se afla tabela de pagini este marcata read-only de catre hipervizor;

- cand SO guest incearca sa modifice zona de memorie, se lanseaza o capcana care este prinsa de hipervizor;

- hipervizorul modifica shadow page table, creaza asocierea in MMU (Memory Management Unit) si intoarce fluxul executiei la SO guest pentru a se repeta instructiunea care a cauzat eroare. [15.2] [19]

In acest caz, avem o eroare de pagina cauzata de existenta acestei virtualizari a memoriei. Se poate observa ca toate aceste erori trec prin hipervizor, ceea ce afecteaza performanta considerabil. Din acest motiv Intel si AMD au introdus suport hardware pentru virtualizarea memorie (Extended, respectiv Nested Page Tables). [20]

**Avantaje si dezavantaje fata de sisteme de operare native**

Virtualizarea prezinta numeroase beneficii, cum ar fi:

Reducerea costurilor

Instaland mai multe masini virtuale pe acelasi server fizic este posibila folosirea mai eficienta a resurselor, ceea ce duce la scaderea numarului de servere fizice necesare. Fiind mai putine servere fizice, poate fi redus si spatiul necesar.

Usor de relocat

Nefiind legat de un hardware specific este foarte usor de mutat o MV intr-o alta locatie fizica. [22]

Snapshot

In eventualitatea unei probleme, se poate face un snapshot ce permite revenirea la setarile initiale, ceea ce evita pierderea timpului cu problemele aparute in urma unui upgrade sau a unor teste. [22]

Monitorizare imbunatatita

Solutiile de hipervizori prezente pe piata permit, in general, o buna monitorizare centralizata a sistemului. Aceasta ajuta la depanarea si analiza sistemului. [22]

Management de la distanta imbunatatit

Lucruri ce in mod obisnuit ar fi necesitat deplasarea pana la serverul fizic, pot fi rezolvate de la distanta printr-o consola. Asta ajuta mult la reducerea costurilor de calatorie si nefunctionare. [22]

Backup

Odata ce sarcina realizata de o MV nu mai e necesara, continutul poate fi usor arhivat si copiat pentru a utiliza resursele in alte scopuri. Daca redevine necesar, poate fi copiat inapoi si pornit intr-un timp foarte scurt. [22]

Usor de administrat

Daca serverul fizic are resurse neutilizate ce sunt necesare la un moment dat intr-un SO guest sau daca acesta nu mai are nevoie de atat de multe resurse, este foarte usor de optimizat modificand cateva setari si dand restart la MV. [26]

Suport legacy

Permite emularea unor sisteme de operare ce nu ruleaza pe hardware-ul curent cu scopul de a folosi aplicatii vechi. [28]

Desi prezinta numeroase avantaje, exista si dezavantaje:

Viteze de transfer reduse

Este posibil ca intr-un mediu virtual sa avem nevoie sa comunicam de pe mai multe masini virtuale cu acelasi dispozitiv (disc / placa de retea). In aceste cazuri, hipervizorul va partaja accesul la acel dispozitiv si vom avea viteze de transfer reduse. [23]

Daca serverul fizic sau hipervizorul are o problema, este posibil ca toate serverele virtuale sa nu pot fi folosite. De exemplu, daca este folosit un server fizic si acesta devine nefunctional, in mod obisnuit exista un server de rezerva, care ii poate lua locul, pana problema e remediata, dar folosind servere virtuale, problema aparuta poate afecta tot sistemul si in cazul solutiilor de tip cloud, chiar numeroase afaceri.

Probleme specifice de securitate

Daca un hacker reuseste sa preia controlul hipervizorului, are acces la mai multe servere. [24]

Complexitate crescuta

Faptul ca masinile virtuale pot rula diferite aplicatii si sisteme de operare pe acelasi sistem hardware aduce un spor de complexitate si poate necesita cunostinte avansate legate de virtualizare, in comparatie cu administrarea unui singur sistem de operare. [25]

Performanta redusa

Anumite aplicatii au cerinte hardware mai mari daca sunt virtualizate [25]. De retinut ca exista cazuri in care performanta poate fi chiar imbunatatita prin virtualizare. [27]

**Bibliografie**

1. http://en.wikipedia.org/wiki/Hypervisor

2. http://en.wikipedia.org/wiki/IBM\_CP-40

3. Popek, Gerald J.; Goldberg, Robert P. (1974). "Formal requirements for virtualizable third generation architectures". Communications of the ACM 17 (7): 412–421. doi:10.1145/361011.361073

4. http://en.wikipedia.org/wiki/File:Hyperviseur.png

5. http://en.wikipedia.org/wiki/Popek\_and\_Goldberg\_virtualization\_requirements

6. http://www.softpanorama.org/VM/index.shtml

7. http://en.wikipedia.org/wiki/Full\_virtualization

8. http://en.wikipedia.org/wiki/Paravirtualization

9. http://en.wikipedia.org/wiki/Binary\_translation

10. http://www.vmware.com/files/pdf/VMware\_paravirtualization.pdf

11. Cifuentes, C.; Malhotra, V., "Binary translation: static, dynamic, retargetable?," Software Maintenance 1996, Proceedings., International Conference on , vol., no., pp.340,349, 4-8 Nov 1996

12. http://www.ravellosystems.com/blog/nested-virtualization-with-binary-translation/

13. A. S. Tanenbaum, H. Bos, Modern Operating Systems, 4th edtion, Mar. 2014

14. http://flylib.com/books/en/1.34.1.80/1/

15. Smith, J. E, and Nair, R. (2005). “Virtual Machines: Versatile platforms for systems and processes”. Morgan Kauffmann

15.1 Capitol 2: Emulation: Interpretation and Binary Translation

15.2 Capitol 8: System Virtual Machines

16. http://en.wikipedia.org/wiki/Dynamic\_recompilation

17. http://www.strchr.com/crc32\_popcnt

18. http://en.wikipedia.org/wiki/Hamming\_weight

19. http://en.wikipedia.org/wiki/Shadow\_table

20. http://www.anandtech.com/show/2480/

21. http://www.quora.com/What-is-shadow-paging-in-virtualization-context

22. http://community.spiceworks.com/how\_to/42797-understanding-the-benefits-of-virtualization

23. http://www.computerworld.com/article/2543722/data-center/virtualization-s-downsides.html/

24. http://www.cio.com/article/2428191/virtualization/server-virtualization--top-five-security-concerns.html

25. https://4sysops.com/archives/seven-disadvantages-of-server-virtualization/

26. http://www.winxnet.com/blog/blog-home/three-benefits-of-virtualization-for-small-and-mid-sized-businesses/

27. http://www.zdnet.com/article/yes-virtualization-is-faster-sometimes-than-native-hardware/

28. https://technet.microsoft.com/en-us/magazine/hh922964.aspx