Robert Čordaš

RealVM

U ovom projektu se razmatra izrada novog tipa virtualne mašine čiji je zadatak paralelno izvršavanje i brzi prelaz između više različitih operativnih sistema. Virtualna mašina koristi realni procesor za izvršavanje koda guest operativnih sistema i u mogućnosti je da koristi realni hardver računara. Realizovan je na x86_64 (AMD64) arhitekturi. To se ostvaruje pokretanjem kernela u user modu. Tako se omogućuje brzi prelazak između različitih operativnih sistema i pokretanje više kernela u isto vreme. Sa ovim tipom virtualne mašine takođe se omogućava veća brzina izvršavanja koda u odnosu na standardne virtualne mašine, pošto se emuliraju samo neki neophodni delovi hardvera. (delovi memory managmenta, IRQ i DMA kontrolera, itd).

Uvod

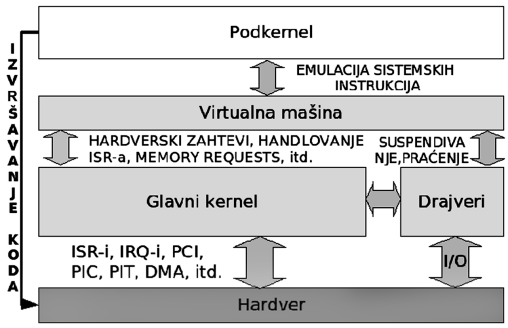
O kernelima. Kernel je glavni deo operativnog sistema. On komunicira sa hardverom računara, upravlja memorijom, pokreće procese, nudi osnovne servise programima (npr: pisanje na disk), itd. Kernel radi u kernel modu, što znači da ima potpunu i isključivu kontrolu nad hardverom. Aplikativni procesi (aplikacije) rade u user modu. Kada aplikacija u user modu proba direktno pristupiti hardveru, kernel je blokira. Sam procesor ima ugrađenu hardversku zaštitu koja sprečava aplikativne procese da direktno pristupaju hardveru. Razlog je zaštita bezbednosti sistema, pošto ako bi više programa nezavisno probali de koriste isti harver, to bi dovelo do rušenja sistema. Pomoću user moda kernel može da prati rad user procesa pomoću exception handling mehanizma.

Kernel u user modu. Osnovna ideja ovog projekta je pokretanje kernela u user modu. Time se omogućava nadzor rada pokrenutih kernela operativnih sistema. Kerneli se pokreću kao user procesi, što znači da se mogu schedulovati, suspendovati, itd. Može da se realizuje razdela hardverskih elemenata računara između operativnih sistema koji rade u isto vreme. Sve to olakšava rad korisnicima koji koriste više operativnih sistema na istom računaru.

Izrada

Projekat je realizovan kao jedna specijalna virtualna mašina implementirana u kernel koji je posebno dizajniran za ovu namenu. Računar kod bootovanja pokreće ovaj kernel (u daljem tekstu glavni kernel), a on učita kernel instaliranog operativnog sistema (u daljem tekstu podkernel) i pokreće ga kao user proces. Time glavni kernel zadržava sebi privilegije, što mu omogućava da ostane u memoriji transparentno za podkernele i da prati rad podkernela.

Struktura virtualne mašine je prikazana na slici 1.



Slika 1. Struktura virtualne mašine

Figure 1. Virtual machine structure

Virtualna mašina je specijalno dizajnirana, da bi mogla pokretati podkernele na realnom hardveru. To znači da se ne emulira ceo procesor, kao što je slučaj u nekim virtualnim mašinama. Postoje i izuzeci, kao što je QEmu koji pokreće podkernele na realnom

Robert Čordaš (1989), Mužlja, Doža Đerđa 25, učenik 2. razreda Zrenjaninske gimnazije

procesoru, ali pošto je i sama virtualna mašina user proces, mora da emulira mnogo veći procenat koda, i to na komplikovaniji način, a samim tim se usporava emulacija. Kada podkernel proba da komunicira sa realnim hardverom, procesor to detektuje (pošto je podkernel user proces), i generiše određene izuzetke. Te izuzetke obrađuje glavni kernel da bi emulirao operaciju koju podkernel želi da izvrši. To omogućava praćenje rada hardvera, što omogućava drajverima glavnog kernela da prate stanje hardverskih elemenata i da ih suspenduju veoma brzo. Ovo je potrebno samo kod potpune promene trenutno aktivnog kernela, tojest kada glavni kernel mora da ponovo mapira hardverske elemente u novu instancu virtualne mašine, a pritom da sačuva stanje hardvera da kasnije prekinuti kernel može da nastavi sa izvršavanjem. Ako se hardverski elementi razdele između operativnih sistema, to nije potrebno. Glavni nedostatak ovakvog suspendovanja je da glavni kernel mora koristiti drajvere koji su specifični za svaki hardver.

Problemi

Najozbiljniji problem je što se u implementaciji javlja potreba da glavni kernel ostane u memoriji, i da bude potpuno transparentan za podkernele. Memory managment procesora se zato mora emulirati. U celom projektu ovaj deo utiče najviše na brzinu izvršavanja podkernela. Za ostvarivanje transparentnosti ima različitih pristupa u zavisnosti od radnog režima procesora virtualne mašine. Pošto je glavni kernel realizovan kao 64-bitni kernel, kod 32-bitnih podkernela to se realizuje tako što se glavni kernel mapira u virtualni adresni prostor posle četvrtog gigabajta virtualne memorije, čemu podkernel ne može pristupiti. Kernel se pokreće u ring 1, što omogućava potpuni protected mode podkernelu. Kod 64-bitnih podkernela problem je ozbiljniji, pošto podkernel može pristupiti celom virtualnom adresnom prostoru. Iz ovog razloga podkernel se mora pokrenuti u ring 3, a delovima memorije kojima ne sme da ima pristup, mora postaviti supervisor privilegiju. Iz ovoga sledi da potpuni protected mode neće raditi kod podkernela. Proveravanje privilegija kod pristupa memoriji neće raditi (user programi će moći da pišu na bilo koji deo virtual address space-a koji je mapiran). Na ostale delove protected moda ne utiče ova promena, jer virtualna mašina i dalje proverava pristupe hardveru. Pritom mora da se vodi računa o tome da podkernel ima potpuni pristup (bez preusmeravanja) virtualnom adresnom prostoru u koji je mapiran sam kernel i koju često koristi (gde podkernel mapira procese). U suprotnom bi se mnogo usporavalo izvršenje podkernela.

Potrebna je emulacija hardvera koji treba da se deli između svih kernela (glavnog i svih podkernela). To se realizuje u zavisnosti od određenog hardvera.

Kod switchovanja celog kernela ili remapiranja nekog hardvera za drugu instancu virtualne mašine neophodno je da glavni kernel prati izvršavanje podkernela, ali na takav način da ne uspori previše rad podkernela. To je potrebno jer glavni kernel i drajveri moraju biti informisani o radu podkernela da bi mogli da ih suspenduju.

Realizacija

Kernel i virtualna mašina je realizovana u programskom jeziku C i u asemblerskom jeziku. Implementiran je za x86_64 arhitekturu (AMD64 i IA64). Sastoji se od dva dela:

- Glavni kernel koji pokreće virtualnu mašinu
- Virtualna mašina je proces na glavnom kernelu. Ona se brine o zahtevima podkernela, i o emulaciji određenih delova mašine.

Glavni kernel

Glavni kernel je razvijen za potrebe ovog projekta. U kernelu je implementirana osnovna kontrola hardvera računara, tj: memory manager, IRQ i ISR management, scheduler za procese i threadove, drajveri za razne hardverske elemente (npr: za hard disk, PIT, PIC). Tako je dizajniran da lako sarađuje sa virtualnom mašinom.

Memory manager je dizajniran tako da lako omogući manipulaciju nad potrebnim page translation tablicama od strane virtualne mašine. To znači da mora da bude u stanju da izvrši neke operacije, kao što je brisanje tablica, veoma brzo (detaljnije u sekciji *Emulacija 32 i 64 bitnog koda*).

ISR i IRQ managment je implementiran tako da na bilo koji interrupt može da se povezuje handler sa virtualne mašine ili iz samog podkernela.

Scheduler radi sa priority scheduling algoritmom. Prioritet threadova se automatski menja: što više koriste CPU, dobijaju manji prioritet, da bi se odzivnost sistema učinila većom (ima mogućnosti da proces, koji korisnik trenutno koristi, dobije veći prioritet).

Drajver za disk radi u PIO modu, što ima nekoliko prednosti u ovom projektu. Zbog toga glavni kernel ne zauzima ni DMA kanal, ni memoriju iz donjih 16Mb-a, koja može zatrebati podkernelima.

Detekcija PCI uređaja se vrši direktnim skeniranjem PCI magistrale. Podaci o svakom uređaju se čuvaju, da bi virtualna mašina mogla emulirati PCI BIOS.

Bootovanje

Bootovanje je pokretanje operativnog sistema. Kada bootloader učita glavni kernel, on inicijalizuje hardver računara, a zatim pokrene virtualnu mašinu. Ona učita boot sector HDD-a ili particije i počne izvršavati bootloader podkernela. Bootloader mora da se pokreće u 16-bitnom modu, što izaziva dodatne probleme u procesu bootovanja. U 16-bitnom modu posle bootovanja glavnom kernelu se ne može više pristupiti (procesor nije u mogućnosti da izvršava 16 bitni kod), pošto 64-bitni mod ne podržava virtual 8086 mod. To se može rešiti samo potpunom emulacijom real moda. Kada bootloader pređe u protected mode (32-bitni mod), virtualna mašina prelazi u polu-emulacioni mod (tj. emulira samo sistemske delove hardvera, kojima se ne može pristupati iz user moda). To radi preko 32-bitnog compatibility moda CPU-a. Ako se operativni sistem prebacuje u 64-bitni mod, onda kernel menja samo deskriptor kod segmenta i nastavi sa izvršavanjem.

Emulacija 16-bitnog CPU-a

Emulacija 16-bitnog CPU-a je realizovana tako što emulator sekvencijalno čita niz instrukcija (code segment), i emulira izvršavanje svake instrukcije. Emulator je isti kao emulator za 32 bitni i 64 bitni mod, sem u nekim parametrima. Izuzeci se detektuju tokom emulacije, a drugi interrupti se obrađuju nakon izvršenja instrukcije koja se trenutno emulira.

Većina 16-bitnih programa koristi BIOS za I/O funkcije. Zbog toga je potrebno emulirati BIOS pozive. 16 bitni protected mode nije implementiran, jer nije potreban za današnje operativne sisteme koji se široko koriste (Windows i Linux).

Emulacija 32 i 64- bitnog koda

Kada podkernel pređe u protected mode (ako podkernel uključi PE bit u CR0 registru, emulator čeka far jump koji će skočiti u 32-bitni kod), sve strukture iz emuliranog procesora kopira u realni CPU, kreira compability mode segemente, i pokrene 32-bitni kod.

Ako podkernel izvršava neku sistemsku instrukciju, procesor generiše General Protection Fault. Tada virtualna mašina proba da emulira tu instrukciju. Ako je uspela, onda se vraća izvršavanju koda. Ako nije, onda prosledi interrupt podkernelu.

Emulacija memory managementa radi tako što page translation tablice obrađuje virtualna mašina. Ona vodi računa o preusmeravanju određenih fizičkih i virtualnih adresa. To je potrebno iz razloga što glavni kernel mora da se sakrije u virtualnom adresnom prostoru. Pri izvršavanju INVLPG instrukcije (updateuje page tablicu), koja generiše izuzetak General Protection Fault (u daljem tekstu #GP), i kod Page Faulta (u daljem tekstu #PF) emulator proverava tablice koje koristi podkernel i prepisuje ih u tablice realnog CPU-a. Pritom vodi računa da fizičke stranice koje su već korišćene od strane glavnog kernela i podkernela kernela ne budu mapirane i da iste fizičke adrese uvek budu mapirane na istu fizičku stranu. Kada glavni kernel detektuje da podkernel pokušava promeniti baznu adresu glavnog pagetabla (PML4 tablica, u koju se mapiraju sve ostale tablice hijerarhijski, proba da piše u CR2 registar), briše sve tablice na realnom procesoru, koje pripadaju adresnom prostoru tog podkernela. To zahteva poseban mehanizam za brzo brisanje page tablice. Ovaj mehanizam koristi osnovnu ideju da se unmapuju samo page tablice najvišeg nivoa (PDP tablice), a ostale se unmapuju kada zatreba novi fizički page. Kada podkernel proba da pristupi memoriji koja nije mapirana ili je supervisor page, CPU generiše #PF i emulator proverava fizičku adresu te stranice u page tableu podkernela, i vodi računa o preusmeravanju memorijskih lokacija. Ako je podkernel izazvao #PF onda virtualna mašina prosledi #PF virtualnoj mašini. Ovakav metod omogućava vrlo brzu promenu celog page tablea.

Emulacija 32-bitne segmentacije se vrši pomoću compatibility mode segmenata. Kada glavni kernel detektuje da je podkernel inicijalizovao novi segment, on napravi ekvivalentan na realnom procesoru.

Potrebno je emulirati i neke važne delove računara, koji se ne mogu deliti između kernela. Dva najvažnija dela hardvera su PIT (Programmable Interrupt Timer) i PIC (Programmable Interrupt Controller). PIT se mora emulirati da bi kernel mogao schedulovati procese. Potrebno je da se PIT programira na frekvenciju, najveću među onima koje koriste kerneli (i glavni i podkerneli) u sistemu, a

ostalima se ta frekvencija deli sa nekom konstantom koja se izračunava od strane glavnog kernela. Određeni delovi PIC-a se moraju emulirati da bi glavni kernel mogao da handluje IRQ-e (interrupti od strane hardvera). Podkernelu se ne sme dozvoliti da reprogramira PIC na neki drugi ISR, jer u tom slučaju bi glavni kernel izgubio kontrolu nad hardverom. Zato je potrebno IRQ-e prerađivati u glavnom kernelu pre nego što se proslede podkernelu.

Handlovanje određenih interrupta (prekida) se takođe emulira. To znači da glavni kernel detektuje interrupt i proverava da li ga je izazvao podkernel ili glavni kernel. Ako je izazivač podkernel, onda mu se prosledi interrupt, tako što se emulira izvršavanje INT instrukcije. Interrupti koje ne koristi glavni kernel se ne emuliraju, nego se direktno vezuju za podkernel.

Emulacija PCI BIOS-a i detekcije hardvera se vrši tako što glavni kernel detektuje PCI hardver na računaru, a kad podkernel skenira PCI bus on prosledi nađene hardverske elemente, a ako koristi PCI BIOS glavni kernel ga emulira. To je realizovano pomoću emulacije I/O-a koje koristi PCI magistrala (0xCF8 i 0xCFC). Ovakva realizacija omogućava da se hardverski elementi podele između podkernela, pošto glavni kernel može sakriti određene hardverske elemente od određenih kernela. Tu se mora voditi računa i o IRQ konfliktima, koji mogu da nastanu ako više podkernela mapiraju različite uređaje na isti IRQ. To se postiže proveravanjem već mapiranih IRQ-a, i alociranjem još neiskorišćenih IRQ kanala za taj hardver. Kasnije se isti IRQ preusmerava na originalni IRQ koga je podkernel mapirao.

U glavnom kernelu se ne koristi DMA zato što zahteva korišćenje DMA kanala i donjih 16Mb-a fizičkog adresnog prostora, što može biti potrebno podkernelima. Ako glavni kernel detektuje da podkernel mapira u svom adresnom prostoru adresu iz donjih 16Mb-a, on sačuva informacije o mapiranju. Ako posle toga glavni kernel detektuje da podkernel pokušava programirati DMA kontroler sa tom adresom, on alocira stranicu u donjih 16Mb realne memorije i promeni informacije u page translation tablicama da bi podkernel pristupao toj memorijskoj lokaciji. DMA kontroleru se prenosi adresa realne stranice.

Praćenje rada podkernela sa hardverom je potrebno da bi se mogli suspendovati podkerneli (ova vrsta suspendovanja se koristi da bi hardverski elementi povremeno mogli da budu remapirani za drugi podkernel, a da se pritom ne mora isključivati prvi podkernel). Druga vrsta suspendovanja se koristi ako više podkernela žele da koriste isti DMA kanal ili zbog nekog konflikta pri pristupanju nekom drugom resursu, koji je vezan za konkretnu instancu virtualne mašine. Ova druga vrsta suspendovanja se koristi i za schedulovanje podkernela. Za ovu vrstu nije potrebno praćenje stanja hardvera. Zato je potrebno da se prate neki I/O portovi i memorijske lokacije. Za I/O portove to se rešava proveravanjem adrese: kada virtualna mašina detektuje pristup I/O portu ona proverava da li je neki drajver tražio praćenje tog porta, i ako jeste onda taj pristup signalizira drajveru. Za memorijske lokacije ovo se realizuje pomoću paginga, tako što se mapira nepostojeći page (ili se isključi Page Available bit za taj page). Tada se kod svakog pristupa memoriji generiše #PF preko koga drajveri mogu znati šta zapravo radi podkernel. Zatim se emulira izvršavanje instrukcije koja je izazvala #PF. To omogućava vraćanje trenutnog stanja hardvera posle suspendovanja i brzo suspendovanje. Ovaj metod se koristi samo u takvim slučajevima gde glavni kernel ne može trenutno da dobije dovoljno infromacija od hardverskog elementa kako bi ga mogao vratiti u originalno stanje hardvera.

Zaključak

Ovaj projekat pruža osnovu za novi tip virtualne mašine. Usavršavanje projekta, implementacija suspendovanja i schedulovanja kernela je takođe moguća, ali u momentu izrade projekta nije bila od interesa. Do sada su implementirani kernel, osnovni delovi potrebni za implementaciju virtualnih mašina, kao što su memory managment virtualne mašine, preusmeravanje memorijskih lokacija potpuno transparentno za podkernel i 16-bitna emulacija. Emulacija protected moda nije obuhvaćena ovim projektom jer je implementacija takve mogućnosti slična mogućnostima koje jesu implementirane (treba implementirati kopiranje segment descriptora, skeniranje page tablice podkernela, i treba pokrenuti podkernel kao user proces). Ovaj projekat pokazuje da se kerneli mogu pokrenuti u user modeu bez emulacije celog procesora i da se u virtualnim mašinama može koristiti realan hardver (za razliku od sadašnjih, gde se svaki hardverski element emulira). To će mnogo olakšati i ubrzati korišćenje više operativnih sistema na jednom računaru. Jako je

korisno u slučajevima kada neki žhardverski element nema drajver za korišćeni operativni sistem i za njega se mora koristiti drugi (npr: poznat je problem korišćenja softverskih modema na Linux operativnom sistemu).

Glavna prednost ovog tipa virtualne mašine je brzina i mogućnost korišćenja realnog hardvera. Brzina je veća pošto se emuliraju samo sistemske instrukcije. Takva vrsta emulacije je mnogo brža od standardne gde se svaka instrukcija dekodira i emulira, iako ona zahteva veći broj instrukcija od standardne emulacije zbog suspendovanja procesa.To je opravdano činjenicom da su sistemske instrukcije mnogo ređe i u samom kernelu kao user instrukcije.

Nedostatak projekta je u tome što se ne mogu koristiti supervisor page tablice u protected modu. To običnim korisnicima ne znači mnogo, ali kod kritičnih aplikacija predstavlja veliki problem. Za tu primenu može se implemetirati modifikovana vrsta virtualne mašine, koja koristi potpuno emulaciju kernel moda i samo user procese pokreće na realnom procesoru. To je međutim mnogo sporije iz razloga što se svaka instrukcija mora emulirati.

Slični projekti su XEN (http://xensource.com) i Qemu (http://fabrice.bellard.free/fr.qemu/). Najbitnija razlika između XEN-a i ovog projekta je što se kernel mora rekompajlirati da bi radio na XEN-u, dok u ovom projektu to nije potrebno. Ovo je vrlo bitno kod komercijalnih operativnih sistema, kao što je Windows. Glavna osobina Qemu-a je da ne koristi realan hardver računara, i radi u user modu kao običan proces, i koristi mnogo komplikovaniji (sto automatski znači sporiji) mehanizam za emulaciju memory managmenta (http://fabrice.bellard.free.fr/qemu-tech.html). Qemu ima i mogućnost korišćenja Linux kernel modula (Qemu accelerator) za poboljšanje performansi, ali i dalje ne obuhavta mogućnost korišćenja realnog hardvera.

Planovi za razvoj:

- Podržavanje protected mode (32-bitnih) podkernela. To je moguće korišćenjem već implementiranih osnova. Treba implementirati kopiranje segment descriptora, skeniranje page tablice podkernela i pokrenuti podkernel kao user proces.
- Podržavanje 64-bitnih podkernela. Izvelo bi se modifikacijama na 32-bitnoj virtualnoj mašini.
- Rad sa virtualnim hardverom. To bi omogućilo podelu realnog hardvera između podkernela, i korišćenje nekog hardvera u isto vreme na više podkernela.
- Suspendovanje operativnih sistema. Implementacija je već diskutovana u sekciji Emulacija 32 i 64-bitnog koda. Moguće je implementirati dve vrste suspenzije. Prva vrsta je brža, radi na osnovama praćenja rada hardvera (opisan u sekciji Emulacija 32 i 64 bitnog koda). Glavni nedostatak ovog postupka je u tome što je vezan za određeni hardver.
  Drugi metod je univerzalan za sve hardverske elemente, međutim sporiji je od prvog metoda i zahteva da ga podkernel podržava. To bi se implementiralo pomoću emulacija ACPI BIOSa.
- Podržavanje potpunog protected moda kao opcija emuliranja, tojest da kernel koristi supervisor pageove (diskutovano u sekciji *Problemi*). To se može uraditi tako što se za kernel koristi potpuna emulacija. To negativno utiče na brzinu izvršavanja, međutim povećava bezbednost sistema.
- Schedulovanje operativnih sistema (pokretanje više kernela u isto vreme). Tu je problem kako što brže suspendovati operativni sistem. To bi zahtevalo da se koristi što više virtualnih hardvera, pri čemu glavni kernel može vrlo brzo da sačuva stanje. Svaki podkernel treba da bude u memoriji. Na taj način realna memorija računara se mora podeliti na više delova (po jedan za svaki podkernel). To bi omogućilo da svaki podkernel ima svoju fiksnu veličinu memorije.
- Suspendovanje na HDD. To je mala ekstenzija suspendovanju operativnih sistema, koja bi strukture kreirane od strane suspend algoritma snimala na hard disk.
- Usavršavanje malloc algoritma glavnog kernela, da oristi AVL stablo za što brže nalaženje oprimalnog prostora memorije.
- Guest OS debugger. Omogućanje debuggovanja celog operativnog sistema, što je od velike koristi u debagovanju kernela.
- Direktno loadovanje Linux kernela iz fajla (bez pokretanja GUB-a ili LILO-a)

Literatura

Knjige

AMD 2005*a*. AMD64 Technology: AMD64 Architecture Programmer's Manual Volume 2: System Programming

AMD 2005*b*. AMD64 Technology: AMD64 Architecture Programmer's Manual Volume 2: General Purpose and system instructions

Carter P. A. 2003. *The PC assembly language*

Hyde R. *The art of assembly language*

McLean P. T. Information Technology – AT Attachement with Packet Interface – 6 (ATA/ATAPI – 6)

Brumm P., Brumm D. 1987. 80386 – *A Programming and Design Handbook*

Ostalo

Devlopment of a pmode multitasked disk driver. Dostupno na: http://clicker.sourceforge.net/docs/teacup/Disk_Programing.html

Deltener J. *DMA programming*

Scheduling algorithms. Dostupno na: http://www.ctl.ua.edu/math103/schwduling/scheduling_algorithms.html

Job Scheduling Algoritms in Linux Virtual Server. Dostupno na: http://www.w3.org/TR/html14/strict.dtd

The 'standard' cpu scheduling algorithm. Dostupno na: http://tunes.org/čunios/std-sched.html

Robert Čordaš

RealVM

The goal of this project is to develop a new type of virtual machine which would allow parallel execution and fast switching between different OS. The virtual machine uses the real processor to execute guest OS code and is able to use real computer hardware. The virtual machine is implemented on x86_64 (AMD64) architecture. The goal is accomplished by executing kernel in user mode, thus enabling fast switching and parallel execution. This type of parallel machine also enables greater speed of code execution when compared to standard virtual machines, because only some parts of hardware are emulated (parts of memory management, IRQ and DMA controllers, etc.)