

Statični nadzornik navideznih strojev za procesorsko arhitekturo RISC-V

Marko Križman, Jurij Mihelič

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana, Slovenija
E-pošta: jurij.mihelic@fri.uni-lj.si

Povzetek. V članku predstavimo razvoj in izvedbo nadzornika navideznih strojev za procesorsko arhitekturo RISC-V. Nadzornik omogoča statično razdelitev virov, pri kateri se procesorska jedra, pomnilnik in naprave navideznim strojem dodelijo ob zagonu, med delovanjem pa se ta dodelitev ne spreminja. Z ustrezno razdelitvijo virov zagotovimo medsebojno izolacijo navideznih strojev in predvidljivo delovanje, kar je ključno pri varnostno kritičnih sistemih. Nadzornik uporablja razširitev za virtualizacijo in podpira dvostopenjsko preslikovanje naslovov, virtualizacijo serijskih naprav, emulacijo zunanega prekinitvenega krmilnika ter paravirtualizacijo bločnih naprav prek razsvetljenega gonilnika. Delovanje smo preizkusili v emulatorju QEMU na platformi riscv64-virt v različnih konfiguracijah, vključno s sočasnim izvajanjem dveh navideznih strojev z operacijskim sistemom xv6, od katerih ima vsak svojo serijsko napravo, disk in emulirani prekinitveni krmilnik.

Ključne besede: virtualizacija, nadzornik navideznih strojev, statična razdelitev virov, RISC-V

Static virtual machine monitor for the RISC-V processor architecture

We present the development and implementation of a virtual machine monitor (VMM) for the RISC-V processor architecture. The VMM enables static resource partitioning, where processor cores, memory, and devices are assigned to virtual machines at startup and remain fixed throughout operation. Such resource partitioning ensures mutual isolation between virtual machines and predictable behavior, which is essential for safety-critical systems. The VMM leverages the hardware virtualization extension and supports two-stage address translation, virtualization of serial devices, emulation of the external interrupt controller, and paravirtualization of block devices via an enlightened driver. We evaluate the VMM on the QEMU riscv64-virt platform across several configurations, including the concurrent execution of two virtual machines running the xv6 operating system, each with a dedicated serial device, disk, and emulated interrupt controller.

Keywords: Virtualization, Hypervisor, Static resource allocation, RISC-V

1 UVOD

V sodobnih vgrajenih sistemih se pogosto sočasno izvajajo naloge z različnimi varnostnimi zahtevami, od varnostno kritičnih do tistih brez posebnih omejitev. Pri tem kritične naloge zahtevajo predvidljivo in robustno delovanje ter omejen odzivni čas, saj je od njih odvisna stabilnost celotnega sistema. Napaka v eni nalogi ne sme vplivati na delovanje preostalih, še zlasti ne varnostno kritičnih.

Tradicionalno to dosežemo s fizično ločeno namensko strojno opremo za vsako nalogo, vendar je to drago, prostorsko potratno in zahtevno za vzdrževanje. Danes te izzive rešujemo z *virtualizacijo*, ki znotraj istega fizičnega računalnika vzpostavi ločeno obravnavo posameznih nalog, kot da bi se te izvajale na fizično ločenih računalnikih.

Pri tem je ključna programska komponenta *nadzornik navideznih strojev* (tudi *hipervizor*), ki omogoča hkratno delovanje več *navideznih strojev* (oz. računalnikov) na skupni fizični strojni opremi [13], [16]. Ločimo hipervizorje tipa 1, ki delujejo neposredno na strojni opremi, in tipa 2, ki delujejo kot aplikacija znotraj gostiteljskega operacijskega sistema.

Nadzornik vsakemu navideznemu stroju zagotovi lasten navidezni pogled na strojno opremo in skrbi, da napaka v enem navideznem stroju ne vpliva na delovanje preostalih. Tako je mogoče na enem fizičnem sistemu sočasno izvajati aplikacije različnih stopenj kritičnosti ob ohranitvi potrebne izolacije.

Za vgrajene sisteme s strogimi zahtevami glede odzivnega časa je zlasti primerna *statična razdelitev virov* [15]. Procesorska jedra, pomnilnik in naprave se navideznim strojem dodelijo ob zagonu sistema, nato dodelitev virov ostane enaka ves čas delovanja, s čimer se zagotavlja predvidljivejša in varnejša delovanje. Slabost tega pristopa je slabša izkoriščenost virov, kadar so obremenitve navideznih strojev neenakomerne.

Primere uporabe virtualizacije najdemo v sodobnih vozilih, kjer so aktivni varnostni mehanizmi (npr. protiblokirni zavorni sistem, protizdrsni sistem in zasilno zaviranje) združeni s funkcijami udobnosti (npr. navigacija, informacije in zabava), ali v medicinski opremi

Prejet 12. avgust, 2026

Odobren 10. september, 2026



Avtorske pravice: © 2026
Creative Commons Attribution 4.0
International License

(npr. ventilatorji in infuzijske črpalke), kjer kritična regulacija deluje vzporedno z nekritičnim prenosom podatkov v bolnišnični informacijski sistem.

Procesorska arhitektura RISC-V [3], [12], [18] pridobiva vse večji pomen tako v industriji kot v akademskem okolju. Z razširitvijo H prinaša neposredno strojno podporo za virtualizacijo, ki odpravi potrebo po neučinkovitih pristopih, kot so prestrezanja in programska emulacija privilegiranih operacij. Ta vključuje nove ravni privilegiranih operacij za nadzornik in gostujoče operacijske sisteme, dvostopenjsko preslikovanje naslovov ter mehanizme za posredovanje prekinitve gostom.

V tem prispevku predstavimo programsko izvedbo statičnega nadzornika navidezni strojev tipa 1 za procesorsko arhitekturo RISC-V, ki vključuje razširitev H za virtualizacijo. V izvedbi se osredotočimo na prostorsko izolacijo, enostavnost izvedbe, učinkovito obravnavo pasti (izjeme in prekinitve) in podporo za gostujoče operacijske sisteme, kar ponazarjamo s sočasnim izvajanjem dveh navidezni strojev z operacijskim sistemom xv6. V naslednjem razdelku podamo pregled področja, nato v razdelku 3 predstavimo lastnosti arhitekture RISC-V, ki so pomembne za virtualizacijo. V razdelku 4 podrobno predstavimo izvedbo nadzornika, v razdelku 5 jo ovrednotimo, v zadnjem razdelku pa podamo še zaključek.

2 PREGLED PODROČJA

V pregledu področja zajamemo odprtokodne programske rešitve za izvedbo nadzornika navidezni strojev. Pri tem se osredotočimo na rešitve, ki podpirajo procesorsko arhitekturo RISC-V ali pa so primerne za manj zahtevne računalniške sisteme (npr. vgrajeni sistemi z manj viri).

Kot prvi odprtokodni nadzornik navidezni strojev za arhitekturo RISC-V (podpira tudi arhitekturi ARM in x86-64) se ponaša Xvisor [11]. Gre za celovit monolitni hipervizor tipa 1 s podporo za polno virtualizacijo in dinamično upravljanje gostov.

Hipervizor RVirt [2] je akademski prikaz zmožnosti virtualizacije arhitekture RISC-V brez izkoriščanja posebnih virtualizacijskih razširitev. V ta namen izvedba uporablja programsko emulacijo, kjer vsak privilegirani ukaz gosta sproži past, ki jo hipervizor obdela z emulacijo, kar omejuje učinkovitost in praktično uporabnost rešitve.

Jailhouse [14] je odprtokodni hipervizor tipa 1 podjetja Siemens, ki je uveljavil pristop statičnega particioniranja za vgrajene sisteme, s čimer zagotavlja prostorsko izolacijo z minimalnim posredovanjem hipervizorja. Slabost te rešitve je odvisnost od operacijskega sistema Linux za zagon in upravljanje particij.

Projekt Bao [10] je odprtokodni hipervizor tipa 1, ki se osredotoča na varnost in zanesljivost. Namenjen je vgrajenim sistemom, osnovanim na arhitekturah ARM in RISC-V. Samostojna (od Linuxa neodvisna) prostorska

Tabela 1 Primerjava izbranih rešitev za prostorsko izolacijo

Rešitev	Tip	RISC-V	Pristop
Xvisor [11]	1	da	polna virt. emulacija
RVirt [2]	1	da	emulacija
Jailhouse [14]	1	ne	statično part.
Bao [10]	1	da	statično part.
Xen [1]	1	delno	polna virt.
KVM [7]	2	da	polna virt.
Hafnium [17]	1	ne	varno okolje
seL4 [8]	1	da	mikrojedrni OS

izolacija temelji na statični razdelitvi pomnilnika, kar mu omogoča minimalen vpliv na učinkovitost delovanja.

Med zahtevnejše rešitve sodi odprtokodni projekt Xen [1], ki je eden izmed najbolj znanih in razširjenih hipervizorjev s podporo za polno virtualizacijo, pri čemer podpora za arhitekturo RISC-V trenutno še ni dokončna. Gre za mikrojedrni hipervizor tipa 1, pri čemer je upravljanje strojne opreme prepuščeno privilegiraniemu gostu (običajno to nalogo opravlja operacijski sistem Linux).

Podobno zahteven je projekt KVM [7], kjer gre za odprtokodno rešitev za polno virtualizacijo, ki je vgrajena neposredno v jedro operacijskega sistema Linux. Posledično deluje kot hipervizor tipa 2, pri čemer učinkovito izkorišča ustrezne strojne razširitve za virtualizacijo (Intel VT-x, AMD-V ali razširitev H arhitekture RISC-V).

Kot primer drugačnega pristopa k izolaciji omenimo odprtokodni projekt Hafnium [17], ki je referenčna implementacija upravljalnika varnih particij za arhitekturo ARM s podporo za TrustZone. Ne gre torej za splošnoimenski hipervizor, ampak za prostorsko izolacijo, ki temelji na strojni podpori (razširitev Secure-EL2), arhitektura RISC-V pa zato ni podprta.

Za konec omenimo še projekt seL4 [8] kot primer formalno verificiranega mikrojedrnega operacijskega sistema (zagotavlja le nekaj sistemskih klicev, zato ga lahko obravnavamo tudi kot hipervizor). Pravilnost jedra je dokazana s formalno verifikacijo v dokazovalnem pomočniku Isabelle/HOL, pri čemer je varnost obravnavana prek treh lastnosti: zaupnost, celovitost in razpoložljivost.

Glavni programski jezik za razvoj hipervizorjev je še vedno C v kombinaciji z zbirniki izbranih procesorskih arhitektur. Med zgoraj omenjenimi rešitvami edino RVirt uporablja programski jezik Rust. Povzetek primerjave izbranih lastnosti je prikazan v tabeli 1.

3 ARHITEKTURA RISC-V

V tem razdelku predstavimo osnove arhitekture RISC-V, pri čemer se osredotočimo predvsem na lastnosti, ki omogočajo izvedbo virtualizacije.

3.1 Splošno o arhitekturi

RISC-V [19], [20] je odprta procesorska arhitektura, zasnovana po načelih arhitektur z zmanjšanim naborom ukazov [3], [12]. Posledično definira precej enostavne strojne ukaze, pri čemer jih le peščica dostopa do pomnilnika (*angl.* load/store). Ukazi so običajno iste dolžine in izvedljivi v istem številu urin ciklov.

Arhitektura ponuja visoko stopnjo modularnosti, pri čemer je jedro arhitekture sestavljeno iz nekaj deset osnovnih ukazov, razširitve pa ponujajo dodatne ukaze in funkcionalnosti. Razširitev M na primer dodaja množenje in deljenje, razširitvi F in D dodajata računanje s plavajočo vejico z enojno oz. dvojno natančnostjo, razširitev H pa dodaja virtualizacijske zmožnosti.

Procesor vsebuje eno ali več procesorskih jeder z eno ali več strojnimi nitmi, ki jim v terminologiji RISC-V pravimo hart (*angl.* hardware thread). Vsak hart hrani svoje stanje in deluje kot samostojna enota, ki izvaja ukaze.

Stanje procesorja je sestavljeno iz 32 splošnonamenskih registrov (označenih z x0-x31), pri čemer register x0 vedno vsebuje vrednost 0 (vsi biti so 0). Poleg tega lahko hart omogoča še kontrolno-statusne registre, ki vsebujejo informacije o njegovem delovanju in izvedenih ukazih.

3.2 Ravni privilegiranosti

Za omogočanje zaščite med različnimi komponentami programskega sklada (npr. operacijski sistem in uporabniški programi) sodobni procesorji omogočajo izvajanje ukazov na različnih *ravneh privilegiranosti*. Standard RISC-V ponuja tri osnovne ravni: M (*angl.* machine), S (*angl.* supervisor) in U (*angl.* user), pri čemer mora implementacija arhitekture omogočati vsaj raven M. Pomen posameznih ravni je predstavljen v tabeli 2.

Tabela 2 Osnovne ravni privilegiranosti arhitekture RISC-V

M	Raven M omogoča popoln nadzor nad sistemom in je običajno namenjena strojno-programski opreči.
S	Raven S delno omejuje neposredni dostop do virov in je namenjena operacijskemu sistemu.
U	Raven U nima neposrednega dostopa do virov in je namenjena uporabniškim aplikacijam.

3.3 Kontrolni in statusni registri

Razširitev Zicr definira podporo za do 4096 kontrolnih in statusnih registrov (*angl.* control and status register, CSR) ter strojne ukaze za njihovo branje in spreminjanje. Posamezni kontrolno-statusni registri so definirani v ustrezni razširitvi glede na njeno funkcionalnost.

Pri uporabi registrov so stranski učinki dovoljeni le ob pisalnih dostopih, pri bralnih dostopih pa ne. Vsak CSR pripada določeni ravni privilegiranosti, pri čemer

ima najbolj privilegirana raven dostop do vseh, manj privilegirane pa samo do določene podmnožice.

3.4 Pasti

Past se proži ob posebnih dogodkih sistema, kot je napačen pomnilniški dostop, izvedba privilegiranega ukaza ali prekinitev naprave. Pri tem se izvajanje trenutnega ukaza prekine in preusmeri na rokovalnik pasti. Poleg tega se v arhitekturi RISC-V nastavijo še vrednosti nekaterih CSR-jev, npr. *xcause* (vzrok pasti), *xepc* (vrednost programskega števca pred prožitvijo pasti) in *xstatus* (stanje privilegiranosti in virtualizacije pred prožitvijo pasti). Našteti registri imajo različna imena pri različnih ravneh privilegiranosti (namesto *x* nastopa oznaka ravni, npr. M ali S).

Sinhronim pastem pravimo *izjeme*, pri čemer programsko izjemo prožimo z ukazom *ecall*, s katerim implementiramo okoljske klice (npr. sistemski klic jedra operacijskega sistema). Za vračanje v okolje, kjer je bil izveden okoljski klic, sta na voljo ukaza *xret* za raven M oz. S. Ob vračanju oba upoštevata polji *xPP* registra *xstatus*, ki hrani prejšnjo raven privilegiranosti.

Asinhronim pastem pravimo *prekinitve*. Pri tem s prekinitvami programske opreme in časovniki upravlja *lokalni prekinjevalnik jeder* (*angl.* core local interruptor, CLINT), z zunanjimi prekinitvami pa *prekinitveni krmilnik* (*angl.* platform-level interrupt controller, PLIC). Slednji podpira 1023 prekinitvenih virov in 15872 prejemnikov prekinitev.

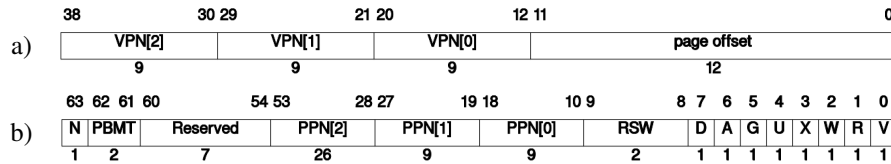
3.5 Ostranjevanje – Sv39

Ostranjevanje je mehanizem, ki se uporablja za virtualizacijo fizičnega naslovnega prostora, ki ga razdelimo na enako velike bloke (*okvirje strani*). Preko preslikovalnih tabel strani nato preslikamo strani navideznega naslovnega prostora (*angl.* virtual page number, VPN) v okvirje strani (*angl.* physical page number, PPN).

Arhitektura RISC-V omogoča več shem ostranjevanja. Omenimo le shemo Sv39, ki podpira 39-bitni navidezni naslovni prostor in velikost strani 4 KiB. Pri tem je navidezni naslov (glej sliko 1 a)) sestavljen iz treh 9-bitnih števil strani (polja VPN) in 12-bitnega odmika strani. Preslikovanje je tako trinivojsko, preslikovalna tabela strani pa hrani 512 vnosov.

Vnos v preslikovalni tabeli strani (glej sliko 1 b)) je 64-bitni. Pomembna so tri polja PPN za hranjenje števil strani in zastavice DAGUXWRV – D kot umazano (*angl.* dirty), A kot dostopano (*angl.* accessed), G kot globalno, U kot uporabniško, X kot izvajanje (*angl.* execute), W kot pisanje (*angl.* write), R kot branje (*angl.* read) in V kot veljavno (*angl.* valid). Vnos v tabeli strani je lahko naslov tabele strani naslednjega nivoja ali pa končni vnos, pri čemer velja prva možnost, kadar je bit V nastavljen, biti X, W in R pa počiščeni.

Algoritem za preslikovanje naslovov (glej sliko 2) zaradi učinkovitosti običajno izvaja enota za upravljanje pomnilnika. Za izvedbo nadzornika navideznih strojev



Slika 1 Shema odstranjanja Sv39: a) navidezni naslov in b) vnos preslikovalne tabele strani

pa ga je treba tudi programsko izvesti: enkrat pri konfiguraciji preslikovalnih tabel ter drugič pri razsvetljenem gonilniku, kjer nadzornik pretvori gostov fizični naslov v dejanski fizični naslov.

```
zero:
    long level = 2;
one:
    // fizični naslov korenske tabele strani
    long *pgtable = satp.PPN << 12;
two:
    // vnos tabele strani s pravim indeksom
    long index = va.VPN[level];
    long pte = pgtable[index];
three:
    // preveri vnos
    if ((*pte).V == 0)
        invalid_page();
    // list ali notranje vozlišče
    if ((*pte).X || (*pte).R || (*pte).W)
        goto four;
    else {
        // poglobi preslikavo
        level--;
        if (level < 0) goto four;
        pgtable = (*pte).PPN << 12;
        goto two;
    }
four:
    // preveri dostopne pravice RWX in U
    ...
    // prevedi naslov
    void *pa = va; // fizični naslov
    for (int i = 2; i >= level; i--)
        pa.PPN[i] = pte.PPN[i];
```

Slika 2 Algoritem za sprehanje po tabelah strani

3.6 Razširitev H

Če platforma podpira virtualizacijo, lahko v registru misa vklopimo (bit 7) razširitev H (*angl.* hypervisor). Vsak hart s tem pridobi bit V, ki hrani stanje omogočenosti virtualizacije in deluje podobno kot stanje ravni privilegiranosti. Poleg tega ukaza mret in sret upoštevata dodatni polji mstatus.MPV in hstatus.SPV, ki hranita prejšnje stanje omogočenosti virtualizacije.

Nadalje, razširitev H omogoči tudi dvostopenjsko preslikovanje naslovov, ki je aktivno samo v času omogočene virtualizacije. Pri tem razlikujemo stopnji VS in G, katerih preslikovanje je opisano v tabeli 3. Tabele strani samega nadzornika navideznih strojev so ločene in ne vplivajo na dvostopenjsko preslikovanje naslovov.

Vklop razširitve H predrugači ravni privilegiranosti (glej tabelo 4), ki so odvisne tudi od bita V. Pri tem nova raven HS prevzame lastnosti stare ravni S in doda

Tabela 3 Dvostopenjsko preslikovanje naslovov

VS	Naslov preslikovalne tabele strani se hrani v registru vsatp. Preslikuje gostove navidezne naslove v gostove fizične naslove.
G	Naslov preslikovalne tabele strani se hrani v registru hgatp. Preslikuje gostove fizične naslove v dejanske fizične naslove.

Tabela 4 Kodiranje vseh ravni privilegiranosti: razširitev H izklopljena (1. stolpec) in vklopljena (2. in 3. stolpec)

H=0	H=1, V=0	H=1, V=1
M	M	/
S	HS	VS
U	U	VU

funkcionalnosti virtualizacije, saj je namenjena izvedbi nadzornika navideznih strojev. Ravni VS in VU sta novi in se virtualizacije ne zavedata. Prva (VS) je namenjena izvedbi (gostujočega) operacijskega sistema, druga (VU) pa njegovi uporabniški programski opremi.

4 RAZVOJ IN IZVEDBA

V tem razdelku predstavimo pomembnejše vidike razvoja in izvedbe naše rešitve. Njena izvorna koda v programskem jeziku C je javno dostopna*.

4.1 Platforma riscv64-virt

Platforma riscv64-virt je virtualna strojna oprema, ki jo zagotavlja emulator QEMU za razvoj in testiranje programske opreme na arhitekturi RISC-V. Njen fizični naslovni prostor je 56-biten, njegova razdelitev pa je prikazana v tabeli 5.

Prostor od naslova 0x80000000 naprej je namenjen programski opremi. Tu je strojno-programska oprema OpenSBI [5], ki zagotavlja osnovne V/I-storitve in jo priskrbi emulator QEMU, tej pa sledijo nadzornik navideznih strojev in navidezni stroji. Med pomnilniško preslikanimi območji so za naš nadzornik ključna naslednja: CLINT (upravljanje časovnikov in lokalnih prekinitvev), PLIC (upravljanje zunanjih prekinitvev), UART (serijska V/I-komunikacija) in naprave virtio (virtualizirani bločni V/I).

4.2 Naslovni prostor nadzornika

Nadzornik navideznih strojev uporablja svoj naslovni prostor, ločen od naslovnih prostorov navideznih strojev.

*https://github.com/krizmanmarko/riscv_hypervisor

Tabela 5 Fizični naslovni prostor platforme riscv64-virt

ime	naslov	velikost
test	0x100000	0x1000
RTC (ura realnega časa)	0x101000	0x1000
CLINT	0x2000000	0x10000
PCIe (PMIO)	0x3000000	0x10000
platform-bus	0x4000000	0x2000000
PLIC	0xc000000	0x60000
UART	0x10000000	0x100
naprave virtio	0x10001000	0x8000
fw-cfg	0x10100000	0x18
flash	0x20000000	0x4000000
PCIe (ECAM)	0x30000000	0x10000000
PCIe (MMIO)	0x40000000	0x40000000
pomnilnik	0x80000000	konfiguracija

S tem dosežemo boljšo varnost zaradi izolacije naslovnih prostorov. Prav tako je razhroščevanje programa nadzornika lažje. Uporabili smo razporeditev naslovnega prostora, kot je prikazano v tabeli 6.

Tabela 6 Navidezni naslovni prostor nadzornika navideznih strojev

ime	navidezni naslov
pomnilniško preslikane naprave	nespremenjeno
slika jedra	0xfffffc0000000000
sklad	0xfffffc0400000000
preostali pomnilnik	0xfffffe0000000000

Pomnilniško preslikane naprave pri tem ostanejo na izvornih naslovih, kot je prikazano v tabeli 5. S tem dosežemo enostavnost in jasnost komunikacijskih vmesnikov in gonilnikov V/I-naprav.

Preostali trije vnosi v tabeli 6 so vezani na razporeditev glavnega pomnilnika. Pri tem slika jedra predstavlja kodo jedra in statične podatke, sklad je namenjen za hranjenje podatkov, pomembnih za izvajanje jedra, preostali pomnilnik pa je namenjen dinamični alokaciji pomnilnika.

4.3 Virtualizacija serijskih naprav

Nadzornik navideznih strojev podpira dve serijski napravi vrste UART, katerih konfiguracija je prikazana na sliki 3. Prva je vgrajena (oznaka `uart`, fizični naslov 0x10000000 in prekinitveni vir 10), druga pa je vzpostavljena prek vmesnika PCI in dostopna prek področja PCIe (PMIO) (oznaka `pci_uart`, fizični naslov 0x30000000 in prekinitveni vir 33); oba naslova sta razvidna iz tabele 5.

Znotraj navideznega stroja sta obe napravi dostopni na istem navideznem naslovu (0x10000000) in z istim navideznim prekinitvenim virom (10), kar gostujočemu programu oz. operacijskemu sistemu zagotovi enoten serijski vmesnik ne glede na fizično podlago.

4.4 Virtualizacija bločnih naprav

Virtualizacijo bločnih naprav dosežemo prek naprave `virtio-blk`, ki je del standardnega vmesnika `virtio` za para-

```
struct mmio_dev mmio_dev_uart = {
    .phys_base = 0x10000000,
    .virt_base = 0x10000000,
    .phys_irq = 10,
    .virt_irq = 10,
    .size = PAGE_SIZE,
    .perm = PTE_R | PTE_W,
};
struct mmio_dev mmio_dev_pci_uart = {
    .phys_base = 0x30000000,
    .virt_base = 0x10000000,
    .phys_irq = 33,
    .virt_irq = 10,
    .size = PAGE_SIZE,
    .perm = PTE_R | PTE_W,
};
```

Slika 3 Konfiguracija vgrajene in virtualizirane naprave UART

virtualizirane V/I naprave in gostujočemu OS-u zagotavlja dostop do diska. Platforma `riscv64-virt` ponuja do 8 naprav `virtio` na pomnilniško preslikanem območju od naslova 0x10001000.

Pri uporabi naprave `virtio-blk` v virtualiziranem okolju se pojavi težava z dvostopenjskim preslikovanjem naslovov (razširitev H, razdelek 3.6). Gonilnik `virtio-blk` v gostujočem OS-u konfigurira napravo z gostovimi fizičnimi naslovi vmesnih pomnilnikov ob predpostavki, da gre za dejanske fizične naslove. Naprava za prenos podatkov med diskom in pomnilnikom uporablja neposredni pomnilniški dostop (*angl.* `direct memory access`, DMA), ki obide MMU in s tem dvostopenjsko preslikovanje naslovov. Posledično naprava podatke bere oz. zapisuje na napačne fizične lokacije.

Rešitev je t.i. razsvetljeni gonilnik (*angl.* `enlightened driver`): navidezni stroj z okoljskim klicem zaprosi nadzornika, ki gostov fizični naslov preslika v dejanski fizični naslov. Gonilnik nato napravo skonfigurira s praviimi naslovi. Ta pristop zahteva spremembo gonilnika v gostujočem OS-u, kar smo storili pri operacijskem sistemu `xv6` (glej razdelek 5.2).

4.5 Upravljanje lokalnih prekinitvev

Krmilnik CLINT upravlja lokalne prekinitve za posamezna jedra. Vsebuje pomnilniško preslikana registra `mtime`, ki hrani vrednost systemskega časovnika, in `mtimencmp`, ki določa mejo, ob kateri se proži prekinitvev časovnika. V izbrani platformi `riscv64-virt` emulatorja QEMU vsi fizični harti souporabljajo isti register `mtime`.

Razširitev `Sstc` uvede CSR-je za upravljanje časovnikov pri vseh ravneh privilegiranosti: `stimecmp` za raven HS in `vstimecmp` za raven VS. Branje vrednosti časovnika je pri obeh ravneh dosegljivo prek CSR-ja `time`. Naša izvedba uporablja izključno razširitev `Sstc` – ob vstopu v past časovnika vrednost `stimecmp` oz. `vstimecmp` ustrezno ponastavimo, preden potrdimo prekinitvev. Da je razširitev `Sstc` dosegljiva tudi gostom na ravni VS, je treba pri inicializaciji harta nastaviti bita `henvcfg.STCE` (omogočanje `Sstc` za gosta) in `hcounteren.TM` (dovoljenje branja časovnika iz ravni VS).

4.6 Virtualizacija zunanjih prekinitev

Zunanje prekinitev upravlja prekinitveni krmilnik PLIC [6], ki posreduje prekinitev V/I-naprav prejemnikom – hartom na ravneh M in HS. Vsak hart ima tako dva konteksta, katerima PLIC dodeli zaporedni identifikacijski številki.

Za konfiguracijo in nadzor so ključni pomnilniško preslikani registri: prioriteta vira (interrupt priority), čakajoče prekinitev (interrupt pending), omogočanje po kontekstu (interrupt enable), prag prioritete (interrupt threshold) ter par registrov za potrditev obdelave prekinitev (interrupt claim/complete).

Ker se PLIC virtualizacije ne zaveda, ne pozna ravni VS in ne more neposredno prekinjati navideznih strojev. Poleg tega si ga vsi navidezni stroji delijo, zato bi brez posredovanja nadzornika dostopi enega navideznega stroja do registrov PLIC vplivali na delovanje drugega.

Rešitev je emulirani PLIC (*angl.* virtual PLIC, VPLIC). Nadzornik v preslikovalnih tabelah navideznih strojev preslika področje PLIC brez dostopnih pravic, tako da vsak dostop gosta sproži past. Nadzornik past obdela tako, da iz registra htval razbere naslov zahtevanega PLIC-registra (vrednost je premaknjena za 2 bita desno, kar je treba upoštevati pri dekodiranju) in iz registra htinst dolžino sprožilnega ukaza (2 ali 4 bajte) za pravilno povečanje sepc. Nato po potrebi preslika navidezno identifikacijsko številko prekinitvenega vira (*virt_irq*) v fizično (*phys_irq*) ter navidezni kontekst v fizičnega. Pri emulaciji registrov s po enim bitom na vir (npr. interrupt pending in interrupt enable) nadzornik rekonstruira celotno 32-bitno vrednost s preslikavo vsakega posameznega bita. Nato dostop izvede nad dejanskim PLIC-em. S tem je dosežena izolacija med navideznimi stroji, hkrati pa vsak navidezni stroj vidi le sebi dodeljene prekinitvene vire.

Ko PLIC proži prekinitev ravni HS, jo nadzornik prevzame ter z branjem registra claim pobriše čakajočo prekinitev in pridobi fizično identifikacijsko številko vira. To preslika v navidezno in prekinitev vbrizga ravni VS prek registra hvip. Gostu se ob poskusu branja registra claim in pisanju v register complete spet sproži past, ki jo nadzornik emulira z ustrezno preslikavo identifikacijskih številok.

4.7 Inicializacija sistema

Inicializacija sistema se začne z zagonom platforme riscv64-virt, ki za vsak hart vzpostavi začetno stanje: raven privilegiranosti se nastavi na M, prekinitev so onemogočene, uporablja se pravilo tankega konca, omogočene so vse dostopne razširitve, enota za zaščito fizičnega pomnilnika (*angl.* physical memory protection, PMP) je izklopljena.

Programski števec vsebuje vrednost 0x1000, kjer je strojna koda, ki v register a0 zapiše identifikacijsko številko harta, v register a1 pa naslov dvojiškega drevesa naprav. Koda nato preda nadzor strojno-programski

opremi OpenSBI, ki inicializira izvajalno okolje ravni M: nastavi rokovalnike pasti, pri čemer jih večino delegira ravni S. Prav tako izpostavi še nekaj okoljskih klicev, kot so npr. *sbi_hart_start*, *sbi_hart_stop* in *sbi_set_timer*.

Poleg tega nastavi še enoto PMP tako, da je preostalim ravnem onemogočeno spreminjanje strojno-programске opreme. Nato z ukazom mret preide na raven HS in skoči na naslov 0x80100000, kjer je koda nadzornika navideznih strojev.

4.8 Inicializacija nadzornika

Inicializacija nadzornika poteka v dveh fazah, ki ju ločuje vstop v končni navidezni naslovni prostor. Koda je namreč prevedena za izvajanje na visokih naslovih (0xfffffc0000000000, izvajalni naslov), ob zagonu pa se izvaja na fizičnih naslovih (0x80000000).

Pri tem je poseben izziv dinamični alokator pomnilnika, ki za upravljanje prostih okvirjev uporablja seznam s kazalci, ki morajo biti skladni z naslovnim prostorom, v katerem bo alokator deloval. Rešitev je vzpostavitev primitivnega navideznega naslovnega prostora z giganstrami, s čimer alokator že v prvi fazi deluje z naslovi, ki ostanejo veljavni tudi po vstopu v končni naslovni prostor. Takšna vzpostavitev omogoči izenačitev programskega števca in izvajalnega naslova, kar zagotovi pravilno delovanje programske kode v jeziku C. Prehod v končni navidezni naslovni prostor se tako poenostavi: treba je le premakniti sklad in popraviti relativne naslove na njem.

V drugi fazi se izvedejo naslednji koraki:

- 1) Inicializacija alokatorja pomnilnika, ki upravlja proste okvirje fizičnega pomnilnika.
- 2) Inicializacija serijske naprave in vmesnika PCI.
- 3) Vzpostavitev končnih preslikovalnih tabel z ustreznimi preslikavami za pomnilniško preslikane naprave, jedro nadzornika in dinamični pomnilnik.
- 4) Inicializacija posameznih hartov: nastavitve rokovalnika pasti za obravnavo prekinitev in izjem gostov, aktivacija odstranjevanja s končnimi tabelami in konfiguracija prekinitev.
- 5) Zagon navideznih strojev.

Rokovalnik pasti obravnava prekinitev in izjeme, ki nastanejo med izvajanjem gostov. Ob časovniški prekinitvi posodobi register stimecmp in s tem razporedi naslednji časovni dogodek. Ob zunanji prekinitvi pokliče rutino za upravljanje emuliranega VPLIC, ki prekinitev prepošlje na ustrezen navidezni stroj. Ob okoljskih klicih iz ravni VS rokovalnik pretvori gostov fizični naslov v dejanski fizični naslov. Slednjega uporabljajo razsvetljeni gonilniki, da lahko pravilno dostopajo do naprav virtio-blk. Dostopi gosta do pomnilniško preslikanega področja PLIC pa sprožijo napake pri dostopu do strani, ki jih rokovalnik preda emulaciji VPLIC.

5 OVREDNOTENJE

5.1 Zahteve in omejitve

Postavili smo nekaj zahtev in omejitev glede izvedbe funkcionalnosti nadzornika navideznih strojev. V nadaljevanju jih opišemo in predstavimo, kako smo jih upoštevali v naši programski izvedbi nadzornika. Pri tem sta prvi dve zahtevi povezani z varnostjo, tretja do peta z enostavnostjo in učinkovitostjo izvedbe, zadnji dve pa z uporabnostjo programske rešitve.

- 1) Naslovni prostori navideznih strojev morajo biti medsebojno izolirani tako, da (bralni in/ali pisalni) pomnilniški dostop do enega ne more vplivati na delovanje drugega, četudi navidezni stroj ne deluje pravilno.
- 2) Nadalje mora biti tudi naslovni prostor nadzornika navideznih strojev izoliran od naslovnih prostorov navideznih strojev.
- 3) Viri, kot so procesorska jedra, pomnilniški prostor in naprave, so dodeljeni navideznim strojem statično, torej se med delovanjem nadzornika njihova dodelitev ne spreminja.
- 4) Slika nadzornika navideznih strojev mora biti dovolj majhna, da je uporabna tudi za računalniške sisteme z omejenimi viri.
- 5) V programski kodi želimo minimizirati uporabo zbirnika, namesto tega pa (kjer je mogoče) uporabiti višjenivojski sistemski programski jezik, kot je programski jezik C.
- 6) Izvedba nadzornika in navideznih strojev mora biti prilagodljiva in uporabna. Njihova konfiguracija naj poteka med inicializacijo nadzornika.
- 7) Izvedba naj podpira vsaj dve V/I-napravi za zaporeden prenos podatkov (*angl.* universal asynchronous receiver-transmitter, UART).

Prvo zahtevo smo izplonili prek ustrezne konfiguracije enote za upravljanje pomnilnika (*angl.* memory management unit), ki izvaja preslikovanje navideznih pomnilniških naslovov (navideznega stroja) v fizične naslove. Pri tem ima vsak navidezni stroj svoj neodvisni in izolirani naslovni prostor, ki se fizično ne prekriva z naslovnimi prostori drugih navideznih strojev. S podobno konfiguracijo naslovnega prostora nadzornika navideznih strojev smo izplonili tudi drugo zahtevo.

Naša izvedba nadzornika je popolnoma statična, kar pomeni, da se dodelitev procesorskih jeder in razdelitev pomnilniškega prostora zgodi ob inicializaciji, med njegovim izvajanjem pa se ne spreminjata. Statična izvedba je enostavnejša od dinamične, kjer se dodelitev virov lahko spreminja med izvajanjem. Posledično omogoča večjo učinkovitejše delovanje in običajno tudi večjo varnost, saj je manj možnosti za pretok informacij med posameznimi navideznimi stroji.

Statična izvedba prav tako omogoča tudi manjšo dvojiško sliko nadzornika in s tem primernost za naprave z omejenimi viri, npr. naprave interneta stvari. Zaradi majhnosti slike smo pri izvedbi naslovnega prostora

nadzornika lahko uporabili primitivne tabele strani (gigastrani), kar poenostavi tudi inicializacijo in zagon nadzornika.

Pri tem smo se v večji meri izognili uporabi kode zbirnika arhitekture RISC-V, ki je prisotna le za najnujnejša sistemska opravila, predvsem za inicializacijo sklada, potrebnega za izvajanje preostale kode, napisane v programskem jeziku C. S tem smo izplonili peto zahtevo, zaradi učinkovitosti jezika C pa se učinkovitost samega nadzornika ne poslabša.

Prilagodljivost in uporabnost nadzornika navideznih strojev pa dosežemo s konfiguracijo, ki je zapisana v eni sami datoteki programskega jezika C. Ta konfiguracija se uporabi za inicializacijo nadzornika in navideznih strojev. S tem skrbniku nadzornika omogočimo tudi preprosto preverjanje konfiguracije z vidika varnostnih in drugih zahtev.

Izplonitev zadnje zahteve je odvisna od platforme, ki je na voljo. V našem primeru platforma riscv64-virt emulatorja QEMU ponuja le eno vgrajeno napravo UART. Kljub temu nam je uspelo omogočiti podporo za dodatno napravo UART z njeno virtualizacijo prek vmesnika PCI.

5.2 Uporabljeni navidezni stroji

Za testiranje delovanja nadzornika navideznih strojev smo uporabili programe, ki se lahko izvajajo neposredno (*angl.* bare metal) na (morebiti virtualizirani) strojni opremi. Takšni programi seveda nimajo podpore s strani operacijskega sistema, zato mora njihova koda vsebovati vse potrebno za delovanje, vključno z gonilniki uporabljenih V/I naprav. Uporabili smo naslednje tri programe.

- printer: Gre za enostaven program, ki uporablja več procesorskih jeder in časovnike. Pri tem vsako procesorsko jedro izpisuje svojo identifikacijsko številko v enosekundnih intervalih.
- keygrab: Program uporablja napravo UART, pri čemer prejeti bajt izpiše na izhod (zaslon). Zahteva pravilno konfiguracijo naprave UART in uporabo prekinitev.
- xv6 [4]: Gre za izobraževalni operacijski sistem po vzoru Unixa, ki ga je treba predhodno prilagoditi, da lahko deluje v virtualiziranem okolju. Posledično je treba odstraniti uporabo ravni M, dodati razsvetljeni gonilnik za disk ter zunanjo prekinitev preusmeriti prek emuliranega VPLIC.

5.3 Eksperimenti

Naštete programe smo preizkusili v različnih konfiguracijah. Prve tri preverjajo osnovno delovanje nadzornika: enojedrni printer potrdi zagon navideznega stroja in delovanje naprave UART; dvojedrni printer potrdi podporo za več hartov znotraj enega navideznega stroja; dva enojedrna printerja pa potrđita izolacijo med navideznimi stroji in pravilno virtualizacijo identifikacijskih številc hartov.

Konfiguracija s printerjem in keygrabom preizkusi deljenje naprave UART med dvema navideznima strojema. Čeprav v tem primeru ni prišlo do pokvarjenih podatkov, konfiguracija izpostavi omejitve deljene naprave: če navidezna stroja zahtevata različni konfiguraciji naprave, deljenje ni izvedljivo. Rešitev je dodelitev ločene naprave UART vsakemu navideznemu stroju, kar omogoča virtualizacija prek vmesnika PCI.

Najzahtevnejša konfiguracija poganja dva navidezna stroja z operacijskim sistemom xv6 sočasno, vsak na svojem hartu s 16 MiB pomnilnika. Vsak navidezni stroj dobi emuliran VPLIC, napravo UART in disk virtio-blk. Nadzornik pri tem preslika fizične naslove naprav in fizične identifikacijske številke prekinitev v navidezne, ki jih gost pričakuje, tako da oba navidezna stroja delujeta neodvisno in brez medsebojnega vpliva.

6 ZAKLJUČEK

V prispevku smo predstavili statični nadzornik navideznih strojev za procesorsko arhitekturo RISC-V z razširitvijo H. Nadzornik podpira statično dodelitev procesorskih jeder in pomnilnika, dvostopenjsko preslikovanje naslovov ter virtualizacijo naprav UART in virtio-blk. Zunanje prekinitve so virtualizirane prek emuliranega VPLIC, ki preslika prekinitvene vire med navideznimi stroji. Delovanje smo potrdili z vrsto konfiguracij, vključno s sočasnem izvajanjem dveh navidezni strojev z operacijskim sistemom xv6.

Za nadaljnje delo predlagamo uvedbo enote IOMMU (angl. input/output memory management unit), ki bi odpravila potrebo po razsvetljenem gonilniku in poenostavila dodeljevanje naprav navideznim strojem. Prav tako bi bil koristen prekinitveni krmilnik z zavedanjem virtualizacije, ki bi nadomestil emulacijo VPLIC.

Dolgoročni cilj je podpora za polnopravne gostujoče operacijske sisteme, kot je Linux. Razmisliti bi bilo dobro tudi o prepisovanju nadzornika v sistemski programski jezik Rust, ki ob primerljivi učinkovitosti z jezikom C ponuja močnejša zagotovila glede pomnilniške varnosti.

ZAHVALA

Raziskava je bila opravljena v okviru programa P2-0095 Vzoredni in porazdeljeni sistemi, ki ga financira Javna agencija za raziskovalno in inovacijsko dejavnost Republike Slovenije (ARIS).

LITERATURA

- [1] P. Barham, B. Dragović, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003.
- [2] J. Behrens, C. Skeggs, S. S. Ortiz, and F. Kaashoek. RVirt: A trap-and-emulate hypervisor for RISC-V. <https://github.com/mit-pdos/RVirt>, 2020.
- [3] P. Bulić. *Understanding Computer Organization: A Guide to Principles Across RISC-V, ARM Cortex, and Intel Architectures*. Springer, 2024.
- [4] R. Cox, F. Kaashoek, and R. Morris. xv6: A simple, unix-like teaching operating system. Technical report, MIT Parallel and Distributed Operating Systems Group, 2023.
- [5] RISC-V International. OpenSBI: Supervisor binary interface, 2023.
- [6] RISC-V International. RISC-V platform-level interrupt controller specification, 2023.
- [7] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. KVM: The Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, 2007.
- [8] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, 2009.
- [9] M. Križman. Statični nadzornik navidezni strojev za procesorsko arhitekturo RISC-V. Diplomsko delo, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2024.
- [10] J. Martins, A. Tavares, M. Solà, T. Oliveira, J. Pereira, and S. Pinto. Bao: A lightweight static partitioning hypervisor for modern multi-core embedded systems. In *Workshop on Next Generation Real-Time Embedded Systems (NG-RES)*, 2020.
- [11] A. Patel, M. Daftedar, M. Shalan, and M. W. El-Kharashi. Embedded hypervisor Xvisor: A comparative analysis. In *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 682–691. IEEE, 2015.
- [12] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2021.
- [13] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7), 1974.
- [14] R. Ramsauer, J. Kiszka, D. Lohmann, and W. Mauerer. Look mum, no VM exits! (almost). In *Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPRT)*, 2017.
- [15] K. G. Shin and P. Ramanathan. Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1), 1994.
- [16] J. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 2005.
- [17] Trusted Firmware. Hafnium: Reference secure partition manager. <https://www.trustedfirmware.org/projects/hafnium/>, 2023.
- [18] A. Waterman. *Design of the RISC-V Instruction Set Architecture*. PhD thesis, EECS Department, University of California, Berkeley, 2016.
- [19] A. Waterman and K. Asanović. The RISC-V instruction set manual, volume I: Unprivileged ISA, 2019.
- [20] A. Waterman, K. Asanović, and J. Hauser. The RISC-V instruction set manual, volume II: Privileged architecture, 2021.

Marko Križman je diplomiral (2024) na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. V okviru diplomskega dela [9] se je ukvarjal z virtualizacijo na procesorski arhitekturi RISC-V.

Jurij Mihelič je magistriral (2004) in doktoriral (2006) s področja računalništva na Fakulteti za računalništvo in informatiko Univerze v Ljubljani, kjer je trenutno izredni profesor v Laboratoriju za algoritmiko in predstojnik Katedre za teoretično računalništvo. Poleg tega je delno zaposlen kot višji raziskovalec pri podjetju Beyond Semiconductor d.o.o., kjer sodeluje pri temah, kot so ločevalna jedra, sledenje in nadzor informacijskega toka, formalizacija, specifikacija in verifikacija. Njegova raziskovalna področja obsegajo še kombinatorične probleme, algoritme na grafi, inženirstvo algoritmov, formalne metode, sistemsko programsko opremo in operacijske sisteme.