VI. IKERGAZTE
NAZIOARTEKO IKERKETA EUSKARAZ

2025eko maiatzaren 28, 29 eta 30a
Bilbo, Euskal Herria

ANTOLATZAILEA:
Udako Euskal Unibertsitatea (UEU)

INGENIARITZA ETA ARKITEKTURA

zeptOS: segurtasunera bideratutako sistema eragile txertatua RISC-V sistemetarako

Iker Galardi, Jose Antonio Pascual eta *Javier Navaridas*

219-226 or.
https://dx.doi.org/10.26876/ikergazte.vi.03.27

zeptOS: segurtasunera bideratutako sistema eragile txertatua RISC-V sistemetarako

Iker Galardi, Jose Antonio Pascual, Javier Navaridas

Intelligent Sistem Group (ISG),
Konputagailuen Arkitektura eta Teknologia Saila,
Euskal Herriko Unibertsitatea (UPV/EHU).
Manuel Lardizabal Pasealekua 1, 20018 Donostia.

iker.galardi@ehu.eus

Laburpena

Sistema txiki eta espezializatuen erabilerak gorakada egin du, egunerokotasunean erabiltzen ditugun tresna asko kontrolatzera iritsi arte. Gailu horietako asko internet bidez kontrolatzeko aukera dago eta honen ondorio bezala gailu horietako asko hainbat erasotzaileren ikusgai bihurtu dira pribatutasun eta segurtasun munduan kezkak sortuz. Gainera, gailu horietako askok hardware IP eta SDK propietarioak erabiltzen dituzte, garatzaileek duten kontrola eta pertsonalizazio aukerak asko mugatuz. Artikulu honen bitartez sistema txertatuetarako erabili daitekeen plataforma ireki eta aldagarri bat deskribatzen da RISC-V arkitekturaren ekosistema irekiaz eta sistema eragile txiki eta arin batez baliatuz.

Hitz gakoak: Sistema eragileak, sistema txertatuak, unix, RISC-V, OpenSBI

Abstract

The use of small and specialized systems has increased in the last decades, leading to the control of most of the devices in our every day life. As most of those devices can be controlled through the internet the devices get exposed to a lot of attackers, leading to concerns in the privacy and security communities. Additionally, a lot of these devices use proprietary hardware IP and SDKs, leading to less control and customizability in the hands of the developers. This paper describes the development of a software stack designed to provide a great development platform for embedded applications employing the open RISC-V ecosystem and a custom lightweight operating system.

Keywords: *Operating systems, embedded systems, unix, RISC-V, OpenSBI*

1 Sarrera eta motibazioa

Azken aldian sistema txiki eta espezializatuen erabilerak gorakada bat izan du egunerokotasunean erabiltzen ditugun tresna asko kontrolatuz, etxeko garbiketa robotetatik hasita osasun monitorizazio gailuetaraino. Sistema horietako batzuk ez dute interneten erabilerarik egiten, baina beste asko internet bidez kontrolatu daitezke, hainbat erasotzaileren ikusgai bihurtuz. Adopzio masibo honen ondorioz hainbat kezka sortu dira pribatutasun eta segurtasun kontzientziadun komunitatean sistema horiek konputazio modernoak ekarri dituen segurtasun neurri askoren faltaren ondorioz.

Gailu horietan erabiltzen den hardware-a mikrokontrolagailu txikienetatik hasi eta 64 biteko prozesadore ahaltsuetaraino izan daiteke, non azken prozesadore horiek askotan memoria kudeaketa unitatea (MMU hemendik aurrera) bezalako segurtasun neurriak izan ohi dituzte softwarearen banaketan laguntzeko. Hala ere, hardware askok IP eta SDK propietarioak erabiltzen dituzte, askotan garatzaileak horien plataformetara lotuz. Ekosistema irekiagoak erabiliz gero gailu horiek askoz seguru, merke eta espezializatuagoak bihurtu daitezke garatzaileek software eta hardware-aren pila kontrolatzen dutelako.

Lehen aipatutako prozesadore ahaltsu horiek askotan Linux eta BSD sistema eragileak erabiltzen dituzte, non askotan kontsumitzaile eta zerbitzariek erabiltzen dituzten segurtasun neurriak aplikatzen diren. Bestetan aldiz,

segurtasun neurri asko desaktibatzen dira errendimendu arazoak direla medio. Hala ere, sistema eragile horiek oso ahaltsu eta pisutsuak izan daitezke sistema txertatu batzuetarako.

Segurtasunerako kritikoak diren eta prozesadore ahaltsuak dituzten sistema txertatuetan Linux eta BSD sistema eragileak erabiltzeko aukera egonda ere, sistema eragile txiki eta arinago baten falta ikusi da, non haren diseinua segurtasuna buruan edukita egin den. Horretarako RISC-V ekosistema irekiaz baliatuko da aipatutako espezializazioa eta merketasun helburuak lortzeko. Beraz, proiektu honen funtsa sistema eragile propio baten bitartez sistema txertatuetarako plataforma bat sortzea da.

Artikulu honetan lehenik eta behin sistema eragile honek erabiltzen dituen teknologiak aipatuko dira, horietatik garrantzitsuenak RISC-V arkitekturaren ekosistema eta Xv6 sistema eragilea izanik. Ondoren, Xv6 sistema eragileari egindako aldaketak azalduko dira, irakaskuntzarako sortu zen sistema eragile hori Qemu emulatzailea ez den hardware-an martxan jarri ahal izateko.

2 Arloko egoera eta ikerketaren helburuak

Egunerokotasunean erabiltzen ditugun tresna asko kontrolatzen dira konputagailuen bitartez, etxeko garbiketa robotetatik hasita osasun monitorizazio sistemetaraino. Gailu horiek martxan jartzen dituzten prozesadoreak hainbat motatakoak dira, izan ere, horiek behar duten errendimendua eta energia mugak askotarikoak dira.

Energia efizientzia buruan duten proiektu askok mikrokontrolagailu oso sinpleak erabiltzen dituzte eta sistema horiek askotan hardware babes oso gutxi ekartzen dituzte. Sistema pertsonalizatuak erabiltzen dituzte non aplikazio kodeak etenak eta prozesuak kudeatzen dituen. Sistema ahaltsuagoak erabiltzen direnean denbora errealeko sistema eragileak erabili ohi dira, sistema eragile famatuenetakoa FreeRTOS (Barry *et al.*, 2008) izanik. Bestetan MMU gabeko Linux kernel-a ere erabili ohi da.

Sistema txertatu ahaltsuagoetan aldiz Linux[1] edo edozein BSD[2] sistema eragileak erabili ohi dira Yocto[3] eta NanoBSD[4] bezalako proiektuen bitartez. Proiektu horiek zerbitzari edo ordenagailu pertsonaletan erabiltzen diren sistemak eraldatzeko aukera ematen dute sistema eragileak dituen ezaugarriak gutxituz (eta memoria aurreztuz modu horretan). Sistema eragile horien erabileraren abantailarik handiena hardwarearen euskarria da, izan ere, sistema eragile erabilienak izanik hardware mota askotarako kontrolagailuak ekartzen dituzte.

Nahiz eta sistema eragile handi horiek abantaila asko dituzten, errendimendu aldetik ekarri ditzaketen arazoak handiak dira. Sistema eragile horien erabilera handiena zerbitzarien mundua denez, memoria eta nukleo kantitate handietarako daude diseinatuak, eta sistema txertatuek dituzten errendimendu eta tamaina mugak kontuan izan gabe. Ondorioz, sistema eragile txiki eta arin baten beharra ikusten da.

3 Erabilitako teknologiak

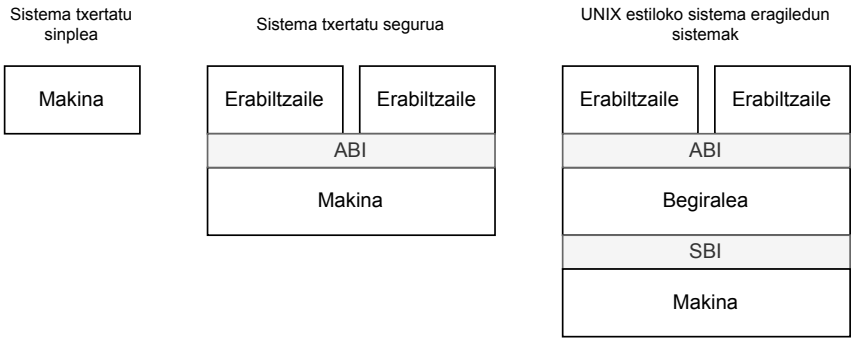
Lehen aipatutako arazoak konpondu ahal izateko software eta hardware pilak irekia izan behar dute, eta horretarako RISC-V ekosistema erabiltzea aukeratu da. Ekosistema honen gainean hainbat elementu dira garrantzitsuak, baina garrantzitsuena RISC-V agindu multzoa da, izan ere, beste piezak horren baitan aukeratu beharko dira eta.

3.1 RISC-V agindu multzoa

RISC-V agindu multzoa RISC (*Reduced Instruction Set Computer* ingelesez) motako filosofian oinarritzen den agindu multzo librea da. X86 eta ARM agindu multzoek ez bezala, RISC-V libre eta irekia da, edozeini RISC-V nukleodun prozesadoreak diseinatu, sortu eta saltzeko aukera emanez.

RISC-V arkitektura hori bi zatietan banatzen da: erabiltzaile-mailako arkitektura (ris, 2024a) eta pribilegiatu arkitektura (ris, 2024b). Arkitekturaren baseak eragiketa oso gutxi inplementatzen ditu, baina estentsioen bidez falta den funtzionalitatea (koma mugikorreko zenbakiak, bektore eragiketak, eragiketa atomikoak, etab) gehitzeko aukera dago. RISC-V arkitektura deitzen duguna izatez arkitektura familia bat da, non 4 arkitektura base definitzen diren RV32I, RV32E, RV64I, RV64E. Horien arteko diferentzia bakarra erregistroen tamaina (eta ondorioz, erakusle tamaina) eta erregistro kantitatea dira.

[1] https://kernel.org/
[2] https://www.freebsd.org/
[3] https://www.yoctoproject.org/
[4] https://docs.freebsd.org/en/articles/nanobsd/



1. Irudia: Exekuzio mailen konfigurazioak

RISC-V *load-store* motako agindu multzo bat da, non eragiketa gehienak erregistroen bitartez egiten diren eta agindu batzuen bitartez memoriatik datuak ekarri eta eraman daitezkeen. Guztira 47 agindu definitzen dira: erregistroen arteko batuketak, kenketak, konparaketak, memoria operazioak eta kontrol transferentziak. Horrez gain, inplementazioek ondorengo estentsio estandarrak gehitu ditzakete:

- **M**: osoen arteko biderketak eta zatiketak.
- **A**: eragiketa atomikoak.
- **F, D, Q**: IEEE754 (iee, 2008) motako koma mugikorreko operazioak (32 bit, 64 bit eta 128 bit).
- **C**: agindu konprimituak (aginduko memoria gutxiago erabiltzeko).
- **B**: bit manipulaziorako eragiketak.
- **V**: tamaina ezberdineko bektore operazioak.

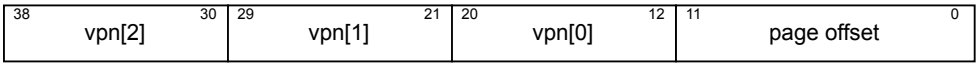
Pribilegiatu arkitekturan aldiz sistema eragile eta firmware-ak beharko lituzkeen funtzionalitateak gehitzen dira estentsio bitartez. Arkitekturak, prozesadorea 4 pribilegio maila definitzen ditu:

- **Makina-maila**: maila honek ez du mugarik. Hemen memoria zatiak babesteko aukera ematen da PNP-en (*Physical Memory Protection* ingelesez) bitartez.
- **Begirale-maila**: maila honek makina-mailako softwareak jarritako mugak bete behar ditu. Hemen memoria birtuala eta erabiltzaile-maila konfiguratu daitezke.
- **Erabiltzaile-maila**: begirale-mailako softwareak jarritako mugak bete behar ditu. Hemen exekutatzen da fidagarria ez den kodea.

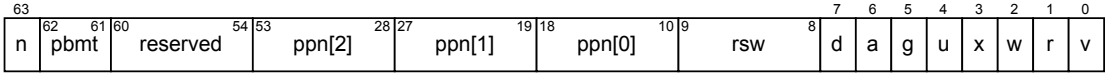
RISC-V prozesadore guztiek ez dituzte exekuzio maila guztiak inplementatzen. Mikroprozesadore txikienek makina-maila bakarrik inplementatu dezake energia eta tamaina mugak direla eta; segurtasun maila handiagoa duten prozesadoreek makina-maila eta erabiltzaile-maila inplementatu ditzakete software isolamendu estrategiak martxan jartzeko; aldiz, prozesadore ahaltsuagoek hiru mailak inplementa ditzakete segurtasun mailarik altuena izateko. 1. irudian ikus daitezke aipatutako konfigurazio horiek. Azken konfigurazio honi *konfigurazio osoa* deituko diogu.

Exekuzio maila bakoitzak bere CSR (*Control and Status Registers* ingelesez) erregistroak ditu bere burua eta azpiko mailak konfiguratu ahal izateko. Erregistro berezi horien bitartez prozesadorearen hainbat zati kontrolatu eta informazioa atzitu daiteke, memoria birtualetik hasita (`satp` erregistroaren bitartez), eten kudeaketaraino (`stvec`, `sie`, `sip` erregistroak etenak konfiguratu eta `scause`, `sepc` edo `stval` erregistroak etenei buruzko informazioa atzitzeko).

Maila arteko trantsizioak `xret` eta `ecall` aginduen bitartez egiten dira. `xret` (maila bakoitzeko agindu ezberdina: `mret` makina-mailako softwarean eta `sret` begirale-mailako softwarean) aginduen bitartez exekuzio maila jaitsi ala mantentzen da, dena `status` erregistroko *Previous Privilege* flag-ak kontuan izanda. `ecall` (ingurune-deia) aginduak aldiz exekuzio maila beti igoko du.



2. Irudia: SV39 memoria birtualeko eskemaren helbide birtualaren formatua



3. Irudia: SV39 memoria birtualeko eskemaren orrialde taularen sarreraren formatua

`ecall` aginduaren bitartez egiten diren maila trantsizioak oso garrantzitsuak dira, izan ere, horiei esker maila batean exekutatzen ari den softwareak zuzenean baimenik ez duen zerbitzuak atzitu ditzake. Exekuzio maila igotzeko mekanismoaz gain, oso garrantzitsua da definitzea nola komunikatuko diren bi mailetako softwareak, hoiek izanik 1. irudian ikus daitezkeen ABI (*Application Binary Interface* ingelesez) eta SBI (*Supervisor Binary Interface* ingelesez). ABI-aren kasuan komunikazio protokolo hori sistema eragilearekin zuzenean lotuta dago, SBI-aren kasuan RISC-V ekosistemak protokoloa bera eta erabili daitezkeen funtzioak estandarizatu (ris, 2024c) ditu sistema eragileak prozesadore eta plataforma guztietan erabilgai izateko.

3.2 SV39 memoria birtual eskema

RISC-V arkitektura hainbat memoria birtual eskema erabiltzeko gai da, bakoitza bere abantaila eta desabantailekin. Garrantzitsuena, eta sistema txertatuetan erabilgarriena, SV39 eskema da. Eskema honen bitartez 39 biteko helbide birtualak 64 biteko helbide fisikoetan itzultzen ditu 3 mailako orrialde taulak erabiliz. Beste eskemek memoria kapazitatea handitzen dute maila gehiago erabiliz, baina latentzia handitzen da itzulpen prozesuan. Aipatutako 39 bit-eko helbide birtualak 2. irudiko formatua jarraitzen du, orrialde taulako sarrera bakoitzak aldiz 3. irudiko formatua.

Helbide birtualen itzulpena zuhaitz moduan organizatzen diren taulen bidez egiten da. Zuhaitzaren erroa `satp` CSR erregistroaren bitartez ezartzen da eta orrialde taularen ibiltzearen prozesua hortik hasten da. Prozesu honek lehenik eta behin helbide birtualeko lehen zatia hartzen du (`vpn[2]`), eta erro taulan erabiltzen du indize bezala hurrengo orrialde taula bilatzeko. Ondoren, lortutako orrilde taularekin pauso berdina aplikatuko da `vpn[1]` eta `vpn[0]` helbide birtual zatiekin azken orrialde taulako sarrera lortzeko. orrialde taulan sarreraren baimenak konprobatu eta helbide fisikoa eraikiko da sarrerako `ppn` eremua eta helbide birtualeko `desplazamendua` eremuarekin.

Memoria eskema honek *superpage* izeneko 2MiB-eko eta GiB bateko orrialdeak sortzeko aukera ere ematen du bi orrialde taula edo orrialde taula bakarra erabiltzen badira hiru taula erabili beharrean.

3.3 SBI eta OpenSBI inplementazioa

SBI (ris, 2024c) makina-mailan exekutatzen den firmwarearen eta begirale-mailan exekutatzen den softwarearen arteko komunikazio protokoloa da, makina-mailan dauden zerbitzuak atzitu ahal izateko. SBI interfazea X86 eta ARM munduan UEFI (uef, 2024b) firmware interfazearen baliokidea kontsideratu daiteke, izan ere, plataformaren funtzionalitate basikoa eskeintzen du modu eramangarri batean sistema eragileari honen inplementazio karga kenduta.

RISC-V agindu multzoaren moduan, SBI espezifikazioak nahitaez inplementatu beharreko funtzio base batzuk espezifikatzen ditu, eta beharrezkoa den funtzionalitate gehigarri guztiak gehigarri moduan eskaintzen ditu. Funtzionalitate base honen bidez estentsioak inplementaturik dauden ala ez konprobatu daiteke prozesadorearen informazio basikoa atzitzeaz gain.

Firmwarearen estentsio baseak makina-mailan agertzen diren erregistro batzuk atzitzeko aukera ematen du SBI inplementazioaren informazioa atzitzeko gaitasuna emateaz gain. Estentsio basean dagoen funtzio garrantzitsuena `sbi_probe_extension` da, funtzio honen laguntzaz beste estentsioak erabilgarri dauden ala ez begiratu dezakegulako.

Ondorengoak dira estandarizatuta dauden estentsioak:

- **Timer extension**: makina-mailan atzigarri dagoen nukleoko tenporizadorea konfiguratzeko aukera ematen

du. RISC-V bertsio berriagoetan ez da beharrezkoa `Zsstc` estentsioak tenporizadorea atzigarri jartzen duelako.

- **IPI extension**: prozesadoreko nukleoen artean etenak bidaltzeko aukera ematen du.
- **RFENCE extension**: `FENCE` agindu multzoa beste nukleoetan exekutatzeko aukera ematen du.
- **Hart State Management extension**: prozesadoreko nukleoak piztu eta itzaltzeko aukera ematen du.
- **System Reset extension**: makina berrabiarazi ala itzaltzeko aukera ematen du.
- **Performance Monitoring Unit extension**: makina moduko nukleoaren errendimendu kontagailuak atzitzeko aukera ematen du.
- **Debug Console extension**: UART bidezko kontsola bat erabiltzeko aukera ematen du.
- **System Suspend extension**: sistemaren exekuzioa eteteko aukera ematen du (RAM-ean bakarrik dago definitua).
- **CPPC extension**: nukleoen energia eta errendimendu kontrola egiteko aukera ematen du.

Funtzio horien erabilera oso sinplea da, izan ere `ecall` aginduaren bitartez ematen diogu firware-ari exekuzioaren kontrola eta erregistroen bitartez aukeratzen dugu zein funtzio behar dugun eta parametroak. Halaber, firmwareak erregistro bidez bueltatuko dizkigu erroreak eta balioak.

3.4 Xv6 sistema eragilea

Xv6 (Cox *et al.*, 2011) MIT-en irakaskuntza mundurako sortu den sistema eragile sinple eta arina da. Sistema eragile honek hasiera batean X86 arkitekturarako euskarria bakarrik zuen, baina RISC-V arkitekturaren sinpletasuna dela medio agindu multzo honetara ekarri zen.

Sistema eragile hori RISC-V arkitektura oraindik oso berria zenean sortu zen, eta beraz, SBI bezalako kontzeptu berrientzat euskarririk ez du. Sistema eragile honek makina-mailan hasten du exekuzioa, firmware geruza txiki bat inplementatuz, eta ondoren begirale modura salto egiten du sistema eragilea bera inplementatzeko. Horrez gain gailu guztien erregistroen helbideak kodean sartuta dauzka.

Hala ere, sistema eragileak linux bezalako sistema eragile konplexuen ezaugarri asko dauzkalako. Kernel mailan memoria birtuala, prozesu kudeaketa eta fitxategi sistema zerbitzuak inplementatzen ditu UNIX-en seigarren bertsioaren sistema-dei eskema berdina erabiliz. Erabiltzaile softwarean ere, UNIX-en seigarren bertsioko komandoak eta *shell* inplementazio sinple bat ekartzen ditu.

3.5 Device tree formatua

Device Tree formatua sistema txertatuetan erabiltzen den hardwarea deskubritzeko modua da. Prozesadorearen firmwareak fitxategi honen erakusle bat entregatzen dio begirale softwareari dagoen hardwarea ikusi dezan eta beharrezko kontrolagailuak martxan jarri ditzan. Fitxategi honen bitartez gailu guztien erregistro blokeen helbideak eta eten topologia eskuragarri jartzen ditu. Prozesadorera konektatutako gailuez gain, prozesadoreari buruz eta memoria RAM-ari buruzko informazioa ere eskaintzen da fitxategi honen bitartez, azken hori bereziki garrantzitsua izanik sistema eragilearen memoria kudeaketa sistemarako.

Izenak adierazten duen moduan fitxategi formatu honek prozesadorera konektatutako gailu guztiak zuhaitz egitura batean organizatzen ditu, non gailuak berak zuhaitzaren hostoak diren eta gainontzeko nodo guztiak informazio gehigarria ala gailuetara konektatutako bus-ak. Zuhaitzeko nodo bakoitzak ondorengo propietateak eduki ditzake:

- **compatible**: kate lista bat da non gailuaren izena azaltzen den. Honen bitartez aukeratu dezake sistema eragile batek zein kontrolagailu erabili dezakeen.
- **model**: gailuaren modeloa deskribatzen du. Normalean 'garatzailea,modeloa' formatua segituz.
- **phandle**: gailua zenbaki batez identifikatzeko aukera ematen du. Oso garrantzitsua da propietate hori etenen topologia eraikitzeko.
- **status**: gailuaren egoera azaltzen du. Honek `okay`, `disabled`, `reserved` edo `fail` balioak izan ditzake.
- **reg**: erregistroen blokeak adierazten ditu.

- **interrupts**: zein eten egin ditzakeen gailuak adierazten du.
- **interrupt-parent**: zein eten kontrolagailuetara konektatuta dagoen adierazten du.

Fitxategi honen bitartez gailuen informazioa lortzeaz gain memoria eremu erreserbatuak eta preferentziak ere atzitu daitezke. Garrantzitsua izan daiteke `/chosen` nodoaren `stdout-path` propietatea ere, honen laguntzaz ikus dezakegulako zein den munduarekin komunikatzeko erabili dezakegun gailuetako bat.

3.6 Virtio gailu birtualak

Virtio (oas, 2022) makina birtualetako gailuen interfaze estandarra da. Nahiz eta gailu horiek makina birtualetan erabiltzeko pentsatuak dauden, beste edozein gailuk erabiltzen dituzten mekanismoak erabiltzen dituzte (lotuta dauden bus-en mekanismoak eta DMA). Gailu horiek MMIO, PCIe edo Channel I/O bitartez konektatu daitezke. Estandar honen bitartez hainbat gailu definitzen dira, ondorengoak dira garrantzitsuenak:

- **Network Device**: ethernet txartel birtuala.
- **Block Device**: disko gogor birtuala.
- **Entropy Device**: kalitate altuko ausazko zenbakiak ematen dituen gailua.
- **GPU Device**: 2D edo 3D motako prozesadore grafikoa.
- **Input Device**: teklatu, arratoi edo tableta gailuak.
- **File System Device**: FUSE motako fitxategi mailako biltegiratze gailua.
- **IOMMU**: DMA gailua konfiguratzeko erabiltzen da.
- **Sound Device**: audio txartel birtuala.
- **GPIO Device**: GPIO motako gailu birtuala.

Gailu guztiak konfiguratu eta erabiltzeko erregistroak eskema berdina jarraitzen dute. Konektatuta dauden bus-aren baitan erabilera aldatzen den arren, bus guztietan ondorengo erregistroak erabiltzen dira:

- **Device Status**: gailuaren egoera adierazten du.
- **Device feature bits**: kontrolagailuak ezartzen ditu gailuaren zein ezaugarri ezagutzen dituen.
- **Notifications**: gailuak prozesadoreari (edo alderantziz) aldaketak notifikatzeko erabiltzen da.
- **Device Configuration space**: hasieratze parametroak ezartzeko.
- **Virtqueues**: datu transferentziak egiteko erabiltzen diren ilarak.

Gailu mota ezberdinen artean dagoen diferentzia *Device feature bits*-ak, *Device Configuration space*-a eta *Virtqueue*-etan erabiltzen diren egiturak dira.

4 Proiektuaren garapena

Proiektuaren garapena bi faseetan banatu da. Lehen fasean Xv6 sistema eragilea OpenSBI firmwarearekin hasieratzea zen helburu, izan ere, firmware honen erabilerak hainbat plataformen euskarria emango digu exekuzio mailan. Azken fasean sistema eragilearen hardware detekzioa landuko da, modu honetan sistema eragileak behar duen hardware-a (disko gogorraren kontrolagailua, UART gailua, etab) detektatu eta kontrolagailuak konfiguratu ahal izango dira.

4.1 Xv6 eta OpenSBI firmware-aren erabilera

Xv6 sistema eragilea RISC-V plataformara ekarri zenean ekosistema oraindik nahiko berria zen eta ez zegoen estandarizaziorik firmware mailan doan software-ari buruz. Beraz, sistema eragile hori makina moduan eta begirale moduan exekutatzeko prestatuta dago. Sistema eragilea oso sinplea izanik, makina moduko zerbitzuetatik erabiltzen duen bakarra tenporizadorea da.

Garapen honetako lehen pausoa OpenSBI firmwarearen funtzionamendua analizatzea izan da. Analisi honetan ikusi da proiektua konpilatzerakoan 3 firmware aldaera azaltzen direla:

- `fw_dynamic`: Qemu plataforman erabiltzen den `-kernel` flag-aren bitartez lortzen du begirale softwarea non dagoen exekutatzen hasi ahal izateko.
- `fw_jump`: Konpilatzerako garaian ematen da salto egin behar duen erakuslea.
- `fw_payload`: Exekutatu behar duen begirale software-a firmware-an sartzen da.

Qemu plataformarako garapena errazteko `fw_dynamic` aldaera aukeratu da. Qemu emulatzaileak pasatako *dynamic structure* egituraz baliatuz jakiten du non dagoen begirale softwarea (normalean `0x80200000` helbidean).

Sistema eragilearen hasieratze sistema aldatu behar izan da, izan ere, makina moduan hasieratzen denean sistema nukleo guztiak exekutatzen hasten dira, eta beraz, sistema zaharrean `mhartid` erregistroaren bitartez aukeratzen zen nukleo printzipala. OpenSBI erabiltzerakoan erregistro hori ez dago eskuragarri eta OpenSBI-k aukeratutako nukleo printzipal bat bakarrik hasten da begirale moduan exekutatzen. Hasieratze sistema berria bi zatitan banatzen da. Nukleo printzipalak sistemaren hasieratzea martxan jarriko du, eta amaitzerakoan *Hart State Management* ingurune-deien bitartez hurrengo nukleoak abiaraziko dira.

Aldi berean, sistemaren hasieratzean tenporizadorea `mtimecmp` erregistroa erabiliz konfiguratu beharrean `sbi_set_timer` *SBI* deiaren bitartez konfiguratzen da. Etenen kudeaketan ez da aldaketarik egin behar, izan ere, etenak jasotzeko sistema berdina erabiltzen da.

4.2 Hardware detekzioa *Device Tree* bitartez

Inplementazio originalak hardwarea erabiltzeko sistema txertatu oso txikiek erabiltzen duten mekanismoa erabiltzen du, gailuen erregistroen erakusleak eskuz idatziz. Baina honek arazo bat du, sistema eragile bereziak behar dira hardware bakoitzerako.

Aldiz, sistema konplexuagoetan firmware-aren bitartez konektatuta dauden gailuak detektatzeko aukera dago. X86 eta ARM zerbitzarietan *ACPI* (uef, 2024a) taulen bitartez detektatzen du sistema eragileak zein gailu dauden konektatuta makinara. Sistema txertatuetan aldiz, *Device Tree* izeneko formatu baten bitartez egiten da detekzio hori. Inplementatutako hasieratze sistemak, beste sistema txertatutako software pilek egiten duten moduan, sistema eragileari *Device Tree* fitxategi bat iritsiko zaio firmwaretik hardwarea detektatu ahal izateko.

Device Tree formatuak prozesadorera konektatutako gailu guztiak zuhaitz baten moduan aurkezten ditu horien erregistroen helbideak eta eten zenbakiak erakutsiz. Horrez gain, memoriaren eremu erreserbatuak, cpu-ari buruzko informazioa eta sistemari buruzko informazio orokorragoa ere lortu daiteke.

Gure kasuan sistema eragilea nahiko espezializatua dago Qemu plataformaren gailuetarako, beraz, *Device Tree* iteratzen doan heinean gailu taula bat bete beharrean aurretik jarritako helbide horiek aldatzen joango da lortutako informaziotik. Garrantzitsua da aipatzea *Device Tree*-ko `/chosen` nodoko `stdout-path` propietateari esker aukeratzen dela begirale softwareko `printf` inplementazioak erabiliko duen irteera gailua.

Gainera, *Virtio* motako gailu guztiek `compatible` propietate berdina dute, beraz zaila da hasiera batean *Virtio* motako zein gailu den jakitea. Horretarako kodean `virtio_disk_probe` funtzio bat gehitu da non gailuari galdetzen zaion ea disko gogor bat emulatzen duen ala ez.

5 Ondorioak

Sistema txertatuen gorakada dela medio segurtasun eta pribatutasun komunitatean hainbat kezka sortu dira, izan ere, gailu horietako asko internetetik atzigarriak dira. Gailu horien segurtasun maila asko aldatzen da, segurtasun neurri gabeko mikrokontrolagailu txikienetatik 64 biteko helburu orokorreko prozesadoreetaraino. Gailu horietako askorentzat garatu nahi izanez gero IP eta SDK pribatuak erabili behar izaten dira, garatzaileak plataforma konkretuetara lotuz.

Helburu orokorreko prozesagaialuen munduan Linux eta BSD sistema eragileak erabili izan dira, askotan aldatuta tamaina eta errendimendu mugak bete ahal izateko. Sistema eragile horien gainean oso erraza da garatzea kontrolagailu asko dagoeneko inplementatuta daudelako.

Proiektu honen bitartez RISC-V ekosistema irekiaz baliatuta sistema eragile txiki eta arin bat sortzea zen helburu. Sistema eragile eta ekosistema honetaz baliatuz garatzaileek proiektuetarako behar dituzten pieza guztiak aldatu eta espezializatu ditzakete.

Hori guztia RISC-V agindu multzoa, OpenSBI firmwarea eta Xv6 sistema eragile aldatu baten bitartez lortu da. Xv6 sistema eragilea hezkuntzarako sortua izan denez, hardware aurkikuntza mekanismoak eta firmware erabilera faltan zituen; beraz, ezaugarri horiek gehitu behar izan dira.

6 Etorkizunerako planteatzen den norabidea

Proiektu honen bitartez RISC-V ekosistema jarraitzen duen sistema eragile baten oinarria sortu da. Sistema eragile hori alderdi askotatik garatu behar da erabilgarria kontsideratu baino lehen. Ondorengoak dira segitu behar diren pausuak: kontrolagailu sistema garatu, hardware errealean habiaratu, sistema eragile markoa bihurtu.

Lehen aipatu bezala, hardware-aren detekzioa egiterakoan begirale software-a erabiltzeko prest dagoen hardwarea bakarrik atzitzen da. Lan egiteko modu hori ez da egokiena, izan ere, sistema txertatuen munduan erabiltzaile softwareak begiraleak zuzenean kontrolatzen ez dituen gailuak ere erabili beharko ditu (*Firewall* motako sistema batek ethernet gailuak kontrolatu nahiko ditu trafikoa analizatu ahal izateko). Ondorioz, begirale softwarean kontrolagailuen azpiegitura bat sortu beharko litzateke eta hori erabiltzaile softwareari erakusteko modu bat.

Horrez gain, oso garrantzitsua da sistema txertatuetan erabiliko den sistema eragile bat benetako hardwarean erabilgarria izatea eta horretarako OpenSBI soportatzen duen edozein makinarekin hasi daiteke. Garrantzitsua da horrelako sistema batean martxan jartzeko hardwarea baliatuko duten kontrolagailu nahikoak daudela eta fitxategi sistema erabiltzeko gai dela ziurtatu behar da. Horretarako *eMMC* (jed, 2007) kontrolagailuaz baliatu daiteke sistema eragilea edo *initrd* motako RAM fitxategi sistema bat erabiliz. Gainera, *u-boot* motako hasieratze sistema erabili beharko da *OpenSBI* zuzenean erabili beharrean.

Azkenik, sistema eragile hori marko batean bihurtu behar da garatzaileak horien aplikazio propioak errazago sartu eta horien gailuetarako sistema eragile txertatua errazago garatu ahal izateko. Sistema eragile funtzionamenduaz aparte, sistema honen bigarren fokoa segurtasunean dago. Honen harira, RISC-V arkitekturak hainbat estentsio ditu segurtasunari lotuta, garrantzitsuenak *CFI* eta *Pointer Masking* izanik; *Morello* estentsioa ere garrantzitsua izan daiteke baina zailagoa da horren euskarria duen hardwarea aurkitzea.

Erreferentziak

2007. *Embedded MultiMediaCard (eMMC) eMMC/Card Product Standard, High Capacity, including Reliable Write, Boot, and Sleep Modes*. JEDEC.

2008. Ieee standard for floating-point arithmetic - redline. *IEEE Std 754-2008 (Revision of IEEE Std 754-1985) - Redline* 1–82.

2022. *Virtual I/O Device (VIRTIO) Version 1.2*. OASIS Open.

2024a. *ACPI Specification*. Unified Extensible Firmware Interface Forum.

2024a. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA*. RISC-V Unprivileged Horizontal Committee.

2024b. *The RISC-V instruction set manual volume II: Privileged architecture*. RISC-V Privileged Horizontal Committee.

2024c. *RISC-V Supervisor Binary Interface Specification*. RISC-V Privileged Software Horizontal Committee.

2024b. *UEFI Specification*. Unified Extensible Firmware Interface Forum.

Barry, Richard, eta others. 2008. Freertos. *Internet, Oct* 4.18.

Cox, Russ, M Frans Kaashoek, eta Robert Morris, 2011. Xv6, a simple unix-like teaching operating system.

7 Eskerrak eta oharrak

MICIU/AEI/10.13039/501100011033 erakundeak finantzatutako CNS2023-144315 dirulaguntzaren bidez eta ”European Union NextGenerationEU/PRTR”programaren bidez. Era berean, PID2023-152390NB-I00 dirulaguntza MICIU/AEI/10.13039/501100011033 erakundeak finantzatuta, ”FEDER funtsen”bidez. Dr. Javier Navaridas-ek Ramón y Cajal beka bat du, Espainiako Zientzia, Berrikuntza eta Unibertsitate Ministerioak (MCIN/AEI/10.13039/501100011033) finantzatua, eta, kasuan kasu, ”ESF Investing in your future”edo ”European Union NextGenerationEU/PRTR”programaren bidez babestua.