TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

NIILA-TINKA TUOMENTO
LINUX-AJUREIDEN TOTEUTUS YTIMEN MODUULEINA JA KÄYTTÄJÄTILASSA

Kandidaatintyö

Tarkastaja: Sakari Lahti

TIIVISTELMÄ

N**IILA-TINKA TUOMENTO:** LINUX-AJUREIDEN TOTEUTUS YTIMEN MODUULEINA JA KÄYTTÄJÄTILASSA

Tampereen teknillinen yliopisto
Kandidaatintyö, 21 sivua
Kesäkuu 2018
Tietotekniikan kandidaatin tutkinto-ohjelma
Pääaine: Ohjelmistotekniikka
Tarkastaja: Sakari Lahti

Avainsanat: Linux, ajuri, ohjain, moduuli, ydintila, kernel mode, käyttäjätila, user mode

Linux-ajureita voidaan valmistaa ja suorittaa sekä ydintilassa että käyttäjätilassa. Ongelmana on, ettei välttämättä tiedetä kumpi tapa olisi parempi ratkaisu laiteajurin valmistukseen. Työssä toteutettiin vertailu ydintilan ja käyttäjätilan ajurien tärkeimpien eroavaisuuksien välillä, ja analysoitiin niiden vaikutusta eri laiteajurityyppien valmistukseen. Havaittiin, että käyttäjätilan ajurit voivat olla rajoittuneempia, mutta moniin ongelmiin on kiertokeino. Kummallekin ajurityypille löytyi tapauksia, joissa ne ovat toista parempia.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

I/O	input/output
DMA	Direct Memory Access, tiedon oikosiirto laitteiston avulla
GPL	General Public Licence

1. JOHDANTO

Ajureita eli ohjaimia käytetään järjestelmään kytkettyjen laitteiden kanssa kommunikoimiseen tai muiden alhaisen tason toimintojen tekemiseen. Linux-käyttöjärjestelmässä ajureita voidaan suorittaa sekä käyttöjärjestelmän ytimen sisällä ydintilassa, jolloin niillä on suora pääsy järjestelmäytimen tarjoamiin palveluihin, että kokonaan tai osittain käyttäjätilassa muiden ohjelmien kanssa. Ohjelmoijan täytyy päättää ennen ajurin valmistamista, kummalla tavalla hän sen tekee. Kummallakin toteutustavalla on omat etunsa ja ne vaihtelevat esimerkiksi sen mukaan, millaista laitetta ajurilla halutaan ohjata.

Työssä vertaillaan Linux-käyttöjärjestelmän laiteajurien toimintaa ja toteutusta ydintilassa moduuleina ja käyttäen käyttäjätilaa. Työn tarkoituksena on, että lukijalle jää selkeä kuva kummankin ajurityypin rajoituksista ja soveltuvuudesta eri käyttötarkoituksiin. Työn perusteella lukija osaa tehdä paremmin päätöksiä ajurin alkuvaiheen toteutussuunnittelusta ja toteutustavan valinnasta. Ajureita ja moduuleita käytetään myös muihin tarkoituksiin kuin laitteiden ohjaukseen, mutta tässä työssä keskitytään laiteajureihin. Linuxin tukee useita erilaisia suoritinarkkitehtuureja, mutta niiden määrän vuoksi työn tiedonhaussa on keskitytty ensisijaisesti x86-arkkitehtuuriin.

Luvussa 2 käsitellään ensin yleisellä tasolla ajurien merkitys sekä Linux-käyttöjärjestelmän ydin- ja käyttäjätilan ero. Ajurien toiminta ytimen moduulina ja käyttäjätilassa esitellään perustasolla. Kolmannessa luvussa vertaillaan ytimen ajurin ja käyttäjätilan ajurin toimintaa eri näkökulmista. Lukuun on etsitty kirjallisuudesta eroavaisuuksia ajurityyppien välillä ja keinoja toteuttaa ne. Perustoiminta-luvussa esitellään ajurin asennusta ja lataamista. Ohjelmointi ja virheenetsintä -luku keskittyy ajurikoodin valmistuksen yksityiskohtiin. Yhteistoiminta ja rajapinnat käsittelee ajurin ohjelmistorajapintoja, muistinhallinta ja laitteisto-I/O alhaisemman tason toimintaa laitteiston ja muistin kanssa. Koska ohjelmiston lisenssi vaikuttaa olennaisesti valmiin ohjelman jakeluun, lisenssirajoitukset esitellään myös omassa luvussaan. Luvun lopuksi toteutetaan vertailu löydettyjen eroavaisuuksien pohjalta ja analysoidaan niiden vaikutusta erityyppisten ajurien valmistukseen. Lopuksi neljännessä luvussa tehdään yhteenveto työssä tehdyistä havainnoista.

2. AJURIEN TOIMINNAN TAUSTAA

2.1 Ajurit yleisesti

Tietokoneen käyttöjärjestelmällä täytyy olla keino pitää yhteyttä siihen kytkettyihin laitteisiin. Laitteiden suuren määrän ja erilaisten rajapintojen vuoksi niiden toiminnallisuuden kattamiseen tarvitaan käyttökelpoinen järjestelmä. Linuxissa tämä järjestelmä on ajurit eli ohjaimet, jotka yhdistävät käyttöjärjestelmän ja laitteiston. Ne kommunikoivat laitteiden kanssa ja toteuttavat tiedon siirrännän (engl. input-output) laitteisiin tai laitteista erilaisten rajapintojen kautta. Laitteiden toiminnallisuus vaihtelee paljon, eikä niille ole yhtä yhteistä rajapintaa. Ajurit ovat usein laitemallikohtaisia ja toteuttavat rajapinnan laitteen toimintojen ja Linux-ytimen tai sovellusten yleiskäyttöisempien rajapintojen välillä. Tavallisesti laitevalmistajat tai käyttöjärjestelmän ylläpitäjät valmistavat ajurit kutakin käyttöjärjestelmää varten. [25, luku 6]

Käyttöjärjestelmässä voi olla monentasoisia ajuriohjelmia samaan aikaan, joista osa on laitteistoläheisempiä ja osa yleiskäyttöisempiä. Esimerkiksi USB-järjestelmässä voi olla alemman tason laitteistokohtainen USB-ajuri, joka on yhdistetty ylemmän tason lohkolaiteajuriin [11, luku 8].

2.2 Ydintila ja käyttäjätila

Linux-käyttöjärjestelmä jakaa toimintansa kahteen eri virtuaaliseen osoiteavaruuteen: ydintilaan ja käyttäjätilaan. Esimerkiksi 32-bittinen järjestelmä antaa ydintilan käyttöön muistiavaruuden ylimmän gigatavun ja käyttäjätilalle alimmat kolme gigatavua. Tällä jaolla varmistetaan, etteivät tavalliset ohjelmat pääse sekoittamaan käyttöjärjestelmän tai toistensa tilaa. Ydintilan koodi on osa käyttöjärjestelmää, ja sitä käytetään käyttöjärjestelmän toimintojen hallintaan. Ydintilassa suoritettavalla ohjelmakoodilla on suurimmat käyttöoikeudet, ja se pääsee käsiksi tietokoneen koko muistiin sekä laitteiston rekistereihin. Ydintilan muistiavaruuden sisältö pysyy samana ja käyttäjätilan muistiavaruus vaihtuu kullakin hetkellä suoritettavan prosessin mukana. Suoritin vaihtaa suoritustilan käyttäjätilaksi siksi ajaksi, kun käyttäjäohjelmaa suoritetaan, ja päinvastoin. [24, s. 18, 257-258]

Käyttäjätilassa suoritetaan tavalliset tietokoneen käyttäjän ohjelmat. Käyttäjätilassa suoritettavilla ohjelmilla ei tavallisesti ole pääsyä toistensa tietoihin tai suoraa pääsyä laitteiston rekistereihin. Käyttäjätilan ohjelmien ei sallita suorittaa tiettyjä toimintoja, kuten suorittaa laitteen mallikohtaisia rekistereitä muuttavia käskyjä [8, s. 173] tai lukea ytimen muistialueita. Käyttäjätilan ohjelman täytyy käyttää käyttöjärjestelmäytimen tarjoamaa rajapintaa, jos se haluaa suorittaa ytimen toimintoja.

2.3 Ajurit ytimen moduulina ja käyttäjätilassa

Linux-ajureita voi ladata osaksi ydintä, jolloin niillä on samat oikeudet ja mahdollisuudet suorittaa toimintoja kuin käyttöjärjestelmäytimen koodilla. Ajurin ohjelmakoodi luetaan sopivalta tallennusmedialta ja ladataan muistiin. Linux-ydin suorittaa ajurille tarvittavat alustustoimenpiteet, ja se alkaa suorittamaan toimintojaan. Ajuria, joka näin ladataan osaksi ydintä, kutsutaan moduuliksi. Moduuleja voidaan asentaa ja poistaa ytimen suorituksen aikana, tai niitä voidaan kääntää pysyvästi osaksi ytimen koodia. Moduuleja voidaan pinota päällekkäin ja käynnistää rinnakkain, mikäli ne muodostavat keskenään riippuvaisuuksia. Ydin huolehtii moduulien linkittämisestä keskenään moduulin metatietojen perusteella. [25, luku 7]

Käyttäjätilassa suoritettava ajuri käyttäytyy kuten mikä tahansa muu käyttäjän ohjelma. Käyttäjätilan ajurilla ei ole pääsyä järjestelmän eri osiin samalla tavalla kuin ydintilan ajurilla. Se pääsee käsiksi laitteistoon erityisten Linux-rajapintojen avulla tai käyttämällä pientä ydintilassa toimivaa sovitinmoduulia. [19, s. 2-3]

3. OHJAINTYYPPIEN VERTAILU

3.1 Ajuri ytimen moduulina

Luvussa esitellään ydintilan ajurille ominaisia toteutustapoja ja käytäntöjä.

3.1.1 Perustoiminta

Ytimen moduulin ajureita voidaan ladata osaksi ydintä käyttämällä Linuxin siihen tarjoamaa rajapintaa. Ydin tarjoaa järjestelmäkutsut init_module() ja delete_module() moduulien lisäystä ja poistoa varten [25, luku 7.3]. Käyttäjä voi hallita ytimen moduulien asennusta komentoriviltä esimerkiksi komennolla *modprobe asennettavan_moduulin_nimi* tai *modprobe -r poistettavan_moduulin_nimi*, jotka vastaavasti lataavat ja poistavat moduulin [20]. Ajuri voidaan asettaa latautumaan automaattisesti, jolloin se latautuu joko osana käyttöjärjestelmän käynnistystä tai kun sitä vastaava laite liitetään järjestelmään. Moduulin sisältävä .ko -tyypin tiedosto pitää ensin tallentaa tiettyyn kansioon, joka on esimerkiksi */lib/modules/ytimen_versio/* . [21] Ydin tuo kobject-tietotyyppien tietoja, kuten viittauslaskurin, näkyville käyttäjätilaan /sysfs/-tietojärjestelmän kautta. [25, s. 366][29, luku 10.3]

Ydin selvittää moduulin riippuvuudet ja linkittää sen oikeaan kohtaan ajuripinoa käyttämällä moduulin osaksi käännettyä metatietoa. Metatiedon avulla tunnistetaan ajurin kohdelaite. Ydin alustaa ajurin lataamalla sen muistiin ja kutsumalla automaattisesti sen init_module()-funktiota. Ajuri voidaan poistaa ytimestä käyttämällä funktiota cleanup_module(). [25, luku 7.2]

3.1.2 Ohjelmointi ja virheenetsintä

Ohjaimet toteutetaan tavallisesti C- ja assembly -ohjelmointikieliä käyttäen. Ydintilan ajureiden täytyy noudattaa tiettyä standardia rakennetta (kuten esimerkiksi toteuttaa edellisen kappaleen lopussa mainitut funktiot), jotta ydin voi käsitellä ajuria oikein. Ydintilan ajureiden ohjelmointiin on saatavilla kehitysympäristöjä, jotka luovat valmiiksi ajuriohjelman vaatiman pohjarakenteen. Tällainen on esimerkiksi Eclipse Link+ [22].

Virheenetsintä ydintilassa on monin tavoin haastavampaa kuin käyttäjätilassa. Mikäli ajurin suoritus keskeytetään, keskeytyy samalla myös koko ytimen suoritus, mikä tarkoittaa käyttöjärjestelmän pysähtymistä. Ohjelmakoodi on myös ladattuna johonkin satunnaiseen osoitteeseen. Linux-ydin ilmoittaa OOPS-virheilmoituksilla, kun ohjelmavirhe tapahtuu ytimessä. OOPS-ilmoitukset sisältävät virheen tapahtumisosoitteen ja syyn sekä tietoa prosessorin tilasta [29, s. 94]. OOPS-viestit ja muut tulosteet ajureista ohjautuvat konsoliin, esimerkiksi sarjaportin kautta, ja viestin tärkeysasteesta riippuen tallentuvat

/var/log/ -kansioon [29, s. 76]. Tehokkaaseen virheenetsintään suorituksen aikana täytyy käyttää erillistä tietokonetta tai virtuaalikonetta, jossa ytimen ja moduulin koodi suoritetaan, ja liittää kone vapaasti saatavilla olevaan KDB- tai KGDB-virheenetsintätyökaluun. [30, s. 378-380] User-Mode Linux on Linux-versio, jossa ydintä suoritetaan käyttäjätilan ohjelmana. Se mahdollistaa virheenetsinnän käyttäen samoja käyttäjätilan työkaluja kuin tavallisten ohjelmien kanssa.[29, s. 104]

Jotta ajurin ohjelmakoodi voidaan kääntää käyttöjärjestelmäytimen kanssa yhteensopivaksi, käännösvaiheessa tarvitaan symboleja. Symbolit sisältävät tiedon tietyn ydinversion muuttujien tai funktioiden nimistä ja niiden vastaavista sijainneista ohjelmakoodissa. Virheenetsinnässä symboleja käytetään koodin suorituspolun hahmottamiseen. [29, s. 28] Symbolit ovat saatavilla useille Linux-jakeluversioille pakettienhallinnan kautta, esimerkiksi Debianille[3]. Kun halutaan julkistaa ajurin omia funktioita muiden käytettäviksi, käytetään EXPORT_SYMBOL-makroa [29, s. 29].

Koska ydinmoduulit suoritetaan samassa tilassa kuin itse käyttöjärjestelmän hallinta, moduulien virheet voivat vaikuttavat koko järjestelmään, jumittaen tai kaataen sen. Kaatumisen estoon on kehitetty joitain kolmannen osapuolen järjestelmiä, kuten esimerkiksi Nook [31], joka eristää ytimen moduulit, tai varjoajuri(shadow driver) [32, s. 2], joka on yksinkertaistettu versio ajurista ja toteuttaa toiminnot, joita tarvitaan sillä aikaa kun ajuri käynnistetään uudelleen. Ne eivät kuitenkaan ole osana tavallista ydintä.

3.1.3 Ajurien yhteistoiminta ja rajapinnat

Linux jakaa ytimen ajurit karkeasti kolmeen eri luokkaan. Nämä ovat merkkipohjaiset ja lohkopohjaiset ajurit sekä verkkoliitäntäajurit. Merkkipohjaisia ajureita käytetään hallitsemaan laitteita joita voi ohjata bittivirtojen avulla. Lohkopohjaiset ajurit toteuttavat tiedonsiirron yksittäisinä datalohkoina. Ajuriluokille on omanlaisensa ydinrajapinnat, mutta merkki- ja lohkopohjaiset ajurit näyttäytyvät samanlaisina käyttäjätilaan. Laitteita voidaan käyttää kirjoittamalla ja lukemalla tiedostojärjestelmän laitetiedostojen kautta. [29, s. 5-7] Verkkolaitteet ovat erityinen luokka siinä mielessä, että ne vastaanottavat tietoa järjestelmän ulkopuolelta ja joutuvat kommunikoimaan ytimen kanssa, kun verkkopaketteja vastaanotetaan. Lohko- ja merkkipohjaisten laitteiden kohdalla tiedonsiirtopyynnöt tulevat ytimeltä päin. Verkkoliitäntäajureilla ei ole laitetiedostoa tiedostojärjestelmässä, vaan erityinen rajapinta ytimen kanssa. [29, s. 497]

Ydintilan ja käyttäjätilan väliseen yhteydenpitoon on laajat, mutta säännellyt mahdollisuudet. Ajurit julkistavat funktioitaan järjestelmäkutsuina. Niiden avulla käyttäjätilan ohjelma voi kutsua ajurin toimintoa. Järjestelmäkutsut suoritetaan kutsuvan ohjelman kontekstissa, mikä tarkoittaa, että kutsufunktiolla on pääsy sekä ytimen että kutsuvan ohjelman muistiavaruuteen. Järjestelmäkutsujen käyttöön liittyy suorituskykyhaittoja, sillä järjestelmän pitää vaihtaa käyttäjätilasta ydintilaan ja takaisin. [12, s. 43-44]

Yleisimmät järjestelmäkutsut read() ja write() esitetään laitetiedostojen kautta. Linux pyrkii esittämään kaikki järjestelmän objektit kuten laitteet ja niiden ajurit tiedostojärjestelmän kautta. Merkki- ja lohkopohjaiset laitteet ottavat tavallisesti käyttäjätilan tietoa vastaan /dev/-kansion laitetiedostojen avulla. Esimerkiksi, käyttäjätilan ohjelman avulla sarjaportin numero 0 tiedostosta /dev/ttyS0 voidaan lukea sarjaporttiin saapuva data. Samaan tiedostoon kirjoittamalla sarjaporttiajuri lähettää kirjoitetun datan sarjaporttiin. Verkkoliikenneajureille ei ole laitetiedostoa niiden erityisasemasta johtuen, vaan niitä käsitellään vain socket-järjestelmäkutsujen avulla. [25, luku 6.2] Aiemmin luvussa 3.1.1 mainittu sysfs-tiedostojärjestelmä sallii myös ajurin parametrien lukemisen ja kirjoittamisen, jos ajuri tukee sellaista.

Kaikille laiteajurien toiminnoille ei välttämättä ole sopivaa järjestelmäkutsua, mutta laitekohtaisen ioctl-komennon avulla voidaan kutsua käyttäjätilasta laitteen tiettyä toimintoa. Käyttäjäpuolen ohjelman pitää tietää kutsussa välitettävän toimintopyynnön tunnuskoodi. [24, s. 112-115]

Ajureita voidaan myös pinota toistensa päälle ja yhdistää toisiinsa käyttämällä symboleilla näkyville tuotuja funktioita [29, s. 28], jolloin muodostuu riippuvaisuussuhteita. Yhdessä ajurit muodostavat puumaisen ajurihierarkian. Ytimen sisältämät alemman tason (esim. väylä-)ajurit ja palvelut toimivat pohjana, jonka päälle oman ajurin voi rakentaa. Ytimen sisällä on runsaasti tukikirjastoja erilaisten laitetyyppien hallintaan, kuten esimerkiksi PCI Support Library tai USB Gadget API. Ydintilasta käsin on myös mahdollista käynnistää käyttäjätilan ohjelma funktion call_usermodehelper [2] avulla.

Linux-ytimen sisäiset rajapinnat eivät kuitenkaan ole pysyviä, vaan muuttuvat ydinversioiden välillä [14]. Muuttuvan rajapinnan kanssa toimimista helpottaa esimerkiksi DKMS-työkalu [5]. Se kääntää moduulit automaattisesti uudestaan, jos ydin päivitetään toiseen versioon. Vaatimuksena on moduulin lähdekoodi ja kääntämiseen kuluva aika. [6] Linux- ytimen ja ajurien tukikirjastojen rajapinnan muuttumista eri versioiden välillä ovat aiemmin tutkineet mm. Y. Padioleau et al., julkaisussaan Understanding Collateral Evolution in Linux Device Drivers[27].

Rinnakkaisuus ydintilassa monimutkaistaa ohjaimen rakennetta ja yhteistoimintaa. Ydin sisältää jo ennestään rinnakkaisesti suoritettavia toimintoja ja myös ajurit voivat käynnistää ydintilassa toimivia säikeitä. Erilaisten rinnakkaisesti tapahtuvien toimintojen ohjaamiseen voidaan käyttää semaforeja, mutexeja ja lukkoja. Ytimessä voidaan suorittaa jakamattomia, kerralla loppuun suoritettavia atomisia operaatioita. Ajurit voivat jakaa resursseja ja muistia keskenään. Käyttäjätilan ohjelmat, samoin kuin keskeytykset, saattavat kutsua ajurin toimintoja samanaikaisesti ja nopeasti peräkkäin. Edellisistä johtuen ajurin funktioiden täytyy olla uudelleenkutsuttavia(reentrant), jotta niiden tila ei sekoa. [29, s. 20-21].

Tehtävien ajastusta varten ajuri pystyy asettamaan erityisesti muotoiltuja funktioita työjonoon odottamaan myöhempää suoritusta. [11, luku 4.5] Säie voi asettua nukkumaan odottaakseen toisen valmistumista, kunhan se ei ole lukkiutuneena odottamassa jotain resurssia tai poistanut keskeytyksiä käytöstä. Nukkuva säie ei kuluta suoritinaikaa. [29, s. 148]

Ydin tarjoaa ajurin toimintojen ajastamiseen eri tarkkuuksisia ajastimia. Alhaisen tarkkuuden ajastimia on jiffies, ”hetki”, jonka arvoa päivitetään keskeytyksillä esimerkiksi 250 kertaa sekunnissa. Korkean tarkkuuden ajastimilla voidaan mitata ajankulua jopa nanosekuntien luokassa. [25, luku 15.2-15.4][30, s. 426]

3.1.4 Muistinhallinta ja laitteisto-I/O

Koska ytimen koodilla on täydet suoritusoikeudet, ajureilla on täysi pääsy laitteen muistiin. Ydin antaa ajurille useita keinoja erilaisten muistialueiden hallintaan. Esimerkkeinä voidaan mainita vmalloc ja kmalloc -funktiot, jotka varaavat yhtenäisen virtuaalimuistin alueen. Kmalloc:in varaama alue on myös fyysisessä muistissa yhtenäinen. Malloc-pohjaiset toiminnot voivat epäonnistua, jos (yhtenäistä) muistia ei ole saatavilla. Kmem cache varaa automaattisesti useita samanlaisia muistirakenteita jolloin niiden alustaminen myöhemmin on nopeaa. [11, luku 10.1]

Yhtenäistä fyysistä muistialuetta voidaan tarvita esimerkiksi DMA-tiedonsiirtoon. DMA tarkoittaa laitteistopohjaista suoraa tiedonsiirtoa. Ajuri asettaa DMA-siirrolle lähde- ja kohdeosoitteen, ja tiedonsiirto käynnistyy automaattisesti keskeytyksen tai manuaalisen käynnistyksen avulla. [29, s. 441]

Datan välittämiseen käyttäjä- ja ydintilan osoiteavaruuksien välillä voidaan käyttää copy_to_user ja copy_from_user -funktioita, jotka kopioivat datan haluttujen käyttäjä- ja ydintilan puskureiden välillä [11, luku 2.1]. Kokonaisen fyysisen muistialueen voi siirtää/kuvata(map) suoraan käyttäjätilan ohjelman muistiavaruuteen funktiolla remap_pfn_range tai mmap, ja I/O-muistialueen funktiolla io_remap_pfn_range [24, s. 295-299].

Laitteiston kanssa voidaan kommunikoida niiden I/O-porttien kautta. Ajuri saa I/O-portin käyttöönsä varaamalla sen funktiolla request_region. I/O-portit voivat olla myös kuvattuna keskusmuistiin, jolloin voidaan siirtää dataa suoraan muistiosoitteisiin porttiluvun ja -kirjoittamisen sijaan. [24, s. 290-293] Tietokoneeseen liitetty laite ei ole välttämättä aina valmiissa tilassa, eikä I/O-operaatio välttämättä onnistu heti kun sitä pyydetään. Toimintoa pyytävä käyttäjätilan ohjelma ei oleta, että operaatio palautuisi vasta pidemmän ajan päästä, vaan saattaisi esimerkiksi kutsua samaa järjestelmäkutsua toisesta säikeestä. Kaikki järjestelmäkutsut eivät tue tätä ja siksi ajurin täytyy ottaa vastuuta myös käyttäjätilan ohjelman toiminnasta. Ajurin vastuulla on laittaa kutsujaohjelma nukkumaan ja herättää se vasta kun operaatio on valmistunut. [29, s. 147-148]

Keskeinen osa laitteiston ja ohjelmiston välistä kommunikointia ovat keskeytykset. Keskeytyksillä laitteisto ilmoittaa jostakin huomiota vaativasta tapahtumasta, johon järjestelmä vastaa kutsumalla vastaavaa keskeytyskäsittelijää tai jättämällä sen huomiotta. Ajuri voi vastaanottaa keskeytyksen rekisteröimällä sen numeroa vastaavan keskeytyskäsittelijän. [24, s. 79] Keskeytyskäsittelijän pitää varmistaa, ettei se turhaan hylkää sen suorituksen aikana saapuvia keskeytyksiä, eikä myöskään keskeydy kun operaatio sen käyttämällä jaetulla datalla on kesken [24, s. 82-83][29, s. 273]. Keskeytysten sijaan ajuri voi käyttää myös pollausmenetelmää, eli lukea tietyin väliajoin esimerkiksi jotain porttia.

3.1.5 Lisenssirajoitukset

Linux-ytimen sisäistä APIa käyttävä jaeltava ohjelma voidaan tulkita ytimestä periytyväksi tuotokseksi, jolloin sen pitää käyttää vapaan lähdekoodin GPL-yhteensopivaa lisenssiä [7]. Jos Linux-ytimen ajuri ei käytä GPL-yhteensopivaa lisenssiä, ydin ilmoittaa ulkopuolisille tilansa olevan *tainted*. Tämä kertoo ytimen kehittäjille, että siihen on ladattu ajuri jolle he eivät välttämättä voi suorittaa virheenetsintää ja -korjausta ohjelmakoodin puuttuessa. Muun kuin GPL-yhteensopivan lisenssin käyttäjälle tästä on haittana, että avun saaminen ajurin virheenetsintään hankaloituu. Ydin voi myös piilottaa osan symboleistaan GPL-yhteensopimattomien ajureiden nähtäviltä, jolloin ne eivät voi käyttää kaikkia ytimen toimintoja [24, s. 27]. Ajurin valmistaja merkitsee ajurin metatietoon, millaista lisenssiä se käyttää. Mikäli tietoa ei ole annettu, Linux-ydin olettaa ajurin olevan suljettua lähdekoodia. [29, s. 30][33]

Tunnetaan poikkeustapaus, jossa Canonical päätti sisällyttää zfs-tiedostojärjestelmän moduulin Ubuntu Linux -jakeluunsa, vaikkei moduuli ollut yhteensopiva Free Software Foundationin GPL-lisenssin kanssa. Zfs käytti avoimen lähdekoodin CDDL-lisenssiä, mutta Canonical päätyi tulokseen ettei se ole ristiriidassa Linux-ytimen GPL-lisenssin kanssa eikä zfs-moduuli ole Linux-ytimestä periytyvä tuotos. [13]

3.2 Ajuri käyttäjätilassa

Luvussa esitellään käyttäjätilan ajurille ominaisia toteutustapoja ja käytäntöjä.

3.2.1 Perustoiminta

Ajuri voidaan käynnistää tavallisena käyttäjätilan prosessina, daemon-prosessina tai kirjaston, jota ohjelmat voivat kutsua. Daemon-prosessi eroaa tavallisesta siten, että se toimii taustalla eikä siihen ole liitettynä terminaalia, jonka kautta sitä voisi kontrolloida ja sulkea

[12, s. 767]. Prosessi voidaan asettaa käynnistymään automaattisesti lisäämällä sen käynnistysskripti osaksi Linuxin käynnistysjärjestelmää. Käynnistysjärjestelmä (init tai systemd) vaihtelee Linux-jakeluversioittain, mutta yleisimmin käynnistysskripti lisätään /etc/init.d- tai /etc/inittab -tiedostoon. [30, s. 265-275] Käynnistysskriptiä voi toki kutsua jossain myöhemmässäkin vaiheessa käynnistystä, tai ytimen apumoduulista tai muusta osasta, kun havaitaan että haluttu laite on liitetty.

Käyttäjätilan ajurit voidaan saada käynnistymään automaattisesti, kun laite liitetään, lisäämällä käynnistysskripti udev-sääntötiedostoon. Udev-palvelu havaitsee kun uusi laite liitetään, etsii sääntötiedostosta (esim. /etc/udev/rules.d) jonkin tunnisteen joka vastaa liitettyä laitetta ja suorittaa käynnistysskriptin.[37]

Joskus ajurissa on jokin yksittäinen toiminto, joka voitaisiin toteuttaa selkeästi paremmin ydintilassa. Tässä tapauksessa voidaan tehdä eräänlainen hybridiajuri, jossa jokin yksittäinen toiminto on pienessä ydintilan sovitinmoduulissa ja suurin osa toiminnallisuudesta on käyttäjätilassa. Käyttäjätilan ajuri voi kutsua tätä ydintilan ajuria esimerkiksi laitetiedoston tai järjestelmäkutsujen kautta. Ydintilan sovitinmoduulia koskee samat vaatimukset kuin luvussa 3.1.1. Käyttäjätilan ajurin omalla vastuulla on yhdistyä siihen ja kutsua sen käyttämiä rajapintoja.

3.2.2 Ohjelmointi ja virheenetsintä

Niin kuin käyttäjätilan ohjelmien ohjelmoinnissa tavallisestikin, ohjelmoija voi käyttää vapaasti erilaisia ohjelmointikielten tarjoamia kirjastoja kuten GNU C -kirjastoa ja muita apukirjastoja. Ajurin rakenteelta ei vaadita erityispiirteitä, mutta sen täytyy käyttää seuraavassa luvussa esiteltäviä rajapintoja, jos se haluaa ottaa yhteyttä ytimeen ja laitteistoon. Ohjelmointikielen valinta on vapaa, kunhan se tukee rajapinnan - kuten järjestelmäkutsujen - käyttöä.

Kehitykseen ja virheenetsintään voi käyttää yleisesti saatavilla olevia työkaluja, kuten gcc-kääntäjää ja gdb-debuggeria. Silti voi olla hyödyllistä seurata tiedonvälitystä myös ytimen puolelle ja laitteelle asti, jolloin tarvitaan ytimeen liitettävää työkalua, niin kuin luvussa 3.1.2.

Ajurin virheet eivät aiheuta Linuxissa laaja-alaista haittaa, koska käyttäjätilan ohjelmat toimivat kukin omassa muistiavaruudessaan erillään ytimestä ja toisistaan. Niillä ei ole normaalisti keinoa vaikuttaa muihin prosesseihin. [23, luku 6.1] Siksi Linux pystyy lopettamaan kaatuneen tai virheellisesti käyttäytyvän ohjelman ilman että sen pitäisi vaikuttaa järjestelmän vakauteen, kunhan laitteisto- ja ydinrajapinnat toimivat oikein. Järjestelmän vaste voi toki huonontua, jos ajuri varaa liiallisesti jotain resurssia, kuten muistia.

Koska normaalin ohjelman mahdollisuudet suorittaa ytimelle etuoikeutettuja toimintoja ovat rajoitetut, ajuri saattaa tarvita erityisoikeuksia. Jos ajurin suorittaa pääkäyttäjän root-

oikeuksien avulla, se saa kaikki käyttöoikeudet, mutta on vakauden ja tietoturvan kannalta suositeltavaa antaa ohjelmalle vain sen toiminnan vaatimat kyvyt Linuxin capabilities-toiminnon avulla. Esimerkiksi kyky CAP_SYS_MODULE sallii moduulien lataamisen ytimeen. [12, s. 797,801] Täysien pääkäyttäjäoikeuksien kanssa toimiva ohjelma pystyisi virhetilanteessa esimerkiksi tuhoamaan järjestelmän tiedostoja.

3.2.3 Ajurien yhteistoiminta ja rajapinnat

Linux Userspace API (uapi) eli käyttäjätilan järjestelmäkutsurajapinta on vakaa ydinversiosta riippumatta [14]. Linux-jakeluversion vastuulla on pitää huolta binäärisestä yhteensopivuudesta, eli siitä että ohjelmat voidaan suorittaa jakeluversiossa ilman uudelleenkääntämistä lähdekoodista. Tämä johtuu siitä, että eri jakeluversiot saattavat kääntää ja sisällyttää erilaisia paketteja ja palveluita järjestelmäänsä. [12, s. 20]

Userspace API muodostaa laajan rajapinnan joka sallii käyttäjäohjelman kontrolloida osaa laitteistosta ytimen välityksellä. Rajapinnan tukemat järjestelmäkutsut löytyvät Linux-lähdekoodin include/uapi-otsikkotiedostoista [38]. UAPI:n lisäksi saatavilla on myös muita kolmannen osapuolen rajapintakirjastoja, joista osa vaatii asentamisen ennen käyttöä. Jos järjestelmästä löytyy ennestään jonkinlainen tuki laitteelle, voidaan käyttää laitetiedostoja /dev/ - tai /sysfs/ -tiedostojärjestelmässä, tai ilman erityistä laitetukea voidaan kirjoittaa suoraan tietokoneen portteihin. Tätä alhaisen tason porttirajapintaa käsitellään enemmän luvussa 3.2.4. Jos valmiit ajurit tarjoavat jonkin sopivan toiminnon järjestelmäkutsuilla, laitetiedostoilla tai muilla keinoilla, niitä voi käyttää oman ajurin pohjana. Lisäksi ajurin tyypistä riippuen voidaan hyödyntää kirjastoja jotka sallivat muuttaa jotain tunnetun laitteen parametria tai asetusta, mutta eivät anna suoraa pääsyä laitteeseen.

Käyttäjätilan rajapinnat, jotka sallivat yhteyden laitteistoon, ovat selkeitä kokonaisuuksia ja ne on usein toteutettu kirjaston avulla. Seuraavalla sivulla Taulukossa 1 on listattu muutamia tunnettuja käyttäjätilan ajurin rajapintoja ja huomioita niiden käyttötavasta sekä tarkoituksesta.

Taulukko 1: Käyttäjätilan ajurien rajapintoja

Rajapinta	Selitys
Userspace API	Useita rajapintoja eri tarkoituksiin: mm. firewire, gpio, industrial io-tapahtumat, CAN-väylä ja komentojen lähettäminen verkkolaitteisiin. [38] Pari suurempaa kokonaisuutta esitelty erikseen alla.
libdrm (osa UAPIa)	Direct Rendering Manager tukee mm. AMD:n, Nvidian ja Intelin näytönohjaimia ja sallii käyttäjätilan ohjelman mm. lähettää komentoja, hallita muistia ja suorittaa DMA-operaatioita. [4]
libusb (osa UAPIa)	Tuo USB-laitteiden toiminnallisuutta käyttäjätilaan. Voidaan käsitellä dataa ja USB-laitteen ominaisuuksia kaikilla USB-siirtomuodoilla. LGPL-lisenssi. [18]
libevdev	Kirjasto tuo laitteiston tapahtumalaitteiden (event device) ioctl-komennot käyttäjätilaan. X11-lisenssi. [15]
libinput	Tuo kaikkien syötelaitteiden tapahtumat, kuten esimerkiksi tiedot näppäimen painalluksista saataville käyttäjätilaan. Liittyy läheisesti libevdeviin. MIT-lisenssi. [17]
Userspace I/O	Vaatii pienen ydintilan ajurin, jonka avulla UIO-laitteen keskeytykset tuodaan laitetiedoston kautta näkyville käyttäjätilaan. [35] Toiminta on selostettu tarkemmin luvussa 3.2.4.
CUSE(FUSE)	FUSE-rajapinta sallii käyttäjätilan ohjelmien hallita tiedostojärjestelmiä. CUSE(character device in userspace) on sen lisäys, jonka avulla merkkipohjaisia laitteita voidaan luoda käyttäjätilassa. CUSE luo laitetiedoston ja kutsuu käyttäjätilan ohjelmaa toteuttamaan sen toiminnot, kuten lukemisen, kirjoittamisen ja ioctl:n. [16]
BUSE	BUSE(block device in user space) on kolmannen osapuolen kokeellinen rajapinta lohkolaitteiden käyttöön käyttäjätilassa. Se vastaa toiminnaltaan CUSE:a. Toteutettu NBD(network block device)-ajurin avulla.[1]

Jos halutaan, että ajuri vastaa muiden käyttäjätilan prosessien pyyntöihin, sen täytyy luoda sitä varten jokin rajapinta. Ioctl ja muut järjestelmäkutsut eivät ole käytettävissä, koska ne toimivat vain ytimen ja käyttäjätilan välillä. Samoin laitetiedostot yhdistettiin suoraan ydintilan ajureiden komentoihin. Jos halutaan laitetiedostojen kaltaista toiminnallisuutta, voidaan luoda yksisuuntaiseen viestintään soveltuva nimetty putki (FIFO) [12, s. 906-908] tai kaksisuuntaiseen, usean prosessin väliseen viestintään soveltuva nimetty pistoke (socket) [12, s. 1150,1166-1167]. Ne luovat viestintäkanavan johon voi kirjoittaa ja lukea tiedostojärjestelmän kautta.

Lisäksi voi käyttää esimerkiksi CUSE-rajapintaa luomaan laitetiedoston käyttäjätilan ohjelmalle (ks. taulukko 1). Ajuri voi asentaa oman versionsa jostain yleisestä kirjastosta. Itse ajuri voidaan toteuttaa kirjastona, jota laitetta käyttävien sovellusten pitää kutsua.

Käyttäjätilassa ei ole ajurien välistä puuhierarkiaa niin kuin ydintilassa. Jos ajureita halutaan yhdistää toisiinsa, ne voidaan linkittää tai sitten käyttää jotain standardia viestintäkeinoa. Jos ajuri koostuu useammasta prosessista, niiden väliseen viestintään voidaan käyttää prosessienvälisen kommunikoinnin(IPC) menetelmiä [12, s. 877-878]. Näitä ovat esimerkiksi SYSTEM V -viestit, muistin jakaminen, putket (pipe) ja signaalit. Signaalia käyttämällä voidaan esimerkiksi vastaanottaa tieto ajastimen laukeamisesta tai käskeä prosessia sulkeutumaan. Vastaanottava prosessi kaappaa signaalin signaalinkäsittelijänsä avulla. [12, s. 388-390]

Useat ohjelmat saattavat kutsua ajurin toimintoja samanaikaisesti ja yrittää käyttää sen laitetta samanaikaisesti. Jos laite ei tue useita samanaikaisia pyyntöjä, ajurin vastuulla on siirtää pyynnöt eteenpäin vasta edellisen valmistuttua. Rinnakkaisuuden hallintaan käytetään normaaleja rinnakkaisen ohjelmoinnin keinoja, kuten semaforeja, mutexeja, lukkoja ja atomisia operaatioita. Ajuri voi käyttää ajastimia, mutta mm. ytimen ohjelmajärjestelijä ja keskeytykset aiheuttavat ajastukseen viivettä, eikä se välttämättä herätä prosessia juuri sillä millisekunnilla kuin on pyydetty. Jos käytössä on tosiaikainen (real-time) ydin, se tarjoaa vakuuksia suoritusajoista, mutta muuten käyttäjätilan ohjelma ei voi olla varma kauanko jonkin järjestelmäkutsun suoritus kestää [30, s. 416-419].

3.2.4 Muistinhallinta ja laitteisto-I/O

Kuten aiemmin mainittiin, käyttäjätilan ajuri toimii virtuaalisessa muistiavaruudessaan, eikä pääse suoraan käsiksi muille varattuun muistiin. mmap-järjestelmäkutsua käyttämällä se voi kuitenkin pyytää ydintä kahdentamaan jonkin fyysisen muistialueen omaan virtuaalimuistiinsa. Tämä voi olla esimerkiksi laitteen käyttämä muistialue. mmap-funktiota voidaan käyttää dev/mem -laitetiedostolle, joka osoittaa keskusmuistiin. mem -laitteen käyttö vaatii ajurilta CAP_SYS_RAWIO-kyvyn [12, s. 801]. Koska mmap-funktiolla on pääsy koko järjestelmän muistiin, sitä kannattaa käyttää harkiten.

Toisin kuin ytimen muistia, käyttäjätilan muistia voidaan sivuttaa [25, luku 18.1.1]. Käyttöjärjestelmä voi siirtää (sivuttaa) harvoin käytettyjä muistin lohkoja (sivuja) kiintolevylle, jos vapaata keskusmuistitilaa on vähän. Muistisivut luetaan takaisin keskusmuistiin, jos niitä käytetään. mlock-järjestelmäkutsulla voi lukita prosessin muistisivuja niin, ettei niitä sivuteta levylle [26].

DMA-siirto käyttää tiedonsiirtoon yhtenäistä fyysistä muistialuetta, eikä käyttäjätilan ohjelmalla ole mahdollisuutta varata sitä kmalloc-funktion puuttuessa. Esimerkiksi valmis BSD-lisensoitu Udmabuf-ydinmoduuli pystyy kuitenkin tekemään tämän ja tuomaan DMA-puskurin saataville käyttäjätilaan.

Laitetiedostoja käytetään write- ja read-komennoilla, joilla kirjoitetaan tai luetaan suoraan /dev/ -tiedostojärjestelmässä esiintyviin tiedostoihin. [12, s. 29-30] Käyttämällä /dev/ -laitetiedostoja voi myös käyttää suoraan niitä tietokoneen portteja tai väyliä, joille löytyy jo tuki käyttöjärjestelmästä. Tietokoneen rinnakkaisportteihin voi kirjoittaa käyttämällä /dev/port/ -laitetta ja /dev/i2c -laitteen kautta pääsee käsiksi I^2C-väylään. Laitetiedosto /dev/bus/usb antaa kirjoittaa USB-portteihin. Sarjaväylälle voi kirjoittaa samankaltaisesti termios-komennolla [34] ja sarjaväylän laitekohtaisia ioctl-komentoja voi lähettää ioctl_tty-komennolla [9]. Tämä mahdollistaa raakadatan kirjoituksen tuettuihin laitetyyppeihin.

Käyttäjätilan ajurit eivät voi käyttää keskeytyksiä, koska niillä ei ole mahdollisuutta rekisteröidä keskeytyskäsittelijää. Muistipaikkojen tai porttien pollaus on kuitenkin mahdollista poll() -komennolla.

Userspace IO on I/O-rajapinta, joka soveltuu erityisesti laitteille joiden muistialueita voidaan peilata käyttäjätilan osoiteavaruuteen ja jotka tuottavat keskeytyksiä. Rajapinta vaatii yksinkertaisen ydinmoduulin käyttöä toimiakseen. UIO:ta käyttävän laitteen osoiteavaruus peilataan /dev/uio- laitetiedostoon, minkä jälkeen ydinmoduuli ohjaa laitteen keskeytykset siihen. Käyttäjätilan ajuri voi laitetiedostoa lukemalla tietää milloin keskeytys on saapunut, koska lukufunktio palautuu. Jos ajuri ei halua olla lukkiutuneena odottamassa funktion palutumista, se voi käyttää myös pollausta select-komennolla. Laitetiedostoon kirjoittamalla keskeytykset voi poistaa tai ottaa käyttöön. UIO-laitteen muistia voidaan peilata ajurin muistiavaruuteen mmap-komennolla. UIO-laite voi tarjota muistialueidensa ja porttiensa nimi-, osoite- ja kokotiedot sysfs-tiedostojärjestelmän kautta. [35]

Linux ei salli suoraa pääsyä I/O-portteihin, vaan niiden käsittelyyn täytyy pyytää lupa käyttämällä ioperm- järjestelmäkutsua [10]. Kutsuvalla ajurilla täytyy olla CAP_SYS_RAWIO-kyky [12, s. 801]. Mikäli portti ei ole varattu jonkin toisen ajurin toimesta, se annetaan pyytävän prosessin käyttöön ja portteja voi käsitellä in- ja out -funktioilla.

3.2.5 Lisenssirajoitukset

Koska ohjelma toimii käyttäjätilassa, ytimelle asetetut vaatimukset GPL-yhteensopivuudesta eivät päde. Linuxin käyttäjätila-järjestelmäkutsurajapinta ei aseta vaatimuksia niitä kutsuvien sovellusten lisenssille [36]. Ajurin linkittäminen muihin käyttäjätilan kirjastoihin voi kuitenkin asettaa tapauskohtaisia vaatimuksia lisenssille.

Jos käyttäjätilan ajuri toteutetaan hybridinä yhdessä pienen ydinmoduulin kanssa, ydinmoduuliin pätevät samat rajoitukset kuin luvussa 3.1.5.

3.3 Ajurityyppien keskinäinen vertailu

Oppimiskynnys on suurempi ydintilan ajureiden valmistuksessa, mutta käyttäjätila-ajuria ohjelmoidessa joutuu tekemään lähes kaiken itse. Ydintilan ajurin virheenetsintä vaatii erityisiä työkaluja ja mahdollisesti ohjelmointikäytäntöjen opettelua. Ytimen rajapinnat, kuten resurssien varaus ja vapautus esimerkiksi kmalloc-toimintoa käyttäen eroaa normaalista käyttäjätilan ohjelmoinnista. Käyttäjätilan ajuri ei vaadi erityistä tuntemusta ytimen sisärakenteista, koska niitä ei voida muutenkaan käyttää suoraan. Toisaalta ydintilan ajurin toimintaperiaatteen tunteminen voi auttaa käyttäjätilan ajurin rakenteen suunnittelussa. Käyttäjätilan ajurilta puuttuu monia ytimen automaation tarjoamia etuja, kuten suuri joukko ytimen tarjoamia tukikirjastoja ja automaattinen virheraportointi, ja niille joutuu kuitenkin toteuttamaan itse jonkinlaisen rajapinnan. Ytimen ajuri voidaan saada yhdistymään tiedostojärjestelmään, jonka kautta käyttäjätilan ohjelmat pystyvät käyttämään sitä ja sen metatietoa. Ytimen ajuripuuhierarkian ja ajurirajapintojen puuttuessa käyttäjätila-ajuri joutuu suunnittelemaan ne itse, niin että ne ovat yhteensopivat järjestelmän ja ajuria kutsuvien prosessien kannalta. Jos käyttäjätilan ajuritoteutukseen tarvitaan erillistä ytimen sovitinajuria, ytimen oppimistaakka tulee vastaan joka tapauksessa, ja voi olla perusteltua tehdä koko ajuri ydintilassa.

Suorituskykyyn vaikuttaa erityisesti tilan vaihto käyttäjätilan ja ydintilan välillä järjestelmäkutsua kutsuttaessa. Tämän takia jatkuva järjestelmäkutsujen käyttö käyttäjätilan ajurin kutsuessa ydintä voi johtaa huonoon suorituskykyyn. Taustalla tapahtuvat keskeytykset, järjestelmäkutsujen viive, muistin sivutus ja ohjelmajärjestelijä luovat käyttäjätilan ajurin toimintoihin viivettä. Toisaalta on myös huono, jos ydinajurien pitkät toiminnot lukitsevat järjestelmän pitkäksi aikaa. Jatkuva pollaus voi vaikuttaa negatiivisesti suorituskykyyn. Jos laite tuottaa runsaasti keskeytyksiä, tehokkuusero pollaukseen vähenee, ja pollaus käyttäjätilassa voi muodostua tehokkaammaksi vaihtoehdoksi. Käyttäjätila ei välttämättä pysty tekemään todella suurta ajallista tarkkuutta vaativia operaatioita, mutta silloin on muutenkin tarpeen asettaa ajurin vaatimukseksi tosiaikainen Linux-ydin.

Kummankin ajurityypin pitää tukea uudelleenkutsuttavia funktioita, kun useat prosessit ottavat niihin yhteyttä samanaikaisesti. Ydintilassa tämän toteuttaminen on yksinkertaisempaa, koska ajuri voi käyttää esimerkiksi ytimen työjonoja ja pysäyttää kaikki prosessit odottamaan.

Käyttäjätilan ajuri sopii erityisen hyvin sellaisille laitteille, joille on jo osittainen alhaisen tason tuki järjestelmässä. Jotain väylää lukeva ajuri näyttäisi sopivan hyvin käyttäjätilassa toteutettavaksi, sillä ajuri voi käyttää niitä suoraan esimerkiksi I/O-kutsujen ja /dev/-tiedostojen kautta. Esimerkiksi FTDI-laitteen päässä olevalle mikrokontrollerille olisi helppoa luoda sarjaporttia lukeva ajuri, kun dataa välittävä laitteenosa on jo tunnistettu. Käyttäjätilan ajuri sopii hyvin pienen kokoluokan rakenteluun, mutta suuremmat ajurit vaativat suhteessa enemmän vaivaa.

Ydintilan ajurilla on täydet oikeudet, mutta käyttäjätilan ajurille täytyy erikseen antaa erityisoikeuksia. Tietojen luottamuksellisuus ja eheys voivat vaarantua, jos käyttäjätilan ajurilla on liian suuret oikeudet. Se voi toimia hyökkäysvektorina käyttöjärjestelmän ytimeen, erityisesti koska sen rajapinnan luominen on haastavampaa ja muut sovellukset pääsevät siihen helpommin käsiksi. Ohjaintoimintoja kutsuvien käyttäjäprosessien oikeuksien tarkastus tehdään tavallisesti ajuritiedostojen luku- ja kirjoitusoikeuksista, ja joskus itse ydinajurissa [29, s. 144]. Käyttäjätilan ajuri ottaa tässä ylimääräisen vastuun myös muista ohjelmista, jos sen täytyy tarkistaa kutsuvan prosessin oikeudet. Tämänkin voi osittain ratkaista käyttämällä laitetiedostoa myös käyttäjätilan ajurin kanssa. Linux-ydin itsessään suojaa ydintilan ajuria, koska ydinmoduulista on vähemmän rajapintaa näkyvissä käyttäjätilaan kuin vastaavasta käyttäjäajurista-prosessista. Tietoturvauhka tulee kuitenkin tunnetusti tavallisesti ensin käyttäjätilasta päin, yrittäen saada suurempia käyttöoikeuksia. Yksi osa tietoturvallisuutta on myös tiedon saatavuus. Ydintilan ajurien välillä ei ole lainkaan muistisuojauksia, jolloin yksi ydintilan ajuri voi kaatuessaan viedä koko järjestelmän mukanaan. Käyttäjätilassa kaatumiset voidaan käsitellä helpommin ja laite saadaan nopeasti takaisin toimintaan ajuriohjelman uudelleenkäynnistämisellä. Tältä kannalta katsoen ydintilan ajurin aiheuttamat ongelmat ovat vakavampia.

Ajurin vakauden lisäksi kuluttaja-loppukäyttäjää kiinnostanee ajurin vaivaton käyttö. Käyttäjätilan ajuri pitää asettaa erikseen käynnistymään automaattisesti, kun laite liitetään, mikä tarvitsee pienen lisäskriptin kehittäjän taholta, mutta muuten prosessi on hyvin samankaltainen kuin ydinajurinkin kanssa. Jos käyttäjätilan ajuri tarvitsee erillisiä kirjastoja, ne pitää joko tarjota ajurin mukana tai varmistaa, että ne ovat saatavilla Linux-jakeluversiolle. Käyttäjätilan koodi on hyvin siirrettävää, jos se käyttää vain vakaita rajapintoja. Jos ydinmoduuli käyttää DKMS-rajapintaa, sen kääntäminen ytimen päivittyessä vie aina hieman aikaa. Asian voi korjata joko lataamalla ja päivittämällä ajuri aina uutta ydinversiota varten tai pyytämällä sen lisäys Linux-jakelun mukaan. Toisaalta DKMS helpottaa ohjelmointitaakkaa, kun ytimen rajapinnan kehitystä ei tarvitse seurata niin tarkasti.

Vaikka ydintilan ajurin mahdollisuudet muistinhallintaan ovat laajemmat kuin käyttäjäajurin, sen muistinkäyttö on rajoitetumpaa. Ydintilan ajurilla on käytettävissään vain alle 1 gigatavu muistia (jos 32-bittisiä järjestelmiä halutaan tukea) josta suurin osa on varattuna toisille ytimen osille. Sen sijaan käyttäjätilan ajuri nauttii mahdollisuuksista käyttää paljon muistia, kuten myös kutsua helposti toisia prosesseja tai käyttää levyaseman tiedostoja. Ydinajurikin voi tehdä saman, mutta se voi joutuu käyttämään alemman tason rajapintoja ja huolehtimaan muistin riittävyydestä.

DMA:ta tarvitsevat laitteet eivät toimi ilman ydinajuria. Laitetyypistä riippuen saatavilla voi olla jo valmis kolmannen osapuolen DMA-ydinajuri, jonka voi liittää omaan käyttäjätilan ajuriin tai jota voi jatkokehittää itselle sopivammaksi. PCI-laitteet käyttävät usein DMA:ta tai keskeytyksiä ja Userspace I/O -järjestelmä sopii esimerkiksi tiettyjen PCI-laitteiden käyttöön [35]. Jos laite tarvitsee DMA-siirtoja, keskeytyksiä, tai hyötyy optimoidusta muistinkäytöstä, ydinajuri voi olla kuitenkin parempi vaihtoehto.

Lohkolaitteista täytyy huomioida, että ne voivat vaatia ajurilta suurta suorituskykyä. Lohkolaitetta, kuten levyasemaa, käytetään myös eri tavalla kuin merkkilaitetta; sektoreittain. [29, s. 464-465] Käyttäjätilan ajurin täytyy luoda sen käyttöön sopiva menetelmä. Alhaisen tason verkkolaiteajuri on niin kiinteä osa ydintä, ettei sitä voi toteuttaa käyttäjätilassa. Sen käsittelyyn on kuitenkin UAPI-rajapinta.

USB-laitteet jotka vaativat nopeaa vastausta keskeytysten avulla voivat olla haastavampia toteuttaa käyttäjätilassa, mutta esimerkiksi libusb tarjoaa tähän libusb_interrupt_transfer -funktion, joka välittää keskeytyksiä ydin- ja käyttäjätilan välillä. Esimerkiksi tulostimen ja sen kaltaisten laitteiden ajurit voisivat toimia kokonaan libusb:n varassa käyttäjätilassa. Käyttäjätilan käyttö tulostimelle on hyvä asia, jos sen tarvitsee esimerkiksi käsitellä tulostetta, koska se voi käyttää siihen käyttäjätilan palveluita.

Vaikka suuri osa ajurityypeistä voidaan toteuttaa kiertotietä pitkin käyttäjätilassa, sen ajurit eivät kuitenkaan taivu kaikkeen. Käyttäjätilan ajurin käyttö osana laitteistoläheistä alusta-ajuria (platform driver) ei välttämättä onnistu, koska käyttäjätila-ajurin hyödyntämät järjestelmäajurit eivät ole vielä latautuneet.

Lisenssivalinta vaikuttaa erityisesti kaupallisten ajurien tekoon. Jaeltavien ydintilan ajurien täytyy olla GPL-yhteensopivia, kun taas käyttäjätilan ajurien lisenssi riippuu lähinnä kolmannen osapuolen käytetyistä ohjelmakirjastoista. Lisenssi ei kuitenkaan estä laiteajurien tekoa; esimerkiksi Nvidia näyttää toteuttavan grafiikka-ajurinsa pienen avoimen lähdekoodin ydintilan sovittimen ja käyttäjätilan ajurikirjastojen yhdistelmänä [28, luku 4C]. Voi olla hyvä huomioida, että osa Linux-yhteisöstä saattaa suhtautua negatiivisesti suljetun koodin ajureihin.

4. YHTEENVETO

Työssä käytiin läpi ilmeisimmät erot käyttäjätilan ja ydintilan ajurin välillä. Esille tuli, että käyttäjätilan ajuri pystyy tekemään suurelta osin sen minkä ydintilan ajurikin. Joitain toimintoja, kuten keskeytyksiä on vaikea saada toimimaan käyttäjätilassa. Monet toiminnot vaativat erityisten rajapintojen käyttöä, jotka kiertävät rajoitukset. Monessa tapauksessa tämä ei ole yhtä käytännöllistä kuin käyttäjätilan ajurin käyttö.

Käyttäjätilan ajuri näyttää olevan parhaimmillaan ja yksinkertaisimmillaan, kun se ja laite on yhdistettynä väylään, jolle on jo olemassa perustason Linux-ajuri. Yksinkertainen I/O-käyttäjätila-ajuri on vaivattomampaa toteuttaa. Toisaalta ydintilan ajuri toimii joka tapauksessa, niissäkin missä käyttäjätilan ajuri ei. Havaittiin myös, että lisenssivaatimukset ja asennusympäristö saattavat asettaa rajoituksia ajureille.

Työtä voisi tarvittaessa jatkaa tai lähestyä eri näkökulmasta esimerkiksi perehtymällä tarkemmin jonkin tietyn laitemallin ajuritoteutukseen sekä käyttäjätilassa että ydintilassa, ja kirjaamalla projektin havaintoja.

LÄHTEET

[1] BUSE - A block device in userspace, BUSE Github repository. Saatavilla (viitattu 21.6.2018): https://github.com/acozzette/BUSE

[2] call_usermodehelper-funktio, Kernel.org dokumentaatio. Saatavilla (viitattu 25.5.2018): https://www.kernel.org/doc/htmldocs/kernel-api/API-call-usermode-helper.html

[3] Debug symbols for linux-image-4.9.0-6-amd64, Debian. Saatavilla (viitattu 17.5.2018): https://packages.debian.org/stretch/linux-image-4.9.0-6-amd64-dbg

[4] DRM-otsikkotiedostot, Linux Github repository. Saatavilla (viitattu 21.6.2018): https://github.com/torvalds/linux/tree/master/include/uapi/drm

[5] Dynamic Kernel Module Support, GitHub-ohjelmavarasto. Saatavilla (viitattu 21.5.2018): https://github.com/dell/dkms

[6] Dynamic Kernel Module Support, Arch linux wiki. Saatavilla (viitattu 21.5.2018): https://wiki.archlinux.org/index.php/Dynamic_Kernel_Module_Support

[7] GNU General Public License, version 2. Saatavilla (viitattu 12.6.2018): https://www.gnu.org/licenses/old-licenses/gpl-2.0.html

[8] Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3, Intel Corporation, July 2017. Saatavilla (viitattu 12.5.2018): https://software.intel.com/en-us/articles/intel-sdm

[9] ioctl_tty, Linux Programmer's Manual. Saatavilla (viitattu 20.6.2018): http://man7.org/linux/man-pages/man4/tty_ioctl.4.html

[10] ioperm - set port input/output permissions, Linux Programmer's Manual. Saatavilla (viitattu 14.6.2018): http://man7.org/linux/man-pages/man2/ioperm.2.html

[11] M. Jangir, Linux Kernel and Device Driver Programming, Laxmi Publications, 2014. Saatavilla (viitattu 17.6.2018): http://www.books24x7.com.libproxy.tut.fi/marc.asp?bookid=69803

[12] M. Kerrisk, The Linux Programming Interface : A Linux and UNIX System Programming Handbook, No Starch Press, Incorporated, 2010.

[13] D. Kirkland, ZFS Licensing and Linux, Ubuntu Blog, 18.2.2016. Saatavilla (viitattu 13.6.2018): https://blog.ubuntu.com/2016/02/18/zfs-licensing-and-linux

[14] G. Kroah-Hartman, The Linux Kernel Driver Interface, Linux kernel documentation. Saatavilla (viitattu 12.6.2018): https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/process/stable-api-nonsense.rst

[15] libevdev 1.5.9, A wrapper library for evdev devices. Saatavilla (viitattu 21.6.2018): https://www.freedesktop.org/software/libevdev/doc/1.5.9/

[16] libfuse cuse.c File Reference, libfuse Doxygen-dokumentaatio. Saatavilla (viitattu 21.6.2018): https://libfuse.github.io/doxygen/cuse_8c.html

[17] libinput, Input device management and event handling library, libinput Github repository. Saatavilla (viitattu 21.6.2018): https://gitlab.freedesktop.org/libinput/libinput

[18] libusb-1.0 API Reference, libusb API dokumentaatio. Saatavilla (viitattu 21.6.2018): http://api.libusb.info/

[19] M. Liljegren, User-Space Device Drivers in Linux: A First Look, Whitepaper, ENEA. Saatavilla (viitattu 17.5.2018): https://www.enea.com/globalassets/downloads/operating-systems/enea-linux/enea-user-space-drivers-in-linux_whitepaper.pdf

[20] Linux.fi wiki, Modprobe. Saatavilla (viitattu 17.5.2018): https://www.linux.fi/wiki/Modprobe

[21] Linux.fi wiki, Ytimen moduulit. Saatavilla (viitattu 17.7.2018): https://www.linux.fi/wiki/Ytimen_moduulit

[22] Linux Kernel Programming IDE (LinK+ IDE), Eclipse Marketplace. Saatavilla (viitattu 22.5.2018): https://marketplace.eclipse.org/content/linux-kernel-programming-ide-link-ide

[23] Y. Lixiang et al., The Art of Linux Kernel Design: Illustrating the Operating System Design Principle and Implementation, Auerbach Publications, 2014. Saatavilla (viitattu 20.6.2018): https://library-books24x7-com.libproxy.tut.fi/toc.aspx?bkid=51988

[24] J. Madieu, Linux Device Drivers Development, Packt Publishing, Birmingham, 2017.

[25] W. Mauerer, Professional Linux Kernel Architecture, Wrox Press, 2008. Saatavilla (viitattu 17.6.2018): https://library-books24x7-com.libproxy.tut.fi/toc.aspx?bookid=27318

[26] mlock(2), Linux manual page. Saatavilla(viitattu 11.6.2018): http://man7.org/linux/man-pages/man2/mlock.2.html

[27] Y. Padioleau, J. Lawall, G. Muller, Understanding collateral evolution in Linux device drivers, Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems, 2006. Saatavilla (viitattu 21.5.2018) https://dl-acm-org.libproxy.tut.fi/citation.cfm?doid=1218063.1217942

[28] README.txt, Linux x64 (AMD64/EM64T) Display Driver, Nvidia Corporation. Saatavilla(viitattu 23.6.2018): us.download.nvidia.com/XFree86/Linux-x86_64/396.24/NVIDIA-Linux-x86_64-396.24.run

[29] A. Rubini, J. Corbet, G. Kroah-Hartman, Linux Device Drivers. 3rd ed., O'Reilly Sebastopol (CA), 2005.

[30] C. Simmonds, Mastering Embedded Linux Programming - Second Edition, Packt Publishing, 2017.

[31] M. Swift, B. Bershad, H. Levy, Improving the Reliability of Commodity Operating Systems, University of Washington, ACM Transactions on Computer Systems, Vol. 23, No. 1, 2005, s.77–110. Saatavilla (viitattu 12.6.2018): http://pages.cs.wisc.edu/~swift/papers/nooks-tocs.pdf

[32] M. Swift et al., Recovering Device Drivers, Department of Computer Science and Engineering, University of Washington, OSDI ’04: 6th Symposium on Operating Systems Design and Implementation. Saatavilla (viitattu 12.6.2018): https://www.usenix.org/legacy/event/osdi04/tech/swift/swift.pdf

[33] Tainted kernel, Suse Support. Saatavilla (viitattu 25.5.2018): https://www.suse.com/support/kb/doc/?id=3582750

[34] termios, Linux Programmer's Manual. Saatavilla (viitattu 20.6.2018): http://man7.org/linux/man-pages/man3/termios.3.html

[35] The Userspace I/O HOWTO, Kernel.org dokumentaatio. Saatavilla (viitattu 14.6.2018): https://www.kernel.org/doc/html/v4.13/driver-api/uio-howto.html

[36] L. Torvalds, Linux syscall note, Linux github repository. Saatavilla (viitattu 12.6.2018): https://github.com/torvalds/linux/blob/master/LICENSES/exceptions/Linux-syscall-note

[37] udev, Linux Dynamic Device Management, Kernel.org dokumentaatio. Saatavilla (viitattu 10.6.2018): https://kernel.org/pub/linux/utils/kernel/hotplug/udev/udev.html

[38] Usermode API-otsikkotiedostot, Linux Github repository. Saatavilla (viitattu 21.6.2018): https://github.com/torvalds/linux/tree/master/include/uapi