

PERUSTASON LINUX-YDIN MODUULIT
Kernel 4.19.x

Raportti
Roger Kupari
09.02.2018

SISÄLLYS

1 LUKIJALLE	3
2 LINUX KERNEL VS. USERSPACE	4
3 LINUX-HEADERIT	5
4 MODUULI, KÄÄNTÄMINEN, ASENNUS/POISTO	6
4.1 ESIMERKKI 1 [moduulin asennus ja poistaminen]	6
4.1.1 MODUULIN KÄÄNTÄMINEN	8
4.1.2 MODUULIN ASENTAMINEN/POISTAMINEN	9
4.1.3 MODUULIN LATAAMINEN KÄYNNISTYKSEN YHTEYDESSÄ	10
5 ESIMERKKI 2 [painikekeskeytys]	11
6 ESIMERKKI 3 [kernel-moduulin toimintojen ohjaus käyttäjätalasta]	14
6.1 Moduulin makefile lisäykset	18
6.2 Käyttäjätalakoodin makefilen sisältö	19
6.3 Esimerkki 3:n yhteenvedo	20
7 YHTEENVETO	23
8 LÄHTEET	24

1 LUKIJALLE

Tämän asiakirjan tarkoitus on toimia raporttina perusopintojen hyväksilukemiseen Tampereen ammatikorkeakoulussa. Tämä ei siis ole opinnäytetyö, eikä laajuus vastaa sitä. Tässä asiakirjassa ei noudateta opinnäytetyön muotovaatimuksia. Tätä asiakirjaa ei ole muodollisesti tarkastettu, joten koittakaa elää myös mahdollisten kirjoitusvirheiden kanssa. Toisinaan joudun itse näkemään monimutkaisten asioiden ymmärtämiseksi valtavasti vaivaa. Tällaisten asioiden ymmärtämiseksi joudun siis luomaan itselleni jonkinlaisen mielikuvan siitä, miten sitä tulisi soveltaa. Tästä johtuen, kirjoittamani asiasisältö ei välttämättä ole juuri pilkulleen niin esitetty, kuin sen kanta virallisesti on. Tämän pohjalta suosittelen vahvasti esitettyjen lähteiden tarkastelua.

Tässä raportissa käsitellään yleisiä perustason asioita ajurimoduulin tekemiseksi Linux-ytimeen. Asiat pyritään esittämään selostamalla ja selostuksen tueksi esimerkkien kautta. Esimerkit käsittävät moduuleja, joilla luodaan ulkoisia efektejä fyysiseen maailmaan -> Ledin ja painikkeen avulla.

Tämä raportti ei anna vastausta siihen, että miten tehdä esimerkiksi tulostimen ajuri tai siihen verrattavan laitteen ajuri, joka vaatii valtavasti kohdelaitteen matalan tason speksejä. Esimerkiksi kohdelaitteen käskytoiminnot&-osoitteet, dataprotokolla sekä käytettävän ohjausväylän ja dataväylän alustukset jne. Verrattuna tuohon, ledillä ja painikkeella on yksinkertainen käskykanta jotka ovat: "1" ja "0". Tekstissä esitettyjen headerien lähdekoodia tutkimalla on todettavissa, että pinnien käskykanta, keskeytystoiminnot jne. on tehty kehittäjälle todella helpoksi.

Olen pyrkinyt esittämään lähteitä syvempään tietoon, lähteet ovat merkattu viittausjärjestyksessä. Joitakin syvemmän tiedon lähteitä olen korostanut, tai merkannut osoitteet suoraan tekstin perään. Ajankapteen vuoksi minulla ei ole ollut aikaa kirjoittaa jokaista nippelitietoa (taulukot, kuvat, kommentit jne.) "omin sanoin" tai "piirtää itse". Tämän vuoksi olen tiettyjä lähteitä korostanut mitkä olen asiayhteydessä kokenut astetta tärkeämmiksi. Tällä toimintatavalla tässä asiakirjassa on säilytetty hyvä tekiänoikeustapa. Lukija voi itse päättää, tarvitseeko hän esitettyä nippelitietoa (lähteistä) vai tyytykö siihen osaan, joka tässä on selostettu. << Kuten ensimmäisessä kappaleessa mainitsin, tämä ei ole opinnäytetyö. Toteutus ei vastaa sitä muodollisesti eikä arvostelulaajuudessaan.

Vielä yksi asia: Pyrin välttämään itseni toistamista, joten kerran esitettyjä syvempiä asioita ei esitellä enää myöhemmissä vaiheissa. Esimerkiksi: esimerkin 1 avauskoodissa käsiteltyjä perusasioita ei syvällisesti käsitellä enää esimerkissä 2 jne.

Raportin viimeisellä sivulla on lähdeluettelo.

2 LINUX KERNEL VS. USERSPACE

Netistä löytyy ihan hyvin tietoa ydintilan ja käyttäjätilan välisistä eroista. Joissakin lähteissä on painotettu sitä, että jos kehittäjä pystyy tekemään ajurin käyttäjätilaan, niin olisi se suositeltavaa tehdä myös sinne.

Syitä tähän on juurikin virheen ilmentymiset. Olisi turvallisempaa, jos mahdolliset virheet tapahtuvat käyttäjätilassa (ks. lista alla). Lisäksi ytimen pinon kokoa on mainittu ”pieneksi”, en kuitenkaan löytänyt ajantasaista tietoa siitä minkä kokoinen se olisi suunnilleen. Kernel-ajurin tekeminen on useimmiten aikaa vievää.

Alla on listattu muutamia eroavaisuuksia ytimen ja käyttäjätilan elementeistä. Lisäksi graafisesti yksinkertaistettuna Linux järjestelmä on seuraavan kuvan mukainen:

<http://derekmolloy.ie/wp-content/uploads/2015/04/userspace-kernelspace.png>

YDINTILA

headerit: linux-headers-<kernelversio>

tietyn prosessin keskeytys (kill): ei

Virhe voi kaataa koko käyttöjärjestelmän

Rajoittamaton pääsy mm. muistiin => saa lukea/kirjoittaa mitä tahansa muistiosoitetta prosessorilta.

KÄYTTÄJÄTILA

headerit: mm. standardikirjastot ym.

tietyn prosessin keskeytys (kill): kyllä

Rajattu pääsy muistiosoitteisiin

Jotta Linux-käyttöjärjestelmää pystyy ajamaan, prosessorilla täytyy olla muistinhallintajärjestelmä (MMU), joka tarjoaa virtuaaliosoitteet fyysisiin muistipaikkoihin. Käyttäjätilasta ja ydintilasta osoitteet ilmenevät eri tavalla, joka on hyvin avattu lähteessä: kernel vs user memory space.

lähteitä: [mmap, kernel introduction, ytimen ja käyttäjätilan eroja, artikkeli kernelin heikkouksista, kernel vs user memory space]

3 LINUX-HEADERIT

Kuten aiemmin mainitsin, kernelissä käytetään header -tiedostoina kyseisen ydinversion headereita. Headereiden asennus tapahtuu paketinhallintajärjestelmän kautta linux-headers-<linux-kernel-versio>. Kernel-koodissa ei voi käyttää käyttäjätilan koodia.

Itselläni on alustana nyt tällä hetkellä Tinker Board S ja siinä käyttöjärjestelmänä Armbian. Kernelin versio on siinä 4.19.14-rockchip ja tällaista pakettia ei paketinhallinnasta löydy. Armbianin configissa pystyy kuitenkin hakemaan headerit.

Virallisemmän version käyttöjärjestelmä TinkerOs:n uusin versio niin sanotusti kaikilla herkuilla, on kirjoitushetkellä varustettu kernelversiolla 4.4.132. Tämäkään ei löydy paketinhallinnasta.

Jos paketinhallinnasta eikä mistään muualtakaan saa käyttöönsä Linux-headereita: Tulee ko. ytimen lähdekoodi ladata ja kääntää. Yleensä kääntämiseen löytyy ohjeet laudan valmistajalta. Tehdyn kernel-moduulin käännöstiedoissa on sitten viitattava käännetyt ytimen polkuun, joka esitellään myöhemmin makefilen yhteydessä.

Kernelin api löytyy: <https://www.kernel.org/doc/html/v4.19/driver-api/>

Jos selaat Apia, huomaat että funktioiden käyttötarkoitusta siellä on selitetty ihan kiitettävästi. Mutta läheskään kaikkien funktioiden headeria ei ole mainittu, jotkut saattavat joltain sivulta löytyä. Nopeimmaksi tavaksi löytää ko. headerin on kopioida funktio Bootliniin josta löytyy itseasiassa kattava määrä lähdekoodeja eri kernelversioilla.

Bootlin: <https://elixir.bootlin.com/linux/v4.19/source/include/linux/>

Yleensä ajurille on joku funktionaalinen tavoite. Tuon tavoitteen perusteella tulisi sitten osata etsiä sellaisia headereita joilla pystyy toteuttamaan halutun funktionaalisuuden. Sopivien headereiden paikallistaminen voi olla aluksi hankalaa jos ei tiedä mitä pitäisi edes löytää, mutta suosittelen käyttämään Googlea, bootlinia ja kernel.orgia.

4 MODUULI, KÄÄNTÄMINEN, ASENNUS/POISTO

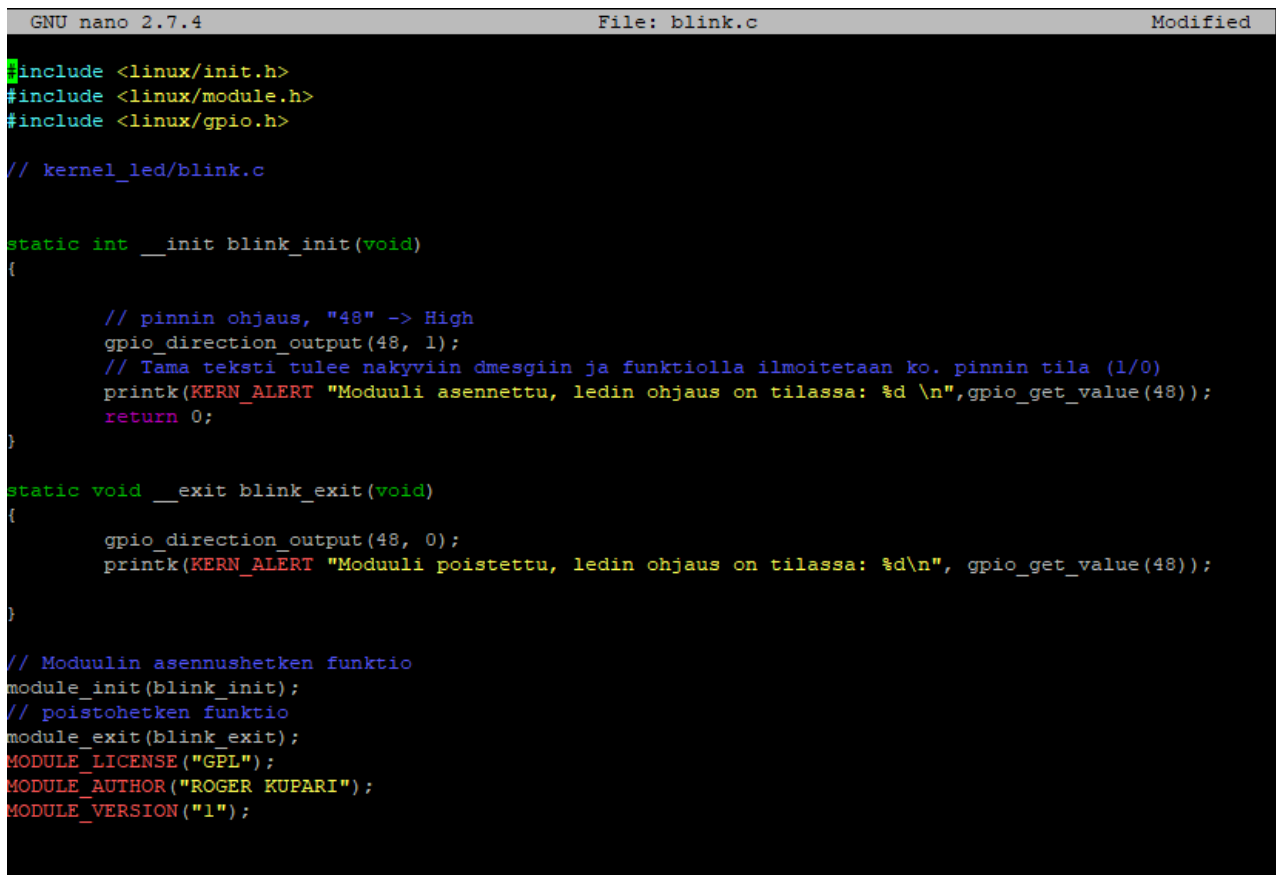
Tässä luvussa käsittelen nyt kernel-moduulin perusrunkoa ja sen kääntämistä make -työkalun avulla. Lediä pystyisi vilkuttelemaan suoraan käyttäjätilassakin eritavoin mutta en ota niihin metodeihin nyt, enkä myöhemmin tässä raportissa kantaa.

Esimerkkikoodit löytyvät osoitteesta: https://github.com/rogerkupari/drivers_public

Esimerkki 1 (kernel_led/blink.c) taustatiedot:

Moduuli ohjaa tässä ja seuraavissa esimerkeissä vihreää "act_lediä", joka fyysisesti sijaitsee Tinker Board S:ssä power-ledin ja "syke" ledin välissä. Kyseisen pinnin numero on määritetty arvoon 48. Pinnin numero tulee ilmeisesti U-bootin tai kernelin devicetree/config rakenteesta, kyseinen puoli on allekirjoittaneelle vielä vähän harmaata aluetta. Rakenne sisältää linkitysrakenteen, joka vaikeuttaa entisestään ymmärtämistä. Kaupallisilla prosessorialustoilla valmistajan dokumentaatiosta kuitenkin yleensä ilmenee tarvittavat arvot. Ainakaan Tinker Boardilla käytettävät gpio arvot eivät ole mitenkään loogisissa numeraalisessa yhteydessä esimerkiksi korkean tason pinneihin 1 – 40, tämä selviää hieman myöhemmin. Tyydymme tällä erää siis tietoon, että gpio arvo 48 vastaa nyt haluttua lediä.

4.1 ESIMERKKI 1 [moduulin asennus ja poistaminen]



```
GNU nano 2.7.4 File: blink.c Modified

#include <linux/init.h>
#include <linux/module.h>
#include <linux/gpio.h>

// kernel_led/blink.c

static int __init blink_init(void)
{
    // pinnin ohjaus, "48" -> High
    gpio_direction_output(48, 1);
    // Tama teksti tulee nakyviin dmesgiin ja funktiolla ilmoitetaan ko. pinnin tila (1/0)
    printk(KERN_ALERT "Moduuli asennettu, ledin ohjaus on tilassa: %d \n", gpio_get_value(48));
    return 0;
}

static void __exit blink_exit(void)
{
    gpio_direction_output(48, 0);
    printk(KERN_ALERT "Moduuli poistettu, ledin ohjaus on tilassa: %d\n", gpio_get_value(48));
}

// Moduulin asennushetken funktio
module_init(blink_init);
// poistohetken funktio
module_exit(blink_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("ROGER KUPARI");
MODULE_VERSION("1");
```

Kuva 1, ensimmäinen esimerkkimoduuli

kernel_led/blink.c:n sisällön selostus

Ylimpänä näkyvät headerit tulevat siis Linux-headereista. Init -headerissa on määritetty muunmuassa ”__init” ja ”__exit” -makrot, joita käytetään merkkamaan ”asennus” ja ”poistofunktiot”. Näitä kyseisiä toimintoja käytetään vain silloin kun moduuli asennetaan tai poistetaan kerneltilasta. Kun moduuli suorittaa jotain muuta tehtävää kuin edellä mainittuja toimintoja, funktioiden käyttämät muistialueet vapautetaan. Lähteessä on hyvin kommentoitu laajemmin. [init.h / Torvalds]

module -headerista ilmenee module_init ja -exit makrot. Tämän headerin avulla ja edellä mainituilla makroilla itse moduuli ”asennetaan” ja ”poistetaan” kerneltilasta. Lisäksi headerin tiedoista löytyy ku-
van 1 kolmen viimeisen rivin makrot. Lisenssistä sen verran: Jos moduulin lisenssi on jokin muu kuin ”GPL”, moduulilla ei välttämättä synny oikeutta käyttää tiettyjä ominaisuuksia, luokat lukeutuvat nii-
hin. Tämän esimerkin mukaisessa koodissa ei lisenssimerkinnällä ole väliä. Jos lisenssiä ei ole mer-
kitty, siitä tulee varoitus käännöksen yhteydessä. [module.h / Torvalds]

Gpio -headerissa on määritetty funktiot pinnien alustukseen, ohjaukseen ja tilakyselyyn. Esimerkki-
koodissa on käytetty ”output” -ohjausta sekä pinnin tilakyselyä. Kuten lähteestä ilmenee, headerissa on
myös funktioita keskeytystoiminnoille, niistä kuitenkin lisää myöhemmissä esimerkeissä. [gpio.h /
Torvalds]

Moduulin toiminta:

Asennettaessa se asettaa blink_init funktion mukaisesti act-ledin tilaan 1 (päälle) ja pois-
tettaessa blink_exit funktion mukaisesti ledi asettuu tilaan 0.

Olen tässä esimerkissä käskyttänyt suoraan pinniä numero 48, gpio_direction_output -
toiminnallisuudella. Eli Tässä määritetään kyseisen pinnin suunnaksi output ja sen tilaksi
1 (päälle). Printk:lla saadaan eritasoisten ilmoitusten lokimerkinnot sekä ainakin osaa
niistä voidaan tarkastella dmesgin kautta. Olen käyttänyt tässä KERN_ALERT -makroa,
sillä saa punaisen huomioväriin dmesg janaan. Tulostuksen yhteydessä kysytään sitten
pinnin tilaa (1/0) ja tulostetaan se ilmoituksen mukana.

Edelleen blink_exit funktiossa käskytetään edellisen kappaleen mukaisesti pinni 48 0-
tilaan ja ilmoituksessa mainitaan moduulin poistamisesta kerneltilasta joka sisältää myös
pinnin sen hetkisen arvon (1/0).

Muistetaan siis edelleen, että tämän moduulin toiminta on asennus- ja poistohetki. Kaik-
kina muina aikoina näiden kyseisten hetkien ulkopuolilla ei ole suoritusta.

Jos ei ole mahdollisuutta käyttää Tinker Boardia, niin tämän moduulin toiminnan pystyy
toteamaan ihan tavallisella pöytäkoneellakin, kun vain jättää gpio:hin liittyvät toiminnot
pois. Toisin sanoen asennushetkellä nähdään dmesgissä tekstiä, moduulin funktioista ei
seuraa ulkoisia toimintoja.

- Printk:sta löytyy lähteestä [printk] ja sieltä otsikon ”log Levels” alta tyhjentävä tau-
lukko printk:n makroista: https://elinux.org/Debugging_by_printing

4.1.1 MODUULIN KÄÄNTÄMINEN

Ennen kernel-tilaan asennusta, moduuli täytyy kääntää. Kääntäminen tehdään make -työkalun avulla jonka rakenne on seuraava:

```
GNU nano 2.7.4 File: Makefile

obj-m = blink.o

moduuli = blink
vers = $(shell uname -r)

all:
    make -C /lib/modules/$(vers)/build M=$(shell pwd) modules
    sudo cp $(moduuli).ko /lib/modules/$(vers)
    sudo depmod -a

clean:
    make -C /lib/modules/$(vers)/build M=$(shell pwd) clean
    sudo rm /lib/modules/$(vers)/$(moduuli).ko
    sudo depmod -a
```

Kuva 2, esimerkki 1:n Makefile (kernel_led/Makefile)

obj-m = tiedosto joka on tyypiltään kerneliin ”asennettava/ladattava” moduuli

moduuli = muuttuja moduulin nimestä makefilen sisään

vers = kernel-versio (4.19.14-rockchip)

make:

tekee moduulitiedoston (joka on .ko päättyinen)

moduulitiedosto kopioidaan /lib/modules sekä päivitetään moduulien riippuvuussuhdelistaus-> jotta voidaan käyttää myös modprobe työkalua, tästä lisää myöhemmin.

clean:

Poistetaan käännös ja muut, lähdekoodi säilytetään.

Suorittaminen komennolla make. Pitäisi näyttää kuvan 3 mukaista tulosta:

```
user@tinkerboard:~/drivers_public/kernel_led$ make
make -C /lib/modules/4.19.14-rockchip/build M=/home/user/drivers_public/kernel_led modules
make[1]: Entering directory '/usr/src/linux-headers-4.19.14-rockchip'
  CC [M] /home/user/drivers_public/kernel_led/blink.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/user/drivers_public/kernel_led/blink.mod.o
  LD [M]  /home/user/drivers_public/kernel_led/blink.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.19.14-rockchip'
sudo cp blink.ko /lib/modules/4.19.14-rockchip
sudo depmod -a
user@tinkerboard:~/drivers_public/kernel_led$
```

Kuva 3, kääntäminen onnistui

Jos prosessissa tulee virheitä, ne paikantuvat yleensä joko koodiin tai virheeseen makefilessä.

4.1.2 MODUULIN ASENTAMINEN/POISTAMINEN

Asentamiseksi on kaksi erilaista tapaa.

1. insmod ”insert module”, jonka syntaksi on: sudo insmod polku_moduuliin/moduulin_nimi.ko.
 - Tämän komennon jälkeen moduuli on kernelissä
2. modprobe, jonka syntaksi on: sudo modprobe moduulin_nimi
 - Tämän komennon jälkeen moduuli on kernelissä

1.Vaihtoehdossa siis täytyy joko olla moduulin polussa tai vastaavasti viitata juuri siihen polkuun, jossa itse moduuli sijaitsee. Lisäksi moduulissa pitää käyttää .ko -päätettä

2.Vaihtoehdossa moduulin nimi on ”globaali” ja sitä voidaan kutsua välittämstä moduulin sijainnista.

- Tämä mahdollistettiin siis makefilen ”all” kohdan kahdella viimeisellä rivillä, jossa moduuli kopioitiin /lib/modules/kernelversio polkuun ja päivitettiin riippuvuussuhdelistaus.

Jos ei halua käyttää ”globaalia” tapaa, niin ko. rivit voi makefilestä poistaa.

Joka tapauksessa, jommankumman komennon suorituksen jälkeen moduulin pitäisi olla kerneltilassa. ”lsmod” komennolla pitäisi tulostua listaus kaikista kernelissä olevista moduuleista ja tässä tapauksessa ilmetä myös ”blink”. Jos näin on, niin kaikki on kunnossa.

Nyt dmesg komennolla pitäisi ilmetä viimeisellä rivillä:

”Moduuli asennettu, ledin ohjaus on tilassa: 1” ja act_ledin pitäisi loistaa Tinker Boardissa.

>> Kaikki toimii

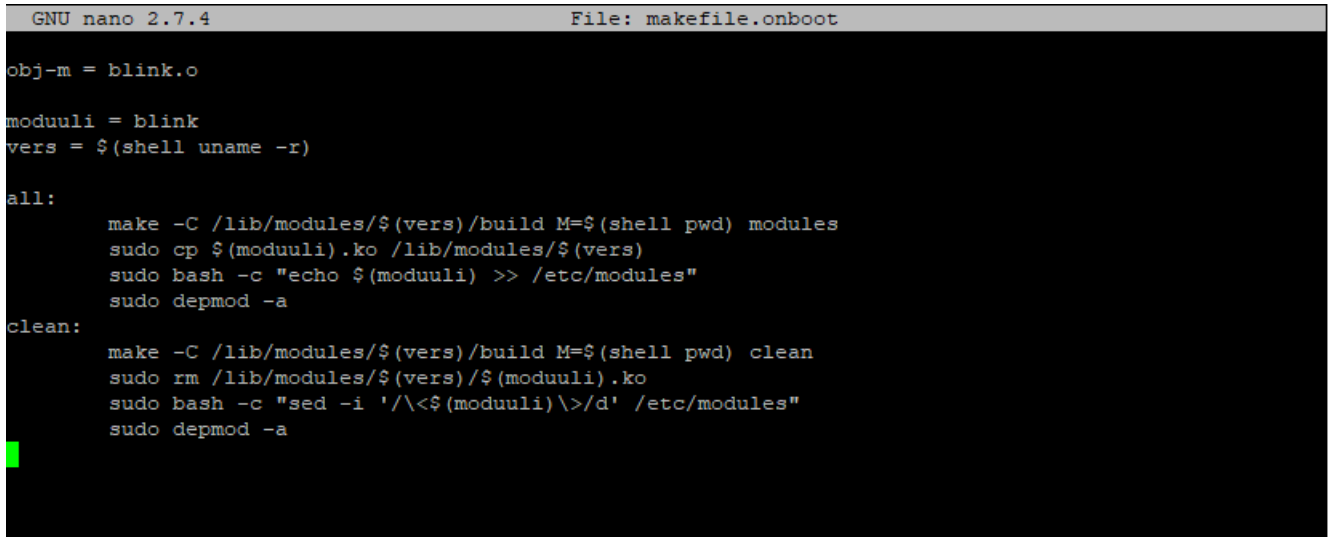
Vastaavasti poistamiseksi on kaksi eri tapaa:

1. rmmod, jonka syntaksi on sudo rmmod moduulin_nimi
 - Tämän komennon jälkeen moduuli on poistettu kernelistä
2. modprobe -r, jonka syntaksi on: sudo modprobe -r moduulin_nimi
 - Tämän komennon jälkeen moduuli on poistettu kernelistä

Jommankumman komennon jälkeen moduuli ei enää esiinny lsmod listauksessa, ja dmesgiin pitäisi olla tullut exit -funktion mukainen ilmoitus: ” Moduuli poistettu, ledin ohjaus on tilassa: 0”, ledin pitäisi olla sammuneena.

4.1.3 MODUULIN LATAAMINEN KÄYNNISTYKSEN YHTEYDESSÄ

Sijoitin moduulin lataamisen käynnistyksen yhteydessä nyt tämän ensimmäisen esimerkin yhteyteen, sillä kernel_led -polusta löytyy ”makefile.onboot” niminen tiedosto, joka on myös esitetty kuvassa 4.



```
GNU nano 2.7.4 File: makefile.onboot

obj-m = blink.o

moduuli = blink
vers = $(shell uname -r)

all:
    make -C /lib/modules/$(vers)/build M=$(shell pwd) modules
    sudo cp $(moduuli).ko /lib/modules/$(vers)
    sudo bash -c "echo $(moduuli) >> /etc/modules"
    sudo depmod -a

clean:
    make -C /lib/modules/$(vers)/build M=$(shell pwd) clean
    sudo rm /lib/modules/$(vers)/$(moduuli).ko
    sudo bash -c "sed -i '/\<$(moduuli)\>/d' /etc/modules"
    sudo depmod -a
```

Kuva 4 moduulin lataaminen käynnistyksen yhteydessä

Kuten kuvaa tarkastelemalla ilmenee; uutta on ”all” kohdan toiseksi viimeinen rivi ja ”clean” kohdan toiseksi viimeinen rivi.

- Käännöksen jälkeen kirjoitetaan /etc/modules -tiedostoon ko. moduulin nimi
- Vastaavasti siivouksen yhteydessä se poistetaan sieltä

>>> Moduuli käynnistyy järjestelmän uudelleen käynnistyksen yhteydessä.

>> Miksi se ei sitten ole käytössä?

- Tinker Boardissa on jossain määritetty toinen ohjuri tälle kyseiselle act_ledille, ja kuten aikaisemmin mainitsin tämän, käsiteltävä moduuli on suorituksella vain ja ainoastaan asennus- ja poistohetkellä. Joka kaikessa yksinkertaisuudessaan tarkoittaa sitä, että tuo act_ledin ohjuri ottaa päätäntävällän, eli tämä meidän moduuli ohjaa sen jossain vaiheessa päälle ja ohjuri vilkuttelee välissä. Jonka jälkeen ledi jää ohjurin käskystä tilaan 0.
- >> dmesgiä tarkastelemalla ilmenee kuitenkin blink moduulin viesti. Eli se on jossain vaiheessa ladattu kerneliin. Ja itseasiassa on siellä vieläkin, mutta ”nukkuu”.
- Kehitysvaiheessa suuri riski tehdä järjestelmästä käyttökelvoton

5 ESIMERKKI 2 [painikekeskeytys]

Tämä esimerkkimoduuli käsittää painikekeskeytyksen käsittelyn kernelmoduulissa.

```
GNU nano 2.7.4 File: keskeytys.c

#include <linux/init.h>
#include <linux/module.h>
#include <linux/gpio.h>
#include <linux/interrupt.h>

// kernel_interrupt/keskeytys.c

static unsigned int keskeytyspalvelunro;
static irq_handler_t keskeytyskasittely(unsigned int irq, void *dev_id, struct pt_regs *regs);

static int __init keskeytys_init(void)
{
    // ACT-led tilaan 0
    gpio_direction_output(48, 0);

    // Fyysinen gpio pinni 3 inputiksi
    gpio_direction_input(252);

    // karkivarauhtelyn arvioitu kesto max 150 ms
    gpio_set_debounce(252, 150);

    // keskeytyspalvelun identifikaattori
    keskeytyspalvelunro = gpio_to_irq(252);

    // Rekisteroidaan keskeytyskasittely: identifikaattori, keskeytysfunktio, laskeva reuna, palvelun alias
    if(request_irq(keskeytyspalvelunro, (irq_handler_t) keskeytyskasittely, IRQF_TRIGGER_FALLING, "painikekeskeytys", NULL)){
        return -1;
    }

    printk(KERN_ALERT "Keskeytysmoduuli init, led: %d painike %d, keskeytyspalvelunro: %d \n", gpio_get_value(48), gpio_get_value(252), keskeytyspalvelunro);
    return 0;
}

static void __exit keskeytys_exit(void)
{
    // nollataan ACT-led
    gpio_direction_output(48, 0);
    // vapautetaan keskeytyspalvelu ja painiketoiminto
    free_irq(252, NULL);
    gpio_free(252);
    printk(KERN_ALERT "Keskeytysmoduuli poistettu ja resurssit vapautettu \n");
}

static irq_handler_t keskeytyskasittely(unsigned int irq, void *dev_id, struct pt_regs *regs)
{
    if(gpio_get_value(48) == 1)
    {
        gpio_direction_output(48, 0);
    }
    else
    {
        gpio_direction_output(48, 1);
    }

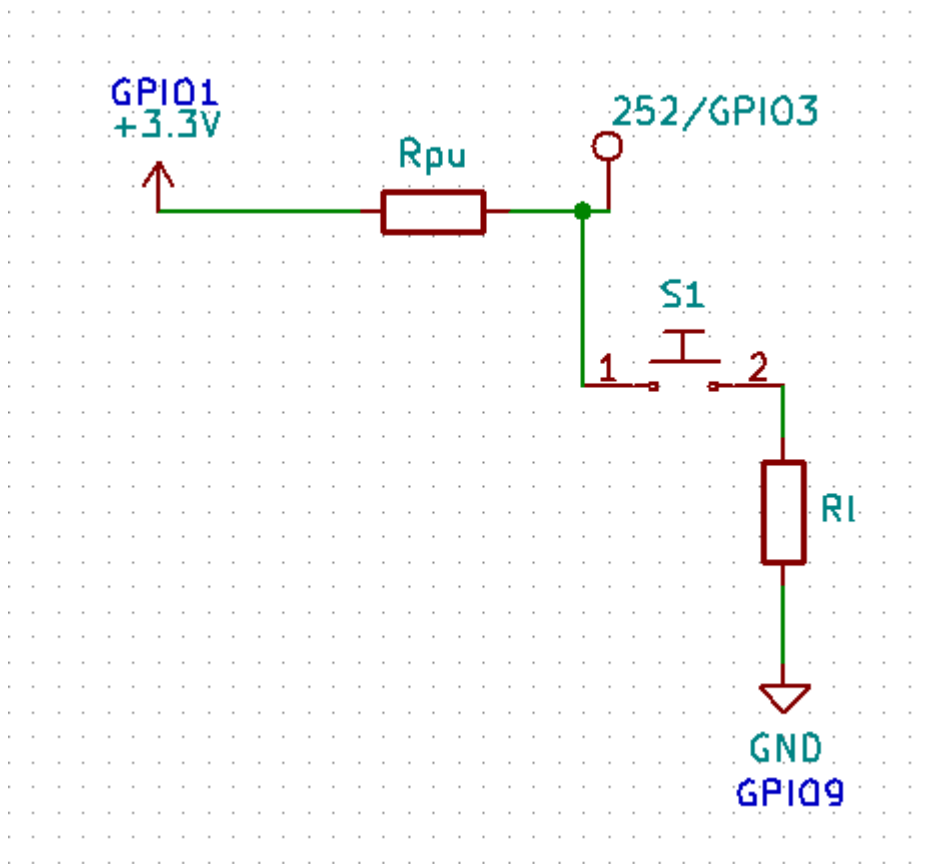
    printk(KERN_ALERT "Keskeytyspalvelu, led: %d painike %d \n", gpio_get_value(48), gpio_get_value(252));
    return (irq_handler_t) IRQ_HANDLED;
}

// Moduulin asennushetken funktio
module_init(keskeytys_init);
// poistohetken funktio
module_exit(keskeytys_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("ROGER KUPARI");
MODULE_VERSION("1");
```

Kuva 5 esimerkkikeskeytyskasittelymoduulin koodi

Kyseinen koodi löytyy edellä mainitusta github-polusta -> kernel_interrupt/keskeytys.c

Kuvassa 5 on edelleen esitetty kytkentäkuva.



Kuva 6 painikkeen kytkentäkaavio

Kytkenän alustus lyhyesti:

Kuormavastusta (RI) ei välttämättä tarvitse kytkeä, mutta kyseessä on kuitenkin opiskelijan budjetilla arvokas lauta, joten en halunnut ottaa riskejä. GPIO3 maksimivirta on 12 mA.

- Tällä kytkennällä ylösvetovastus (Rpu) täytyy olla suurempi arvoinen kuin kuormavastus.
- Kuten kuvasta ilmenee, lepotilassa pinni on asennossa "1".
- GPIO3 on Asuksen asetuksilla I2C1-väylän datalinja, sillä ei kuitenkaan ole merkitystä tässä toteutuksessa.

kernel_interrupt/keskeytys.c:n selostus

Kerron tässä vain sen, mikä on oleellisesti lisätty tätä toiminnallisuutta varten. Perusasiat rungosta ilmenevät esimerkin 1 alta. Asiat ovat selostettu koodista katsoen ylhäältä alaspäin.

Eli koodista ilmenee, että on otettu uusi headeri: linux/interrupt.h jonka avulla alustetaan keskeytysrutiini.

Valitsin input-pinniksi GPIO3 nastan fyysisesti ja siihen viitataan arvolla 252. Funktiolla gpio_set_debounce on annettu painikkeen karkivärähtelylle aikaa 150 millisekuntia, tämäkään ei ole käytetylle tactile switcille riittävä, mutta tässä tulee ajatus selville ja demo toimii riittävän hyvin. Edelleen funktiolla

gpio_to_irq pyydetään kyseisen pinnin keskeytyspalvelun identifioiva numero ja se tallennetaan globaaliin muuttujaan. Tässä olivat uudet gpio.h headerin toiminnallisuudet keskeytys_init -funktiossa.

Seuraavaksi keskeytys_init -funktiossa pyydetään keskeytyspalvelua interrupt.h:n funktiolla request_irq. Keskeytyspalvelua pyydetään edellä kysytylle keskeytyspalvelulle osoittamalla sitä identifioivalla numerolla, osoitetaan keskeytykselle käsittelyfunktio (keskeytyskäsitteily), keskeytyksen tuottaa tapahtuman laskeva reuna ja tämän keskeytyspalvelun aliaksena toimii ”painikekeskeytys”. Jos keskeytyspalvelu ”myönnetään” pyynnön mukaisesti, moduulin asentamisesta tulee lokiin alustusilmoitus, jossa kerrotaan painikkeen&ledin tilat sekä keskeytyspalvelun numero (”identifikaattori”).

Eli jos koodissa on päästy tähän asti, niin moduuli ei tee mitään aina siihen asti kunnes tulee keskeytys. Keskeytyspyynnön tapahtuessa moduuli siirtyy suorittamaan irq_handler_t keskeytyskasittely - funktiota. Funktion toiminnallisuus on käännellä ACT-ledin tilaa vuoroin 1:ksi ja vuoroin 0:ksi. Eli ensimmäinen painikkeen painallus sytyttää sen ja toinen sammuttaa sen jne. Lisäksi merkinnät lokiin keskeytys suorituksen toimista arvoineen.

Kun moduuli poistetaan (keskeytys_exit) -> ACT-ledin tila palautetaan 0:aan sekä vapautetaan pyydetty gpio ja keskeytysresurssit painikkeen osalta. Tästä tulee myös lokimerkintä.

Lokimerkinnät (asennus-keskeytystoimintoja-poistaminen):

```
[ 1279.067633] keskeytysmoduuli init, led: 0 painike 1, keskeytyspalvelunro: 334
[ 2558.386535] keskeytyspalvelu, led: 1 painike 0
[ 2558.588646] keskeytyspalvelu, led: 0 painike 0
[ 2559.523083] keskeytyspalvelu, led: 1 painike 0
[ 2559.794221] keskeytyspalvelu, led: 0 painike 0
[ 2560.886197] keskeytyspalvelu, led: 1 painike 0
[ 2561.715147] keskeytyspalvelu, led: 0 painike 0
[ 2561.720409] keskeytyspalvelu, led: 1 painike 1
[ 2562.804609] keskeytyspalvelu, led: 0 painike 0
[ 2564.375399] keskeytyspalvelu, led: 1 painike 0
[ 2566.206042] keskeytyspalvelu, led: 0 painike 0
[ 2567.666525] keskeytyspalvelu, led: 1 painike 0
[ 2568.110965] keskeytyspalvelu, led: 0 painike 0
[ 2568.116197] keskeytyspalvelu, led: 1 painike 1
[ 2569.091452] keskeytyspalvelu, led: 0 painike 0
[ 2569.539944] keskeytyspalvelu, led: 1 painike 0
[ 2571.295008] keskeytyspalvelu, led: 0 painike 0
[ 2602.550396] Keskeytysmoduuli poistettu ja resurssit vapautettu
```

Kuva 7 Keskeytysmoduulin lokimerkinnät

Koodissa nimetty alias ja keskeytyspalvelun identifikaattori ilmenee tarkastelemalla /proc/interrupts.

lähteitä: [gpio.h / Torvalds, interrupt.h / Torvalds, Interrupts and Interrupt Handlers]

6 ESIMERKKI 3 [kernel-moduulin toimintojen ohjaus käyttäjätilasta]

Tämän viimeisen esimerkin tarkoituksena on luoda kernelmoduuli, character device sekä ohjauskoodi käyttäjätilasta. Eli ohjata lediä käyttäjätilasta käyttäen tehtyä kernelmoduulia. Kommunikointiväylän ominaisuudessa toimiva character devicen tehtävä on toimia rajapintana käyttäjäkoodin sekä moduulin välillä.

Heti alkuun: Loistava artikkeli character devicen toiminnasta esimerkkeineen: <http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device/>

Ylläolevassa lähteessä character device osuuden runko on toteutettu hieman eri tavalla kuin tässä raportissa esiteltävä, mutta antaa hyvin taustatietoa.

Yleiskuvaus esimerkki 3:n paketista:

Itse moduulin lähdekoodi ja sen makefile sijaitsee `drivers_public/character_led` polussa (`ledcmd.c`). Käyttäjätilankoodi sijaitsee edelleen alikansiossa `uspace`. Käyttäjätilan koodista oli raportin vastaanottaja toivonut modulaarista, joten se on toteutettu modulaarisesti: `ohjaus.c` -on pääohjelma. `Toiminto.c` - käsittää kommunikointirungon moduulille ja headerina toimii `ohjaus.h`.

- Halusin pitää käyttäjätilan koodin selkeänä ja yksinkertaisena, joten en lähtenyt soololemaan.

Makefilet: Makefilet ovat sekä moduulilähdekoodin kansiossa, että käyttäjätilaan omansa. Käyttäjätilan `make` ajetaan moduulin käännöksen yhteydessä. Käyttäjätilan ohjauskomento sijoitetaan myös `/usr/local/bin` -kansioon, jolloin sitä voi kutsua mistä vain. Käyttäjätilan ohjauskomento tulee ajaa pääkäyttäjän oikeuksin; `sudo ledohjaus led x`, jossa `x` on numero 1 tai 0.

Moduuli (`drivers_public/character_led/ledcmd.c`):

uudet headerit: `linux/fs.h`, `-uaccess.h`, `-`, `kernel.h`, `-cdev.h` rivillä 6 – 10, avaan käytettyjä toiminnallisuksia esittelyn yhteydessä. Koodin avaan viitaamalla `<rivi>`.

`<24-29>` Globaalit muuttujat ja funktioiden prototyytit.
`dev_t` muuttuja pitää sisällään laitenumerot ”device numbers”, joka on esimerkin koodissa vain esitelty tässä kohtaa. `Cdev` -tietueeseen viitataan myöhemmässä koodissa ”rajapintana”. Laiteluokka on määritetty tarkemmin `linux/device.h`:ssa joka on sisällytetty `cdev.h`:n. globaalina muuttujana on myös yhden tavun kokoinen viesti muuttuja, johon napataan käyttäjän kirjoittama sanoma.

<34-43>

Fileoperaatiot

Aluksi on esitetty funktioiden prototyypit, joista ”lue_data” -funktiolla ”luetaan” käyttäjän kirjoittama data. ”suljettu” -funktio suoritetaan kun tämän esimerkin mukainen character device suljetaan.

Eli kuten varmaan lukijakin huomaa rivillä 40 on määritetty ”.write = lue_data”. Oikeastaan se on loogisempaa näin päin, sillä kun teen moduulia kerneliin niin myös ajatukseni ovat koodia kirjoittaessa siellä rajan toisella puolen.

- Tässä yhteydessä ”write” tarkoittaa sitä, että **käyttäjä kirjoittaa** character deviceen, eli funktion toiminta linkittyy käyttäjän kirjoitustoimeen. Halusin kuitenkin ohjata moduulin lukemaan käyttäjän kirjoitusta, joten nimesin funktion sen mukaisesti. Asia ei kuitenkaan ole ihan näin yksinkertainen kokonaisuudessaan, avaan hetken päästä lisää.
- Toivoin tässä vaiheessa, kun kirjoitin koodia, että olisin selvinnyt vain lue_data toiminnallisuudella, mutta siinä meni pari tuntia tutkiessa kun ei toiminut. Nimittäin, jos moduulissa ei ole mitään funktiota mihin se osoittaa kirjoitustoiminnan jälkeen (esimerkiksi tiedoston sulkemista) niin se ilmeisesti suorittaa kyseisen lue_data toiminnon loppuun vasta kun seuraava kirjoitustapahtuma / pyyntö tulee.

Sama suomeksi: Jos ”release” -ei ole käytössä tässä moduulissa, käyttäjän kirjoittama teksti saadaan käyttöön vasta aina seuraavan kirjoitustapahtuman yhteydessä. Kun ”release” on nyt käytössä, sen mukainen viesti tulee vasta moduulin käyttöön, kun uudelleen käyttäjä alkaa kirjoittamaan.

<46-77>

lue_data -alustus

Eli tämä funktio tulee suoritukseen kun tiedostoon on kirjoitettu, se suoritetaan jostain kohti loppuun kun seuraava operaatio aloitetaan, mikäli se seuraava operaatio on alustettu moduulissa.

Ensimmäiseksi kopioidaan käyttäjän kirjoittama sanoma copy_from_user -funktiolla joka on alustettu uaccess -headerissa. Jos funktio palauttaa negatiivisen luvun, tarkoittaa se epäonnistumista.

Seuraavaksi sanomaa verrataan strcmp:llä minkä toiminnallisuus on täysin sama kuin käyttäjätilan c-koodissakin. Kernelin puolella toiminnallisuus on alustettu linux/string.h headerissa, tieto löytyi googlella. En kuitenkaan kyennyt löytämään helposti sitä, mitä kautta kyseinen funktio linkittyy tähän käsiteltävään moduliin. Tämä on yksi syy, miksi koen Linux-moduulin kehittämisen jokseenkin haastavaksi kokonaisuutena. Joka tapauksessa, jos viesti on ”1” ledi ohjataan päälle ja ”0”:n tapauksessa pois päältä. Viestin käsittely ilmenee tarkemmin, kun käsittelen käyttäjätilapään koodia.

Suljettu -funktio tulee suoritukseen, kun tiedosto suljetaan. Sen sisällä oleva koodi kirjautuu lokiin kun tiedostoon kirjoitetaan seuraavan kerran.

<82-127> Moduulin alustus, character devicen alustus, ledin alkutoimet
Eli ensimmäisenä uutena funktiona tulee character devicen pino-alueen varaus. Kyseinen funktio on esitelty fs.h:ssa, externinä ja se on alustettu fs/char_device.c:ssä. Parametreinä funktiolle välitetään

1. laitenumerot "dev_t"
2. baseminor {first of the requested range of minor numbers}
3. kuinka monta minor numeroa halutaan?
4. toiminta-alueen nimi (ei device characterin nimi)

Tämän jälkeen luodaan luokka, jonka omistaja on kyseinen moduuli ja nimenä toimii "lediohjaus". Parametreinä seuraavat:

1. osoitin luokkaan
2. parent device (jos on)
3. dev_t
4. driverdata
5. ryhmä (laitteen nimi)
6. muuttujat? (device.h = const char *fmt, ...)

Cdev_init - Character device -tietueen alustaminen, ennen laittoa järjestelmän käyttöön.
Cdev_add funktiolla laitetaan character device välittömästi järjestelmään ja se on käytettävissä, jos riveillä 82-127 ei ole esiintynyt virheitä.

Tulos:

Moduuli luo "automaattisesti" character devicen dynaamisesti haetuilla major ja minor numeroilla, joka on välittömästi käytössä. Eli esimerkin tapauksessa ko. character device syntyy nimellä "led".

>> tämä character device toimii kommunikointirajapintana käyttäjän ja moduulin välillä. Kehittäjän määrittämin ominaisuuksin. Ominaisuudet määritellään file-operaatioilla, joita on laajasti. Tarvittaessa suosittelen tarkastelemaan tarkemmin ominaisuuksia file_operations tietuetta headerin fs.h lähdekoodista.

>> Raporttia kirjoittaessa huomasin ko. tietueessa kohdan: int (*flush) (struct file *, fl_owner_t id); Ehkäpä siinä olisi ratkaisu edellä esitettyyn suoritusjärjestys-ongelmaan.

<133 – loppukoodi>

Funktioiden nimistä selviää aika hyvin moduulin exit-funktion toiminta. Vapautetaan kaikki mitä ollaan init -funktiossa määritetty.

-init funktion toiminta on edellä avattu riveillä 82-127, joten sen lukemalla ymmärtää hyvin exit -funktion uudet toiminnallisuudet (mitä ei ole aikaisemmissa esimerkeissä näytetty).

Käyttäjätilankoodi (drivers_public/character_led/ospace/)

Mielestäni perustason C-koodia, jos jaksoit lukea moduulin selostuksen niin ymmärrät varmasti, miksi halusin tehdä käyttäjätilan koodista mahdollisimman simppelin. Sellaisen kuitenkin, että se täyttää modulaarisuuden määritelmän.

Käyttäjätilan pääohjelma: ohjaus.c

Ohjelma ottaa argumentin ja hyväksyy sen muodossa: led <numero>

Ohjelmaa tulee kutsua pääkäyttäjän oikeuksin.

Kriteerit täyttävä argumentti välitetään toteuta -funktioon, joka sijaitsee toiminto.c:ssä.

Se on linkitetty ohjaus -headerin kautta.

toiminto.c

Toteuta-funktio ottaa aikaisemman argumentin kokonaisuutena parametrinä. Koska tässä vaiheessa on jo tarkastettu, että argumentti on täyttänyt vaadittavat muodollisuudet, niin siitä kirjoitetaan vain jälkimmäinen osa character devicelle.

- Eli 1 tai 0
 - o Esimerkissä on vain yksi ledi ohjattavana.

ohjaus.h

Käytännössä sisältää vain funktion prototyypin.

6.1 Moduulin makefile lisäykset

```
GNU nano 2.7.4 File: Makefile

obj-m = ledcmd.o

moduuli = ledcmd
vers = $(shell uname -r)

all:
    make -C /lib/modules/$(vers)/build M=$(shell pwd) modules
    sudo cp $(moduuli).ko /lib/modules/$(vers)
    sudo depmod -a
    make -C $(shell pwd)/uspace
    sudo modprobe ledcmd
    sudo cp $(shell pwd)/uspace/ledohjaus /usr/local/bin

clean:
    make -C /lib/modules/$(vers)/build M=$(shell pwd) clean
    sudo rm /lib/modules/$(vers)/$(moduuli).ko
    sudo depmod -a
    make -C $(shell pwd)/uspace clean
    $(shell sudo modprobe -r ledcmd)
    sudo rm /usr/local/bin/ledohjaus
```

Kuva 8 Character devicemoduulia varten tehty muutoksia makefileen

Itseasiassa kuten mainitsin aikaisemmin, makefileja on kaksi. Tässä esiteltävä makefile hoitaa kaiken tarpeellisen, eli ajaa myös käyttäjätilan maken.

Uutta:

Kolme alinta rivia kummastakin osiosta (selostettuna ylhäältä alas):

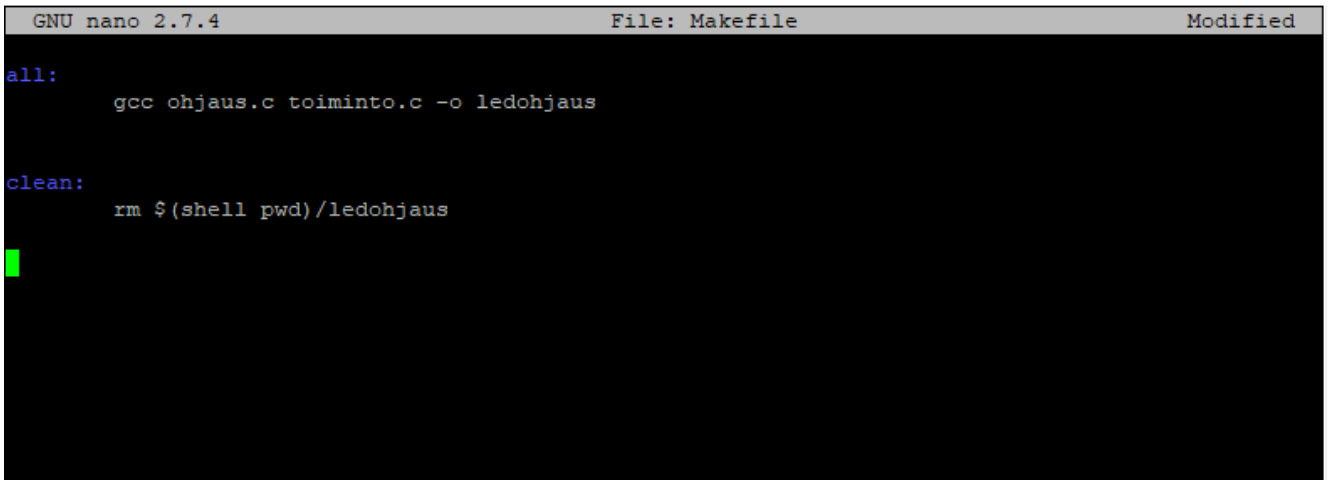
all:

aja käyttäjätilan koodin makefile
asenna moduuli kerneliin
tee käyttäjätilan koodikutsusta kaikkialta kutsuttava

clean:

käyttäjätilan makefilen clean osion suoritus
poista moduuli kernelistä
poista käyttäjätilan koodikutsun globaali ominaisuus

6.2 Käyttäjätilakoodin makefilen sisältö



```
GNU nano 2.7.4           File: Makefile           Modified
all:
    gcc ohjaus.c toiminto.c -o ledohjaus

clean:
    rm $(shell pwd)/ledohjaus
```

Kuva 9 käyttäjätilan koodin makefile

all:

→ käännetään lähdekoodi

clean:

⇒ poistetaan suoritusosa

Harjoitusehdotuksia esimerkkiin 3:

Muokkaa koodia niin, että mitä tahansa pinniä voidaan ohjata käyttäjätilan koodilla tilaan 1/0.

- Vaikeusastetta saa lisää, jos haluaa lisäksi määrittää käyttäjän ohjauksella suunnan ja lisäämällä keskeytyskäsitteilyn moduuliin.
- Tottakai on mahdollista laittaa moduuli keskustelemaan käyttäjätilan suuntaan myös character devicen kautta + asianmukaiset käsittelyt.

6.3 Esimerkki 3:n yhteenveto

Koodimäärä tässä viimeisessä esimerkissä on karkeasti arvioituna kymmenkertainen verrattuna esimerkki 1:n toimintaan.

Kokonaistoiminnallisuus: sovellusmainen.

Koko paketin (moduuli, moduulin asennus, käyttäjätilan koodin alustukset jne.) asennus: yhdellä komennolla: **make**

Koko paketin siivous yhdellä komennolla: **make clean**

Sovelluksen käyttösyntaksi: `sudo ledohjaus <numero>`

Sama kuvina:

```
user@tinkerboard:~/drivers_public/character_led$ pwd
/home/user/drivers_public/character_led
user@tinkerboard:~/drivers_public/character_led$ ls
ledcmd.c Makefile uspace
user@tinkerboard:~/drivers_public/character_led$ make
make -C /lib/modules/4.19.14-rockchip/build M=/home/user/drivers_public/character_led modules
make[1]: Entering directory '/usr/src/linux-headers-4.19.14-rockchip'
  CC [M]  /home/user/drivers_public/character_led/ledcmd.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/user/drivers_public/character_led/ledcmd.mod.o
  LD [M]  /home/user/drivers_public/character_led/ledcmd.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.19.14-rockchip'
sudo cp ledcmd.ko /lib/modules/4.19.14-rockchip
[sudo] password for user:
sudo depmod -a
make -C /home/user/drivers_public/character_led/uspace
make[1]: Entering directory '/home/user/drivers_public/character_led/uspace'
gcc ohjaus.c toiminto.c -o ledohjaus
make[1]: Leaving directory '/home/user/drivers_public/character_led/uspace'
sudo modprobe ledcmd
sudo cp /home/user/drivers_public/character_led/uspace/ledohjaus /usr/local/bin
user@tinkerboard:~/drivers_public/character_led$
```

Kuva 10 make

kaikki tarvittava hoituu siis tällä. Lokista seurauksena ilmenee, että:

```
[ 235.931985] Moduuli asennettu, ledin ohjaus on tilassa: 0
user@tinkerboard:~/drivers_public/character_led$
```

Kuva 11 lokimerkintä heti maken jälkeen.

Ohjataan ledi päälle käyttäjätilasta:

```
user@tinkerboard:~/drivers_public/character_led$ sudo ledohjaus led 1
komento suoritettu
user@tinkerboard:~/drivers_public/character_led$
```

Kuva 12 lediohjaus käyttäjätilan koodia käyttäen

Tarkastellaan lokia heti käyttäjätilan ohjauksen jälkeen:

```
[ 235.931985] Moduuli asennettu, ledin ohjaus on tilassa: 0
[ 463.707061] ledinohjauskomento vastaanotettu 1
[ 463.707106] ledi ohjattu tilaan 1
```

Kuva 13 ohjaus on suoritettu, ledi palaa

Ja nyt seuraavaksi näkyy tilanne, että vasta seuraavan ohjauskomennon (sudo ledohjaus led 0) yhteydessä tulee ”suljettu” -suoritusmerkintä moduulilta { .release, käsitelty koodirivit 34-43 ja 47-77, sivu 24}.

```
[ 235.931985] Moduuli asennettu, ledin ohjaus on tilassa: 0
[ 463.707061] ledinohjauskomento vastaanotettu 1
[ 463.707106] ledi ohjattu tilaan 1
[ 463.712689] /dev/led suljettu
[ 694.908535] ledinohjauskomento vastaanotettu 0
[ 694.908580] ledi ohjattu tilaan 0
```

Kuva 14 Ledi sammutettu ja tiedoston sulkeminen ilmentynyt

Kokeillaan vielä muutama ohjauskomento, jotka eivät ole muodollisesti päteviä:

```
user@tinkerboard:~/drivers_public/character_led$ ledohjaus 0
virhe, ohjaus on muotoa: led numero
user@tinkerboard:~/drivers_public/character_led$ ledohjaus ledi 0
Virhe, ensimmäinen argumentti oltava 'led'
user@tinkerboard:~/drivers_public/character_led$ ledohjaus led 0
/dev/led - ei voida avata, tarkista tiedoston nimi; suoritettava paakayttajan oikeuksin
virhe kirjoitettaessa komentoa /dev/led
komento suoritettu
user@tinkerboard:~/drivers_public/character_led$ sudo ledohjaus 3
virhe, ohjaus on muotoa: led numero
user@tinkerboard:~/drivers_public/character_led$
```

Kuva 15 käyttäjätilan ohjauskomentojen tulkkausta

Ja kaksi viimeistä kuvaa: (15) make clean -> sovellusominaisuuksien poistaminen, (16) lokimerkinnät alusta loppuun.

```
user@tinkerboard:~/drivers_public/character_led$ make clean
make -C /lib/modules/4.19.14-rockchip/build M=/home/user/drivers_public/character_led clean
make[1]: Entering directory '/usr/src/linux-headers-4.19.14-rockchip'
  CLEAN    /home/user/drivers_public/character_led/.tmp_versions
  CLEAN    /home/user/drivers_public/character_led/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-4.19.14-rockchip'
sudo rm /lib/modules/4.19.14-rockchip/ledcmd.ko
sudo depmod -a
make -C /home/user/drivers_public/character_led/ospace clean
make[1]: Entering directory '/home/user/drivers_public/character_led/ospace'
rm /home/user/drivers_public/character_led/ospace/ledohjaus
make[1]: Leaving directory '/home/user/drivers_public/character_led/ospace'
sudo rm /usr/local/bin/ledohjaus
user@tinkerboard:~/drivers_public/character_led$
```

Kuva 16 make clean

```
[ 235.931985] Moduuli asennettu, ledin ohjaus on tilassa: 0
[ 463.707061] ledinohjauskomento vastaanotettu 1
[ 463.707106] ledi ohjattu tilaan 1
[ 463.712689] /dev/led suljettu
[ 694.908535] ledinohjauskomento vastaanotettu 0
[ 694.908580] ledi ohjattu tilaan 0
[ 694.914437] /dev/led suljettu
[ 997.215944] Moduuli poistettu, ledin ohjaus on tilassa: 0
```

Kuva 17 Lokimerkinnät

7 YHTEENVETO

Tiedonhaku ja sen soveltaminen, varmaankin ehdottomia vaatimuksia kun etsitään ajurikehittäjää Linux ympäristöön. Itse ainakin koen sen jollakin tasolla epämukavaksi, kun tieto tuntuu olevan hajautettuna pieninä paloina ympäri internetiä – ainakin jos jotain ytimeen liittyvää haluaa tehdä.

Se, että asioita voi yrittää edes ymmärtää tällä ruohonjuuritasolla, täytyy olla jonkinnäköistä tietoa Linux-järjestelmästä ja sen toimintaperiaatteesta.

Olen mielestäni pyrkinyt laittamaan tähän raporttiin tarpeellisen tiedon siitä, miten lediä pystytään vilkuttelemaan näillä esimerkkien kerneltoiminnoilla. Olen pyrkinyt esittämään esimerkit niin yksinkertaisesti kuin ikinä osaan. Kappaleen 6 yleiskuvaus esimerkki 3:n paketista -osa on kyllä melko kuivaa luettavaa, sitä luettaessa kannattanee varmaan pitää koodi auki toisella näytöllä samaan aikaan.

Tämä viimeinen esimerkki toi myös toteutuksellisia haasteita. Yritin tehdä siitä tiiviin ja selkeän pake-
tin moduulin osalta, mutta olisikohan siinä kuitenkin yli 100 riviä puhdasta koodia. Riviväleinen näyttää olevan 161 riviä.

Itseasiassa onnistuin laittamaan koko käyttöjärjestelmän juntturaan muutamia kertoja kun esimerkki 3:sta tein.

- Siinä oli myös yksi hyvä esimerkki siitä, että jos ydin jää deadlockiin tai muuhun virhetilaan kehitysvaiheessa = on hyvä, että moduuli ei ole bootin yhteydessä asentuvaa sorttia.
 - o Ainakin omat virhetilanteet korjaantuivat katkaisemalla virrat ja kytkemällä ne takaisin. Ainoat haittapuolet tässä oli, että satunnaisesti tällainen virhetilanne kuitenkin hävitti jo tallennettua tekstiä tiedostosta.

8 LÄHTEET

mmap: <http://man7.org/linux/man-pages/man2/mmap.2.html>

kernel introduction: <http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/>

ytimen ja käyttäjätilan eroja: <https://blog.codinghorror.com/understanding-user-and-kernel-mode/>

kernel vs user memory space: <http://www.cs.umd.edu/class/spring2013/cmsc412/DiscussionSection2.pdf>

Kernelin api: <https://www.kernel.org/doc/html/v4.19/driver-api/>

Bootlin: <https://elixir.bootlin.com/linux/v4.19/source/include/linux/>

init.h / Torvalds: <https://github.com/torvalds/linux/blob/master/include/linux/init.h>

module.h / Torvalds: <https://github.com/torvalds/linux/blob/master/include/linux/module.h>

gpio.h / Torvalds: <https://github.com/torvalds/linux/blob/master/include/linux/gpio.h>

printk : https://elinux.org/Debugging_by_printing

interrupt.h / Torvalds: <https://github.com/torvalds/linux/blob/master/include/linux/interrupt.h>

Interrupts and Interrupt Handlers: <https://notes.shichao.io/lkd/ch7/>

Character device: <http://derekmolloy.ie/writing-a-linux-kernel-module-part-2-a-character-device/>

major/minor numbers: https://www.ibm.com/support/knowledgecenter/en/linuxonibm/com.ibm.linux.z.lgdd/lgdd_c_udev.html

fs.h / Torvalds: <https://github.com/torvalds/linux/blob/master/include/linux/fs.h>

uaccess.h / Torvalds: <https://github.com/torvalds/linux/blob/master/include/linux/uaccess.h>

kernel.h / Torvalds: <https://github.com/torvalds/linux/blob/master/include/linux/kernel.h>

cdev.h / Torvalds: <https://github.com/torvalds/linux/blob/master/include/linux/cdev.h>