

# Izgradnja jednostavna modula za Linuxovu jezgru

---

**Maloić, Dominik**

**Undergraduate thesis / Završni rad**

**2022**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:211:632549>

*Rights / Prava:* [Attribution 3.0 Unported](#)/[Imenovanje 3.0](#)

*Download date / Datum preuzimanja:* **2024-12-19**



*Repository / Repozitorij:*

[Faculty of Organization and Informatics - Digital Repository](#)



**SVEUČILIŠTE U ZAGREBU  
FAKULTET ORGANIZACIJE I INFORMATIKE  
VARAŽDIN**

**Dominik Maloić**

**IZGRADNJA JEDNOSTAVNA MODULA ZA  
LINUXOVU JEZGRU**

**ZAVRŠNI RAD**

**Varaždin, 2022.**

**SVEUČILIŠTE U ZAGREBU**  
**FAKULTET ORGANIZACIJE I INFORMATIKE**  
**V A R A Ž D I N**

**Dominik Maloić**

**Matični broj: 0016131955**

**Studij: Informacijski sustavi**

**IZGRADNJA JEDNOSTAVNA MODULA ZA LINUXOVU JEZGRU**

**ZAVRŠNI RAD**

**Mentor:**

Luka Milić, mag. ing. comp.

**Varaždin, rujan 2022.**

*Dominik Maloić*

### **Izjava o izvornosti**

Izjavljujem da je moj završni rad izvorni rezultat mogega rada te da se u izradbi istoga nisam koristio drugim izvorima osim onima koji su u njem navedeni. Za izradbu rada su rabljene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredaba u sustavu FOI-radovi*

---

## Sažetak

U ovom radu proučava se jezgra operacijskoga sustava Linux, koje sastavnice čine jezgru i što jezgra čini. Zatim se opisuje proces izgradnje dodatnih modula za jezgru, isto tako se prolazi kroz potrebne programe za izradbu modula za Linuxovu jezgru. Objašnjavaju se osnovne funkcije tih programa i različite funkcije rečenih. Isto tako se opisuje izradba *makefilea* i što je uopće „*makefile*“. Isto tako se izgrađuje jednostavan modul Linuxove jezgre koji se instalira, ispituje i ugrađuje u Linuxovu jezgru. Nakon ugradnje modula preinačena jezgra operacijskoga sustava se prevodi, instalira, pokreće, ispituje i dokumentira na virtualnu kao i na stvarnu računalo.

**Ključne riječi:** Linux, jezgra, modul, programski jezik C, *makefile*, operacijski sustav, kbuild, virtualno računalo

# Sadržaj

Sadržaj.....	iii
1. Uvod.....	1
2. Linux .....	2
2.1. Linuxova jezgra i funkcije Linuxove jezgre .....	2
2.1.1. Funkcije jezgre u OS-u.....	4
3. Jezgreni moduli.....	6
3.1. Izgradnja jezgrenih modula i potrebni programi .....	6
4. Izrada jednostavnih jezgrenih modula .....	10
4.1. „Helloworld“-modul i modul “Aktivni procesi” .....	10
5. Prevođenje, instalacija, pokretanje, ispitivanje i dokumentacija preinačene jezgre operacijskoga sustava.....	16
5.1. Prevođenje jezgre.....	16
5.1.1. Prevođenje preko PKGBUILD-a.....	16
5.1.2. Prevođenje Linuxove jezgre iz izvornoga koda .....	17
5.2. Instalacija, pokretanje, ispitivanje i dokumentacija modulom preinačene jezgre operacijskoga sustava.....	22
5.3. Prevođenje, instalacija, pokretanje, ispitivanje i dokumentacija preinačene jezgre operacijskog sustava na fizičkom računalu .....	27
6. Zaključak.....	31
Popis literature .....	32
Popis slika .....	33

# 1. Uvod

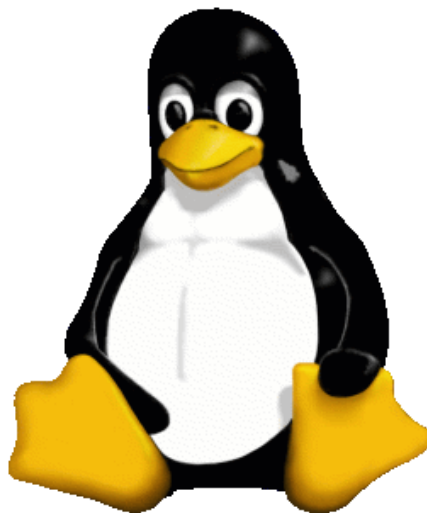
U današnje vrijeme većina ljudi rabi jedan od triju operacijskih sustava na računalima, Windows, macOS ili Linux. U proteklih deset godina popularnost Linuxa kao i FOSS-a (engl. *free and open source software*) je u rastu, ponajviše zbog odluka Microsofta i Applea da uključuju sve više telemetrije u svoje proizvode. Linuxova jezgra (engl. *kernel*) je FOSS tako da svatko ima priliku zaviriti i znati precizno što se nalazi u njihovom računalu. Istodobno to značnuje da svatko može i preinačiti tu jezgru da bi odgovarala njegovim potrebama, a upravo to je i cilj ovoga rada. Preinaka jezgre će se obaviti preko dodavanja modula u izvorni kod jezgre, preinake *makefile*-datoteka kako bi se moduli ispravno preveli prigodom prevođenja jezgre. Zatim će se ta preinačena jezgra instalirati, pokrenuti i ispitati u virtualnom kao i u stvarnom okruženju.

Linuxovi moduli su komadi koda koji se mogu dinamički povezati u jezgru u kojem bilo trenutku pošto se sustav podigne. Mogu se odvojiti od jezgre i ukloniti kad više nisu potrebni. Uglavnom moduli Linuxove jezgre su upravljački programi uređaja, pseudopokretački programi uređaja kao što su mrežni upravljački programi ili datotečni sustavi.

Moduli jezgre Linuxa mogu se učitati i isključiti izrijekom s pomoću naredaba *insmod* i *rmmmod* ili sama jezgra može zahtijevati da demon jezgre (*kerneld*) učitava i istovaruje module po potrebi.

## 2. Linux

Linux je naziv za jezgru računalnoga operacijskoga sustava, ali se rabi i za naziv cijeloga operacijskoga sustava temeljenoga na toj jezgri. Svoje ime je dobio po svojem tvorcu Linusu Torvaldsu koji je izvorni kod objavio na Internetu 1991. godine te je otvorio projekt drugim zainteresiranim programerima kako bi projekt mogao dalje rasti i razvijati se. Mnogo programera se odlučilo na daljnju kolaboraciju, tako da je jezgra Linuxa zajedničko djelo programera diljem svijeta, operacijski sustav nastao na jezgri Linuxa je djelo zajednice tisuća programera.



Slika 1. Linuxov logotip „Tux“ (Izvor: Larry Ewing, 1996)

Za razvoj jezgre Linuxa kao i Linux operacijskog sustava zaslužni su GPL-licencija za porabu i brzi razvoj globalne komunikacijske mreže. To je omogućilo stvaranje i rast globalne zajednice suradnika – korisnika i programera, koji su omogućili da Linux dođe na naslovne stranice i postane prepoznatljiv naziv.

### 2.1. Linuxova jezgra i funkcije Linuxove jezgre

Linuxova jezgra je najvažnija sastavnica Linuxova operacijskoga sustava i glavna veza između računalnoga sklopovlja i njegovih procesa, komunicira između njih i efikasno dodjeljuje sredstva. Odgovorna je za povezivanje svih aplikacija koje se pokreću u „korisničkom načinu“ s fizičkim sklopovljem i omogućuje procesima znanima kao posluživači da međusobno komuniciraju preko međuprocesne komunikacije (IPC).



U slučaju operacijskoga sustava Linux jezgra se sastoji od 4 brojeva koji su razdvojeni točkicama u sljedećem obliću: AA.BB.CC.DD. Objasnidba oblića:

AA – odnosi se na trenutačnu verziju rabljene jezgre;

BB – trenutačna revizija rabljene jezgre;

CC – ova čestica nudi informacije o tom ima li jezgra manje izmjene;

DD – broj zakrpe.

Većina upravljačkih programa uređaja i proširaka jezgre radi u prostoru jezgre (engl. *ring 0* u mnogim CPU-arhitekturama), s punim pristupom sklopovlju. Neke iznimke rade u korisničkom prostoru; značajni primjeri su datotečni sustavi temeljeni na FUSE-u/CUSE-u i dijelovi UIO-a. Nadalje, X Window System i Wayland i protokoli posluživača za prikaz koje većina ljudi rabi s Linuxom ne rade unutar jezgre. S druge strane, stvarno sučelje s GPU-ima grafičkih kartica je podsustav unutar jezgre koji se zove Direct Rendering Manager (DRM).

Za razliku od standardnih monolitnih jezgara upravljački programi uređaja lako se konfiguriraju kao moduli i učitavaju ili isključuju dok sustav radi, a isto tako se mogu unaprijed isključiti pod određenim uvjetima kako bi ispravno obradili sklopovske prekide i bolje poduprli simetričnu višestruku obradbu. Po izboru, Linux nema stabilno binarno sučelje aplikacije upravljačkoga programa uređaja. Sklopovlje je predstavljeno u hijerarhiji datoteka. Korisničke aplikacije komuniciraju s upravljačkim programima uređaja preko unosa u kazalima */dev* ili */sys*. Informacije o procesima isto tako se preslikavaju na datotečni sustav kroz kazalo */proc*.

Postoje različiti načini za izgradnju jezgre i arhitektonskih razmatranja pri izgradnji jedne od ništice. Općenito, većina jezgara spada u jednu od tri vrste: monolitna, mikrojezgra i hibridna. Linux je monolitna jezgra, dok macOS (UNIX) i Windows 7 rabe hibridne jezgre.

Monolitna jezgra ostvaruje temeljne značajke računalnoga sustava kao što su upravljanje datotekama, memorijom i drugim sredstvima. OS-u daje primitivnu arhitekturu u kojoj su sva sredstva povezana s prostorom jezgre. Neki primjeri operacijskih sustava koji rabe monolitne jezgre su DOS, Solaris, AIX, Linux, OpenVMS itd. Monolitna jezgra odgovara sitnim zadacima kao što su raspoređivanje CPU-a, sustavski pozivi itd. Ali, zahvaljujući svojim značajkama kao što su pouzdanost, sigurnost i brzina pristupa, mnogi financijski projekti rabe operacijske sustave koji rabe monolitne jezgre. Sve sastavnice potpore za upravljanje sklopovljem potrebne za obradbu ugrađene su unutar jezgre i stoga mogu izravno međusobno komunicirati. Monolitna jezgra može dinamički učitavati module što stvara vrlo male troškove za razliku od ugradnje modula u sliku OS-a.

## Prednosti

- Monolitna jezgra je brza jer su usluge poput upravljanja memorijom, upravljanja datotekama itd. ostvarene u istom adresnom prostoru
- Proces se u potpunosti izvodi u jednom adresnom prostoru u monolitnim jezgrama
- Monolitna jezgra je jedna statična datoteka
- Malo pogrešaka i manje sigurnosnih problema
- Izvršavanje procesa je brzo zahvaljujući odvojenomu memorijskomu prostoru za korisnika i jezgru

## Nedostatci

- Ako koja bilo usluga zakaže, tad pada i cijeli sustav
- Ako se dodaju kakve bilo nove značajke, problem je u preinačivanju cijeloga sustava
- Kodiranje i otklanjanje pogrešaka u prostoru jezgre je teško
- Pogreške u jednom dijelu prostora jezgre isto tako proizvode snažne učinke u drugim dijelovima
- Ove jezgre su goleme i teške za održavanje i nisu uvijek prenosive

### 2.1.1. Funkcije jezgre u OS-u

#### Raspoređivanje procesa:

- Jezgra daje dio vremena svakomu procesu; kad proces završi s izvršavanjem, jezgra pokreće drugi proces; određuje stanje procesa, koje može biti pokretanje, čekanje ili završetak.

#### Raspodjela sredstava:

- Jezgra kontrolira memoriju, periferne uređaje i CPU-procese. Isto tako djeluje kao veza između sredstava i procesâ. Dodjeljuje memoriju procesima. Ako koji bilo proces zahtijeva pristup nekoj sklopovskoj sastavnici, jezgra mu dodjeljuje tu sastavnicu.

#### Upravljanje uređajima:

- Jezgra upravlja uređajima povezanima sa sustavom, kao što su I/O-uređaji, uređaji za pohranu itd., kao i razmjenu podataka preko tih uređaja. Informacije se primaju iz sustava i prenose u nj iz I/O-uređaja i različitih aplikacija.

Rukovanje prekidima i sustavski pozivi:

- Kad se proces izvodi, može se pojaviti zadatak visoka prioriteta koji se mora prvi izvršiti. Jezgra prebacuje kontrolu s trenutno pokrenutoga procesa na novi po njihovim prioritetima. Jezgra se isto tako bavi sustavskim pozivima, koji su jednostavno rečeno programski prekidi.

Upravljanje memorijom:

- Pošto jezgra stvori i izvrši proces, on živi u memoriji pošto je zauzeo prostor u njoj. Kad proces završi, jezgra uklanja proces iz memorije. Jezgra dodjeljuje memoriju za obradbu i isto tako ju oslobađa.

Upravljanje procesom:

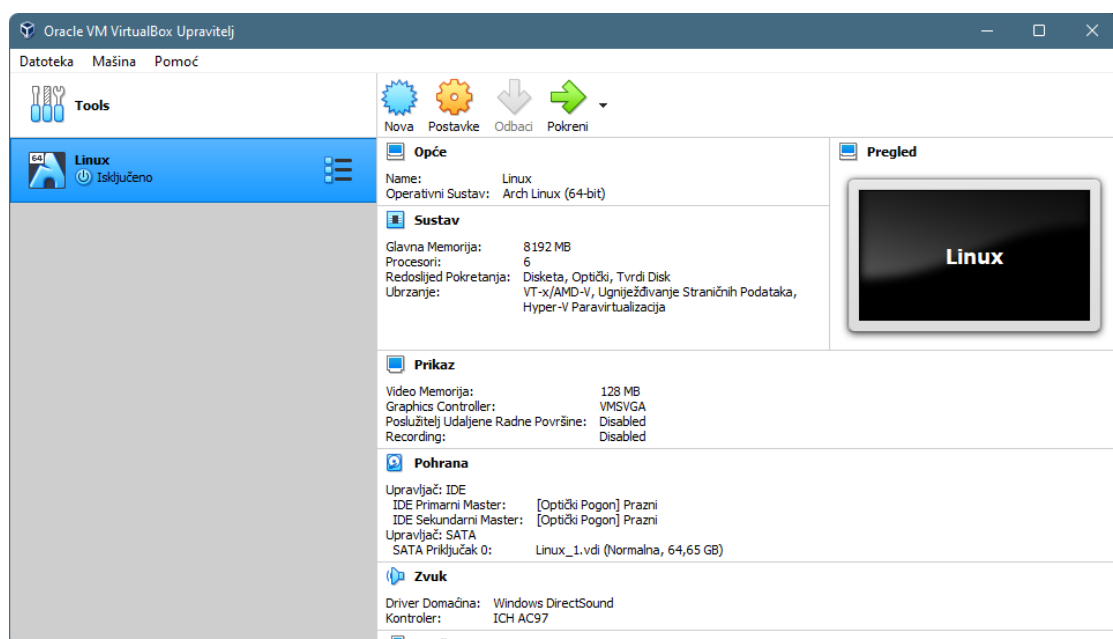
- Jezgra izvodi stvaranje, izvršavanje i završetak procesa koji se izvode u sustavu. Kad sustav mora izvršiti koji bilo zadatak, jezgra stvara procese i njima upravlja.

## 3. Jezgreni moduli

Moduli su dijelovi koda koji se mogu učitavati u jezgru i isključivati iz nje na zahtjev. Proširuju funkcionalnost jezgre bez potrebe za ponovnim pokretanjem sustava. Na primjer, jedna vrsta modula je upravljački program uređaja, koji omogućuje jezgri pristup sklopovlju spojenom na sustav. Bez modula morale bi se izgraditi monolitne jezgre i dodati nova funkcionalnost izravno u sliku jezgre. Osim što stvara veće jezgre, takav pristup ima i nedostatak jer zahtijeva da se ponovno izgradi i ponovno pokrene jezgra svaki put kad se želi nova funkcionalnost. Moduli su pohranjeni pod `/lib/modules/` i ovo kazalo sadržava svaku instaliranu jezgru. Same datoteke modula završavaju s „.ko“ što je kratica za „Kernel Object“.

### 3.1. Izgradnja jezgrenih modula i potrebni programi

Za potrebe ovoga završnoga rada potrebno je bilo virtualno računalo na koje bi se instalirao operacijski sustav Linux. Odlučilo se je na program Oracle VM VirtualBox za izradbu, pokretanje i održavanje virtualnih računala zbog prethodnoga znanja u tom programu. Za operacijski sustav Linux odlučilo se je za ArchLinux, točnije ArcoLinux-distribuciju ArchLinuxa zbog male potrošnje sustavskih sredstava, ali i isto tako zbog prethodnoga znanja i iskustva s ArcoLinux-distribucijom. Unutar VirtualBoxa se je stvorilo novo virtualno računalo kojemu se je dalo šest jezgara procesora i osam gigabajta radne memorije.



Slika 2. Program Oracle VM VirtualBox

```

Alacritty
-----
OS: ArcoLinux
Kernel: 5.18.15-arch1-2
Uptime: 1 min
Packages: 1271 (pacman)
Shell: bash 5.1.16
Resolution: 1920x1009
DE: Xfce 4.16
WM: Xfwm4
WM Theme: Arc-Dark
Theme: Arc-Dark [GTK2/3]
Icons: Sardi-Arc [GTK2/3]
Terminal: alacritty
CPU: AMD Ryzen 5 2600 (6) @ 0MHz
GPU: 00:02.0 VMware SVGA II Adapter
Memory: 589MiB / 7949MiB (7%)

  /-
  ooo:
  yoooo/
  yoooooooo
  yooooooooo
  yooooooooo
  .yoooooooooooo
  .oooooooooooooooo
  .ooooooooarooooooooo
  .ooooooooo-ooooooooo
  .ooooooooo- oooooooooo
  :ooooooooo. :ooooooooo
  :ooooooooo. :ooooooooo
  :ooaroooo .ooaroooo
  :oooooooooy .ooooooooo
  :ooooooooo /oooooooooooooooooooo
  :ooooooooo .-ooooooooooooooooo.
  oooooooooo- -ooooooooooooooooo.
  oooooooooo- .-ooooooooooooooooo.
  oooooooooo. -ooooooooooooo

[dm@dominik-virtualbox ~]$ uname -r
5.18.15-arch1-2
[dm@dominik-virtualbox ~]$

```

Slika 3. ArcoLinux s inačicom 5.18.15-arch1-2 Linuxove jezgre

ArcoLinux-distribucija dolazi s Linuxovom jezgrom 5.18.15-arch1-2 pa se je odlučilo za potrebe ovoga rada raditi na inačici 5.18.16 Linuxove jezgre. Zbog lakšega verzioniranja koda rabio se je GitHubov repozitorij kako bi se lakše pratile promjene u kodu i kako bi se imalo sigurno mjesto za sve potrebne datoteke koje su se rabile prigodom rada. Sav programski kod pisan je u IDE-u VS Codium dok je GitKraken rabljen kao GitHubov klijent. Kako bis se izgradio kakav bilo jezgreni modul potrebno je određeno znanje o sustavima za izgradnju koji se rabe prigodom rada s Linuxovom jezgrom.

Kbuild je sustav za izgradnju koji rabi Linuxova jezgra. Moduli moraju rabiti Kbuild kako bi ostali kompatibilni s promjenama u infrastrukturi izgradnje. Metoda za izgradnju unutar stabla i izvan stabla je slična, a svi su moduli početno razvijeni i izgrađeni izvan stabla, kako bi se moduli ispitali. Isto tako *makefile*-datoteke se razlikuju ovisno o tom gradi li se modul unutar stabla ili izvan stabla. Što se tiče datoteka *makefile* unutar izvornoga koda Linuxove jezgre, postoji pet različitih vrsta datoteka *makefile*.

*linux-5.18.16/Makefile* je glavni *makefile* koji pokreće ostale skripte *makefile*, u ostatku teksta će se pozivati na njega kao „gornji *makefile*“.

*linux-5.18.16/.config* je dokument koji sadržava konfiguraciju jezgre.

*linux-5.18.16/arch/\$(ARCH)/Makefile* je *makefile* koji opskrbljuje informacijama specifičnima za arhitekturu.

*linux-5.18.16/scripts/Makefile.\** sadržava standardna pravila za ostale *Kbuildove makefilee*.

*Kbuild makefiles* su sve ostale datoteke *makefile* koje su u kazalu svakoga modula.

Gornji *makefile* čita datoteku „*config*“ koja dolazi iz procesa konfiguracije jezgre. Gornji *makefile* je odgovoran za izgradnju dvaju glavnih proizvoda: *vmlinuxa* (slike jezgre) i modula (koje bilo datoteke modula). Gradi ih rekurzivnim spuštanjem u potkazala jezgrenoga stabla izvornoga koda.

Popis potkazala koja se posjećuju ovisi o konfiguraciji jezgre. Gornji *makefile* tekstualno uključuje *arch makefile* s nazivom „*arch/\$(ARCH)/Makefile*“. *arch makefile* opskrbljuje informacijama specifičnima za arhitekturu vrh *makefilea*. Svako potkazalo ima *Kbuild makefile* koji izvršava naredbe prenesene odozgor. *Kbuild makefile* rabi informacije iz datoteka „*config*“ za izradbu raznih popisa datoteka koje rabi *Kbuild* za izgradnju svih ugrađenih ili modularnih meta. *scripts/Makefile.\** sadržava sve definicije/pravila itd. koje se rabe za izgradnju jezgre temeljene na datotekama *Kbuild makefiles*. Vrlo je važno spomenuti da se datoteke *makefile* pišu s pomoću tipke Tab a ne razmaka, ako se rabi razmak javit će se pogreške prigodom izvršavanja naredbe *make*.

Najjednostavniji *Kbuild makefile* sadržava jedan redak, npr.:

```
obj-y += foo.o
```

Ovo govori *Kbuildu* da postoji jedan objekt u tom kazalu pod nazivom *foo.o*. Datoteka *foo.o* bit će izgrađena od datoteke *foo.c* ili *foo.S*. Ako se *foo.o* gradi kao modul, rabi se varijabla „*obj-m*“. Stoga se često rabi sljedeći obrazac, npr.:

```
obj-$(CONFIG_FOO) += foo.o
```

Vrijednost varijable „*CONFIG\_FOO*“ je zapisana u konfiguracijskoj datoteci jezgre „*config*“. Vrijednost varijable „*\$(CONFIG\_FOO)*“ daje „*y*“ (za ugrađeni) ili „*m*“ (za modul). Ako „*CONFIG\_FOO*“ nije ni „*y*“ ni „*m*“, tad datoteka ne će biti kompilirana ni povezana. Razlika između naredaba „*obj-y*“ i „*obj-m*“ jest u tom da ako je „*y*“ tad će modul biti ugrađen u jezgru dokle ako se stavi „*m*“ modul će biti preveden kao modul koji će se pozivati po potrebi operacijskoga sustava. U početku će se rabiti „*obj-m*“ zbog lokalnoga ispitivanja, jer ako se dogodi neka pogreška ili ispis modula nije ispravan, modul se može jednostavno isključiti naredbom „*modprobe -r <naziv\_modula>*“, dok kod „*obj-y*“ to nije moguće.

Za potrebe ovoga rada trebali su se moduli izgraditi izvan i unutar stabla, za početak izvan stabla kako bi se moduli mogli ispravno testirati a kad se je bilo sigurno u njihovu ispravnost bilo ih je potrebno izgraditi unutar stabla. Kako bi se izbjegnule moguće pogreške u kodu, koje bi se uočile tek kad se jezgra prevede, odlučeno je prvo ispitati kod tako da se ručno ubaci modul u jezgru. Za to je potrebno napraviti *makefile* koji izgleda ovako.

```
Makefile
1  KERNELDIR = /lib/modules/`uname -r`/build
2  MODULES = helloworld.ko
3  obj-m += helloworld.o
4
5  all:
6      make -C $(KERNELDIR) M=$(PWD) modules
7  clean:
8      make -C $(KERNELDIR) M=$(PWD) clean
9  install:
10     make -C $(KERNELDIR) M=$(PWD) modules_install
11  quickinstall:
12     cp $(MODULES) /lib/modules/`uname -r`/extra
```

Slika 4. Sadržaj *makefile*-datoteke jednostavnoga Helloworld-modula

Postoje dvije varijable koje pohranjuju vrijednosti, varijabla „*KERNELDIR*“ pohranjuje vrijednost staze do kazala u kojem se nalaze moduli trenutno djelatne jezgre. Varijabla „*MODULES*“ pohranjuje naziv izrađenoga modula. Redak „*obj-m += helloworld.o*“ obilježava da će se modul prevesti kao modul a ne kao dio jezgre. Kako bi se pokrenuo *makefile* da se izradi modul potrebno je otvoriti terminal i navigirati u kazalo u kojem se nalaze datoteke *.c* i *makefile* i pozvati naredbu „make“.

## 4. Izrada jednostavnih jezgrenih modula

Kao dobru podlogu prije nego se započne izradba nekoga velikoga i kompliciranoga modula/programa u novom okruženju dobro je ispitati ponašanje okruženja s nečim jednostavnim. U ovom radu to je jednostavni modul „Hello world“.

### 4.1. „Hello world“-modul i modul „Aktivni procesi“

Modul „Hello world“ ispisuje „*Hello world*“ kad je modul učitani u jezgru preko naredbe „*sudo insmod ./helloworld.ko*“. Kad se modul makne iz jezgre naredbom „*sudo rmmod ./helloworld.ko*“ ispisuje se poruka „*Goodbye world*“. Poruke su vidljive u terminalu pozivom naredbe „*sudo dmesg*“. Provjera je li modul učitani u jezgru čini se preko naredbe „*lsmod | grep helloworld*“.

```
C helloworld.c
1  #include <linux/init.h>
2  #include <linux/module.h>
3  MODULE_LICENSE("Dual BSD/GPL");
4
5  static int hello_init(void){
6      printk(KERN_ALERT "Hello world\n");
7      return 0;
8  }
9  static void hello_exit(void){
10     printk(KERN_ALERT "Goodbye world\n");
11 }
12
13 module_init(hello_init);
14 module_exit(hello_exit);
```

Slika 5. Sadržaj *helloworld.c*-datoteke jednostavnoga Hello world-modula

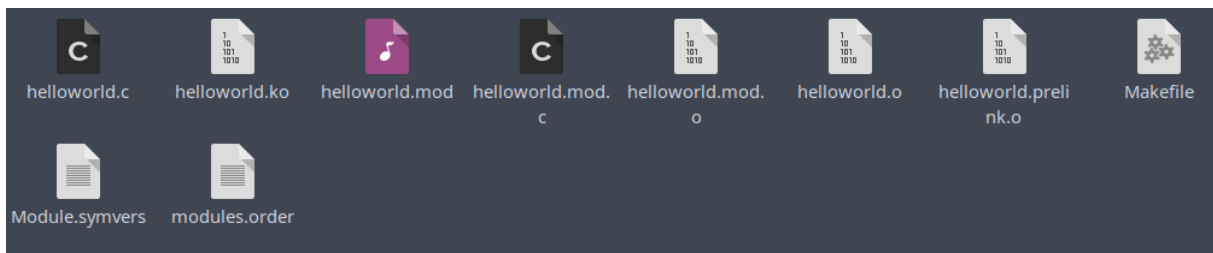
Ovako izgleda ovaj jednostavni kod. Prvo se trebaju uključiti dva Linuxova zaglavlja koja se rabe u razvoju Linuxova modula. Zatim je potrebno specificirati po kojoj licenci se radi ovaj modul. Dolazi se do prve funkcije koja je potpisana „*static int*“ i naziva „*hello\_init*“ koja ne prima argumente, nego samo ispisuje „*Hello world*“ preko funkcije „*printk*“ koja ispisuje poruke u jezgri zapisnik. Specificirana je zapisnička razina „*KERN\_ALERT*“ koja se inače rabi za probleme koji zahtijevaju pozornost, ali se ovdje rabi tako da ispisana poruka bude



istaknuta crvenom bojom kako bi se lakše vidjela. Vraća se ničtica kako bi proces koji pokreće module znao da je sve u redu.

Zatim dolazi se do funkcije „*static void*“ naziva „*hello\_exit*“ koja isto tako ne prima argumente nego samo preko funkcije „*printf*“ ispisuje poruku „*Goodbye world\n*“. Potrebno je rabiti „*\n*“ na kraju teksta koji se ispisuje kako bi se popunio ispisni međuspremnik, jer se u protivnom ništa ne će ispisati. Na kraju se nalaze dvije funkcije koje govore koja funkcija će se pokrenuti kad se modul učita u jezgru („*module\_init*“) i („*module\_exit*“) koja funkcija će se pokrenuti kad makne se modul iz jezgre.

*makefile*-datoteka je prikazana na Sl. 4. i ondje objašnjena. Kako bi se izgradio ovaj modul izvan stabla potrebno je pokrenuti naredbu *make* nakon koje nastaje kazalo koje izgleda ovako.



Slika 6. Kazalo HelloWorld-modula nakon izgradnje

Naredba koja služi da se modul učita u jezgru je „*sudo insmod ./helloworld.ko*“. Naredba koja služi da se modul istovari iz jezgre je „*sudo rmmmod ./helloworld.ko*“. Naredba da se ispišu svi trenutačno učitani moduli je „*lsmod*“. Naredba da se provjeri je li modul učitani jest „*lsmod | grep helloworld*“, ako je modul pokrenut u jezgri njegov naziv će se pokazati kad se pokrene ta naredba, u protivnom ispis će biti prazan.

```
Terminal - dm@dominik-virtualbox:~/Documents/zavrsni_rad
[dm@dominik-virtualbox zavrsni_rad]$ ls
helloworld.c  helloworld.mod  helloworld.mod.o  helloworld.prelink.o  modules.order
helloworld.ko  helloworld.mod.c  helloworld.o  Makefile  Module.symvers
[dm@dominik-virtualbox zavrsni_rad]$ lsmod | grep "helloworld"
[dm@dominik-virtualbox zavrsni_rad]$ sudo insmod ./helloworld.ko
[sudo] password for dm:
[dm@dominik-virtualbox zavrsni_rad]$ lsmod | grep "helloworld"
helloworld      16384  0
[dm@dominik-virtualbox zavrsni_rad]$
```

Slika 7. Rezultat pokretanja naredba *ls*, *lsmod | grep helloworld* i *sudo insmod ./helloworld.ko*

„*sudo dmesg*“ može se porabiti kako bi se pregledao ispis modula. Tekst je istaknut crvenom bojom zbog atributa „*KERN\_ALERT*“ unutar funkcije „*printk*“ u modulu.

```
[11007.078369] Hello world
[11011.541534] Goodbye world
```

Slika 8. Ispis međuspremnika poruka jezgre nakon učitavanja i isključivanja Helloworld-modula

Složenija inačica modula „Helloworld“ jest modul koji ispisuje početnu poruku, sve djelatne procese s njihovim ID-om, te, kad se ispišu svi djelatni procesi, ispisat će se broj djelatnih procesa.

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched/signal.h>

static int procesi_init(void){
    struct task_struct *task_list;
    unsigned int broj_procesa = 0;
    printk(KERN_ALERT "%s: Pozdrav!\n Lista trenutno aktivnih procesa:\n", __func__);
    for_each_process(task_list) {
        pr_info("Ime procesa: %s\t PID:[%d]\t\n",
                task_list->comm, task_list->pid);
        broj_procesa++;
    }
    printk(KERN_ALERT "Broj aktivnih procesa:%u\n", broj_procesa);
    return 0;
}

static void procesi_exit(void){
    printk(KERN_ALERT "%s: Gašenje modula!\n", __func__);
}

MODULE_LICENSE("GPL");
module_init(procesi_init);
module_exit(procesi_exit);
```

Slika 9. Sadržaj *procesi\_aktivni.c*-datoteke jednostavnoga modula „Aktivni procesi“

Uz već spomenuta zaglavlja „*kernel.h*“ i „*module.h*“ potrebno je uključiti i „*sched/signal.h*“. Potreban je zbog strukture „*task\_struct*“ i zbog funkcije „*for\_each\_process*“. Linuxova jezgra, kao i svaki drugi sustav, ima puno procesa koji se izvode u kojem bilo trenutku. Struktura „*task\_struct*“ definirana u „*sched/signal.h*“ pohranjuje sve potankosti svakoga procesa koji postoji u sustavu, a svi procesi su pohranjeni na kružnom dvostruko povezanom popisu.

Unutar funkcije „*procesi\_init*“ koja ne prima argumente deklarira se struktura „*task\_struct*“ i varijabla „*unsigned int broj\_procesa*“. Ispisuje se poruka koja pozdravlja korisnika, zatim se ulazi u petlju „*for\_each\_process*“ koja se izvršava za svaki proces koji je trenutačno djelatna. Kako bi se ispisao naziv procesa potrebno je ispisati vrijednost u članu „*task\_list->comm*“, a ID procesa se nalazi u „*task\_list->pid*“. Isto tako uz ispisivanje povećava se brojač „*broj\_procesa*“, pošto se petlja izvrti ispisuje se ukupni broj djelatnih procesa. U funkciji za izlaz se nalazi poruka koja glasi „*Gašenje modula!\n*“. *Makefile* ovoga modula prigodom ispitivanja izgleda jednako kao i *makefile* modula „*HelloWorld*“ samo s promijenjenim nazivima izlaznoga modula. Potrebno je izmijeniti dva retka datoteke *makefile*.

```
MODULES = procesi_aktivni.ko
```

```
obj-m += procesi_aktivni.o
```

```
1  KERNELDIR = /lib/modules/`uname -r`/build
2  MODULES = procesi_aktivni.ko
3  obj-m += procesi_aktivni.o
4
5  all:
6      make -C $(KERNELDIR) M=$(PWD) modules
7  clean:
8      make -C $(KERNELDIR) M=$(PWD) clean
9  install:
10     make -C $(KERNELDIR) M=$(PWD) modules_install
11 quickinstall:
12     cp $(MODULES) /lib/modules/`uname -r`/extra
```

Slika 10. Sadržaj *makefile*-datoteke modula „Aktivni procesi“

Učitavanjem ovoga modula u jezgru naredbom *insmod procesi\_aktivni.ko* dobiva se sljedeći ispis.

```
[ 0.509434] procesi_init: Pozdrav!  
Lista trenutno aktivnih procesa:  
[ 0.509437] Ime procesa: swapper/0 PID:[1]  
[ 0.509438] Ime procesa: kthreadd PID:[2]  
[ 0.509438] Ime procesa: rcu_gp PID:[3]  
[ 0.509439] Ime procesa: rcu_par_gp PID:[4]  
[ 0.509440] Ime procesa: netns PID:[5]  
[ 0.509440] Ime procesa: kworker/0:0 PID:[6]  
[ 0.509441] Ime procesa: kworker/0:0H PID:[7]  
[ 0.509442] Ime procesa: kworker/u12:0 PID:[8]  
[ 0.509442] Ime procesa: kworker/0:1H PID:[9]  
[ 0.509443] Ime procesa: mm_percpu_wq PID:[10]  
[ 0.509444] Ime procesa: kworker/u12:1 PID:[11]  
[ 0.509444] Ime procesa: rcu_tasks_kthre PID:[12]  
[ 0.509445] Ime procesa: rcu_tasks_rude_ PID:[13]  
[ 0.509446] Ime procesa: rcu_tasks_trace PID:[14]  
[ 0.509446] Ime procesa: ksoftirqd/0 PID:[15]
```

Slika 11. Ispis međuspremnika poruka jezgre nakon učitavanja modula „Aktivni procesi“, 1/2

```
[ 0.509497] Ime procesa: blkcg_punt_bio PID:[62]  
[ 0.509498] Ime procesa: ata_sff PID:[63]  
[ 0.509498] Ime procesa: edac-poller PID:[64]  
[ 0.509499] Ime procesa: devfreq_wq PID:[65]  
[ 0.509499] Ime procesa: watchdogd PID:[66]  
[ 0.509500] Ime procesa: kworker/1:1 PID:[67]  
[ 0.509501] Ime procesa: kswapd0 PID:[68]  
[ 0.509501] Ime procesa: kworker/u12:2 PID:[69]  
[ 0.509502] Ime procesa: kworker/u12:3 PID:[71]  
[ 0.509503] Ime procesa: kthrotld PID:[76]  
[ 0.509504] Ime procesa: kworker/1:2 PID:[79]  
[ 0.509504] Ime procesa: kworker/2:1 PID:[81]  
[ 0.509505] Ime procesa: acpi_thermal_pm PID:[82]  
[ 0.509505] Ime procesa: scsi_eh_0 PID:[83]  
[ 0.509506] Ime procesa: scsi_tmf_0 PID:[84]  
[ 0.509506] Ime procesa: scsi_eh_1 PID:[85]  
[ 0.509507] Ime procesa: scsi_tmf_1 PID:[86]  
[ 0.509507] Broj aktivnih procesa:78
```

Slika 12. Ispis međuspremnika poruka jezgre nakon učitavanja modula „Aktivni procesi“, 2/2

Modul je isto tako moguće pokrenuti na podiznom procesu a da on nije ugrađen u jezgru. Potrebna je mapa u kojoj je kompilirani modul, kako se u ovom primjeru rabi modul „Helloworld“ to značenjuje da postoje datoteke *helloworld.ko* i *helloworld.o*. Zatim je tu mapu potrebno smjestiti u kazalo „*/lib/modules/<inačica djelatne jezgre>/build/drivers*“ tako da postane vidljiva Linuxovu programu „*modprobe*“. Po tom je potrebno pokrenuti naredbu „*depmod*“.

Naredba „*depmod*“ (Dependency Modules) rabi se za generiranje popisa opisa ovisnosti modula jezgre i pridruženih datoteka mape. Analizira module jezgre u kazalu „*/lib/modules/<inačicajezgre>*“ i stvara datoteku ovisnosti nalik na „*makefile*“ pod nazivom „*modules.dep*“ na temelju simbola nazočnih u skupu modula. Ovi se moduli općenito uzimaju

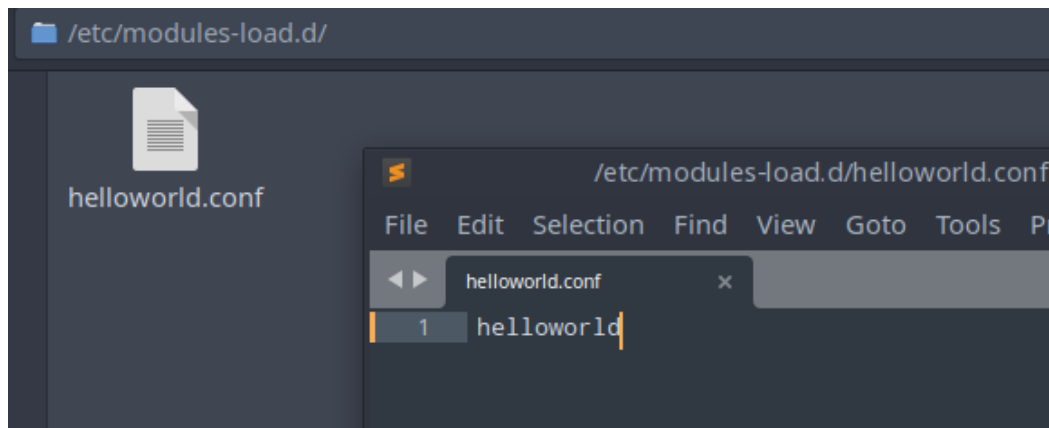
iz kazala navedenih u konfiguracijskoj datoteci ili navedenih u naredbenom retku. U slučaju kad se hrpa modula doda i automatski ukloni naredbom „*modprobe*“ nijedan modul ne ostaje bez drugih povezanih modula koji su im potrebni. Istodobno, stvara pridruženu mapu koja povezuje sklopovske identifikatore i odgovarajuće module koji njima rukuju u svrhu porabe od infrastrukture „*hotplug*“. Ovo specifično povezano preslikavanje rabi se za traženje i pronalaženje ispravnoga modula kad jedinica sklopovlja to zatraži.

Po završetku izvođenja naredbe „*depmod*“ može se ispitati radi li sve ispravno tako da se u terminalu pokrene naredba „*modprobe*“ „*modprobe hellow*“ i pritisne Tab kako bi se naredba samozavršila, ako se naredba automatski završi tad se može biti siguran da „*modprobe*“ vidi modul i može se nastaviti s pisanjem konfiguracijske datoteke. Za automatsko učitavanje modula pri pokretanju sustava može se rabiti usluga „*systemd-modules-load.service*“. To zahtijeva da naziv modula bude naveden u datoteci koja odgovara jednomu od sljedećih uzoraka:

```
/etc/modules-load.d/*.conf
```

```
/run/modules-load.d/*.conf
```

```
/usr/lib/modules-load.d/*.conf
```



Slika 13. Kazalo */etc/modules-load.d/* i sadržaj datoteke *helloworld.conf*

Uzorak „*/etc/modules-load.d/\*.conf*“ je najprikladniji za lokalnu administraciju sustava pa kako na virtualnom računalu postoje administratorske ovlasti to će se i rabiti. Prvo se mora otvoriti kazalo kao administrator jer u protivnom nije moguće izraditi nove datoteke unutar toga kazala. Unutar kazala „*/etc/modules-load.d/*“ izradi se nova datoteka s nazivom „*helloworld.conf*“ i sa sadržajem naziva modula koji se želi automatski učitati na podizanju, u ovom slučaju to je modul „Helloworld“ tako da je potrebno u datoteku napisati „*helloworld*“.

# 5. Prevođenje, instalacija, pokretanje, ispitivanje i dokumentacija preinačene jezgre operacijskoga sustava

## 5.1. Prevođenje jezgre

Postoji više različitih načina kompiliranja jezgre, prvi koji se je iskušao jest preko PKGBUILD-a jer je taj proces preporučen od stranice „ArchLinux Wiki“. Nažalost taj proces se ne može rabiti za ovaj rad zbog nemogućnosti dodavanja vanjskih modula u jezgru. Ali kako je ovo službeni način izradbe jezgre odlučeno ga je iskušati.

### 5.1.1. Prevođenje preko PKGBUILD-a

*makepkg* se ne može pokrenuti kao *root/sudo*, stoga je potrebno prvo stvoriti kazalo za izgradnju.

```
$ mkdir ~/build/
```

```
$ cd ~/build/
```

Zatim je potrebno instalirati paket „*asp*“ i skupinu paketâ „*base-devel*“. Potrebna je čista jezgra za početak prilagodbe, nju se može dohvatiti preko sljedećih naredaba.

```
$ asp update linux
```

```
$ asp export linux
```

U ovom trenutku stablo kazala bi trebalo izgledati ovako:

```
~/build/linux/+-
```

```
  +--konfig
```

```
  \_PKGBUILD
```

Uredi se PKGBUILD i potraži se parametar „*pkgbase*“. Promijeni se ovo u svoj prilagođeni naziv paketa, npr.:

```
PKGBUILD
```

```
pkgbase=linux-dm
```

Zatim je moguće preko naredbe „*makepkg -s*“ pokrenuti izradbu jezgre, parametar „-s“ će skinuti i instalirati kakve bilo potrebne skripte ili programe.

Nakon izradbe nastat će dva paketa u mapi „~/build/linux“, jedan za jezgru i jedan za zaglavlja jezgre. Mogu imati nazive poput:

```
linux-custom-5.18.16-x86_64.pkg.tar.zst
```

```
linux-custom-headers-5.18.16-x86_64.pkg.tar.zst
```

Najbolja praksa je instalirati oba paketa zajedno jer bi oba mogla biti potrebna:

```
pacman -U linux-custom-headers-5.18.16-x86_64.pkg.tar.zst linux-custom-5.18.16-x86_64.pkg.tar.zst
```

Nakon toga koraka potrebno je izmijeniti *bootloader* kako bi on ispravno vidio novu jezgru i kako bi ju postavio kao primarnu.

### 5.1.2. Prevođenje Linuxove jezgre iz izvornoga koda

Sad se dolazi do načina kompliranja jezgre koji se je rabio kako bi se postignuo cilj ovoga rada. Potrebni su bili sljedeći paketi: *base-devel*, *xmlto*, *kmod*, *inetutils*, *bc*, *libelf*, *git*, *cpio*. Njih je moguće instalirati preko sljedeće naredbe:

```
sudo pacman -S base-devel xmlto kmod inetutils bc libelf git cpio
```

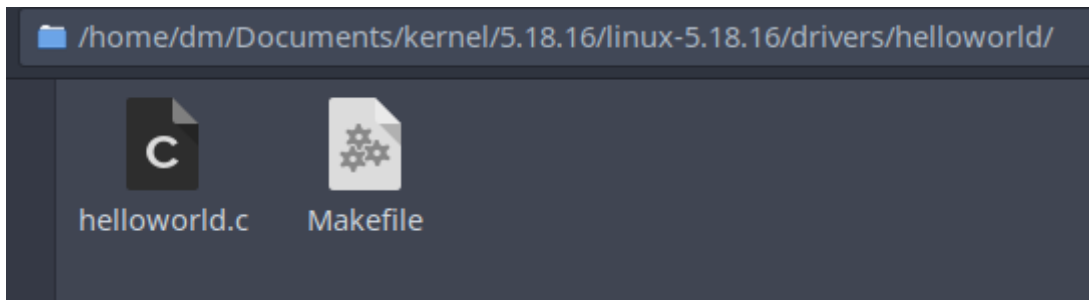
Jezgra se je preuzela iz web-mjesta [www.kernel.org](http://www.kernel.org). Jezgreni izvorni koda je u datoteki *.tar* pa je potrebno izvršiti sljedeću naredbu kako bi se ekstrahirao kod:

```
tar -xvJf linux-5.18.16.tar.xz
```

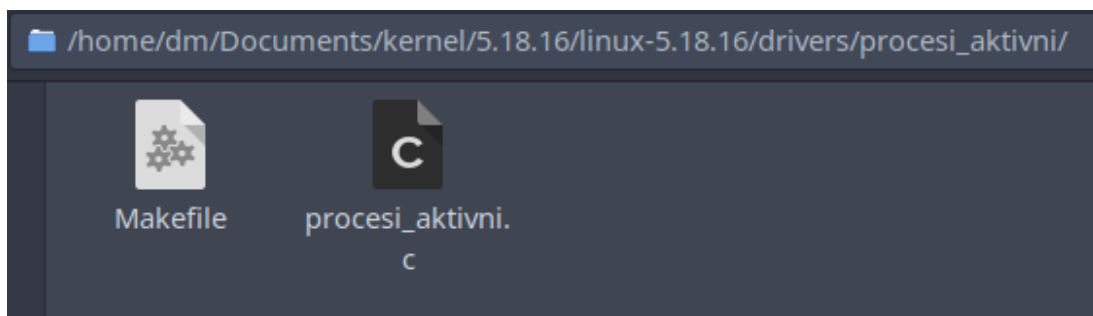
Nakon ekstrahiranja uđe se u izrađenu mapu jezgre i izvrši se naredba „*make mrproper*“. Ta naredba briše sve nepotrebne stvari iz kazala i pripravlja datoteke za kompiliranje. Zatim je potrebno izraditi ili prebaciti dvije mape unutar kazala */drivers/* u koje će se smjestiti moduli. Kako bi se moduli ugradili u jezgru potrebno je napisati *makefile* koji izgleda ovako:

```
obj-y += helloworld.o u mapi „/drivers/helloworld“ za modul „Helloworld“ i
```

```
obj-y += procesi_aktivni.o u mapi „/drivers/procesi_aktivni“ za modul koji ispisuje djelatne procese
```



Slika 14. Struktura kazala /linux-5.18.16/drivers/helloworld



Slika 15. Struktura kazala /linux-5.18.16/drivers/procesi\_aktivni

Datoteke C-koda su ostale nepromijenjene. Isto tako kako bi kompilator vidio te datoteke *makefile* potrebno je u mapi „/drivers/“ dodati retke u *makefile* koji glase:

```
obj-y          += helloworld/
obj-y          += procesi_aktivni/
```

```
obj-y          += auxdisplay/
obj-y          += helloworld/
obj-y          += procesi_aktivni/
obj-$(CONFIG_PCCARD) += pcmcia/
```

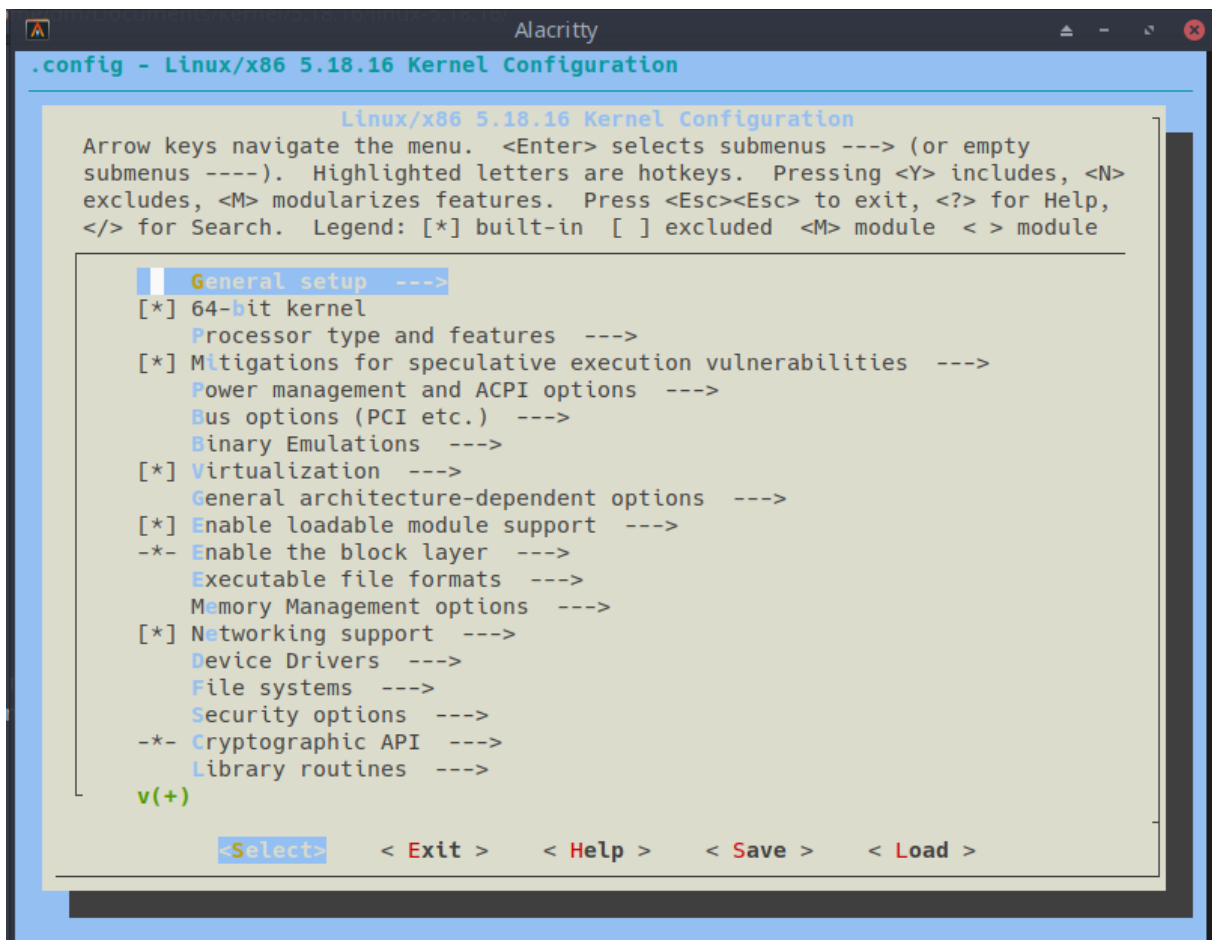
Slika 16. Sadržaj datoteke /linux-5.18.16/drivers/makefile

Zatim treba konfiguracijska datoteka jezgre, postoji nekoliko načina kako se ta datoteka „*.config*“ stvori. Jedan od mogućih načina jest poraba već postojeće datoteke „*.config.gz*“ unutar datoteke „/proc/*config.gz*“. Konfiguracijska datoteka postojeće jezgre može se prebaciti u kazalo preko sljedeće naredbe:

```
zcat /proc/config.gz > .config
```

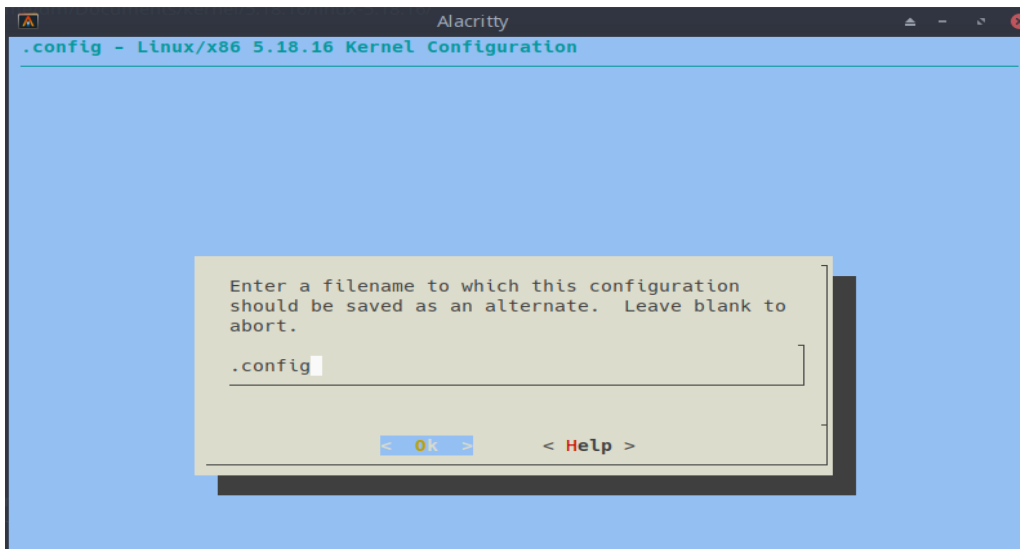


Preko tekstualnoga uređivača postoji mogućnost ručnog mijenjanja konfiguracijske datoteke. Ovo se preporučuje samo ako se dobro zna što se čini jer jedna pogreška ili brisanje jednoga retka može dovesti do pogreške u radu prigodom prevođenja jezgre. Izvršavanjem naredbe „*make olddefconfig*“ može se provjeriti jesu li sve vrijednosti u konfiguracijskoj datoteci postavljene, ako nisu program će zatražiti unos. Drugi način konfiguriranja datoteke „*.config*“ se čini preko grafičkoga sučelja koje se poziva naredbom *make menuconfig*



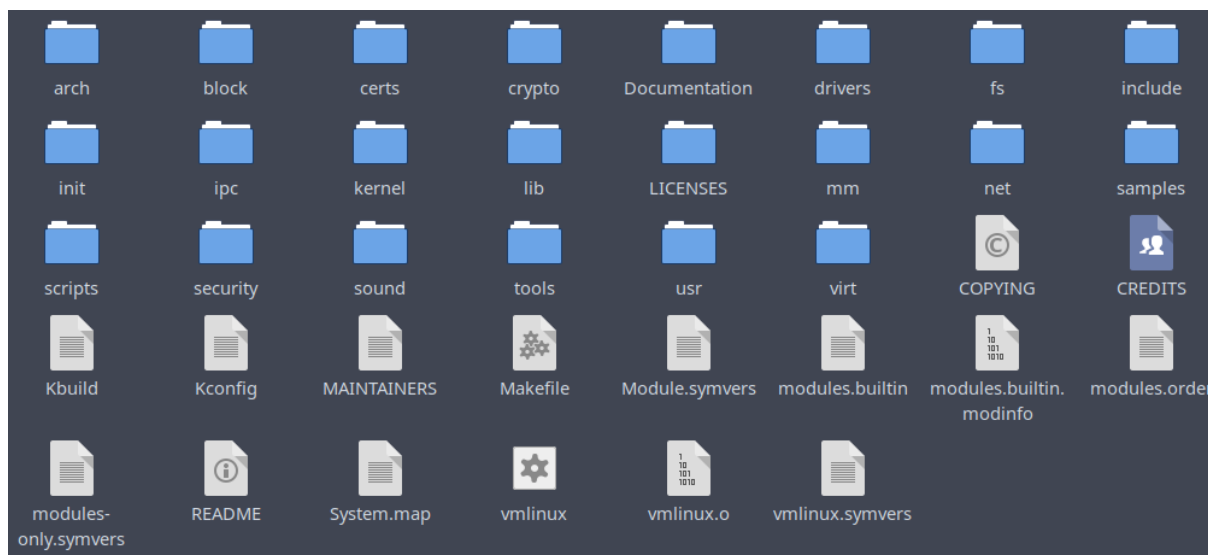
Slika 17. *menuconfig* – grafički konfigurator Linuxove jezgre

Ovdje se može konfigurirati jezgra po potrebi, ali kako je samo potrebno da se moduli ugrade u jezgru a to se je postignulo tako da se je unutar datoteka „*makefile*“ u kazalima modula napisalo „*obj-y*“ umjesto „*obj-m*“, nije potrebno činiti dalju konfiguraciju jezgre unutar ovoga izbornika. Tako da se odabire „*Save*“ i ostavlja se naziv kao „*.config*“.



Slika 18. Pohrana konfiguracijske datoteke „.config“

Nakon što je konfiguracija jezgre pripravna može se započeti izradba jezgre naredbom „make“. Ovisno o broju jezgara procesora i brzini diska taj proces može trajati sve od pola sata do nekoliko sati. Nakon uspješnoga sastavljanja jezgre struktura kazala bi trebala izgledati ovako.



Slika 19. Struktura kazala nakon prevođenja jezgre

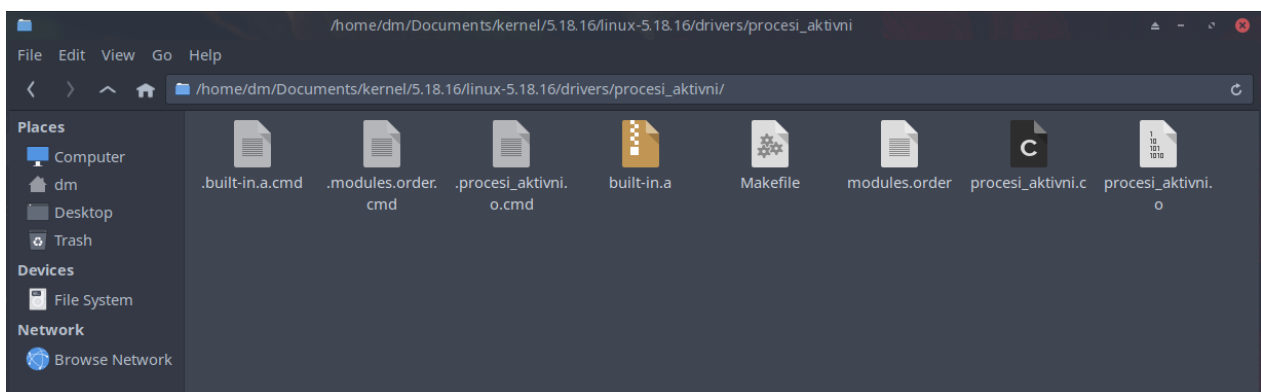
*moduls.order*-datoteka bilježi redoslijed kojim se moduli pojavljuju u procesu Makefile. Rabi „*modprobe*“ za deterministično rješavanje *aliasa* kad se podudara više modula. *modules.builtin*-datoteka navodi sve module koji su ugrađeni u jezgru. Ovo rabi „*modprobe*“

da ne pogriješi kad pokušava učitati ugrađeni modul. Unutar ove datoteke može se provjeriti jesu li moduli ugrađeni u jezgru, ako jesu oni će se pojaviti na tom popisu. Pošto se moduli nalaze na tom popisu može se biti siguran da su oni ugrađeni u Linuxovu jezgru.

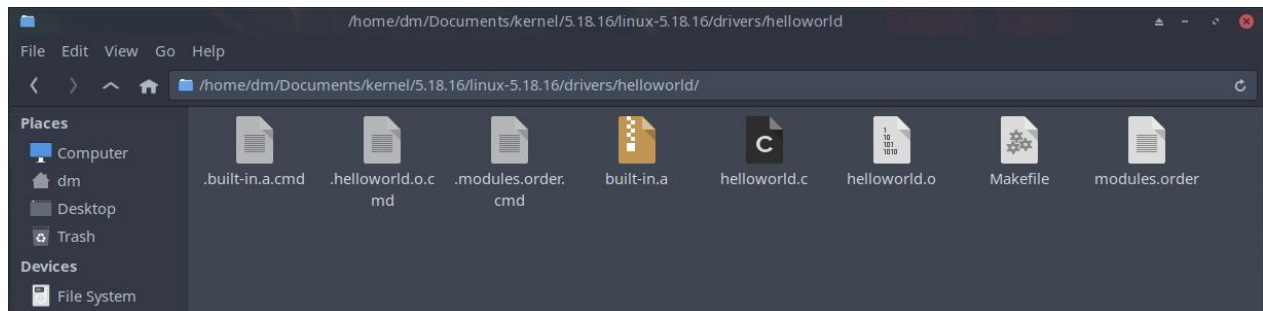
```
modules.builtin
181 kernel/drivers/mfd/lp8788.ko
182 kernel/drivers/mfd/da9055.ko
183 kernel/drivers/mfd/tps6586x.ko
184 kernel/drivers/mfd/palmas.ko
185 kernel/drivers/mfd/intel-soc-pmic.ko
186 kernel/drivers/nvdimmm/libnvdimmm.ko
187 kernel/drivers/dax/dax.ko
188 kernel/drivers/dma-buf/heaps/system_heap.ko
189 kernel/drivers/dma-buf/heaps/cma_heap.ko
190 kernel/drivers/dma-buf/udmabuf.ko
191 kernel/drivers/scsi/scsi_mod.ko
192 kernel/drivers/scsi/scsi_common.ko
193 kernel/drivers/scsi/sd_mod.ko
194 kernel/drivers/ata/libata.ko
195 kernel/drivers/ata/ahci.ko
196 kernel/drivers/ata/libahci.ko
197 kernel/drivers/net/wwan/wwan.ko
198 kernel/drivers/helloworld/helloworld.ko
199 kernel/drivers/procesi_aktivni/procesi_aktivni.ko
200 kernel/drivers/usb/common/usb-common.ko
201 kernel/drivers/usb/core/usbcore.ko
```

Slika 20. Sadržaj datoteke *modules.builtin*

*moduli.builtin.modinfo*-datoteka sadržava „*modinfo*“ iz svih modula koji su ugrađeni u jezgru. Za razliku od zasebnoga modula „*modinfo*“ sva polja imaju predmetak s nazivom modula. Kazala modula trebaju izgledati ovako ako se je odabralo da moduli budu ugrađeni u jezgru.



Slika 21. Struktura kazala „*procesi\_aktivni*“ nakon prevođenja jezgre



Slika 22. Struktura kazala „*helloworld*“ nakon prevođenja jezgre

## 5.2. Instalacija, pokretanje, ispitivanje i dokumentacija modulom preinačene jezgre operacijskoga sustava

Za sljedeće korake je potrebno da ih izvrši administratorski korisnik, kako je potrebno kopirati datoteke u kazalo */boot/* koje je zaštićeno. Prebacivanje u administratorskoga korisnika se čini preko naredbe „*su*“, terminal traži da se unese zaporka administratorskoga računara. Zatim je potrebno instalirati sve module, naredba „*make modules\_install*“ će se pobrinuti da postoje izrađene binarne datoteke i instalirat će te binarne datoteke u kazalo *modules* jezgre, u ovom slučaju to kazalo glasi */usr/lib/modules/5.18.16/*. Stvorenu podiznu sliku je potrebno premjestiti na specifično mjesto gdje ju sustav može vidjeti.

```
cp -v arch/x86_64/boot/bzImage /boot/vmlinuz-linux51816-dm
```

Zatim je potrebno stvoriti početni *ramdisk*, to se čini preko sljedeće Basheve skripte, preko argumenta „*-k 5.18.16*“ određuje se inačica Linuxove jezgre za koju se radi *ramdisk*, a zastavica „*-g*“ određuje na koje se mjesto želi generirati slika *ramdisk*.

```
mkinitcpio -k 5.18.16 -g /boot/initramfs-linux51816-dm.img
```

Potrebno je kopirati datoteku „*System.map*“. U Linuxu je datoteka „*System.map*“ tablica simbola koju rabi jezgra. Tablica simbola je pregled između naziva simbola i njihovih adresa u memoriji. Naziv simbola može biti naziv varijable ili naziv funkcije. Datoteka „*System.map*“ je potrebna kad je potrebna adresa naziva simbola ili naziv simbola adrese. Posebno je koristan za otklanjanje pogrešaka u jezgri.

```
cp System.map /boot/System.map-linux51816-dm
```

Isto tako je potrebno ažurirati *bootloader* kako bi se omogućilo da je novostvorena jezgra primarna.

```
grub-mkconfig -o /boot/grub/grub.cfg
```

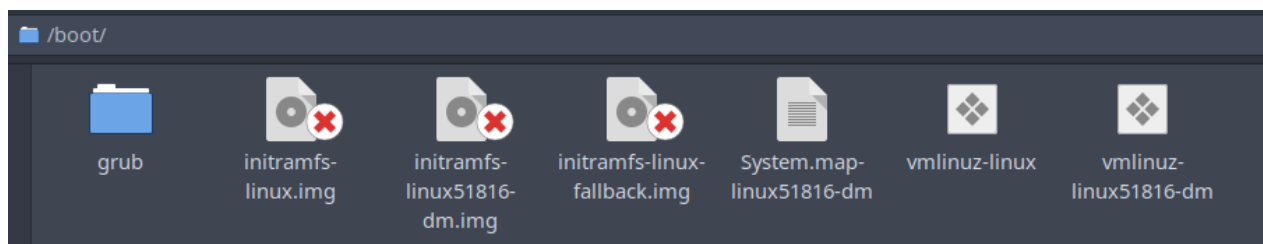
```

[root@dominik-virtualbox linux-5.18.16]# cp -v arch/x86_64/boot/bzImage /boot/vmlinuz-linux51816-dm
'arch/x86_64/boot/bzImage' -> '/boot/vmlinuz-linux51816-dm'
[root@dominik-virtualbox linux-5.18.16]# mkinitcpio -k 5.18.16 -g /boot/initramfs-linux51816-dm.img
==> Starting build: 5.18.16
-> Running build hook: [base]
-> Running build hook: [udev]
-> Running build hook: [autodetect]
-> Running build hook: [modconf]
-> Running build hook: [block]
-> Running build hook: [keyboard]
-> Running build hook: [keymap]
-> Running build hook: [consolefont]
-> Running build hook: [filesystems]
-> Running build hook: [fsck]
==> Generating module dependencies
==> Creating zstd-compressed initcpio image: /boot/initramfs-linux51816-dm.img
==> Image generation successful
[root@dominik-virtualbox linux-5.18.16]# cp System.map /boot/System.map-linux51816-dm
[root@dominik-virtualbox linux-5.18.16]# grub-mkconfig -o /boot/grub/grub.cfg
Generating grub configuration file ...
Found theme: /boot/grub/themes/Vimix/theme.txt
Found linux image: /boot/vmlinuz-linux51816-dm
Found initrd image: /boot/initramfs-linux51816-dm.img
Found linux image: /boot/vmlinuz-linux
Found initrd image: /boot/initramfs-linux.img
Found fallback initrd image(s) in /boot:  initramfs-linux-fallback.img
Warning: os-prober will be executed to detect other bootable partitions.
Its output will be used to detect bootable binaries on them and create new boot entries.
done

```

Slika 23. Ispis procesa instalacije preinačene Linuxove jezgre

Kazalo „/boot“ će po završetku ovoga procesa izgledati ovako. Mogu se opaziti tri nove datoteke koje su nastale prigodom procesa prevođenja i instaliranja ove Linuxove jezgre.



Slika 24. Struktura kazala „/boot“ nakon instalacije jezgre

Unutar datoteke „System.map“ mogu se potražiti nazivi modula kako bi se vidjelo koju memorijsku adresu oni zauzimaju. Može se opaziti da funkcije „hello\_init“, „hello\_exit“, „procesi\_init“ i „procesi\_exit“ imaju uzastopne memorijske adrese što ima smisla kako su se ti moduli uzastopno preveli prigodom prevođenja Linuxove jezgre.

```
System.map-linux51816-dm x
50104 ffffffff81ba7e12 t spi_setup.cold
50105 ffffffff81ba7e95 t __spi_add_device.cold
50106 ffffffff81ba7f33 t spi_add_device.cold
50107 ffffffff81ba7f51 t spi_new_device.cold
50108 ffffffff81ba7f82 t spi_register_controller.cold
50109 ffffffff81ba8093 t slave_store.cold
50110 ffffffff81ba80ab t acpi_register_spi_device.cold
50111 ffffffff81ba80ec t spi_new_ancillary_device.cold
50112 ffffffff81ba8129 t spi_register_board_info.cold
50113 ffffffff81ba8141 t __spi_pump_messages.cold
50114 ffffffff81ba81d7 t loopback_net_init.cold
50115 ffffffff81ba81e5 t blackhole_netdev_xmit.cold
50116 ffffffff81ba81fd t sfp_parse_port.cold
50117 ffffffff81ba8215 t sfp_select_interface.cold
50118 ffffffff81ba822c t sfp_parse_support.cold
50119 ffffffff81ba8244 t hello_init
50120 ffffffff81ba8260 t hello_exit
50121 ffffffff81ba8275 t procesi_init
50122 ffffffff81ba82e7 t procesi_exit
50123 ffffffff81ba8303 t usb_find_alt_setting.cold
50124 ffffffff81ba831d t __usb_get_extra_descriptor.cold
50125 ffffffff81ba8340 t usb_debugfs_cleanup
```

Slika 25. Sadržaj datoteke *System.map-linux51816-dm*

Isto tako unutar kazala „*/boot/grub*“ nalazi se datoteka „*grub.cfg*“ koja se je preinačila u postupku instaliranja Linuxove jezgre, promjene nastale mogu se vidjeti ako se uđe u to kazalo kao administrator i otvori datoteka „*grub.cfg*“. Dodan je *menuentry* pod nazivom „*menuentry "ArcoLinux Linux, with Linux linux51816-dm"*“, „*ArcoLinux*“ je naziv distribucije ArchLinuxa koja se je rabila na virtualnom računalu dok je „*linux51816-dm*“ slika prevedene preinačene jezgre s ovim modulima. Isto se tako može opaziti da se na kraju podiznoga procesa učitava slika *ramdiska* koja se je generirala.

```
echo 'Loading initial ramdisk ...'
```

```
initrd /boot/initramfs-linux51816-dm.img
```

```
grub.cfg x
menuentry "ArcoLinux Linux, with Linux linux51816-dm" --class arcoLinux --class
gnu-linux --class gnu --class os $menuentry_id_option
'gnulinux-linux51816-dm-advanced-fae1c02b-f145-4859-a801-c2f1a90456ee' {
    load_video
    set gfxpayload=keep
    insmod gzio
    insmod part_msdos
    insmod ext2
    set root='hd0,msdos1'
    if [ x$feature_platform_search_hint = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1
        --hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1
        fae1c02b-f145-4859-a801-c2f1a90456ee
    else
        search --no-floppy --fs-uuid --set=root
        fae1c02b-f145-4859-a801-c2f1a90456ee
    fi
    echo 'Loading Linux linux51816-dm ...'
    linux /boot/vmlinuz-linux51816-dm
    root=UUID=fae1c02b-f145-4859-a801-c2f1a90456ee rw quiet loglevel=3 audit=0
    nvme_load=yes
    echo 'Loading initial ramdisk ...'
    initrd /boot/initramfs-linux51816-dm.img
}
```

Slika 26. Sadržaj datoteke *grub.cfg*

Po završetku ovoga procesa moguće je računalo ugasiti pa ponovo upaliti, na podiznom zaslonu pozdravit će sljedeće poruke. To znaменуje da se je svaki korak ispravno učinio i da ovi moduli se ispravno čitaju i izvršavaju.

```
[ 0.546568] Hello world
[ 0.546581] procesi_init: Pozdrav!
[ 0.546581] Lista trenutno aktivnih procesa:
[ 0.546637] Broj aktivnih procesa:78
```

Slika 27. Ispis podiznoga procesa preinačene Linuxove jezgre

Poruke se isto tako mogu vidjeti preko sljedećih naredaba kad se sustav u potpunosti pokrene:

*sudo dmesg i journalctl -k*

```

kol 30 18:05:13 dominik-virtualbox kernel: Hello world
kol 30 18:05:13 dominik-virtualbox kernel: fbcon: Taking over console
kol 30 18:05:13 dominik-virtualbox kernel: procesi_init: Pozdravi
                               Lista trenutno aktivnih procesa:
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: swapper/0          PID: [1]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kthreadd          PID: [2]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: rcu_gp          PID: [3]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: rcu_par_gp       PID: [4]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: netns          PID: [5]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/0:0     PID: [6]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/0:0H    PID: [7]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/u12:0   PID: [8]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/0:1H    PID: [9]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: mm_percpu_wq    PID: [10]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/u12:1   PID: [11]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: rcu_tasks_kthre PID: [12]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: rcu_tasks_rude_  PID: [13]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: rcu_tasks_trace PID: [14]

```

Slika 28. Ispis *journalctl*a preinačene Linuxove jezgre, 1/2

```

kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: ata_sff          PID: [63]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: edac-poller      PID: [64]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: devfreq_wq      PID: [65]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: watchdogd      PID: [66]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/1:1     PID: [67]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kswapd0         PID: [68]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/u12:2   PID: [69]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/u12:3   PID: [71]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kthrotld       PID: [76]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/1:2     PID: [79]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: kworker/2:1     PID: [81]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: acpi_thermal_pm PID: [82]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: scsi_eh_0       PID: [83]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: scsi_tmf_0      PID: [84]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: scsi_eh_1       PID: [85]
kol 30 18:05:13 dominik-virtualbox kernel: Ime procesa: scsi_tmf_1      PID: [86]
kol 30 18:05:13 dominik-virtualbox kernel: Broj aktivnih procesa:78

```

Slika 29. Ispis *journalctl*a preinačene Linuxove jezgre, 2/2

Iz ovih ispisa vidi se da ovi moduli rade ispravno i, ako se u terminalu izvrši naredba *uname -r* koja vraća naziv trenutno djelatne Linuxove jezgre, može se opaziti da će se sad ispisati „5.18.16“ što odgovara broju jezgre koja se je sad prevela i instalirala.

```

[dm@dominik-virtualbox ~]$ uname -r
5.18.16
[dm@dominik-virtualbox ~]$ █

```

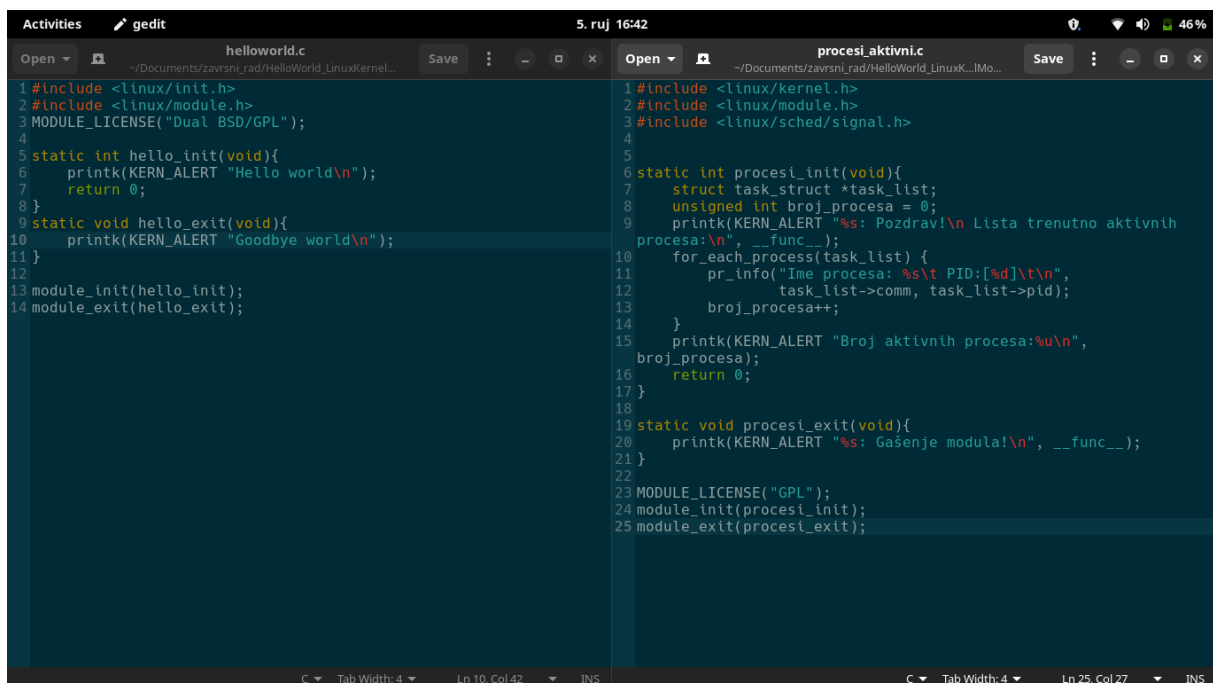
Slika 30. Ispis naredbe *uname -r* nakon instalacije preinačene jezgre



### 5.3. Prevođenje, instalacija, pokretanje, ispitivanje i dokumentacija preinačene jezgre operacijskog sustava na fizičkom računalu

Nakon uspjeha kod ugrađivanja modula u jezgru prevođenja, instaliranja i pokretanja te jezgre na virtualnom računalu, sljedeći korak je bio da se to isto ponovi na stvarnom računalu. Za tu potrebu iskoristio se je stariji prijenosnik na kojem je instaliran OS Linux Manjaro. Kako je Manjaro Linuxova distribucija koja je isto tako zasnovana na ArchLinuxu nije bilo potrebno mijenjati korake potrebne za izvedbu ovoga cilja. Rabila se je ista jezgra Linuxa 5.18.16 skinuta s iste stranice kernel.org. Potrebne datoteke modula i datoteke *makefile* prebacile su se preko GitHubova kazala kako se ne bi trebale ponovno pisati. Potrebno je bilo instalirati potrebne pakete preko naredbe

```
sudo pacman -S base-devel xmlto kmod inetutils bc libelf git cpio
```



```
helloworld.c
1 #include <linux/init.h>
2 #include <linux/module.h>
3 MODULE_LICENSE("Dual BSD/GPL");
4
5 static int hello_init(void){
6     printk(KERN_ALERT "Hello world\n");
7     return 0;
8 }
9 static void hello_exit(void){
10    printk(KERN_ALERT "Goodbye world\n");
11 }
12
13 module_init(hello_init);
14 module_exit(hello_exit);

procesi_aktivni.c
1 #include <linux/kernel.h>
2 #include <linux/module.h>
3 #include <linux/sched/signal.h>
4
5
6 static int procesi_init(void){
7     struct task_struct *task_list;
8     unsigned int broj_procesa = 0;
9     printk(KERN_ALERT "%s: Pozdrav!\n Lista trenutno aktivnih
procesa:\n", __func__);
10    for_each_process(task_list) {
11        pr_info("Ime procesa: %s\t PID:[%d]\t\n",
task_list->comm, task_list->pid);
12        broj_procesa++;
13    }
14    printk(KERN_ALERT "Broj aktivnih procesa:%u\n",
broj_procesa);
15    return 0;
16 }
17 }
18
19 static void procesi_exit(void){
20    printk(KERN_ALERT "%s: Gašenje modula!\n", __func__);
21 }
22
23 MODULE_LICENSE("GPL");
24 module_init(procest_init);
25 module_exit(procesti_exit);
```

Slika 31. Sadržaji datoteka *helloworld.c* i *procesi\_aktivni.c*

Može se vidjeti da su kodovi modula identični onima rabljenima u virtualnom računalu, datoteke *makefile* su isto tako jednake i glase:

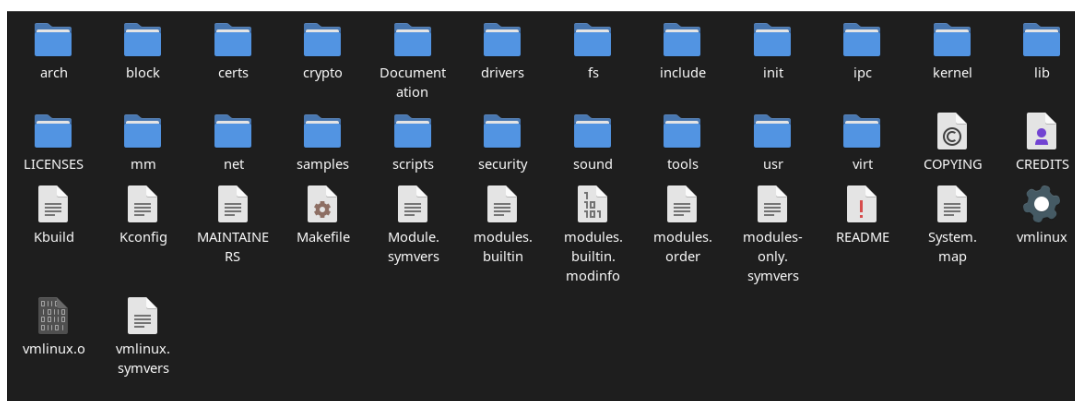
```
obj-y          += helloworld/  
obj-y          += procesi_aktivni/
```

Kao i u virtualnom računalu bilo je potrebno izmijeniti datoteku *makefile* unutar kazala „*/drivers/*“ kako bi moduli bili ugrađeni u jezgru.

```
93 obj-$(CONFIG_VFIO)      += vft0/  
94 obj-y                  += cdrom/  
95 obj-y                  += auxdisplay/  
96 obj-y                  += helloworld/  
97 obj-y                  += procesi_aktivni/  
98 obj-$(CONFIG_PCCARD)   += pcmcia/  
99 obj-$(CONFIG_DIO)      += dio/  
100 obj-$(CONFIG_SBUS)    += sbus/
```

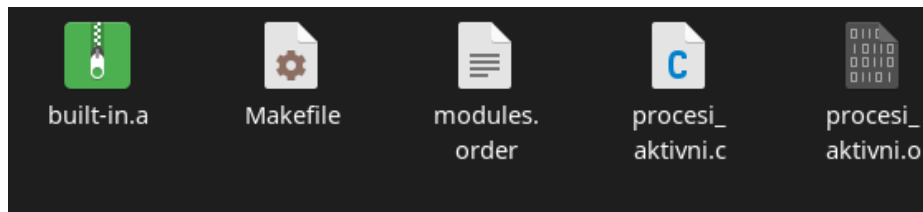
Slika 32. Sadržaj datoteke */drivers/makefile*

Proces prevođenja jezgre nastavlja se tako da se pokrene naredba „*make mrproper*“ koja pripravlja kazala i datoteke za prevođenje jezgre, zatim je potrebno generirati konfiguracijsku datoteku jezgre preko naredbe „*make menuconfig*“. Početak prevođenja jezgre započinje se naredbom *make*, identično kao i na virtualnom računalu.



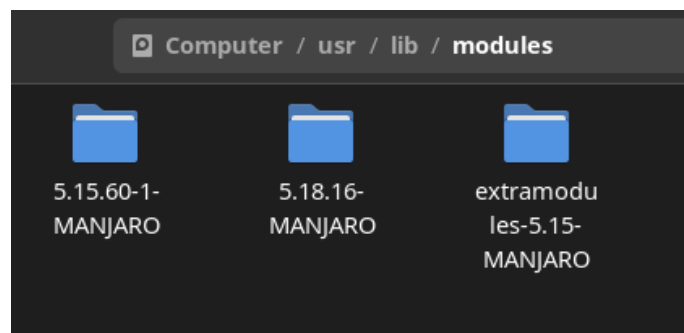
Slika 33. Struktura kazala „5.18.16-MANJARO“ nakon prevođenja jezgre

Po završetku prevođenja koje je u slučaju ovoga prijenosnika trajalo 15 sati kazalo izgleda identično virtualnomu računalu.



Slika 34. Struktura kazala „/drivers/procesi\_aktivni“ nakon prevođenja jezgre

Nastavlja se naredbom za prebacivanje u administratorskoga korisnika *su*. Instaliraju se moduli u odgovarajuće kazalo preko naredbe „*make modules\_install*“, prebacuje se Linuxova slika preko sljedeće naredbe: „*cp -v arch/x86\_64/boot/bzImage /boot/vmlinuz-linux51816-dm*“. Prigodom generiranja *ramdiska* dolazi se do jedine različitosti u usporedbi s virtualnim računalom, a to jest da se mapa u kojoj se nalaze moduli na virtualnom računalu naziva *5.18.16*, a na Manjaru ima naziv *5.18.16-MANJARO*.



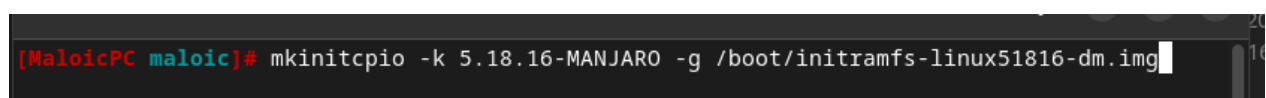
Slika 35. Struktura kazala /usr/lib/modules

Potrebno je preinačiti rabljenu naredbu:

```
mkinitcpio -k 5.18.16 -g /boot/initramfs-linux51816-dm.img
```



```
mkinitcpio -k 5.18.16-MANJARO -g /boot/initramfs-linux51816-dm.img
```



Slika 36. Preinačena naredba *mkinitcpio*

Nastavlja se proces kopiranjem datoteke „System.map“ u odgovarajuće kazalo naredbom

```
cp System.map /boot/System.map-linux51816-dm
```

i ažurira se konfiguracija Gruba preko naredbe

```
grub-mkconfig -o /boot/grub/grub.cfg.
```

Ako su se sve naredbe ispravno napisale kad se ponovno pokrene računalo unutar dnevnika vidjet će se ovaj ispis koji govori da su ovi moduli ispravno ugrađeni u Linuxovu jezgru na stvarnom računalu.

```
ruj 06 13:57:53 MaloicPC kernel: ata2: SATA max UDMA/133 abai m2048@0x91413000 p
ruj 06 13:57:53 MaloicPC kernel: Hello world
ruj 06 13:57:53 MaloicPC kernel: procesi_init: Pozdrav!
                               Lista trenutno aktivnih procesa:
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: swapper/0          PID: [1]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: kthreadd         PID: [2]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: rcu_gp            PID: [3]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: rcu_par_gp        PID: [4]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: netns            PID: [5]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: kworker/0:0      PID: [6]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: kworker/0:0H     PID: [7]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: kworker/u8:0     PID: [8]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: mm_percpu_wq     PID: [9]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: kworker/u8:1     PID: [10]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: rcu_tasks_kthre   PID: [11]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: rcu_tasks_rude_   PID: [12]
```

Slika 37. Ispis dnevnika preinačene jezgre, 1/2

```
Activities Terminal 6. ruj 13:59
journalctl -k -xe
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: scsi_tmf_0          PID: [62]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: scsi_eh_1         PID: [63]
ruj 06 13:57:53 MaloicPC kernel: Ime procesa: scsi_tmf_1        PID: [64]
ruj 06 13:57:53 MaloicPC kernel: Broj aktivnih procesa:57
ruj 06 13:57:53 MaloicPC kernel: ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
```

Slika 38. Ispis dnevnika preinačene jezgre, 2/2

## 6. Zaključak

Jedan dio ovoga završnoga rada je bilo upoznavanje s izradbom preinačene Linuxove jezgre, ali više vremena je bilo potrebno za proučavanje kako funkcioniraju moduli i kako izraditi modul za Linuxovu jezgru. Ponajviše zbog toga jer je izradba Linuxove jezgre iz izvornih datoteka dobro dokumentirana u različitim člancima, Linuxovim *wiki*-stranicama kao i u mnogobrojnim YouTubeovim videima, dok je izradba modula tema koja je vrlo malo obrađena na Internetu. Potrebno je bilo traženje ne samo po stranicama ArchWikija nego i po službenoj dokumentaciji o Linuxovoj jezgri. Kako su moduli u Linuxu jednakovrijednice pogoniteljima u Windowsima vrlo malen broj ljudi će se susresti s problemom pisanja rečenih, tako da se ne smatra prevelikim problemom malen broj dostupne dokumentacije.

OS Linux nije nešto novo u svijetu tehnologije, ali kako njegova popularnost raste većina ljudi će se kad-tad susresti s Linuxom. Što više ljudi rabi Linux tim će biti veća potražnja za programerima koji rade ili su radili programe za OS Linux. Zbog toga se smatra da je ovaj rad vrlo koristan autoru osobno; više se je upoznao s radom Linuxa nego što je mogao zamisliti na početku ovoga rada.

# Popis literature

[1] "Dokumentacija kbuild", [Na internetu] Dostupno:

<https://www.kernel.org/doc/Documentation/kbuild/kbuild.txt> [Pristupano 6.8.2022.]

[2] "Dokumentacija makefiles", [Na internetu] Dostupno:

<https://www.kernel.org/doc/Documentation/kbuild/makefiles.txt> [Pristupano 6.8.2022.]

[3] "Dokumentacija modules", [Na internetu] Dostupno:

<https://www.kernel.org/doc/Documentation/kbuild/modules.txt> [Pristupano 6.8.2022.]

[4] "Kernel/Arch Build System", [Na internetu]

Dostupno: [https://wiki.archlinux.org/title/Kernel/Arch\\_Build\\_System](https://wiki.archlinux.org/title/Kernel/Arch_Build_System) [Pristupano 6.8.2022.]

[5] Jake Edge (25 November 2008). "Character devices in user space", [Na internetu]

Dostupno: <https://lwn.net/Articles/308445/> [Pristupano 6.8.2022.]

[6] "Chapter 12 Modules", [Na internetu] Dostupno:

<https://tldp.org/LDP/tlk/modules/modules.html> [Pristupano 6.8.2022.]

[7] "Learn Makefiles", [Na internetu] Dostupno: <https://makefiletutorial.com/> [Pristupano 6.8.2022.]

[8] "How to compile and install Linux Kernel 5.16.9 from source code", [Na internetu] Dostupno:

<https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html> [Pristupano 13.8.2022.]

[9] "Dokumentacija sched.h zaglavlja", [Na internetu] Dostupno:

[https://docs.huihoo.com/doxygen/linux/kernel/3.7/include\\_2linux\\_2sched\\_8h\\_source.html#01337](https://docs.huihoo.com/doxygen/linux/kernel/3.7/include_2linux_2sched_8h_source.html#01337) [Pristupano 27.8.2022.]

[10] "Tux.png", [Na internetu] Dostupno:

<https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Tux.png/220px-Tux.png>

[Pristupano 8.9.2022.]

# Popis slika

Slika 1. Linuxov logotip „Tux“ (Izvor: Larry Ewing, 1996).....	2
Slika 2. Program Oracle VM VirtualBox.....	6
Slika 3. ArcoLinux s inačicom 5.18.15-arch1-2 Linuxove jezgre.....	7
Slika 4. Sadržaj <i>makefile</i> -datoteke jednostavnoga Helloworld-modula .....	9
Slika 5. Sadržaj <i>helloworld.c</i> -datoteke jednostavnoga Helloworld-modula.....	10
Slika 6. Kazalo Helloworld-modula nakon izgradnje .....	11
Slika 7. Rezultat pokretanja naredba <code>ls, lsmod   grep helloworld</code> i <code>sudo insmod ./helloworld.ko</code> 11	
Slika 8. Ispis međuspremnikova poruka jezgre nakon učitavanja i isključivanja Helloworld-modula.....	12
Slika 9. Sadržaj <i>procesi_aktivni.c</i> -datoteke jednostavnoga modula „Aktivni procesi“ .....	12
Slika 10. Sadržaj <i>makefile</i> -datoteke modula „Aktivni procesi“ .....	13
Slika 11. Ispis međuspremnikova poruka jezgre nakon učitavanja modula „Aktivni procesi“, 1/2 14	
Slika 12. Ispis međuspremnikova poruka jezgre nakon učitavanja modula „Aktivni procesi“, 2/2 14	
Slika 13. Kazalo <i>/etc/modules-load.d/</i> i sadržaj datoteke <i>helloworld.conf</i> .....	15
Slika 14. Struktura kazala <i>/linux-5.18.16/drivers/helloworld</i> .....	18
Slika 15. Struktura kazala <i>/linux-5.18.16/drivers/procesi_aktivni</i> .....	18
Slika 16. Sadržaj datoteke <i>/linux-5.18.16/drivers/makefile</i> .....	18
Slika 17. <i>menuconfig</i> – grafički konfigurator Linuxove jezgre .....	19
Slika 18. Pohrana konfiguracijske datoteke „ <i>config</i> “ .....	20
Slika 19. Struktura kazala nakon prevođenja jezgre .....	20
Slika 20. Sadržaj datoteke <i>modules.builtin</i> .....	21
Slika 21. Struktura kazala „ <i>procesi_aktivni</i> “ nakon prevođenja jezgre .....	21
Slika 22. Struktura kazala „ <i>helloworld</i> “ nakon prevođenja jezgre .....	22
Slika 23. Ispis procesa instalacije preinačene Linuxove jezgre .....	23
Slika 24. Struktura kazala „ <i>boot</i> “ nakon instalacije jezgre.....	23

Slika 25. Sadržaj datoteke <i>System.map-linux51816-dm</i> .....	24
Slika 26. Sadržaj datoteke <i>grub.cfg</i> .....	25
Slika 27. Ispis podiznoga procesa preinačene Linuxove jezgre .....	25
Slika 28. Ispis <i>journalctl</i> preinačene Linuxove jezgre, 1/2.....	26
Slika 29. Ispis <i>journalctl</i> preinačene Linuxove jezgre, 2/2.....	26
Slika 30. Ispis naredbe <i>uname -r</i> nakon instalacije preinačene jezgre.....	26
Slika 31. Sadržaji datoteka <i>helloworld.c</i> i <i>procesi_aktivni.c</i> .....	27
Slika 32. Sadržaj datoteke <i>/drivers/makefile</i> .....	28
Slika 33. Struktura kazala „5.18.16-MANJARO“ nakon prevođenja jezgre .....	28
Slika 34. Struktura kazala „ <i>/drivers/procesi_aktivni</i> “ nakon prevođenja jezgre .....	29
Slika 35. Struktura kazala <i>/usr/lib/modules</i> .....	29
Slika 36. Preinačena naredba <i>mkinitcpio</i> .....	29
Slika 37. Ispis dnevnika preinačene jezgre, 1/2 .....	30
Slika 38. Ispis dnevnika preinačene jezgre, 2/2 .....	30