



# Univerzitet u Novom Sadu

## Fakultet tehničkih nauka

### Odsek za računarsku tehniku i računarske komunikacije



## Uvod u linuks



Linuks sistem datoteka

# UVOD U LINUKS



# Sadržaj (1/4)



- ❖ Šel, sistem datoteka i rukovanje datotekama
  - ❖ Sve je datoteka
  - ❖ GNU / Linuks struktura sistema datoteka
  - ❖ Interpreteri komandne linije
  - ❖ Rukovanje datotekama i direktorijumima
  - ❖ Prikaz, pretraga sadržaja i sortiranje datoteka
  - ❖ Simbolički i hard linkovi
  - ❖ Prava pristupa datotekama



## Sadržaj (2/4)

- ❖ Standardni U/I, preusmeravanja, pajpovi
  - ❖ Standardni ulaz i izlaz, preusmeravanje u datoteku
  - ❖ Pajpovi: preusmeravanje standardnog izlaza na ulaz drugih komandi
  - ❖ Izlaz standardne greške



## Sadržaj (3/4)

### ❖ Kontrola zadataka

- ❖ Puna kontrola zadataka
- ❖ Izvršavanje zadatka u pozadini, suspendovanje, nastavljanje i prekidanje
- ❖ Prikaz aktivnih zadataka
- ❖ Ubijanje procesa Killing processes
- ❖ Varijable okruženja
- ❖ PATH varijabla okruženja
- ❖ Šel alijsi, .bashrc datoteka



# Sadržaj (4/4)

## ❖ Razno

- ❖ Tekst editori
- ❖ Kompresovanje i arhiviranje
- ❖ Štampanje datoteka
- ❖ Poređenje datoteka i direktorijuma
- ❖ Pretraga datoteka
- ❖ Informacije o korisnicima
- ❖ Razne komande
- ❖ Sistem administracija
- ❖ SSH
- ❖ Razvoj aplikacija



# Linuks sistem datoteka

- ❖ Skoro sve je datoteka
  - ❖ Regularna datoteka
  - ❖ Direktorijum
    - ❖ Datoteka sa listom drugih datoteka
  - ❖ Simbolički linkovi
    - ❖ Datoteka koja se odnosi na neku drugu datoteku
  - ❖ Uređaji i periferije
    - ❖ Izlazno ulazni uređaji kao datoteke
  - ❖ Pajpovi
    - ❖ Služe za vezivanje programa (npr. **cat log | grep error**)
  - ❖ Soketi
    - ❖ Međuprocesna komunikacija



# Imena datoteka

- ❖ Razlikuju mala i velika slova
- ❖ Nisu ograničene dužine
- ❖ Mogu da sadrže bilo koji znak osim "/"
- ❖ Tip datoteke u samoj datoteci
- ❖ Ekstenzija nije bitna
- ❖ Primeri:
  - ❖ README
  - ❖ .bashrc (tačka na početku označava sakrivenu datoteku)
  - ❖ Index.htm
  - ❖ Index.html (ekstenzija nije bitna)
  - ❖ Index.html.old.123
  - ❖ Lista grešaka



# Putanje do datoteka

- ❖ Putanja je niz ugnježdenih direktorijuma razdvojenih znakom / sa direktorijumom ili datotekom na kraju
- ❖ Tipovi putanje:
  - ❖ Relativna
    - ❖ Documents/txt/README
      - ❖ Počinje imenom direktorijuma ili datoteke
      - ❖ Relativna u odnosu na trenutni direktorijum
    - ❖ Apsolutna
      - ❖ /home/rtrk/Documents/txt/README
        - ❖ Uvek počinje znakom /
  - ❖ / - korenski direktorijum
    - ❖ Početak apsolutne putanje svih datoteka u sistemu



# Struktura sistema datoteka (1/3)

- ❖ Ne postoji obavezna struktura
- ❖ Najčešće se koristi:
  - ❖ / - korenksi direktorijum
  - ❖ **/bin/** - osnovne, esencijalne sistemske komande
  - ❖ **/boot/** - slike kernela, initrd, konfiguracijske datoteke
  - ❖ **/dev/** - datoteke koje predstavljaju uređaje
  - ❖ **/etc/** - sistemske konfiguracijske datoteke
  - ❖ **/home/** - korisnički direktorijumi
  - ❖ **/lib/** - osnovne sistemske deljene biblioteke
  - ❖ **/lost+found/** - oštećene datoteke koje je sistem pokušao da oporavi
  - ❖ **/media/** - odredište za mauntovanje eksternih medijuma (npr. `/media/usbdisk`, `/media/cdrom`)



## Struktura sistema datoteka (2/3)

- ❖ **/mnt/** - odredište za mauntovanje privremeno mauntovanih sistema datoteka
- ❖ **/opt/** - specifični alati instalirani od strane administratora (**/usr/local/** se često koristi kao alternativa)
- ❖ **/proc/** - pristup sistemskim informacijama (**/proc/cpuinfo**, **/proc/version**, ...)
- ❖ **/root/** - direktorijum korenskog korisnika
- ❖ **/sbin/** - komande koje koristi samo administrator
- ❖ **/sys/** - kontrola sistema i uređaja (frekfencija CPU, snaga uređaja, itd)
- ❖ **/tmp/** - privremene datoteke
- ❖ **/usr/** - korisnički alati (nisu esencijalni za sistem) - **/usr/bin/**, **/usr/lib/**, **/usr/sbin/**, ...



## Struktura sistema datoteka (3/3)

- ❖ **/usr/local/** - specifični alati instalirani od strane administratora (alternativa **/opt**)
- ❖ **/var/** - podaci koje koriste sistem ili sistemski serveri (**/var/log/**, **/var/spool/mail/** (primljena pošta), **/var/spool/lpd/** (zadaci za štampač), ...)
- ❖ Linuks sistem datoteka je definisan „Filesystem Hierarchy Standard“ (FHS) standardom
- ❖ <http://www.pathname.com/fhs/>



Šel i rad sa datotekama

# UVOD U LINUKS



# Šel – interpreter komandne linije

- ❖ Šel – alat koji izvršava korisničke komande
- ❖ Naziv šel (čaura, omotač) zato što sakriva detalje sistemskih operacija
- ❖ Komande se unose u tekstualni terminal (prozor u grafičkom okruženju ili čisto tekstualna konzola)
- ❖ Rezultati se ispisuju u terminal (nema potrebe za grafikom)
- ❖ Moguće je pisati skripte (obezbeđuju sve potrebne resurse za pisanje kompleksnih programa (variable, uslovi, petlje...))



# Popularni šelovi

- ❖ Najpoznatiji šelovi su:
  - ❖ **sh:** Bornov šel (zastareo)
    - ❖ Tradicionalan, bazični šel u linuks sistemima
    - ❖ Razvio ga je Stiv Born
  - ❖ **csh:** C šel (zastareo)
    - ❖ Sintaksa slična C programskom jeziku
  - ❖ **tcsch:** TC šel (i dalje prilično popularan)
    - ❖ Naprednija verzija C shella
    - ❖ Dodate funkcionalnosti (završavanje komandi, menjanje prošlih komandi, ...)
  - ❖ **bash:** „*The Bourne Again shell*“ (najpopularniji)
    - ❖ Naprednija verzija sh šela sa mnogo dodatih funkcionalnosti
- ❖ Čest slučaj – preusmeravanje sh i csh šelova na bash i tcsh šelove respektivno



# Šabloni u imenu datoteke

- ❖ \* - menja bilo koju grupu karaktera
- ❖ ? - menja jedan (bilo koji) karakter
- ❖ Lakše pokazati na primerima
- ❖ **ls \*txt**

❖ šel prvo zameni \*txt sa svim datotekama i direktorijumima koji se završavaju sa txt (uključujući i .txt) sem onih koji počinju sa ., a zatim izvršava ls komandu

- ❖ **ls -d .\***

❖ Izlistava sve datoteke i direktorijume koji počinju sa .  
❖ -d govori ls komandi da ne izlistava sadržaj direktorijuma

- ❖ **ls ?.log**

❖ izlistava sve datoteke čije ime počinje sa 1 karakterom, pa sledi .log



# Specijalni direktorijumi (1/2)

❖ . /

- ❖ trenutni direktorijum
- ❖ Koristi se u komandama koje uzimaju direktorijum kao argument
- ❖ koristi se za pokretanje komandi ili skripti iz trenutnog direktorijuma
- ❖ ./readme.txt i readme.txt su ista komanda

❖ .. /

- ❖ roditeljski (prethodni) direktorijum
- ❖ uvek pripada . (trenutnom direktorijumu, videti ls -a)
- ❖ Jedina veza sa roditeljskim direktorijumom
- ❖ Tipična primena **cd ..(../..)** - vrati se jedan (ili više) direktorijuma iznad u hijerarhiji



## Specijalni direktorijumi (2/2)

❖ ~ /

- ❖ zapravo nije specijalni direktorijum, šelovi ga zamene putanjom do korisničkog (**/home/<korisnik>**) direktorijumom trenutnog korisnika
- ❖ nije ga moguće koristiti u svim programima jer nije pravi direktorijum

❖ ~rtrk/

- ❖ slično kao i ~ /
- ❖ šel ga zameni putanjom do korisničkog direktorijuma korisnika rtrk



# cd i pwd komande, pushd, popd

## ❖ **cd <dir>**

- ❖ menja trenutni direktorijum u <dir>

## ❖ **cd -**

- ❖ postavlja prethodni direktorijum kao trenutni
- ❖ zgodno za kretanje između dva direktorijuma

## ❖ **pwd**

- ❖ ispisuje absolutnu putanju do trenutnog direktorijuma

## ❖ **pushd <dir>**

- ❖ postavlja trenutni direktorijum (rezultat pwd komande) na stek i izvršava cd <dir>

## ❖ **popd**

- ❖ skida poslednji direktorijum sa steka i postavlja ga kao trenutni



# Prikazivanje sadržaja datoteke

- ❖ Nekoliko načina za prikazivanje sadržaja datoteke
  - ❖ **cat dat1 dat2 dat3 ... (concatenate)**
    - ❖ Konkatenira i prikazuje sadržaj datih datoteka
  - ❖ **more dat1 dat2 dat3 ...**
    - ❖ Posle svake strane staje i čeka komandu od korisnika da prikaže sledeću
    - ❖ Moguća pretraga - / <tražena\_reč>
  - ❖ **less dat1 dat2 dat3 ...**
    - ❖ **less** je bolji od **more**
    - ❖ Ne čita celu datoteku pre početka
    - ❖ Moguća pretraga unazad ? <tražena\_reč>



# grep komanda

## ❖ grep <šablon> <datoteke>

- ❖ Skenira date datoteke i prikazuje linije koje se poklapaju sa šablonom

## ❖ grep error \*.log

- ❖ Prikazuje sve linije koje sadrže reč **error** u **\*.log** datotekama

## ❖ grep -i error \*.log

- ❖ Isto kao prethodno, samo što ne pravi razliku između malih i velikih slova

## ❖ grep -ri error .

- ❖ Isto kao prethodno, samo što rekurzivno pretražuje sve datoteke u trenutnom direktorijumu i njegovim poddirektorijumima

## ❖ grep -v info \*.log

- ❖ postavlja negativni filter
- ❖ prikazuje sve linije sem onih koje sadrže reč **info**



# Regularni izrazi

- ❖ Primer korišćenja: sed komanda
- ❖ Pronalaze podudaranja u tekstu sa izrazom
  - ❖ . odgovara bilo kom karakteru
  - ❖ [ ] odgovara bilo kom karakteru napisanom između zagrada
  - ❖ [^ ] odgovara bilo kom karakteru koji nije naveden između zagrada
  - ❖ ^ odgovara početku linije
  - ❖ \$ odgovara kraju linije
  - ❖ \* odgovara ponavljanju prethodnog elementa 0 ili više puta
  - ❖ + odgovara ponavljanju prethodnog elementa 1 ili više puta
  - ❖ ? odgovara ponavljanju prethodnog elementa 0 ili 1 put
  - ❖ \(\backslash\)\(n\) definije podizraz koji se kasnije može pozvati sa \n, gde je n redni broj definisanog podizraza



# Simbolički linkovi

- ❖ Simbolički link je specijalna datoteka koja predstavlja referencu na ime druge datoteke ili direktorijuma
- ❖ Korisno da se smanji zauzeće memorije na disku kada dve datoteke imaju isti sadržaj
- ❖ Primer:
  - ❖ **anakin\_skywalker\_biography -> darth\_vader\_biography**
- ❖ Kako prepoznati simbolčke linkove
  - ❖ **ls -l** prikazuje **<naziv\_linka> -> <naziv\_datoteke>**
  - ❖ GNU **ls** prikazuje linkove u drugačijoj boji

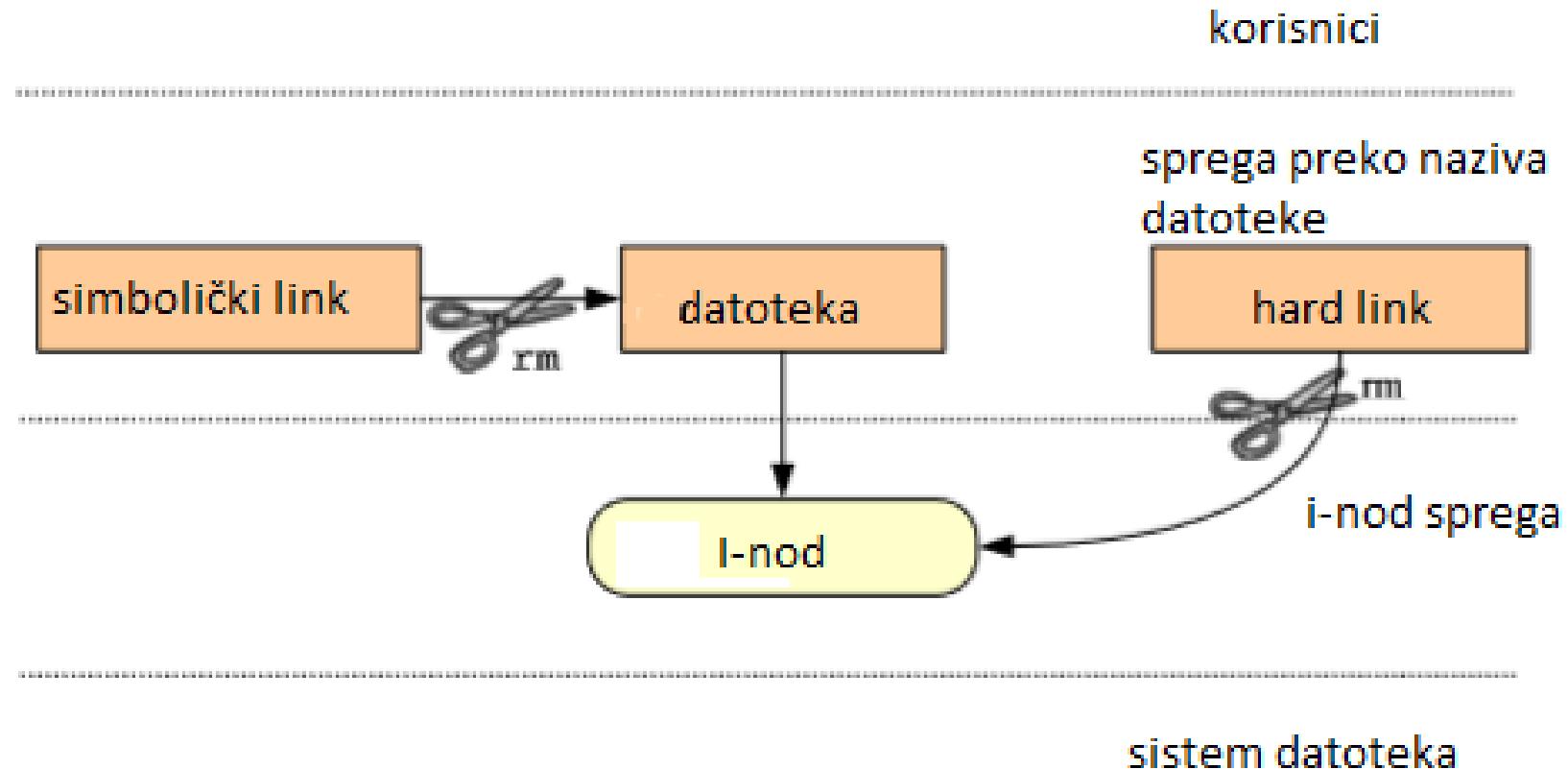


# Hard linkovi

- ❖ Podrazumevano ponašanje **In** komande je da pravi hard linkove (bez **-s** parametra)
- ❖ Hard link koji pokazuje na neku datoteku je regularna datoteka sa potpuno istim sadržajem
- ❖ Hard linkovi se ne mogu razlikovati od običnih datoteka, a opet štede memorijski prostor na disku
- ❖ Brisanje originalne datoteke ne utiče na sadržaj hard linka
- ❖ Sadržaj je obrisan kad više ne postoje hard linkovi koji pokazuju na njega

# Nazivi datoteka i i-nodovi

- ❖ Za bolje razumevanje simboličkih (*soft*) i hard linkova





Dokumentacija o komandama

# UVOD U LINUKS



# Pomoć oko komandi

- ❖ Većina Linuks komandi ima barem jednu opciju prosleđenu kao argument za pomoć oko korišćenja
- ❖ **-h** (- se uglavnom koristi za argumente od jednog karaktera)
- ❖ **--help** ( -- se uglavnom koristi za duže verzije argumenata)
- ❖ U slučaju pogrešnog korišćenja komande uglavnom se ispiše kratko uputstvo za upotrebu



# Priručnik



## ❖ **man <komanda>**

❖ Prikazuje jednu ili više strana priručnika za komandu

## ❖ **man man**

❖ Pored Linuks komandi, postoje i priručnici o nekim C funkcijama, zaglavljima ili strukturama podataka ili o konfiguracijskim datotekama

## ❖ **man stdio.h**

## ❖ **man fstab (za /etc/fstab)**

❖ Priručnici se traže na putanjama navedenim u okviru **MANPATH** varijable okruženja



Korisnici i prava pristupa

# UVOD U LINUKS



# Prava pristupa datotekama

- ❖ Prava pristupa se proveravaju sa **ls -l** komandom
- ❖ 3 tipa prava pristupa
  - ❖ pravo čitanja (**r**)
  - ❖ pravo pisanja (**w**)
  - ❖ pravo izvršavanja (**x**)
- ❖ 3 nivoa grupa pristupa
  - ❖ korisnik (**u** - *user*): vlasnik datoteke
  - ❖ grupa (**g**): grupa predstavlja listu korisnika
  - ❖ Ostali (**o**): svi korisnici



# Ograničenja vezana za prava pristupa



- ❖ **x** bez **r** je legitimno ali beskorisno, datoteka se mora pročitati da bi se izvršila
- ❖ Direktorijumi moraju da imaju **x** i **r** prava pristupa
  - ❖ **x** za pristup
  - ❖ **r** za čitanje, odnosno listanje sadržaja
- ❖ Nemoguće je preimenovati, obrisati ili kopirati datoteke u direktorijumu ako ne postoje **w** prava na direktorijumu
- ❖ Ukoliko postoje **w** prava na direktorijumu, moguće je obrisati datoteku i ako ne postoje **w** prava na datoteci
  - ❖ Ovo omogućava izmenu na datoteci bez **w** prava (obrisati i napraviti novu sa istim imenom)



# Primeri prava pristupa

❖ - - - - -

**tip ru wr xu rg wg xu ro wo xo**

❖ **-rw-r--r--**

❖ Vlasnik može da čita i piše a ostali da čitaju

❖ **-rw-r-----**

❖ Vlasnik može da čita i piše, korisnici iz grupe kojoj datoteka pripada da čitaju

❖ **drwx-----**

❖ Direktorijum kojem može da pristupi samo vlasnik

❖ **-----r-x**

❖ Datoteka kojoj mogu da pristupe ostali, ali ne može vlasnik ni korisnici iz grupe kojoj datoteka pripada



# Vlasnik datoteke

- ❖ Posebno korisno u namenskim sistemima kada se kreiraju datoteke za drugi sistem
- ❖ **chown -R rtrk /home/linuks/src**
  - ❖ postavlja korisnika **rtrk** za vlasnika svih datoteka na putanji **/home/linuks/src**
  - ❖ **-R:** rekurzivno
- ❖ **chgrp -R rtrk-grupa /home/linuks/src**
  - ❖ postavlja sve na putanji **/home/linuks/src** u grupu **rtrk-grupa**
- ❖ **chown -R rtrk:rtrk-grupa /home/linuks/src**
  - ❖ obe promene mogu da se izvrše istovremeno u okviru jedne komande



# Korišćenje korenskih prava

- ❖ Za određene komande potrebna su korenska (*root*) prava
- ❖ Ukoliko korisnik poseduje korensku šifru
  - ❖ **su** - (*switch user*)
- ❖ Moguće je koristiti korenska prava i sa svojim nalogom kucanjem komande **sudo** pre komande koja se izvršava
  - ❖ Primer:
    - ❖ **sudo mount /dev/sdb1 /mnt/usb**



Standardni ulaz/izlaz, redirekcije, pajpovi

# UVOD U LINUKS



# Standardni izlaz

- ❖ Sve komande prikazuju tekst u terminalu tako što pišu na svoj standardni izlaz
- ❖ Standardni izlaz može da se preusmeri u datoteku korišćenjem simbola > (piše od početka datoteke)
- ❖ Standardni izlaz može da se preusmeri na kraj datoteke korišćenjem simbola >> (piše posle poslednje linije u datoteci)



# Primeri preusmeravanja standardnog izlaza

- ❖ **ls ~rtrk/knjige/\* > biblioteka/spisak\_knjiga.txt**
- ❖ **cat obiwan\_kenobi.txt > starwars\_biographies.txt**  
**cat han\_solo.txt >> starwars\_biographies.txt**
- ❖ **echo „README: No such file or directory“ > README**
  - ❖ Jedan od načina da se napravi nova datoteka



# Standardni ulaz

❖ Mnoge komande mogu da uzmu ulaz sa standardnog ulaza (tastatura) ukoliko im se ne proslede ulazni argumenti pri pokretanju

❖ Primer

❖ Ulaz:  
**sort**  
**windows**  
**linux**  
**[Ctrl][D]**

Izlaz:  
**linux**  
**windows**

**sort** u ovom slučaju  
uzima ulazne parametre  
sa standardnog ulaza do  
pojave **[Ctrl][D]**

❖ **sort < spisak.txt**

❖ Standardni ulaz komande **sort** je uzet iz datoteke  
**spisak.txt**



# Pajpovi

- ❖ Pajpovi su veoma korisni za prosleđivanje standardnog izlaza jedne komande na standardni ulaz druge
- ❖ Primeri:
  - ❖ `cat *.log | grep -i error | sort`
  - ❖ `grep -ri error . | grep -v "ignored" | sort -u \> serious_errors.log`
  - ❖ `cat /home/*/homework.txt | grep mark | more`
- ❖ Jedna od najmoćnijih funkcionalnosti u Linuks šelovima



# tee komanda

- ❖ **tee [-a] <datoteka>**
  
- ❖ **tee** komanda se koristi da pošalje standardni izlaz i na ekran i u datoteku
  
- ❖ Primeri:
  - ❖ **make | tee build.log**
    - ❖ Pokreće **make** komandu i smešta izlaz u datoteku **build.log** i prikazuje ga na ekranu
  - ❖ **make install | tee -a build.log**
    - ❖ Pokreće **make install** komandu i smešta izlaz na kraj datoteke **build.log** (u nastavku) i prikazuje ga na ekranu
    - ❖ **-a:** (*append*)



# Izlaz standardne greške

- ❖ Poruke vezane za greške se uglavnom prikazuju na standardnom izlazu za greške, a ne na standardnom izlazu ukoliko je program dobro napisan
- ❖ Preusmeravanje standardnog izlaza greške se postiže sa **2>**, odnosno **2>>**
- ❖ Primer:
  - ❖ **cat f1 f2 nofile > newfile 2> errfile**
- ❖ Napomena: **1** je deskriptor za standardni izlaz, odnosno **1>** je isto što i **>**
- ❖ Standardni izlaz i standardni izlaz greške se mogu zajedno preusmeriti korišćenjem **&>**
  - ❖ **cat f1 f2 nofile &> wholefile**



# Specijalni uređaji (1/3)

- ❖ Uređaji (*devices*) sa specijalnim ponašanjem ili sadržajem
- ❖ **/dev/null**
  - ❖ Slivnik za podatke, uništava sve podatke koji mu se proslede
  - ❖ Koristan za uklanjanje neželjenih ispisa
  - ❖ **mplayer black\_adder\_4th.avi &> /dev/null**
- ❖ **/dev/zero**
  - ❖ Čitanje ove datoteke uvek vraća **\0**
  - ❖ Koristan za kreiranje datoteke ispunjene nulama
  - ❖ **dd if=/dev/zero of=disk.img bs=1k count=2048**



# Specijalni uređaji (2/3)

## ❖ /dev/random

- ❖ Čitanje ove datoteke vraća nasumične bajte
- ❖ Uglavnom se koristi u kriptografskim programima
- ❖ Koristi prekide na drugim uređajima kao izvor entropije
- ❖ Može da blokira dok se ne sakupi dovoljan broj prekida

## ❖ /dev/urandom

- ❖ Koriste ga programi kojima je dovoljna pseudo nasumičnost
- ❖ Uvek generiše nasumične bajte i ako nema dovoljno prekida na drugim uređajima
- ❖ lakši za predvideti, ali i dalje predviđanje ne spada u lak posao



## Specijalni uređaji (3/3)

### ❖ /dev/full

- ❖ Imitira uređaj sa popunjrenom memorijom
- ❖ Koristan za testiranje programa u slučaju popunjenoštiti memorijskog diska



Kontrola zadatka

# UVOD U LINUKS



# Potpuna kontrola zadataka

- ❖ Od samih početaka Linuks podržava konkurentne zadatke sa istiskivanjem
- ❖ Mogućnost da se pokrene veliki broj zadataka u paraleli, kao i da se zadaci prekinu čak i ako se dovedu u nekonzistentno stanje
- ❖ Mogućnost izbora programa koji se pokreću
- ❖ Mogućnost izbora odakle pokrenuti programi uzimaju ulazne argumente i gde će se ispisivati njihov izlaz



# Procesi



- ❖ U linuksu je skoro sve datoteka
- ❖ Ono što nije datoteka je proces
- ❖ Procesi su instance pokrenutih programa
- ❖ Više instanci istog programa može biti pokrenuto istovremeno
- ❖ Podaci koji se dodeljuju procesima:
  - ❖ otvorene datoteke
  - ❖ alocirana memorija
  - ❖ stek
  - ❖ id procesa
  - ❖ roditeljski proces
  - ❖ prioritet
  - ❖ ...



# Pokretanje zadataka u pozadini

- ❖ Ista notacija u svim šelovima
  
- ❖ Za šta se koristi?
  - ❖ Za komande čiji izlaz može da bude pregledan naknadno, pogotovo za one koje troše dosta vremena
  - ❖ Za pokretanje grafičkih programa iz komandne linije, a da ostane omogućeno korišćenje istog terminala
  
- ❖ Kako se koristi?
  - ❖ Dodaje se & na kraj linije za pokretanje komande
  - ❖ **pozadinski\_program &**
  - ❖ **gedit &**



# Kontrola pozadinskih zadataka

## ❖ **jobs**

- ❖ Vraća listu pozadinskih zadataka pokrenutih iz trenutnog šela
- ❖ **[1]-- Running ~/bin/find\_meaning\_of\_life ----without-- god &**
- [2]+ Running make mistakes &**

## ❖ **fg**

**fg %<n>**

- ❖ prebacuje poslednji (n-ti) zadatak u fokus

## ❖ **[Ctrl] Z**

**bg**

- ❖ prebacuje trenutni zadatak u pozadinu

## ❖ **kill %<n>**

- ❖ Prekida n-ti zadatak



# Primer kontrole zadatka

## ❖ **jobs**

[1]-- Running ~/bin/find\_meaning\_of\_life --  
--without-god &  
[2]+ Running make mistakes &

## ❖ **fg**

make mistakes

## ❖ **[Ctrl] Z**

[2]+ Stopped make mistakes

## ❖ **bg**

[2]+ Stopped make mistakes

## ❖ **kill %1**

[1]+ Terminated ~/bin/find\_meaning\_of\_life --without-g  
od



# Listanje svih procesa

- ❖ Bez obzira iz kojeg šela su pokrenuti
- ❖ **ps -ux**
  - ❖ Izlistava sve procese koji pripadaju trenutnom korisniku
- ❖ **ps -aux**
  - ❖ Izlistava sve procese pokrenute u sistemu
- ❖ **ps -aux | grep bart | grep bash**

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
bart	3039	0.0	0.2	5916	1380	pts/2	S	14:35	0:00	/bin/bash
bart	3134	0.0	0.2	5388	1380	pts/3	S	14:36	0:00	/bin/bash
bart	3190	0.0	0.2	6368	1360	pts/4	S	14:37	0:00	/bin/bash
bart	3416	0.0	0.0		0	0 pts/2	RW	15:07	0:00	[bash]

- ❖ PID: ID procesa
- ❖ VSZ: Virtuelna veličina procesa (kod + podaci + stek)
- ❖ RSS: Trenutna količina zauzetih MB u RAM-u za proces
- ❖ TTY: Terminal
- ❖ STAT: Status: R (*Runnable*), S (*Sleep*), W (*paging*), Z (*Zombie*), ...



# Nadgledanje aktivnosti procesa

## ❖ top

❖ Prikazuje najvažnije procese, sortirane po potrošnji procesorskog vremena

❖ top - 15:44:33 up 1:11, 5 users, load average: 0.98, 0.61, 0.59

Tasks: 81 total, 5 running, 76 sleeping, 0 stopped, 0 zombie

Cpu(s): 92.7% us, 5.3% sy, 0.0% ni, 0.0% id, 1.7% wa, 0.3% hi, 0.0% si

Mem: 515344k total, 512384k used, 2960k free, 20464k buffers

Swap: 1044184k total, 0k used, 1044184k free, 277660k cached

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3809	jdoe	25	0	6256	3932	1312	R	93.8	0.8	0:21.49	bunzip2
2769	root	16	0	157m	80m	90m	R	2.7	16.0	5:21.01	X
3006	jdoe	15	0	30928	15m	27m	S	0.3	3.0	0:22.40	kdeinit
3008	jdoe	16	0	5624	892	4468	S	0.3	0.2	0:06.59	autorun
3034	jdoe	15	0	26764	12m	24m	S	0.3	2.5	0:12.68	kscd
3810	jdoe	16	0	2892	916	1620	R	0.3	0.2	0:00.06	top

❖ Moguće je promeniti redosled sortiranja

❖ M: Sortiranje po memoriji P: CPU T: Vreme

❖ Proces se može prekinuti sa **k** i proces **id**-jem



# Ubijanje procesa (1/2)

## ❖ **kill <pidovi>**

- ❖ Šalje signal za obustavljanje procesu, čeka da proces sačuva podatke i sam završi
- ❖ Uvek treba prvo koristiti ovu komandu
- ❖ Primer: **kill 3039 3134 3190 3416**

## ❖ **kill -9 <pidovi>**

- ❖ Šalje signal za trenutni kraj procesu
- ❖ Sam sistem prekida proces
- ❖ Korisno kada je proces stvarno zaglavljen

## ❖ **kill -9 -1**

- ❖ Ubija procese trenutnog korisnika
- ❖ **-1** znači sve procese



## Ubijanje procesa (2/2)

### ❖ **killall [-<signal>] <komanda>**

- ❖ Ubija sve zadatke koji izvršavaju komandu **<komanda>**
- ❖ Primer:
  - ❖ **killall bash**

### ❖ **xkill**

- ❖ Dопушта кориснику да убие графичку апликацију кликом мијем на њу
- ❖ Корисно када име апликације није познато



# Oporavak od zaglavljenе grafike

- ❖ Ukoliko je grafička sesija zaglavljena i nije moguće koristiti terminal nema potrebe za restartovanjem mašine
- ❖ Sistem je vrlo verovatno u konzistentnom stanju
  - ❖ Pristup tekstualnoj konzoli **[Ctrl][Alt][F1]** (ili **[F2], [F3]**)
  - ❖ Iz tekstualne konzole je moguće ubiti aplikaciju koja uzrokuje problem
  - ❖ Povratak u grafičku sesiju **[Ctrl][Alt][F5]** ili **[Ctrl][Alt][F7]**
  - ❖ Ukoliko ne može da se identificuje proces koji pravi problem mogu se ubiti svi procesi sa **kill -9 -1**
    - ❖ Povratak na login



# Sekvence komande

- ❖ Moguće je uneti sledeću komandu pre nego što je prethodna završena
- ❖ Moguće je razdvajati komande sa ;
  - ❖ **echo „neki tekst“; sleep 10; echo „nastavak teksta“**
- ❖ Uslovna izvršavanja komandi
  - ❖ koriste se **&&** i **|||**
- ❖ **&&**: pokreće drugu komandu samo ako je prva bila uspešna
- ❖ **|||**: pokreće drugu komandu samo ako je prva bila neuspešna



# Navodnici (1/2)

- ❖ Dupli navodnici ("") služe da:
  - ❖ spreče šel da interpretira razmake kao separatore argumenata
  - ❖ spreče razvijanje šablonu u imenima datoteka
- ❖ **echo "Hello World"**  
Hello World
- ❖ **echo "You are logged as \$USER"**  
You are logged as bgates
- ❖ **echo \*.log**  
find\_prince\_charming.log cosmetic\_buys.log
- ❖ **echo "\* .log"**  
\*.log



## Navodnici (2/2)

- ❖ Jednostruki navodnici (`) služe da:
  - ❖ spreče šel da interpretira razmake kao separatore argumenta
  - ❖ spreče razvijanje šablona u imenima datoteka
  - ❖ spreče razvijanje varijabli

### **❖ echo 'You are logged as \$USER'**

You are logged as \$USER

- ❖ (`) služe da:
  - ❖ pozovu drugu komandu iz komande
  - ❖ mogu se koristiti unutar duplih navodnika

### **❖ cd /lib/modules/`uname --r`; pwd**

/lib/modules/2.6.9-1.6\_FC2

### **❖ echo "You are using Linux `uname --r`"**

You are using Linux 2.6.9-1.6\_FC2



# Merenje proteklog vremena

❖ **time <zadatak>**

❖ izlaz:

- ❖ real 0m2.304s (stvarno proteklo vreme)
- ❖ user 0m0.449s (vreme izvršavanja koda na CPU)
- ❖ sys 0m0.106s (vreme izvršavanja sistemskih poziva na CPU)

- ❖  $\text{real} = \text{user} + \text{sys} + \text{waiting}$
- ❖  $\text{waiting} = \text{I/O čekanje} + \text{vreme mirovanja}$  (izvršavanje drugih zadataka)



# Varijable okruženja

- ❖ Šelovi dopuštaju korisnicima da definišu varijable
  - ❖ mogu da se koriste u šel komandama
  - ❖ Konvencija: imena sastavljena od malih slova
- ❖ Moguće je definisati i varijable okruženja
  - ❖ vidljive i u skriptama i programima pokrenutim iz šela
  - ❖ Konvencija: imena sastavljena od velikih slova
- ❖ **env**
  - ❖ Izlistava sve definisane varijable okruženja i njihove vrednosti



# Primeri šel varijabli

- ❖ Šel varijable (bash)
  - ❖ **projdir=/home/rt-rk/projekat**  
**ls -la \$projdir; cd \$projdir**
- ❖ Varijable okruženja (bash)
  - ❖ **cd \$HOME**
  - ❖ **export DEBUG=1**  
.program\_za\_debagovanje (ispisuje debag ispise ako je varijabla **DEBUG** postavljena na **1**)



# Glavne standardne varijable okruženja (1/2)



- ❖ Koriste ih mnoge aplikacije
- ❖ LD\_LIBRARY\_PATH
  - ❖ putanja do deljenih biblioteka
- ❖ DISPLAY
  - ❖ id displeja za grafičke aplikacije
- ❖ EDITOR
  - ❖ podrazumevani editor
- ❖ HOME
  - ❖ Trenutni korisnički direktorijum
- ❖ HOSTNAME
  - ❖ naziv lokalne mašine



# Glavne standardne varijable okruženja (2/2)



- ❖ MANPATH
  - ❖ putanja do priručnika
- ❖ PATH
  - ❖ putanja do programa (komandi)
- ❖ PRINTER
  - ❖ podrazumevani štampač
- ❖ SHELL
  - ❖ Naziv trenutnog šela
- ❖ TERM
  - ❖ tip trenutnog terminala
- ❖ USER
  - ❖ naziv trenutnog korisnika



# PATH varijabla okruženja

## ❖ PATH

- ❖ definiše redosled potrage za komandama

**/home/acox/bin:/usr/local/bin:/usr/kerberos/bin  
:/usr/bin:/bin:/usr/X11R6/bin:/bin:/usr/bin**

## ❖ LD\_LIBRARY\_PATH

- ❖ Definiše redosled potrage za deljenim bibliotekama za linker

**/usr/local/lib:/usr/lib:/lib:/usr/X11R6/lib**

## ❖ MANPATH

- ❖ Definiše redosled potrage za priručnikom (detalji o komandama)

**/usr/local/man:/usr/share/man**



# Upozorenje za korišćenje PATH varijable



- ❖ Preporučuje se da . ne bude eksportovana u PATH varijablu, ili bar da nije na početku
  - ❖ štetni program može da se zove kao neka standardna komanda i pokušajem da se izvrši komanda pokrenuće se program
  - ❖ ukoliko postoji izvršna datoteka sa nazivom **test**, ona će se pozvati umesto standardne **test** komande i neke skripte neće više raditi
  - ❖ posle svake **cd** komande, šel će trošiti vreme na ažuriranje liste dostupnih komandi
- ❖ Lokalne komande treba pozivati sa **./<komanda>**



# Alijasi

- ❖ Šel dozvoljava definisanje alijsa za komande
  - ❖ prečice za često korišćene komande
- ❖ Primeri
  - ❖ **alias ls='ls -la'**
    - ❖ komanda se uvek pokreće sa podrazumevanim parametrima
  - ❖ **alias rm='rm -I'**
    - ❖ rm će uvek tražiti potvrdu
  - ❖ **alias frd='find\_rambaldi\_device --asap --risky'**
    - ❖ korisno za zamenu dugih i često korišćenih komandi
  - ❖ **alias rtrk='./home/rtrk/env/rtrk.env'**
    - ❖ Korisno za brzo podešavanje okruženja
    - ❖ . je šel komanda za izvršavanje sadržaja skripte
      - ❖ Alternativa: **source**



# which komanda

❖ **which** pokazuje gde se nalazi komanda

❖ **bash> which ls**

alias ls='ls --color=tty' /bin/ls

❖ **tcs> which ls:**

aliased to ls --color=tty

❖ **bash> which alias**

/usr/bin/which: no alias in  
(/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin)

❖ **tcs> which alias**

alias: shell built-in command.



# ~/.bashrc datoteka

## ❖ ~/.bashrc

- ❖ šel skripta koja se učitva svaki put kad se šel startuje
- ❖ Može se koristiti za definisanje
  - ❖ podrazumevanih vrednosti varijabli okruženja (PATH, EDITOR,...)
  - ❖ alijasa
  - ❖ prompta (više informacija u bash manualu)
  - ❖ pozdravne poruke



# Menjanje komandi

- ❖ Strelicama levo i desno se može kretati kroz trenutnu komandu
- ❖ [Ctrl][a] ili [home] za pozicioniranje na početku linije
- ❖ [Ctrl][e] ili [end] za pozicioniranje na kraj linije
- ❖ Strelicama gore i dole se može kretati kroz prethodne komande
- ❖ [Ctrl][r] rekurzivna potraga kroz istoriju komandi



# Istoriја команди

## ❖ history

- ❖ Prikazuje poslednje pokrenute komande i njihov broj
- ❖ Moguće je kopirati komande

## ❖ !!

- ❖ Ponovo poziva poslednju komandu

## ❖ !1003

- ❖ Ponovno pozivanje komande na osnovu njenog broja

## ❖ !cat

- ❖ Ponovno pozivanje komande koja počinje sa cat

## ❖ ^more^less

- ❖ Izmene u okviru poslednje komande (zameni more sa less)

## ❖ more !\*

- ❖ Pozivanje druge komande sa istim argumentima



Tekstualni editori

# UVOD U LINUKS



# Tekstualni editori

## ❖ Grafički tekstualni editori

- ❖ **nedit**
- ❖ **Emacs, Xemacs**
- ❖ **Gedit, Kate**

## ❖ Čisto tekstualni editori

- ❖ **vi, vim**
- ❖ **nano**



# Nedit textualni editor

- ❖ Najznačajnije funkcionalnosti
  - ❖ Lako označavanje i pomeranje teksta
  - ❖ Bojenje sintakse za mnoge programske jezike
  - ❖ Moguće je menjati pravila bojenja sintakse kako bi se prilagodila korisničkim datotekama (npr. log datoteke)
  - ❖ Lako se konfiguriše kroz menije
- ❖ Nije preinstaliran na svim distribucijama



# Primer nedit tekstu editora

Makefile - /data/mike/handhelds/stock\_kernel/linux-2.6.8.1/arch/arm/

```
# arch/arm/Makefile
#
# This file is subject to the terms and conditions of the GNU General Public
# License. See the file "COPYING" in the main directory of this archive
# for more details.
#
# Copyright (C) 1995-2001 by Russell King

LDFLAGS_vmlinux := -p --no-undefined -X
LDFLAGS_BLOB := -format binary
AFLAGS_vmlinux.lds.o = -DTEXTADDR=$(TEXTADDR) -DDATAADDR=$(DATAADDR)
OBJCOPYFLAGS := -O binary -R .note -R .comment -S
GZFLAGS := -9
#CFLAGS += -pipe

ifeq ($(CONFIG_FRAME_POINTER),y)
CFLAGS += -fno-omit-frame-pointer -mcpu=arm7 -mno-sched-prolog
endif

ifeq ($(CONFIG_CPU_BIG_ENDIAN),y)
CFLAGS += -mbig-endian
AS += -EB
LD += -EB
AFLAGS += -mbig-endian
else
CFLAGS += -mlittle-endian
AS += -EL
LD += -EL
AFLAGS += -mlittle-endian
endif

comma = ,

# This selects which instruction set is used.
# Note that GCC does not numerically define an architecture version
# macro, but instead defines a whole series of macros which makes
# testing for a specific architecture or later rather impossible.
```



# Emacs/Xemacs

- ❖ Veoma slični editori
- ❖ Jako moćni editori
- ❖ Odlični za korisnike koji dugo rade u njima
- ❖ Mnogo prečica za komande
- ❖ Mnogo više od editora (igrice, e-mail, šel, pretraživač, ...)
- ❖ Neke prečice moraju da se nauče

```
* linux/arch/arm/mach-pxa/generic.c
*
* Author:      Nicolas Pitre
* Created:    Jun 15, 2001
* Copyright:  MontaVista Software Inc.
*
* Code common to all PXA machines.
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation.
*
* Since this file should be linked before any other machine specific file,
* the _initcall() here will be executed first. This serves as default
* initialization stuff for PXA machines which can be overridden later if
* need be.
*/
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/device.h>
#include <linux/pv.h>

#include <asm/hardware.h>
#include <asm/system.h>
#include <asm/pgtable.h>
#include <asm/mach/map.h>
#include <asm/arch/irqs.h>
#include <asm/arch/udc.h>
#include <asm/arch/pxafb.h>

#include "generic.h"
#include "../drivers/serial/pxa-serial.h"

/*
 * Handy function to set GPIO alternate functions
 */
void pxa_gpio_mode(int gpio_mode)
{
    unsigned long flags;
    int gpio = gpio_mode & GPIO_MD_MASK_NR;
    int fn = (gpio_mode & GPIO_MD_MASK_FN) >> 8;
    int gafr;

    local_irq_save(flags);
    if (gpio_mode & GPIO_MD_MASK_DIR) {
        /* if output and active low, then first set the bit to make it inactive */
        if ((gpio mode & GPIO_ACTIVE_LOW)
-- generic.c                                     (C CVS-1.15 Abbrev)--LI--Top
--- Loading cc-mode... done
```



# Kate i gedit

- # ❖ **Kate** - osnovni editor u kde okruženju

A screenshot of a KDE desktop environment. In the foreground, there's a Konsole terminal window titled 'gedit-xml x' containing a C program. Behind it, a Kate browser window is open, showing a PHP file with syntax highlighting. The Kate window has tabs for 'index.html', 'drupal.css', 'page.tpl.php', and 'style.css'. A file browser sidebar on the left shows a directory structure under 'Filesystem Browser / Documents'. Another gedit window is visible in the background, showing an XML configuration file with several menu entries like 'Viewmenu', 'Searchmenu', and 'Helpmenu'. The desktop background is a light blue gradient.

- # ❖ **Gedit** - osnovni editor u gnome okruženju



vi



- ❖ Čisto textualni editor
- ❖ Napravljen pre nego što se pojavio miš za računar
- ❖ Težak za početnike, veoma dobar za iskusne korisnike
- ❖ Često neophodan u sistemskoj administraciji i namenskim sistemima kada imamo samo terminal na raspolaganju



vim



- ❖ Unapređenje vi editora
- ❖ Mnoge osobine modernih editora
  - ❖ bojenje sintakse
  - ❖ pretraga
  - ❖ neograničeno poništavanje komandi (*undo*)
  - ❖ istorija komandi
  - ❖ ...
- ❖ Može da otvara kompresovane datoteke



# nano



- ❖ još jedan mali čisto tekstualni editor bez podrške za miša
- ❖ Lakši za početnike zbog spiska komandi koje su uvek vidljive na ekranu
- ❖ Dostupan za više platformi
- ❖ Alternativa **vi/vim**-u u namenskim sistemima
- ❖ Nije deo biziboksa (*busybox*)



# Prikaz nano editora



GNU nano 1.2.3

File: fortune.txt

The herd instinct among economists makes sheep look like independent thinkers.

Klingon phaser attack from front!!!!  
100% Damage to life support!!!

Spock: The odds of surviving another attack are 13562190123 to 1, Captain.

Quantum Mechanics is God's version of "Trust me."

I'm a soldier, not a diplomat. I can only tell the truth.  
-- Kirk, "Errand of Mercy", stardate 3198.9

Did you hear that there's a group of South American Indians that worship  
the number zero?

Is nothing sacred?

They are called computers simply because computation is the only significant  
job that has so far been given to them.

As far as the laws of mathematics refer to reality, they are not  
certain, and as far as they are certain, they do not refer to reality.  
-- Albert Einstein

Tact, n.:

The unsaid part of what you're thinking.

Support bacteria -- it's the only culture some people have!

**^G** Get Help **^O** WriteOut **^R** Read File **^Y** Prev Page **^K** Cut Text **^C** Cur Pos  
**^X** Exit **^J** Justify **^W** Where Is **^V** Next Page **^U** UnCut Txt **^T** To Spell



Poređenje datoteka i direktorijuma

# UVOD U LINUKS



# Poređenje datoteka i direktorijuma

- ❖ **diff dat1 dat2**
  - ❖ Prijavljuje razlike između dve datoteke ili ništa ukoliko su datoteke identične
- ❖ **diff -r dir1/ dir2/**
  - ❖ Prijavljuje sve razlike između datoteka sa istim imenom u okviru navedenih direktorijuma
- ❖ Rezultat diff komande se uglavnom smešta u datoteku i kasnije koristi u kombinaciji sa patch komandom
- ❖ Za bolji pregled razlika do detalja lakše je koristiti grafičke alate



# Grafički alati za poređenje

- ❖ Velik broj alata
  - ❖ tkdiff
  - ❖ kompare
  - ❖ gvimdiff
  - ❖ meld
  - ❖ beyond compare
  - ❖ hexdiff - za binarne datoteke
  - ❖ hexcompare - za binarne datoteke
  - ❖ .....



Razne komande

# UVOD U LINUKS



# Informacije o korisniku

## ❖ **who**

- ❖ Izlistava sve korisnike trenutno na sistemu

## ❖ **whoami**

- ❖ Pokazuje kako je trenutni korisnik prijavljen

## ❖ **groups**

- ❖ Pokazuje kojoj grupi pripada trenutni korisnik

## ❖ **groups <korisnik>**

- ❖ Pokazuje kojoj grupi pripada korisnik **<korisnik>**

## ❖ **finger <korisnik>**

- ❖ Daje više detalja o korisniku **<korisnik>** (pravo ime, i sl.)
  - ❖ Onemogućeno na nekim sistemima iz bezbednosnih razloga



# Promena korisnika

- ❖ Nije potrebno odjavljivati se i ponovo se prijavljivati kao drugi korisnik
- ❖ **su korisnik1**
  - ❖ Promena na nalog **korisnik1**, ali okruženje ostaje podešeno kao za originalnog korisnika
- ❖ **su - korisnik2**
  - ❖ Promena na nalog **korisnik2** sa njegovim podešavanjima okruženja
- ❖ **SU -**
  - ❖ Bez argumenata komanda su prebacuje na korenski nalog



# wget komanda

- ❖ Umesto skidanja datoteka iz pretraživača dovoljno je samo kopirati URL u terminal i skinuti datoteku sa wget komandom
- ❖ **wget** - osnovne osobine
  - ❖ http i ftp podrška
  - ❖ Može da nastavi prekinuta preuzimanja
  - ❖ Može da preuzme čitave sajto ili barem da proveri ispravnost linkova
  - ❖ Veoma korisna komanda u skriptama ili kada nam grafika nije dostupna (sistem administrator, namenska platforma)
  - ❖ Podrška za proksi (koristi http\_proxy i ftp\_proxy varijable okruženja)



# wget primeri

- ❖ **wget --c \**  
**<http://microsoft.com/customers/dogs/winxp4dogs.zip>**
  - ❖ Nastavlja prekinuto preuzimanje
- ❖ **wget --m <http://lwn.net/>**
  - ❖ Pravi ogledalo (*mirror*) sajta
- ❖ **wget --r --np <http://www.xml.com/ldd/chapter/book/>**
  - ❖ Rekurzivno skida knjigu za pristup bez mreže
  - ❖ **--np:** (*no-parent*) prati samo linkove iz trenutnog direktorijuma



# Razne komande (1/2)

## ❖ sleep 60

- ❖ uspavljuje proces na 60 sekundi
- ❖ ne troši sistemske resurse

## ❖ wc izveštaj.txt (word count)

- ❖ 438 2115 18302 izveštaj.txt
- ❖ Prikazuje broj linija, reči i karaktera u datoj datoteci ili na standardnom ulazu



## Razne komande (2/2)

### ❖ **bc** (*basic calculator*)

- ❖ bc je kalkulator sa punom funkcionalnošću
- ❖ obuhvata i programski jezik
- ❖ -l parametar uključuje podršku pokretnog zareza

### ❖ **date**

- ❖ Vraća trenutni datum
- ❖ Korisno u skriptama da se zabeleži datum pokretanja ili završetka neke komande



# Komande kontrolne sume

- ❖ Kontrolna suma ili heš suma je podatak fiksne veličine izračunat na osnovu bloka digitalnih podataka i služi da proveri da li je došlo do greške u podacima, pogotovo nakon prenosa
- ❖ MD5 algoritam je implementiran `md5sum` komandom
- ❖ SHA algoritam je implementiran `shaXsum` komandama (`sha1sum`, `sha256sum`, ...)
- ❖ Provera ispravnosti datoteke se pokreće sa `-c` parametrom i datotekom sa izračunatim sumama kao ulaznim argumentom



Sistem administracije

# UVOD U LINUKS



# Konfiguracija mreže (1/2)

- ❖ **ifconfig -a**
  - ❖ Ispisuje detalje o svim mrežnim spregama dostupnim u sistemu
- ❖ **ifconfig eth0**
  - ❖ Ispisuje detalje o **eth0** sprezi
- ❖ **ifconfig eth0 192.168.0.100**
  - ❖ Dodeljuje IP adresu **192.168.0.100** **eth0** sprezi (1 IP adresa po sprezi)
- ❖ **ifconfig eth0 down**
  - ❖ Gasi **eth0** spregu (oslobađa njenu IP adresu)



# Konfiguracija mreže (2/2)

- ❖ **route add default gw 192.168.0.1**
  - ❖ Podešava podrazumevanu rutu za pakete van lokalne mreže
- ❖ **route -n**
  - ❖ Izlistava postojeće rute
    - ❖ **-n**: odmah prikaži IP adresu umesto pokušaja da se pronađu njihova domenska imena
- ❖ **route del default**
  - ❖ Briše podrazumevanu rutu
- ❖ **route del <IP>**
  - ❖ Briše datu rutu
  - ❖ Korisno da se redefiniše ruta



# Konfiguracija mreže - pregled

- ❖ Samo za jednostavne slučajeve sa 1 spregom, bez DHCP poslužioca, moguće je:
  - ❖ Ostavriti konekciju na mrežu (kablovska, bežična)
  - ❖ Pronaći svoju mrežnu spregu
    - ❖ **ifconfig -a**
  - ❖ Dodeliti IP adresu svojoj spregi (**eth0** u ovom primeru)
    - ❖ **ifconfig eth0 192.168.0.100**
  - ❖ Dodati rutu do kapije (*gateway*)(**192.168.0.1** u ovom primeru)
    - ❖ **route add default gw 192.168.0.1**



# Razrešivanje imena

- ❖ Programi moraju da znaju koja IP adresa odgovara datom imenu (npr. **kernel.org**)
- ❖ Poslužilac domenskih imena (**DNS**) vodi računa o ovome
- ❖ Potrebno je navesti IP adresu jednog ili više **DNS** poslužilaca u datoteci **/etc/resolv.conf**
  - ❖ **nameserver 192.168.231.10**
  - ❖ **nameserver 192.168.231.11**
- ❖ Promene odmah postaju aktivne



# Testiranje mreže

❖ ping freashmeat.net  
ping 192.168.1.1

- ❖ Pokušava da pošalje pakete datoj mašini i da dobije pakete potvrde prijema za uzvrat
- ❖ Isprobati ping komandu sa adresom kapije
  - ❖ Ovo će potvrditi da mrežni adapter radi ispravno
- ❖ Isprobati ping komandu sa DNS poslužiocem
  - ❖ Ovo će potvrditi da je kapija dobro konfigurisana
- ❖ Isprobati ping komandu sa bilo kojom mašinom navodeći njeni ime
  - ❖ Ovo će potvrditi da je DNS poslužilac ispravno konfigurisan



# Pravljenje sistema datoteka

- ❖ Primeri:
  - ❖ **mkfs.ext2 /dev/sda1**
    - ❖ Formatira **/dev/sda1** particiju (**USB, HDD**) u **ext2** format
  - ❖ **mkfs.ext2 -F disk.img**
    - ❖ Formatira sliku diska u **ext2** format
    - ❖ **-F**: (*force*) izvrši čak i ako u pitanju nije pravi uređaj
  - ❖ **mkfs.vfat -v -F 32 /dev/sda1**
    - ❖ Formatira **/dev/sda1** particiju u **FAT32** format
    - ❖ **-v**: (*verbose*)
  - ❖ **mkfs.vfat -v -F 32 disk.img**
    - ❖ Formatira sliku diska u **FAT32** format
- ❖ Prazna slika diska se može napraviti sa komandom (**64MB** veličina datoteke)
  - ❖ **dd if=/dev/zero of=disk.img bs=1M count=64**



# Mauntovanje uređaja (1/3)

- ❖ Da bi sistemi datoteka na bilo kom uređaju (internom ili eksternom) bili vidljivi u sistemu moraju da se mauntuju
- ❖ Prvi put treba napraviti direktorijum na koji će se mauntovati sistem datoteka
  - ❖ **mkdir /mnt/usbdisk** (primer)
- ❖ Zatim mauntovati sistem datoteka
  - ❖ **mount -t vfat /dev/sda1 /mnt/usbdisk**
    - ❖ **/dev/sda1**: fizički uređaj
    - ❖ **-t**: određuje tip sistema datoteka (**ext2, ext3, vfat, reiserfs, iso9660...**)



## Mauntovanje uređaja (2/3)

- ❖ mount komanda ima mnogo opcija - pogledati priručnik za detalje
- ❖ Opcije za mauntovanje svakog uređaja se mogu čuvati u datoteci  
**/etc/fstab**
  - ❖ Ukoliko je ova datoteka popunjena potrebno je samo navesti odredište mauntovanja
  - ❖ # /etc/fstab: static file system information.

```
# <file system> <mount point> <type> <options> <dump> <pass>
proc          /proc        proc      defaults    0      0
/dev/hda3     /           ext3      defaults,errors=remount-ro 0      1
/dev/hda4     /home       ext3      defaults    0      2
/dev/hda2     /root2      ext3      defaults    0      2
/dev/hda1     none        swap      sw         0      0
/dev/hdc      /media/cdrom0 udf,iso9660 user,noauto 0      0
```
- ❖ Primeri mount komande sa **/etc/fstab** datotekom
  - ❖ **mount /proc**
  - ❖ **mount /media/cdromo**



## Mauntovanje uređaja (3/3)

- ❖ Mogu se mauntovati i sistemi datoteka koji se nalaze u regularnim datotekama (**loop** uređaji)
- ❖ Korisno za razvoj sistema datoteka za drugu platformu
- ❖ Korisno za pristupanje sadržaju **ISO** slike cd-a bez potrebe za rezanjem
- ❖ Korisno za čuvanje **Linuks** sisetma datoteka u datoteci na **Windows** particiji
- ❖ **cp /dev/sda1 usbkey.img**  
**mount -o loop -t vfat usbkey.img /mnt/usbdisk**



# Izlistavnjje mauntovanih sistema datoteka



- ❖ Koristi se mount komanda bez parametara
  - ❖ /dev/hda6 on / type ext3 (rw,noatime)  
none on /proc type proc (rw,noatime)  
none on /sys type sysfs (rw)  
none on /dev/pts type devpts (rw,gid=5,mode=620)  
usbfs on /proc/bus/usb type usbfs (rw)  
/dev/hda4 on /data type ext3 (rw,noatime)  
none on /dev/shm type tmpfs (rw)  
/dev/hda1 on /win type vfat (rw,uid=501,gid=501)  
none on /proc/sys/fs/binfmt\_misc type binfmt\_misc (rw)



# Odmauntovanje uređaja

## ❖ **umount /mnt/usbdisk**

- ❖ Izvršava sve zakazane upise i odmauntaže datu uređaj koji se zatim može ukloniti bezbedno
- ❖ Da bi odmauntovanje bilo moguće moraju se zatvoriti sve otvorene datoteke sa uređaja
- ❖ Zatvoriti aplikacije koje koriste podatke sa mautovanih particija
- ❖ Proveriti da ni jedan šel nije pozicioniran u mauntovanom direktorijumu
- ❖ Komande **fuser -mv <odredište>** i **lsof <odredište>** (*list open files*) pokazuju koji procesi koriste datoteke sa **<odredišta>**
- ❖ Opcijom **k** se mogu ubiti ovi procesi



# Paketi

- ❖ Sistem distribucije programa je drugačiji na **GNU/Linuksu** u odnosu na **Windows**
- ❖ U Linuksu se za instaliranje, unapređenje i brisanje aplikacija i biblioteka koriste paketi
- ❖ Paketi sadrže datoteke vezane za aplikaciju ili biblioteku i dodatne meta informacije kao što su verzija i zavisnosti
  - ❖ **.deb** za **Debian** i **Ubuntu**, **.rpm** za **Mandriva**, **Fedora**, **OpenSUSE**
- ❖ Paketi se nalaze u repozitorijuma na **HTTP** ili **FTP** poslužiocima
- ❖ Preporuka je da se koriste samo paketi sa zvaničnih repozitorijuma ukoliko je moguće



# Upravljanje paketima (1/2)

- ❖ Instrukcije za **GNU/Linuks** sisteme bazirane na **Debian** (**Debian, Ubuntu...**)
- ❖ Repozitorijumi sa paketima se navode u datoteci **/etc/apt/sources.list**
- ❖ Osvežavanje liste repozitorijuma
  - ❖ **sudo apt-get update**
- ❖ Ime paketa može da se pronađe korišćenjem pretraživača na <http://packages.debian.org> ili <http://packages.ubuntu.com>
- ❖ Može se koristiti i **apt-cache search <ključna\_reč>** komanda



# Upravljanje paketima (2/2)

- ❖ Instaliranje datog paketa
  - ❖ **`sudo apt-get install <paket>`**
- ❖ Uklanjanje paketa
  - ❖ **`sudo apt-get remove <paket>`**
- ❖ Osvežavanje svih instaliranih paketa sa inteligentnim rukovanjem zavisnostima između novih paketa
  - ❖ **`sudo apt-get dist-upgrade`**
- ❖ Prikaz informacija o paketu
  - ❖ **`sudo apt-cache show <paket>`**



SSH

# UVOD U LINUKS



## Uvod

- ❖ **SSH** - Secure Shell
- ❖ **SSH** je siguran komunikacijski protokol koji omogućava udaljeno prijavljivanje na sistem, transfer datoteka i tunelovanje prolaza
  - ❖ Normalizovan je od strane **RFC 4251, 4252, 4253 i 4254**
- ❖ Zamena za **telnet, rlogin, rsh**, itd.
- ❖ Glavna implementacija na Linuksu je **OpenSSH** koja sadrži i stranu korisnika i poslužioca
- ❖ Manja implementacija namenjena namenskim sistemima je **Dropbear**
- ❖ Na Windowsu često korišćeni korisnik je **Putty**



# Instalacija i osnovno korišćenje



- ❖ **OpenSSH** je paket koji je dostupan u svim **GNU/Linuks** distribucijama
- ❖ Na **Ubuntu** distribuciji dostupna su dva paketa
  - ❖ **openssh-client**: program korisnik
  - ❖ **openssh-server**: program poslužilac
- ❖ Povezivanje na poslužioca je jednostavno
  - ❖ **ssh korisnik@poslužilac**
  - ❖ **ssh** će tražiti korisnikovu šifru i po unosu se ulogovati na udaljeni sistem



# Prenos datoteka i grafike

- ❖ Datoteke se mogu prenositi korišćenjem programa **scp**
  - ❖ **scp dat1 dat2  
korisnik@poslužilac:/<odredišni\_direktorijum>**
  - ❖ **scp -r <direktorijum>  
korisnik@poslužilac:/<odredište>**
- ❖ Sa **-X** parametrom uključuje se prosleđivanje **X11**
  - ❖ Omogućava prikaz grafičkih aplikacija izvršavanih na udaljenoj mašini na lokalnoj mašini
  - ❖ Potrebno je omogućiti prosleđivanje **X11** na poslužiocu u datoteci **/etc/ssh/sshd\_config**



# Udaljeno izvršavanje komandi

- ❖ Umesto prijavljivanja na udaljeni sistem mogu se i izvršavati komande na njemu
  - ❖ **ssh korisnik@poslužilac ls**
  - ❖ Korisno u šel skriptama
- ❖ Drugi programi koriste ssh kao transportni sloj
  - ❖ alat za sinhronizaciju **rsync** može da radi preko **ssh**
    - ❖ **rsync -e ssh ~ /dir korisnik@poslužilac:~ /dir**
  - ❖ Alati za kontrolu verzija (**CVS, SVN**) mogu da rade preko **ssh**



Razvoj aplikacija

# UVOD U LINUKS



# Prevođenje jednostavne aplikacije



- ❖ Prevodilac koji se koristi na svim Linuks sistemima je **GCC**
  - ❖ <http://gcc.gnu.org>
- ❖ Prevođenje jedne jednostavne aplikacije
  - ❖ **gcc -o test test.c**
    - ❖ Genriše izvršnu binarnu datoteku **test** na osnovu koda iz **test.c**
  - ❖ **g++ -o test test.cpp**
    - ❖ za **C++**
    - ❖ **-Wall** opcija uključuje dodatna upozorenja pri prevođenju
- ❖ Prevođenje izvornih datoteka i povezivanje u aplikaciju
  - ❖ **gcc -c test1.c**
  - ❖ **gcc -c test2.c**
  - ❖ **gcc -o test test1.o test2.o**
- ❖ **gcc** automatski poziva povezivač **ld**



# Korišćenje biblioteka (1/2)

- ❖ Na svakom Linuks sistemu je prisutna C biblioteka koja nudi veliki broj API-ja za razvoj aplikacije
  - ❖ <http://www.gnu.org/software/libc/manual/>
- ❖ Pored C biblioteke, dostupne su i mnoge druge biblioteke za rad sa grafikom, multimedijom, mrežama i slično
- ❖ Većina biblioteka je već dostupna u distribuciji u okviru dva paketa:
  - ❖ **libfoo**
    - ❖ sama biblioteka potrebna za izvršavanje već prevedenih aplikacija, ali nedovoljna za razvoj novih
  - ❖ **libfoo-dev**
    - ❖ paket koji sadrži zaglavlja i konfiguracije potrebne za razvoj novih aplikacija



## Korišćenje biblioteka (2/2)

- ❖ U okviru koda potrebno je uvući zaglavje biblioteke
  - ❖ Obično **#include <foo.h>** ili **#include <foo/foo.h>**
  - ❖ zaglavlja se nalaze u **/usr/include/**
- ❖ Za prevodenje aplikacije sa bibliotekom najlakše je da se koristi **pkg-config** koji je podržan od većine biblioteka
  - ❖ **gcc -o test test.c \$(pkg-config --cflags -libs)**
- ❖ Podrazumevano je dinamičko povezivanje sa bibliotekom
  - ❖ biblioteka mora da se nalazi u **/lib** pri pokretanju aplikacije
  - ❖ **ldd** komanda pokazuje koje biblioteke su potrebne za određenu aplikaciju



## make alat

- ❖ Proces prevodenja se automatizuje korišćenjem **make** alata
- ❖ make čita datoteku sa nazivom **Makefile** i izvršava pravila napisana u njoj
- ❖ svako pravilo mora biti određenog formata
- ❖ pozivanje **make** bez parametara poziva pravilo **all**



# Bild sistemi

- ❖ **make** olakšava prevođenje ali ne može lako da se adaptira na različita podešavanja okruženja i različite opcije pri prevođenju
- ❖ Razvijeniji bild sistemi su:
  - ❖ **Autotools** (**automake**, **autoconf**) - baziran na **make**-u i šel skriptama - jedan od najpopularnijih alata
  - ❖ **Buildroot**
  - ❖ **Cmake**
  - ❖ **Scons**
- ❖ Tipičan redosled
  - ❖ konfiguriši
  - ❖ **make**
  - ❖ **make install**



# Debagovanje

- ❖ Oficijalni debager koji dolazi sa GNU distribucijama je **gdb**
- ❖ Aplikacija mora da se prevede sa **-g** opcijom
  - ❖ dodaje debag simbole u izvršnu datoteku
- gcc --o test test.c --g**
- ❖ Aplikacija se može pokrenuti u debageru
  - ❖ **gdb test**
- ❖ Debager se može zakačiti za pokrenutu aplikaciju
  - ❖ **gdb test -p PID**
    - ❖ **PID** - id proces pokrenute aplikacije



# Korišćenje gdb

- ❖ gdb je tekst-bazirani debager sa korisničkom spregom u vidu komandne linije, poput šela, i kroz njega omogućava namenske komande.
- ❖ Najbitnije komande su:
  - ❖ break (b) za postavljanje tačke prekida (breakpoint) u kodu. Može se vezati za ime funkcije ili postaviti na neko mesto u izvornom kodu ili vezati za absolutnu memorijsku adresu.
  - ❖ print (p) za prikaz vrednosti promenljive. Vezuje se za ime promenljive, čak i ako je promenljiva kompleksna (što uključuje dereferenciranje strukture i sl.).
  - ❖ (c) za nastavak izvršenja do sledeće tačke prekida.
  - ❖ next (n) za izvršavanje samo naredne linije koda (preskače bilo koji poziv funkcije tj. izvršava je bez zaustavljanja u istoj) i step (s) za izvršavanje samo naredne linije koda (ulazi u funkciju i zaustavlje se u njoj)
  - ❖ backtrace (bt) za prikaz steka poziva funkcija.



# Primer korišćenja gdb



```
thomas@surf:/tmp$ gcc -o test test.c -g
thomas@surf:/tmp$ gdb test
GNU gdb 6.8-debian
[...)
(gdb) break foo
Breakpoint 1 at 0x80483c7: file test.c, line 5.
(gdb) run
Starting program: /tmp/test2

Breakpoint 1, foo (a=2, b=3) at test.c:5
5          return a + b;
(gdb) p a
$1 = 2
(gdb) p b
$2 = 3
(gdb) c
Continuing.
foo=5

Program exited normally.
```



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u namenskim sistemima



Uvod u linuks u namenskim sistemima

# **LINUKS U NAMENSKIM SISTEMIMA**



# Nastanak besplatnog softvera

- ❖ 1983, Richard Stallman, GNU projekat i koncept besplatnog softvera. Početak razvoja gcc, gdb, glibc i drugih važnih alata
- ❖ 1991, Linus Torvalds, Linuks kernel projekat, kernel Unix-like operativnog sistema. Zajedno sa GNU softverom i mnogim drugim komponentama otvorenog koda: potpuno besplatan operativni sistem, GNU/Linuks
- ❖ 1995, Linuks je sve popularniji na sistemima sa poslužiocem
- ❖ 2000, Linuks je sve popularniji u ugrađenim (namenskim) sistemima
- ❖ 2008, Linuks je sve popularniji u prenosivim uređajima
- ❖ 2010, Linuks je sve popularniji na telefonima.



# Besplatan softver?

- ❖ Program se smatra **besplatnim** kada njegova licenca nudi svim korisnicima sledeće **4** mogućnosti:
  - ❖ Slobodu da se softver koristi (pokreće) u bilo koju svrhu
  - ❖ Slobodu da se softver izučava i menja
  - ❖ Slobodu da se redistribuiraju kopije
  - ❖ Slobodu da se distribuiraju izmenjene verzije
- ❖ Ove slobode su date za kako komercijalnu tako i nekomercijalnu upotrebu
- ❖ Podrazumevaju postojanje izvornog koda, te da softver može biti modifikovan i distribuiran mušterijama
- ❖ **Veoma pogodno za namenske sisteme**



# Šta je Linuks u namenskim sistemima?



- ❖ Linuks u namenskim sistemima je upotreba Linuks jezgra (kernela) i različitih komponenti otvorenog koda u namenskim sistemima



Zašto linuks u namenskim sistemima?

Prednosti Linuksa i otvorenog koda u namenskim sistemima

# UVOD U LINUKS U NAMENSKIM SISTEMIMA



# Ponovna upotreba komponenti

- ❖ Glavna prednost Linuksa i otvorenog koda u namenskim sistemima je **mogućnost** ponovnog korišćenja komponenti
- ❖ Ekosistem otvorenog koda već nudi mnoge komponente za uobičajene zahteve, od hardverske podrške do mrežnih protokola, multimedije, grafike, kriptografskih biblioteka, itd.
- ❖ Čim se dovoljno raširi upotreba hardverskog uređaja, protokola ili karakteristike, postoji velika šansa da je komponenta otvorenog koda koja je podržava već dostupna
- ❖ Omogućava brz dizajn i razvoj komplikovanih komponenti.
- ❖ Niko ne bi trebalo da ponovo razvija još jedan operativni sistem, TCP/IP stek, USB stek ili druge biblioteke iz grafičke grupe alata.
- ❖ Dozvoljava fokusiranje novu vrednost produkta.



## Mala cena

- ❖ Besplatan softver može da se umnoži na željenom broju uređaja, potpuno besplatno
- ❖ Ukoliko vaš namenski sistem koristi isključivo besplatan softver, možete svesti i cenu softverskih licenci na 0. Čak su i alati za razvoj besplatni, osim ukoliko odaberete komercijalni Linuks za namenske sisteme
- ❖ Dozvoljava postojanje višeg budžeta za hardver, odnosno povećanje veština i znanja u kompaniji.



# Potpuna kontrola

- ❖ Sa otvorenim kodom, imate izvorni kod svih komponenti u sistemu
- ❖ Dozvoljava neograničene modifikacije, promene, podešavanja, debagovanje, optimizacije, u neograničenom vremenskom periodu
- ❖ Bez zaključavanja ili zavisnosti od “third-party” prodavaca
  - ❖ Da bi se ovo ispoštovalo, komponente koje ne pripadaju otvorenom kodu moraju da se izbegnu prilikom dizajna i razvoja
- ❖ **Dozvoljava punu kontrolu nad softverom u vašem sistemu**



# Kvalitet



- ❖ Mnoge komponente otvorenog koda imaju široku upotrebu, na milionima sistema
- ❖ Često imaju viši kvalitet od komponenti koje se mogu napraviti unutar kompanije, pa čak i od izvornog prodavca
- ❖ Naravno, nisu sve komponente otvorenog koda kvalitetne, ali one sa širom upotrebom jesu.
- ❖ Omogućava dizajniranje vašeg sistema sa visokokvalitetnim komponentama od samog početka



# Olakšava testiranje novih funkcionalnosti



- ❖ S obzirom da je otvoreni kod dosupan besplatno, lako je doći do nekog softvera i isprobati ga
- ❖ Omogućava jednostavno proučavanje više opcija dok pravite neki odabir
- ❖ Mnogo jednostavnije od kupovine i demonstracionih procedura potrebnih za većinu izvornih produkata
- ❖ **Omogućava jednostavni pregled novih mogućnosti i rešenja**



## Podrška u zajednici

- ❖ Programske komponente otvorenog koda razvija zajednica developera i korisnika
- ❖ Ova zajednica nudi podršku visokog kvaliteta: možete direktno da kontaktirate glavne developere komponenti koje koristite. Pritom, verovatnoća da ćete dobiti odgovor ne zavisi od toga za koju kopiju radite.
- ❖ Često je ova podrška bolja od tradicionalne, ali zahteva razumevanje kako zajednica funkcioniše da bi se mogučnosti podrške zajednice adekvatno iskoristile
- ❖ **Omogućava ubrzavanje rešavanja problema kada razvijate vaš sistem**



# Preuzimanje uloge u zajednici

- ❖ Postoji mogućnost preuzimanja uloge u zajednici razvoja za neku komponentu korišćenu u namenskim sistemima: prijava greške, testiranje novih verzija ili svojstva, zakrpa koje popravljaju greške ili dodaju nova svojstva, itd.
- ❖ Najčešće komponente otvorenog koda nisu glavna vrednost proizvoda, već je to interes svakoga da uzvrati doprinos.
- ❖ Za *inženjere* to predstavlja: veoma **motivišući** način da budu prepoznati izvan kompanije, komunikaciju sa drugima u datoј oblasti, **otvaranje novih mogućnosti**, itd.
- ❖ Za *menadžere*: **motivišući faktor** za inženjere, omogućava kompaniji da bude prepoznata u zajednici otvorenog koda i stoga lakše dođe do podrške i da bude **atraktivnija** developerima otvorenog koda.



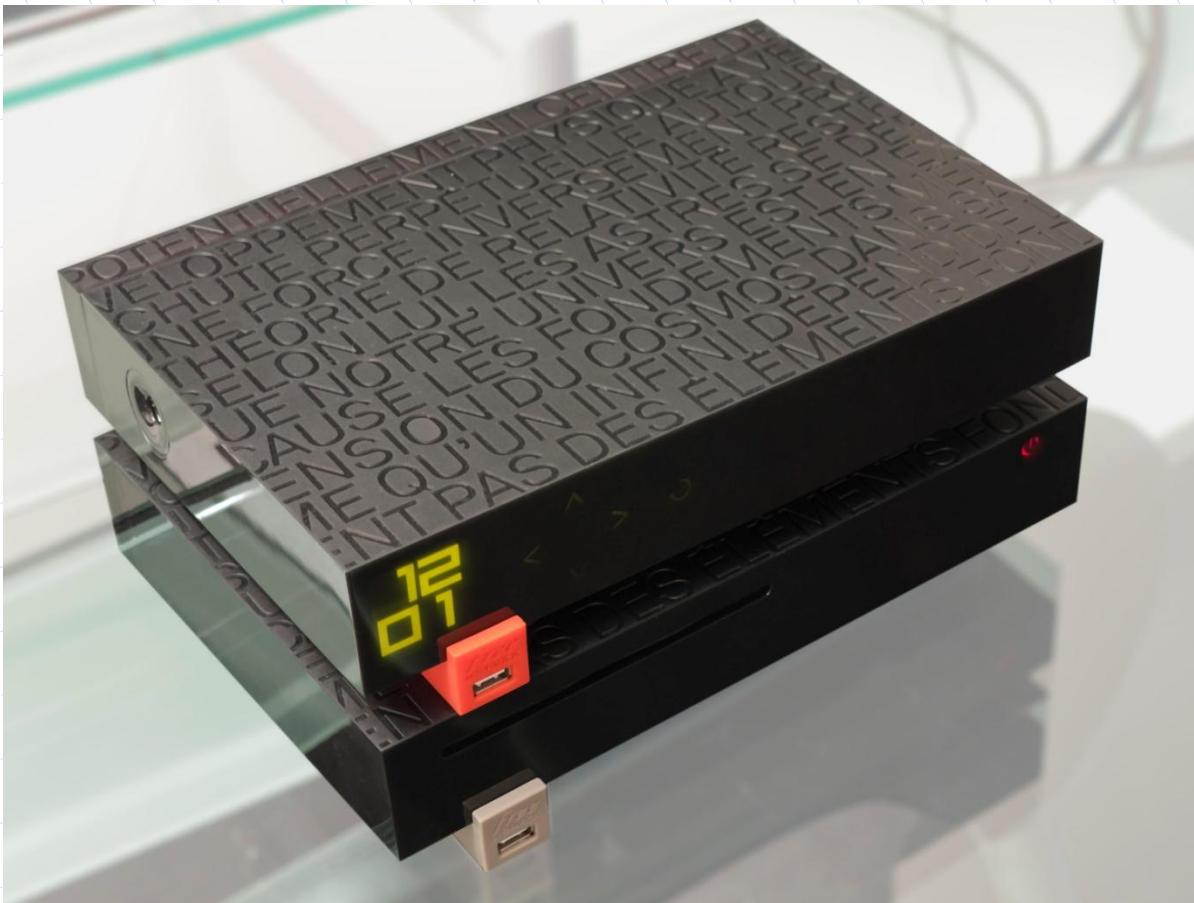
Sistemi sa Linuksom

Nekoliko primera namenskih sistema na kojima se nalazi Linuks

# UVOD U LINUKS U NAMENSKIM SISTEMIMA



# Personalni ruteri





# Televizori





# POS terminali



# Laserska mašina za sečenje



# Mašine za vinogradarstvo





Namenski hardver za Linuks sisteme

# UVOD U LINUKS U NAMENSKIM SISTEMIMA



# Procesori i arhitekture (1/2)

- ❖ Linuks kernel i većina drugih komponenti koje zavise od arhitekture podržavaju širok spektar 32-bitnih i 64-bitnih arhitektura
  - ❖ x86 i x86-64, kako na PC platformama, tako i u namenskim sistemima (multimedijalnim, industrijskim)
  - ❖ ARM, sa stotinama različitih SoC (multimedijalnim, industrijskim)
  - ❖ PowerPC (uglavnom u realnom vremenu, industrijska primena)
  - ❖ MIPS (uglavnom mrežna primena)
  - ❖ SuperH (uglavnom set top box i multimedijalna primena)



## Procesori i arhitekture (2/2)

- ❖ Blackfin (DSP architecture)
- ❖ Microblaze (softverski procesor za Xilinx FPGA)
- ❖ Coldfire, SCore, Tile, Xtensa, Cris, FRV, AVR32, M32R
- ❖ Podržane su arhitekture sa i bez MMU, iako arhitekture bez MMU imaju određena ograničenja
- ❖ Linuks nije napravljen za male mikrokontrolere.
- ❖ Osim alata za prevodenje, bootloader-a i kernela, sve ostale komponente su generalno nezavisne od arhitekture



# Ram i prostor za skladištenje

- ❖ **RAM:** osnovna verzija Linuks sistema može da radi sa svega 8 MB RAM, ali malo realističniji sistem obično zahteva bar 32 MB RAM memorije. Zavisno od tipa i veličine aplikacija.
- ❖ **Prostor za skladištenje:** osnovna verzija Linuks sistema može da radi sa svega 4 MB prostora, ali je obično potrebno više.
  - ❖ Podržan je Flash tip, uključujući NAND i NOR flash, sa specifičnim sistemima datoteka
  - ❖ Podržano je blokovsko skladištenje uključujući SD/MMC kartice i eMMC
- ❖ Nije neophodno biti previše restriktivan po pitanju količine RAM memorije ili memorije za skladištenje: na ovom nivou imamo fleksibilnost da iskoristimo sve dostupne komponente.



# Komunikacija

- ❖ Linuks kernel ima podršku za mnoge ustaljene komunikacione magistrale
  - ❖ I2C
  - ❖ SPI
  - ❖ CAN
  - ❖ 1-wire
  - ❖ SDIO
  - ❖ USB
- ❖ Takođe i punu podršku za mrežu:
  - ❖ Ethernet, Wifi, Bluetooth, CAN, itd.
  - ❖ IPv4, IPv6, TCP, UDP, itd.
  - ❖ Firewall, napredno rutiranje, multikast



# Tipovi hardverskih platformi

- ❖ **Platforme za evaluaciju** od proizvođača SoC-a. Najčešće su skupe, ali poseduju mnoge ugrađene periferije. Generalno su neprikladne za krajnje proizvode.
- ❖ **Komponenta u vidu modula**, pločica samo sa CPU/RAM/flash i još ponekom komponentom i konektorima za pristup svim drugim periferijama. Može se koristiti za pravljenje krajnjih proizvoda u malim ili srednjim serijama.
- ❖ **Platforme za razvoj u zajednici**, popularizuju određeni SoC i čine ga lako dostupnim. Odmah su spremni za upotrebu uz dobru podršku i nisku cenu, ali često imaju manje periferija od platformi za evaluaciju. Donekle mogu biti korištene i za krajnje proizvode.
- ❖ **Platforme po meri**. Šematski prikazi i platforme za evaluaciju ili razvojne platforme su sve češće besplatno dostupni, što olakšava razvoj platformi po meri.



# Kriterijumi za odabir hardvera

- ❖ Uverite se da je hardver koji planirate da koristite već podržan od strane Linuks kernela, da ima bootloader sa otvorenim kodom, pre svega SoC koji ciljate.
- ❖ Postojanje podrške u zvaničnim verzijama projekata (kernel, bootloader) je daleko bolje: kvalitetnije i dostupne su nove verzije.
- ❖ Neki proizvođači SoC i/ili ploča ne doprinose svojim izmenama glavnom Linuks kernelu. Pitajte ih da to učine ili koristite drugi hardver ako možete. Dobra mera je proveriti razliku između njihovog i zvaničnog kernela.
- ❖ **Postojaće velika razlika, u vremenu i ceni razvoja, između hardvera sa korektnom podrškom u zvaničnom Linuks kernelu i hardvera sa slabom podrškom**



Arhitekture za namenske Linuks sisteme

# UVOD U LINUKS U NAMENSKIM SISTEMIMA



# Globalna arhitektura

Razvojna mašina

Alati  
kompajler  
debager  
...

Odredišni ugrađeni sistem

Aplikacija

Aplikacija

Biblioteka

Biblioteka

Biblioteka

Biblioteka

C biblioteka

Linuks kernel

Bootloader



# Programske komponente

- ❖ Alat za prevodenje za drugu platformu (unakrsno prevodenje, Cross-compilation)
  - ❖ Kompajler koji se pokreće na razvojnoj mašini, ali generiše kod za odredišnu
- ❖ Bootloader
  - ❖ Pokreće ga hardver, odgovoran je za izvršavanje kernela
- ❖ Linuks Kernel
  - ❖ Sadrži proces menadžment i memorijski menadžment, mrežni stek, rukovaoce uređajima i nudi servise za aplikacije u korisničkom prostoru
- ❖ C biblioteka
  - ❖ Sprega između kernela i aplikacija u korisničkom prostoru
- ❖ Biblioteke i aplikacije
  - ❖ Dobijene od drugih ili napravljene "u kući"



# Rad na Linuksu za namenske sisteme

- ❖ Nekoliko jasnih zadataka je potrebno za razvoj Linuksa za određeni proizvod:
  - ❖ **Razvoj Board Support Package**
    - ❖ BSP sadrži bootloader i kernel sa odgovarajućim rukovaocima uređajima za ciljni hardver
    - ❖ Ovo je ujedno i cilj ovog kursa
  - ❖ **Integracija sistema**
    - ❖ Integracija svih komponenti u sistem: bootloader-a, kernela, tuđih biblioteka i aplikacija kao i onih razvijenih "u kući"
    - ❖ Ovo je takođe cilj ovog kursa
  - ❖ **Razvoj aplikacija**
    - ❖ Regularne Linuks aplikacije, ali koriste posebno odabrane biblioteke



Razvojno okruženje Linuksa u namenskim sistemima

# LINUKS U NAMENSKIM SISTEMIMA



# Rešenja Linuksa u namenskim sistemima

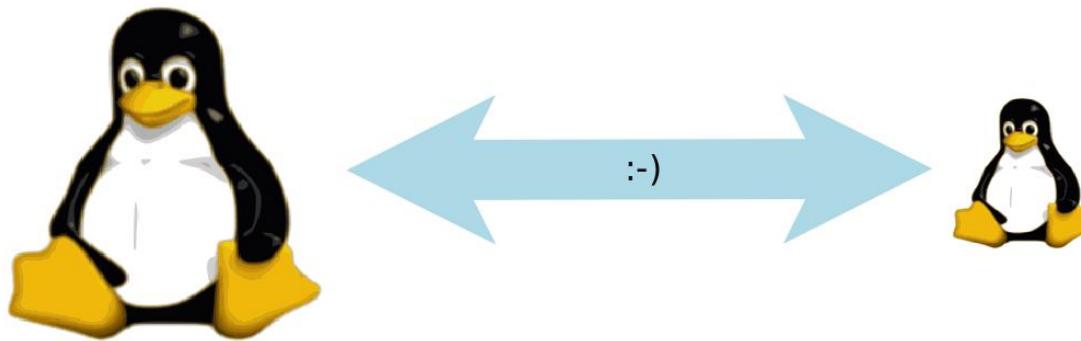


- ❖ Dva načina za prebacivanje na Linuks za namenske sisteme
  - ❖ Korišćenje **rešenja i podrške od proizvođača** kao što su MontaVista, Wind River ili TimeSys. Ova rešenja dolaze sa razvojnim alatima i okruženjem. Predstavljaju kombinaciju komponenti otvorenog koda i vlasničkih alata.
  - ❖ Korišćenje **rešenja zajednice**. Potpuno su otvorena i podržana od strane zajednice.
- ❖ Treba pribegavati korišćenju rešenja zajednice otvorenog koda.
  - ❖ Znajući koncepte, ni prelazak na rešenja proizvođača neće biti problem



# OS za razvoj Linuksa

- ❖ Preporučuje se upotreba Linuksa kao desktop operativnog sistema za razvoj Linuksa za namenske sisteme iz više razloga.
- ❖ Svi alati zajednice su razvijeni i dizajnirani za pokretanje na Linuksu. Pokušaj korišćenja istih na drugom OS (Windows, Mac, OS X) bi doveo do problema, i njihova upotreba na ovim sistemima u principu nije podržana od strane zajednice.
- ❖ Kako je Linuks i na namenskom uređaju, svo znanje stečeno korišćenjem Linuksa na desktop računaru se može slično primeniti na namenski uređaj.





# Desktop Linuks distribucija



- ❖ **Bilo koja dobra i dovoljno sveža Linuks desktop distribucija** se može koristiti na razvojnoj radnoj stanici
  - ❖ Ubuntu, Debian, Fedora, openSUSE, Red Hat, itd.
- ❖ Na kursu se koristi Ubuntu, kao **često upotrebljavana i jednostavna** desktop Linuks distribucija
- ❖ Ubuntu podešavanja na virtuelnim mašinama su namerno ostavljena nedirnuta nakon uobičajenog instalacionog procesa. Učenje o Linuksu za namenske sisteme takođe obuhvata i učenje o alatima potrebnim na razvojnoj radnoj stanici



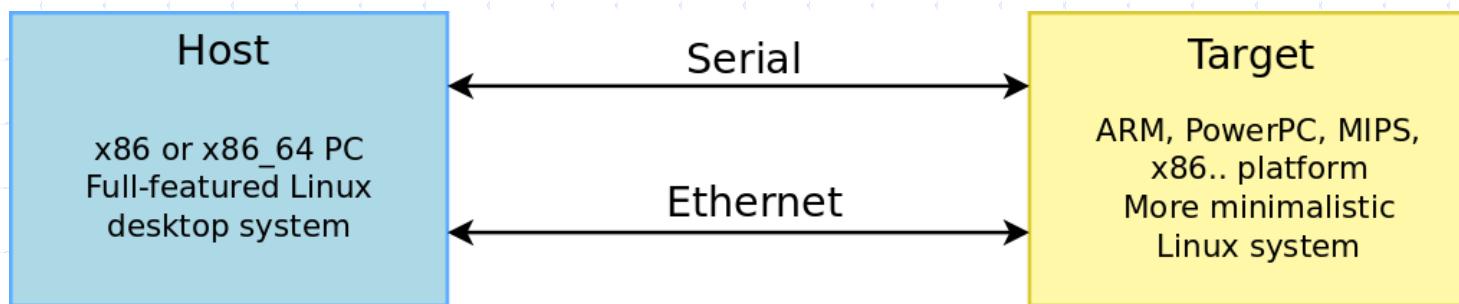


# Linuks korenski i ostali korisnici

- ❖ Linuks je operativni sistem sa više korisnika
  - ❖ Korenski korisnik je **administrator** i može da sprovodi privilegovane operacije kao što su: mauntovanje sistema datoteka, konfigurisanje mreže, stvaranje datoteka uređaja, promena konfiguracije sistema, instaliranje ili uklanjanje paketa, itd.
  - ❖ Svi **ostali su neprivilegovani** korisnici i ne mogu da sprovode pomenute operacije predviđene za administratora
- ❖ Na Ubuntu sistemu nije moguće logovanje kao `root` korisnik već samo kao regularan korisnik.
- ❖ Sistem je konfigurisan tako da je prvom stvorenom nalogu omogućeno da pokreće privilegovane operacije kroz program pod nazivom `sudo`.
- ❖ Primer: `sudo mount /dev/sda2 /mnt/disk`

# Razvojna i odredišna platforma

- ❖ Prilikom razvoja namenskog uređaja, uvek postoji podela između
  - ❖ *host*, razvojne radne stanice, koja je tipično moćan PC
  - ❖ *target*, koji je namenski sistem koji se razvija
- ❖ Mogu biti povezani različitim sredstvima:
  - ❖ Skoro uvek serijskom linijom u svrhe debagovanja
  - ❖ Često Ethernet vezom
  - ❖ Ponekad JTAG spregom za debagovanje na niskom nivou





# Program za komunikaciju preko serijske linije



- ❖ Izuzetno važan alat za razvoj namenskih uređaja je program za komunikaciju preko serijske linije, poput program HyperTerminal u OS Windows.
- ❖ Postoje razne opcije u Linuksu: Minicom, Picocom, Gtkterm, Putty, itd.
- ❖ Na ovom kursu se predlaže picocom
  - ❖ Instalacija pomoću sudo apt-get install picocom
  - ❖ Pokretanje komandom picocom -b BAUD\_RATE /dev/SERIAL\_DEVICE
  - ❖ Prekid rada komandom Control-A Control-X
  - ❖ SERIAL\_DEVICE je tipično
    - ❖ ttyUSBx za USB na serijsku vezu
    - ❖ ttySx za fizičke serijske portove



## Saveti za komandnu liniju

- ❖ Korišćenje komandne linije je mandatorno zbog mnogih operacija potrebnih za razvoj Linuksa za namenske uređaje
- ❖ Veoma je moćan način za interakciju sa sistemom, te može da uštedi dosta vremena.
- ❖ Možete koristiti nekoliko tabova u Gnome Terminalu
- ❖ Setite se da imate relativnu putanju (npr: ../../linux) in addition to absolute paths (npr: /home/user)
- ❖ U šelu, pritisnite [Control] [r], i tada će se komanda pretraživati u istoriji komandi. Pritisnite ponovo [Control] [r] da biste pretražili istoriju unazad.
- ❖ Možete da prenesete putanje direktno sa drugih menadžera u terminal pomoću drag-and-drop



Alati za unakrsno prevodenje - definicija i komponente

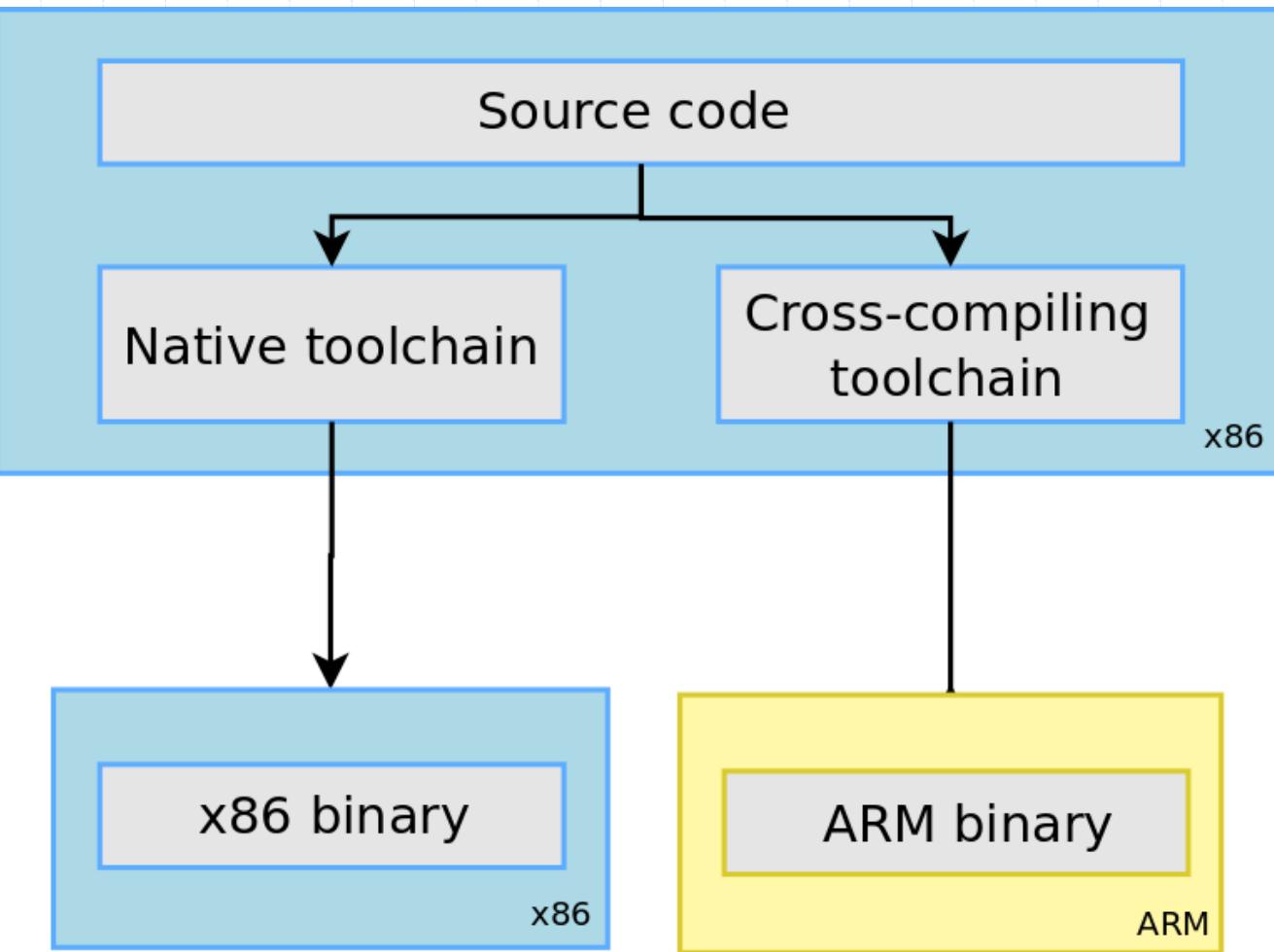
# RAZVOJNO OKRUŽENJE LINUKSA U NAMENSKIM SISTEMIMA



# Definicija (1/2)

- ❖ Uobičajeni alati za razvoj dostupni na GNU/Linuks radnoj stanici su **ugrađeni alati za prevodenje**
- ❖ Ovi alati za prevodenje se pokreću na radnoj stanici i generišu kod za radnu stanicu, najčešće je to x86.
- ❖ Za razvoj namenskih sistema, najčešće je nemoguće ili neinteresantno korišćenje ugrađenih alata za prevodenje
  - ❖ Odredišna platforma ima preveliku restrikciju po pitanju prostora za skladištenje i/ili memorije.
  - ❖ Odredišna platforma je spora u poređenju sa radnom stanicom
  - ❖ Verovatno ne želite da instalirate sve razvojne alate na odredišnu platformu.
- ❖ Stoga su **alati za unakrsno prevodenje** najčešće korišćeni. Oni se pokreću na radnoj stanici, ali generišu kod za odredišnu platformu.

## Definicija (2/2)



Compilation  
machine

Execution  
machine



# Mašine u bild procedurama

- ❖ Tri mašine se moraju razdvojiti kada je reč o pravljenju opisanog alata za prevodenje
  - ❖ **bild** mašina, gde se pravi alat za unakrsno prevodenje.
  - ❖ **host** mašina, gde se izvršava alat za unakrsno prevodenje i pravi binarnu, izvršnu datoteku za odredišnu platformu.
  - ❖ **target** mašina, gde se izvršavaju binarne datoteke napravljene od strane alata za unakrsno prevodenje.
- ❖ Četiri uobičajena bild tipa postoje za alate za prevodenje



# Različite bild procedure alata za prevodenje



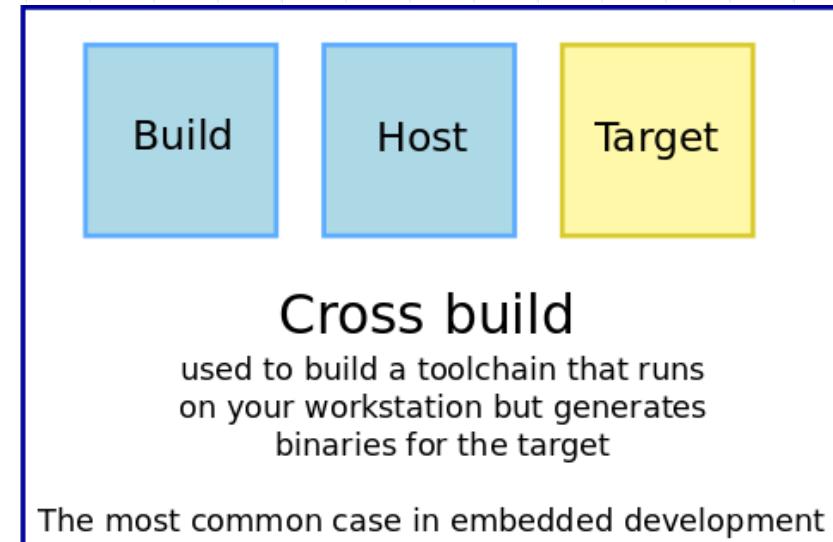
## Native build

used to build the normal gcc  
of a workstation



## Cross-native build

used to build a toolchain that runs on your target and generates binaries for the target



## Cross build

used to build a toolchain that runs on your workstation but generates binaries for the target

The most common case in embedded development



## Canadian build

used to build on architecture A a toolchain that runs on architecture B and generates binaries for architecture C



# Komponente

Binutils

Kernel headers

C/C++ libraries

GCC compiler

GDB debugger  
(optional)

Cross-compilation toolchain

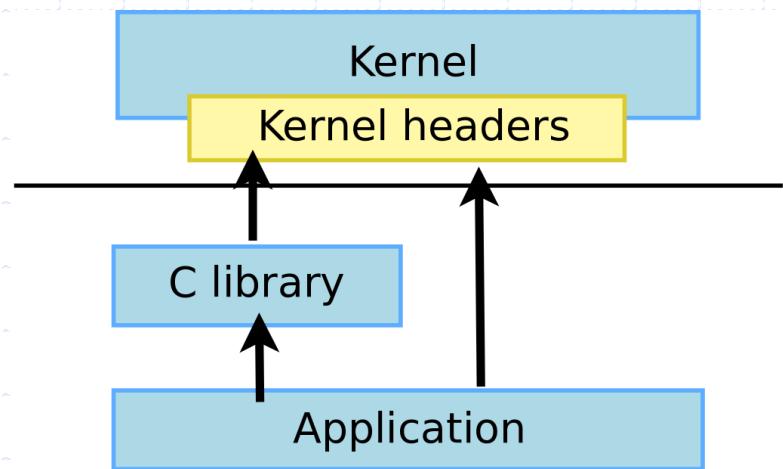


# Binutils

- ❖ **Binutils** je skup alata za generisanje i manipulisanje binarnim datotekama za zadatu CPU arhitekturu
  - ❖ `as`, asembler, koji generiše binarnu datoteku na osnovu asemblerskog izvornog koda
  - ❖ `ld`, povezivač
  - ❖ `ar`, `ranlib`, za generisanje `.a` arhiva, korišćenih za biblioteke
  - ❖ `objdump`, `readelf`, `size`, `nm`, `strings`, za inspekciju binarnih datoteka. Veoma koristan alat za analizu!
  - ❖ `Strip`, za odbacivanje dela binarnih datoteka koje se koriste samo za debagovanje (umanjujući im veličinu).
- ❖ <http://www.gnu.org/software/binutils/>
- ❖ GPL licenca

# Kernel zaglavlja (1/3)

- ❖ C biblioteka i prevedeni programi zahtevaju interakciju sa kernelom
  - ❖ Dostupni sistemski pozivi i njihovi brojevi
  - ❖ Definicije konstanti
  - ❖ Strukture podataka, itd.
- ❖ Stoga, prevodenje C biblioteke zahteva zaglavlja kernela i mnoge aplikacije ih takođe zahtevaju.
- ❖ Dostupno u `<linux/...>` i `<asm/...>` i još nekim drugim direktorijumima koji odgovaraju onima vidljivim u `include/` u izvornom kodu kernela





# Kernel zaglavlja (2/3)

- ❖ Brojevi sistemskih poziva u <asm/unistd.h>

```
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
```

- ❖ definicije konstanti, u <asm-generic/fcntl.h>, uključeno sa putanje <asm/fcntl.h>, uključeno sa putanje <linux/fcntl.h>

```
#define O_RDWR 00000002
```

- ❖ Strukture podataka, u <asm/stat.h>

```
struct stat {
    unsigned long st_dev;
    unsigned long st_ino;
    [...]
};
```



## Kernel zaglavlja (3/3)

- ❖ Sprega (ABI) kernela i korisničkog prostora je **kompatibilna sa starijim verzijama**
  - ❖ Binarne datoteke generisane alatom za prevodenje uz korišćenje kernel zaglavlja starijih od pokrenutog kernela će raditi bez problema, ali neće moći da koriste nove sistemske pozive, strukture podataka, itd.
  - ❖ Binarne datoteke generisane alatom za prevodenje uz korišćenje kernel zaglavlja novijih od pokrenutog kernela mogu raditi ukoliko ne koriste skorašnje funkcionalnosti, inače bi se neuspšno izvršile
  - ❖ Korišćenje poslednjih zaglavlja Linuks kernela nije neophodno osim ako je pristup novim kernel funkcionalnostima potreban.
  - ❖ Kernel zaglavlja se mogu raspakovati iz kernel izvornog koda, korišćenjem `headers_install` kernel Makefile ciljem.



# GCC



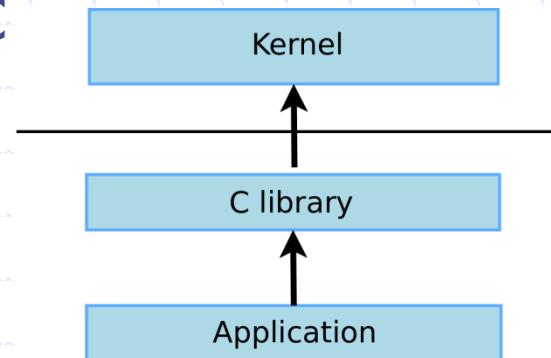
- ❖ GNU Compiler Collection, poznati besplatan kompjajler
- ❖ Prevodi C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, i generiše kod za velik broj procesorskih arhitektura, uključujući ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86\\_64, IA64, Xtensa, itd.
- ❖ <http://gcc.gnu.org/>
- ❖ Dostupan pod GPL licencom, biblioteke pod LGPL licencom.





# C biblioteka

- ❖ C biblioteka je važna komponenta Linuks sistema
  - ❖ Sprega između aplikacija i kernela
  - ❖ Obezbeđuje dobro poznatu standardnu C spregu za olakšani razvoj aplikacija
- ❖ Nekoliko C biblioteka je dostupno: *glibc*, *uClibc*, *musl*, *dietlibc*, *newlib*, itd.
- ❖ Izbor C biblioteke mora biti napravljen u vreme generisanja alata za unakrsno prevodenje, jer se GCC kompjajler prevodi nasuprot specifičnoj C biblioteci.





# glibc



- ❖ Licenca: LGPL
- ❖ C biblioteka sa GNU projekta
- ❖ Dizajnirana za performansu, usklađenost sa standardima i portabilnost
- ❖ Nalazi se na svim GNU / Linuks host sistemima
- ❖ Naravno, aktivno se održava
- ❖ Podrazumevano, prilično je velika za male namenske sisteme: približno 2.5 MB na ARM (version 2.9 - libc: 1.5 MB, libm: 750 KB)
- ❖ Ipak, neke funkcionalnosti koje nisu potrebne u namenskim sistemima se mogu isključiti u konfiguraciji (uvučeni su iz starog *eglibc* projekta).
- ❖ <http://www.gnu.org/software/libc/>





# uClibc-ng (1/2)

- ❖ <http://uclibc-ng.org/>
- ❖ Nastavak starog uClibc projekta
- ❖ Licenca: LGPL
- ❖ Mala C biblioteka za male namenske sisteme
  - ❖ Visoka konfigurabilnost: mnoge funkcionalnosti se mogu uključiti/isključiti kroz menuconfig spregu
  - ❖ Podržava većinu namenskih arhitektura
  - ❖ Podržava arhitekture bez MMU (ARM Cortex-M, Blackfin, itd.)
  - ❖ Ne garantuje binarnu kompatibilnost. Može da zahteva rekomplajliranje aplikacija kada se promeni konfiguracija biblioteke.
  - ❖ Fokusira se na veličinu biblioteke pre nego na performansu
  - ❖ Malo vreme prevodenja



## uClibc-ng (2/2)

- ❖ Većina aplikacija se prevodi sa uClibc-ng. Ovo se odnosi na sve aplikacije u namenskim sistemima.
- ❖ Veličina (arm): 3.5x manja od glibc!
  - ❖ uClibc-ng 1.0.14: približno 716kB (libuClibc: 282kB, libm: 73kB)
  - ❖ glibc 2.22: približno 2.5 MB
- ❖ Neke funkcionalnosti nisu dostupne ili su ograničene: npr. nasleđivanje prioriteta mutex-a.
- ❖ Koristi se na velikom broj namenskih proizvoda koji su u produkciji, uključujući uređaje iz grupe potrošačke elektronike.



# Poređenje uClibc-ng i glibc

- ❖ Veličina izvršne datoteke poređena na ARM, testirana sa *glibc 2.22* i *uClibc-ng 1.0.14*

C program	Prevedeno dinamički sa glibc	Prevedeno dinamički sa uClibc	Prevedeno statički sa glibc	Prevedeno statički sa uClibc
Običan "hello world"	2.7 KB	2.5 KB	479 KB	33.4 KB
Busybox	503 KB	664 KB	1206 KB	818 KB



# musl C biblioteka

❖ <http://www.musl-libc.org/>

- ❖ Veoma mala, brza i jednostavna biblioteka za namenske sisteme
- ❖ Napravljena dok je razvoj uClibc bio obustavljen
- ❖ U suštini, odlična je za pravljenje malih izvršnih datoteka sa statickim uvezivanjem
- ❖ Permissive license (MIT)
- ❖ Poređenje sa drugim C bibliotekama:  
[http://www.etalabs.net/compare\\_libcs.html](http://www.etalabs.net/compare_libcs.html)
- ❖ Podržana od strane bild sistema kao što je Buildroot



## Ostale manje C biblioteke

- ❖ Razvijeno je još nekoliko manjih C biblioteka, ali nijedna od njih nema za cilj omogućavanje prevodenja velikih postojećih aplikacija
- ❖ Mogu da se koriste za relativno jednostane program, tipično za pravljenje veoma malih izvršnih datoteka sa statičkim uvezivanjem i pokretanje istih u veoma malim korenskim sistemima datoteka.
- ❖ Mogućnosti:
  - ❖ Dietlibc, <http://fefe.de/dietlibc/>. Približno 70 KB.
  - ❖ Newlib, <http://sourcware.org/newlib/>
  - ❖ Klibc, <http://www.kernel.org/pub/linux/libs/klibc/>, dizajnirana za *initramfs* ili *initrd* za vreme boot-a.



Opcije alata za prevodenje

# RAZVOJNO OKRUŽENJE LINUKSA U NAMENSKIM SISTEMIMA



# ABI



- ❖ Kada se pravi alat za prevodenje, ABI, koji se koristi za generisanje binarnih datoteka, mora biti definisan
- ❖ ABI, *Application Binary Interface*, definiše pozivne konvencije (kako se prosleđuju argumenti funkcija, kako se prosleđuje povratna vrednost i kako se prave sistemske pozive) i organizacije struktura (poravnanja i sl)
- ❖ Sve binarne datoteke u sistemu moraju biti prevedene korišćenjem istog ABI, i kernel mora da ume da rukuje istim ABI
- ❖ Na ARM arhitekturi, postoji dva glavna ABI: *OABI* i *EABI*
  - ❖ U novije vreme se koristi *EABI*
- ❖ Na MIPS arhitekturi, postoji nekoliko ABI: *o32*, *o64*, *n32*, *n64*
- ❖ [http://en.wikipedia.org/wiki/Application\\_Binary\\_Interface](http://en.wikipedia.org/wiki/Application_Binary_Interface)



# Podrška za račun u pokretnom zarezu



- ❖ Neki procesori imaju jedinicu za račun u pokretnom (FPU) zarezu, dok drugi nemaju
  - ❖ Npr, mnogi ARMv4 i ARMv5 procesori nemaju FPU. Od ARMv7, VFP jedinica je postala mandatorna.
- ❖ Za procesore koji imaju FPU, alat za prevodenje mora da generiše *hard float* kod, da bi koristio instrukcije u pokretnom zarezu direktno.
- ❖ Za procesore koji nemaju FPU, postoje dva rešenja:
  - ❖ Generisati *hard float* kod i osloniti se na kernel koji će emulirati instrukcije u pokretnom zarezu. Ovo je veoma sporo.
  - ❖ Generisati *soft float* kod, tako da se umesto generisanja instrukcija u pokretnom zarezu, generišu pozivi biblioteka iz korisničkog prostora
- ❖ Odluka mora biti doneta u vreme konfiguracije alata za prevodenje
- ❖ Moguće je konfigurisati koja će FPU biti korišćena



# CPU optimizacione zastavice

- ❖ Skup alata za unakrsno prevodenje je specifičan za svaku CPU arhitekturu (ARM, x86, MIPS, PowerPC)
- ❖ Međutim, uz `-march=`, `-mcpu=`, `-mtune=` opcije, moguće je preciznije odabratи tip ciljnog procesora
  - ❖ Npr, `-march=armv7 -mcpu=cortex-a8`
- ❖ U vreme prevodenja alata za prevodenje, mogu se odabratи vrednosti. One se koriste:
  - ❖ Kao podrazumevane vrednosti za alate za unakrsno prevodenje, kada druge vrednosti za `-march`, `-mcpu`, `-mtune` opcije nisu prosleđen
  - ❖ Za prevodenje C biblioteke
- ❖ Čak i ako je C biblioteka prevedena za armv5t, to ne sprečava prevodenje drugih programa za armv7



Obezbeđivanje alata za prevodenje

# RAZVOJNO OKRUŽENJE LINUKSA U NAMENSKIM SISTEMIMA



# Pravljenje alata za prevodenje

- ❖ Pravljenje alata za unakrsno prevodenje je težak i naporan zadatak! Može da traje danima ili nedeljama!
  - ❖ Treba naučiti dosta detalja: mnogo komponenti treba prevoditi, komplikovane konfiguracije
  - ❖ Mnogo odluka je potrebno napraviti (vezija C biblioteke, ABI, floating point mehanizmi, verzije komponenti, itd)
  - ❖ Potrebna su kernel zaglavla i izvorni kod C biblioteka
  - ❖ Potrebno je poznavanje trenutnih `gcc` problema i zakrpa na ciljnoj platformi
  - ❖ Korisno je za upoznavanje sa alatima za konfiguraciju i prevodenje
  - ❖ Pogledajte *Crosstool-NG* docs / direktorijum za detalje o tome kako se pravi alat za prevodenje.



# Preuzmite preveden alat za prevodenje



- ❖ Mnogi biraju ovo rešenje
  - ❖ Prednost: najjednostavnije i najzgodnije rešenje
  - ❖ Mane: ne postoji mogućnost za fina podešavanja alata prema potrebama
- ❖ Uverite se da alat koji ste pronašli odgovara zahtevima: CPU, endianness, C biblioteka, verzije komponenti, ABI, softverska ili harverska jedinica u pokretnom zarezu, itd.
- ❖ Mogući izbor:
  - ❖ Alati sadržani u paketima vaše distribucije. Ubuntu primer:

```
sudo apt-get install gcc-arm-linux-gnueabi
sudo apt-get install gcc-arm-linux-gnueabihf
```
  - ❖ Sourcery CodeBench alati sada podržavaju samo MIPS, NIOS-II, AMD64 i Hexagon. Stare verzije sa podrškom za ARM su i dalje dostupni kroz bild sisteme (Buildroot...)
  - ❖ Alat obezbeđen od strane proizvođača hardvera.



# Uslužni programi za pravljenje alata za prevodenje (1/3)



- ❖ Drugo rešenje je korišćenje usložnog programa za **automatizaciju procesa pravljenja alata za prevodenje**
  - ❖ Ista prednost kao i kod korišćenja prevedenog alata: ne morate da se zamarate oko detalja u procesu prevodenja.
  - ❖ Ali takođe nudi veću fleksibilnost po pitanju konfiguracije alata za prevodenje, odabira verzije neke komponente, itd.
  - ❖ Obično sadrže i nekoliko zagrpa za popravljanje poznatih problema sa raznim komponentama na nekoj arhitekturi.
  - ❖ Više alata po istom principu: šel skripte ili Makefile automatski dobavljaju arhive, raspakuju ih, konfigurišu, prevode i instaliraju različite komponente



# Uslužni programi za pravljenje alata za prevođenje (2/3)



- ❖ Crosstool-ng
  - ❖ Zamena za stariji Crosstool program, sa konfiguracionim sistemom nalik na menuconfig
  - ❖ Razne funkcionalnosti: podržava uClibc, glibc, musl, hard i soft float, mnoge arhitekture
  - ❖ Aktivno se održava
  - ❖ <http://crosstool-ng.org/>



# Uslužni programi za pravljenje alata za prevodenje (3/3)

- ❖ Mnogi sistemi za pravljenje korenskog sistema datoteka dozvoljavaju i pravljenje alata za prevodenje
  - ❖ **Buildroot**
    - ❖ Makefile-baziran. Može da napravi (e)glibc, uClibc i musl bazirane alate za širok spektar arhitektura.
    - ❖ <http://www.buildroot.net>
  - ❖ **PTXdist**
    - ❖ Makefile-bazirani, uClibc ili glibc, održavan pre svega od strane *Pengutronix*
    - ❖ <http://pengutronix.de/software/ptxdist/>
  - ❖ **OpenEmbedded / Yocto**
    - ❖ Komplikovaniji bild sistem pun opcija
    - ❖ <http://www.openembedded.org/>
    - ❖ <https://www.yoctoproject.org/>



# Crosstool-NG: installation and usage

- ❖ Instalacija Crosstool-NG se može uraditi na sistemu ili samo lokalno u direktorijumu gde je izvorni kod. Za lokalnu instalaciju:

```
./configure --enable-local
```

```
make
```

```
make install
```

- ❖ Neki primeri konfiguracija za različite arhitekture su dostupni među primerima, mogu se izlistati pomoću:

```
./ct-ng list-samples
```

- ❖ Za učitavanje primera konfiguracije:

```
./ct-ng <sample-name>
```

- ❖ Za promenu konfiguracije:

```
./ct-ng menuconfig
```

- ❖ Za prevođenje alata:

```
./ct-ng build
```



# Sadržaj alata za prevodenje

- ❖ Izvršne datoteke alata za unakrsno prevodenje, u `bin/`
  - ❖ Ovaj direktorijum treba dodati u `PATH` varijablu za lakše korišćenje alata za unakrsno prevodenje
- ❖ Jedan ili više `sysroot`, svaki sadrži
  - ❖ C biblioteku i povezane biblioteke, prevedene za odredište
  - ❖ Zaglavla C biblioteke i kernela
- ❖ Postoji po jedan `sysroot` za svaku varijantu: alat može biti *multilib* ukoliko postoji nekoliko kopija C biblioteke sa različitim konfiguracijama (npr. ARMv4T, ARMv5T, itd)
  - ❖ Stari CodeSourcery ARM alati su bili multilib, `sysroot`-ovi u:  
`arm-none-linux-gnueabi/libc/`,  
`arm-none-linux-gnueabi/libc/armv4t`,  
`arm-none-linux-gnueabi/libc/thumb2`
  - ❖ Crosstool-NG alati mogu takođe biti multilib (još u eksperimentalnoj fazi), u suprotnom `sysroot` je u  
`arm-unknown-linux-uclibcgnueabi/sysroot`



Sekvenca pokretanja (boot)

# RAZVOJNO OKRUŽENJE LINUKSA U NAMENSKIM SISTEMIMA

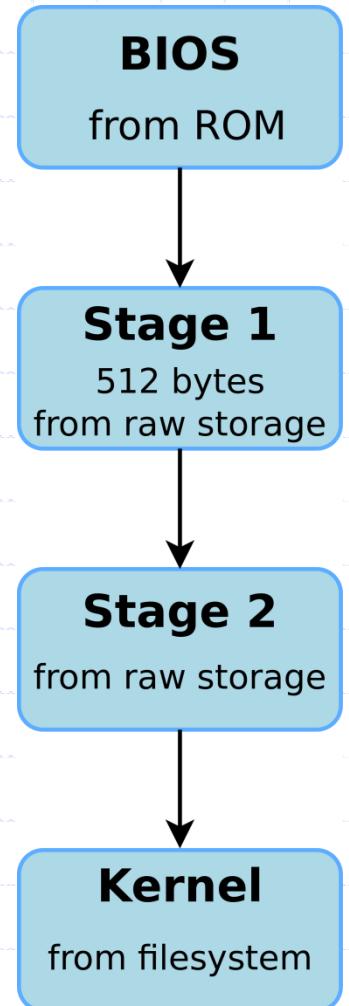


# Bootloader-i

- ❖ Bootloader je deo koda odgovoran za
  - ❖ Osnovnu inicijalizaciju hardvera
  - ❖ Učitavanje binarne datoteke aplikacije, najčešće kernela operativnog sistema, iz flash memorije, sa mreže ili sa drugog tipa memorije.
  - ❖ Može da uključi i dekompresiju binarne datoteke aplikacije.
  - ❖ Izvršavanje aplikacije
- ❖ Osim ovih osnovnih funkcionalnosti, većina bootloader-a nudi šel sa različitim komandama koji implementira različite operacije.
- ❖ Učitavanje podataka iz skladištene memorije ili sa mreže, provera memorije, hardverska dijagnostika, testiranje, itd.

# Bootloader-i na BIOS-baziranoj x86 arhitekturi (1/2)

- ❖ x86 procesori su tipično upakovani u ploču sa trajnom memorijom koja sadrži program pod nazivom BIOS.
- ❖ Na stariim BIOS-baziranim x86 platformama: BIOS je odgovoran za inicijalizaciju osnovnog hardvera i učitavanje veoma malog dela koda iz trajne memorije.
- ❖ Ovaj deo koda je tipično bootloader prve faze koji će učitati potpun bootloader.
- ❖ Tipično ima podršku za različite formate sistema datoteka tako da može da učita sliku kernela direktno sa uobičajenog sistema datoteka.
- ❖ Ova sekvenca se razlikuje na modernijim, EFI-baziranim sistemima.





# Bootloader-i na BIOS-baziranoj x86 arhitekturi (2/2)



- ❖ GRUB, Grand Unified Bootloader, najmoćniji bootloader.

<http://www.gnu.org/software/grub/>

- ❖ Može da čita mnoge formate sistema datoteka, a učitava slike kernela i konfiguracije, nudi moćan šel sa različitim komandama, može da učita sliku kernela preko mreže, itd.
- ❖ Za detalje pogledajte prezentaciju:  
<http://free-electrons.com/docs/grub/>
- ❖ Syslinux, za boot-ovanje preko mreže ili prenosnog medijuma (USB key, CD-ROM)

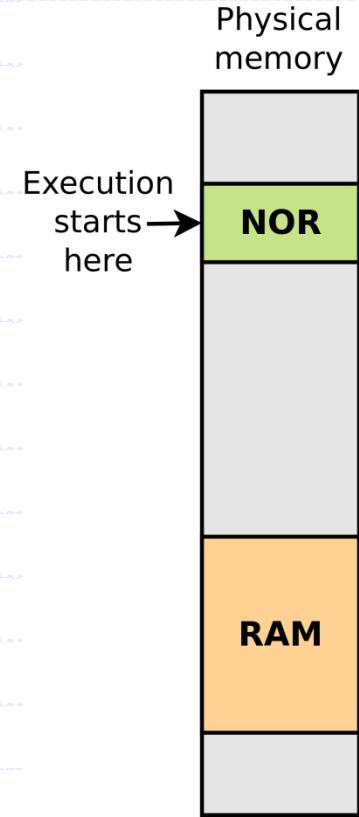
<http://www.kernel.org/pub/linux/utils/boot/syslinux/>



# Boot-ovanje na namenskim procesorima: slučaj 1



- ❖ Kada se uključi, CPU počne izvršavanje koda na određenoj, fiksnoj adresi
- ❖ Ne postoji drugi mehanizam boot-ovanja omogućen od CPU
- ❖ Hardver mora biti dizajniran tako da osigura da NOR flash čip bude povezan tako da je dostupan na adresi na kojoj CPU započinje izvršenje instrukcija
- ❖ Bootloader prve faze mora da se nalazi na ovoj adresi u NOR memoriji
- ❖ NOR je mandatoran jer omogućava slučajan pristup, koji NAND nedozvoljava
- ❖ **Nije više toliko uobičajen** (nepraktičan, zahteva NOR flash)



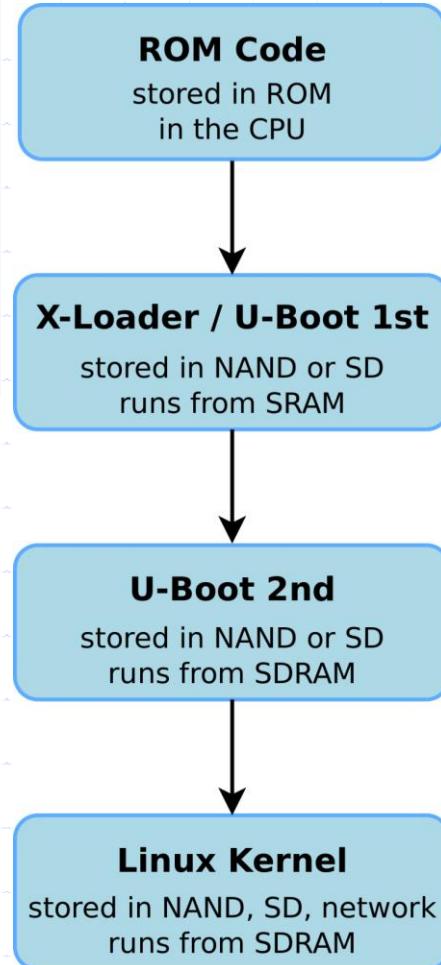


# Boot-ovanje na namenskim procesorima: slučaj 2



- ❖ CPU ima integriran kod za boot-ovanje u ROM-u
  - ❖ BootROM na AT91 CPU, "ROM code" na OMAP, itd.
  - ❖ Detalji zavise od CPU
- ❖ Ovaj kod za boot-ovanje može da pokrene bootloader prve faze iz skladišta u SRAM (DRAM još nije inicijalizovan u to vreme)
  - ❖ Skladište može biti tipa: MMC, NAND, SPI flash, UART (prenos podataka preko serijske linije), itd.
- ❖ Bootloader prve faze je
  - ❖ Ograničene veličine zbog hardverskih ograničenja (veličina SRAM)
  - ❖ Obezbeđen od strane proizvođača CPU ili kroz projekte zajednice
- ❖ Bootloader prve faze mora da inicijalizuje DRAM i ostale hardverske uređaje i učita bootloader druge faze u RAM

# Booting on ARM TI OMAP3



- ❖ **ROM Code:** pokušava da pronađe validnu bootstrap sliku na različitim lokacijama i učitava je u SRAM ili RAM (RAM može biti inicijalizovancan ROM Code-om kroz konfiguraciona zaglavlja). Veličina je ograničena na <64 KB. Nije moguća interakcija sa korisnikom.
- ❖ **X-Loader ili U-Boot:** pokreće se iz SRAM. Inicijalizuje DRAM, NAND ili MMC kontroler i učitava u RAM bootloader druge faze i pokreće ga. Nije moguća interakcija sa korisnikom. Datoteka pod nazivom MLO.
- ❖ **U-Boot:** pokreće se iz RAM-a. Inicijalizuje još neke uređaje (mreža, USB, itd). Učitava sliku kernela iz trajne memorije ili preko mreže u RAM i pokreće ga. Dostupan je šel sa komandama. Datoteka pod nazivom u-boot.bin ili u-boot.img.
- ❖ **Linux Kernel:** pokreće se iz RAM-a. Preuzima kompletan sistem (bootloader-i više ne postoje).



# Generički bootloader-i za namenske CPU

- ❖ Fokusiraćemo se na generički deo, glavni bootloader, koji nudi najvažnije funkcionalnosti.
- ❖ Postoji nekoliko generičkih bootloader-a otvorenog koda.
- ❖ Najpopularniji:
  - ❖ **U-Boot**, univerzalni bootloader razvijen od Denx na ARM arhitekturama, ali i na PPC, MIPS, x86, m68k, NIOS, itd. To je danas de-facto standard. Izučavaćemo ga detaljno.  
<http://www.denx.de/wiki/U-Boot>
  - ❖ **Barebox**, novi bootloader nezavisan od arhitekture, pisan kao naslednik U-Boot-a. Bolji dizajn, bolji kod, aktivni razvoj, ali i dalje nema toliko harverske podrške koliko U-Boot.  
[http://www.barebox.org}](http://www.barebox.org)
- ❖ Postoje još mnogo drugih namenskih bootloaders otvorenog koda, često zavisnih od arhitekture
  - ❖ RedBoot, Yaboot, PMON, itd.



U-boot

# RAZVOJNO OKRUŽENJE LINUKSA U NAMENSKIM SISTEMIMA



# U-Boot



- ❖ U-Boot je tipičan projekat besplatnog softvera
  - ❖ Licenca: GPLv2 (kao i Linuks)
  - ❖ Besplatno dostupno na <http://www.denx.de/wiki/U-Boot>
  - ❖ Dokumentacija dostupna na  
<http://www.denx.de/wiki/U-Boot/Documentation>
  - ❖ Poslednji razvojni izvorni kod se nalazi na Git repozitorijumu:  
<http://git.denx.de/?p=u-boot.git;a=summary>
  - ❖ Razvoj i diskusija se dešavaju kroz otvorenu mejling listu  
<http://lists.denx.de/pipermail/u-boot/>
  - ❖ Od kraja 2008, prati fiksni intervali izbacivanja novih verzija.
  - ❖ Svaka tri meseca se izbacuje nova verzija. Verzije su imenovane po konvenciji u formatu YYYY.MM



# U-Boot konfigurisanje



- ❖ Preuzmite izvorni kod sa sajta i raspakujte ga
- ❖ configs/ direktorijum sadrži po jednu konfiguracionu datoteku za svaku podržanu ploču.
  - ❖ Ona definiše tip CPU, periferije i njihove konfiguracije, memorijsku mapu, U-Boot funkcionalnosti koje treba prevesti i sl.
- ❖ Napomena: U-Boot migrira sa konfiguracionih datoteka ploče u zaglavljima na putanji (`include/configs/`) na *defconfig* kao što je i u Linux kernelu (`configs/`)
  - ❖ Nisu sve ploče prebačene na novi konfiguracioni sistem.
  - ❖ Starije verzije U-Boot-a ponuđene od strane proizvođača možda još ne koristi ovaj konfiguracioni sistem.



# U-Boot konfiguraciona datoteka

## ❖ CHIP\_defconfig

```
CONFIG_ARM=y
CONFIG_ARCH_SUNXI=y
CONFIG_MACH_SUN5I=y
CONFIG_DRAM_TIMINGS_DDR3_800E_1066G_1333J=y
# CONFIG_MMC is not set
CONFIG_USB0_VBUS_PIN="PB10"
CONFIG_VIDEO_COMPOSITE=y
CONFIG_DEFAULT_DEVICE_TREE="sun5i-r8-chip"
CONFIG_SPL=y
CONFIG_SYS_EXTRA_OPTIONS="CONS_INDEX=2"
# CONFIG_CMD_IMLS is not set
CONFIG_CMD_DFU=y
CONFIG_CMD_USB_MASS_STORAGE=y
CONFIG_AXP_ALDO3_VOLT=3300
CONFIG_AXP_ALDO4_VOLT=3300
CONFIG_USB_MUSB_GADGET=y
CONFIG_USB_GADGET=y
CONFIG_USB_GADGET_DOWNLOAD=y
CONFIG_G_DNL_MANUFACTURER="Allwinner Technology"
CONFIG_G_DNL_VENDOR_NUM=0x1f3a
CONFIG_G_DNL_PRODUCT_NUM=0x1010
CONFIG_USB_EHCI_HCD=y
```



# Konfiguracija i prevodenje U-Boot-a



- ❖ U-Boot mora biti konfigurisan pre prevodenja
  - ❖ make BOARDNAME\_defconfig
  - ❖ Gde je BOARDNAME ime konfiguracije, kako se vidi u configs direktorijumu.
  - ❖ Nakon toga možete pokrenuti make menuconfig za dalju definiciju U-Boot konfiguracije!
- ❖ Uverite se da je alat za unakrsno prevodenje dostupan u PATH
- ❖ Prevedite U-Boot, navođenjem prefiksa za unakrsno prevodenje.
- ❖ Npr, ukoliko je izvršna datoteka vašeg unakrsnog prevodioca arm-linux-gcc: make CROSS\_COMPILE=arm-linux-
- ❖ Glavni rezultat prevodenja je u datoteci u-boot.bin, koja predstavlja sliku U-Boot-a. Zavisno od vaše specifične platforme, mogu postojati i druge slike: u-boot.img, u-boot.kwb, MLO, itd.



# Instalacija U-Boot-a

- ❖ U-Boot najčešće mora biti instaliran u flash memoriju da bi ga hardver mogao pokrenuti. Zavisno od hardvera, instalacija U-Boot se radi na različite načine:
  - ❖ CPU nudi neku vrstu posebnog boot monitora sa kojim se može komunicirati kroz serijski port ili USB korišćenjem specifičnog protokola.
  - ❖ CPU boot-uje prvi spoljašnji medijum (MMC) pre boot-ovanja sa fiksnog medijuma (NAND). U ovom slučaju, boot-uje se sa MMC-a da bi se upisala nova verzija
  - ❖ U-Boot je već instaliran i može se koristiti za upis nove verzije U-Boot-a. Međutim, treba biti oprezan: Ukoliko nova verzija U-Boot-a ne radi, ploča je neupotrebljiva
  - ❖ Ploča nudi JTAG spregu koja omogućava upis u flash memoriju, bez ikakvog sistema pokrenutog na ploči. Takođe omogućava i spašavanje ploče u slučaju da bootloader ne radi.



# U-boot prompt

- ❖ Povežite odredišnu platformu sa razvojnom stanicom serijskim kablom, odnosno serijskom konzolom.
- ❖ Uključite ploču. Na serijskoj konzoli ćete videti nešto poput:

```
U-Boot 2016.05 (May 17 2016 - 12:41:15 -0400)

CPU: SAMA5D36
Crystal frequency:      12 MHz
CPU clock      :      528 MHz
Master clock   :      132 MHz
DRAM: 256 MiB
NAND: 256 MiB
MMC: mci: 0

In:    serial
Out:   serial
Err:   serial
Net:   gmac0

Hit any key to stop autoboot: 0
=>
```

- ❖ U-Boot šel nudi skup komandi. Izučićemo one najvažnije, a za potpunu referencu pogledajte dokumentaciju ili help komandu.



# Komande sa informacijama

## ❖ Flash informacije (NOR i SPI flash)

```
U-Boot> flinfo
DataFlash:AT45DB021
Nb pages: 1024
Page Size: 264
Size= 270336 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C0001FFF (RO) Bootstrap
Area 1: C0002000 to C0003FFF Environment
Area 2: C0004000 to C0041FFF (RO) U-Boot
```

## ❖ NAND flash informacije

```
U-Boot> nand info
Device 0: nand0, sector size 128 KiB
  Page size    2048 b
  OOB size     64 b
  Erase size 131072 b
```

## ❖ Detalji verzije

```
U-Boot> version
U-Boot 2016.05 (May 17 2016 - 12:41:15 -0400)
```



# Važne komande (1/2)



- ❖ Tačan skup komandi zavisi od U-Boot konfiguracije
- ❖ help i help command
- ❖ ext2load, učitava datoteku sa ext2 sistema datoteka u RAM
  - ❖ Takođe ext2ls lista datoteke, ext2info daje dodatne informacije o datoteci
- ❖ fatload, učitava datoteku sa FAT sistema datoteka u RAM
  - ❖ Takođe fatls lista datoteke, fatinfo daje dodatne informacije o datoteci
- ❖ tftp, učitava datoteku sa mreže u RAM (sa TFTP servera)
- ❖ ping, testira mrežu
- ❖ boot, pokreće podrazumevanu boot komandu, sačuvanu u promenljivoj bootcmd
- ❖ bootz <address>, pokreće sliku kernela učitanu na datu adresu u RAM memoriji



## Važne komande (2/2)

- ❖ loadb, loads, loady, učitava datoteku sa serijske linije u RAM
- ❖ usb, inicijalizuje i kontroliše USB podsistem,
- ❖ uglavnom se koristi za USB uređaje za skladištenje podataka, kao što je USB flash
- ❖ mmc, za inicijalizaciju i kontrolu MMC podistema, koristi se za SD i microSD kartice
- ❖ nand, za brisanje, čitanje i pisanje sadržaja u NAND flash
- ❖ erase, protect, cp, za brisanje, izmenu, zaštitu i upis u NOR flash
- ❖ md, prikazuje sadržaj memorije. Može biti korisno za proveru sadržaja učitanog u memoriju ili za pregled hardverskih registara.
- ❖ mm, modifikuje sadržaj memorije. Može biti korisno za direktnu modifikaciju hardverskih registara u cilju testiranja.



# Varijable okruženja: principi

- ❖ U-Boot može da se konfiguriše kroz varijable U-Boot okruženja
  - ❖ Neke specifične varijable utiču na ponašanje različitih komandi
  - ❖ Dodatne varijable se mogu dodati i koristiti u skriptama
- ❖ Varijable okruženja se učitavaju iz flash memorije u RAM u toku pokretanja U-Boot-a, mogu biti modifikovane i sačuvane trajno na flash.
- ❖ Postoji namenska lokacija u flash-u (ili u MMC) za čuvanje U-Boot okruženja, definisano u konfiguracionoj datoteci ploče



# Komande varijabli okruženja

## ❖ Komande za manipulisanje varijablama:

❖ `printenv`

Prikazuje sve varijable i njihove vrednosti

❖ `printenv <variable-name>`

Prikazuje vrednost navedene varijable

❖ `setenv <variable-name> <variable-value>`

Postavlja novu vrednost varijable, samo u RAM

❖ `editenv <variable-name>`

Menja vrednost varijable, samo u RAM

❖ `saveenv`

Čuva trenutno stanje okruženja na flash



# Komande varijabli okruženja - Primer

```
u-boot # printenv
baudrate=19200
ethaddr=00:40:95:36:35:33
netmask=255.255.255.0
ipaddr=10.0.0.11
serverip=10.0.0.1
stdin=serial
stdout=serial
stderr=serial
u-boot # printenv serverip
serverip=10.0.0.1
u-boot # setenv serverip 10.0.0.100
u-boot # saveenv
```



# Važne U-Boot promenljive okruženja



- ❖ bootcmd, sadrži komandu koju će U-Boot automatski izvršiti u vreme boot-a nakon isteka konfigurisanog vremena bootdelay, ukoliko proces nije prekinut
- ❖ bootargs, sadrži argumente koji se prosleđuju Linuks kernelu
- ❖ serverip, IP adresa servera na koji će se U-Boot spojiti prilikom izvršavanja komandi vezanih za mrežu
- ❖ ipaddr, IP adresa koju će U-Boot koristiti
- ❖ netmask, mrežna maska za povezivanje sa serverom
- ❖ ethaddr, MAC adresa, može se postaviti samo jednom
- ❖ autostart, ukoliko je postavljeno na yes, U-Boot automatski pokreće sliku koja je učitana u memoriju
- ❖ filesize, veličina datoteke poslednje kopirane u memoriju (kroz tftp, fatload, nand read...)



# Skripte i promenljive okruženja

- ❖ Promenljive okruženja mogu da sadrže manje skripte, da izvršavaju nekoliko komandi i testiraju rezultat komandi.
  - ❖ Korisno za automatizaciju booting ili upgrade procesa
  - ❖ Nekoliko komandi može biti ulančano korišćenjem operatora ;
  - ❖ Testovi se sprovode konstrukcijom

```
if command ; then ... ; else ... ; fi
```
  - ❖ Skripte se izvršavaju korišćenjem `run <variable-name>`
  - ❖ Možete referencirati druge promenljive korišćenjem  
`$(variable-name)`
- ❖ Primer
  - ❖ `setenv mmc-boot 'if fatload mmc 0 80000000 boot.ini; then source; else if fatload mmc 0 80000000 zImage; then run mmc-do-boot; fi; fi'`



# Prenos podataka do odredišne platforme

- ❖ U-Boot je najčešće korišćen za učitavanje, prenos i pokretanje slike kernela, ali takođe omogućava i izmenu slike kernela i korenskog sistema datoteka
- ❖ Datoteke moraju biti razmenjene između odredišne platforme i razvojne radne stanice. Ovo je moguće:
  - ❖ Kroz mrežu ukoliko odredišna platforma ima Ethernet konekciju i U-Boot sadrži rukovalac za Ethernet čip. Ovo je najbrže i najefikasnije rešenje.
  - ❖ Kroz USB flash, ako U-Boot podržava USB kontroler vaše platforme
  - ❖ Kroz SD ili microSD karticu, ukoliko U-Boot podržava MMC kontroler na vašoj platformi
  - ❖ Kroz serijski port



# TFTP



- ❖ Mrežni prenos od razvojne stanice do U-Boot-a na ciljnoj platformi se odvija kroz TFTP
  - ❖ *Trivial File Transfer Protocol*
  - ❖ Donekle sličan FTP-u, ali bez autentifikacije preko UDP-a
- ❖ TFTP server je potreban na razvojnoj stanici
  - ❖ sudo apt-get install tftpd-hpa
  - ❖ Sve datoteke u /var/lib/tftpboot su vidljive kroz TFTP
  - ❖ TFTP klijent je dostupan u tftp-hpa paketu, za testiranje
- ❖ TFTP klijent je integrisan u U-Boot
  - ❖ Konfiguriši ipaddr i serverip promenljive okruženja
  - ❖ Koristi tftp <address> <filename> za učitavanje datoteke



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Upoznavanje sa BASH skriptama



## Zašto? (1/2)

- ❖ Radno znanje Shell Scripting-a je od suštinskog značaja za svakoga ko želi da postane razumno vešt u administraciji sistema
- ❖ Čak i ako nemate potrebe za stvarnim pisanjem skripti, treba ih razumeti
- ❖ Kad se Linuks mašina podigne, izvršava se shell skripta u /etc/rc.d da povrati konfiguraciju sistema i uspostavi usluge



## Zašto? (2/2)

- ❖ Detaljno razumevanje ovih početnih skripti je važno za analizu ponašanja sistema, kao i za modifikaciju istog
  
- ❖ Korisno je znanje i za kreiranje sopstvenih rešenja za prevodenje programskog koda



## Odlike

- ❖ „Zanat skriptinga“ nije teško savladati
- ❖ skripta može biti izgrađena u delovima veličine sekcija
- ❖ postoji samo (relativno) mali skup Shell-specifičnih operatora i opcija da se nauče
- ❖ Sintaksa je jednostavna



# I opet, zašto koristiti bash?

- ❖ Shell skript je brz i prljavi (quick and dirty) metod kreiranja prototipa složene aplikacije
- ❖ Dobijanje čak i ograničenog podskupa funkcionalnosti za rad u scenariju je često korisna prva faza u razvoju projekta
- ❖ Na ovaj način, struktura aplikacije može biti testirana i menjana
- ❖ Glavne zamke mogu biti pronađene pre nastavka kodiranja



## Kada NE koristiti bash (1/2)

- ❖ Resource-intenzivni zadaci, naročito kada je brzina bitan faktor (sortiranje, heširanje, rekurzije)
- ❖ Postupci koji se odnose na teške uslove rada matematičkih operacija, posebno aritmetika u pokretnom zarezu, proizvoljne precizne kalkulacije, ili kompleksni brojevi
- ❖ Kada je potrebna cross-platform portabilnost
- ❖ Kompleksne aplikacije, gde je strukturirano programiranje neophodno (provera tipa promenljivih, funkcija prototipa, itd)



## Kada NE koristiti bash (2/2)

- ❖ Kritične aplikacije od kojih zavisi budućnost (npr. kompanije)
- ❖ Situacije u kojima je važna bezbednost, gde bi trebalo da se garantuje integritet vašeg sistema i štiti od upada, pucanja i vandalizama
- ❖ Ekstenzivan rad sa datotekama
- ❖ Potrebna ugrađena podrška za višedimenzionalne nizove
- ❖ Treba da se generiše / manipuliše grafikom ili sa GUI



# Šta je bash?

- ❖ BASH je akronim za „Bourne-Again shell“
- ❖ Bash je postao defakto standard za shell skripting na većini Uniks sistema
- ❖ Predstavljen će biti, ukratko, uvod u bash skripting



# Primer 1

- ❖ Ništa neobično, naredjane komande
- ❖ Mogu se pozivati jedna po jedna u konzoli
- ❖ Jedna od prednosti je što ne moramo ponovo da kucamo sve ove komande
- ❖ Skripta postaje program ili alat koji možemo po želji da modifikujemo i koristimo za razne aplikacije

```
# Cleanup
# Pokreni kao root.
cd /var/log
cat /dev/null > messages
cat /dev/null > wtmp
echo "Log files cleaned up."
```



## Primer 2

- ❖ Ovo što sad imamo počinje da liči na skriptu, ali može još više da se unapredi

```
#!/bin/bash
# Pravilan header za Bash skriptu.
# Cleanup, verzija 2
# Pokreni kao root, naravno.
# Ubaciti kod ovde da završi izvršavanje ako nije root.
LOG_DIR=/var/log
# Varijable su bolje nego zakucane vrednosti.
cd $LOG_DIR
cat /dev/null > messages
cat /dev/null > wtmp
echo "Logs cleaned up."
exit # Pravilan našin izlaženja iz skripte.
# Čist "exit" (bez parametara) vraća izlazni status
#+ prethodno pozvane komande.
```



# Primer 3



```
#!/bin/bash
# Cleanup, verzija 3
LOG_DIR=/var/log
ROOT_UID=0 # Samo korisnici sa $UID 0 imaju root prava.
LINES=50 # Broj linija koji se čuva.
E_XCD=86 # Ne može da se promeni direktorijum?
E_NOTROOT=87 # Ako nije root, exit greška.
# Pokreni kao root, naravno.
if [ "$UID" -ne "$ROOT_UID" ]
then
echo "Must be root to run this script."
exit $E_NOTROOT
fi
if [ -n "$1" ]
# Testiraj da li imamo prosleđen argument.
then
lines=$1
else
lines=$LINES # Po default-u, ako nije specifirano argumentom
Fi
cd $LOG_DIR
if [ `pwd` != "$LOG_DIR" ] # or if [ "$PWD" != "$LOG_DIR" ]
# Nema u /var/log?
then
echo "Can't change to $LOG_DIR."
exit $E_XCD
fi # Proveri da li si u pravom direktorijumu pre nego što se dira log fajl.
tail -n $lines messages > mesg.temp
mv mesg.temp messages
cat /dev/null > wtmp
echo "Log files cleaned up."
exit 0
```



## Sha-bang (1/4)

- ❖ SHA-Bang (# !) na čelu skripte govori vašem sistemu da je fajl skup komandi koje se daju navedenom komandnom prevodiocu
- ❖ # ! je zapravo dvobajtni magični broj, poseban marker koji označava tip datoteke
- ❖ U ovom slučaju izvršnu shell skriptu



## Sha-bang (2/4)

- ❖ Odmah nakon # ! je putanja
- ❖ To je putanja za program koji tumači komande u tekstu, bilo da je shell, programski jezik, ili uslužni program
- ❖ Komandni prevodilac zatim izvršava komande u tekstu, sa početkom u vrhu (linija nakon Sha-bang linije) i ignoriše komentare



## Sha-bang (3/4)

- ❖ Svaki od header-a koristi različit komandni interpreter
- ❖ Korišćenjem #!/bin/sh podrazumevani Bourne shell čini da će skripta biti portabilna na većini varijanti UNIX-a
- ❖ Čak može biti i portabilna na različitim ne-Linuks sistemima, ali zarad toga bismo morali žrtvovati bash-specifične alate

```
#!/bin/sh  
#!/bin/bash  
#!/usr/bin/perl  
#!/usr/bin/tcl  
#!/bin/sed -f  
#!/bin.awk -f
```



## Sha-bang (4/4)

- ❖ Putanja koja se prosledi nakon # ! mora biti tačna, u suprotnom, jedino što se dobije je "Command not found." kada se pokrene skripta
- ❖ # ! može biti izostavljeno ukoliko skripta koristi samo set generičkih sistemskih komandi, bez internih shell direktiva
- ❖ #!/bin/sh poziva podrazumevani shell interpreter, koji je najčešće /bin/bash na Linuks mašini



# Pokretanje skripte

- ❖ Skripta može biti pokrenuta na više načina:
  - ❖ sh ime\_skripte
  - ❖ bash ime\_skripte
- ❖ Učiniti da skripta bude izvršiva chmod komandom i pokrenuti sa ./ime\_skripte



# Rezime i šta dalje

- ❖ Na primerima smo videli upotrebu:
  - ❖ varijabli
  - ❖ parametara
  - ❖ grananja
  - ❖ Petlji
- ❖ Dalje ćemo se detaljnije dotaći tih tema



# Variable

- ❖ Jako je bitno raspoznavati ime varijable od njene vrednosti
- ❖ variable1 je ime i dodeljena joj je vrednost 23
- ❖ Kad hoćemo da ispišemo tu vrednost, moramo koristiti \$variable1
- ❖ A kad ispisujemo bez \$, što je znak interpreteru da nailazi na varijablu, ispisace običan string
- ❖ Dodata vrednosti mora biti bez razmaka

```
bash$ variable1=23
```

```
bash$ echo variable1  
variable1
```

```
bash$ echo $variable1  
23
```



# Korišćenje navodnika sa varijablama

- ❖ Ubacivanje varijable u dvostruke navodnike ("...") se ne meša sa izmenom varijable
- ❖ Ovo se zove delimično citiranje, ponekad nazivano i "slabo citiranje"
- ❖ Koristeći jednostrukе navodnike ('...') varijabla će se koristiti bukvalno, a ne zamena za njenu vrednost
- ❖ Ovo je puno citiranje, ponekad nazivano i "jako citiranje"



# Varijable - napomene

## ❖ Napomena:

- ❖ `$variable` je zapravo uprošćena forma od `$(variable)`
- ❖ U kontekstu gde je `$variable` sintaksa uzrok greške, duža forma bi mogla da radi
- ❖ Neinicijalizovana varijabla ima vrednost `NULL` – nema dodeljenu vrednost (*NIJE* nula!)
- ❖ Ne zameniti = `i-eq`, koji testira, a ne dodeljuje vrednost!
- ❖ = može biti i dodela vrednost, a i korišćena za proveru, zavisno od konteksta upotrebe



## Vrste varijabli

- ❖ Variable mogu biti lokalne i globalne
- ❖ Lokalne
  - ❖ Žive su samo tokom izvršavanja skripte
- ❖ Globalne
  - ❖ Žive su i pre i posle izvršavanja skripte, ali u skripti može biti promenjena njena vrednost i promena ostaje trajna



## Parametri (1/2)

- ❖ Argumenti prosleđeni skripti iz komandne linije: \$0, \$1, \$2, \$3 . . .
- ❖ \$0 je ime same skripte
- ❖ \$1 je prvi argument, \$2 je drugi, \$3 je treći, i tako dalje. Nakon \$9 argumenti moraju biti zatvoreni u vitičaste zagrade, na primer: \${42}, \${43}, \${159}



## Parametri (2/2)

- ❖ Postoji i indirektna referenca na argumente
- ❖ \$# - # označava broj prosleđenih argumenata
- ❖ Na ovaj način možemo saznati broj argumenata, odnosno da li to zadovoljava uslove naše skripte
- ❖ Napomena:
  - ❖ Na nekim sistemima je potrebno prvo proveriti \$0 argument za ime skripte pa je tek onda moguće pozvati \$# za broj argumenata

```
args=$# # Number of args passed.  
lastarg=${!args}  
# Note: This is an *indirect reference* to $args ...  
# Or: lastarg=${!#}  
# This is an *indirect reference* to the $# variable.  
# Note that lastarg=${!$#} doesn't work.
```



# Petlje (1/4)

- ❖ For petlja:

- ❖ 

```
for arg in [list]
do
    command(s) ...
done
```

- ❖ Ako je do u istoj liniji sa for, moramo posle liste da stavimo ;

- ❖ 

```
for arg in [list] ; do
```



# Petlje (2/4)

## ❖ Primer For petlje:

```
#!/bin/bash
# Listing the planets.
for planet in Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune Pluto
do
echo $planet # Each planet on a separate line.
done
echo; echo
for planet in "Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune Pluto"
# All planets on same line.
# Entire 'list' enclosed in quotes creates a single variable.
# Why? Whitespace incorporated into the variable.
do
echo $planet
done
echo; echo "Whoops! Pluto is no longer a planet!"
exit 0
```



## Petlje (3/4)

- ❖ While petlja:

- ❖ while [condition]  
do  
command(s) . . .  
done

- ❖ Važi isto pravilo za do u istoj liniji sa petljom:

- ❖ while [condition] ; do



# Petlje (4/4)

## ❖ Primer While petlje:

```
#!/bin/bash
var0=0
LIMIT=10
while [ "$var0" -lt "$LIMIT" ]
#           ^
#           ^
# Spaces, because these are "test-brackets" . . .
do
echo -n "$var0 "      # -n suppresses newline.
#           ^
#           Space, to separate printed out numbers.
var0=`expr $var0 + 1` # var0=$((var0+1)) also works.
# var0=$((var0 + 1)) also works.
# let "var0 += 1" also works.
# Various other methods also work.
done
echo
exit 0
```



# Kontrola petlji

- ❖ Break i continue mogu da se koriste u bash petljama
- ❖ Njihova upotreba je skoro identična upotrebi u C-u
- ❖ Break može imati i parametar uz sebe da se naglasi da iz ugnježdenih petlji želimo da izađemo i iz više nivoa petlji
- ❖ Continue takođe može imati parametar, ali continue na taj način preskače  $N$  sledećih iteracija petlje



# If - then - else

❖ if [ condition ]  
then  
    commands  
else  
    commands  
fi

```
#!/bin/bash
if [ $# -ne 5 ]
then
    return 0 # true
else
    return 1 # false
fi
```



# Case (1/2)

❖ **case** "\$variable" **in**  
    "\$condition1" **)**  
    *command...*  
    **;;**  
    "\$condition2" **)**  
    *command...*  
    **;;**  
**esac**

```
#!/bin/bash
# Testing ranges of characters.
echo; echo "Hit a key, then hit return."
read Keypress
case "$Keypress" in
    [[:lower:]] ) echo "Lowercase letter";;
    [[:upper:]] ) echo "Uppercase letter";;
    [0-9] ) echo "Digit";;
    * ) echo "Punctuation, whitespace, or other";;
esac
```



## Case (2/2)

- ❖ Citiranje varijabli nije obavezno, jer se ne događa razdvajanje reči
- ❖ Svaki test linija završava desno zagradom - )
- ❖ Svaki blok uslov završava se dvostrukim tačka-zarezom - ; ;
- ❖ Ako je uslov istinit, onda se pridružene komande izvršavaju i case blok prestaje
- ❖ Čitav case blok završava sa esac (case napisano unazad)



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Upoznavanje sa **MAKE** alatom i **MAKEFILE** datotekama



## Proces prevodenja (1/2)

- ❖ Prevodenje programa koji se sastoji od velikog broja izvornih datoteka i zaglavlja može da bude komplikovano i da oduzme mnogo vremena
- ❖ Kucanje dugačkih komandi za prevodenje koda je skljono greškama



## Proces prevođenja (2/2)

- ❖ Primer: program se sastoji od izvornih datoteka, zaglavlja, biblioteka i njihovih zaglavlja kao na slici

```
app
|   |
|   +-- inc
|   |   |-- app.h
|   |   |-- factorial.h
|   |   |-- math.h
|   |
|   +-- src
|       |-- app.c
|       |-- factorial.c
|       |-- math.c
|
|   +-- lib
|       |-- libwrite.so
|
|   +-- libinc
|       |-- libwrite.h
|
|   +-- build
```

```
gcc -O2 -c -I../inc -I../libinc ../src/app.c -o app.o
gcc -O2 -c -I../inc ../src/factorial.c -o factorial.o
gcc -O2 -c -I../inc ../src/math.c -o math.o

gcc -L../lib -lwrite app.o factorial.o math.o -o
```

- ❖ Ovo se može zameniti jednom složenijom komandom  
`gcc -O2 -I../inc -I../libinc -L../lib ../src/app.c -lwrite ../src/factorial.c ../src/math.c -o app`
- ❖ Drugi način je lakši za korišćenje ali nije efikasan kada se samo nekoliko datoteka promenilo
- ❖ Prvi metod je brži kada se promenilo samo nekoliko datoteka, ali postoji rizik da se ne prevede ponovo neka datoteka u kojoj je došlo do promene



## make alat (1/2)

- ❖ Obično su ulazi projekta izvorne datoteke, a izlaz je jedna ili više izvršnih datoteka ili biblioteka
- ❖ make je alat koji se koristi za efikasno prevođenje programa
- ❖ Automatski detektuje delove programa koji treba da se prevedu i izdaje komande da se prevođenje izvrši
  - ❖ Pokreće prevođenje ukoliko izlaz već ne postoji ili ako je došlo do promene u nekoj datoteci od koje izlaz zavisi
- ❖ make čita instrukcije iz tekstualne datoteke
- ❖ Podrazumevano ime datoteke je `makefile` ili `Makefile`



## make alat (2/2)

- ❖ Opcijom -f možemo iskoristiti bilo koju datoteku za ulazne instrukcije
  - ❖ make -f InstrukcijeZaPrevođenje
- ❖ Opšti oblik sintakse make-a:
  - ❖ make [-f makefile] [opcije] ...  
[ciljevi] ...
- ❖ Razni debag ispisi u toku prevođenja se mogu uključiti opcijom -d
- ❖ Više informacija o opcijama se može naći u priručniku
  - ❖ man make



# makefile

- ❖ Sadrži instrukcije uz pomoć kojih se od ulaza generišu izlazi
- ❖ Tekstualna datoteka
- ❖ Sastoji se od:
  - ❖ eksplisitnih pravila - opisuju kada i kako da se prevede jedna ili više datoteka
  - ❖ implicitnih pravila - opisuje kada i kako da se preveda grupa datoteka na osnovu njihovog imena
  - ❖ definicija promenljivih - definiše tekstualne promenljive koje će kasnije biti zamenjene
  - ❖ direktiva - naređuju izvršavanje akcija dok se analizira makefile
    - ❖ uključivanje drugog makefile-a
    - ❖ uslovno izvršavanje određenog dela
    - ❖ definicija promenljive u više linija
  - ❖ komentara - počinju sa # i traju do kraja linije



# Eksplicitna pravila

- ❖ Pravilo definiše kada i kako da se prevedu datoteke da se dobije odredište pravila
- ❖ Sintaksa:
  - ❖ Odredište: zavisnosti  
[tab] komanda
- ❖ Uglavnom odredište predstavlja izlaz, a zavisnosti ulaz u pravilo
- ❖ make će izvršiti komandu ako
  - ❖ odredište ne postoji
  - ❖ datoteka navedena kao zavisnosti se promenila od poslednjeg prevodenja
  - ❖ komanda se promenila od poslednjeg prevodenja
- ❖ Eksplicitna pravila - eksplicitno data u makefile-u



# Pravljenje objektne datoteke

- ❖ Mora da postoji pravilo gde je objektna datoteka odredište
- ❖ Zavisnosti su izvorne datoteke (i zaglavija) potrebne za pravljenje objektne datoteke
- ❖ Komanda mora da obuhvata poziv prevodioca kako bi se prevele izvorne datoteke  
app.o: app.c app.h math.h factorial.h libwrite.h  

```
gcc -O2 -c -I../inc -I../libinc ../src/app.c -o app.o
```
- ❖ Zavisnosti obuhvataju sva zaglavija korišćena u app.c jer se u slučaju promene u nekom od zaglavija prevodenje mora opet izvršiti



# Pravljenje izvršne datoteke

- ❖ Mora da postoji pravilo gde je izvršna datoteka odredište
- ❖ Zavisnosti su sve objektne datoteke potrebne za pravljenje izvršne datoteke
- ❖ Komanda mora da obuhvata poziv povezivača koji pravi izvršnu datoteku  
app: app.o factorial.o math.o  
`gcc -L..;/lib -Iwrite app.o factorial.o math.o -o app`
- ❖ Ukoliko odredište ne postoji (app) ili se neka od zavisnosti promenila (bilo koja ulazna datoteka) komanda će se izvršiti i izvršna datoteka će biti napravljena



# Promenljive

- ❖ Još jedna bitna komponenta makefile-ova
- ❖ Identifikator definisan u okviru makefile-a koji predstavlja tekstualni string (vrednost promenljive)
- ❖ Prilikom izvršenja pravila promenljive se zamenjuju sa njihovim vrednostima
- ❖ Čine kod čitljivijim i lakšim za konfigurisanje  
CC = gcc  
app.o: app.c app.h math.h factorial.h libwrite.h  
    \${CC} -O2 -c -I../inc -I../libinc ../src/app.c -o app.o
- ❖ Promena prevodioca = promena vrednosti promenljive
- ❖ Vrednost promenljive se može postaviti i:
  - ❖ iz komandne linije - make all CC=/usr/bin/gcc
  - ❖ preko promenljive okruženja - export CC=/usr/bin/gcc



# Korišćenje specijalnih karaktera (vajld-karti)



- ❖ Vajld-karte se koriste za referenciranje više datoteka jednim imenom
  - ❖ Vajld-karte su:
    - ❖ \* : menja bilo koji string od 0 ili više karaktera
    - ❖ ? : menja jedan karakter
    - ❖ [lista\_karaktera] : menja jedan karakter sa nekim iz liste
  - ❖ clean:
    - rm -f \*.o (briše sve objektne datoteke)
    - print: print?.o (sve datoteke sa nazivom print i još tačno jedan karakter će biti zavisnosti)
    - print: print[0-9].o (sve datoteke sa nazivom print i još jednim numeričkim karakterom će biti zavisnosti)
  - ❖ Ukoliko je potrebno da se u imenu iskoristi karakter koji je i vajld-karta pre njega se piše \
  - ❖ print: print\\*.o (samo print\*.o će biti zavisnost)



## wildcard funkcija

- ❖ Razvijanje vajld-karti se automatski obavlja u okviru definicija mete i zavisnosti
- ❖ Šel skripta je zadužena za razvijanje u delu sa komandama (uključujući zavisnosti)
- ❖ Razvijanje se ne izvršava u ostalim delovima - npr. definicija promenljivih
- ❖ Za razvijanje vajld-karti u ovim delovima koristi se wildcard funkcija
- ❖ Sintaksa:
  - ❖ \$(wildcard šablon)
- ❖ Primer
  - ❖ sources=\$(wildcard \*.c \*.h) - sve datoteke sa ekstenzijama .c i .h



# Zamene

- ❖ Zamene se koriste da izvrše određene promene nad vrednošću varijabla
- ❖ Sintaksa:
  - ❖ \$(var:a=b)
  - ❖ \${var:a=b}
    - ❖ U vrednosti var svako a na kraju reči se menja sa b
    - ❖ a mora biti na kraju vrednosti ili praćeno razmakom
- ❖ Primer:
  - ❖ var1:=f1.o f2.o f3.o
  - var2:=\$(var1:.o=.c)
  - ❖ Posle izvršenja zamene var2 ima vrednost f1.c f2.c f3.c



# Implicitna pravila (1/3)

- ❖ Nije uvek neophodno definisati pravilo za svako odredište
- ❖ Za često korišćene standardne procedure postoje implicitna pravila
  - ❖ Npr. postoji pravilo za pravljenje .o od .c korišćenjem CC
- ❖ Moguće je sekvencijalno primeniti implicitna pravila
  - ❖ Npr. od .c napraviti .o, pa iskoristiti povezivač za pravljenje izvršne datoteke
- ❖ Da bi se koristilo implicitno pravilo potrebno je napisati pravilo bez komande ili ne napisati pravilo uopšte
  - ❖ app: app.o math.o factorial.o  
\$(CC) -o app app.o math.o factorial.o
- ❖ Pošto se .o koristi kao zavisnost, a ne postoji eksplicitno pravilo u kom je odredište, traži se odgovarajuće implicitno pravilo



## Implicitna pravila (2/3)

- ❖ Lista implicitnih pravila zavisi od operativnog sistema
- ❖ Uvek dostupna pravila su:
  - ❖ Prevođenje C programa - koristi CC
  - ❖ Prevođenje asemblerskih programa - koristi AS
  - ❖ Povezivanje objektnih datoteka - koristi LD
- ❖ Lista svih implicitnih pravila se može videti pokretanjem komande make -p
- ❖ Zabrana korišćenja implicitnih pravila se može postići na dva načina:
  - ❖ -r parametar
  - ❖ -no-builtin-rules



## Implicitna pravila (3/3)

- ❖ Implicitna pravila koriste predefinisane promenljive
  - ❖ Menjanjem promenljivih utiče se na izvršenje pravila
  
- ❖ Imena programa: Argumenti programa:

❖ AS – asembler	ASFLAGS – argumenti asemblera
❖ AR – arhiver	ARFLAGS – argumenti arhivera
❖ CC – C prevodilac	CFLAGS – argumenti C prevodioca
❖ CPP – C preprocesor	CFLAGS – argumenti C preprocesora
❖ CXX – C++ prevodilac	CXXFLAGS – argumenti C++ prevodioca



# Definicija i redefinicija implicitnih pravila (1/2)



- ❖ Implicitno pravilo se definiše definisanjem šablonu pravila
- ❖ Šablon pravila se razlikuje od običnog pravila po tome što sadrži znak % u okviru odredišta
- ❖ % odgovara podstringu koji ne može biti prazan
  - ❖ %.c - odgovara svakoj .c datoteci
  - ❖ %.o - odgovara svakoj .o datoteci
- ❖ % u zavisnostima odgovara istom podstringu kao i u odredištu
  - ❖ %.o:%.c komanda

(definiše kako se bilo koja .o datoteka pravi od .c datoteke)



# Definicija i redefinicija implicitnih pravila (2/2)



- ❖ Sablon pravila može da ima više od jednog odredišta
  - ❖ make smatra da će izvršavanje komande napraviti sva odredišta
  - ❖ Primer:
    - ❖ %.tab.c %.tab.h:%.y
    - bison -d \$<
- ❖ Implicitno pravilo se poništava definisanjem pravila sa istim odredištem i zavisnostima, ali bez komandi
  - ❖ Primer:
    - ❖ %.o : %.c



# Automatske promenljive

- ❖ make dodeljuje vrednosti automatskim promenljivama kada se pronađe pravilo i počne izvršavanje
- ❖ Vidljive su samo iz komandi
- ❖ Ne mogu se koristiti u definiciji pravila

\$@	Naziv datoteke iz odredišta
\$%	Naziv člana odredišta, kada je odredište član arhive
\$<	Naziv prve zavisnosti
\$?	Naziv svih zavisnosti novijih od odredišta
\$^	Nazivi svih zavisnosti razdvojeni razmacima bez duplikata
\$+	Slično kao \$^ s tim što se duplikati pojavljuju u redosledu u kom su navedeni



# Primeri upotrebe automatskih promenljivih



- ❖ Napravi izvršnu datoteku povezivanjem svih zavisnosti
  - ❖ app: app.o math.o factorial.o  
\$(CC) \$^ -o app
- ❖ Napravi izvršnu datoteku, koja se zove isto kao odredište, povezivanjem svih zavisnosti
  - ❖ app: app.o math.o factorial.o  
\$(CC) \$^ -o \$@
- ❖ Napravi izvršnu datoteku, koja se zove isto kao odredište, prevodenjem prve zavisnosti
  - ❖ math.o : math.c math.h  
\$(CC) \$(CFLAGS) \$< -o \$@



## Lažno odredište (1/2)

- ❖ Korisno je imati komande koje ne prave direktno datoteke, već logički predstavljaju deo procesa prevođenja
  - ❖ clean:  
rm \*.o app
- ❖ Postoje dva problema u vezi sa pravilom napisanim iznad:
  - ❖ konflikt sa postojećim imenima datoteka (ukoliko neko napravi datoteku clean u istom direktorijumu, pravilo se nikad neće izvršiti)
  - ❖ make će tražiti implicitno pravilo
- ❖ Rešenje - korišćenje specijalnog .PHONY odredišta
  - ❖ .PHONY clean:  
clean:  
rm \*.o app



## Lažno odredište (2/2)

- ❖ Moguće je dodati zavisnosti lažnim (.Phony) odredištima
- ❖ Primer - kada makefile pravi više od jedne izvršne datoteke zgodno je napraviti lažno pravilo koje pravi sve izlaze
  - ❖ all: app1 app2 app3
  - .PHONY : all
  - app1: app1.o math.o  
cc -o app1.o math.o
  - app2: app2.o factorial.o  
cc -o app2.o factorial.o
  - app3: app3.o factorial.o math.o  
cc -o app2.o factorial.o math.o
- ❖ Česta lažna odredišta su:
  - ❖ all - pravi sve izlaze
  - ❖ clean - briše sve izvršne i privremene datoteke
  - ❖ install - kopira sve izlaze na predefinisanu lokaciju



# Primer makefile-a

```
SHELL = /bin/sh

OBJS = app.o factorial.o math.o
CFLAGS = -O2
CC = gcc
INCLUDES = -I ../inc -I ../libinc
LIBS = -lwrite
LDFLAGS = -L ../lib

hello:$OBJ
    ${CC} ${CFLAGS} ${LDFLAGS} ${INCLUDES} -o $@ ${OBJS} ${LIBS}

clean:
    rm -f *.o app

%.o: %.c
    ${CC} ${CFLAGS} ${INCLUDES} -c $< -o $@
```



# Univerzitet u Novom Sadu

## Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Uvod u GIT



# Šta je git?

- ❖ Sistem za verzionisanje softvera kao i CVS, SVN, Perforce ili ClearCase
- ❖ Originalno razvijen za razvoj Linuks kernela
  - ❖ Koriste ga i mnogi drugi projekti
    - ❖ U-Boot, GNOME, Buildroot, uClibc i drugi
- ❖ Za razliku od CVS-a i SVN-a, Git je distribuirani sistem
  - ❖ Ne postoji centralni repozitorijum
  - ❖ Svako ima lokalni repozitorijum
  - ❖ Moguće je praviti lokalne brenčeve (i jako važno)
  - ❖ Lako je deliti kod
  - ❖ Odličan za model zajedničkog razvoja softvera koji postoji na projektima otvorenog koda



# Instalacija i podešavanje

- ❖ Git je moguće preuzeti kao paket u Linuks distribuciji
  - ❖ `sudo apt-get install git`
- ❖ Sve se radi preko `git` komande
  - ❖ Git ima mnogo komandi koje se pozivaju sa `git <komanda>`
    - ❖ Primeri komandi: `clone`, `checkout`, `branch` i dr.
  - ❖ Pomoć u vezi komande
    - ❖ `git help <komanda>`
- ❖ Podešavanje imena i adrese elektronske pošte
  - ❖ Koriste se u svakom komitu
  - ❖ `git config --global user.name <ime>`
  - ❖ `git config --global user.email <e-mail>`



# Kloniranje repozitorijuma

- ❖ Na početku rada na projektu potrebno je klonirati repozitorijum
- ❖ CVS i SVN koriste checkout operaciju za dobavljanje radne verzije projekta (poslednja verzija)
- ❖ Sa gitom dobija se potpuna kopija repozitorijuma, uključujući i istoriju, što omogućava izvršavanje većine operacija u offline režimu
- ❖ Kloniranje Linus Torvaldsovog Linuks repozitorijuma
  - ❖ `git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git`
  - ❖ `git://` je specijalni git protokol; Većini repozitorijuma se može pristupiti preko `http://` protokola, ali je sporije
  - ❖ Nakon kloniranja u `linux/` direktorijumu se nalaze repozitorijum i radna verzija master brenča



# Pregled istorije

## ❖ git log izlistava sve komite, najnoviji je prvi

- ❖ commit 4371ee353c3fc41aad9458b8e8e627eb508bc9a3

Author: Florian Fainelli [florian@openwrt.org](mailto:florian@openwrt.org)

Date: Mon Jun 1 02:43:17 2009 -0700

MAINTAINERS: take maintainership of the cpmac Ethernet driver

This patch adds me as the maintainer of the CPMAC (AR7)

Ethernet driver.

Signed-off-by: Florian Fainelli [florian@openwrt.org](mailto:florian@openwrt.org)

Signed-off-by: David S. Miller [davem@davemloft.net](mailto:davem@davemloft.net)

## ❖ git log –p izlistava komite sa odgovarajućim razlikama

## ❖ Istorija u gitu nije linearna kao u CVS-u ili SVN-u

- ❖ Prikazana je kao graf komita

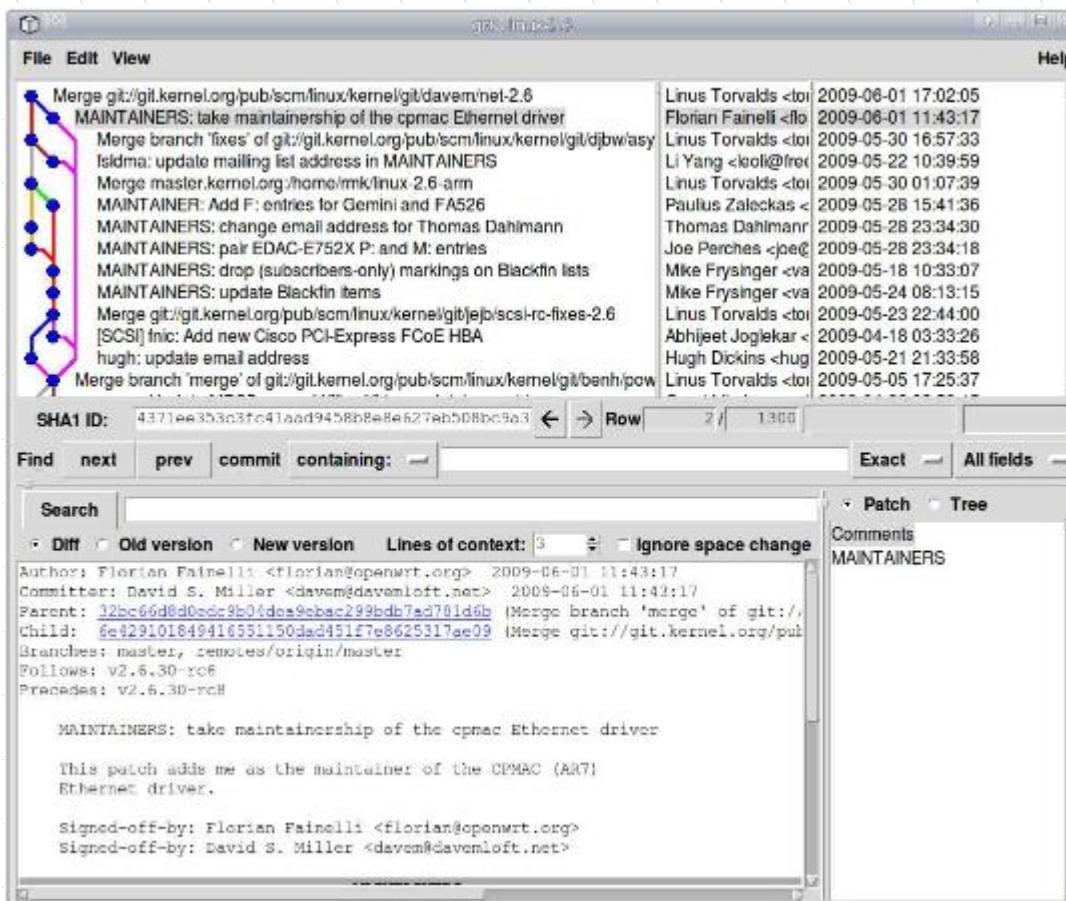
- ❖ Malo je teže za razumevanje na početku

- ❖ Ovo omogućava moćne funkcionalnosti gita (distribuiranost, brenčevi, spajanje komita)



# Grafički pregled istorije - gitk

- ❖ gitk je grafički alat za pregled istorije git repozitorijuma
- ❖ Instalira se preko gitk paketa





# Grafički pregled istorije - cgit



❖ cgit je web sprega za git

index : kernel/git/torvalds/linux.git

Linux kernel source tree

master | switch Linus Torvalds

summary refs log tree commit diff stats log msg search

author Ezequiel Garcia <ezequiel.garcia@free-electrons.com> 2013-11-14 21:25:28 (GMT)  
committer Brian Norris <computersforpeace@gmail.com> 2014-01-03 19:22:12 (GMT)  
commit 770f205e279744ef3f327cc3e7eb378046311373 (patch)  
tree 545d7b8d0e6c225efdb0bc78fe59f06bd856780  
parent 56204d057ac9e1944ec581fbff748dcabdb9d50 (diff)

**mtd: nand: pxa3xx: Add bad block handling**

Add support for flash-based bad block table using Marvell's custom in-flash bad block table layout. The support is enabled via 'flash\_bbt' platform data or device tree parameter.

Signed-off-by: Ezequiel Garcia <ezequiel.garcia@free-electrons.com>  
Tested-by: Daniel Mack <conque@gmail.com>  
Signed-off-by: Brian Norris <computersforpeace@grail.com>

**Diffstat**

-rw-r--r-- Documentation/devicetree/bindings/mtd/pxa3xx-nand.txt	2
-rw-r--r-- drivers/mtd/nand/pxa3xx_nand.c	37
-rw-r--r-- include/linux/platform_data/mtd/nand_pxa3xx.h	3

3 files changed, 42 insertions, 0 deletions

```
diff --git a/Documentation/devicetree/bindings/mtd/pxa3xx-nand.txt b/Documentation/devicetree/bindings/mtd/pxa3xx-nand.txt
index bed8390..86e0a50 100644
--- a/Documentation/devicetree/bindings/mtd/pxa3xx-nand.txt
+++ b/Documentation/devicetree/bindings/mtd/pxa3xx-nand.txt
@@ -15,6 +15,8 @@ Optional properties:
 - marvell,nand-keep-config: Set to keep the NAND controller config as set
 by the bootloader
 - num-cs: Number of chipselect lines to use
+- nand-on-flash-bbt: boolean to enable on flash_bbt option if
+not present false
```

diff options context: 3 space: include mode: unified



# Ažuriranje repozitorijuma

- ❖ Klonirani repozitorijum će se vremenom menjati
- ❖ Potrebno je ažurirati lokalnu verziju da bi promene postale vidljive
  - ❖ git pull
- ❖ Ispod haube obavlja dve radnje
  - ❖ Dobavljanje novih promena na udaljenom repozitorijumu
    - ❖ git fetch
  - ❖ Spajanje promena u trenutni brenč
    - ❖ git merge



# Tagovi

- ❖ Lista postojećih tagova
  - ❖ **git tag -l**
- ❖ Dobavljanje sadržaja određenog taga
  - ❖ **git checkout <ime taga>**
- ❖ Lista promena između taga i poslednje dostupne verzije
  - ❖ **git log v2.6.30..master**
- ❖ Lista promena sa razlikama u određenoj datoteci između dva taga
  - ❖ **git log -p v2.6.29..v2.6.30 MAINTAINERS**
- ❖ Sa gitk
  - ❖ **gitk v2.6.30..master**



# Brenčevi

- ❖ Za početak rada najbolje je napraviti novi brenč
  - ❖ Postoji samo u lokalnu, niko drugi ga ne vidi
  - ❖ Brzo je
  - ❖ Omogućava podelu rada na različite celine
  - ❖ Omogućava isprobavanje nove funkcionalnosti i odbacivanje ukoliko nije zadovoljavajuće
  - ❖ Nije skupa operacija (i ako je zadatak mali i brzo se završi treba napraviti brenč)
- ❖ Za razliku od drugih sistema za verzionisanje softvera, git preporučuju korišćenje brenčeva



# Brenčevi

- ❖ Kreiranje brenča
  - ❖ **git branch <ime brenča>**
- ❖ Prelazak na brenč
  - ❖ **git checkout <ime brenča>**
- ❖ Kreiranje i prelazak zajedno
  - ❖ **git checkout -b <ime brenča>**
- ❖ Lista lokalnih brenčeva
  - ❖ **git branch**
- ❖ Lista svih brenčeva, uključujući i udaljene
  - ❖ **git branch -a**



# Pravljenje izmena

- ❖ Izmenite datoteku u tekst editoru
- ❖ Provera statusa radne kopije
  - ❖ **git status**
- ❖ Git poseduje sposobnost indeksiranja koja omogućava pripremanje komita pre samog komitovanja; omogućava komitovanje samo dela umesto kompletnih imena
- ❖ Nad svakom izmenjenom datotekom
  - ❖ **git add <ime datoteke>**
- ❖ Komanda za komitovanje (ne zahteva da je korisnik na mreži)
  - ❖ Linuks zahteva potpisivanje izmena sa **-s** opcijom
  - ❖ **git commit -s**
- ❖ Ukoliko sve izmene treba da budu komitovane
  - ❖ **git commit -as**



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Pregled kernela



# Sadržaj



- ❖ Pregled kernela
  - ❖ Odlike Linuksa
  - ❖ Kod kernela
  - ❖ Podistemi kernela
  - ❖ Proces razvoja i način davanja verzija Linuksa
  - ❖ Legalni problemi
  - ❖ Korisnička sprega kernela



Odlike Linuksa

# PREGLED KERNELA



# Verzija Linuksa – 2.6

## ❖ Linuks 2.6

- ❖ Linuks 2.6.0 je objavljen zvanično u decembru 2003. godine
- ❖ Dodato je mnogo novih funkcionalnosti i novih drajvera od tada i i dalje se dodaju
- ❖ Za raniju verziju, 2.4, je jako teško pronaći podršku za drajvere za novije fizičke arhitekture
- ❖ Od verzije 3.0 menja se sistem verzionisanja, no osnova se oslanja na verziju 2.6
- ❖ Na ovom kursu bavićemo se verzijama Linuksa 4.4 i naviše



# Glavne odlike Linuks kernela

- ❖ Portabilnost i podrška za fizičke arhitekture
- ❖ Skalabilnost
  - ❖ Može da se pokrene na super računarima, kao i na jako malim uređajima (4 MB RAM-a je sasvim dovoljno)
- ❖ Podržava standarde i interoperabilnost
- ❖ Izdašna podrška za rad sa mrežama
- ❖ Sigurnost
  - ❖ Ne može da sakrije svoje mane jer kod pregleda veliki broj stručnjaka
- ❖ Stabilnost i pouzdanost
- ❖ Modularnost
  - ❖ Može da uključi potrebnu funkcionalnost u toku rada
- ❖ Lak za programiranje
  - ❖ Može da se uči iz postojećeg koda
  - ❖ Postoji puno korisnih resursa na internetu



# Podržane fizičke arhitekture

- ❖ Pogledati arch/ direktorijum u izvornom kodu kernela
- ❖ Najmanje: 32-bitni procesori, sa ili bez jedinice za upravljanje memorijom i podrška za *gcc*
- ❖ 32-bitne arhitekture (iz arch/)
  - ❖ alpha, arm, avr32, cris, frv, h8300, i386, m32r, m68k, m68knommu, mips, parisc, powerpc, ppc, s390, sh, sparc, um, v850, xtensa
- ❖ 64-bitne arhitekture
  - ❖ ia64, mips, powerpc, sh64, sparc64, x86\_64
- ❖ Za detalje, pogledati arch/<arch>/Kconfig, arch/<arch>/README ili Documentation/<arch>/



Kernel kod

# PREGLED KERNELA



# Implementiran u C-u

- ❖ Implementiran je u C-u kao i svi ostali Uniks sistemi
  - ❖ C je i kreiran da bi se prvi Uniks sistemi implementirali
- ❖ Korišćeno je i malo asemblera
  - ❖ CPU i inicijalizacija mašine, izuzeci i kritične rutine biblioteka
- ❖ Pogledati <http://www.tux.org/lkml/#s15-3> za razlog zašto se ne koristi C++
  - ❖ Glavni razlog je to što kernel zahteva efikasan kod



# Programski jezici u izvornom kodu kernela

Linuks  
kernel  
2.6.32  
sadrži  
sledeće  
programske  
jezike:

- ansic: 7849119 (96.55%)
- asm: 225129 (2.77%)
- xml: 33008 (0.41%)
- perl: 9015 (0.11%)
- sh: 3745 (0.05%)
- cpp: 3366 (0.04%)
- yacc: 2964 (0.04%)
- lex: 1824 (0.02%)
- python: 701 (0.01%)
- lisp: 218 (0.00%)
- pascal: 116 (0.00%)
- awk: 109 (0.00%)
- sed: 30 (0.00%)



## Prevodi se sa GNU C

- ❖ GNU C ekstenzije su potrebne da bi se preveo kernel
  - ❖ Ne može da se koristi bilo koji ANSI C prevodilac
  - ❖ Mogu da se koriste i Intel i Marvell prevodioci (za njihove platforme) koji se identifikuju kao GNU prevodioci
- ❖ Neke GNU C ekstenzije koje se koriste u kernelu su:
  - ❖ Inline C funkcije
  - ❖ Inline assembler
  - ❖ Organizacija članova struktura u bulo kom redu (ANSI C99)
- ❖ Potreban je bar gcc 3.2



# Optimizacija koda

- ❖ Može se „pomoći“ prevodiocu za optimizaciju koda
- ❖ Korišćenje `likely` i `unlikely` ključnih reči
  - ❖ `#include/linux/compiler.h`

## ❖ Primer:

```
if (unlikely(err))  
{  
    ...  
}
```

- ❖ GNU C prevodilac će učiniti kod bržim za najčešći slučaj (most `likely`)
- ❖ Koristi se na mnogo mesta u kernel kodu
- ❖ Ne zaboravite da koristite ove ključne reči



# Ne postoji C biblioteka

- ❖ Kernel mora biti samostalan i ne sme da koristi kod iz korisničkog prostora (user-space)
  - ❖ Korisnički prostor je implementiran nad servisima kernela
  - ❖ Kernel kod mora da ima svoje implementacije C biblioteke
- ❖ Standardne funkcije iz C biblioteke se ne mogu koristiti
  - ❖ printf(), memset(), malloc()...
- ❖ Na sreću u kernelu postoje slične C funkcije za korišćenje
  - ❖ printk(), memset(), kmalloc()...



# Koji endian se koristi?

- ❖ Linuks podržava i big i little endian arhitekture
- ❖ Svaka arhitektura definiše `__BIG_ENDIAN` ili `__LITTLE_ENDIAN` u `<asm/byteorder.h>`
  - ❖ Može biti konfigurisan da podržava oba na nekim platformama
- ❖ Da bi kod bio portabilan, kernel sadrži makroe za konverziju (navedeni su najkorisniji):
  - ❖ `u32 cpu_to_be32(u32);` // CPU byte order to big endian
  - ❖ `u32 cpu_to_le32(u32);` // CPU byte order to little endian
  - ❖ `u32 be32_to_cpu(u32);` // Big endian to CPU byte order
  - ❖ `u32 le32_to_cpu(u32);` // Little endian to CPU byte order



# Način kodiranja u kernelu

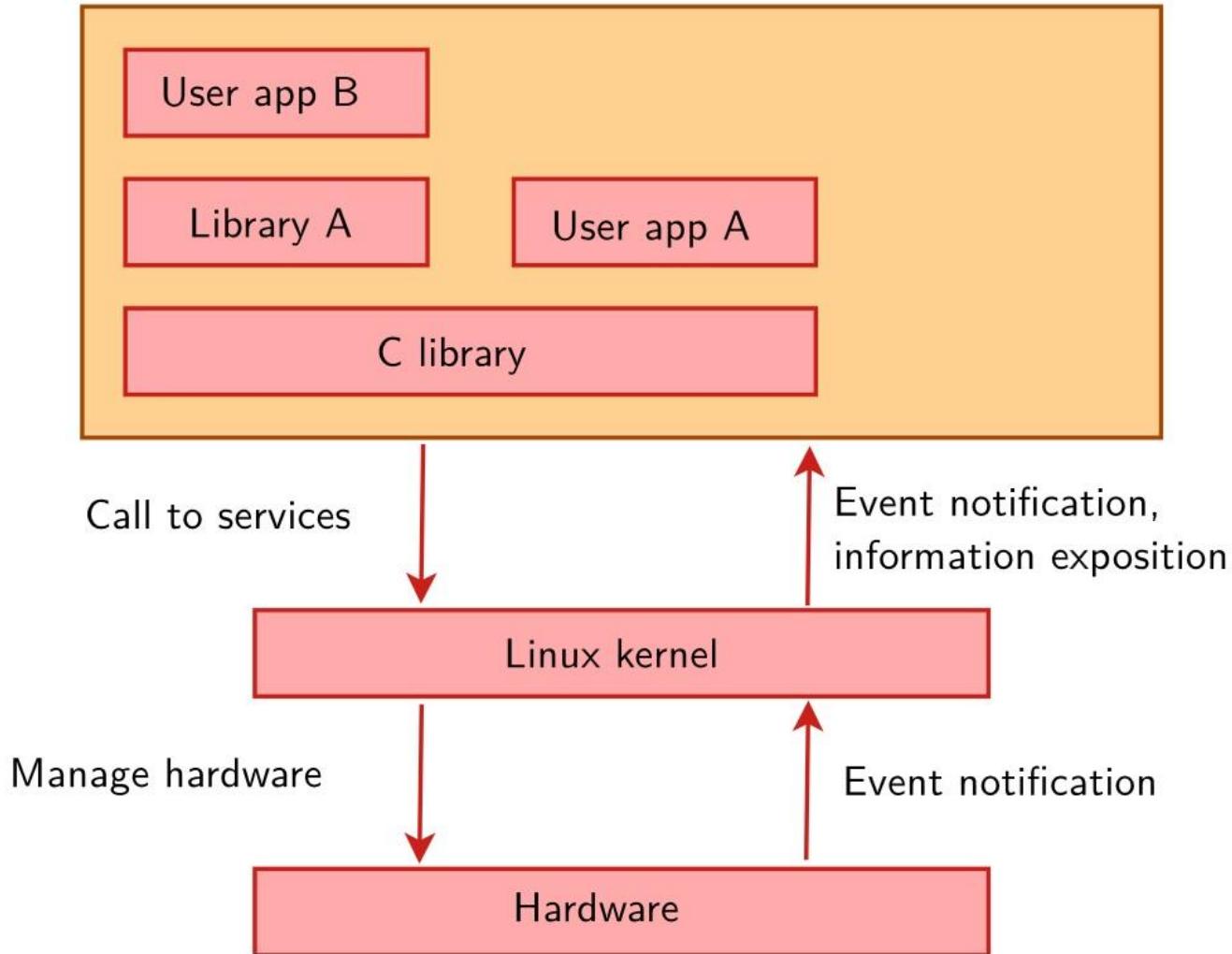
- ❖ Nikad se ne koriste brojevi sa pokretnim zarezom
  - ❖ Kod može biti pokrenut na procesoru koji nema podršku za brojeve sa pokretnim zarezom
  - ❖ Brojevi sa pokretnim zarezom mogu biti simulirani u kernelu, ali je to jako spor proces
- ❖ Svi simboli se definišu kao statični, osim onih koji se eksportuju



Podsistemi kernela

# PREGLED KERNELA

# Arhitektura kernela



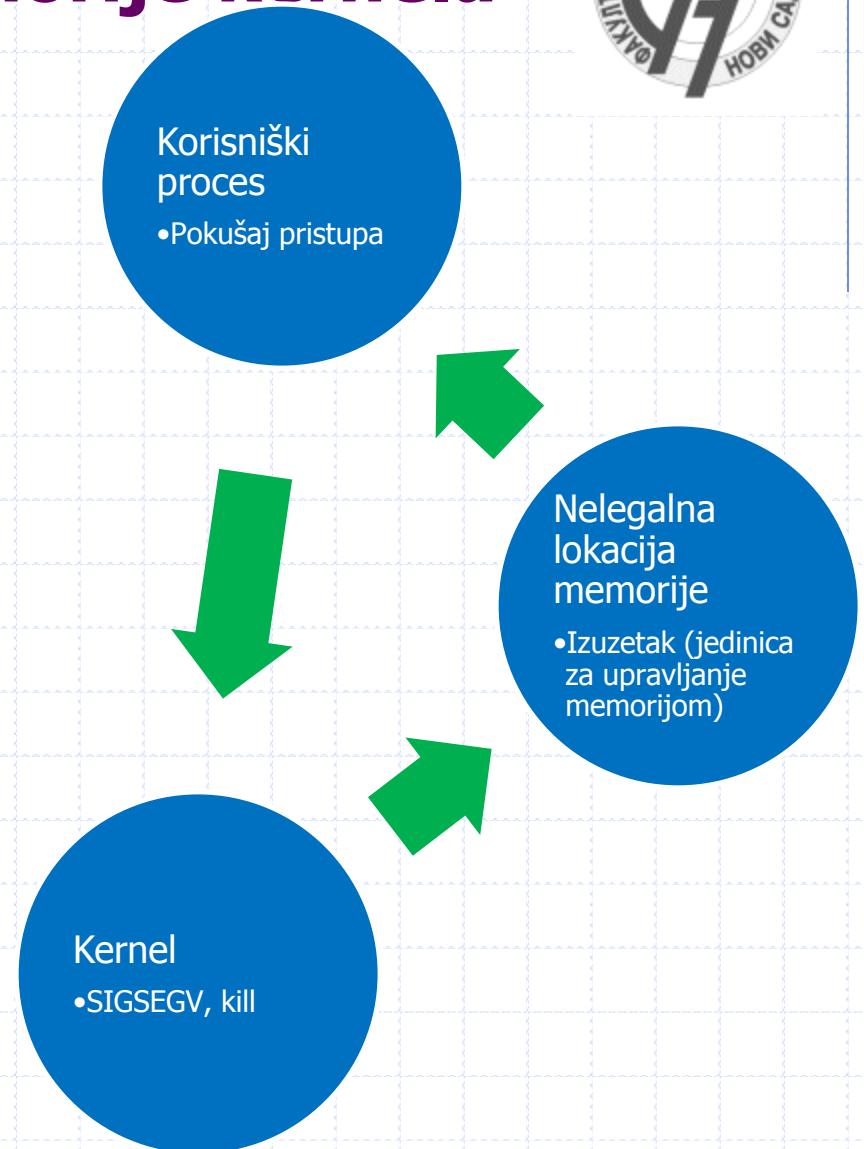


# Arhitektura kernela

- ❖ Korisnički prostor
  - ❖ Aplikacija1, aplikacija2, ...
  - ❖ C biblioteka
- ❖ Kernel prostor
  - ❖ Sprega za sistemske pozive koja komunicira sa C bibliotekom
  - ❖ Upravljanje procesima, memorijom i uređajima, rad sa mrežom, podrška za sistem datoteka, tipovi sistema datoteka, kod za upravljanje fizičkom arhitekturom (drajveri)
- ❖ Fizička arhitektura
  - ❖ CPU, RAM, ulazno-izlazni uređaji, mrežne kartice, hard diskovi

# Ograničenja memorije kernela

- ❖ Ko može da nadgleda kernel?
  - ❖ Ne postoji zaštita memorije
    - ❖ Pristup nelegalnoj memoriji rezultira najčešće fatalnim greškama
- ❖ Veličina steka je zaključana (4 ili 8 KB)
  - ❖ Za razliku od korisničkog prostora, ne može se uvećavati
- ❖ Memorija kernela se ne može menjati (memory swapping)





Proces razvoja i način davanja verzija Linuksa

# PREGLED KERNELA



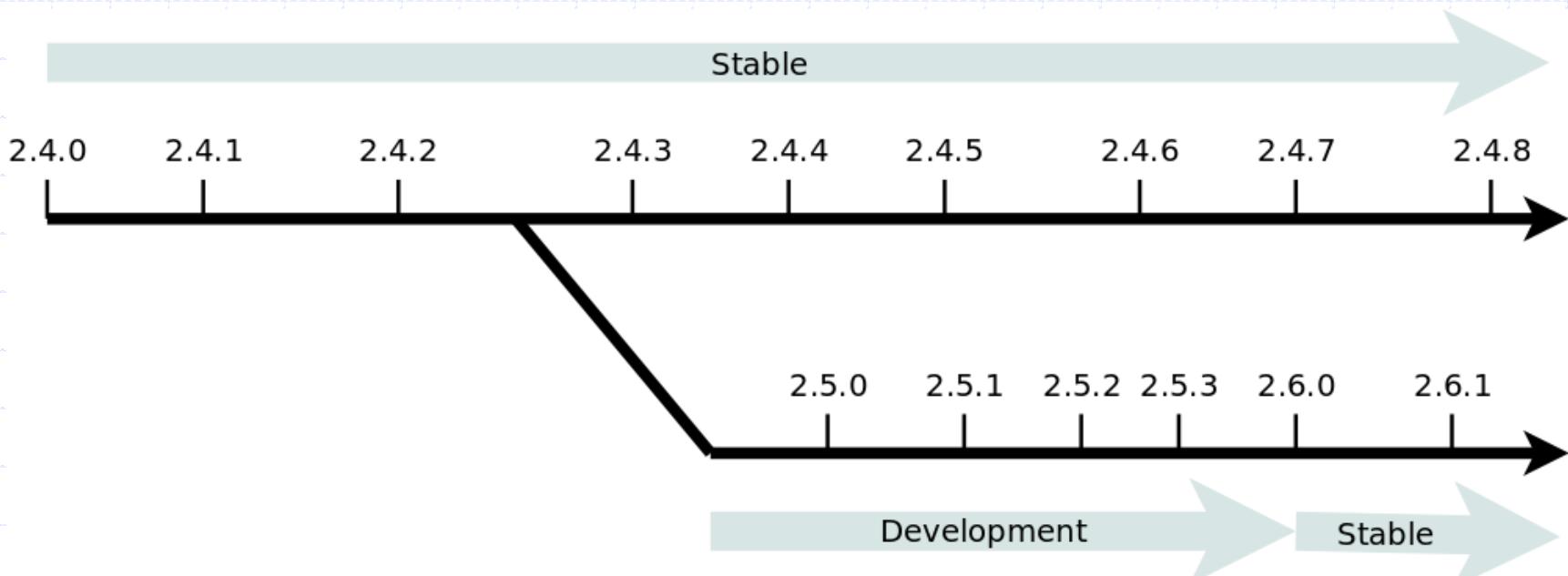
## Sve do 2.6 verzije (1/2)

- ❖ Jedna stabilna velika grana svake 2 ili 3 godine
  - ❖ Identificuje se parni srednjim brojem
  - ❖ Primeri: 1.0.x, 2.0.x, 2.2.x, 2.4.x
- ❖ Jedna grana za razvijanje programske podrške gde se dodaju nove funkcionalnosti i velike izmene
  - ❖ Identificuju se neparnjim srednjim brojem
  - ❖ Primeri: 2.1.x, 2.3.x, 2.5.x
  - ❖ Posle nekog vremena, razvojna verzija postaje nova bazna verzija za granu sa stabilnim verzijama
- ❖ Manja izdanja se daju s vremena na vreme
  - ❖ Primeri: 2.2.23, 2.5.12, itd.



## Sve do 2.6 verzije (2/2)

- ❖ U relanosti, postoji mnogo više manjih verzija unutar velike grane i grane za razvoj





## Izmene nakon Linuksa 2.6 (1/2)

- ❖ Od verzije 2.6.0, kernel programeri mogu da unose mnogo novih funkcionalnosti jednu po jednu stalnim tempom, bez pravljenja promena koje bi ometale postojeće podsisteme.
- ❖ Od tada ne postoji potreba za pravljenje nove grane za razvoj koja bi značajno narušila kompatibilnost sa stabilnom granom.
- ❖ Zahvaljujući ovome, više mogućnosti je omogućeno korisnicima bržim tempom

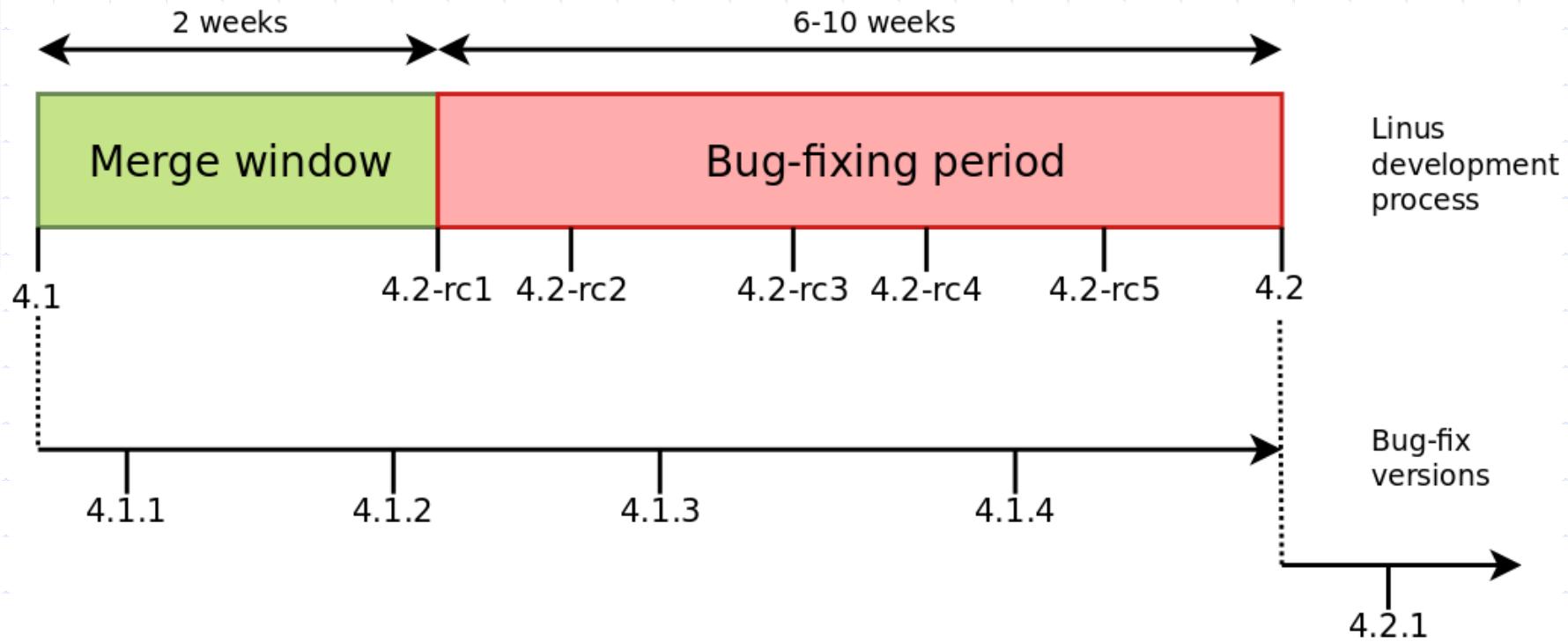


## Izmene nakon Linuks 2.6 (2/2)

- ❖ Od 2003. do 2011. i verzije 2.6.14, dogovoren je novi model razvoja i verzije su imenovane kao 2.6.x
- ❖ Linuks 3.0 je objavljen u julu 2011
- ❖ Linuks 4.0 je objavljen u aprilu 2015
  - ❖ Ne postoji izmena u modelu razvoja, samo u shemi imenovanja verzija
  - ❖ Zvanične verzije kernela se imenuju kao x.y  
(3.0, 3.1, 3.2, ..., 3.19, 4.0, 4.1, itd.)
  - ❖ Stabilne verzije se imenuju kao x.y.z  
(3.0.2, 4.2.7, itd.)
  - ❖ U suštini, uklonjena je samo jedna cifra u poređenju sa prethodnim sistemom



# Novi model razvoja





## Novi model razvoja - detalji

- ❖ Nakon izdavanja npr. verzije 4.x, otvara se dvonedeljni prozor za unošenje izmena i za to vreme se unose najveće izmene.
- ❖ Prozor se zatvara izdavanjem testne verzije 4.(x+1)-rc1
- ❖ Tada počinje period ispravke grešaka koji traje 6-10 nedelja
- ❖ Tokom perioda popravke grešaka, u redovnom intervalima, izdaju se testne verzije 4.(x+1)-rcY
- ❖ Kada se izmene smatraju dovoljno stabilnim, kernel 4.(x+1) se izdaje i proces počinje ponovo



# Više stabilnosti za kernel izvorni kod



- ❖ Problem: greške i sigurnosne popravke se izdaju samo za skorije stabilne verzije kernela
- ❖ Neki ljudi zahtevaju noviji kernel, ali sa dugoročnom podrškom za sigurnosne zakrpe
- ❖ Dugoročna podrška može da se dobije od komercijalnog snabdevača ugrađenim Linuksom
- ❖ Mogu da se koriste kodovi kernela koji se koriste u Ubuntu dugoročnoj podršci (5 godina besplatnih sigurnosnih ažuriranja)
- ❖ Na kernel.org stranici postoje „long term“ izdanja kernela koji imaju podršku na 2 ili 3 godine
- ❖ <http://kernel.org> stranica pokazuje koje verzije će biti podržane još neko vreme (do 2 ili 3 godine) i koje više neće biti podržane ("EOL: End Of Life")

mainline:	<b>4.4-rc4</b>	2015-12-06
stable:	<b>4.3.2</b>	2015-12-10
stable:	<b>4.2.7</b>	2015-12-09
longterm:	<b>4.1.14</b>	2015-12-09
longterm:	<b>3.18.24</b>	2015-10-31
longterm:	<b>3.14.58</b>	2015-12-09
longterm:	<b>3.12.51</b>	2015-11-25
longterm:	<b>3.10.94</b>	2015-12-09
longterm:	<b>3.4.110</b>	2015-10-22
longterm:	<b>3.2.74</b>	2015-11-27
longterm:	<b>2.6.32.69</b>	2015-12-05
linux-next:	<b>next-20151211</b>	2015-12-11



# Nema stabilnog Linuks internog API-ja (1/2)



- ❖ Eksterni API se ne sme menjati (sistemske pozive, /proc, /sys) jer mogu da prekinu izvršavanje postojećih programa
  - ❖ Nove mogućnosti se mogu dodavati, ali treba zadržati kompatibilnost sa starim verzijama bar na godinu ili više
- ❖ Interni kernel API može da ima izmene između dva 2.5.x izdanja
  - ❖ Drajver sa jedne verzije ne mora da radi na nekoj drugoj verziji
- ❖ Kada neko menja interni kernel API, mora takođe da ažurira sav kod koji koristi taj deo koda
  - ❖ Nište ne sme biti pokvareno novom izmenom



# Nema stabilnog Linuks internog API-ja (2/2)



## ❖ USB primer:

- ❖ Linuks je ažuriraio svoj USB interni API bar 3 puta (popravke, sigurnosni problemi, podrška za brze uređaje) i sada ima najbrže USB brzine
- ❖ Windows je morao da napiše ponovo svoj USB stek 3 puta, ali zbog zatvorenog izvornog koda, binarni drajveri ne mogu da se ažuriraju i mora da se zadrži komaptibilnost sa starim verzijama što dovodi do mnogo veće cene razvoja (razvoj, sigurnost, stabilnost, performanse)



# Šta ima novo u svakom novom izdanju? (1/2)



- ❖ Zvanična lista izmena za svako izdanje Linuksa je samo velika lista jedinstvenih zakrpa
- ❖ Jako je teško pronaći ključne izmene i doći do globalne slike od svih tih jedinstvenih zakrpa



# Šta ima novo u svakom novom izdanju? (2/2)



- ❖ Na sreću, postoji rezime svih ključnih izmena sa dovoljno detalja na <http://wiki.kernelnewbies.org/LinuxChanges>
- ❖ Za svako novo izdanje kernela, mogu se videti izmene u internom API-ju kernela na  
<http://lwn.net/Articles/2.6-kernel-api/>
- ❖ Šta dalje? U Documentation/feature-removal-schedule.txt se mogu videti sve odlike, podsistemi i API-ji koji su planirani za uklanjanje
  - ❖ Objavljuje se godinu dana pre



Legalni problemi

# PREGLED KERNELA



## Linuks licenca

- ❖ Ceo Linuks izvorni kod je besplatna programska porška i izdaje se pod GNU General Public Licence version 2 (GPL v2)
- ❖ Za Linuks kernel, to implicira sledeće:
  - ❖ Kada god se dobije ili kupi uređaj sa Linuksom na njemu, treba da se dobije i ceo Linuks izvorni kod sa pravima za proučavanje, izmenu i deljenje koda
  - ❖ Kad se proizvodi uređaj zasnovan na Linuksu, mora da se izda i izvorni kod primaocu, sa istim pravima, bez restrikcija



# Ograničenja licenciranja Linuks kernela



- ❖ Ograničenja u vremenu izdavanja (bez prethodnih ograničenja)
  - ❖ Za svaki uređaj mora da se izda kod krajnjem korisniku. Ne mora nikom drugom da se izda kod.
  - ❖ Prema GPL, sav izvedeni rad koji je pod GPL mora biti izdat pod GPL
  - ❖ Oko Linuks drajvera postoji siva zona
  - ❖ Vlasnički drajveri su sve manje tolerisani
  - ❖ Vlasnički drajveri ne smeju biti statički prevedeni u kernelu
  - ❖ Nije dozvoljeno koristiti kod od drugih kernel drajvera (GPL) u vlasničkom drajveru



# Prednosti GPL drajvera

- ❖ Sa strane kreatora drajvera
  - ❖ Ne mora da se piše drajver ispočetka
  - ❖ Dobija se besplatna podrška, pregled koda i testiranje od strane zajednice, dok vlasnički drajveri ne dobiju ništa
  - ❖ Vaši drajveri mogu biti besplatno prosleđeni (uglavnom u distribucijama Linuksa)
  - ❖ Zatvoren kod drajvera često podržava samo jednu verziju kernela
- ❖ Korisnici i zajednica kreiraju pozitivnu sliku o kompaniji i lakše je unajmiti talentovane inženjere
- ❖ Ne mora da se daje binrani drajver za svaku verziju kernela i zakrpe
- ❖ Drajveri imaju privilegije pristupa, te se mora paziti da drajver ne bude sigurnosni rizik
- ❖ Drajveri mogu biti statički prevedeni u kernel



# Prednosti ugrađenih kernel drajvera



- ❖ Prednosti kad imate drajver u glavnoj grani kernel koda
  - ❖ Kad je prihvaćen kod u glavnu granu, održavaju ga ljudi koji prave izmene
  - ❖ Održavanje ne košta ništa, kao ni bezbednosne zatvare i poboljšanja
  - ❖ Korisnici lako pristupaju vešem kodu
  - ❖ Mnogo ljudi pregleda vaš kod



# Legalni vlasnički Linuks drajveri (1/2)



- ❖ Zaobilazi se GPL tako što se kreira GPL wrapper
- ❖ Vlasnički deo ne dovodi do prekida kad se prevodi ili ažurira kernel i/ili drajver.
- ❖ Vlasnički drajver ne može biti smatrani izvedenim radom
- ❖ Međutim, kernel je monolitan, te deo vlasničkog dela pripada jednom izvršivom delu
- ❖ Ovo je kontroverzna tema



# Legalni vlasnički Linuks drajveri (2/2)



- ❖ 2 primera:
  - ❖ Nvidia drajver za grafičke kartice
  - ❖ Podrška za bežične mrežne kartice koje koriste Windows drajvere
    - ❖ NdisWrapper projekat implementira Windows kernel API i NDIS API u Linuks kernelu
    - ❖ Korisno je za korišćenje kartica gde ne postoje specifikacije
- ❖ Mane
  - ❖ Postoje problemi kod održavanja
  - ❖ Problemi kod performansi (overhead wrapper-a, nema optimizacija)
  - ❖ Sigurnosni problemi (drajveri imaju pune privilegije)
  - ❖ Ostali problemi



Korisnička sprega kernela

# PREGLED KERNELA



# Virtuelni sistemi datoteka (1/2)

- ❖ Virtuelni sistemi datoteka omogućavaju pristup korisničkom prostoru ka sistemskim i kernel informacijama
- ❖ Dozvoljavaju aplikacijama da pristupe direktorijumima i datotekama koje ne postoje realno u memoriji, na disku. Oni su napravljeni i kernel ih osvežava u toku rada
- ❖ Dva najvažnija virtuelna sistema datoteka su
  - ❖ proc, najčešće mauntovan na /proc:  
Informacije vezane za operativni sistem (procesi, memorija, menadžment...)
  - ❖ sysfs, najčešće mauntovan na /sys:  
Prikaz sistema kao seta uređaja i magistrala.  
Informacije o ovim uređajima.



# Virtuelni sistemi datoteka (2/2)

- ❖ Postavka /proc
  - ❖ sudo mount -t proc none /proc
- ❖ Postavka /sys
  - ❖ sudo mount -t sysfs none /sys
- ❖ sysfs i proc – tip fajl sistema
- ❖ none – uređaj ili slika fajl sistema
- ❖ /proc i /sys – tačke postavke



# Sprega za kernel korisnički prostor



- ❖ Primeri:
- ❖ /proc/cpuinfo: informacije o procesoru
- ❖ /proc/meminfo: stanje memorije
- ❖ /proc/version: verzija kenela
- ❖ /proc/cmdline: komandna linija kernela
- ❖ /proc/<pid>/environ: okruženje poziva
- ❖ /proc/<pid>/cmdline: komandna linija procesa
- ❖ I još mnogo drugih



# Dokumentacija sprege korisničkog prostora



- ❖ Dosta dokumentacije o /proc sprezi se može naći komadnom man proc



# Drajveri uređaja korisničkog prostora (1/2)



- ❖ Moguće je implementirati drajvere samo u korisničkom prostoru
  - ❖ Takvim drajverima samo treba pristup uređajima kroz minimalne generičke kernel drajvere
  - ❖ Primeri:
    - ❖ Drajveri za štampač i skener
    - ❖ X drajveri: drajveri niskog nivoa kernel drajvera + X drajveri korisničkog prostora



# Drajveri uređaja korisničkog prostora (2/2)



## ❖ Prednosti

- ❖ Nema potrebe za znanjem kodiranja u kernelu
- ❖ Drajveri mogu biti napisani u bilo kom jeziku
- ❖ Mogu biti vlasnički
- ❖ Kod može biti prekinut i dabagovan, ne može da stopira kernel
- ❖ Može da koristi aritmetiku pokretnog zareza
- ❖ Manje kernel kompleksnosti

## ❖ Mane

- ❖ Komplikovanije rukovanje prekidima
- ❖ Povećano kašnjenje u odnosu na kernel kod



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Prevođenje i pokretanje Linuksa  
Prvi deo



# Sadržaj



- ❖ Prevodenje i pokretanje Linuksa
  - ❖ Izvorni kod kernela
  - ❖ Menadžeri izvornog koda
  - ❖ Konfiguracija kernela
  - ❖ Prevodenje kernela
  - ❖ Linuks datoteke uređaja



Kod Linuks kernela

# **PREVOĐENJE I POKRETANJE LINUKSA**



# Struktura Linuks izvornog koda (1/2)



- ❖ arch/<arch>
- ❖ arch/<arch>/mach-<mach>
- ❖ block/
- ❖ COPYING
- ❖ CREDITS
- ❖ crypto/
- ❖ Documentation/
- ❖ drivers/
- ❖ fs/
- ❖ include/
- ❖ include/asm-<arch>
- ❖ include/linux
- ❖ init/



# Struktura Linuks izvornog koda (2/2)



- ❖ ipc/
- ❖ Kbuild
- ❖ kernel/
- ❖ lib/
- ❖ MAINTAINERS
- ❖ Makefile
- ❖ mm/
- ❖ net/
- ❖ README
- ❖ REPORTING-BUGS
- ❖ scripts/
- ❖ security/
- ❖ sound/
- ❖ usr/



# Veličina Linuks kernela (1/5)

## ❖ Linux 2.6.17:

- ❖ Veličina: 224 MB (20400 files, približno 7 miliona linija koda)
- ❖ gzip kompresovana tar arhiva: 50 MB
- ❖ bzip2 kompresovana tar arhiva : 40 MB (bolje)
- ❖ 7zip kompresovana tar arhiva : 33 MB (najbolje)

## ❖ Linux 2.6.37:

- ❖ Veličina: 412 MB (37300 files, približno 14 miliona linija)
- ❖ gzip kompresovana tar arhiva : 89 MB
- ❖ bzip2 kompresovana tar arhiva : 71 MB (bolje)
- ❖ Izma kompresovana tar arhiva : 61 MB (najbolje)

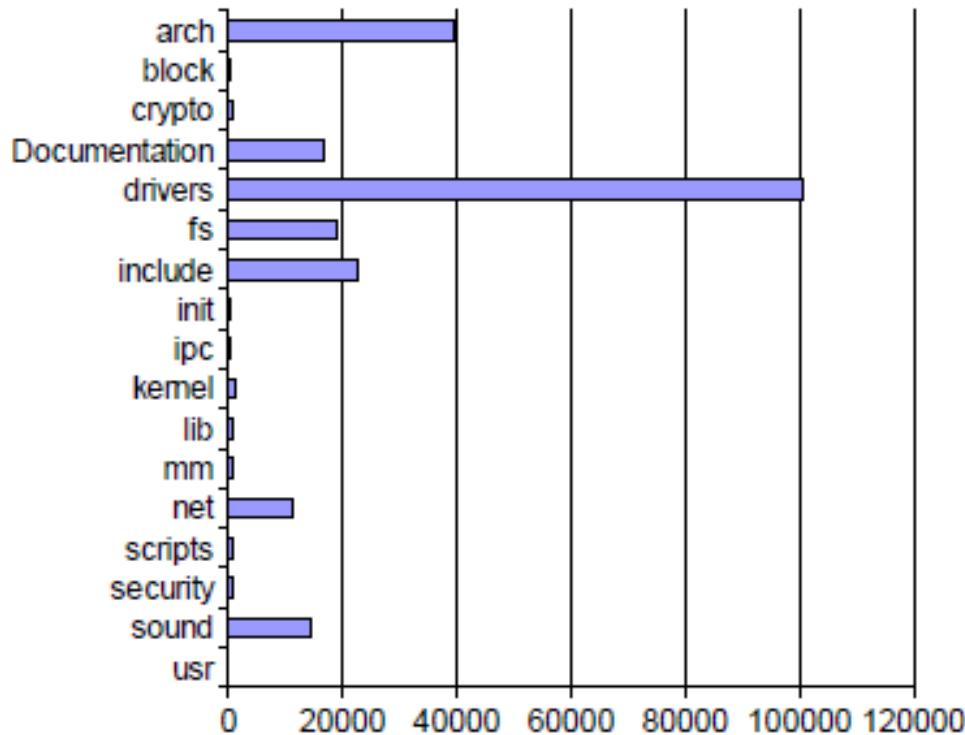


## Veličina Linuks kernela (2/5)

- ❖ Minimalna veličina prevedenog Linuks kernela iznosi ~300 KB kompresovana, odnosno 800 KB (2.6.14)
- ❖ Minimalna veličina Linuksa 2.6.29 prevedenog sa CONFIG\_EMBEDDED, za kernel koji pokreće QEMU PC (IDE, ext2, ELF) iznosi 532 KB kompresovan, 1325 KB inače
- ❖ Razlike su velike jer uključuju hiljade drajvera za uređaje, razne mrežne protokole, podržavaju mnogo arhitektura i fajl sistema
- ❖ Linuks jezgro je zasebno jako malo!

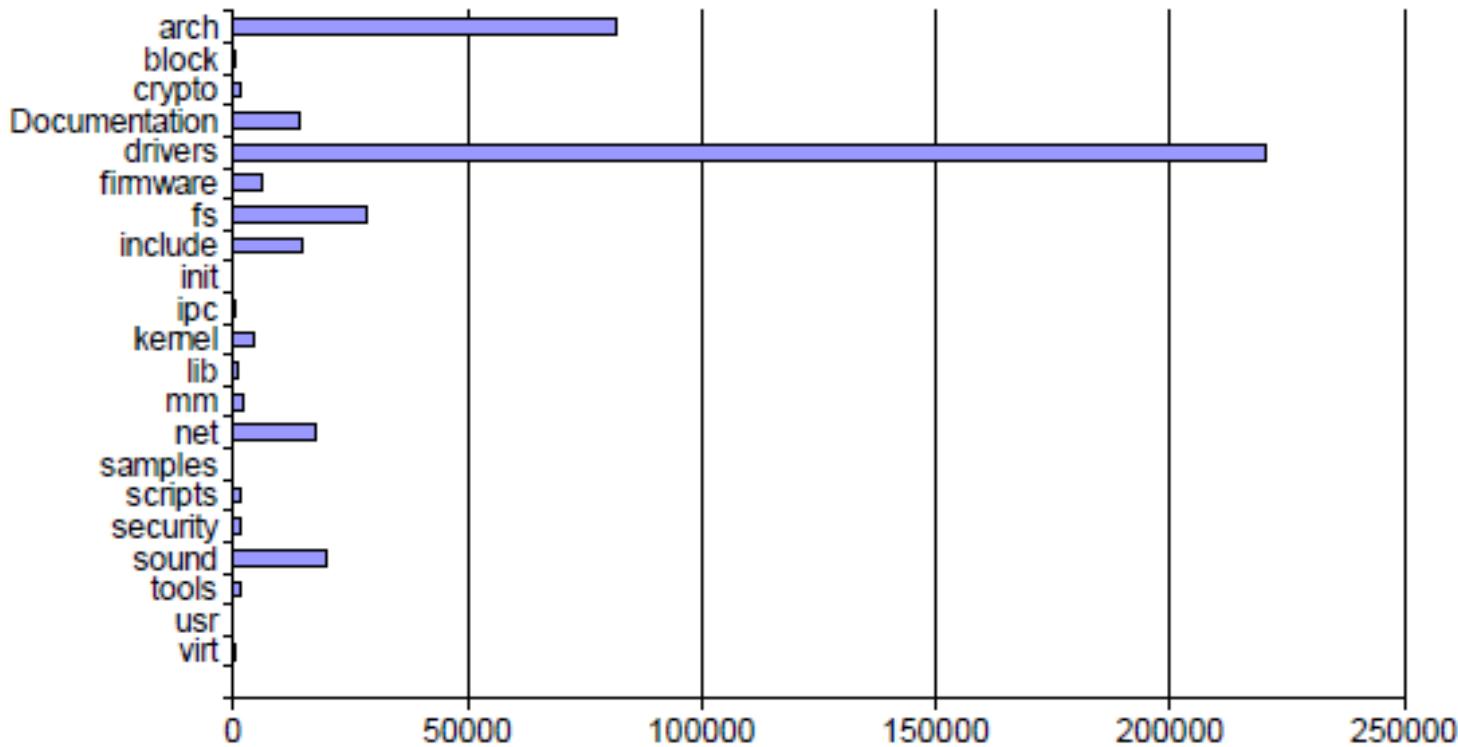
# Veličina Linuks kernela (3/5)

- ❖ Veličina datoteka Linuks koda (KB) – 2.6.17



# Veličina Linuks kernela (4/5)

- ❖ Veličina datoteka Linuks koda (KB) – 2.6.37





# Veličina Linuks kernela (4/5)

- ❖ Linuks 4.6 sources:
  - ❖ Raw veličina: 730 MB (53,600 files, približno 21,400,000 linija koda)
  - ❖ Gzip kompresovana tar arhiva: 130 MB
  - ❖ Xz kompresovana tar arhiva: 85 MB
- ❖ Minimum Linux 3.17 prevedena veličina kernela, pokrenuta na ARM Versatile ploči (hard disk na PCI, ext2 sistem datoteka, ELF podrška, framebuffer konzola i ulazni uređaji): 876 KB (kompresovan), 2.3 MB (raw)
- ❖ Zašto je izvorni kod toliki?
  - ❖ drivers/: 57.0% sistema inače
  - ❖ Mrežni protokoli, podrška za razne arhitekture, sisteme datoteka...
  - ❖ Linuks jezgro (rasporedjivač i sl) je prilično malo!



# Preuzimanje Linuks izvornog koda

- ❖ 2 načina
- ❖ Pun izvorni kod
  - ❖ Najlakši način, duže vreme preuzimanja
    - ❖ <http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.14.7.tar.bz2>
  - ❖ Uradi se git clone željenog repozitorijuma
    - ❖ git clone  
`git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git`
- ❖ Zakrpa za prethodnu verziju
  - ❖ Prepostavka da imamo pun izvorni kod prethodne verzije
  - ❖ <http://kernel.org/pub/linux/kernel/v2.6/patch-2.6.14.7.bz2>  
(2.6.14 to 2.6.14.7)



# Preuzimanje punog koda

- ❖ Pomoću komandne linije
  - ❖ Pomoću web pretraživača, pronaći verziju koja je potrebna na <http://kernel.org>
  - ❖ U ogovarajućem direktorijumu, preuzeti arhivu i njen potpis wget <http://kernel.org/pub/linux/kernel/v2.6/linux-2.6.11.12.tar.bz2>
  - ❖ Ekstrahovati sadržaj arhive tar jxf linux-2.6.11.12.tar.bz2



# Preuzimanje zakrpe za kernel kod (1/2)



- ❖ Pod prepostavkom da imamo linux-x.xy.<n-1> verziju
- ❖ Identificuje se zakrpa koja nam je potrebna
- ❖ Preuzme se zakrpa
- ❖ Primer:
  - ❖ 2.6.10 na 2.6.11  
`wget ftp://ftp.kernel.org/pub/linux/kernel/v2.6/patch-2.6.11.bz2`
  - ❖ 2.6.11 na 2.6.11.12  
`wget http://www.kernel.org/pub/linux/kernel/v2.6/patch-2.6.11.12.bz2`



# Preuzimanje zakrpe za kernel kod (2/2)



- ❖ Primena zakrpa u odgovarajućem redosledu
  - ❖ cd linux-2.6.10/
  - ❖ bzcat ..../patch-2.6.11.bz2 | patch -p1
  - ❖ bzcat ..../patch-2.6.11.12.bz2 | patch -p1
  - ❖ cd ..
  - ❖ mv linux-2.6.10 linux-2.6.11.12



# Izgled zakrpe

## ❖ Zakrpa je izlaz iz diff komande

```
diff -Nurp linux_clean/arch/arm/configs/bcm2709_rtrk_linux_kurs_defconfig
linux_patch/arch/arm/configs/bcm2709_rtrk_linux_kurs_defconfig
--- linux_clean/arch/arm/configs/bcm2709_rtrk_linux_kurs_defconfig      1970-01-01
01:00:00.000000000 +0100
+++ linux_patch/arch/arm/configs/bcm2709_rtrk_linux_kurs_defconfig      2016-08-02
08:22:46.905892655 +0200
@@ -0,0 +1,1350 @@
+#
+## Automatically generated file; DO NOT EDIT.
+# Linux/arm 4.4.16 Kernel Configuration
+#
+CONFIG_ARM=y
+CONFIG_SYS_SUPPORTS_APM_EMULATION=y
+CONFIG_HAVE_PROC_CPU=y
+CONFIG_STACKTRACE_SUPPORT=y
+CONFIG_HAVE_LATENCYTOP_SUPPORT=y
+CONFIG_LOCKDEP_SUPPORT=y
+CONFIG_TRACE_IRQFLAGS_SUPPORT=y
+CONFIG_RWSEM_XCHGADD_ALGORITHM=y
+CONFIG_FIX_EARLYCON_MEM=y
+CONFIG_GENERIC_HWEIGHT=y
+CONFIG_GENERIC_CALIBRATE_DELAY=y
+CONFIG_NEED_DMA_MAP_STATE=y
+CONFIG_ARCH_SUPPORTS_UPROBES=y
...
```



# Korišćenje komande patch

- ❖ patch komanda primenjuje promene na datoteke u trenutnom direktorijumu
  - ❖ Pravi izmene na postojeće datoteke
  - ❖ Kreira ili uklanja datoteke i direktorijume
- ❖ Primeri upotrebe patch komande
  - ❖ patch -p<n> < diff\_file>
  - ❖ cat diff\_file | patch -p<n>
  - ❖ bzcat diff\_file.bz2 | patch -p<n>
  - ❖ zcat diff\_file.gz | patch -p<n>
- ❖ n je broj nivoa direktorijuma koji se preskaču u putanjama
- ❖ Može da se vrati patch sa -R opcijom
- ❖ Zakrpa može da se testira sa -dry-run opcijom



# Primena Linuks zakrpe

- ❖ Uvek treba primeniti na `x.y.<z-1>` verziju koja je preuzeta u komrpesovanom formatu
- ❖ Uvek se kreira za `n=1`
- ❖ Primeri iz komandne linije za primenu zakrpe
  - ❖ `cd linux-2.6.10`
  - ❖ `bzcat ..patch-2.6.11.bz2 | patch -p1`
  - ❖ `cd ..; mv linux-2.6.10 linux-2.6.11`
  - ❖ `cd linux-3.9`
  - ❖ `xzcat ..patch-3.10.xz | patch -p1`
  - ❖ `xzcat ..patch-3.10.9.xz | patch -p1`
  - ❖ `cd ..; mv linux-3.9 linux-3.10.9`
- ❖ Treba držati Linuks zakrpe kompresovanim, jer sa `vi` komandom se mogu pregledati i kompresovane zakrpe



# Pristupanje razvojnom kodu (1/2)



- ❖ Razvojnim kodom se upravlja pomoći git-a
- ❖ Može da se pregleda Linuks git stablo
- ❖ Ako ste iza proxy-ja, treba postaviti Unix promenljive za okruženje koje definišu podešavanja proxy-ja



# Pristupanje razvojnom kodu (2/2)

- ❖ Preuzme se git razvojno stablo sa <http://git.kernel.org/>
- ❖ Preuzme se lokalna kopija (klon) stabla
- ❖ Ažurirajte svoju kopiju kad god je potrebno



# Kernel dokumentacija na mreži

- ❖ <http://free-electrons.com/kerneldoc/>
- ❖ Lakše je nego preuzimanje celog kernel koda radi pristupa dokumentaciji
- ❖ Indeksirano je
  - ❖ Lakše je pronaći delove dokumentacije koje želimo
- ❖ Za razliku od ostalih stranica koje nude dokumentaciju, takođe postoji HTML prevod kernel dokumenata u DocBook formatu



Alatke za upravljanje izvornim kodom kernela

# **PREVOĐENJE I POKRETANJE LINUKSA**



# Cscope



❖ <http://cscope.sourceforge.net/>

- ❖ Alatka za pregled izvornog koda (uglavnom C, može da se koristi i za C++ i Javu)
- ❖ Podržava ogromne projekte poput Linuks kernela
  - ❖ Treba mu manje od minute da indeksira 2.6.17 kod
- ❖ Može da se koristi iz alata poput vim i emacs
- ❖ U izvornom kodu, pokreće se sa
  - ❖ cscope –Rk
- ❖ Dozvoljava pretraživanje koda po:
  - ❖ Svim referencama na simbol, globalnim definicijama, pozivanim funkcija, funkcijama koje prave pozive, regeks, datotekama, datotekama koje uključuju datoteke



# Cscope - изглед

```
xterm
C symbol: request_irq

File           Function           Line
0 omap_udc.c   omap_udc_probe    2821 status = request_irq(pdev->resource[1].start, omap_udc_irq,
1 omap_udc.c   omap_udc_probe    2830 status = request_irq(pdev->resource[2].start, omap_udc_pio_irq,
2 omap_udc.c   omap_udc_probe    2838 status = request_irq(pdev->resource[3].start, omap_udc_iso_irq,
3 pxa2xx_udc.c  pxa2xx_udc_probe  2517 retval = request_irq(IRQ_USB, pxa2xx_udc_irq,
4 pxa2xx_udc.c  pxa2xx_udc_probe  2528 retval = request_irq(LUBBOCK_USB_DISC_IRQ,
5 pxa2xx_udc.c  pxa2xx_udc_probe  2539 retval = request_irq(LUBBOCK_USB_IRQ,
6 hc_crшив10.c etrax_usb_hc_init 4423 if (request_irq(ETRAX_USB_HC_IRQ, etrax_usb_hc_interrupt_top_half,
          0,
7 hc_crшив10.c etrax_usb_hc_init 4431 if (request_irq(ETRAX_USB_RX_IRQ, etrax_usb_rx_interrupt, 0,
8 hc_crшив10.c etrax_usb_hc_init 4439 if (request_irq(ETRAX_USB_TX_IRQ, etrax_usb_tx_interrupt, 0,
9 amifb.c       amifb_init      2431 if (request_irq(IRQ_AMIGA_COPPER, amifb_interrupt, 0,
a arcfb.c       arcfb_probe     564 if (request_irq(par->irq, &arcfb_interrupt, SA_SHIRQ,
b atafb.c       atafb_init      2720 request_irq(IRQ_AUTO_4, falcon_vbl_switcher, IRQ_TYPE_PRIO,
c atyfb_base.c  aty_enable_irq  1562 if (request_irq(par->irq, aty_irq, SA_SHIRQ, "atyfb", par)) {

* 155 more lines - press the space bar to display more *
Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
```



# LXR: Linuks Cross Reference

- ❖ <http://sourceforge.net/projects/lxr>
- ❖ Alatka za indeksiranje i pregled koda
  - ❖ Zasnovana na web serveru
    - ❖ Lako i brzo za korišćenje
    - ❖ Pretraživanje po tekstu ili identifikatoru
    - ❖ Lako za pronalaženje deklaracije, implementacije ili upotrebe simbola
  - ❖ Podržava C i C++
  - ❖ Podržava ogromne projekte
  - ❖ Treba malo vremena i strpljenja dok se ne nameste postavke
  - ❖ Indeksranje novije verzije koda je prilično brzo
  - ❖ Potoje online web serveri, tako da ne moraju da se nameštaju serverska podešavanja



# Izgled LXR-a



## Linux Cross Reference

Free Electrons

Embedded Linux Experts

• Source Navigation • Diff Markup • Identifier Search • Freetext Search •

Version: 2.0.40 2.2.26 2.4.37 3.11 3.12 3.13 3.14 3.15 3.16 3.17 3.18 3.19 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8

### Linux/kernel/user.c

```
1 /*
2  * The "user cache".
3  *
4  * (C) Copyright 1991-2000 Linus Torvalds
5  *
6  * We have a per-user structure to keep track of how many
7  * processes, files etc the user has claimed, in order to be
8  * able to have per-user limits for system resources.
9 */
10
11 #include <linux/init.h>
12 #include <linux/sched.h>
13 #include <linux/slab.h>
14 #include <linux/bitops.h>
15 #include <linux/key.h>
16 #include <linux/interrupt.h>
17 #include <linux/export.h>
18 #include <linux/user_namespace.h>
19 #include <linux/proc_ns.h>
20
21 /*
22  * userns count is 1 for root user, 1 for init_uts_ns,
23  * and 1 for... ?
24 */
```



Konfiguracija kernela

# PREVOĐENJE I POKRETANJE LINUKSA



# Konfiguracija kernela (1/2)

- ❖ Kernel sadrži hiljade drajvera za uređaje, drajvere za fajl sisteme, mrežne protokole i druge podesive stvari
- ❖ Dostupne su hiljade opcija koje se koriste da se odabirom prevedu delovi izvornog koda kernela
- ❖ Konfiguracija kernela je proces definisanja seta opcija sa kojima želimo da prevedemo naš kernel
- ❖ Set opcija zavisi od:
  - ❖ Fizičke arhitekture
  - ❖ Mogućnosti koje želimo da kernel ima



## Konfiguracija kernela (2/2)

- ❖ Kofiguracija definiše šta želimo da uključimo u kernel
- ❖ Čuva se u .config datoteci u root-u kernel koda
- ❖ Najkorisnije komande za kreiranje konfiguracione datoteke
  - ❖ make [xconfig|gconfig|menuconfig|oldconfig]
- ❖ Da bismo izmenili kernel u GNU/Linuks distribuciju, konfiguracione datoteke se najčešće nalaze u /boot/ zajedno sa slikama kernela /boot/config-2.6.17-11-generic
- ❖ Ako je uključeno u General Setup -> Kernel .config support, konfigraciona datoteka se može naći i u samom kernelu
  - ❖ > zcat /proc/config.gz



# make xconfig

- ❖ make xconfig
  - ❖ Najčešća grafička sprega koja se koristi za konfiguraciju kernela
  - ❖ help -> introduction: useful options! Treba pročitati pre rada
  - ❖ Pretraživač datoteka čini lakše učitavanje konfiguracionih datoteka
  - ❖ Nova sprega za pretraživanje za pretragu parametara
  - ❖ Potrebni paketi:
    - ❖ libqt3-mt-dev
    - ❖ g++



# Izgled make xconfig

Linux/arm 3.4.0 Kernel Configuration

File Edit Option Help

Option

- General setup
  - IRQ subsystem
  - RCU Subsystem
  - Control Group support
  - Namespaces support
  - Configure standard kernel features (expert users)
  - Kernel Performance Events And Counters
  - GCOV-based kernel profiling
  - Enable loadable module support
  - Enable the block layer
    - Partition Types
    - IO Schedulers
  - System Type
    - TI OMAP Common Features
    - TI OMAP2/3/4 Specific Features
  - Bus support
    - PCCard (PCMCIA/CardBus) support
  - Kernel Features
  - Boot options
- CPU Power Management
  - CPU Frequency scaling
  - Floating point emulation
  - Userspace binary formats
  - Power management options
- Networking support
  - Networking options

## Option

- ..
- OMAP System Type
  - OTI OMAP1
  - OTI OMAP2/3/4
- OMAP Feature Selections
- SmartReflex support
- Reset unused clocks during boot
- OMAP multiplexing support
  - Multiplexing debug output
  - Warn about pins the bootloader didn't set up
- Mailbox framework support
- Use 32KHz timer

## TI OMAP2/3/4 (ARCH\_OMAP2PLUS)

CONFIG\_ARCH\_OMAP2PLUS:

"Systems based on OMAP2, OMAP3 or OMAP4"

Symbol: ARCH\_OMAP2PLUS [=y]

Type : boolean

Prompt: TI OMAP2/3/4

Defined at arch/arm/plat-omap/Kconfig:24

Depends on: <choice>

Location:

- > System Type
- > TI OMAP Common Features
- > OMAP System Type (<choice> [=y])



# Sprega za pretraživanje u make xconfig

- ❖ Postoji polje za pretragu i dozvoljava da se uključe ili isključe pronađeni parametri

Search Config

Find: mtd

Search

Option

- (0) Physical address of DiskOnChip
- NAND Flash support for Samsung S3C SoCs
- Support software BCH ECC
- ST Nomadik 8815 NAND support
- CFI Flash device mapped on AMD NetSc520
- M-Systems Disk-On-Chip Millennium-only alternative driver (DEPRECATED)
- ARM Firmware Suite partition parsing (NEW)
- PMC551 Debugging
- Command line partition table parsing

**Physical address of DiskOnChip (MTD\_DOCPROBE\_ADDRESS)**

**CONFIG\_MTD\_DOCPROBE\_ADDRESS:**

By default, the probe for DiskOnChip devices will look for a DiskOnChip at every multiple of 0x2000 between 0xC8000 and 0xEE000. This option allows you to specify a single address at which to probe for the device, which is useful if you have other devices in that range which get upset when they are probed.



# Opcije za konfiguraciju kernela

- ❖ Prevedno kao modul
  - ❖ CONFIG\_ISO9660\_FS=m
    - ISO 9660 CDROM file system support
    - Microsoft Joliet CDROM extensions
    - Transparent decompression extension
    - UDF file system support
- ❖ Opcija rukovaoca  
CONFIG\_JOLIET=y
- ❖ CONFIG\_ZISOFS=y
- ❖ CONFIG\_UDF\_FS=y
  - ❖ Statički prevedeno u kernelu



# Izvod iz .config datoteke

❖ Ime skicije

❖ Parametri:  
CONFIG\_\*

```
#  
#CD-ROM/DVD Filesystems  
  
#  
CONFIG_ISO9660_FS=m  
CONFIG_JOLIET=y  
CONFIG_ZISOFS=y  
CONFIG_UDF_FS=y  
CONFIG_UDF_NLS=y  
  
#  
#DOS/FAT/NT Filesystems  
  
#  
#CONFIG_MSDOS_FS is not set  
#CONFIG_VFAT_FS is not set  
CONFIG_NTFS_FS=m  
#CONFIG_NTFS_DEBUG is not set  
CONFIG_NTFS_RW=y
```



# Zavisnosti kernel opcija

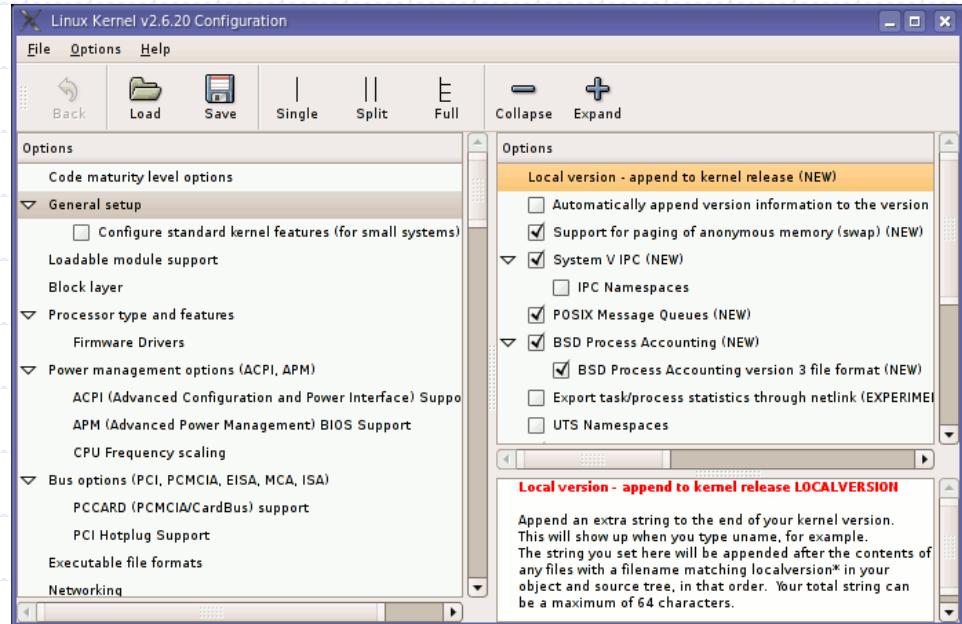
- ❖ Postoje zavisnosti između opcija kernela
- ❖ Na primer, uključivanje mrežnog drajvera zahteva da je već uključen mrežni stek
- ❖ Dva tipa zavisnosti
  - ❖ depends on zavisnosti
    - ❖ Opcija A koja zavisi od opcije B nije vidljiva dok opcija B nije uključena
  - ❖ select zavisnosti
    - ❖ Ako opcija A zavisi od opcije B, kada se uključi opcija A, automatski se uključuje i opcija B
  - ❖ make xconfig dozvoljava da se vide sve opcije, čak i one koje ne mogu biti vidljive zvog neispunjene zavisnosti
    - ❖ Ako su „nevidljive“ prikazane su zasivljene



# make gconfig



- ❖ make gconfig
- ❖ Novija konfiguraciona sprega zasnovana na GTK
  - ❖ Funkcionalnost slična sa make xconfig
- ❖ Nema funkcionalnost pretraživanja
- ❖ Potrban paket: libglade2-dev





# make menuconfig

- ❖ make menuconfig
- ❖ Koristan kad ne postoji grafika
- ❖ Potreban Debian paket: libncurses-dev

```
.config - Linux/arm 3.0.6 Kernel Configuration

System Type
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend:
[*] built-in [ ] excluded <M> module < > module capable

[*] MMU-based Paged Memory Management Support
ARM system type (TI OMAP) --->
    TI OMAP Common Features --->
    TI OMAP2/3/4 Specific Features --->
    *** System MMU ***
    *** Processor Type ***
    Marvell Sheeva CPU Architecture
    *** Processor Features ***

[*] Support Thumb user binaries (NEW)
[ ] Enable ThumbEE CPU extension (NEW)
[ ] Run BE8 kernel on a little endian machine (NEW)
[ ] Disable I-Cache (I-bit) (NEW)
[ ] Disable D-Cache (C-bit) (NEW)
[ ] Disable branch prediction (NEW)
[*] Enable lazy flush for v6 smp (NEW)
[*] stop_machine function can livelock (NEW)
[ ] Spinlocks using LDREX and STREX instructions can livelock (NEW)
[ ] Enable S/W handling for Unaligned Access (NEW)
[*] Enable the L2x0 outer cache controller (NEW)
[ ] ARM errata: Invalidation of the Instruction Cache operation can fai
v(+)

<Select>  < Exit >  < Help >
```



# make oldconfig

- ❖ make oldconfig
- ❖ Često je potreban
- ❖ Koristan za nadogradnju .config datoteke sa ranijeg izdanja kernela
- ❖ Izbacuje upozorenja za zastarele simbole
- ❖ Pita za vrednosti novih simbola
- ❖ Ako se ručno menja .config datoteka, preporučuje se da se pokrene make oldconfig nakon toga



## make allnoconfig

- ❖ make allnoconfig
- ❖ Podešava samo strogo preporučene vrednosti na y
- ❖ Postavlaj ostale vrednosti na n
- ❖ Korisno u ugrađenim sistemima kad želimo da odaberemo najamnji set odlika i drajvera
- ❖ Mnogo je pogodnije neko ručno isključivanje na stotine odlika jednu po jednu



# Poništavanje konfiguracionih izmena

## ❖ Čest problem

- ❖ Nakon menjanja nekoliko podešavanja konfiguracije kernela, kernel prestane da radi
- ❖ Ako se ne sećamo izmena koje smo napravili, možemo da se vratimo na prethodnu konfiguraciju sa:
  - ❖ cp .config.old .config
- ❖ Sve konfiguracione sprege kernela čuvaju .config.old kopiju (xconfig, menuconfig, allnoconfig...)



## make help

- ❖ make help
- ❖ Izlistava sve dostupne make opcije
- ❖ Korisno je za podsećanje ili da se pogleda koje su nove ili napredne opcije



# Podešavanje teksta verzije

- ❖ Da bi se identifikovala slika kernela između ostalih kreiranih od istog koda (ali sa različitom konfiguracijom) koristi se **LOCALVERSION** podešavaje (u General Setup)
- ❖ Primer

```
#  
# General setup  
#  
CONFIG_LOCALVERSION="-acme1"
```

- ❖ **uname -r** komanda (u pokrenutom sistemu) će vratiti **2.6.20-acme1**



Prevodenje kernela

# PREVOĐENJE I POKRETANJE LINUKSA



# Prevodenje i instaliranje kernela

## ❖ Prevodenje:

- ❖ make

## ❖ Koraci instaliranja:

- ❖ sudo make install
- ❖ sudo make modules\_install



# Brže prevodenje na višejezgarnim sistemima



- ❖ Ako se koristi računar sa  $n$  procesora, može skoro da se podeli preve prevodenja sa  $n$  tako što se prevode datoteke u paraleli
- ❖ make -j <n>
  - ❖ Pokreće više radnji u paraleli kad god je to moguće
- ❖ Korišćenjem make -j 2 ili make -j 3 na jednom procesoru je pomaže mnogo
  - ❖ Teoretski, više paralelizovanih poslova drže procesor zauzet dok ostali procesi čekaju da datoteke budu pročitane ili napisane
  - ❖ U praksi, ne dobija se neko značajno ubrzanje (ne više od 10%), osim ako su I/O uređaji jako spori



# Brže prevodenje sa ccache

- ❖ Korisno kad su izmene na .config datoteci česte
- ❖ Koristi se tako što se doda ccache prefiks na CC i HOSTCC definicije u Makefile
  - ❖ CC = `ccache $(CROSS_COMPILE)gcc`
  - ❖ HOSTCC = `ccache gcc`
- ❖ Merenja performansi
  - ❖ 63% sa Fedora Core 3 konfiguracionom datotekom
  - ❖ 82% sa konfiguracionom datotekom ugrađenog Linuksa
  - ❖ Prvi sadrži mnogo više modula od drugog



# Saveti za prevodenje kernela

- ❖ Videti punu komandnu liniju (gcc, ld...)
  - ❖ make V=1
- ❖ Počistiti generisane datoteke (da se osigura ponovno prevodenje drajvera)
  - ❖ make clean
- ❖ Ukloniti sve generisane datoteke (Uklanja i .config datoteku)
  - ❖ make mrproper
- ❖ Uklanjanje i rezervih datoteka teksta editora i odbijenih zakrpa (patch datoteka)
  - ❖ make distclean



# Generisane datoteke

- ❖ Kreiraju se kada se pokrene make komanda
- ❖ vmlinuz
  - ❖ Slika Linuks kernela, nekompresovana
- ❖ arch/<arch>/boot/zImage (uobičajena slika za arm)
  - ❖ Kompresovana slika kernela u zlib formatu
- ❖ arch/<arch>/boot/bzImage (uobičajena slika za i386)
  - ❖ Kompresovana slika kernela u zlib formatu
  - ❖ Napomena: bz znači „big zipped“ a ne „bzip2 kompresovano“
    - ❖ Podrška za bzip2 kompresiju je na i386 dostupna samo kao zakrpa.
    - ❖ Nije atraktivno za korišćenje na minorm ugrađenim sistemima jer koristi oko 1 MB RAM za dekompresiju



# Datoteke kreirane sa make install



- ❖ /boot/vmlinuz-<version>
  - ❖ Kompresovana slika kernela
  - ❖ Ista kao i ona u arch/<arch>/boot
- ❖ /boot/System.map-<version>
  - ❖ Ovde se čuvaju adrese simbola
- ❖ /boot/config-<version>
  - ❖ Konfiguracija kernela za ovu verziju



# Datoteke kreirane sa make modules\_install (1/2)



- ❖ /lib/modules/<version>/: moduli kernela + dodaci
- ❖ build/
  - ❖ Sve što je potrebno da se kreira jos modula za ovaj kernel
    - ❖ Makefile
    - ❖ .config
    - ❖ module.symVers – informacija o simbolu modula
    - ❖ include/ i include/asm/ – zaglavlja kernela (header files)
- ❖ kernel/
  - ❖ Datoteke .ko (Kernel Object) modula
  - ❖ Imaju istu strukturu direktorijuma kao i u kodu



# Datoteke kreirane sa make modules\_install (2/2)



- ❖ /lib/modules/<version>/ (nastavak)
  - ❖ modules.alias
    - ❖ Alijasi modula za alate za učitavanje modula, primer:
      - ❖ alias sound-service-?-0 snd\_mixer\_oss
  - ❖ modules.dep
    - ❖ Zavisnosti modula
  - ❖ modules.symbols
    - ❖ Informacija o pripadnosti simbola modulima
  - ❖ Sve datoteke u ovom direktorijumu su tekstualne datoteke, možete ih pregledati ☺



# Prevodenje kernela, ukratko

- ❖ make xconfig
- ❖ make
- ❖ sudo make install
- ❖ sudo make modules\_install



Datoteke uređaja Linuksa

# **PREVOĐENJE I POKRETANJE LINUKSA**



# Karakterne datoteke uređaja

- ❖ Prestupa im se kroz sekvencijalni protok jedinstvenih simbola
- ❖ Mogu biti identifikovani po njihovom c tipu (ls -l)

```
crw-rw---- 1 root uucp 4, 64 Feb 23 2016 /dev/ttys0
crw--w---- 1 jdoe tty 136, 1 Feb 23 2016 /dev/pts/1
crw----- 1 root root 13, 32 Feb 23 2016< /dev/input/mouse0
crw-rw-rw- 1 root root 1, 3 Feb 23 2016 /dev/null
```

- ❖ Primeri: tastature, miševi, IrDA, Bluetooth prolaz, konzole, terminali, zvuk, video



# Blok datoteke uređaja

- ❖ Pristupa im se kroz blokove podataka date veličine
  - ❖ Mogu biti poređani u bilo kod redosledu
- ❖ Blok uređaji mogu biti identifikovani sa b tipom (ls -l)

```
brw-rw---- 1 root disk 3, 1 Feb 23 2004 hda1
brw-rw---- 1 jdoe floppy 2, 0 Feb 23 2004 fd0
brw-rw---- 1 root disk 7, 0 Feb 23 2004 loop0
brw-rw---- 1 root disk 1, 1 Feb 23 2004 ram1
brw----- 1 root root 8, 1 Feb 23 2004 sda1
```

- ❖ Primeri: hard diskovi, flopi diskovi, ram diskovi, loop uređaji



# Major i minor brojevi uređaja

- ❖ Kao što smo primetili datoteke uređaja imaju dva broja asocirana sa njima
- ❖ Prvi broj: major broj
- ❖ Drugi broj: minor broj
- ❖ Major i minor brojeve koristi kernel da veže drajver za datoteku uređaja. Imena datoteka uređaja ništa ne znače kernelu
- ❖ Da se pronađe koji drajver odgovara kojoj datoteci ili kad je ume uređaja previše komplikovano pogledati:
  - ❖ Documentation/devices.txt



# Kreiranje datoteka uređaja

- ❖ Datoteke uređaja se ne kreiraju pri učitavanju drajvera
- ❖ Moraju biti unapred napravljenje
  - ❖ sudo mknod /dev/<device> [c|b] <major> <minor>
- ❖ Primeri:
  - ❖ sudo mknod /dev/ttyS0 c 4 64
  - ❖ sudo mknod /dev/hda1 b 3 1



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Prevođenje i pokretanje Linuksa  
Drugi deo



# Sadržaj



- ❖ Prevodenje i pokretanje Linuksa
  - ❖ Pokretanje sistema
  - ❖ Pokretanje kernela
  - ❖ Prevodenje kernela na drugu platformu
  - ❖ Učitavanje modula



Uopšteno pokretanje sistema

# **PREVOĐENJE I POKRETANJE LINUXA**



# Sekvenca pokretanja Linuksa 2.4

## ❖ Bootloader

- ❖ Izvršava ga fizička arhitektura na podešenoj lokaciji u ROM-u ili Flash-u
- ❖ Inicijalizuje podršku za uređaj gde se nalazi slika kernela
- ❖ Učitava sliku kernela u RAM
- ❖ Izvršava sliku kernela (sa definisanom komandom)

## ❖ Kernel

- ❖ Dekompresuje se
- ❖ Inicijalizuje jezgro kernela i staticki prevedene drajvere
- ❖ Postavlja korenski sistem datoteka (**root** parametar)
- ❖ Izvršava prvi program korisničkog prostora (**init** parametar)

## ❖ Prvi program korisničkog prostora

- ❖ Konfiguriše korisnički prostor i pokreće sistemske servise



# Sekvenca podizanja Linuksa 2.6 (1/2)



- ❖ Bootloader
  - ❖ Izvršava ga fizička arhitektura na podešenoj lokaciji
  - ❖ Inicijalizuje podršku za uređaj gde se nalazi slika kernela
  - ❖ Učitava sliku kernela u RAM
  - ❖ Izvršava sliku kernela (sa definisanim komandom)
- ❖ Kernel
  - ❖ Dekompresuje se
  - ❖ Inicijalizuje jezgro kernela i staticki prevedene drajvere
  - ❖ Dekompresuje initramfs cpio arhivu koja je uključena u keš datoteku kernela
  - ❖ Ako postoji u initramfs, izvršava prvi program korisničkog prostora (`/init`)



# Sekvenca podizanja Linuksa 2.6 (2/2)



- ❖ Korisnički prostor: `/init` skripta (tipičan sled događaja)
  - ❖ Pokreće komande da konfiguriše uređaje (postavka mreže, `/proc`, `/sys...`)
  - ❖ Postavlja novi korenski sistem datoteka i prebacuje se na njega (`switch_root`)
  - ❖ Pokreće `/sbin/init` (ili novu `/linuxrc` skriptu)
- ❖ Korisnički prostor: `/sbin/init`
  - ❖ Pokreće komande da konfiguriše uređaje
  - ❖ Pokreće sistemske servise (demoni, serveri) i korisničke programe



# Sekvenca podizanja Linuksa 2.6 sa initrd (1/2)



## ❖ Bootloader

- ❖ Izvršava ga fizička arhitektura na podešenoj lokaciji u ROM-u ili flešu
- ❖ Inicijalizuje podršku za uređaj gde se nalaze slike
- ❖ Učitava sliku kernela i sliku init ramdisk-a (**initrd**) u RAM
- ❖ Izvršava sliku kernela (sa definisanom komandom)

## ❖ Kernel

- ❖ Dekompresuje se
- ❖ Inicijalizuje statički prevedene drajvere
- ❖ Dekompresuje initramfs cpio arhivu koja je uključena u keš datoteku kernela, postavlja je i ne pokreće **/init** jer ne postoji
- ❖ Postavlja korenski sistem datoteka određen root kernel parametrom (**initrd**)
- ❖ Izvršava prvi korisnički program, obično bude **/linuxrc**



# Sekvenca podizanja Linuksa 2.6 sa initrd (2/2)



- ❖ Korisnički prostor: /linuxrc skripta u initrd (tipičan sled događaja)
  - ❖ Pokreće komande da konfiguriše uređaje (postavka mreže, /proc, /sys...)
  - ❖ Učitava kernel module smeštene u initrd koji su potrebni da se pristupi novom korenskom sistemu datoteka
  - ❖ Postavlja novi korenski sistem datoteka i prebacuje se na njega (pivot\_root)
  - ❖ Pokreće /sbin/init (ili ponekad /linuxrc skriptu)
- ❖ Korisnički prostor: /sbin/init
  - ❖ Pokreće komande da konfiguriše uređaje (ako već nije urađeno u initrd)
  - ❖ Pokreće sistemske servise (demoni, serveri) i korisničke programe



# Mane sekvence pokretanja Linuksa 2.4



- ❖ Pokušavanje postavljanja sistema datoteka naznačenog sa **root** kernel parametrom je kompleksno
  - ❖ Potrebni su drajveri uređaja i drajveri sistema datoteka da se učitaju
  - ❖ Naznačavanje korenskog sistema datoteka zahteva „ružno imenovanje uređaja crnom magijom“ (npr. **/dev/ram0**, **/dev/hda1**...) dok „/“ jos uvek ne postoji
  - ❖ Može da zahteva kompleksnu inicijalizaciju da bi se implementiralo u kernelu (**NFS**, **RAID**...)
- ❖ Ukratko: previše kompleksan kernel kod



# Dodatne mane init ramdiska

- ❖ Init ramdiskovi su implementirani kao standardni blok uređaji
  - ❖ Potreban je ramdisk i drajver sistema datoteka
  - ❖ Podešena veličina: ne mogu da je povećavaju
    - ❖ Slobodan prostor se ne može koristiti ni za šta drugo
  - ❖ Mora da bude kreiran i menjan kao blok uređaj
    - ❖ Formatiranje, postavljanje, menjanje, završavanje
    - ❖ Potrebne su mu korenske privilegije
  - ❖ Kao i u svakom blok uređaju, datoteke se prvo čitaju iz skladišta pa se tek onda kopiraju u keš
    - ❖ Sporo je i dobija se duplikat u RAM-u



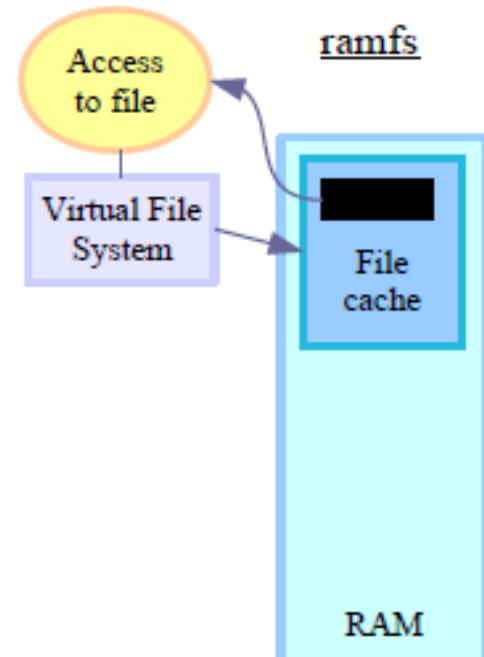
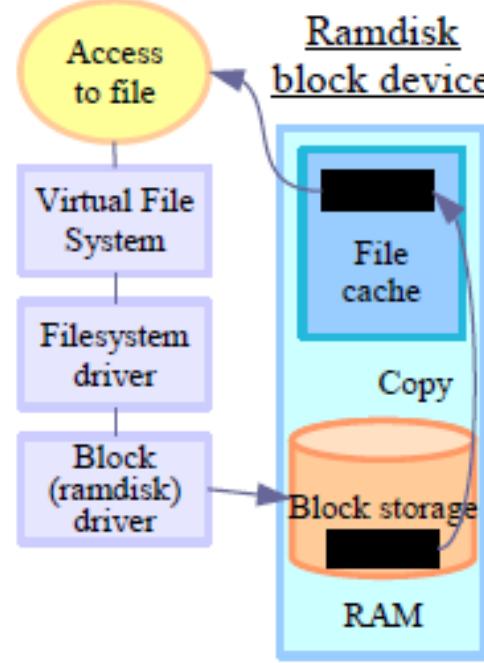
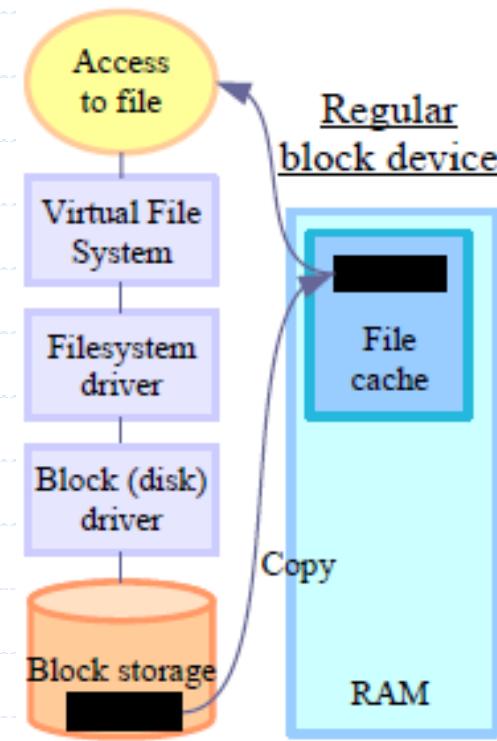
# Odlike i prednosti initramfs-a (1/4)



- ❖ Kernel slika kreira korenski sistem datoteka (ugrađen kao kompresovana **cpio** arhiva)
- ❖ Jako je lako za kreirati (u vreme izgradnje kernela)
  - ❖ Nema potrebe za korenskim privilegijama (**mount** i **mknod**)
- ❖ U poređenju sa init ramdiskovima, rukuje se samo 1 datotekom u bootloader-u
- ❖ Uvek je prisutan u Linuks kernelu 2.6 (uobičajeno prazan)
- ❖ To je kompresovana **cpio** arhiva
  - ❖ Ne treba mu ni blok drajver ni drajver sistema datoteka

# Odlike i prednosti initramfs-a (2/4)

- ❖ ramfs: implementiran u kešu, nema dupliciranja u RAM-u, nema sistemskih datoteka ni rukovanja istim i može da se širi memorija ako je potrebno





# Odlike i prednosti initramfs-a (3/4)



- ❖ Kernel ga učitava ranije
  - ❖ Više koda za inicijalizaciju je pomereno u korisnički prostor
- ❖ Lakše je postaviti kompleksne sisteme datoteka iz raznih skripti korisničkog prostora nego iz kernel koda
  - ❖ Više kompleksnosti je izmešteno u korisnički prostor
- ❖ Nema više magičnog imenovanja korenskog uređaja
  - ❖ Nije nam potreban više **pivot\_root**



# Odlike i prednosti initramfs-a (4/4)

- ❖ Moguće je dodati datoteke koje nisu GPL u sistem datoteka (frimver, vlasničke drajvere)
  - ❖ Ovo nije uvezivanje, već samo sakupljanje datoteka (ne smatra se izvedenim radom – zbog licence GPL)
- ❖ Moguće je ukloniti ove datoteke kad više nisu potrebne
- ❖ Više tehničkih detalja o initramfs se može videti u kodu kernela na lokacijama
  - ❖ [Documentation/filesystems/ramfs-rootfs-initramfs.txt](#)
  - ❖ [Documentation/early-userspace/README](#)



# Kako populisati initramfs

- ❖ Korišćenjem `CONFIG_INITRAMFS_SOURCE` u konfiguraciji kernela (`General Setup` sekcija)
  - ❖ Ili dati postojeću `cpio` arhivu
  - ❖ Ili dati listu datoteka i direktorijuma koji se dodaju u arhivu
  - ❖ Ili dati tekstualnu datoteku sa specifikacijama



# Primer datoteke za initramfs specifikaciju



- ❖ Nema potrebe za korenskim privilegijama

```
dir /dev 755 0 0
nod /dev/console 644 0 0 c 5 1
nod /dev/loop0 644 0 0 b 7 0
dir /bin 755 1000 1000
file /bin/busybox /stuff/initramfs/busybox 755 0 0
slink /bin/sh busybox 777 0 0
dir /proc 755 0 0
dir /sys 755 0 0
dir /mnt 755 0 0
file /init /stuff/initramfs/init.sh 755 0 0
```



# Kako rukovati sa kompresovanim cpio arhivama



- ❖ Korisno je kada želimo da izgradimo kernel sa **cpio** arhivom koja je već napravljena
  - ❖ Bolje je pustiti da kernel to odradi za nas
- ❖ Ekstrahovanje
  - ❖ **gzip -dc initramfs.img | cpio -id**
- ❖ Kreiranje
  - ❖ **find <dir> -print -depth | cpio -ov | gzip -c > initramfs.img**



# Kako kreirati initrd

- ❖ U slučaju da nam zaista treba initrd

```
sudo mkdir /mnt/initrd
dd if=/dev/zero of=initrd.img bs=1k count=2048
mkfs.ext2 -F initrd.img
sudo mount -o loop initrd.img /mnt/initrd

sudo umount /mnt/initrd
gzip --best -c initrd.img > initrd
```

- ❖ Više detalja na lokaciji [Documentation/initrd.txt](#)



# Varijante pokretanja

- ❖ XIP (Execute In Place)
  - ❖ Slika kernela se direktno izvršava iz skladишta
  - ❖ Može biti brže i čuva RAM
    - ❖ Slika kenela ne može biti kompresovana
- ❖ Nema initramfs / initrd
  - ❖ Direktno podizanje završnog korenskog sistema datoteka
    - ❖ Opcija kernel komandne linije **root**
- ❖ Nema novog korenskog sistema datoteka
  - ❖ Pokreće se ceo sistem iz initramfs



Pokretanje kernela

# PREVOĐENJE I POKRETANJE LINUKSA



# Parametri komandne linije kernela

- ❖ Kao i većina C programa, Linuks kernel prihvata argumente komandne linije
  - ❖ Oni su deo bootloader-ovih konfiguracionih podešavanja
  - ❖ Korisni su za izmene ponašanja kernela u vreme pokretanja, bez potrebe za novim prevodenjem
  - ❖ Korisno je za izvođenje naprednih inicijalizacija kernela i drajvera, a bez potrebe za kompleksnim skriptama korisničkog prostora



# Primer kernel komandne linije

- ❖ HP iPAQ h2200 PDA primer pokretanja:

```
root=/dev/ram0 \
rw \
init=/linuxrc \
console=ttyS0,115200n8 \
console=tty0 \
ramdisk_size=8192 \
cachepolicy=writethrough
```

Root filesystem (first ramdisk)  
Root filesystem mounting mode  
First userspace program  
Console (serial)  
Other console (framebuffer)  
Misc parameters...

- ❖ Stotine parametara komandne linije su opisani u [Documentation/kernel-parameters.txt](#)



# Korisnost rootfs-a na NFS-u

- ❖ Kada radi mreža, korenski sistem datoteka može biti direktorijum na GNU/Linux mašini za razvoj, izvedena od strane NFS-a (Network File System)
- ❖ Vrlo je korisno za razvoj sistema
  - ❖ Jednostavno je za ažuriranje datoteka (naročito drajver modula) na korenskom sistemu datoteka bez ponovnog pokretanja
    - ❖ Mnogo je brže nego preko serijskog prolaza
  - ❖ Može se imati veliki korenski sistem datoteka čak iako ne postoji podrška za dodatno skladište podataka
    - ❖ Čak se mogu i izgraditi alati host sistema za prevođenje i svi ostali alati koji su nam potrebni na samom sistemu (što je bolje od unakrsnog prevođenja)



# Postavka za NFS pokretanje (1/2)

- ❖ Na eksternoj mašini (NFS poslužitelj)
  - ❖ U /etc/exports datoteku dodati sldeću liniju
  - /home/nfsroot 192.168.0.100(rw,no\_root\_squash,no\_subtree\_check)
- ❖ Start ili restart NFS poslužitelja (Debian, Ubuntu)  
`/etc/init.d/nfs-kernel-server restart`



# Postavka za NFS pokretanje (2/2)



- ❖ Na ciljanoj platformi (NFS korisnik)
- ❖ Prevedite kernel sa **CONFIG\_NFS\_FS=y**,  
**CONFIG\_IP\_PNP=y** (konfiguriše IP u vremenu pokretanja)  
i **CONFIG\_ROOT\_NFS=y**
- ❖ Pokrenite kernel sa komandnim opcijama:
  - ❖ **root=/dev/nfs**
  - ❖ **rw**
  - ❖ **ip=192.168.0.100**
  - ❖ **nfsroot=192.168.0.1:/home/nfsroot**



# Prvi program korisničkog prostora



- ❖ Specifira ga parametar komandne linije `/init`
  - ❖ `init=/bin/sh` ili `init=/sbin/init`
- ❖ Izvršava se na kraju pokretanja kernela
- ❖ Brine se o startovanju svih ostalih programa korisničkog prostora (sistemske servisi i korisnički programi)
- ❖ Dobija broj procesa (pid) `1`
  - ❖ Roditelj ili predak svih ostalih programa korisničkog prostora
  - ❖ Sistem neće dozvoliti njegovu terminaciju



## /linuxrc



- ❖ Kada se `init` parametar ne prosledi kernelu, to je program koji se uobičajeno izvršava kada se pokreće kernel pomoću init ramdiska
- ❖ Najčešće je to šel skripta koja je bazirana na lakoj verziji šela, poput `nash` i `busybox sh`
- ❖ Ova skripta može da implementira kompleksne zadatke: detekcija drajvera za učitavanje, postavljanje mreže, postavljanje particija, prebacivanje na novi korenski sistem datoteka



# Init program

- ❖ /sbin/init je podrazumevano drugi init program
- ❖ Brine se o startovanju sistemskih servisa i eventualno korisničkih sprega (sshd, X server...)
- ❖ Brine se o stopiranju sistemskih servisa, takođe
- ❖ Lak je, delimična implementacija je dostupna kroz BusyBox
  - ❖ Međutim, jednostavne skripte za pokretanje su često sasvim dovoljne u ugrađenim sistemima



Unakrsno prevodenje kernela (CROSS-COMPILING)

# PREVOĐENJE I POKRETANJE LINUKSA



# Unakrsno prevodenje kernela

- ❖ Kada se prevodi Linuks kernel za drugu CPU arhitekturu
  - ❖ Mnogo je brže nego nativno prevodenje, kad je ciljni sistem mnogo sporiji od GNU/Linuks radne stanice
  - ❖ Mnogo je lakše jer su alati za razvoj mnogo lakši za naći za GNU/Linuks radnu stanicu
  - ❖ Da bi napravili razliku sa nativnim prevodiocem, izvršioci unakrsnog prevodioca imaju prefiks po imenu ciljanog sistema, arhitekture i ponekad biblioteke:
    - ❖ mips-linux-gcc
    - ❖ m68k-linux-uclibc-gcc
    - ❖ arm-linux-gnueabihf-gcc



# Specifiranje unakrsnog izvršioca (1/3)

- ❖ CPU arhitektura i prefiks unakrsnog izvršioca su definisani kroz ARCH i CROSS\_COMPILE promenljive u vrhovnom Makefile-u
  - ❖ Makefile definiše CC = \$(CROSS\_COMPILE)gcc
  - ❖ Najlakše rešenje je da se modifikuje Makefile
  - ❖ Primer, ARM platforma, unakrsni prevodilac: arm-linux-gcc
    - ❖ ARCH ?= arm
    - ❖ CROSS\_COMPILE ?= arm-linux-



# Specifiranje unakrsnog izvršioca (2/3)

- ❖ Drugo rešenje je da se **ARCH** i **CROSS\_COMPILE** postave kroz **make** komandnu liniju
  - ❖ Objašnjenje: bilo koja promenljiva postavljena kroz **make** komandnu liniju nadglašava bilo koje drugo podešavanje u Makefile-u
  - ❖ Primeri:
    - ❖ **make ARCH=sh CROSS\_COMPILE=sh-linux- xconfig**
    - ❖ **make ARCH=sh CROSS\_COMPILE=sh-linux-**
    - ❖ **make ARCH=sh CROSS\_COMPILE=sh-linux- modules\_install**
  - ❖ Velika mana:
    - ❖ Nikad ne smeju da se zaborave ova podešavanja kada se pokreće **make**
    - ❖ Sklonost ka grešci je velika i nije pogodno za upotrebu



# Specifiranje unakrsnog izvršioca (3/3)

- ❖ Još jedno rešenje: postaviti **ARCH** i **CROSS\_COMPILE** kao promenljive okruženja u vašem terminalu:
  - ❖ `export ARCH=arm`
  - ❖ `export CROSS_COMPILE=arm-linux-`
- ❖ Može biti postavljeno u specifičnim okruženjima vezana za projekat
- ❖ Nije fiksirano u **Makefile-u** – ne smeta zakrpama
- ❖ Nema zaboravljanja postavljanja u **make** komandi
- ❖ Oprez: samo se odnosi na šelove u kojima su ove promenljive postavljene



# Postavka unakrsnog prevodenja

- ❖ Primer:
  - ❖ Ako postoji ARM alat za unakrsni prevodilac u **/usr/local/arm/3.3.2/**
  - ❖ Samo treba da se doda u Uniks putanju za pretragu: **export PATH=/usr/local/arm/3.3.2/bin:\$PATH**
- ❖ Odabir alata
  - ❖ Za detalje o zahtevima za minalnu verziju alata pogledati u kodu u **Documentation/Changes**



# Konfiguracija kernela

- ❖ make xconfig
- ❖ Ili menuconfig, gconfig, itd.
- ❖ Isto što i nativno prevodenje
- ❖ Ne zaboravite da postavite tačnu ploču ili tip mašine



# Već postojeće config datoteke

## ❖ arch/arm/configs example

acs5k_defconfig	fig	hackkit_defconfig
acs5k_tiny_defconfig	colibri_pxa300_defcon	hisi_defconfig
am200epdkit_defconfig	fig	imote2_defconfig
assabet_defconfig	collie_defconfig	imx_v4_v5_defconfig
at91_dt_defconfig	corgi_defconfig	imx_v6_v7_defconfig
axm55xx_defconfig	davinci_all_defconfig	integrator_defconfig
badge4_defconfig	dove_defconfig	iop13xx_defconfig
bcm2709_defconfig	ebsa110_defconfig	iop32x_defconfig
bcm2835_defconfig	efm32_defconfig	iop33x_defconfig
bcm_defconfig	em_x270_defconfig	ixp4xx_defconfig
bcmrpipi_defconfig	ep93xx_defconfig	jornada720_defconfig
cerfcube_defconfig	eseries_pxa_defconfig	keystone_defconfig
clps711x_defconfig	exynos_defconfig	ks8695_defconfig
cm_x2xx_defconfig	ezx_defconfig	lart_defconfig
cm_x300_defconfig	footbridge_defconfig	lpc18xx_defconfig
cns3420vb_defconfig	h3600_defconfig	lpc32xx_defconfig
colibri_pxa270_defcon	h5000_defconfig	...



# Korišćenje već postojećih config datoteka



- ❖ Uobičajene konfiguracione datoteke su dostupne za mnoge ploče / mašine
  - ❖ Suštinski to su minimalne .config datoteke
  - ❖ Proverite da li postoji za vašu ciljnu ploču u arch/<arch>/configs/
  - ❖ Možete proveriti i pokretanjem make help
  - ❖ Najčešći način konfigurisanja kernela za ugrađene platforme
- ❖ Primer: ako ste pronašli acme\_defconfig datoteku, možete pokrenuti make acme\_defconfig
  - ❖ Ovo će izmeniti postojeću .config datoteku
- ❖ Kao i sve ostale make komande, morate pokrenuti make <machine>\_defconfig u korenu izvornog direktorijuma
- ❖ Možete napraviti i svoju config datoteku komandom
  - ❖ make savedefconfig



# Device Tree

- ❖ Mnoge ugrađene arhitekture imaju puno hardvera koji se ne može automatski otkriti
- ❖ Zavisno od arhitekture, takav hardver je ili opisan C kodom direktno unutar kernela ili korišćenjem posebnog jezika za opis hardvera u Device Tree
- ❖ ARM, PowerPC, OpenRISC, ARC, Microblaze su primeri arhitekture koje koriste Device Tree.
- ❖ Device Tree Source, koji piše programer, prevodi se u binarnu datoteku Device Tree Blob, koja se prosleđuje kernelu u toku boot procesa.
  - ❖ Postoji po jedna Device Tree datoteka za svaku ploču/platformu koju kernel podržava, dostupnu na putanji [arch/arm/boot/dts/<board>.dtb](#).
  - ❖ Bootloader mora pre pokretanja kernela da učita u memoriju i sliku kernela i Device Tree Blob.



# Izgradnja kernela

- ❖ Pokrenuti
  - ❖ make
- ❖ Kopirati
  - ❖ arch/<arch>/boot/zImage u skladište na ciljnoj platformi
- ❖ Možete izmeniti arch/<arch>/boot/install.sh tako da make install radi ovo automatski
- ❖ make modules\_install INSTALL\_MOD\_PATH=<dir>/ i kopirajte <dir>/ u /lib/modules/ u skladište na ciljnoj platformi ili ga već postavite na tu poziciju u slučaju NFS



# Rezime unakrsnog prevodenja

- ❖ Izmeniti **Makefile**: postaviti **ARCH** i **CROSS\_COMPILE** promenljive
- ❖ Preuzeti uobičajenu konfiguraciju za željenu mašinu: **make <machine>\_defconfig** (ako postoji u **arch/<arch>/configs**)
- ❖ Podesiti konfiguraciona podešavanja prema sopstvenim željama i zahtevima sa **make xconfig**
- ❖ Dodati putanju za unakrsni prevodilac u **PATH** promenljivu okruženja
- ❖ Prevesti kernel: **make**
- ❖ Kopirati kernel sliku iz **arch/<arch>/boot/** na ciljanu platformu
- ❖ Kopirati module u direktorijum koji se replicira na ciljnoj platformi **make INSTALL\_MOD\_PATH=<dir> modules\_install**



# Pokretanje pomoću U-Boot-a

- ❖ Novije verzije U-Boot-a mogu da pokrenu zImage datoteku
- ❖ Starije verzije su zahtevale specijalni format slike kernela – uImage
  - ❖ uImage se generiše iz zImage korišćenjem mkimage alata. Može se uraditi i automatski komandom make uImage
  - ❖ Na nekim ARM platformama se prilikom pokretanja komande make uImage mora proslediti i LOADADDR promenljiva okruženja, koja definiše sa koje adrese će kernel fizički biti pokrenut.
- ❖ Kao dodatak na sliku kernela, U-Boot može takođe da prosledi kernelu i Device Tree Blob datoteku
- ❖ Stoga je tipičan boot proces sledeći:
  - ❖ Učitaj zImage ili uImage na adresu X u memoriji
  - ❖ Učitaj <board>.dtb na adresu Y u memoriji
  - ❖ Pokreni kernel sa bootz X - Y (zImage slučaj), ili bootm X - Y (uImage slučaj)
  - ❖ Znak - u sredini ukazuje da se ne koristi initramfs



Učitavanje modula

# **PREVOĐENJE I POKRETANJE LINUXA**



# /dev – limitacije i problemi

- ❖ Na Red Hat-u 9, 18000 unosa ima u **/dev**!
  - ❖ Svi unosi za sve moguće drajvere moraju da budu kreirani pri instalaciji sistema
- ❖ Potreban je autoritet za dodelu *major* brojeva
- ❖ Nema dovoljno brojeva u 2.4, granice proširene u 2.6 verziji
- ❖ Korisnički prostor ne zna koji su uređaji prisutni u sistemu, niti koji pravi uređaj odgovara kom **/dev** unosu



# devfs rešenje i ograničenja

- ❖ devfs – prvo rešenje implementirano u Linuksu 2.3
  - ❖ Pokazuje samo prisutne uređaje
  - ❖ Koristi različita imena u /dev, što dovodi do problema u skriptama
  - ❖ Nema fleksibilnost u imenovanju uređaja, za razliku od /dev/, npr. prvi IDE disk uređaj morao je da se zove ili /dev/hda ili /dev/ide/hd/c0b0t0u0
  - ❖ Ne dozvoljava dinamičku alokaciju *major* i *minor* brojeva
  - ❖ Zahteva čuvanje načina imenovanja uređaja u kernel memoriji
    - ❖ Čuva se zauvek u RAMu kernela iako nije više potreban
  - ❖ devfs je kompletno uklonjen u Linuksu 2.6.18



# Pokretanje udev-a (1/2)

- ❖ Na samom početku pokretanja korisničkog prostora, postaviti **/dev/** direktorijum kao tmpfs sistem datoteka:  
**sudo mount -t tmpfs udev /dev**
- ❖ **/dev/** se populiše statičkim uređajima dostupnim u  
**/lib/udev/devices/**

Ubuntu 6.10 primer:

```
crw----- 1 root root 5, 1 2007-01-31 04:18 console
lrwxrwxrwx 1 root root 11 2007-01-31 04:18 core -> /proc/kcore
lrwxrwxrwx 1 root root 13 2007-01-31 04:18 fd -> /proc/self/fd
crw-r---- 1 root kmem 1, 2 2007-01-31 04:18 kmem
brw----- 1 root root 7, 0 2007-01-31 04:18 loop0
lrwxrwxrwx 1 root root 13 2007-01-31 04:18 MAKEDEV -> /sbin/MAKEDEV
drwxr-xr-x 2 root root 4096 2007-01-31 04:18 net
crw----- 1 root root 1, 3 2007-01-31 04:18 null
crw----- 1 root root 108, 0 2007-01-31 04:18 ppp
drwxr-xr-x 2 root root 4096 2006-10-16 14:39 pts
drwxr-xr-x 2 root root 4096 2006-10-16 14:39 shm
lrwxrwxrwx 1 root root 24 2007-01-31 04:18 sndstat -> /proc/asound/oss/sndstat
lrwxrwxrwx 1 root root 15 2007-01-31 04:18 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root 15 2007-01-31 04:18 stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 2007-01-31 04:18 stdout -> /proc/self/fd/1
```



## Pokretanje udev-a (2/2)

- ❖ **udevd** demon je startovan
  - ❖ Osluškuje **uevents** od jezgra drajvera, koji se šalju kad god se uređaj priključi ili isključi
- ❖ **udevd** demon čita i parsira sva pravila koja se nalaze u **/etc/udev/rules.d/** i čuva ih u memoriji
- ❖ Kad god se pravilo doda, ukloni ili izmeni, **udevd** primi **inotify** događaj i ažurira svoj set pravila u memoriji
- ❖ Kad se događaj primi, **udevd** startuje proces da:
  - ❖ Pronađe događaj u udev pravilima
  - ❖ Kreira / ukloni datoteke za uređaj
  - ❖ Pokrene programe (učitavanje ili uklanjanje drajvera, obaveštavanje korisničkog prostora...)



# Upravljanje redosledom događaja

- ❖ Udevd se brine od procesuiranju događaja u pravom redosledu
  - ❖ Ovo je korisno za procesuiranje događaja nakon drugih od kojih ti događaji zavise
- ❖ udevd ograničava broj procesa koji pokreće
  - ❖ Kad je broj prevaziđen, samo događaji sa TIMEOUT ključem se momentalno obrađuju
- ❖ /etc/.udev/queue/ direktorijum predstavlja trenutno pokrenuti ili događaj na čekanju
  - ❖ Sadrži simboličke veze ka odgovarajućim sysfs uređajima
  - ❖ Direktorijum se uklanja nakon uklanjanja poslednje veze
- ❖ Neuspele obrade događaja su predstavljene u /etc/.udev/failed/
  - ❖ Veze ovde su uklonjene kad se događaj za isti uređaj uspešno obradi



# netlink soketi

- ❖ Kernel netlink soketi se koriste za prenos uevents
- ❖ Prednosti:
  - ❖ Asinhroni su. Poruke se stavljaju u red. Prihvatalac može da bira poruke za obradu
  - ❖ Ostala komunikacija na relaciji korisnički prostor – kernel prostor je sinhrona: sistemski pozivi, ioctl, /proc/ i /sys/
  - ❖ Sistemski pozivi moraju biti prevedeni staticki u kernel
    - ❖ Drajveri uređaja bazirani na modulima ih ne mogu dodati
  - ❖ Multitasking je dostupan – više aplikacija može biti manjeno odjednom



# uevent premier poruke

## ❖ Primer priključenja USB miša

```
recv(4,                                // socket id
"add@/class/input/input9/mouse2\0        // message
ACTION=add\0                            // action type
DEVPATH=/class/input/input9/mouse2\0      // path in /sys
SUBSYSTEM=input\0                        // subsystem (class)
SEQNUM=1064\0                           // sequence number
PHYSDEVPATH=/devices/pci0000:00/0000:00:1d.1/usb2/2-2/2-2:1.0\0
                                         // device path in /sys
PHYSDEVBUS=usb\0                         // bus
PHYSDEVDRIVER=usbhid\0                  // driver
MAJOR=13\0                               // major number
MINOR=34\0",                             // minor number
2048,                                    // message buffer size
0)                                         // flags
= 221                                     // actual message size
```



## udev pravila

- ❖ Kada se pronađe da je odgovarajuće udev pravilo tačno, može biti korišćeno za:
  - ❖ Definisanje imena i putanje datoteke uređaja
  - ❖ Definisanje vlasnika, grupe i dozvola za datoteku uređaja
  - ❖ Izvršavanje naznačenog programa
- ❖ Pravila se obrađuju u leksičkom redosledu



# Udevo sposobnosti imenovanja

- ❖ Imena uređaja mogu biti definisana preko:
  - ❖ labele ili serijskog broja
  - ❖ broja uređaja magistrale
  - ❖ lokacije u topologiji magistrale
  - ❖ imena kernela
  - ❖ izlaza iz programa



# udev pravila imenovanja - primer

```
# Imenovanje testiranja izlaza programa
BUS=="scsi", PROGRAM="/sbin/scsi_id", RESULT=="OEM 0815", NAME="disk1"

# USB štampač će se zvati lp_color
BUS=="usb", SYSFS{serial}=="W09090207101241330", NAME="lp_color"

# SCSI disk sa specifičnom proizvođačem i brojem modela će se zvati boot
BUS=="scsi", SYSFS{vendor}=="IBM", SYSFS{model}=="ST336", NAME="boot%n"

# Zvučna kartica sa PCI magistralom id 00:0b.0 će se zvati dsp
BUS=="pci", ID=="00:0b.0", NAME="dsp"

# USB miš na trećem prolazu drugog hub-a će se zvati mouse1
BUS=="usb", PLACE=="2.3", NAME="mouse1"

# ttyUSB1 bi trebalo uvek da se zove pda sa dva dodatna symlink-a
KERNEL=="ttyUSB1", NAME="pda", SYMLINK="palmtop handheld"

# više USB web kamera sa symlink-ovima će se zvati webcam0, webcam1...
BUS=="usb", SYSFS{model}=="XV3", NAME="video%n", SYMLINK="webcam%n"
```



# udev primeri pravila dozvola

- ❖ Isešci iz `/etc/udev/rules.d/40-permissions.rules`

```
# Block devices
SUBSYSTEM!="block", GOTO="block_end"
SYSFS{removable}!="1", GROUP="disk"
SYSFS{removable}=="1", GROUP="floppy"
BUS=="usb", GROUP="plugdev"
BUS=="ieee1394", GROUP="plugdev"
LABEL="block_end"

# Other devices, by name
KERNEL=="null", MODE="0666"
KERNEL=="zero", MODE="0666"
KERNEL=="full", MODE="0666"
```



# Identifikovanje modula drajvera uređaja

- ❖ Prevodenje kernela / modula
  - ❖ Svaki drajver objavi koji uređaj i kojeg proizvođača podržava
  - ❖ depmod komanda obrađuje datoteke modula i generiše `/lib/modules/<version>/modules.alias`
- ❖ Svakodnevni rad sistema
  - ❖ Jezgro drajvera čita id uređaja, id prodavca i ostale atribute uređaja
  - ❖ Kernel šalje događaj ka udevd, postavljajući **MODALIAS** promenljivu okruženja, kodirajući podatke
  - ❖ udev događaj proces pokreće **modprobe \$MODALIAS**
  - ❖ **modprobe** pronađe modul za učitavanje u **modules.alias** datoteci



# Alijasi modula

- ❖ MODALIAS je promenljiva okruženja
- ❖ Primer za USB miš
  - ❖ MODALIAS=usb:v046DpC03Ed2000dc00dsc00dp00ic03isc01ip02
  - ❖ Odgovarajuća linija u /lib/modules/<version>/modules.alias: alias usb:v\*p\*d\*dc\*dsc\*dp\*ic03isc01ip02\* usbmouse



# udev modprobe primeri pravila

- ❖ Čak se i učitavanje modula radi sa **udev**
- ❖ Isečci iz **/etc/udev/rules.d/90-modprobe.rules**

```
ACTION!="add", GOTO="modprobe_end"

SUBSYSTEM!="ide", GOTO="ide_end"
IMPORT{program}="ide_media --export $devpath"
ENV{IDE_MEDIA}=="cdrom", RUN+="/sbin/modprobe -Qba ide-cd"
ENV{IDE_MEDIA}=="disk", RUN+="/sbin/modprobe -Qba ide-disk"
ENV{IDE_MEDIA}=="floppy", RUN+="/sbin/modprobe -Qba ide-floppy"
ENV{IDE_MEDIA}=="tape", RUN+="/sbin/modprobe -Qba ide-tape"

LABEL="ide_end"
SUBSYSTEM=="input", PROGRAM="/sbin/grepmap --udev", \
RUN+="/sbin/modprobe -Qba $result"
# Load drivers that match kernel-supplied alias
ENV{MODALIAS}=="?*", RUN+="/sbin/modprobe -Q $env{MODALIAS}"
```



# Koldplaging

- ❖ Problem: gubitak svih događaja uređaja koji se dešavaju tokom inicijalizacije kernela jer `udev` još uvek nije spremam
- ❖ Rešenje: nakon pokretanja `udevd`, kernel emituje uevents za sve uređaje prisutne u `/sys`
- ❖ Ovo može da se uradi alatom `udevtrigger`
- ❖ Velika korist: kompletno transparentan za korisnički prostor
  - ❖ Zaostalim i uklonjivim uređajima se rukuje i imenuju se na potpuno isti način



# Debagovanje događaja – udevmonitor (1/2)

- ❖ udevmonitor vizualizuje događaje jezgra drajvera i udev procese događaja (primer – povezivanje USB miša)

```
UEVENT [1170452995.094476] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2
UEVENT [1170452995.094569] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
UEVENT [1170452995.098337] add@/class/input/input28
UEVENT [1170452995.098618] add@/class/input/input28/mouse2
UEVENT [1170452995.098868] add@/class/input/input28/event4
UEVENT [1170452995.099110] add@/class/input/input28/ts2
UEVENT [1170452995.099353] add@/class/usb_device/usbdev4.30
UDEV [1170452995.165185] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2
UDEV [1170452995.274128] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
UDEV [1170452995.375726] add@/class/usb_device/usbdev4.30
UDEV [1170452995.415638] add@/class/input/input28
UDEV [1170452995.504164] add@/class/input/input28/mouse2
UDEV [1170452995.525087] add@/class/input/input28/event4
UDEV [1170452995.568758] add@/class/input/input28/ts2
```

❖ Informacije o vremenu se mere u mikrosekundama



# Debagovanje događaja – udevmonitor (2/2)

- ❖ udevmonitor -env prikazuje kompletno okruženje događaja za svaku liniju

```
UDEV [1170453642.595297] add@/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
UDEV_LOG=3
ACTION=add
DEVPATH=/devices/pci0000:00/0000:00:1d.7/usb4/4-3/4-3.2/4-3.2:1.0
SUBSYSTEM=usb
SEQNUM=3417
PHYSDEVBUS=usb
DEVICE=/proc/bus/usb/004/031
PRODUCT=46d/c03d/2000
TYPE=0/0/0
INTERFACE=3/1/2
MODALIAS=usb:v046DpC03Dd2000dc00dsc00dp00ic03isc01ip02
UDEVD_EVENT=1
```



# Ostali udev alati

- ❖ **udevinfo**
  - ❖ Daje korisnicima upit u **udev** bazu podataka
- ❖ **udevtest <sysfs\_device\_path>**
  - ❖ Simulira **udev** pokretanje da testira konfigurisana pravila



# Hotplaging firmvera

- ❖ Takođe implementirano pomoću **udev**
- ❖ Firmver podaci se drže izvan drajvera uređaja
  - ❖ Mogu biti nelegalni ili nedovoljno besplatni za distribuciju
  - ❖ Firmver u kernel kodu bi zauzeo memoriju permanentno, čak i da se koristi samo jednom
- ❖ Konfiguracija kernela: mora biti podešena u **CONFIG\_FW\_LOADER** (Device Drivers -> Generic Driver Options -> hotplug firmware loading support)



# udev datoteke

- ❖ /etc/udev/udev.conf
  - ❖ udev konfiguraciona datoteka
  - ❖ Najčešće se koristi za konfiguraciju syslog prioriteta izveštavanja
  - ❖ Primer podešavanja: `udev_log="err"`
- ❖ /etc/udev/rules.d/\*.rules
  - ❖ Pravila upoređivanja udev događaja.
- ❖ /lib/udev/devices/\*
  - ❖ statički /dev sadržaj (`/dev/console`, `/dev/null`...).
- ❖ /lib/udev/\*
  - ❖ Pomoćni programi koje pozivaju udev pravila
- ❖ /dev/\*
  - ❖ Kreirane datoteke uređaja



# Konfiguracija kernela za udev

- ❖ Kreiran za 2.6.19
- ❖ Primer:

```
# General setup
CONFIG_HOTPLUG=y
# Networking, networking options
CONFIG_NET=y
CONFIG_UNIX=
CONFIG_NETFILTER_NETLINK=y
CONFIG_NETFILTER_NETLINK_QUEUE=y
# Pseudo filesystems
CONFIG_PROC_FS=y
CONFIG_SYSFS=y
CONFIG_TMPFS=y
CONFIG_RAMFS=y
```



# Tipična operacija – udev sažetak

- ❖ Jezgro kernel drajvera
  - ❖ šalje uevent ka udevd
- ❖ Proces udev događaja
  - ❖ upoređuje događaj sa pravilima i kreira ili uklanja datoteke uređaja
- ❖ /lib/udev/ programi ili nešto drugo
  - ❖ Učitava odgovarajući modul
  - ❖ Obaveštava programe korisničkog prostora



# Resursi udev-a

## ❖ Početna stranica

- ❖ <https://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html>

## ❖ Izvorni kod

- ❖ <https://www.kernel.org/pub/linux/utils/kernel/hotplug/>

## ❖ Stranica za udev

- ❖ [man udev](#)



# mdev – udev za ugrađane sisteme



- ❖ Udev može biti pretežak za neke ugrađene sisteme, odnosno rad udevd demona koji u pozadini čeka događaje
- ❖ BusyBox obezbeđuje jednostavniju alternativu nazvanu mdev koje je dostupno uključivanjem MDEV konfiguracione opcije
- ❖ Upotreba mdev-a je dokumentovana u doc/mdev.txt u Busybox izvornom kodu
- ❖ Kao i udev, mdev može da učita firmvere u kernel



# Upotreba mdev-a

- ❖ Da bi koristili **mdev**, **proc** i **sysfs** sistemi datoteka moraju biti postavljeni
- ❖ **mdev** mora biti uključen kao hotplug upravljač  
`echo /bin/mdev > /proc/sys/kernel/hotplug`
- ❖ Potrebno je reći **mdev-u** da kreira **/dev** unose odgovarajuće detektovanim uređajima tokom pokretanja, dok **mdev** još uvek nije radio **mdev -s**
- ❖ Ponašanje je naznačeno u **/etc/mdev.conf** sa formatom  
`<device regex> <uid>:<gid> <octal permissions>`
- ❖ Primer: `hd[a-z][0-9]* 0:3 660`



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Razvoj ugrađenih sistema



# Sadržaj



- ❖ Razvoj ugrađenih sistema
  - ❖ Alati za prevođenje za drugu platformu
  - ❖ Emulatori
  - ❖ Razni alati
  - ❖ Busybox
  - ❖ http i ssh poslužioci
  - ❖ U-boot



Alati za prevodenje za drugu platformu

# РАЗВОЈ УГРАЂЕНИХ СИСТЕМА



# Obezbeđivanje alata za prevodenje



- ❖ Samostalno pravljenje alata za unakrsno prevodenje
  - ❖ Težak i naporan zadatak
  - ❖ Može da traje danima ili nedeljama!
  - ❖ Treba naučiti dosta detalja, mnogo odluka
  - ❖ Potrebna su kernel zaglavlja i izvorni kod C biblioteka
  - ❖ Potrebno je poznавање trenutnih `gcc` problema i zakrpa
- ❖ Preuzimanje prevedenog alata za prevodenje
  - ❖ Prednost: najjednostavnije i najzgodnije rešenje
  - ❖ Mane: ne postoji mogućnost za fina podešavanja alata prema potrebama
  - ❖ Treba se uveriti da pronađeni alat odgovara zahtevima



# Alati za prevodenje sa uClibc bibliotekom

- ❖ <http://free-electrons.com/community/tools/uclibc>
- ❖ Pokreću se na i386 arhitekturi
- ❖ Podržane platforme
  - ❖ Arm, armeb, i386, m68k, ppc, mips, mipsel, sh



# Alati za prevodenje za specifičnu platformu

- ❖ ARM
  - ❖ Code Sourcery (podržava GNU/Linuks, EABI i uClinux)
    - ❖ [http://www.codesourcery.com/gnu\\_toolchains/arm](http://www.codesourcery.com/gnu_toolchains/arm)
    - ❖ Dostupni i alati koji se pokreću na Windows platformama
- ❖ MIPS
  - ❖ <http://www.linux-mips.org/wiki/Toolchains/>



# Alati za generisanje prevodioca

- ❖ Buildroot
  - ❖ <http://buildroot.uclibc.org/>
  - ❖ Sistem za prevodenje zasnovan na make-u
  - ❖ Generiše alate za prevodenje sa uClibc bibliotekom ili čitave korenske sisteme datoteka
  
- ❖ Crosstool
  - ❖ <http://www.kegel.com/crosstool/>
  - ❖ Sistem za prevodenje zasnovan na skriptama
  - ❖ Generiše alate za prevodenje sa glibc bibliotekom
  - ❖ Dobavlja izvorni kod i primenjuje zakrpe



# Pravljenje korenskog sistema datoteka pomoću buildroot-a



- ❖ Podržava većinu arhitektura
- ❖ Automatski dobavlja izvorne kodove, primenjuje zakrpe i instalira programe koji su mu potrebni
- ❖ Pored korisničkih programa može da prevede većinu standardnih, često korišćenih programa
  - ❖ BusyBox, bzip2, Cairo, dbus, Dillo, DirectFB, Dropbear, lighthttpd, Python, Qtopia4, sqlite, thttpd, tinyX, xorg...
- ❖ Veoma lak za korišćenje
  - ❖ make menuconfig
  - ❖ make



# Alati za prevodenje za drugu platformu - pregled



- ❖ Samostalno pravljenje alata za prevodenje
  - ❖ Težak posao
  - ❖ Zahteva mnogo vremena da se savlada u potpunosti
- ❖ Spesjni alati za prevodenje za različite platforme
  - ❖ Mogu se pronaći na više lokacija
  - ❖ Većina platformi je podržana
- ❖ Alati za pomoć pri generisanju
  - ❖ Buildroot i Crosstool
  - ❖ Olakšavaju generisanje alata za prevodenje
  - ❖ Mogu da generišu i korenske sisteme datoteka
- ❖ Korisni materijali vezani za alate za prevodenje
  - ❖ [http://elinux.org/Tool Chain](http://elinux.org/Tool_Chain)



Emulatori

# RAZVOJ UGRAĐENIH SISTEMA



# qemu - korisnička emulacija

## ❖ Korisnička emulacija

- ❖ Može da pokreće programe prevedene za drugu platformu
  - ❖ Podržane arhitekture: x86, ppc, arm, sparc, mips, m68k
- ❖ Primer pokretanja BusyBox-a za arm arhitekturu na i386 GNU/Linuksu
  - ❖ `qemu-arm -L /usr/local/arm/3.3.2 \ /home/bart/arm/busybox-1.4.1/busybox ls`
  - ❖ -L: putanja do C biblioteke na odredištu (u ovom slučaju putanja do alata za prevodenje)



# ARM emulatori

- ❖ Samo besplatni programi
- ❖ SkyEye: <http://skyeye.sourceforge.net>
  - ❖ Emulira nekoliko ARM platformi i može da pokrene nekoliko različitih operativnih sistema (Linux, uLinux...)
- ❖ Softgun: <http://softgun.sourceforge.net>
  - ❖ Virtualni ARM sistem sa velikim brojem periferija
  - ❖ Pokreće Linux
- ❖ SWARM: Softverski ARM - arm7 emulator
  - ❖ <http://www.cl.cam.ac.uk/~mwd24/phd/swarm.html>
  - ❖ Može da pokreće uLinux



# Drugi emulatori

- ❖ ColdFire emulator
  - ❖ <http://www.slicer.ca/coldfire/>
  - ❖ Može da pokreće uClinux



# Emulatori - pregled

- ❖ Emulatori sistema
  - ❖ Korisni za isporbavanje čitavih sistema, uključujući i kernel
  - ❖ qemu: x86, x86\_64, am, sparc, ppc, mips
  - ❖ SkyEye: nekoliko arm arhitektura
- ❖ Korisnički emulatori
  - ❖ Korisni za pokretanje i debagovanje korisničkih programa namenjenih drugim arhitekturama
  - ❖ qemu: x86, arm, sparc, ppc, mips



Razni alati

# РАЗВОЈ УГРАЂЕНИХ СИСТЕМА



# chroot



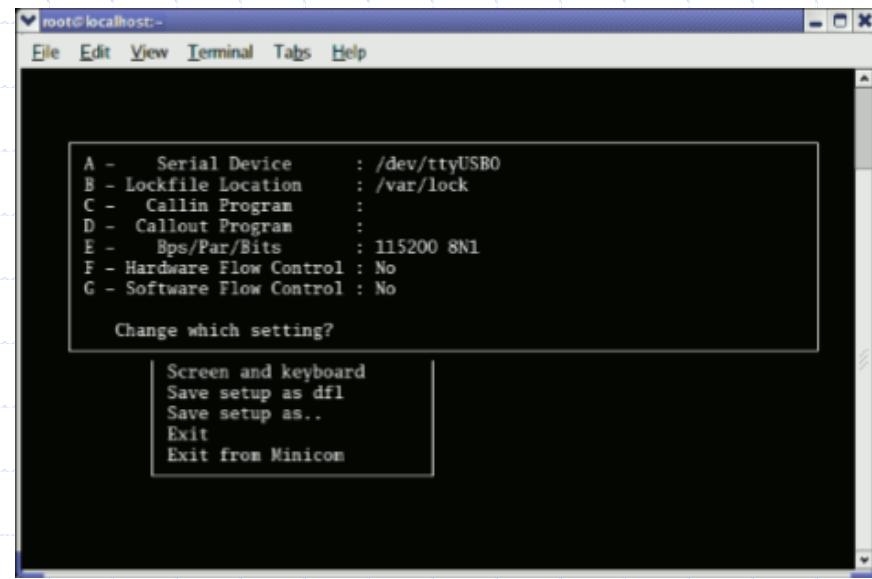
- ❖ Dostupan u svim GNU/Linuks distribucijama
- ❖ Korišćenje
  - ❖ chroot <dir> [komanda]
- ❖ Izvršava komande sa specijalnim korenskim direktorijskim putanjama
- ❖ Standardno korišćenje - internet serveri
  - ❖ Servisi se izvršavaju u zatvornom chroot okruženju: ukoliko dođe do greške servis ne može da pristupa ostatku sistema, pa ne može ni da ga ugrozi
- ❖ Korišćenje za ugrađene sisteme
  - ❖ Razvoj i testiranje novog korenskog sistema datoteka na lokalnom računaru
  - ❖ Jako lako za korišćenje kada lokalni i ciljni uređaj imaju isti set instrukcija



# Minicom



- ❖ Program za serijsku komunikaciju
- ❖ Dostupan u svim GNU/Linuks distribucijama
- ❖ Mogućnosti (sve se izvršava preko serijske veze):
  - ❖ Konzola na udaljenoj platformi
  - ❖ Transfer datoteka
  - ❖ Kontrola modema i dial-up
  - ❖ Konfiguracija serijske veze





# Drugi terminal emulatori

- ❖ GTKTerm: <http://www.jls-info.com/julien/linux/>
  - ❖ Grafički emulator
  - ❖ Jednostavniji i atraktivniji izgled u odnosu na Minicom ali manje mogućnosti
- ❖ CuteCom: <http://cute.com.sourceforge.net/>
  - ❖ Grafički emulator
  - ❖ Lak za korišćenje za nove korisnike
- ❖ GNU Screen: može da se koristi preko serijske veze
  - ❖ screen <device> <baudrate>
  - ❖ Primer
    - ❖ screen /dev/ttyS0 115200



BusyBox

# RAZVOJ UGRAĐENIH SISTEMA



# BusyBox



- ❖ <http://www.busybox.net/>
- ❖ Većina komandi Unix komandne linije u jednoj izvršnoj datoteci
- ❖ Uključuje i web server
- ❖ Veličina:
  - ❖ ~ 800 KB (statički povezana sa uClibc)
  - ❖ ~ 1200 KB (statički povezana sa eglibc)
- ❖ Lako se konfigurise
- ❖ Najbolji izbor za
  - ❖ Initramfs/initrd sa kompleksnim skriptama
  - ❖ Male i srednje ugrađene sisteme



# BusyBox komande

- ❖ addgroup, adduser, adjtimex, ar, arping, ash, awk, basename, bunzip2, bzcat, cal, cat, chgrp, chmod, chown, chroot, chvt, clear, cmp, cp, cpio, crond, crontab, cut, date, dc, dd, deallocvt, delgroup, deluser, devfsd, df, dirname, dmesg, dos2unix, dpkg, dpkgdeb, du, dumpkmap, dumpleases, echo, egrep, env, expr, false, fbset, fdflush, fdformat, fdisk, fgrep, find, fold, free, freeramdisk, fsck.minix, ftpget, ftpput, getopt, getty, grep, gunzip, gzip, halt, hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock, id, ifconfig, ifdown, ifup, inetd, init, insmod, install, ip, ipaddr, ipcalc, iplink, iproute, iptunnel, kill, killall, klogd, lash, last, length, linuxrc, ln, loadfont, loadkmap, logger, login, logname, logread, losetup, ls, lsmod, makedevs, md5sum, mesg, mkdir, mkfifo, mkfs.minix, mknod, mkswap, mktemp, modprobe, more, mount, msh, mt, mv, nameif, nc, netstat, nslookup, od, openvt, passwd, patch, pidof, ping, ping6, pipe\_progress, pivot\_root, poweroff, printf, ps, pwd, rdate, readlink, realpath, reboot, renice, reset, rm, rmdir, rmmod, route, rpm, rpm2cpio, run-parts, rx, sed, seq, setkeycodes, sha1sum, sleep, sort, start-stop-daemon, strings, stty, su, sulogin, swapoff, swapon, sync, sysctl, syslogd, tail, tar, tee, telnet, telnetd, test, tftp, time, top, touch, tr, traceroute, true, tty, udhcpc, udhcpd, umount, uname, uncompress, uniq, unix2dos, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat



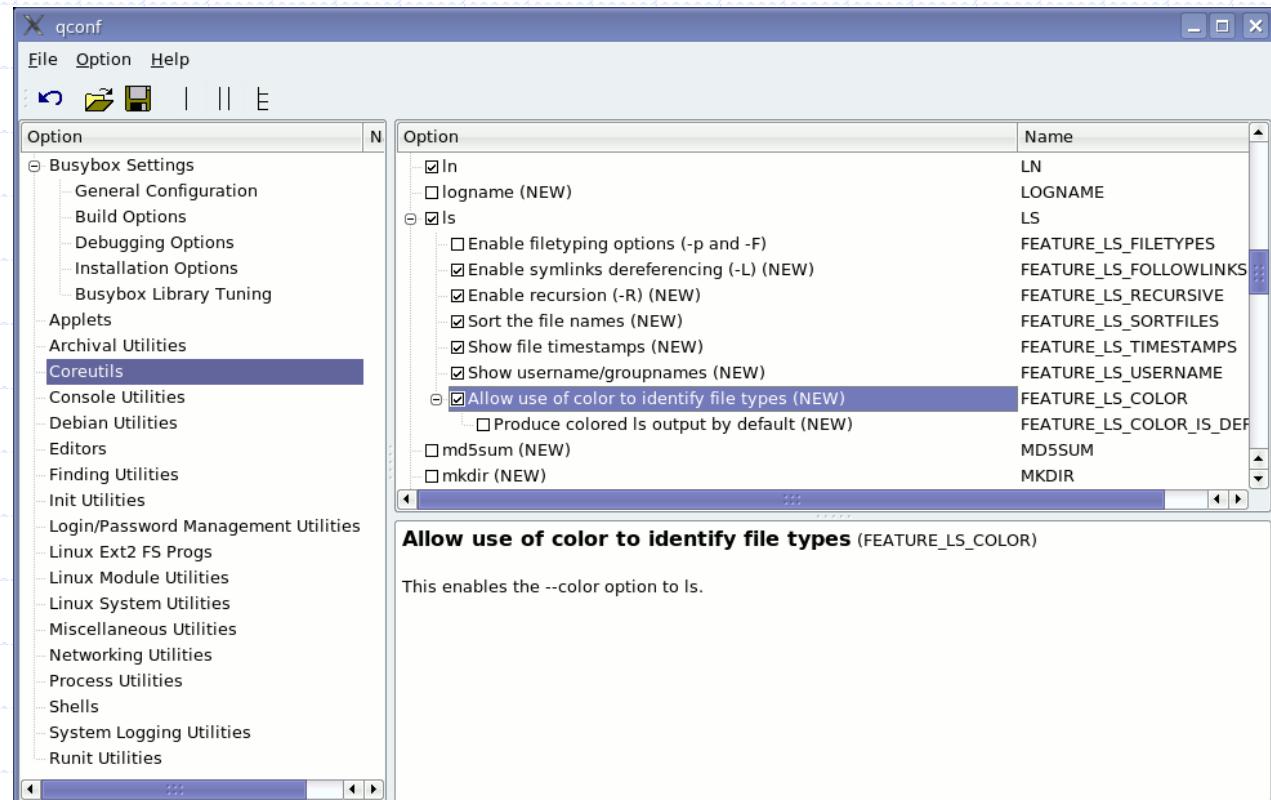
# Konfiguracija BusyBox-a

- ❖ Najnoviji izvorni kod se može pronaći na  
<http://busybox.net>
- ❖ Konfiguracija BusyBox-a (stvara .config datoteku)
  - ❖ make defconfig
    - ❖ Dobra opcija za početak korišćenja BusyBox-a
    - ❖ Konfiguriše BusyBox sa svim opcijama za običnog korisnika
  - ❖ make allnoconfig
    - ❖ Isključuje sve opcije
    - ❖ Koristi se kad se traži minimalni skup (posle ove komande se uključe samo opcije koje su potrebne)
  - ❖ make xconfig (grafički) ili make menuconfig (tekstualni)
    - ❖ Isti alati za konfiguraciju koji se koriste i za Linuks kernel



# BusyBox - make xconfig

- ❖ Mogu da se biraju
- ❖ komande koje se uključuju
- ❖ opcije komandi i mogućnosti





# Prevodenje BusyBox-a

- ❖ Ukoliko se prevodi za drugu platformu, potrebno je izabrati odgovarajuću arhitekturu ciljne platforme
  - ❖ ARCH ?= arm
  - ❖ CROSS\_COMPILE ?= arm-linux-
- ❖ Dodati putanju do alata za prevodenje u varijablu okruženja PATH
  - ❖ export PATH=/usr/local/arm/3.3.2/bin:\$PATH
- ❖ Prevodenje BusyBox-a
  - ❖ make
- ❖ Instalacija (stvara strukturu direktorijuma u \_install sa simboličkim linkovima na busybox izvršnu datoteku)
  - ❖ make install
- ❖ Posle logovanja mauntuje se slika sistema datoteka (npr. /mnt/rootfs) u ime korenskog korisnika
  - ❖ rsync -a \_install/ /mnt/rootfs/



# Pravljenje novog BusyBox apleta



- ❖ Korisno za pravljenje sopstvenih malih izvršnih datoteka
- ❖ Mogu da se koriste sve prednosti koje donosi BusyBox
- ❖ Dobijaju se manje datoteke nego kada bismo pravili izolovanu izvršnu datoteku
- ❖ Koriste se i prednosti koje pruža BusyBox vezane za konfiguraciju i prevodenje za druge platforme
- ❖ Licenca aplikacije će morati da bude GPL



# Novi BusyBox aplet - readahead primer (1/3)

- ❖ Dodate linije u (testirano sa BusyBox verzijom 1.21.1):
- ❖ include/applets.src.h:  
IF\_READAHEAD(APPLET(readahead, BB\_DIR\_USR\_BIN,  
BB\_SUID\_DROP))
- ❖ include/usage.src.h:  

```
#define readahead_trivial_usage \
"[FILE]..." "  
#define readahead_full_usage \
"Preloads FILE(s) in RAM cache so that subsequent reads for  
those files do not block on disk I/O."
```



# Novi BusyBox aplet - readahead primer (2/3)

- ❖ miscutils/Config.src

config CONFIG\_READAHEAD

    bool "readahead"

    default n

    depends on LFS

    help

    Preload the given list of files in RAM cache so  
    that subsequent reads on these files will not  
    block on disk I/O.

    This applet just calls the readahead(2)  
    system call on each file.

- ❖ miscutils/Kbuild.src

lib-\$(CONFIG\_READAHEAD) += readahead.o



# Novi BusyBox aplet - readahead primer (3/3)



## ❖ miscutils/readahead.c

```
/*
 * readahead implementation for BusyBox
 * Copyright (C) 2006 Michael Opdenacker <michael@free-electrons.com>
 * Licensed under GPLv2 or later, see file License in this tarball for details.
 */

#include "busybox.h"

int readahead_main(int argc, char **argv)
{
    FILE *f;
    int retval = EXIT_SUCCESS;

    if (argc == 1) bb_show_usage();

    while (*++argv) {
        if ((f = fopen_or_warn(*argv, "r")) != NULL) {
            int r, fd=fileno(f);
            r = readahead(fd, 0, fdlenth(fd));
            fclose(f);
            if (r >= 0) continue;
        }
        retval = EXIT_FAILURE;
    }
    return retval;
}
```



# Alternativa BusyBox-u: embutils



- ❖ <http://www.fefe.de/embutils/>
- ❖ Razvijen od strane kreatora diet libc
- ❖ Mali skup osnovnih komandi za komandnu liniju namenjen ugrađenim sistemima
- ❖ Isključivo se prevodi sa diet libc-om i povezuje se staticki
- ❖ U poređenju sa BusyBox-om manje je rasprostranjen i fale mu neke od osnovnih komandi i mogućnosti
- ❖ Postižu se manje veličine izolovanih izvršnih datoteka



<http://ssh.nsu.ac.rs> i <ssh://ssh.nsu.ac.rs>

# РАЗВОЈ УГРАЂЕНИХ СИСТЕМА



# ssh poslužilac i korisnik - dropbear



- ❖ <http://matt.ucc.asn.au/dropbear/dropbear.html>
- ❖ Mali memorjski otisak
- ❖ Sadrži i korisnika i poslužioca
- ❖ Namenjen za ugrađene sisteme
- ❖ Veličina
  - ❖ 110 KB (statički povezan sa uClibc na i386 arhitekturi)
  - ❖ OpenSSH korisnik i poslužilac: oko 1200 KB
- ❖ Omogućava
  - ❖ Udaljenu konzolu na ciljnoj platformi
  - ❖ Kopiranje datoteka na i sa ciljne platforme (scp ili rsync -e ssh)



# thttpd



- ❖ <http://acme.com/software/thttpd> (*Tiny/Turbo/Throttling HTTP server*)
- ❖ Jednostavan
  - ❖ implementira HTTP/1.1 minimum
  - ❖ jednostavan za konfiguraciju i pokretanje
- ❖ Mali
  - ❖ Izvršna datoteka - 88 KB
- ❖ Prenosiv
  - ❖ Prevodi se bez problema na većini UNIX zasnovanih operativnih sistema
- ❖ Brz
  - ❖ Približne brzine kao i veći poslužioci
- ❖ Siguran
  - ❖ Dizajniran da štiti mašinu od napada



# Drugi web poslužioci (1/2)

- ❖ BusyBox http poslužilac: <http://busybox.net>
  - ❖ Mali poslužilac (dodaje samo 9 KB veličini BusyBox-a)
  - ❖ Dovoljan set mogućnosti za većinu uređaja
  - ❖ Licenca: GPL
  
- ❖ Kfone: <http://koanlogic.com/kl/cont/gb/html/klone.html>
  - ❖ Mali web poslužilac sa punom funkcionalnošću namenjen ugrađenim sistemima
  - ❖ Veličina - oko 150 KB
  - ❖ Licenca: Dvostruka GPL/Commercial



# Drugi web poslužioci (2/2)

- ❖ Boa: <http://www.boa.org/>
  - ❖ Dizajniran da bude jednostavan, brz i siguran
  - ❖ Za razliku od thttpd ne vodi računa o memorijskom otisku
  - ❖ Prilično popularan u ugrađenim sistemima
  
- ❖ lighthttpd: <http://lighthtpd.net>
  - ❖ Mali memorijski otisak
  - ❖ Dobro podnosi velika opterećenja
  - ❖ Može biti koristan u ugrađenim sistemima



U-boot

# RAZVOJ UGRAĐENIH SISTEMA



# Priprema slike kernela za U-boot

- ❖ U-boot mora da ima dodatne informacije u slici kernela ili initrd-a
- ❖ mkimage - alat za pripremu slike kernela koji se nalazi u sklopu U-boot izvornog koda
- ❖ Pravljenje odgovarajuće slike kernela
  - ❖ make uImage



# Priprema slike initrd-a za U-boot

## ❖ mkimage

```
-n initrd \
-A arm \
-O linux \
-T ramdisk \
-C gzip \
-d rd-ext2.gz \
uInitrd
```

Ime  
Arhitektura  
Operativni sistem  
Tip  
Kompresija  
Ulazna datoteka  
Izlazna datoteka



# Prevodenje U-boot-a

- ❖ Izvorni kod U-boot-a se može preuzeti sa adrese  
<http://www.denx.de/wiki/UBoot>
- ❖ U direktorijumu sa izvornim kodom U-boot-a
  - ❖ Pronaći ime konfiguracije za željenu platformu u direktorijum include/configs (npr. omap1710h3.h)
- ❖ Konfiguracija U-boot-a
  - ❖ make omap1710h3\_config (.h se zameni sa \_config)
- ❖ Ukoliko je potrebno promeniti alate za prevodenje u Makefile datoteci
  - ❖ ifeq (\$(ARCH),arm)  
CROSS\_COMPILE = arm-linux-  
endif
- ❖ Prevodenje
  - ❖ make



# Prevodenje U-boot mkimage-a

- ❖ Ukoliko je U-boot već instaliran i potrebno je instalirati samo mkimage
- ❖ mkimage je u potpunosti nezavistan od arhitekture i platforme
- ❖ Konfigurisati U-boot za bilo koju arhitekturu i platformu
- ❖ Prevodenje
  - ❖ make
- ❖ Instalacija mkimage-a
  - ❖ cp tools/mkimage /usr/local/bin/



# Konfiguracija tftp-a (1/2)

- ❖ Česta pojava u toku razvoja je preuzimanje slike kernela preko mreže
- ❖ Instrukcije za xinetd bazirane sisteme
- ❖ Instalirati tftp-server paket ako je potrebno
- ❖ Ukloniti disable = yes u datoteci /etc/xinetd.d/tftp
- ❖ Kopirati datoteke u /tftpboot/ direktorijum (ili na drugu lokaciju ukoliko je tako navedeno u /etc/xinetd.d/tftp datoteci)
- ❖ Može se desiti da mora da se isključi SELinux u /etc/selinux/config datoteci
- ❖ Restartovati xinetd
  - ❖ /etc/init.d/xinetd restart



## Konfiguracija tftp-a (2/2)

- ❖ Instrukcije za GNU/Linuks sisteme bazirane na Debian-u
- ❖ **Instalirati tftpd-hpa paket** ako je potrebno
- ❖ Postaviti RUN\_DAEMON="yes" u datoteci /etc/default/tftpd-hpa
- ❖ **Kopirati datoteke u direktorijum**  
/var/lib/tftpboot
- ❖ U datoteci /etc/hosts.allow zameniti ALL : ALL@ALL : DENY sa ALL : ALL@ALL : ALLOW
- ❖ U datoteci /etc/hosts.deny zakomentarisati ALL : PARANOID
- ❖ **Restartovati poslužioca**
  - ❖ /etc/init.d/tftpd-hpa restart



# U-boot - terminal

- ❖ Povezati ciljnu platformu i lokalni računar serijskom vezom
- ❖ Nakon paljenja ploče u serijskoj konzoli bi trebalo da se dobije ovakav (ili sličan) ispis
  - ❖ U-Boot 1.1.2 (Aug 3 2004 - 17:31:20)  
RAM Configuration:  
Bank #0: 00000000 8 MB  
Flash: 2 MB  
In: serial  
Out: serial  
Err: serial  
u-boot #



# U-boot - Informacije o ploči

- ❖ u-boot # bdinfo  
DRAM bank = 0x00000000  
-> start = 0x00000000  
-> size = 0x00800000  
ethaddr = 00:40:95:36:35:33  
ip\_addr = 10.0.0.11  
baudrate = 19200 bps



# U-boot - Varijable okruženja (1/2)

- ❖ u-boot # printenv  
baudrate=19200  
ethaddr=00:40:95:36:35:33  
netmask=255.255.255.0  
ipaddr=10.0.0.11  
serverip=10.0.0.1  
stdin=serial  
stdout=serial  
stderr=serial  
u-boot # setenv serverip 10.0.0.2  
u-boot # printenv serverip  
serverip=10.0.0.2

Podešavanja mreže  
za TFTP  
i NFS



# U-boot - Varijable okruženja (2/2)



- ❖ Varijable okruženja mogu biti trajno sačuvane sa komandom saveenv
- ❖ Može se čak napraviti i mala šel skripta u varijabli okruženja
  - ❖ setenv myscript 'tftp 0x21400000 uImage; bootm 0x21400000'
- ❖ Ovakva skripta se može pokrenuti sa
  - ❖ run myscript
- ❖ Složenije skripte zahtevaju posebne datoteke za skripte koje se obrađuju sa mkimage-om



# U-boot - Mrežne komande

- ❖ u-boot # tftp 8000 u-boot.bin  
From server 10.0.0.1; our IP address is 10.0.0.11  
Filename 'u-boot.bin'.  
Load address: 0x8000  
Loading: #####  
done  
Bytes transferred = 95032 (17338 hex)
- ❖ Adresa i veličina prenešene datoteke se čuvaju u varijablama okruženja fileaddr i filesize



# U-boot - Rad sa SD karticom

## (1/3)



### ❖ Izlistavanje svih particija

mmc part

### ❖ Izlistavanje sadržaja particije

#### ❖ Za ext4 particije

ext4ls mmc 0:<broj particije>

#### ❖ Za fat particije

fatls mmc 0:<broj particije>

### ❖ Upis na postojeću particiju

fatwrite mmc 0:<broj particije> <adresa iz memorije sa koje čitamo> <ime datoteke> <koliko bajta upisujemo>

ext4write ...



# U-boot - Rad sa SD karticom (2/3)



## ❖ Čitanje sa particije u RAM memoriju

### ❖ Za ext4 particije

```
ext4load mmc 0:<broj particije> <adresa u  
memoriji na koju učitavamo> <ime datoteke>  
<broj bajta koliko iščitavamo> (broj bajta i adresa  
se unose u hexa obliku uvek)
```

### ❖ Za fat particije

```
fatload mmc 0:<broj particije> <adresa u  
memoriji na koju učitavamo> <ime datoteke>  
<broj bajta koliko iščitavamo>
```



# U-boot - Rad sa SD karticom (3/3)



## ❖ Prženje image-a particije

- ❖ učitati image preko tftp na neku adresu - X (po završenom preuzimanju ispiše se veličina datoteke - Y)
- ❖ izlistati particije i videti gde počinje particija koju hoćemo da pržimo - Z (adrese su prikazane u decimalnom obliku, a posle ih koristimo u hex)

## ❖ Komanda za prženje

```
mmc write <X> <Z u hexa obliku> <Y u hexa  
obliku izrazena kroz broj blokova od 512  
bajta>
```



# Komande za pokretanje kernela

- ❖ Podešavanje parametara kernela pri pokretanju
  - ❖ u-boot # setenv bootargs mem=64M \  
console=ttyS0,115200 init=/sbin/init \  
root=/dev/mtdblock0
- ❖ Pokretanje kernela sa date adrese (RAM)
  - ❖ bootz 0x01000000 - 0x02000000



# Korisni linkovi

- ❖ U-boot stranica:
  - ❖ <http://www.denx.de/wiki/UBoot/WebHome>
- ❖ Lep pregled U-boot-a:
  - ❖ <http://linuxdevices.com/articles/AT5085702347.html>
- ❖ U-boot priručnik:
  - ❖ <http://www.denx.de/wiki/view/DULG/UBoot>



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Osnove razvoja rukovalaca



# Sadržaj



- ❖ Osnove razvoja rukovalaca
  - ❖ Kernel moduli za učitavanje
  - ❖ Dodavanje koda na kernel stablo
  - ❖ Parametri modula



Kernel moduli za učitavanje

# OSNOVE RAZVOJA RUKOVALACA



# Kernel moduli za učitavanje (1/2)



- ❖ Moduli: dodaju neku funkcionalnost kernelu (rukovaoci, podrška za sistem datoteka, itd)
- ❖ Mogu biti učitani i isključeni bilo kad, odnosno kad je njihova funkcionalnost potrebna. Kada su učitani, imaju pun pristup celom adresnom prostoru kernela. Nema neke posebne zaštite.
- ❖ Korisni su da bi se veličina slike kernela držala na minimumu



# Kernel moduli za učitavanje (2/2)



- ❖ Korisni su za dostavljanje isključivo binarnih rukovalaca (loša ideja) bez potrebe za ponovnom izgradnjom kernela
  
- ❖ Moduli čine lakši razvoj rukovalaca bez ponovnog pokretanja: učitaj, proveri, isključi, ponovo izgradi, učitaj...



# Hello modul

- ❖ \_\_init:  
uklanja se  
nakon  
inicijalizacije  
(kernel modul  
ili statički  
prevedeno)
- ❖ \_\_exit:  
odbacuje se  
kada je modul  
preveden  
statički u  
kernel

```
/* hello.c */
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
static int __init hello_init(void) {
    pr_alert("Good Morrow");
    pr_alert("to this fair assembly.\n");
    return 0;
}
static void __exit hello_exit(void) {
    pr_alert("Alas, poor world, what treasure");
    pr_alert("hast thou lost!\n");
}
module_init(hello_init);
module_exit(hello_exit);
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Greeting module");
MODULE_AUTHOR("William Shakespeare");
```



# Hello modul - objašnjenje

- ❖ Zaglavljiva specifična za Linuks kernel: linux/xyz.h
  - ❖ Nema pristupa standardnoj C biblioteci, ovo je kernel programiranje
- ❖ Funkcija za inicijalizaciju
  - ❖ Poziva se svaki put kada se modul učita, povratna vrednost je kod greške (0 uspešno, negativna vrednost u slučaju greške)
  - ❖ Deklariše se module\_init() makroom: naziv funkcije nije važan iako je \_init() po konvenciji
- ❖ Funkcija za čišćenje
  - ❖ Poziva se kada se modul uklanja
  - ❖ Deklariše se module\_exit() makroom
  - ❖ Meta informacije se deklarišu makroima MODULE\_LICENSE(), MODULE\_DESCRIPTION() i MODULE\_AUTHOR()

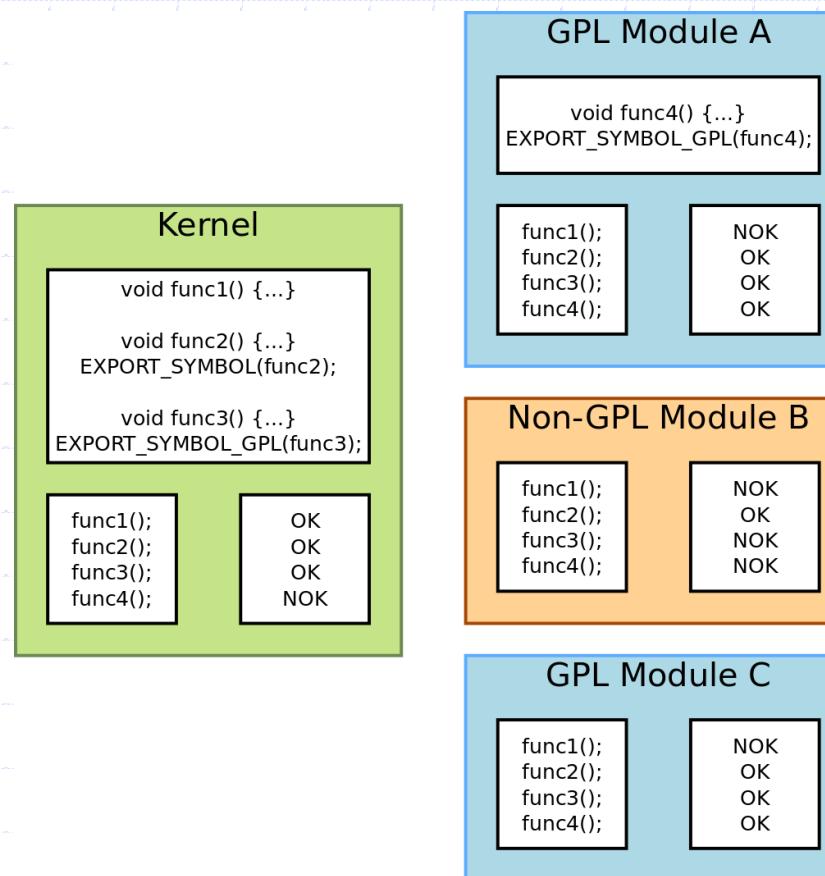


# Simboli eksportovani u module (1/2)

- ❖ Iz kernel modula, samo ograničen broj kernel funkcija može biti pozvan
- ❖ Funkcije i promenljive moraju eksplicitno biti eksportovane od strane kernela da bi bile vidljive u kernel modulu
- ❖ Dva makroa se koriste u kernelu da se eksportuju funkcije i promenljive:
  - ❖ EXPORT\_SYMBOL(symbolName) – eksportuje funkciju ili promenljivu svim modulima
  - ❖ EXPORT\_SYMBOL\_GPL(symbolName) – eksportuje funkciju ili promenljivu samo GPL modulima

# Simboli eksportovani u module (2/2)

- ❖ Korektnom rukovaocu ne bi trebali da su potrebne neeksportovane funkcije





# Zavisnosti modula

- ❖ Zavisnosti modula se čuvaju u  
`/lib/modules/<version>/modules.dep`
- ❖ Automatski se računaju tokom izgradnje kernela iz eksportovanih simbola u modulima. `modul2` zavisi od `modul1` ako `modul2` koristi simbol eksportovan od `modul1`
- ❖ Primer: `usb_storage` zavisi od `usbcore` zato što koristi neke funkcije koje su eksportovane od `usbcore`
- ❖ Možete ažurirati samo `modules.dep`, pokretanjem (kao root) `depmod -a [<version>]`



## Korisnost licence modula

- ❖ Koriste je ljudi koji razvijaju kernel da identifikuju probleme koji proističu iz vlasničkih rukovalaca, gde ne mogu ništa da učine (Tainted poruka)
- ❖ Korisna je za korisnike da provere da je njihov sistem 100% besplatan (/proc/sys/kernel/tainted)
- ❖ Korisna je za distributere GNU/Linuksa za njihove provere za izdavanje polisa



# Mogući stringovi licence modula

- ❖ Dostupni stringovi za licence su dati u include/linux/module.h
  
- ❖ GPL - GNU Public License v2 ili kasnija
- ❖ GPL v2 - GNU Public License v2
- ❖ GPL i dodatna prava
- ❖ Dual MIT/GPL - GNU Public License v2 ili MIT
- ❖ Dual BSD/GPL - GNU Public License v2 ili BSD
- ❖ Dual MPL/GPL - GNU Public License v2 ili Mozilla
- ❖ Proprietary – Proizvodi koji nisu besplatni



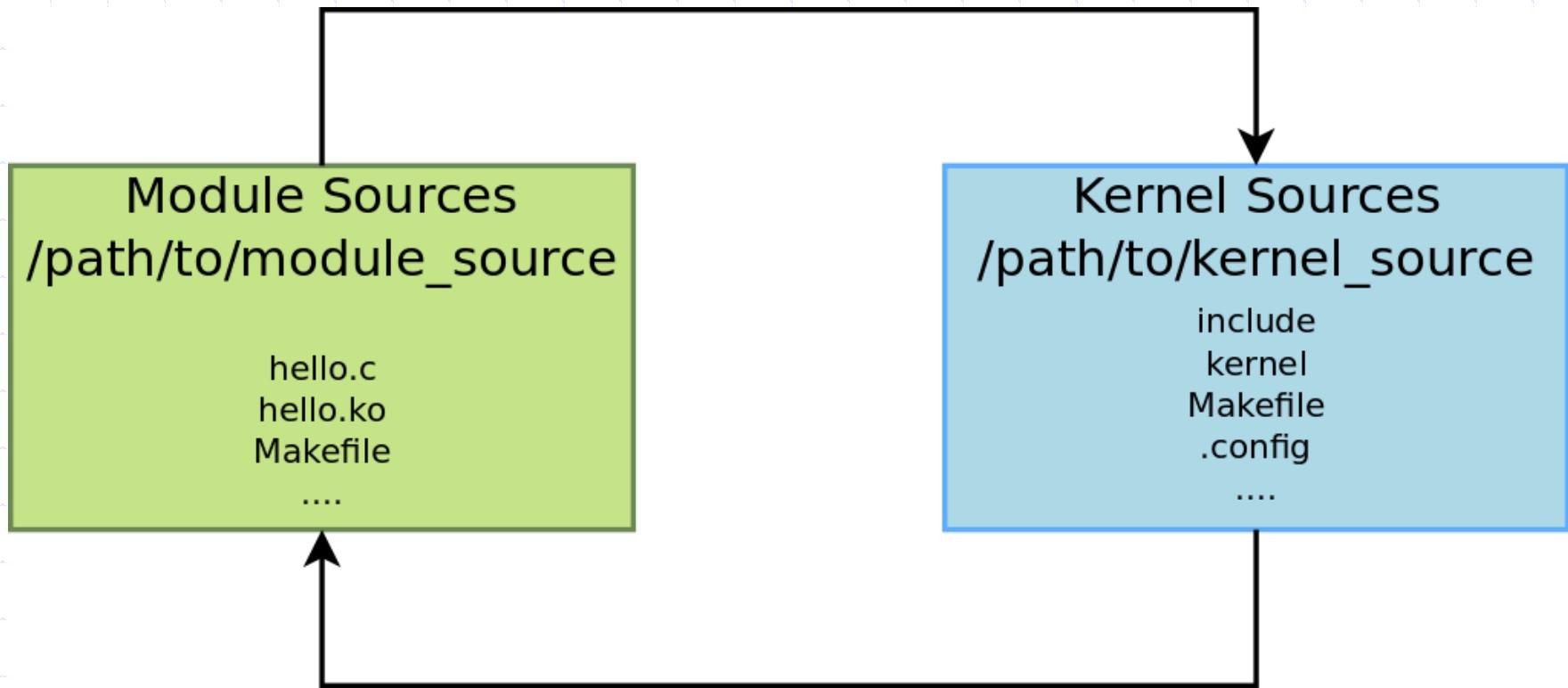
# Prevodenje modula (out of tree) (1/2)



- ❖ Makefile primer bi trebao biti ponovo iskoristiv za bilo koji Linuks modul
- ❖ Treba samo pokrenuti make da se izgradi hello.ko datoteka
- ❖ Oprez: budite sigurni da je [ TAB ] karakter na početku \$(MAKE) linije (sintaksa za make)

```
# Makefile for the hello module
ifeq ($(KERNELRELEASE),)
    obj-m := hello.o
else
    KDIR := /path/to/kernel/sources
all:
    $(MAKE) -C $(KDIR) M=$$PWD
endif
```

# Prevodenje modula (out of tree) (2/2)





# Log kernela

- ❖ Naravno, kernel ne čuva svoj log u datoteci jer datoteke pripadaju korisničkom prostoru
- ❖ Kernel čuva printk poruke u kružnom baferu
  - ❖ printk (KERN\_ALERT "...")
    - ❖ KERN\_ALERT, KERN\_CRIT, KERN\_ERR, KERN\_WARNING  
KERN\_NOTICE, KERN\_INFO, KERN\_DEBUG,  
KERN\_DEFAULT
    - ❖ pr\_alert ("...")
      - ❖ pr\_alert, pr\_crit, pr\_err, pr\_warning/pr\_warn,  
pr\_info, pr\_debug, pr-devel, ...
  - ❖ Kernel logovima može da se pristupi iz korisničkog prostora kroz sistemske pozive ili kroz /proc/kmsg
  - ❖ Kernel logovi su takođe prikazani u sistemskoj konzoli



# Pristupanje kernel logovima

- ❖ Postoji mnogo načina
  - ❖ Posmatranjem sistemske konzole
  - ❖ syslogd /klogd
    - ❖ Demon za prikupljanje kernel poruka u /var/log/messages
    - ❖ Mogu da se prate promene sa tail komandom
    - ❖ Pošto ova datoteka raste u veličini, treba koristiti logrotate da se to iskontroliše
  - ❖ cat /proc/kmsg
    - ❖ Čeka na kernel poruke i prikazuje ih
    - ❖ Korisno je kad imamo mali sistem
  - ❖ dmesg (dijagnostička poruka)
    - ❖ Nalazi se u svim sistemima i prikazuje bafer kernel loga



# Korišćenje modula

- ❖ Učitaj modul
  - ❖ sudo insmod ./hello.ko
- ❖ Videćete sledeće u kernel logu
  - ❖ Good Morrow to this fair assembly
- ❖ Uklonite modul
  - ❖ sudo rmmod hello
- ❖ I sad će se videti
  - ❖ Alas, poor world, what treasure hast thou lost!



# Razumevanje problema učitavanja modula

- ❖ Kada učitavanje modula ne uspe, insmod često ne daje dovoljno detalja
- ❖ Detalji su dostupni u kernel logu
- ❖ Primer:
  - ❖ > sudo insmod ./intr\_monitor.ko
  - ❖ insmod: error inserting './intr\_monitor.ko': -1 Device or resource busy
  - ❖ > dmesg
  - ❖ [17549774.552000] Failed to register handler for irq channel 2



# Korisni saveti za rad sa modulima (1/3)



- ❖ modinfo <module\_name>
- ❖ modinfo <module\_path>.ko
  - ❖ Daje informacije o modulu: parametri, licenca, opis i zavisnosti
  - ❖ Korisno je pre nego što odlučimo da li da učitamo modul ili ne
- ❖ sudo insmod <module\_path>.ko
  - ❖ Pokušava da učita dati modul



# Korisni saveti za rad sa modulima (2/3)



- ❖ sudo modprobe <module\_name>
  - ❖ Najčešća upotreba modprobe: pokušaj učitavanja svih modula od kojih dati modul zavisi praćen učitavanjem datog modula
  - ❖ Postoji još mnogo dostupnih opcija
- ❖ lsmod
  - ❖ Prikazuje listu učitanih modula
  - ❖ Uporedite izlaz te komande sa sadržajem direktorijuma /proc/modules



# Korisni saveti za rad sa modulima (3/3)



- ❖ sudo rmmod <module\_name>
  - ❖ Pokušava da ukloni dati modul
  
- ❖ sudo modprobe -r <module\_name>
  - ❖ Pokušava da ukloni dati modul i sve module koji su zavisni a nisu više potrebni nakon uklanjanja



# kdevelop



- ❖ Čini lakisim kreiranje modula sa datim kosturom iz već postojećeg šablonata
- ❖ Može da se koristi za prevod modula

```
try main - file:/home/duke/play/testit/trymain/src/main.rb - KDevelop
File Edit View Project Build Debug Bookmarks Window Tools Settings Help
File Edit View Project Build Debug Bookmarks Window Tools Settings Help
trymain.rb trymainface.rb pref.rb main.rb
variable Value
app
  children Array (5 element(s))
    [0] #<KDE::Application:0x3...
    [1] #<Qt::Object:0x300490...
    [2] #<Qt::SessionManager:...
    [3] #<Qt::EventLoop:0x300...
    [4] #<Qt::Translator:0x3004...
  metaObject #<Qt::MetaObject:0x0>
  name "trymain"
  receivers Hash (1 element(s))
    ["aboutToQuit"]
      [0] #<Qt::Connection:0x300...
        memberName "shutDown"
        memberType SIGNAL
        object #<KDE::Application:0x3...
        args nil
        description "A KDE Application"
  args
  description
Expression to watch:
Application Diff Messages Find in Files Replace Konsole Breakpoints CTAGS Frame Stack F
1 #<Thread:0x3005b798 run> /home/duke/play/testit/trymain/src/main.rb:22
#1 /home/duke/play/testit/trymain/src/main.rb:22
```

The screenshot shows the KDevelop IDE interface. The title bar says "try main - file:/home/duke/play/testit/trymain/src/main.rb - KDevelop". The menu bar includes File, Edit, View, Project, Build, Debug, Bookmarks, Window, Tools, Settings, and Help. The toolbar has icons for file operations like Open, Save, Print, and Build. The central area has tabs for "trymain.rb", "trymainface.rb", "pref.rb", and "main.rb". The "trymain.rb" tab contains Ruby code for a KDE application, including `AboutData` and `CmdLineArgs` initialization. The "Variables / Watch" tool on the left shows a tree view of variables for the "app" object, including its children (five Qt objects), metaObject, name ("trymain"), receivers (one "aboutToQuit" signal with a connection to "shutDown"), and arguments. The bottom status bar shows tabs for Application, Diff, Messages, Find in Files, Replace, Konsole, Breakpoints, CTAGS, Frame Stack, and F.



Dodavanje koda na kernel stablo

# OSNOVE RAZVOJA RUKOVALACA



# Nov rukovalac u kernel kodu (1/2)

- ❖ Da biste dodali novi rukovalac u kernel kod:
  - ❖ Dodajte novu izvornu datoteku u odgovarajući izvorni direktorijum
  - ❖ Primer: drivers/usb/serial/navman.c
  - ❖ Opišite konfiguracionu spregu za novi rukovalac tako što ćete dodati sledeće linije u Kconfig datoteku u tom direktorijumu:

```
config USB_SERIAL_NAVMAN
    tristate "USB Navman GPS device"
    depends on USB_SERIAL
    help
```

To compile this driver as a module, choose M here: the module will be called navman.



# Nov rukovalac u kernel kodu (2/2)

- ❖ Dodajte liniju u Makefile datoteku baziranu na Kconfig podešavanju:

```
obj-$ (CONFIG_USB_SERIAL_NAVMAN) += navman.o
```

- ❖ Pokrenite make xconfig i pogledajte svoju novu opciju
- ❖ Pokrenite make i vaše nove datoteke se prevode
- ❖ Pogledajte Documentation/kbuild/ za detalje



# Kako se kreiraju Linuks zakrpe



- ❖ Preuzmite verziju kernela koju menjate (u primeru ovde 4.4.16)

- ❖ Napravite kopiju preuzetog koda

```
rsync -a linux-4.4.16/ linux-4.4.16-patch/
```

- ❖ Primenite svoje izmene na kopirani kod i testirajte ih

- ❖ Pokrenite make distclean da zadržite samo izvorne datoteke

- ❖ Kreirajte datoteku zakrpe

```
diff -Nurp linux-4.4.16/ linux-4.4.16-patch/ >  
patchfile
```

- ❖ Uvek poredite cele strukture izvornog koda (zbog patch -p1)

- ❖ Ime zakrpe bi trebalo da vas podseti čemu zakrpa služi



Parametri modula

# OSNOVE RAZVOJA RUKOVALACA



# Hello modul sa parametrima

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/moduleparam.h>
MODULE_LICENSE("GPL");
/* A couple of parameters that can be passed in: how many times we say
hello, and to whom */

static char *whom = "world";
module_param(whom, charp, 0);

static int howmany = 1;
module_param(howmany, int, 0);

static int __init hello_init(void)
{
    int i;
    for (i = 0; i < howmany; i++)
        pr_alert("(%d) Hello, %s\n", i, whom);
    return 0;
}
static void __exit hello_exit(void)
{
    pr_alert("Goodbye, cruel %s\n", whom);
}

module_init(hello_init);
module_exit(hello_exit);
```



# Deklarisanje parametra modula

```
#include <linux/moduleparam.h>
module_param(
    name, /* name of an already defined variable */
    type, /* either byte, short, ushort, int, uint, long,
            ulong, charp, or bool.
            checked at compile time! */
    perm /* for /sys/module/<module_name>/parameters/<param>
          0: no such module parameter value file */
);
```

## ❖ Primer:

```
int irq=5;
module_param(irq, int, S_IRUGO);
```



# Deklarisanje niza parametara modula



```
#include <linux/moduleparam.h>
module_param_array(
    name, /* name of an already defined array */
    type, /* same as in module_param */
    num, /* number of elements in the array, or NULL (no check?) */
    perm /* same as in module_param */
);
```

## ❖ Primer:

```
static int base[MAX_DEVICES] = {0x820, 0x840};
module_param_array(base, int, NULL, 0);
```



# Prosleđivanje parametara modulu

## ❖ Preko insmod-a

```
sudo insmod ./hello_param.ko howmany=2  
whom=universe
```

## ❖ Preko modprobe-a

- ❖ Postaviti parametre u /etc/modprobe.conf ili u bilo kojoj datoteci u /etc/modprobe.d/:

```
options hello_param howmany=2 whom=universe
```

## ❖ Preko kernel komandne linije kada je modul preveden statički u kernel

```
options hello_param.howmany=2  
hello_param.whom=universe
```



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Razvoj rukovalaca  
Prvi deo



# Sadržaj



- ❖ Razvoj rukovalaca
  - ❖ Karakter rukovaoci
  - ❖ Upravljanje memorijom
  - ❖ I/O memorija i portovi
  - ❖ mmap



Karakter rukovaoci

# RAZVOJ RUKOVALACA

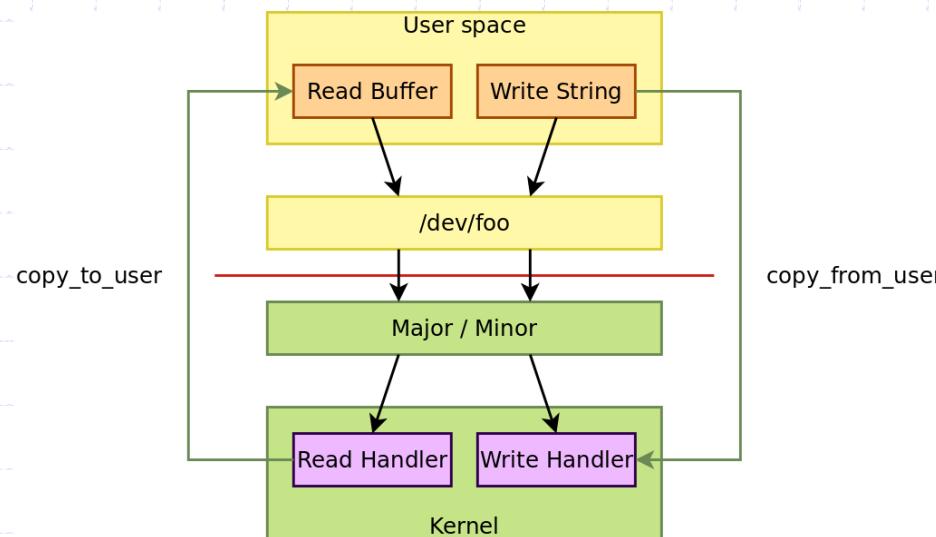


## Korist od karakter rukovalaca

- ❖ Osim rukovalaca za smeštajne uređaje, većina rukovalaca za uređaje sa I/O tokovima su implementirani kao karakter rukovaoci
  
- ❖ Tako da, većina rukovalaca sa kojima se susretnete će biti karakter rukovaoci

# Kreiranje karakter rukovalaca

- ❖ Potrebe korisničkog prostora
  - ❖ Ime datoteke uređaja u /dev za interakciju sa rukovaocem uređaja kroz operacije sa datotekama (open, read, write, close...)
- ❖ Potrebe kernela
  - ❖ Da zna koji rukovalac je zadužen za datoteke uređaja sa datim major, odnosno minor brojem
  - ❖ Za dati rukovalac, potrebno je imati upravljač da se izvrše operacije sa datotekama korisničkog prostora





# Deklarisanje karakter rukovalaca

- ❖ Registracija broja uređaja
  - ❖ Potrebno je registrovati jedan ili više brojeva uređaja (parova major / minor broj) u zavisnosti od broja uređaja kojima rukovalac upravlja
  - ❖ Potrebno je naći one slobodne
- ❖ Registracija operacija nad datotekama
  - ❖ Potrebno je registrovati upravljačke funkcije koje se zovu kad programi iz korisničkog prostora pristupe datotekama uređaja sa: open, read, write, ioctl, close...



# Informacije o registrovanim uređajima



- ❖ Registrovani uređaji su vidljivi u /proc/devices
- ❖ Može biti korišćeno za pronađak slobodnih major brojeva
  
- ❖ Broj predstavlja major broj
- ❖ Tekst predstavlja ime koje je registrovano

Character devices:

1 mem  
4 /dev/vc/0  
4 tty  
4 ttys  
5 /dev/tty  
5 /dev/console  
5 /dev/ptmx  
6 lp  
10 misc  
13 input  
14 sound

Block devices:

1 ramdisk  
3 ide0  
8 sd  
9 md  
22 ide1  
65 sd  
66 sd  
67 sd  
68 sd



## Dev\_t tip podataka

- ❖ Kernel tip podataka koji predstavlja par major/minor broj
- ❖ Zove se i broj uređaja
- ❖ Definisan je u <linux/kdev\_t.h>
- ❖ Linux 2.6: 32-bitna veličina (veliki: 12 bita, mali: 20 bita)
- ❖ Makro za kreiranje broja uređaja
  - ❖ MKDEV(int major, int minor);
- ❖ Makroi za ekstrakciju minor i major brojeva
  - ❖ MAJOR(dev\_t dev);
  - ❖ MINOR(dev\_t dev);



# Zauzimanje podešenih brojeva uređaja

```
#include <linux/fs.h>
int register_chrdev_region(
    dev_t from, /* Starting device number */
    unsigned count, /* Number of device numbers */
    const char *name); /* Registered name */
```

- ❖ Vraća 0 ako je zauzimanje bilo uspešno
- ❖ Primer:

```
if (register_chrdev_region(MKDEV(202, 128),
                           acme_count, "acme")) {
    pr_err("Failed to allocate device number\n");
    ...
}
```



# Dinamičko zauzimanje brojeva uređaja



- ❖ Sigurnije je jer kernel zauzima slobodne brojeve za nas

```
#include <linux/fs.h>
int alloc_chrdev_region(
    dev_t *dev, /* Output: starting device number */
    unsigned baseminor, /* Starting minor number, usually 0 */
    unsigned count, /* Number of device numbers */
    const char *name); /* Registered name */
```

- ❖ Vraća 0 ako je zauzimanje bilo uspešno
- ❖ Primer:

```
if (alloc_chrdev_region(&acme_dev, 0, acme_count, "acme")) {
    pr_err("Failed to allocate device number\n");
    ...
}
```



# Kreiranje datoteka uređaja

- ❖ Problem: ne mogu više da se kreiraju /dev unosi unapred
- ❖ Kreiraju se nakon učitavanja nekog rukovaoca prema zauzetom velikom broju
- ❖ Skripta koja učitava module može tada da koristi /proc/devices

```
module=foo; name=foo; device=foo
rm -f /dev/$device
insmod $module.ko
major=`awk "\$2==\"$name\" {print \$1}" /proc/devices`
mknod /dev/$device c $major 0
```

- ❖ Bolje: koristiti udev za automatsko kreiranje datoteke uređaja



# Operacije datoteka (1/6)

- ❖ Pre registrovanja karakter uređaja, morate definisati file\_operations (fops) za datoteke uređaja
- ❖ Ovo su najvažnije operacije za karakterne uređaje. Sve su opcione.

```
#include <linux/fs.h>

struct file_operations {
    ssize_t (*read) (struct file *, char __user *,
size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *,
size_t, loff_t *);
    long (*unlocked_ioctl) (struct file *, unsigned int,
unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*release) (struct inode *, struct file *);
};
```



# Operacije datoteka (2/6)

- ❖ int **foo\_open**(struct inode \***i**, struct file \***f**)
  - ❖ Poziva se kada se iz korisničkog prostora otvori datoteka uređaja
  - ❖ struct inode je struktura koja jedinstveno predstavlja datoteku u sistemu (bilo regularnu datoteku, simbolički link, direktorijum, karakterni ili blokovski uređaj)
  - ❖ struct file se prosleđuje svim drugim operacijama i predstavlja vezu sa jednim otvaranjem datoteke
  
- ❖ int **foo\_release**(struct inode \***i**, struct file \***f**)
  - ❖ Poziva se kada se iz korisničkog prostora zatvori datoteka uređaja



## struct file

- ❖ Kreira je kernel tokom open poziva, svaki put kada je datoteka uređaja otvorena. Predstavlja jednu otvorenu datoteku, a sve pokazuju na istu inode strukturu
- ❖ mode\_t f\_mode;
  - ❖ Mod otvaranja datoteke (FMODE\_READ i/ili FMODE\_WRITE)
- ❖ loff\_t f\_pos;
  - ❖ Trenutni offset u datoteci
- ❖ struct file\_operations \*f\_op;
  - ❖ Dozvoljava menjanje operacija datoteka za različite otvorene datoteke
- ❖ struct dentry \*f\_dentry
  - ❖ Korisno za dobijanje pristupa inode-u:  
`f_dentry->d_inode`



## Operacije datoteka (3/6)

- ❖ `ssize_t foo_read(struct file *f, char __user *buf, size_t sz, loff_t *off)`
  - ❖ Ova operacija se poziva kada korisnički prostor poziva funkciju za čitanje iz datoteke (`read()` sistemski poziv).
  - ❖ Mora da pročita podatke sa uređaja i upiše najviše `sz` bajtova u bafer korisničkog prostora `buf`, i osveži trenutnu poziciju u datoteci `off`. `f` je pokazivač na istu strukturu datoteke koja je prosleđena u `open()` operaciji
  - ❖ Mora da vrati broj pročitanih bajtova. 0 se najčešće sa strane korisničkog prostora interpretira kao kraj datoteke (*EOF*)
  - ❖ Na *UNIX*, `read()` operacija se tipično blokira dok nema dovoljno podataka da se pročitaju iz uređaja



## Operacije datoteka (4/6)

- ❖ `ssize_t foo_write(struct file *f, const char __user *buf, size_t sz, loff_t *off)`
  - ❖ Nasuprot operaciji `read`, mora da pročita najviše `sz` bajtova iz `buf`, upiše ih u uređaj, osveži `off` i vrati broj upisanih bajtova.
  - ❖ Ova operacija se poziva kada korisnički prostor poziva funkciju za upis u datoteku uređaja (`write()` sistemski poziv).



# Razmena podataka sa korisničkim prostorom (1/3)



- ❖ U kodu rukovaoca, ne može samo da se pozove memcpv između adrese koju dostavi korisnički prostor i adrese bafera u kernel prostoru
  - ❖ Odgovara kompletno drugačijem adresnom prostoru (zahvaljujući virtuelnoj memoriji)
  - ❖ Adrese korisničkog prostora mogu biti menjane ka disku
  - ❖ Adrese korisničkog prostora mogu biti nevalidne (korisnički prostor može da pokuša pristup neautorizovanim podacima)



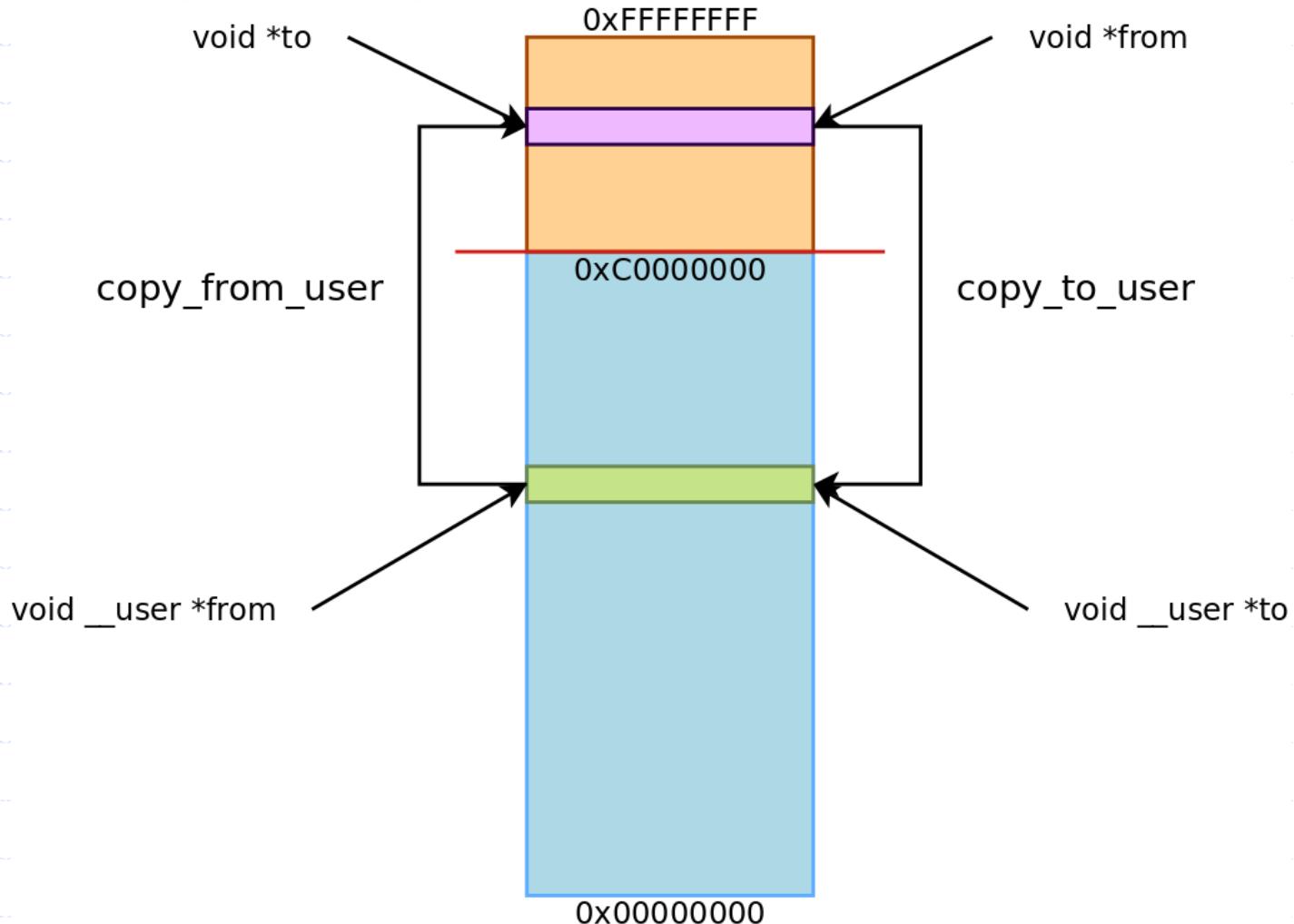
# Razmena podataka sa korisničkim prostorom (2/3)

- ❖ Moraju se koristiti posebne funkcije u read i write operacijama datoteka

```
#include <asm/uaccess.h>
unsigned long copy_to_user (void __user *to,
    const void *from,
    unsigned long n);
unsigned long copy_from_user (void *to,
    const void __user *from,
    unsigned long n);
```

- ❖ Postarati se da ove funkcije vrate 0
- ❖ Druge vrednosti označavaju neuspeh
- ❖ Alternativa za rad bajt po bajt
  - ❖ put\_user
  - ❖ get\_user

# Razmena podataka sa korisničkim prostorom (3/3)





## Operacije datoteka (5/6)

- ❖ `int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);`
- ❖ Može da se koristi za slanje specifičnih komandi uređaju, koje nisu ni čitanje ni pisanje (npr. menjanje brzine serijskog prolaza, postavljanje formata video izlaza, zahtev za serijskim brojem uređaja...)



## unlocked\_ioctl()

- ❖ long unlocked\_ioctl(struct file \*f,  
                  unsigned int cmd, unsigned long arg)
- ❖ Povezana sa ioctl() sistemskim pozivom.
- ❖ Zove se unlocked jer ne zauzima Big Kernel Lock (više se ne koristi).
- ❖ cmd je broj koji identificuje operaciju koju treba obaviti
- ❖ arg je opcioni argument prosleđen kao treći parametar ioctl() sistemskog poziva. Može biti integer, adresa, itd.
- ❖ Semantika cmd i arg parametra je specifična za svaki rukovaoc.



# ioctl() primer - kernel strana

## ❖ Primer: drivers/misc/phantom.c

```
static long phantom_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
    struct phm_reg r;
    void __user *argp = (void __user *)arg;

    switch (cmd) {
        case PHN_SET_REG:
            if (copy_from_user(&r, argp, sizeof(r)))
                return -EFAULT; /* Do something */
            break;
        case PHN_GET_REG:
            if (copy_to_user(argp, &r, sizeof(r)))
                return -EFAULT; /* Do something */
            break;
        default:
            return -ENOTTY;
    }

    return 0;
}
```



# ioctl() primer - strana aplikacije

```
int main(void)
{
    int fd, ret;
    struct phm_reg reg;

    fd = open("/dev/phantom");
    assert(fd > 0);

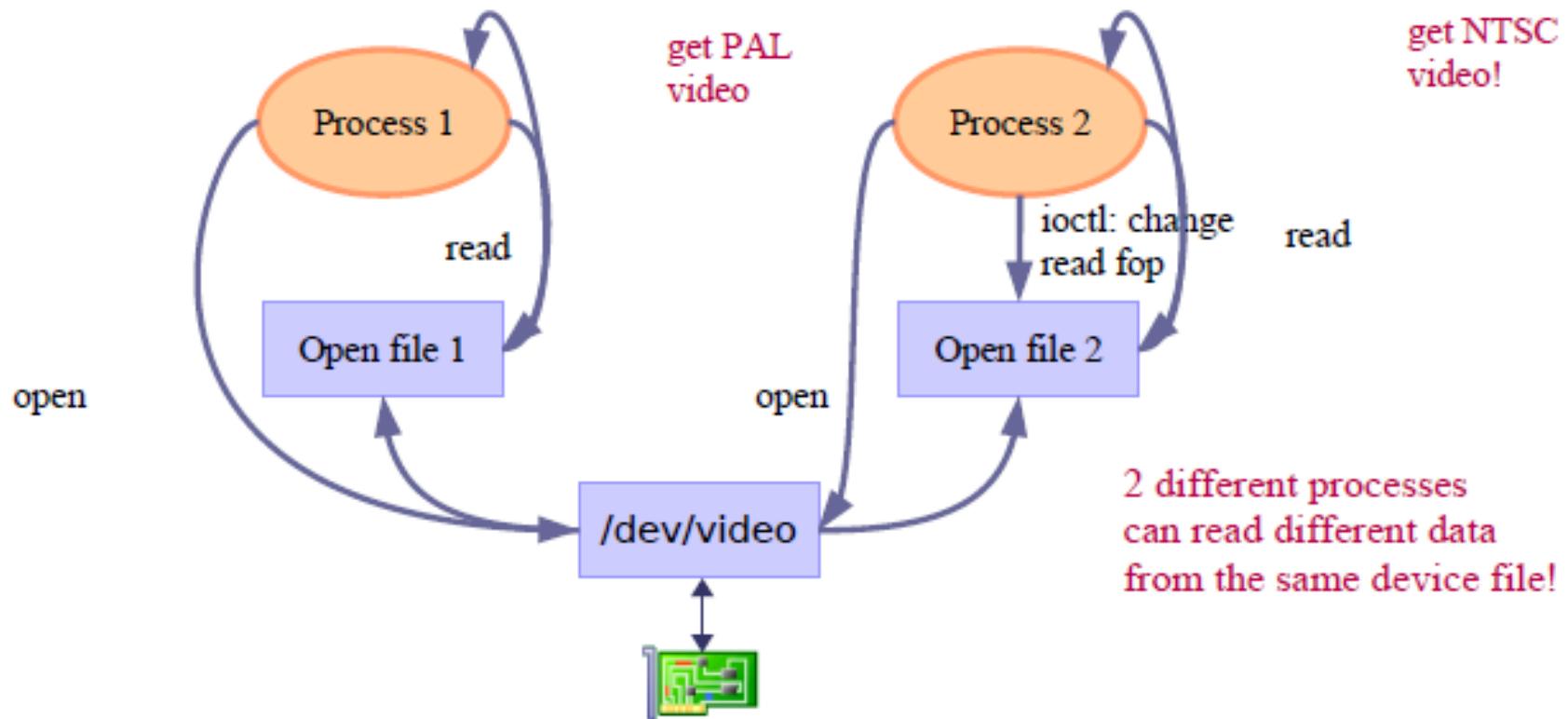
    reg.field1 = 42;
    reg.field2 = 67;

    ret = ioctl(fd, PHN_SET_REG, & reg);
    assert(ret == 0);

    return 0;
}
```

# Operacije su specifične za svaku otvorenu datoteku

- ❖ Koristi se mogućnost redefinisanja operacija za svaku otvorenu datoteku





# Operacije datoteka (6/6)

- ❖ `int (*mmap) (struct file *, struct vm_area_struct *);`
- ❖ Zahteva od memorije uređaja da se mapira u adresni prostor korisničkog procesa
- ❖ Pređene funkcije su glavne
  - ❖ Oko 25 operacija datoteka mogu biti postavljene, što odgovara svim sistemskim pozivima koji mogu biti izvršeni nad otvorenim datotekama



# Read primer

```
static ssize_t
acme_read(struct file *file, char __user *buf, size_t count, loff_t *ppos)
{
    /* The acme_buf address corresponds to a device I/O memory area */
    /* of size acme_bufsize, obtained with ioremap() */
    int remaining_size, transfer_size;
    remaining_size = acme_bufsize - (int) (*ppos); // bytes left to transfer
    if (remaining_size == 0) { /* All read, returning 0 (End Of File) */
        return 0;
    }
    /* Size of this transfer */
    transfer_size = min(remaining_size, (int) count);
    if (copy_to_user(buf /* to */, acme_buf + *ppos /* from */, transfer_size)) {
        return -EFAULT;
    } else { /* Increase the position in the open file */
        *ppos += transfer_size;
        return transfer_size;
    }
}
```



# Write primer

```
static ssize_t
acme_write(struct file *file, const char __user *buf, size_t count, loff_t
*ppos)
{
    int remaining_bytes;
    /* Number of bytes not written yet in the device */
    remaining_bytes = acme_bufsize - (*ppos);
    if (count > remaining_bytes) {
        /* Can't write beyond the end of the device */
        return -EIO;
    }
    if (copy_from_user(acme_buf + *ppos /* to */, buf /* from */, count)) {
        return -EFAULT;
    } else {
        /* Increase the position in the open file */
        *ppos += count;
        return count;
    }
}
```



# Primer definicija operacija datoteka



## ❖ Definisanje file\_operations strukture

```
#include <linux/fs.h>
static struct file_operations acme_fops =
{
    .owner = THIS_MODULE,
    .read = acme_read,
    .write = acme_write,
};
```

- ❖ Samo je potrebno da se dostave funkcije koje su implementirane
- ❖ Ako ništa nije novo implementirano, koristiće se postojeće funkcije



# Registrovanje karakter uređaja (1/2)



- ❖ Kernel predstavlja karakter uređaje cdev strukturu
- ❖ Struktura se deklariše globalno (u modulu)
  - ❖ #include <linux/cdev.h>
  - ❖ static struct cdev acme\_cdev;
- ❖ U init funkciji se inicijalizuje struktura
  - ❖ cdev\_init(&acme\_cdev, &acme\_fops);



# Registrovanje karakter uređaja (2/2)



❖ Nakon toga, struktura je spremna da se doda u sistem

❖ `int cdev_add(struct cdev *p, dev_t dev,  
unsigned count);`

❖ Primer:

❖ `if (cdev_add(&acme_cdev, acme_dev,  
acme_count)) {  
pr_err("Char driver registration  
failed\n");  
...  
}`



# Odjava karakter uređaja

- ❖ Prvo se obriše karakter uređaj

  - ❖ `void cdev_del(struct cdev *p);`

- ❖ Nakon, i samo nakon toga se oslobođa broj uređaja

  - ❖ `void unregister_chrdev_region(dev_t from,  
unsigned count);`

- ❖ Primer:

  - ❖ `cdev_del(&acme_cdev);`
  - ❖ `unregister_chrdev_region(acme_dev,  
acme_count);`



# Kodovi greške Linuksa

- ❖ Pokušajte da prijavite greške sa brojevima što je tačnije moguće
- ❖ Na sreću, imena makroa su eksplisitna i mogu se lako zapamtiti
- ❖ Generičke greške:
  - ❖ include/asm-generic/errno-base.h
- ❖ Greške specifične za platformu:
  - ❖ include/asm/errno.h



# Rezime primera karakter uređaja (1/3)



```
static void *acme_buf;
static int acme_bufsize=8192;
static int acme_count=1;
static dev_t acme_dev;
static struct cdev acme_cdev;
static ssize_t acme_write(...) {...}
static ssize_t acme_read(...) {...}
static struct file_operations acme_fops =
{
    .owner = THIS_MODULE,
    .read = acme_read,
    .write = acme_write
};
```



# Rezime primera karakter uređaja (2/3)

```
static int __init acme_init(void)
{
    int err;
    acme_buf = ioremap(ACME_PHYS, acme_bufsize);
    if (!acme_buf) {
        err = -ENOMEM;
        goto err_exit;
    }
    if (alloc_chrdev_region(&acme_dev, 0, acme_count, "acme")) {
        err=-ENODEV;
        goto err_free_buf;
    }
    cdev_init(&acme_cdev, &acme_fops);
    if (cdev_add(&acme_cdev, acme_dev, acme_count)) {
        err=-ENODEV;
        goto err_dev_unregister;
    }
    ...
}
```



# Rezime primera karakter uređaja (3/3)



```
...
    return 0;

err_dev_unregister:
unregister_chrdev_region(acme_dev, acme_count);

err_free_buf:
iounmap(acme_buf);

err_exit:
return err;
}

static void __exit acme_exit(void)
{
    cdev_del(&acme_cdev);
    unregister_chrdev_region(acme_dev,
    acme_count);
    iounmap(acme_buf);
}
```

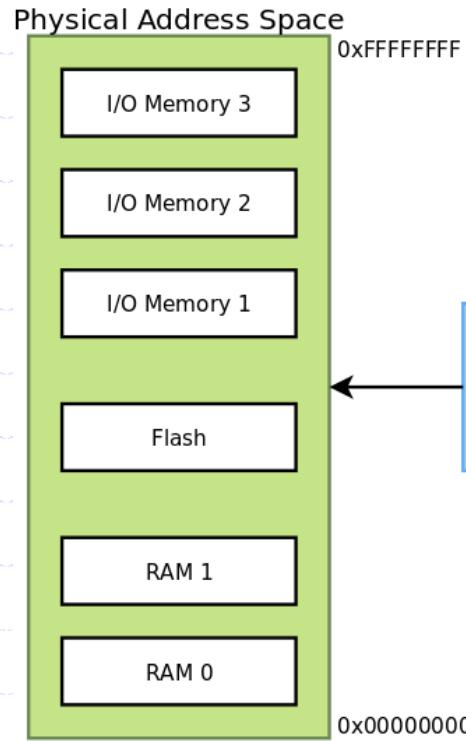


Rukovanje memorijom

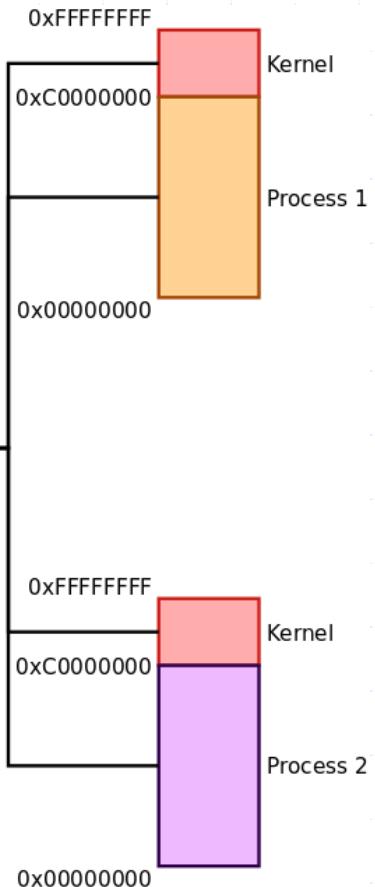
# RAZVOJ RUKOVALACA

# Fizička i virtuelna memorija

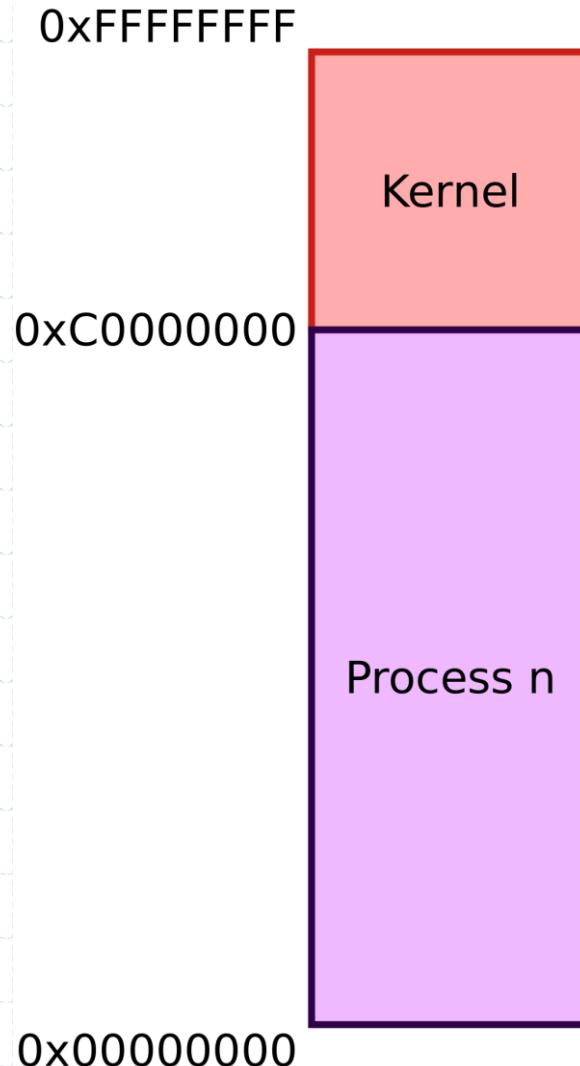
## Fizički adresni prostor



## Virtuelni adresni prostor



# Organizacija virtuelne memorije



- ❖ 1GB rezervisan za kernel prostor
  - ❖ Sadrži kod kernela i strukture podataka jezgra, jednake u svim adresnim prostorima
  - ❖ Veći deo memorije može biti direktno mapiran na fizičku memoriju na fiksnom odstojanju
- ❖ 3GB ekskluzivnog mapiranja je dostupno za svaki korisnički proces
  - ❖ Kod procesa i podaci (program, stek, ...)
  - ❖ Memorijski-mapirane datoteke
  - ❖ Ne mora biti mapiran na fizičku memoriju
  - ❖ Razlikuje se od jednog do drugog adresnog prostora

# Mapiranje fizičke / virtuelne memorije

0xFFFFFFFF

Kernel

0xC0000000

Process n

0x00000000

Physical Address Space

I/O Memory

RAM

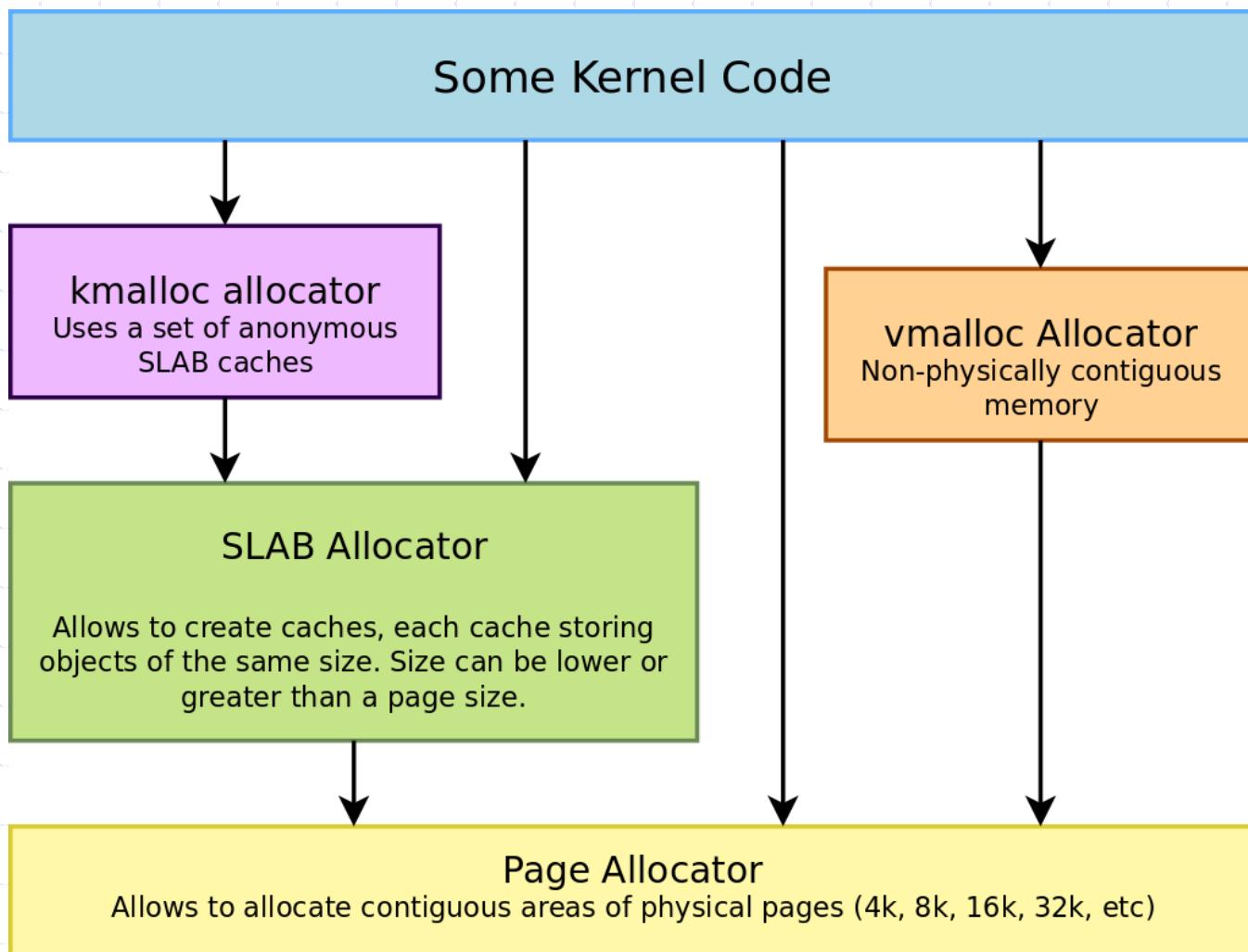
ZONE\_HIGHMEM

ZONE\_NORMAL

ZONE\_DMA



# Alokatori u kernelu





# kmalloc i kfree

- ❖ Osnovni alokatori, kernelovi ekvivalenti *glibc*-ovim malloc i free
- ❖ #include <linux/slab.h>
- ❖ static inline void \*kmalloc(size\_t size,  
int flags);
  - ❖ size: broj bajta za zauzimanje
  - ❖ flags: prioritet
- ❖ void kfree (const void \*objp);
- ❖ Primer: (drivers/infiniband/core/cache.c)

```
struct ib_update_work *work;
work = kmalloc(sizeof *work, GFP_ATOMIC);
...
kfree(work);
```



## Odlike kmalloc-a

- ❖ Brz (ukoliko nije blokiran čekajući memoriju da se oslobodi)
- ❖ Ne inicijalizuje zauzeti prostor
- ❖ Zauzeti prostor je susedni u RAM-u
- ❖ Zauzima po  $2^n$  veličine i koristi malo bajtova za upravljanje, tako da ako vam je potrebno 1000, ne zahtevajte 1024 jer ćete dobiti 2048
- ❖ Oprez: rokovaoci ne bi trebalo da zauzmu više od 128KB (gornja granica na nekim arhitekturama)
- ❖ Minimalno zauzimanje: 32 ili 64 bajta (zavisno od veličine stranice)



# Glavni flegovi (prioriteti) kmalloc-a (1/2)



- ❖ Definisani u include/linux/gfp.h (GFP:  
    \_\_get\_free\_pages)
- ❖ GFP\_KERNEL
  - ❖ Standardno zauzimanje kernel memorije, može da blokira i u većini slučajeva je dovoljno
- ❖ GFP\_ATOMIC
  - ❖ Zauzeti RAM iz koda kojem nije dozvoljeno da blokira (rukovaoci prekidima) ili koji ne želi da blokira (kritične sekcije), nikad ne blokira
- ❖ GFP\_USER
  - ❖ Zauzima memoriju za korisničke procese, može da blokira i ima najniži prioritet



# Glavni flegovi (prioriteti) kmalloc-a (2/2)



- ❖ Dodatni flegovi koji mogu biti dodati sa |
  - ❖ \_\_\_\_ GFP\_DMA ili GFP\_DMA
    - ❖ Zauzimanje u DMA zoni
  - ❖ \_\_\_\_ GFP\_ZERO
    - ❖ Vraća stranicu popunjenu nulama.
  - ❖ \_\_\_\_ GFP\_NOFAIL
    - ❖ Ne sme da ne uspe! Oprez: koristiti samo kad je neophodno!
  - ❖ \_\_\_\_ GFP\_NORETRY
    - ❖ Ako zazuimanje ne uspe, ne pokušava ponovo da dobije slobodne stranice.
- ❖ Primer:
  - ❖ GFP\_KERNEL | \_\_\_\_ GFP\_DMA
    - ❖ Napomena: skoro samo \_\_\_\_ GFP\_DMA ili GFP\_DMA se koriste u rukovaocima uređaja.



# Funkcije za zauzimanje

## ❖ Imena su slična funkcijama C biblioteke

- ❖ static inline void \*kzalloc(size\_t size,  
gfp\_t flags);
  - ❖ Popunjavam nulama zauzeti bafer.
- ❖ static inline void \*kcalloc(size\_t n, size\_t  
size, gfp\_t flags);
  - ❖ Zauzima memoriju za niz od n elemenata veličine size i  
postavlja vrednosti na nule.
- ❖ void \* \_\_must\_check krealloc(const void \*,  
size\_t, gfp\_t);
  - ❖ Menja veličinu datog bafera.



# Dostupni zauzimači

- ❖ Memorija je zauzeta korišćenjem slabs (grupa jedne ili više stranica iz kojih se objekti zauzimaju). Više kompatibilnih slab zauzimača je dostupno:
- ❖ *SLAB* – početni i dokazan zauzimač u Linuksu 2.6
- ❖ *SLOB* – mnogo jednostavniji, efikasan za čuvanje memorije na manjim sistemima (`CONFIG_EMBEDDED`)
- ❖ *SLUB* – novi podrazumevani zauzimač od 2.6.23, jednostavniji od *SLAB*, skalira mnogo bolje (za ogromne sisteme) i kreira manje fragmentacije

⊖ Choose SLAB allocator (NEW)

SLAB

SLUB (Unqueued Allocator) (NEW)

SLOB (Simple Allocator)

SLAB

SLUB

SLOB



# Slab keš i bazeni memorije

- ❖ Slab keševi: čine mogućim zazuzimanje više objekata iste veličine bez rasipanja RAM-a
- ❖ Do sada, najviše se koristi u podsistemima jezgra i ne previše u rukovaocima uređaja osim u USB i SCSI rukovaocima
- ❖ Pulovi (bazeni) memorije: pulovi već rezervisanih objekata služe da bi se povećala šansa da zauzimanje uspe, često se koristi sa keširanjem datoteka



# Zauzimanje po stranicama

- ❖ Prikladnije je kada su potrebni veliki delovi RAM-a
  - ❖ Stranica je najčešće 4K, ali može biti i veća na nekim arhitekturama (sh, mips: 4, 8, 16 ili 64K, ali ne i na i386 i arm)
  - ❖ `unsigned long get_zeroed_page(int flags);`
    - ❖ Vraća pokazivač na slobodnu stranicu i popunjava je nulama
  - ❖ `unsigned long __get_free_page(int flags);`
    - ❖ Isto, samo ne inicijalizuje sadržaj
  - ❖ `unsigned long __get_free_pages(int flags, unsigned int order);`
    - ❖ Vraća pokazivač na prostor od nekoliko susednih stranica u fizičkom RAM-u.  
`order: log2(<number_of_pages>)`
      - ❖ order se može izračunati od size pomoću `get_order()` funkcije.
    - ❖ Max: 8192 KB (`MAX_ORDER=11` u `include/linux/mmzone.h`) osim na nekim arhitekturama gde je prepisano sa `CONFIG_FORCE_MAX_ZONEORDER`



# Oslobađanje stranica

- ❖ void free\_page(unsigned long addr);
- ❖ void free\_pages(unsigned long addr, unsigned int order);
- ❖ Potrebno je koristiti isti redosled (order) kao i u zauzimanju



## vmalloc



- ❖ `vmalloc` može biti korišćen da se dobiju susedne memorijske zone u virtuelnom adresnom prostoru (čak iako stranice nisu susedne u fizičkoj memoriji)
  
- ❖ `void *vmalloc(unsigned long size);`
- ❖ `void vfree(void *addr);`



## „Alati“ memorije

- ❖ `void * memset(void * s, int c, size_t count);`
  - ❖ Puni region memorije sa datom vrednošću
- ❖ `void * memcpy(void * dest, const void *src, size_t count);`
  - ❖ Kopira deo memorije u drugu.
  - ❖ memmove se koristi za prostore koji se preklapaju
- ❖ Još mnogo funkcija ekvivalentnih standardnoj C biblioteci su definisane u `include/linux/string.h` kao i u `include/linux/kernel.h` (`sprintf`, itd.)



# Upravljanje kernel memorijom

- ❖ Debagovanje odliva je moguće od 2.6.31
- ❖ Kmemcheck
  - ❖ Dinamičko proveravanje pristupa neinicijalizovanoj memoriji. Pogledati dokumentaciju za više detalja na Documentation/kmemcheck.txt
- ❖ Kmemleak
  - ❖ Dinamičko proveravanje za curenje memorije
  - ❖ Pogledati u Documentation/kmemleak.txt za više informacija
- ❖ Oba ova alata imaju značajan overhead i treba ih koristiti samo pri razvoju



# Upravljanje memorijom - sažetak

- ❖ Malo zauzimanje
  - ❖ kmalloc, kzalloc (i kfree!)
  - ❖ Slab keševi i bazeni memorije
- ❖ Veliko zauzimanje
  - ❖ \_\_get\_free\_page[s], get\_zeroed\_page,  
free\_page[s]
  - ❖ vmalloc, vfree
- ❖ Alati slični libc
  - ❖ memset, memcpy, memmove...

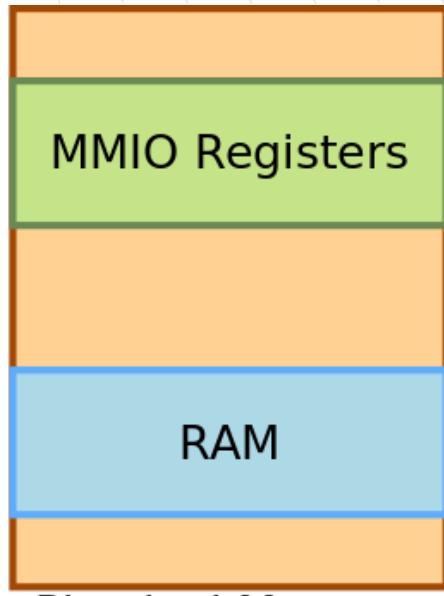


Ulazno izlazna memorija i prolazi

# RAZVOJ RUKOVALACA



# U/I prolazi naspram U/I memorije



Physical Memory  
address space, accessed with  
normal load/store instructions



Separate I/O address space,  
accessed with specific instructions



# Zahtevanje U/I prolaza



## /proc/ioports primer (x86)

```
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0100-013f : pcmcia_socket0
0170-0177 : ide1
01f0-01f7 : ide0
0376-0376 : ide1
0378-037a : parport0
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial
0800-087f : 0000:00:1f.0
0800-0803 : PM1a_EVT_BLK
0804-0805 : PM1a_CNT_BLK
0808-080b : PM_TMR
```

- ❖ struct resource \*request\_region(  
    unsigned long start, unsigned  
    long len, char \*name);
  - ❖ Pokušava da rezerviše dati region i vraća  
    NULL ako nije uspešno
- ❖ request\_region(0x0170, 8,  
    "ide1");
- ❖ void release\_region(unsigned  
    long start, unsigned long len);
- ❖ **Pogledati** include/linux/ioport.h i  
    kernel/resource.c za više informacija



# Čitanje/pisanje na U/I prolazima



- ❖ Implementacija sledećih funkcija i isti *unsigned* tip može da se razlikuje u zavisnosti od platforme
- ❖ Bajti
  - ❖ *unsigned inb(unsigned port);*
  - ❖ *void outb(unsigned char byte, unsigned port);*
- ❖ Reči
  - ❖ *unsigned inw(unsigned port);*
  - ❖ *void outw(unsigned char byte, unsigned port);*
- ❖ „long“ celobrojni tip
  - ❖ *unsigned inl(unsigned port);*
  - ❖ *void outl(unsigned char byte, unsigned port);*



# Čitanje/pisanje stringova na U/I prolazima



- ❖ Često je efikasnije nego odgovarajuća C petlja, ukoliko procesor podržava operacije
- ❖ Bajt stringovi
  - ❖ void insb(*unsigned port, void \*addr, unsigned long count*);
  - ❖ void outsb(*unsigned port, void \*addr, unsigned long count*);
- ❖ Reč stringovi
  - ❖ void insw(*unsigned port, void \*addr, unsigned long count*);
  - ❖ void outsw(*unsigned port, void \*addr, unsigned long count*);
- ❖ Long stringovi
  - ❖ void insl(*unsigned port, void \*addr, unsigned long count*);
  - ❖ void outsl(*unsigned port, void \*addr, unsigned long count*);



# Zahtevanje I/O memorije



/proc/iomem primer

00000000-0009efff : System RAM  
0009f000-0009ffff : reserved  
000a0000-000bffff : Video RAM area  
000c0000-000cffff : Video ROM  
000f0000-000fffff : System ROM  
00100000-3ffadfff : System RAM  
00100000-0030afff : Kernel code  
0030b000-003b4bff : Kernel data  
3ffae000-3fffffff : reserved  
40000000-400003ff : 0000:00:1f.1  
40001000-40001fff : 0000:02:01.0  
40001000-40001fff : yenta\_socket  
40002000-40002fff : 0000:02:01.1  
40002000-40002fff : yenta\_socket  
40400000-407fffff : PCI CardBus #03  
40800000-40bfffff : PCI CardBus #03  
40c00000-40ffffff : PCI CardBus #07  
41000000-413fffff : PCI CardBus #07  
a0000000-a0000fff : pcmcia\_socket0  
a0001000-a0001fff : pcmcia\_socket1  
e0000000-e7ffffff : 0000:00:00.0  
e8000000-efffffff : PCI Bus #01  
e8000000-efffffff : 0000:01:00.0

❖ Ekvivalentno funkcijama sa istom spregom

❖ struct resource \*  
request\_mem\_region(unsigned long start, unsigned  
long len, char \*name);

❖ void release\_mem\_region(  
unsigned long start,  
unsigned long len);



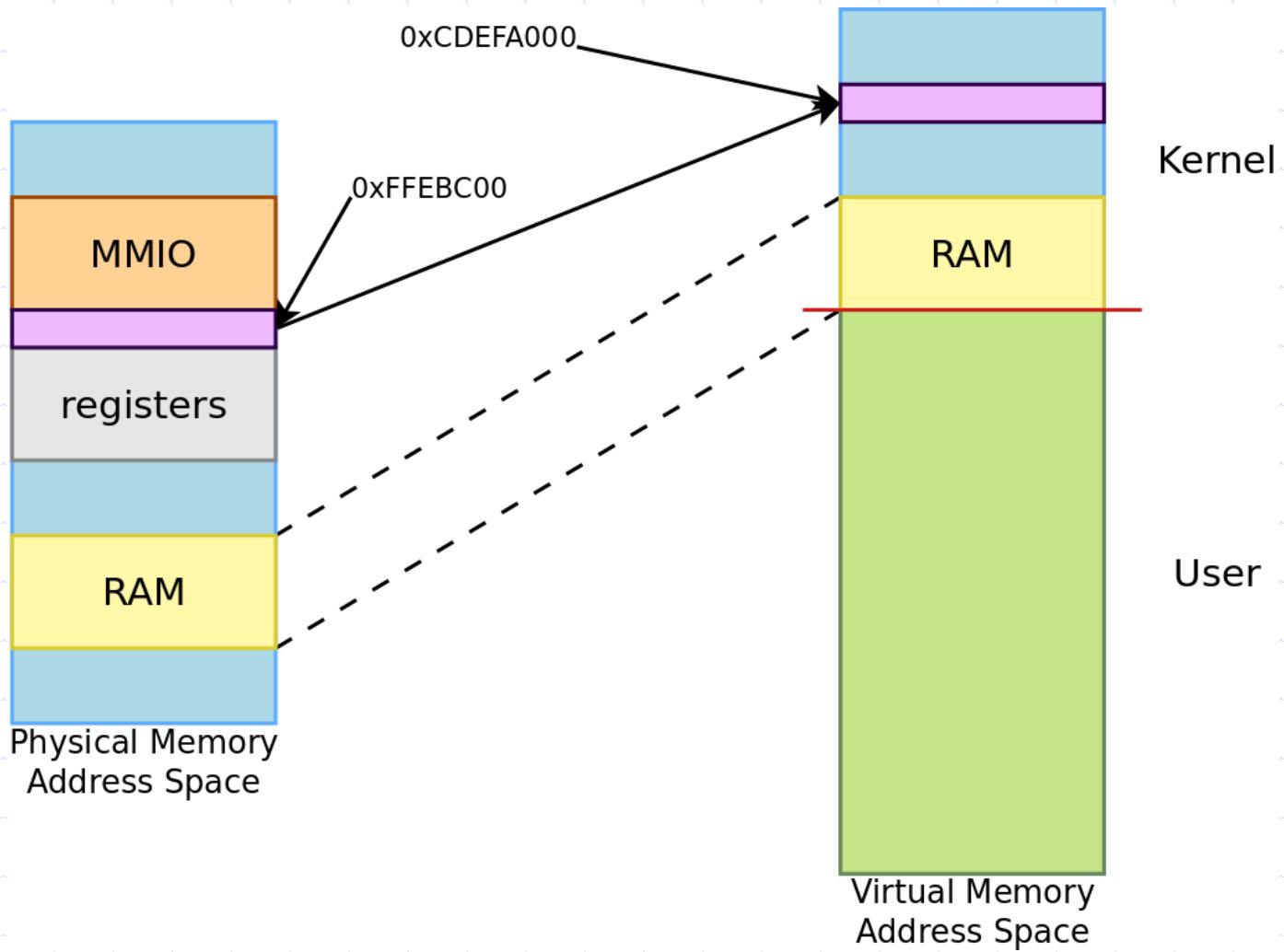
# Mapiranje U/I memorije u virtuelnu memoriju



- ❖ Load/store instrukcije rade sa virtuelnom memorijom
- ❖ Da bi pristupili I/O memoriji, rukovaoci moraju da imaju virtuelnu adresu koju procesor može da obradi
- ❖ Ovo se zadovoljava ioremap funkcijama
  - ❖ #include <asm/io.h>;
  - ❖ void \*ioremap(unsigned long phys\_addr,  
unsigned long size);
  - ❖ void iounmap(void \*address);
- ❖ Proveriti da ioremap ne vrati NULL adresu!!!



# ioremap()





# Razlike sa standardnom memorijom



- ❖ Čitanja i pisanja po memoriji mogu biti keširana
- ❖ Prevodilac može da odabere da piše vrednosti u procesorski register i može da nikada to ne upiše u glavnu memoriju
- ❖ Prevodilac može da odluči da optimizuje ili da preuredi funkcije za čitanje i pisanje



# Izbegavanje problema pristupa I/O



- ❖ Keširanje I/O prolaza ili memorije koji su već onesposobljeni od hardvera ili Linuks init koda
- ❖ Koristiti volatile iskaze u C kodu da se izbegne korišćenje registara umesto memorije pri prevodenju
- ❖ Memorijske barijere se podižu da bi se izbeglo menjanje redosleda
  - ❖ Nezavisno od fizičke arhitekture
    - ❖ #include <asm/kernel.h>
    - ❖ void barrier(void);
  - ❖ Zavisno od fizičke arhitekture
    - ❖ #include <asm/system.h>
    - ❖ void rmb(void);
    - ❖ void wmb(void);
    - ❖ void mb(void);



# Pristup I/O memoriji

- ❖ Direktno čitanje ili pisanje na adresu vraćenu sa ioremap-om može da ne radi na nekim arhitekturama
- ❖ Umesto toga, koristiti:
  - ❖ `unsigned int ioread8(void *addr);` (isto za 16 i 32)
  - ❖ `void iowrite8(u8 value, void *addr);` (isto za 16 i 32)
- ❖ Za čitanje ili pisanje niza vrednosti
  - ❖ `void ioread8_rep(void *addr, void *buf, unsigned long count);`
  - ❖ `void iowrite8_rep(void *addr, const void *buf, unsigned long count);`
- ❖ Ostale korisne funkcije
  - ❖ `void memset_io(void *addr, u8 value, unsigned int count);`
  - ❖ `void memcpy_fromio(void *dest, void *source, unsigned int count);`
  - ❖ `void memcpy_toio(void *dest, void *source, unsigned int count);`
- ❖ Mnogi rukovaoci i dalje koriste stare funkcije poput
  - ❖ `readb, readl, readw, writeb, writel, writew`



## /dev/mem

- ❖ Koristi se da obezbedi aplikacijama korisničkog prostora direktni pristup fizičkim adresama
- ❖ Upotreba: otvoriti /dev/mem i čitati ili pisati na dатој poziciji
  - ❖ Ono što se čita ili piše je vrednost odgovarajuće fizičke adrese
- ❖ Koristi se u aplikacijama poput X server-a da se direktno piše u memoriju uređaja



mmap

# RAZVOJ RUKOVALACA



# mmap (1/2)

- ❖ Mogućnost da se deo virtuelnog adresnog prostora mapira na sadržaj datoteke
- ❖ **cat /proc/1/maps (init proces)**

start	end	perm	offset	major:minor	inode	mapped file name
00771000-0077f000	r-xp	00000000	03:05	1165839		/lib/libselinux.so.1
0077f000-00781000	rw-p	0000d000	03:05	1165839		/lib/libselinux.so.1
0097d000-00992000	r-xp	00000000	03:05	1158767		/lib/ld-2.3.3.so
00992000-00993000	r--p	00014000	03:05	1158767		/lib/ld-2.3.3.so
00993000-00994000	rw-p	00015000	03:05	1158767		/lib/ld-2.3.3.so
00996000-00aac000	r-xp	00000000	03:05	1158770		/lib/tls/libc-2.3.3.so
00aac000-00aad000	r--p	00116000	03:05	1158770		/lib/tls/libc-2.3.3.so
00aad000-00ab0000	rw-p	00117000	03:05	1158770		/lib/tls/libc-2.3.3.so
00ab0000-00ab2000	rw-p	00ab0000	00:00	0		
08048000-08050000	r-xp	00000000	03:05	571452		/sbin/init (text)
08050000-08051000	rw-p	00008000	03:05	571452		/sbin/init (data, stack)
08b43000-08b64000	rw-p	08b43000	00:00	0		
f6fdf000-f6fe0000	rw-p	f6fdf000	00:00	0		
fefd4000-ff000000	rw-p	fefd4000	00:00	0		
ffffe000-fffff000	---p	00000000	00:00	0		



## mmap (2/2)

- ❖ Posebno korisno kada je datoteka uređaj
- ❖ pristup I/O memoriji uređaja i prolazima bez potrebe da se ide kroz **read**, **write** i **ioctl** funkcije
- ❖ primer sa **X serverom**

start	end	perm	offset	major:minor	inode	mapped file
-------	-----	------	--------	-------------	-------	-------------

<b>name</b>
-------------

<b>08047000-081be000 r-xp 00000000 03:05 310295</b>
---

<b>/usr/X11R6/bin/Xorg</b>
----------------------------

<b>081be000-081f0000 rw-p 00176000 03:05 310295</b>
---

<b>/usr/X11R6/bin/Xorg</b>
----------------------------

<b>...</b>
------------

<b>f4e08000-f4f09000 rw-s e0000000 03:05 655295</b>	<b>/dev/dri/card0</b>
---	-----------------------

<b>f4f09000-f4f0b000 rw-s 4281a000 03:05 655295</b>	<b>/dev/dri/card0</b>
---	-----------------------

<b>f4f0b000-f6f0b000 rw-s e8000000 03:05 652822</b>	<b>/dev/mem</b>
---	-----------------

<b>f6f0b000-f6f8b000 rw-s fcff0000 03:05 652822</b>	<b>/dev/mem</b>
---	-----------------

- ❖ Lakši način za dobijanje ovih informacija je **pmap <pid>**

# mmap - pregled

**mmap**  
sistemska  
poziv  
(jednom)

Proces

Rukovalac uređajem  
inicijalizuje  
mapiranje na  
osnovu **mmap**  
poziva

Pristup  
virtuelnoj  
adresi

MMU

Pristup  
fizičkoj  
adresi



# Kako implementirati mmap - korisnički prostor

- ❖ Otvoriti datoteku deskriptor uređaja
- ❖ Poziv **mmap** sistemskog poziva

```
void * mmap(
```

```
    void *start, // Željena startna adresa, često 0
    size_t length, // Dužina mapirane oblasti
    int prot, // Prava pristupa: čitanje, upis...
    int flags, // Opcije: deljeno mapiranje...
    int fd, // Deskriptor otvorene datoteke
    off_t offset // Pomeraj u datoteci
```

```
);
```

- ❖ Dobija se virtuelna adresa na kojoj mogu da se vrše čitanje i upis



# Kako implementirati mmap - kernelski prostor

## ❖ Rukovalac

- ❖ Implementirati mmap operaciju nad datotekom i dodati je operacijama rukovaoca

```
int (*mmap) (
    struct file *,           // pokazivač na otvorenu datoteku
    struct vm_area_struct * // kernelska struktura VM
);
```

## ❖ Inicijalizacija mapiranja

- ❖ U većini slučajeva može da se izvrši sa **remap\_pfn\_range()** koja obavlja većinu posla



## remap\_pfn\_range()

- ❖ **pfn** (*page frame number*)
  - ❖ Najznačajniji biti adrese stranice (bez bita koji odgovaraju veličini stranice)
- ❖ **#include <linux/mm.h>**

```
int remap_pfn_range(  
    struct vm_area_struct *,  
    unsigned long virt_addr, // Početna virtuelna adresa  
    unsigned long pfn,       // pfn početne fizičke adrese  
    unsigned long size,      // Veličina mapiranja  
    pgprot_t                // Prava pristupa stranice  
);
```



# Jednostavna implementacija mmap

```
❖ static int acme_mmap (
    struct file * file, struct vm_area_struct * vma)
{
    size = vma->vm_start - vma->vm_end;
    if (size > ACME_SIZE)
        return -EINVAL;

    if (remap_pfn_range(vma,
                        vma->vm_start,
                        ACME_PHYS >> PAGE_SHIFT,
                        size,
                        vma->vm_page_prot))
        return -EAGAIN;

    return 0;
}
```



## devmem2



- ❖ Koristan alat za direktni pristup fizičkoj memoriji iz šela radi čitanja ili pisanja
- ❖ Korisno za rane faze testiranja
  - ❖ nema potrebe da se piše ceo rukovalac da bi se deo testirao
- ❖ Koristi **mmap** nad uređajem **/dev/mem**
  
- ❖ Primeri (**b**: bajt, **h**: polu reč, **w**: reč)
  - ❖ **devmem2 0x000c0004 h** (čitanje)
  - ❖ **devmem2 0x000c0008 w 0xffffffff** (pisanje)



## mmap - sumirano

- ❖ Učitava se rukovalac uređajem
  - ❖ Definiše **mmap** operaciju nad datotekom
- ❖ Korisnički program poziva **mmap** sistemski poziv
- ❖ **mmap** operacija nad datotekom inicijalizuje mapiranje koristeći fizičku adresu uređaja
- ❖ Proces dobija početnu virtuelnu adresu sa koje može da čita ili da na nju piše (u zavisnosti od prava pristupa)
- ❖ **MMU** automatski vrši konverzije adresa između virtuelnog i fizičkog adresnog prostora
- ❖ Direktan pristup hardveru
  - ❖ Bez skupih **read** ili **write** sistemskih poziva



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Razvoj rukovalaca  
Drugi deo



# Sadržaj



- ❖ Razvoj rukovalaca
  - ❖ DMA
  - ❖ Blok rukovaoci



DMA

# RAZVOJ RUKOVALACA



# DMA memorjska ograničenja

- ❖ Mora da koristi kontinualnu fizičku memoriju
- ❖ Može da koristi memoriju alociranu sa **kmalloc** (do 128 KB) ili **\_\_get\_free\_pages** (do 8MB)
- ❖ Može da koristi blokovski I/O i mrežne bafere dizajnirane da podrže **DMA**
- ❖ Ne može da koristi memoriju alociranu sa **vmalloc**



# Rezervisanje memorije za DMA

- ❖ Osiguranje da će uvek biti dovoljno **RAM** memorije za **DMA** prenose
  - ❖ Primer: raspoloživo **32 MB RAM** memorije, **2 MB** rezervisana za **DMA**
    - ❖ Startovati kernel sa **mem=30**
    - ❖ Kernel će koristiti samo prvih **30 MB RAM** memorije
    - ❖ Rukovalac može da uzme pod kontrolu preostala **2 MB**
- ❖ **dmabuf = ioremap (**
- 0x1e00000, // Start: 30 MB**
- 0x200000 // Veličina: 2 MB**
- );**



# Problemi sa sinhronizacijom

- ❖ Keširanje memorije može da utiče na **DMA**
- ❖ Pre prenosa od **DMA** ka uređaju
  - ❖ Mora se obezbediti da se svi upisi u **DMA** bafer završe pre nego što se prebace do uređaja
- ❖ Posle prenosa od uređaja do **DMA**
  - ❖ Pre nego što rukovaoci započnu čitanje iz **DMA** bafera mora se obezbediti da se svi keširani upisi zaista upišu
- ❖ Bidirekcionni **DMA**
  - ❖ Svi upisi se moraju završiti pre prenosa od i ka **DMA**



# Linuksov DMA API

- ❖ DMA prisutan u kernelu može da:
  - ❖ alocira koherentnu fizičku memoriju
  - ❖ obezbedi pravovremeno upisivanje keširane memorije
  - ❖ kontroliše DMA mapiranje i IOMMU
- ❖ Dokumentacija
  - ❖ [Documentation/DMA-API.txt](#)
- ❖ Većina podsistema (**PCI, USB...**) koriste sopstveni **DMA API** koji je izведен iz generičkog



# Ograničavanje DMA adresnog prostora

- ❖ Kernel podrazumeva da uređaj može da koristi **DMA** za pristup bilo kojoj 32-bitnoj adresi
  - ❖ Ne mora da bude tačno za sve uređaje
- ❖ Kernelu se može reći da radi samo sa manjim adresama (npr. 24-bitnim)

```
if (dma_set_mask (dev,      // struktura uređaja  
                  0x00fffff // 24 bita  
                ))  
    use_dma = 1;          // DMA može da se  
                          // koristi  
  
else  
    use_dma = 0;          // DMA ne može da se koristi
```



# Koherentno ili striming DMA mapiranje



## ❖ Koherentno mapiranje

- ❖ Kernel alocira odgovarajući bafer i vrši mapiranje za rukovaoca
- ❖ Omogućen istovremeni pristup od strane procesora i uređaja
- ❖ Mora da bude u koherentnoj keširanoj memoriji
- ❖ Uglavnom alocirano sve vreme dok je modul učitan
- ❖ Može biti skupo na nekim platformama

## ❖ Striming mapiranje

- ❖ Kernel vrši mapiranje za bafer koji obezbeđuje rukovalac
- ❖ Bafer je već alociran od strane rukovaoca
- ❖ Mapiranje se postavlja za svaki prenos - DMA registri su slobodni
- ❖ Moguće neke optimizacije
- ❖ Preporučeno rešenje



# Alokacija koherentnog mapiranja

- ❖ Kernel je zadužen za alokaciju bafera i mapiranje
- ❖ **#include <asm/dma-mapping.h>**

```
void *  
dma_alloc_coherent (  
    struct device *dev, // Struktura uređaja  
    size_t size, // Veličina bafera u bajtima  
    dma_addr_t *handle, // Izlazni par: DMA adresa  
                        // magistrale  
    gfp_t gfp // Standardne GFP nazanake
```

```
);
```

- ❖ **void dma\_free\_coherent ( struct device \*dev,**  
**size\_t size, void \*cpu\_addr, dma\_addr\_t**  
**handle );**



# DMA rezervoari (1/2)

- ❖ `dma_alloc_coherent` obično alocira bafera pomoću `_get_free_pages` (minimum 1 stranicu)
- ❖ Za manje koherentne alokacije se mogu koristiti DMA rezervoari
  - ❖ `#include <linux/dmapool.h>`
- ❖ Kreiranje DMA rezervoara
  - ❖ `struct dma_pool *`  
`dma_pool_create (`  
    `const char *name,` // Ime  
    `struct device *dev,` // Struktura uređaja  
    `size_t size,` // Veličina bafera u rezervoaru  
    `size_t align,` // Hardversko poravnanje (u bajtima)  
    `size_t allocation` // Adresa koja se ne sme prekoraciti  
`);`



## DMA rezervoari (2/2)

- ❖ Alociranje iz rezervoara

- ❖ **void \* dma\_pool\_alloc ( struct dma\_pool \*pool, gfp\_t mem\_flags, dma\_addr\_t \*handle );**

- ❖ Oslobađanje bafera iz rezervoara

- ❖ **void dma\_pool\_free ( struct dma\_pool \*pool, void \*vaaddr, dma\_addr\_t dma);**

- ❖ Uništavanje rezervoara (prvo treba oslobiti sve bafere)

- ❖ **void dma\_pool\_destroy ( struct dma\_pool \*pool);**



# Postavljanje streaming mapiranja

- ❖ Radi sa baferima koji su već alocirani od strane rukovaoca
- ❖ `#include <linux/dmapool.h>`

```
dma_addr_t dma_map_single (
    struct device *, // Struktura uređaja
    void *,          // ulazni par: bafer
    size_t,          // veličina bafera
    enum dma_data_direction // DMA_BIDIRECTIONAL,
                           // DMA_TO_DEVICE ili
                           // DMA_FROM_DEVICE
);
```

- ❖ vodi `dma_unmap_single ( struct device *dev, dma_addr_t handle,`  
`size_t size, enum dma_data_direction dir );`



# DMA striming mapiranje

- ❖ Kada je mapiranje aktivno samo uređaj bi trebao da pristupa baferu
- ❖ Procesor sme da pristupi baferu posle poziva **unmap** - koristiti zaključavanje
- ❖ Ukoliko je potrebno API može da obezbedi dodatni bafer između (Ukoliko ulazni bafer ne može da se koristi za **DMA**)
- ❖ Linuks API podržava i raspi i sakupi **DMA striming** mapiranje



# DMA sumarno

- ❖ Većina rukovalaca koristi specifičan API od njihovog podsistema, dok ostali koriste generički Linuksov API

## ❖ Koherentno mapiranje

- ❖ Kernel alocira **DMA** bafer
- ❖ Mapiran tokom celog života modula
- ❖ Skup, ne preporučuje se
- ❖ Procesor i uređaj mogu da pristupaju baferu istovremeno
- ❖ Glavne funkcije
  - ❖ **dma\_alloc\_coherent**
  - ❖ **dma\_free\_coherent**

## ❖ Striming mapiranje

- ❖ Rukovalac alocira **DMA** bafer
- ❖ Mapira se za svaki prenos posebno
- ❖ Jeftinije, **DMA** registri slobodni
- ❖ Samo uređaj sme da pristupa baferu kad je mapiranje aktivno
- ❖ Glavne funkcije
  - ❖ **dma\_map\_single**
  - ❖ **dma\_unmap\_single**



Rukovaoci blokovskim uređajima

# RAZVOJ RUKOVALACA



# Blokovski uređaji

- ❖ Pored znakovnih i mrežnih uređaja blokovski uređaji su još jedan važan tip uređaja u svakom sistemu
- ❖ Koriste se za skladištenje aplikacijskog koda, podataka i korisničkih podataka i često imaju važan uticaj na celokupno ponašanje sistema
- ❖ Poseban podsistem, blokovski sloj je zadužen za kontrolu blokovskih uređaja



# Iz korisničkog prostora

- ❖ Iz korisničkog prostora blok uređajima se pristupa preko datoteka koje se uglavnom čuvaju u /dev direktorijumu
  - ❖ Kreiraju se ručno ili automatski preko udev-a
- ❖ U većini slučajeva ove datoteke sadrže sisteme datoteka
  - ❖ Ne pristupa im se direktno, već se mauntuju
- ❖ Blok uređaji su vidljivi i kroz sysfs sistem datoteka, u /sys/block/ direktorijumu

```
$ ls /sys/block/  
dm-0 dm-2 loop0 loop2 ram0 ram2 ram4 ram6 sda  
dm-1 dm-3 loop1 loop3 ram1 ram3 ram5 ram7
```



# Arhitektura (1/2)

Korisnička aplikacija

Korisnički prostor

Kernelski prostor

Korisnička aplikacija

Virtuelni sistem datoteka

Pojedinačni sistemi  
datoteka (ext3, vfat...)

Keširanje bafera i stranica

Blokovski sloj

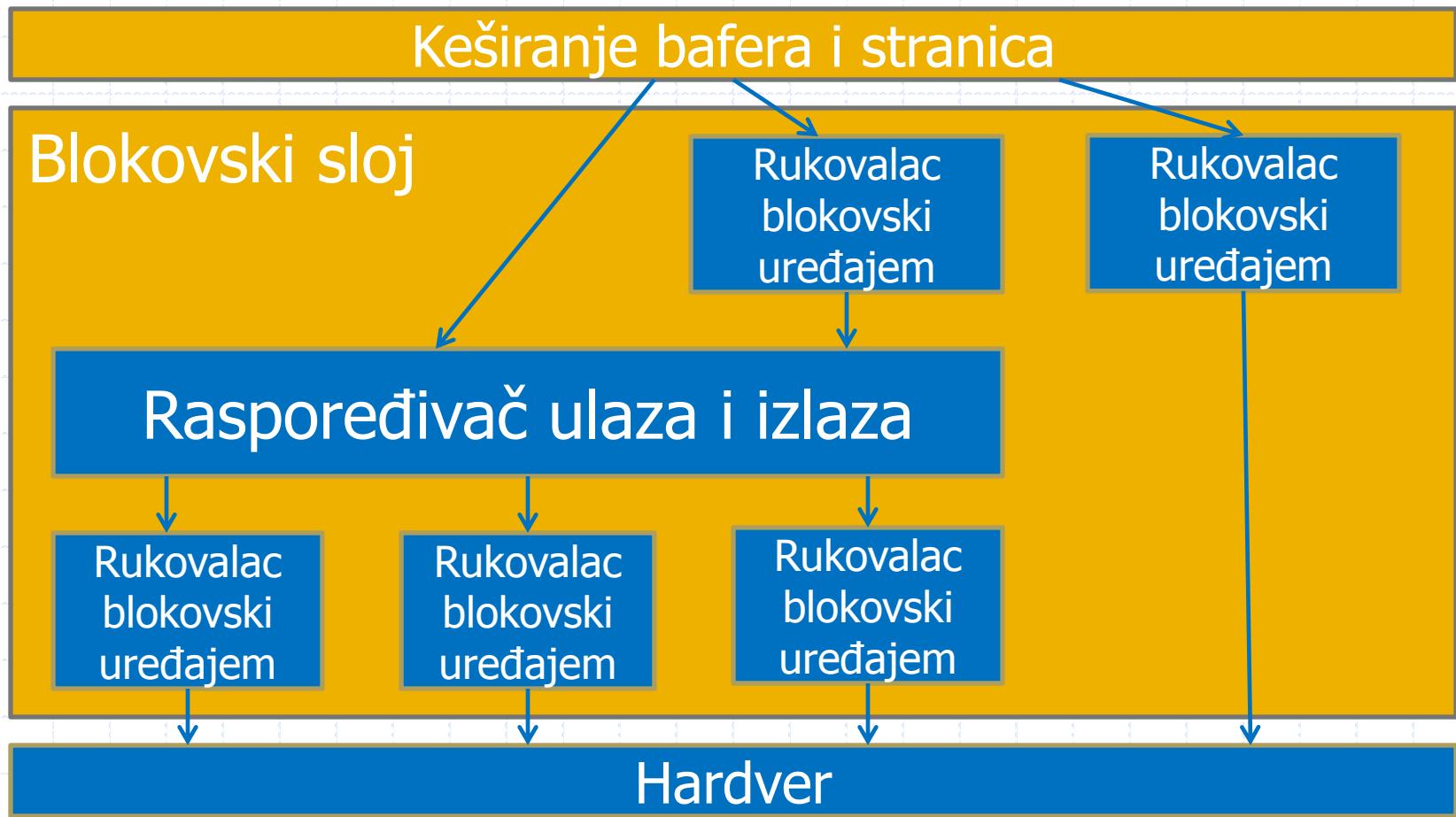


## Arhitektura (2/2)

- ❖ Korisnička aplikacija može da koristi blok uređaj
  - ❖ Kroz sistem datoteka, čitanjem, pisanjem ili mapiranjem datoteka
  - ❖ Direktno čitanjem, pisanjem ili mapiranjem datoteke koja predstavlja blok uređaj u /dev
- ❖ U oba slučaja ulazna tačka u krenelu je virtuelni sistem datoteka
  - ❖ Rukovalac sistemom datoteka je uključen ukoliko se pristupa normalnoj datoteci
- ❖ Sloj za keširanje bafera i stranica čuva nedavno čitane i pisane delove blokovskih uređaja
  - ❖ Ova komponenta je kritična za performanse sistema



# Blokovski sloj (1/2)





## Blokovski sloj (2/2)

- ❖ Blokovski sloj omogućava rukovaocima blokovskim uređajima da primaju zahteve za ulaz i izlaz i zadužen je za raspoređivanje ulaza i izlaza
- ❖ Raspoređivanje ulaza i izlaza omogućava
  - ❖ Spajanje zahteva kako bi se dobila veća veličina
  - ❖ Promena redosleda zahteva kako bi se optimizovalo kretanje glave diska
- ❖ U Linuksu postoji više raspoređivača ulaza i izlaza sa različitim algoritmima
- ❖ Rukovaoc blokovskim uređajem može da obradi zahtev pre ili posle toga što on prođe kroz raspoređivač



## Dva glavna tipa rukovalaca

- ❖ Većina rukovalaca blokovskim uređajima se implementira ispod rasporedišvača kako bi se iskoristile prednosti raspoređivanja ulaza i izlaza
  - ❖ Rukovaoci hard diskom, rukovaoci CD-ROM-om...
- ❖ Za neke rukovaće nema smisla da se koristi rasporedišvač ulaza i izlaza
  - ❖ RAID i rukovaoc glasnoćom
  - ❖ Specijalni loop rukovalac
  - ❖ Blokovksi uređaji bazirani na memoriji



# Raspoloživi raspoređivači ulaza i izlaza

- ❖ Četiri različita raspoređivača
  - ❖ NOOP za blokovske uređaje koji nisu bazirani na disku
  - ❖ Predviđajući - pokušavaju da predvide gde će se desiti naredni pristup
  - ❖ Sa krajnjim rokom - pokušavaju da garantuju da će se izlaz/ulaz desiti u predviđenom roku
  - ❖ CFQ (*Complete Fairness Queueing*) - podrazumevani raspoređivač, pokušava da garantuje pravedno ponašanje prema svim korisnicima
- ❖ Trenutno korišćeni raspoređivač se može pronaći ili podesiti na putanji /sys/block/<dev>/queue/scheduler



# Opcije u kernelu

## ❖ CONFIG\_BLOCK

- ❖ Omogućava selektivno uključivanje ili isključivanje blokovskog sloja
- ❖ Kernel bez blokovskog sloja je koristan ukoliko se koriste MTD uređaji, skladištenje podataka preko mreže ili initramfs
- ❖ Opcija je dostupna jedino ako je uključen CONFIG\_EMBEDDED

## ❖ CONFIG\_IOSCHED\_NOOP, CONFIG\_IOSCHED\_AS, CONFIG\_IOSCHED\_DEADLINE, CONFIG\_IOSCHED\_CFG

- ❖ Omogućuju uključivanje ili isključivanje različitih raspoređivača



# Pogled u kod

- ❖ Blokovski sloj je implementiran u block/ direktorijumu
  - ❖ Kod rasporedjivača se nalazi u istom direktorijumu u \*-iosched.c datotekama
- ❖ Nekoliko jednostavnih blokovskih uređaja su implementirani u drivers/block/
  - ❖ loop.c - rukovalac koji omogućava da se obične datoteke vide kao blokovski uređaji
  - ❖ brd.c - rukovalac ramdiskom
  - ❖ nbd.c - rukovalac blokovskim uređajima zasnovanim na mreži

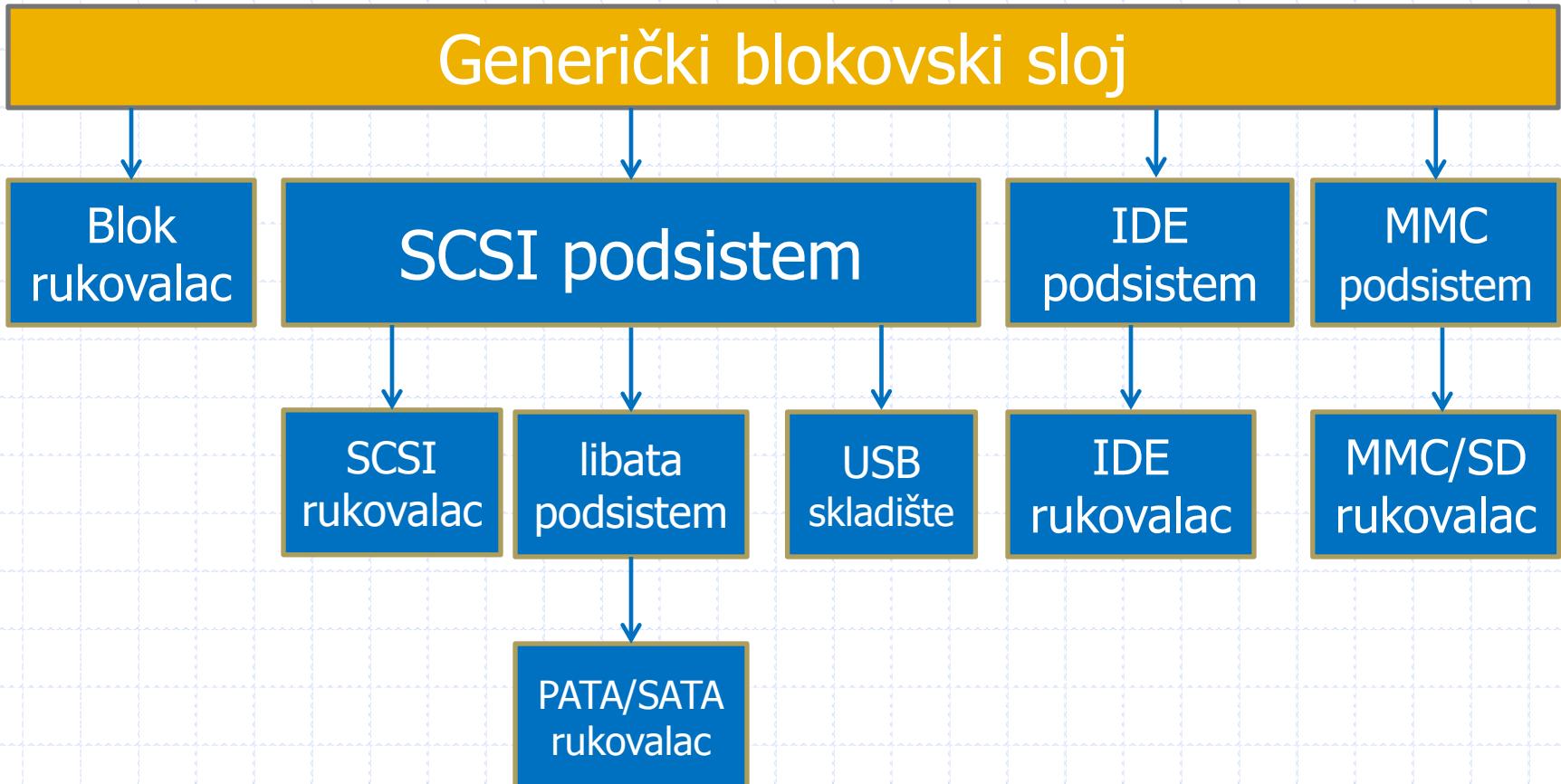


# Implementacija rukovaoca blokovskim uređajima (1/2)



- ❖ Rukovalac blokovskim uređajima mora da implementira set operacija koje će biti registrovane u blokovskom sloju i koje će primati zahteve od kernela
- ❖ Ove operacije mogu biti implementirane direktno, ali postoje i podsistemi koji objedinjuju uređaje istog tipa
  - ❖ SCSI uređaji
  - ❖ PATA/SATA uređaji
  - ❖ MMC/SD uređaji
  - ❖ ...

# Implementacija rukovaoca blokovskim uređajima (2/2)





# Registracija major broja

- ❖ Prvi korak u inicijalizaciji rukovaoca blokovskim uređajem je registracija major broja
  - ❖ `int register_blkdev (unsigned int major,  
const char *name);`
  - ❖ Može se dodeliti 0 - u tom slučaju se broj dinamički alocira
  - ❖ Registrovani uređaji se mogu videti u `/proc/devices`
- ❖ Pri gašenju rukovaoca broj se mora odregistrovati
  - ❖ `void unregister_blkdev (unsigned int major, const char  
*name);`
  - ❖ Ove funkcije se nalaze u `<linux/fs.h>`



# gendisk struktura

- ❖ Struktura koja predstavlja jedan blok uređaj, definisan je u <linux/genhd.h>
  - ❖ int major - major broj rukovaoca uređajem
  - ❖ int first\_minor - minor broj ovog uređaja (ukoliko uređaj može da se particioniše može imati više minor brojeva)
  - ❖ int minors - broj minor brojeva
  - ❖ struct block\_device\_operations \*fops - pokazivač na listu operacija blokovskog uređaja
  - ❖ struct request\_queue \*queue - red za zahtevima
  - ❖ sector\_t capacity - veličina uređaja u sektorima



# Inicijalizacija diska (1/3)

- ❖ Alokacija gendisk strukture
  - ❖ struct gendisk \*alloc\_disk(int minors)
    - ❖ minors - broj minora koji treba da bude alociran za ovaj uređaj (1 ukoliko uređaj ne može da se particioniše)
- ❖ Alokacija reda sa zahtevima
  - ❖ struct request\_queue \*blk\_init\_queue (request\_fn\_proc \*rfn, spinlock\_t \*lock);
    - ❖ rfn - funkcija za obradu zahteva
    - ❖ lock - opcioni spinlok za zaštitu reda od konkurentnog pristupa (ukoliko se prosledi NULL, koristi se podrazumevani spinlok)



# Inicijalizacija diska (2/3)

- ❖ Inicijalizacija gendisk strukture
  - ❖ Polja major, first\_minor, fops, disk\_name i queue moraju da budu inicijalizovana
  - ❖ Polje private\_data može da se koristi za čuvanje nekih podataka vezanih za disk
- ❖ Određivanje kapacitete
  - ❖ void set\_capacity (struct gendisk \*disk, sector\_t size)
    - ❖ size je broj 512-bitnih sektora
    - ❖ sector\_t je širok 64 bita na 64-bitnim arhitekturama, a 32 bita na 32-bitnim ukoliko CONFIG\_LBD (*large block device*) nije uključen



## Inicijalizacija diska (3/3)

- ❖ Dodavanje diska u sistem

- ❖ void add\_disk (struct gendisk \*disk)

- ❖ Posle ovog koraka sisem može da pristupa blok uređaju, pa rukovalac mora biti u potpunosti spremam da obrađuje ulaz/izlaz pre nego što se pozove add\_disk



# Odregistracija diska

- ❖ Odregistracija diska
  - ❖ void del\_gendisk (struct gendisk \*gp)
- ❖ Oslobođanje red za zahteve
  - ❖ void blk\_cleanup\_queue (struct request\_queue \*)
- ❖ Oslobođanje reference zauzete sa alloc\_disk()
  - ❖ void put\_disk (struct gendisk \*disk)



# block\_device\_operations

- ❖ Set pokazivača na funkcije
  - ❖ open() i release() - pozivaju se kada se uređaj kojim rukuje rukovalac otvara i zatvara
  - ❖ ioctl() - specifične operacije rukovaoca
  - ❖ direct\_access() - potrebna za XIP podršku
  - ❖ media\_changed() i revalidate() - potrebno za podršku uređaja koji se mogu ukloniti
  - ❖ getgeo() - obezbeđuje geometrijske informacije korisničkom prostoru



# Jednostavna request() funkcija (1/2)



```
static void foo_request(request_queue_t *q)
{
    struct request *req;
    while ((req = elv_next_request(q)) != NULL) {
        if (! blk_fs_request(req)) {
            __blk_end_request(req, 1,
                              req->nr_sectors << 9);
            continue;
        }
        /* Do the transfer here */
        __blk_end_request(req, 0, req->nr_sectors << 9);
    }
}
```

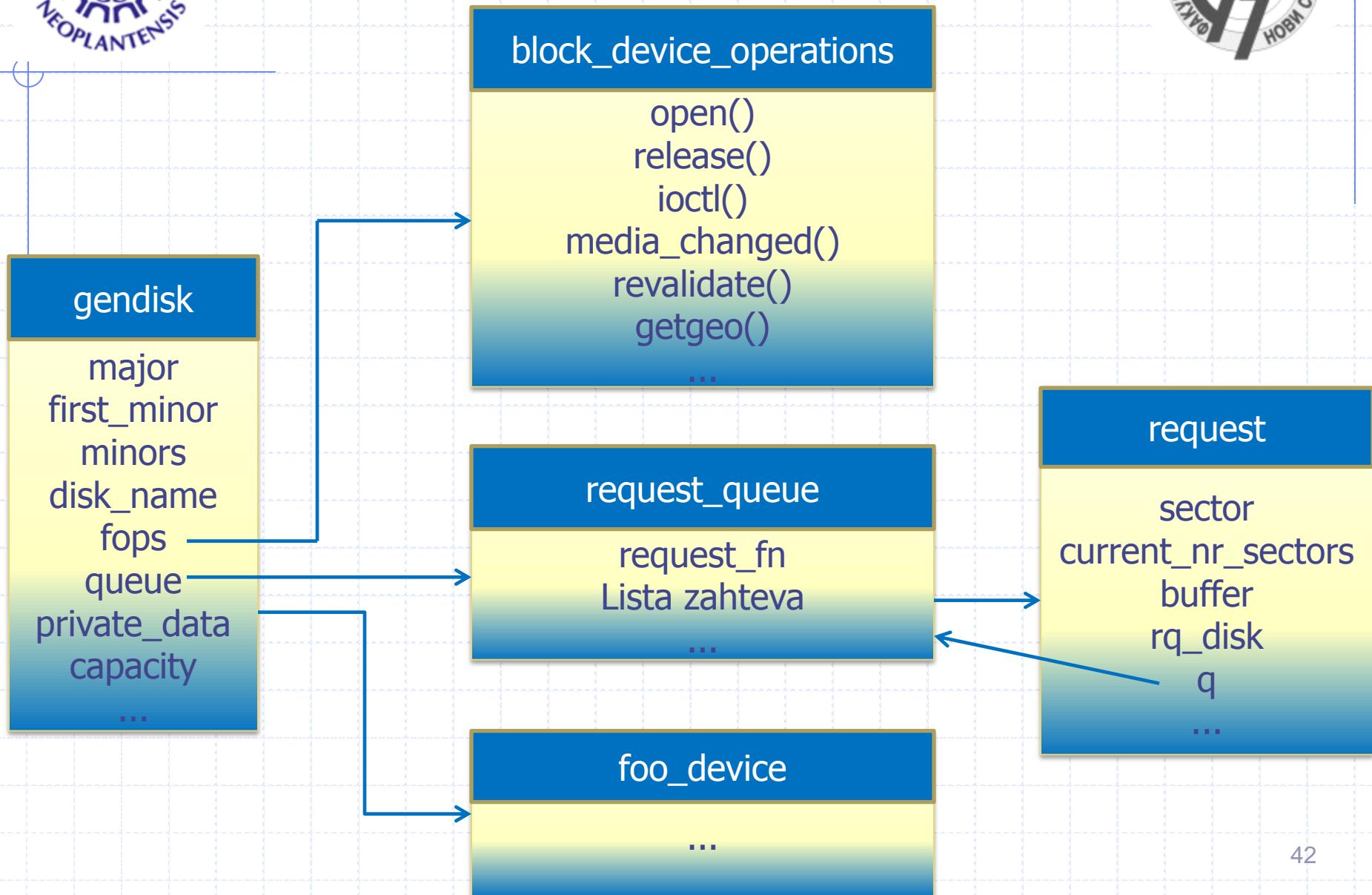


# Jednostavna request() funkcija (2/2)



- ❖ Informacije o transferu su dostupne u strukturi request
  - ❖ sector - pozicija u uređaju na kojoj treba da se izvrši transfer
  - ❖ current\_nr\_sectors - broj sektora za transfer
  - ❖ buffer - lokacija u memoriji odakle podaci treba da se pročitaju ili da se upišu
  - ❖ req\_data\_dir - tip transfera, može biti READ ili WRITE
- ❖ \_\_blk\_end\_request() ili blk\_end\_request() se koristi da se objavi završetak obrade zahteva
  - ❖ red mora da se zaključa pre korišćenja funkcije

# Strukture podataka





# Konfiguracija reda sa zahtevima (1/3)



- ❖ blk\_queue\_bounce\_limit (queue, u64 dma\_addr)
  - ❖ Saopštava kernelu najveću fizičku adresu koju uređaj može da obradi
  - ❖ Iznad te adrese dolazi do preslikavanja (*bouncing*)
- ❖ BLK\_BOUNCE\_HIGH
  - ❖ preslikavanje se uključuje ako se stranice nalaze u „visokoj“ memoriji
- ❖ BLK\_BOUNCE\_ISA
  - ❖ preslikavanje se uključuje ako stranice nisu u ISA 16 Mb zoni
- ❖ BLK\_BOUNCE\_ANY
  - ❖ preslikavanje se nikad ne uključuje



# Konfiguracija reda sa zahtevima (2/3)



- ❖ `blk_queue_max_sectors()`
  - ❖ Saopštava kernelu maksimalni broj 512-bitnih sektora za svaki zahtev
- ❖ `blk_queue_max_phys_segments()`  
`blk_queue_max_hw_segments()`
  - ❖ Saopštava kernelu maksimalni broj nepovezanih segmenta koje može da obradi u jednom zahtevu
- ❖ `blk_queue_max_segment_size()`
  - ❖ Saopštava kernelu maksimalnu veličinu jednog segmenta



# Konfiguracija reda sa zahtevima (3/3)



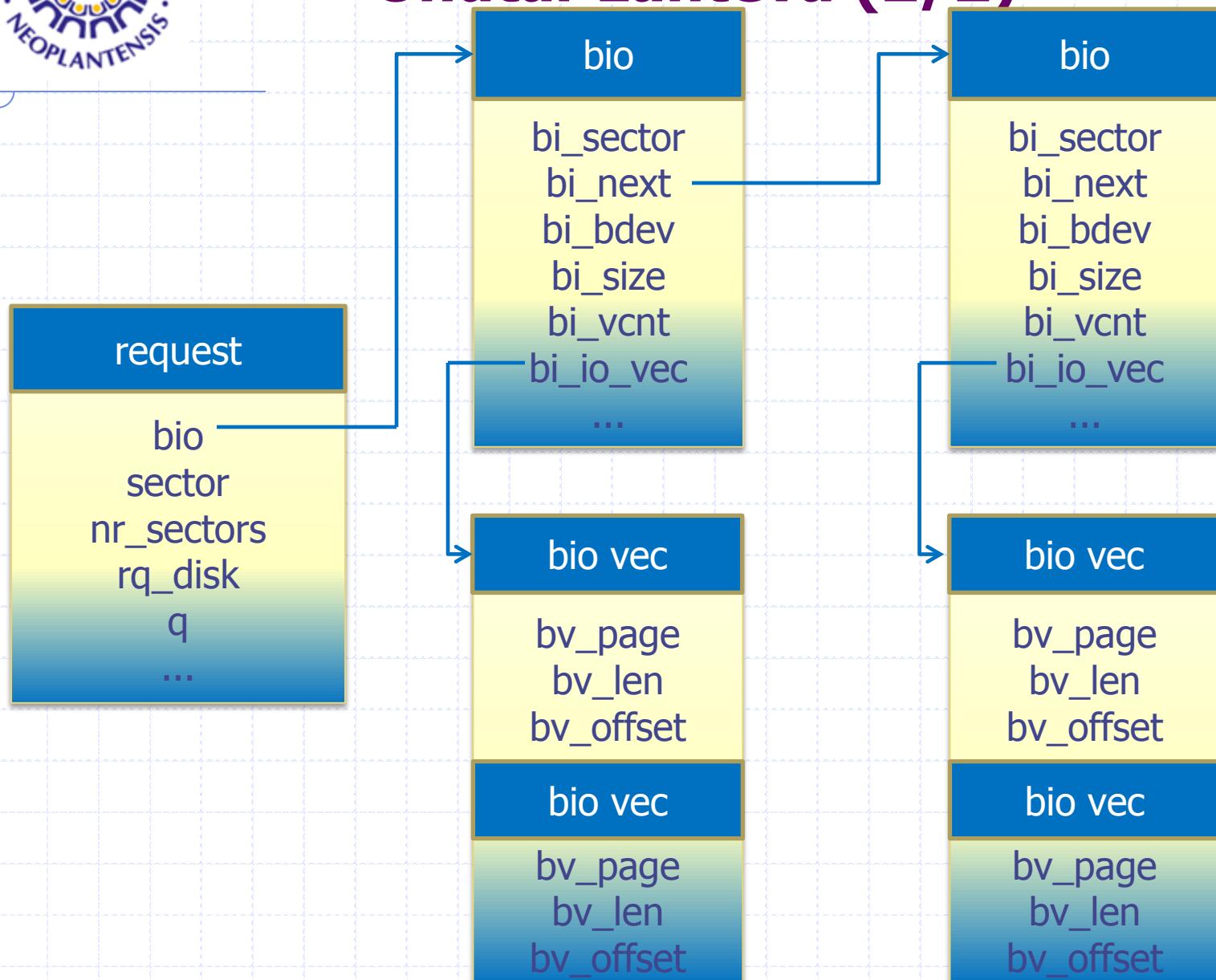
- ❖ blk\_queue\_segment\_boundary (queue, unsigned long mask)
  - ❖ Saopštava kernelu koja su memorijska ograničenja uređaja
- ❖ blk\_queue\_dma\_alignment (queue, int mask)
  - ❖ Saopštava kernelu ograničenja vezana za poravnanje memorije
- ❖ blk\_queue\_hardsect\_size (queue, unsigned short max)
  - ❖ Saopštava kernelu veličinu sektora uređaja
  - ❖ Zahtevi će biti poravnati
  - ❖ Komunikacija i dalje ostaje u broju 512-bitnih sektora



## Unutar zahteva (1/2)

- ❖ Zahtev je sastavljen od nekoliko segmenata koji su kontinualni na blok uređaju, ali ne moraju biti kontinualni i u fizičkoj memoriji
- ❖ Struktura request je u stvari lista struktura bio
- ❖ bio je deskriptor i/O zahteva poslatog blokovskom sloju
  - ❖ Više bio struktura se spaja u strukturu request od strane rasporedioca
- ❖ Pošto bio može da predstavlja nekoliko stranica podataka sasavljen je od nekoliko struktura bio\_vec od kojih svaka predstavlja jednu stranicu u memoriji

## Unutar zahteva (2/2)





## Asinhronе operacije (1/2)

- ❖ Ukoliko se zahtevi obrađuju asinhrono potrebno je obrađivati više zahteva istovremeno
  - ❖ zahtevi se moraju izvlačiti iz reda
  - ❖ `void blkdev_dequeue_request (struct request *req)`
- ❖ Ukoliko je potrebno zahtev se može vratiti u red
  - ❖ `void elv_requeue_request (request_queue_t *queue, struct request * req)`



## Asinhronne operacije (2/2)

- ❖ Kada je zahtev izvađen iz reda rukovalac je dužan da obradi sve segmente zahteva
  - ❖ Prolaskom u petlji dok blk\_end\_request() ne vrati 0
  - ❖ korišćenjem rq\_for\_each\_segment() makroa

```
struct bio_vec *bvec;
struct req_iterator iter;
rq_for_each_segment(bvec, rq, iter)
{
    /* rq->sector contains the current sector
       page_address(bvec->bv_page) + bvec->bv_offset points to the data
       bvec->bv_len is the length */

    rq->sector += bvec->bv_len / KERNEL_SECTOR_SIZE;
}

blk_end_request(rq, 0, rq->nr_sectors << 9);
```



## DMA (1/3)

- ❖ Blokovski sloj sadrži pomoćnu funkciju za pretvaranje zahteva u raspi-sakupi listu
  - ❖ int blk\_rq\_map\_sg (struct request\_queue \*q, struct request \*rq, struct scatterlist \*sglist)
- ❖ sglist mora da bude pokazivač na niz struktura scatterlist sa dovoljno članova da se drži maksimalni broj segmenata u zahtevu
  - ❖ ovaj broj je zadat funkcijom blk\_queue\_max\_hw\_segments()
- ❖ Funkcija vraća stvarni broj scatterlist struktura



## DMA (2/3)

- ❖ Kada je lista generisana, individualni segmenti se moraju mapirati na adrese koje odgovaraju DMA
  - ❖ `int dma_map_sg (struct device *dev,  
                  struct scatterlist *sglist,  
                  int count,  
                  enum dma_data_direction dir)`
- ❖ dev je uređaj na kom će biti izvršen DMA transfer
- ❖ dir je pravac transfera (DMA\_TO\_DEVICE,  
DMA\_FROM\_DEVICE, DMA\_BIDIRECTIONAL)
- ❖ Adrese i dužine svih segmenata se mogu naći pomoću `sg_dma_addr()` i `sg_dma_len()` funkcija



## DMA (3/3)

- ❖ Posle izvršenog transfera segmenti se odmapiraju sa
  - ❖ int dma\_unmap\_sg (struct device \*dev,  
                      struct scatterlist \*sglist,  
                      int hwcount,  
                      enum dma\_data\_direction dir)



# MMC/SD

## Blokovski sloj



Rukovalac MMC blok uređajem  
CONFIG\_MMC\_BLOCK  
drivers/mmc/card/{block, queue}.c



MMC jezgro  
CONFIG\_MMC  
drivers/mmc/core/



Rukovalac pojedinačnim MMC-om  
CONFIG\_MMC\_...  
/drivers/mmc/host/...



# Rukovalvac pojedinačnim MMC-om (1/2)



- ❖ Za svaki MMC
  - ❖ `struct mmc_host *mmc_alloc_host(int extra, struct device *dev)`
  - ❖ Inicijalizovati polja u strukturi mmc\_host: caps, ops, max\_phys\_segs, max\_hw\_segs, max\_blk\_size, max\_blk\_count, max\_req\_size
  - ❖ `int mmc_add_host(struct mmc_host *host)`
- ❖ Odregistracija
  - ❖ `void mmc_remove_host(struct mmc_host *host)`
  - ❖ `void mmc_free_host(struct mmc_host *host)`



# Rukovalvac pojedinačnim MMC-om (2/2)



- ❖ Polje mmc\_host->ops pokazuje na mmc\_host\_ops strukturu
  - ❖ Obrada I/O zahteva
    - ❖ void (\*request)(struct mmc\_host \*host,  
struct mmc\_request \*req);
  - ❖ Konfiguracija
    - ❖ void (\*set\_ios)(struct mmc\_host \*host,  
struct mmc\_ios \*ios);
  - ❖ Provera statusa
    - ❖ int (\*get\_ro)(struct mmc\_host \*host);
  - ❖ Provera prisutnosti kartice
    - ❖ status int (\*get\_cd)(struct mmc\_host \*host);



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Razvoj rukovalaca  
Treći deo



# Sadržaj



- ❖ Razvoj rukovalaca
  - ❖ Debugovanje
  - ❖ Udaljeno debugovanje



Debagovanje

# RAZVOJ RUKOVALACA



# Debagovanje sa printk (1/2)

- ❖ Univerzalna tehnika debagovanja sa ispisima
- ❖ Ispisi se pojavljuju u konzoli ili u `/var/log/messages` u zavisnosti od prioriteta
  - ❖ Kontrolisano pomoću kernel parametra `loglevel` ili kroz `/proc/sys/kernel/printk`
- ❖ Dostupni prioriteti ( `include/linux/kernel.h` )

```
#define KERN_EMERG  "<0>" /* system is unusable */  
#define KERN_ALERT   "<1>" /* action must be taken  
                      immediately */  
#define KERN_CRIT    "<2>" /* critical conditions */  
#define KERN_ERR     "<3>" /* error conditions */  
#define KERN_WARNING  "<4>" /* warning conditions */  
#define KERN_NOTICE   "<5>" /* normal but significant  
                           condition */  
#define KERN_INFO    "<6>" /* informational */  
#define KERN_DEBUG   "<7>" /* debug-level messages */
```



## Debagovanje sa printk (2/2)

- ❖ Zastarela tehnika, više nije preporučljiva
- ❖ pr\_\* familija funkcija: pr\_emerg(), pr\_alert(), pr\_critic(), pr\_err(), pr\_warning(), pr\_notice(), pr\_info(), pr\_cont() i specijalna pr\_debug()
  - ❖ Argument je string
  - ❖ Definisane u zaglavlju include/linux/printk.h
- ❖ dev\_\* familija funkcija: dev\_emerg(), dev\_alert(), dev\_crit(), dev\_err(), dev\_warning(), dev\_notice(), dev\_info() i specijalna dev\_debug()
  - ❖ Prvi argument je pokazivač na struct device strukturu, a drugi je string
  - ❖ Definisane u zaglavlju include/linux/device.h
  - ❖ Koriste se u rukovaocima koji koiste Linuksov model rukavaoca



## pr\_debug() i dev\_debug() (1/2)

- ❖ Kada se rukovalac prevede sa definisanim simbolom DEBUG, sve ove poruke se prevode i prikazuju se na debug nivou
- ❖ DEBUG simbol se može definisati
  - ❖ Iskozom #define DEBUG na pocetku rukovaoca
  - ❖ Korišćenjem ccflags-\$(CONFIG\_DRIVER) += -DDEBUG u Makefile-u



## pr\_debug() i dev\_debug() (2/2)

- ❖ Kada se kernel prevede sa CONFIG\_DYNAMIC\_DEBUG uključenim, ove poruke se mogu dinamički uključivati na nivou datoteke, modula ili poruke
  - ❖ Više detalja u Documentation/dynamic-debug-howto.txt
- ❖ Kada DEBUG simbol nije definisan ove poruke se ne prevode



# Debagovanje sa /proc i /sys (1/2)



- ❖ Umesto ispisivanja poruka u kernelski log informacije se mogu učiniti dostupnim korisničkom prostoru kroz datoteku u **/proc** ili **/sys** koja se registruje u rukovaocu
- ❖ Može da prikaže bilo kakvu informaciju o uređaju ili rukovaocu
- ❖ Može da se koristi i za slanje podataka i za kontrolu rukovaoca
- ❖ Upozorenje: Dostupno svima
  - ❖ U fazi proizvodnje treba ukloniti sprege za debagovanje
  - ❖ Od pojave **debugfs**-a nije više preferirani način debagovanja



# Debagovanje sa /proc i /sys (2/2)



## ❖ Primeri

- ❖ `cat /proc/acme/stats`
  - ❖ Prikazuje statistiku o acme rukovaocu
- ❖ `cat /proc/acme/globals`
  - ❖ Prikazuje vrednosti globalnih varijabli koje koristi rukovalac
- ❖ `echo 600000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed`
  - ❖ Podešava brzinu procesora



# Debugfs

- ❖ Virtuelni sistem datoteka za prikaz debag informacija u korisničkom prostoru
- ❖ Konfiguracija kernela
  - ❖ Kernel hacking > Debug Filesystem (DEBUG\_FS)
- ❖ Jednostavnije za kodovanje od sprega u /proc ili /sys
  - ❖ Debag sprega nestaje kada se isključi u konfiguraciji
- ❖ Primer mauntovanja
  - ❖ `sudo mount -t debugfs none /mnt/debugfs`



# Debugfs API (1/2)

- ❖ Pravljenje poddirektorijuma :
  - ❖ `struct dentry *debugfs_create_dir(const char *name, struct dentry *parent);`
- ❖ Prikaz celobrojne promenljive kao datoteke u DebugFS-u
  - ❖ `struct dentry *debugfs_create_{u,x}{8,16,32} (const char *name, mode_t mode, struct dentry *parent, u8 *value);`
    - ❖ u je decimalni prikaz
    - ❖ x je heksadecimalni prikaz



## Debugfs API (2/2)

- ❖ Prikaz binarnog bloba u DebugFS-u
  - ❖ `struct dentry *debugfs_create_blob(const char *name, mode_t mode, struct dentry *parent, struct debugfs_blob_wrapper *blob);`
- ❖ Moguće je koristiti i generičku funkciju `debugfs_create_file()` u razne svrhe



# Jednostavan debugfs primer

```
#include <linux/debugfs.h>

static char *acme_buf;                                // module buffer
static unsigned long acme_bufsize;
static struct debugfs_blob_wrapper acme_blob;
static struct dentry *acme_buf_dentry;

static u32 acme_state;                               // module variable
static struct dentry *acme_state_dentry;

/* Module init */
acme_blob.data = acme_buf;
acme_blob.size = acme_bufsize;
acme_buf_dentry = debugfs_create_blob("acme_buf", S_IRUGO,           // Create
                                      NULL, &acme_blob);          // new files
acme_state_dentry = debugfs_create_bool("acme_state", S_IRUGO,        // in debugfs
                                       NULL, &acme_state);

/* Module exit */
debugfs_remove (acme_buf_dentry);                   // removing the files from debugfs
debugfs_remove (acme_state_dentry);
```



# Debagovanje sa ioctl

- ❖ ioctl() sistemski poziv se može koristiti za upit o informacijama ili za slanje komandi rukovaocu
- ❖ Poziva ioctl operaciju nad datotekom koja se registruje u rukovaocu
- ❖ Prednost - debag sprega nije javna (može se čak ostaviti u sistemu i kada dospe do korisnika)

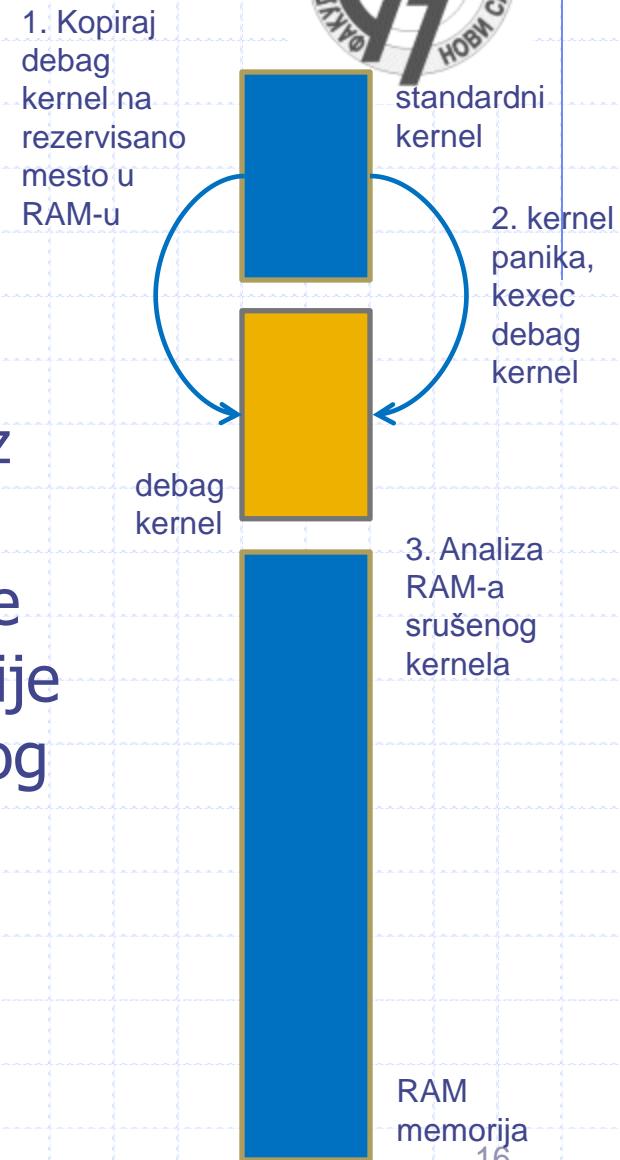


# Debagovanje sa GDB-om

- ❖ Ukoliko se kernel pokrene iz debagera na istom računaru uticaće na ponašanje kernela na računaru
- ❖ GDB može da pristupi trenutnom stanju kernela
  - ❖ `gdb /usr/src/linux/vmlinux /proc/kcore`
- ❖ Može da se pristupi kernelskim strukturama, da se prate pokazivači (samo čitanje je dozvoljeno)
- ❖ Kernel mora da bude preveden sa `CONFIG_DEBUG_INFO` (Kernel hacking sekcija)

# Analiza kernelske greške sa kexec-om

- ❖ **kexec sistemski poziv**
  - ❖ omogućeva pozivanje novog kernela bez restartovanja sistema
- ❖ Ideja: posle kernel panike novi kernel se automatski učitava sa rezervisane lokacije u RAM-u i vrši analizu memorije srušenog kernela





# Još saveta za debagovanje

- ❖ Omogućiti CONFIG\_KALLSYMS\_ALL
  - ❖ General Setup > Configure standard kernel features
  - ❖ poruke o oops-evima se dobijaju sa imenima simbola umesto sa sirovim adresama
- ❖ Ukoliko kernel ne može da se pokrene a ne ispisuje nikakve poruke može da se uključi debagovanje na niskom nivou
  - ❖ CONFIG\_DEBUG\_LL=y



Udaljeno debagovanje

# RAZVOJ UGRAĐENIH SISTEMA

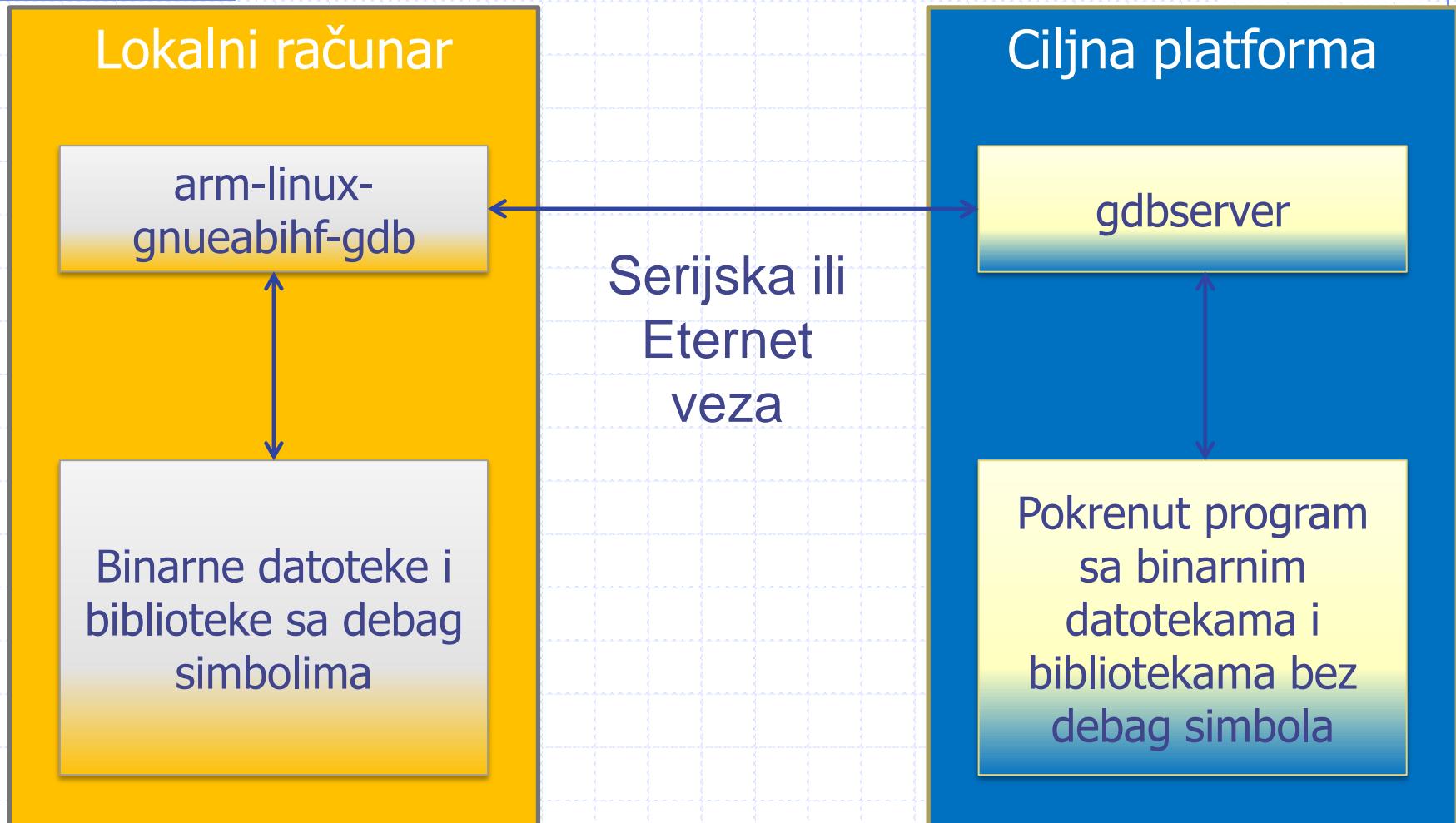


# Udaljeno debagovanje

- ❖ U neugrađenim sistemima debagovanje se uglavnom izvršava **gdb-om**
- ❖ gdb ima direktni pristup binarnim datotekama i bibliotekama prevednim sa debag simbolima
- ❖ U ugrađenim sistemima ciljna platforma je često ograničena memorijom i ne može da dozvoli direktno debagovanje **gdb-om**
- ❖ Bolja opcija je udaljeno debagovanje
  - ❖ gdb se koristi na lokalnom računaru
  - ❖ gdbserver se koristi na ciljnoj platformi



# Udaljeno debagovanje - arhitektura





# Udaljeno debagovanje - zahtevi

- ❖ Zahtevi
  - ❖ Na lokalnom računaru
    - ❖ alati za prevodenje sa podrškom za `gdb` (`arm-linux-gnueabihf-gdb`)
    - ❖ Verzija `gdb`-a koja se pokreće na lokalnom računaru, ali razume specifičnosti ciljne platforme
  - ❖ Na ciljnoj platformi
    - ❖ `gdbserver` program preveden za ciljnu arhitekturu
    - ❖ Često se nalazi kao deo alata za prevodenje
  - ❖ Serijska ili Ethernet veza između lokalnog računara i ciljne platforme



# Udaljeno debagovanje - korišćenje

- ❖ Povezivanje gdbservera na pokrenut program na ciljnoj platformi

```
$ gdbserver program -p <pid>
```

- ❖ Pokrenuti na lokalnom računaru

```
$ arm-linux-gnueabihf-gdb
```

- ❖ Ostvarivanje veze sa gdbserver-om

```
(gdb) target remote <host:port>
```

- ❖ Učitavanje simbola

```
(gdb) symbol-file <program>
```

- ❖ Nakon ovog koraka **gdb** može da se koristi kao i pri normalnom debagovanju na lokalnom računaru



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Linuks uređaj i model rukovaoca  
Prvi deo



# Sadržaj



- ❖ Linuks uređaj i model rukovaoca
  - ❖ Uvod
  - ❖ Primer USB magistrale
  - ❖ Rukovaoci platforme
  - ❖ Uvod u I2C podsistem



Uvod

# LINUX UREĐAJ I MODEL RUKOVАОCA



# Potreba za modelom uređaja? (1/2)



- ❖ Linuks kernel se pokreće na širokom spektru arhitektura i platformi fizičke arhitekture. Stoga, postoji potreba da se maksimizuje upotreba istog koda na različitim platformama.
  
- ❖ Na primer, želimo isti USB rukovalac da se koristi na x86 PC-ju, kao i na ARM platformi, čak iako su kontrolери за USB koji se koriste na ovim platformama različiti.



# Potreba za modelom uređaja? (2/2)



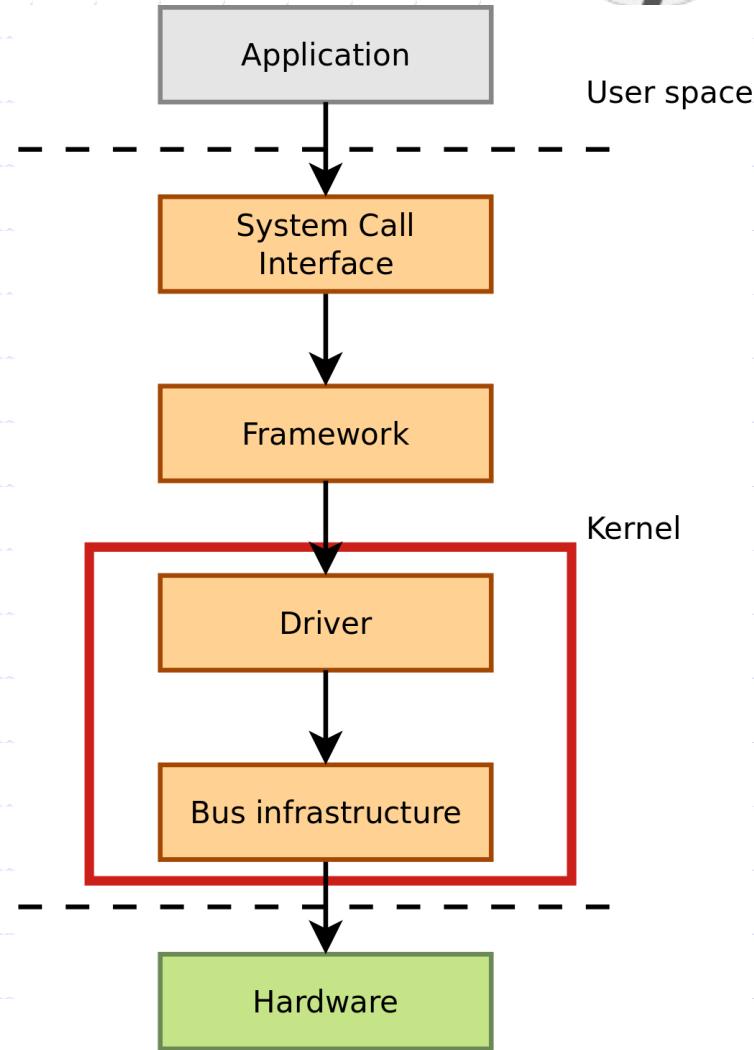
- ❖ Ovo zahteva jasnu organizaciju koda, sa rukovaocima uređaja odvojenim od rukovalaca kontrolera, sa opisom fizičke arhitekture odvojenom od samih rukovalaca, itd.
  
- ❖ Ovo nam dozvoljava model uređaja Linuksovog kernela, uz dodatne prednosti koji su pokriveni u ovoj sekciji.

# Kernel i rukovalac uređaja

- ❖ U Linuksu, rukovalac je uvek u sprezi sa:

- ❖ Razvojnim okvirom koji dozvoljava rukovaocu da generički koristi odlike fizičke arhitekture.
- ❖ Infrastrukturom magistrale koja je deo modela uređaja da otkrije/komunicira sa fizičkom arhitekturom

- ❖ Ova sekcija se fokusira na model uređaja, dok se radni okviri kernela pokrivaju drugom prilikom.





# Strukture podataka modela uređaja



- ❖ Model uređaja je organizovan oko tri glavne strukture podataka:
  - ❖ `struct bus_type` struktura, koja predstavlja jedan tip magistrale (USB, PCI, I2C, itd.)
  - ❖ `struct device_driver` struktura, koja predstavlja jedan rukovalac koji je sposoban da rukuje određenim uređajima na određenoj magistrali.
  - ❖ `struct device` struktura, koja predstavlja jedan uređaj povezan na magistralu
- ❖ Kernel koristi nasleđivanje da kreira specijalizovanije verzije `struct device_driver` i `struct device` za svaki podsistem magistrale.
- ❖ Da bi istražili model uređaja, mi ćemo:
  - ❖ Prvo pregledati popularne magistrale koje nude dinamičku enumeraciju, *USB magistrala*
  - ❖ Nastaviti sa pregledom kako se rukuje magistralama koje ne nude dinamičku enumeraciju.



# Rukovaoci magistralama

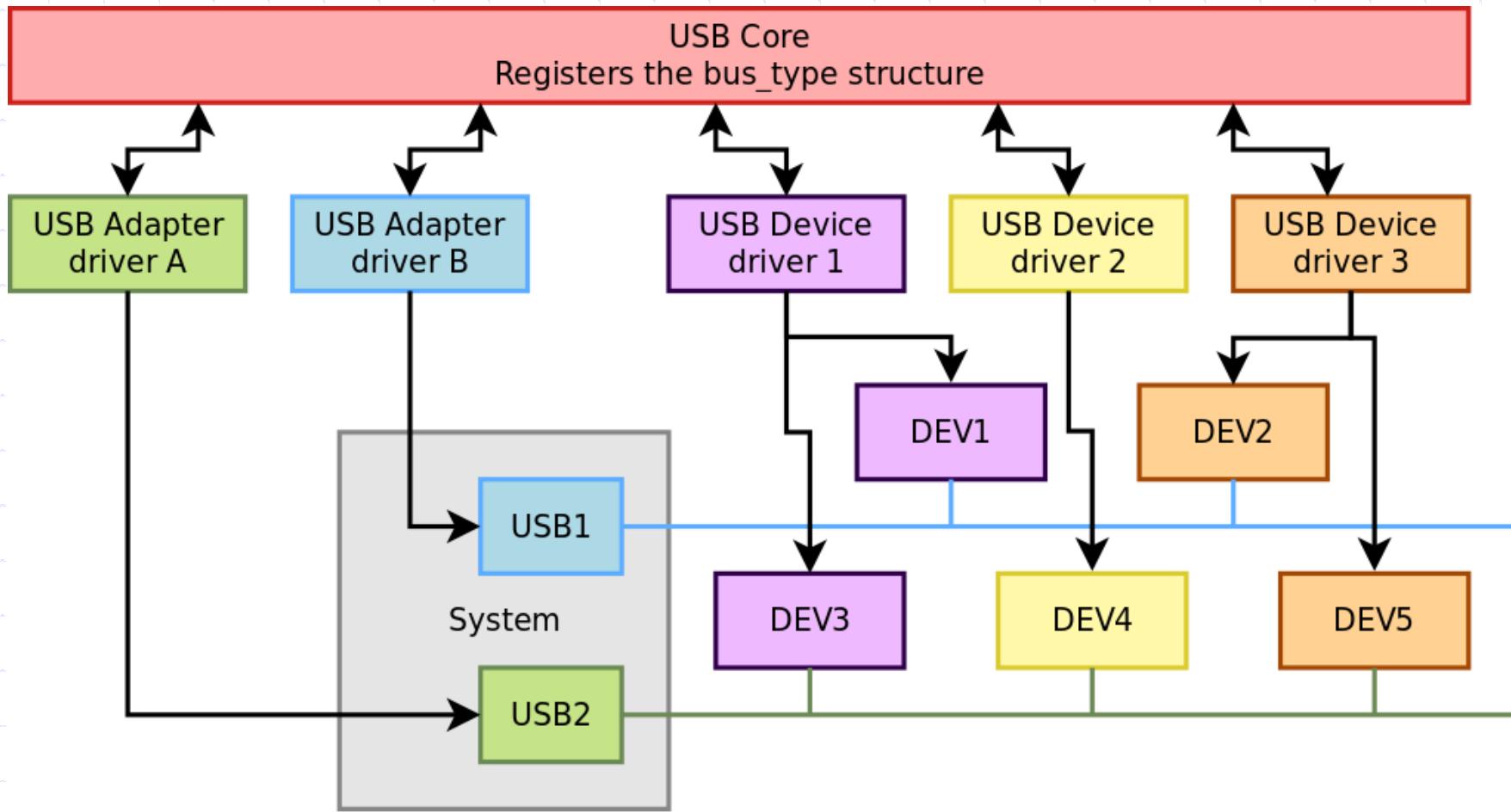
- ❖ Prva komponenta modela uređaja je rukovalac magistrale
  - ❖ Jedan rukovalac magistrale za svaku magistralu: USB, PCI, SPI, MMC, I2C, itd.
- ❖ Odgovoran je za:
  - ❖ Registrovanje tipa magistrale(**struct bus\_type**)
  - ❖ Dozvoljavanje registracije rukovalaca adaptera (USB kontrolери, I2C adapterи, itd), sposobnih da detektuju povezane uređaje i da obezbede mehanizam za komunikaciju sa uređajima
  - ❖ Dozvoljavanje registracije rukovalaca uređaja (USB uređaji, I2C uređaji, PCI uređaji, itd), za upravljanje uređajima
  - ❖ Uparivanje rukovalaca uređajima sa uređajima detektovanih od strane rukovalaca adaptera.
  - ❖ Obezbeđuje API i za rukovaće adaptera i za rukovaće uređaja
  - ❖ Definiše specifične strukture za rukovaće i uređaje, uglavnom **struct usb\_driver** i **struct usb\_interface**



Primer USB magistrale

# **LINUX UREĐAJ I MODEL RUKOVODIĆA**

# Primer: USB magistrala (1/2)





## Primer: USB magistrala (2/2)

- ❖ Infrastruktura jezgra (rukovalac magistrale)
  - ❖ drivers/usb/core
  - ❖ struct bus\_type je definisana u drivers/usb/core/driver.c i registrovana u drivers/usb/core/usb.c
- ❖ Rukovaoci adaptera
  - ❖ drivers/usb/host
  - ❖ Za EHCI, UHCI, OHCI, XHCI, i njihove implementacije na raznim sistemima (Atmel, IXP, Xilinx, OMAP, Samsung, PXA, itd)
- ❖ Rukovaoci uređaja
  - ❖ Svugde u stablu kernela, klasifikovani po njihovom tipu



# Primer rukovaoca uređaja

- ❖ Da bi ilustrovali kako su rukovaoci implementirani da rade sa modelom uređaja, pregledaćemo izvorni kod rukovaoca za USB mrežnu karticu
  - ❖ To je USB uređaj, dakle mora biti USB rukovalac uređaja
  - ❖ To je mrežni uređaj, dakle mora biti mrežni rukovalac
  - ❖ Većina rukovalaca se oslanja na infrastrukturu neke magistrale (u primeru **USB**) i registruju se u nekom radnom okviru (u primeru **network**)
- ❖ Pregledaćemo samo stranu rukovaoca uređajem, ne i stranu rukovaoca adaptera
- ❖ Rukovalac koji ćemo pregledati: **drivers/net/usb/rtl8150.c**



# Identifikatori uređaja

- ❖ Definiše se set uređaja kojima rukovalac može da upravlja, tako da jezgro USB-a zna za koje uređaje treba rukovalac da se koristi
- ❖ **MODULE\_DEVICE\_TABLE()** makro dozvoljava **depmod** da u vremenu prevođenja izvuče relaciju između identifikatora uređaja i rukovalaca, tako da **udev** može automatski da učita rukovaoce. Vidite:  
**/lib/modules/\$(uname -r)/modules.{alias,usbmap}**

```
static struct usb_device_id rtl8150_table[] = {
    { USB_DEVICE(VENDOR_ID_REALTEK, PRODUCT_ID_RTL8150) },
    { USB_DEVICE(VENDOR_ID_MELCO, PRODUCT_ID_LUAKTX) },
    { USB_DEVICE(VENDOR_ID_MICRONET, PRODUCT_ID_SP128AR) },
    { USB_DEVICE(VENDOR_ID_LONGSHINE, PRODUCT_ID_LCS8138TX) },
    [...]
}
MODULE_DEVICE_TABLE(usb, rtl8150_table);
```



# Instanciranje usb\_driver

- ❖ struct usb\_driver je struktura koju definiše USB jezgro. Svaki USB rukovalac uređajem mora da je instancira i da se registruje USB jezgru za korišćenje ove strukture
- ❖ Struktura se nasleđuje iz struct device\_driver, koja je definisana od strane modela uređaja.

```
static struct usb_driver rtl8150_driver = {  
    .name = "rtl8150",  
    .probe = rtl8150_probe,  
    .disconnect = rtl8150_disconnect,  
    .id_table = rtl8150_table,  
    .suspend = rtl8150_suspend,  
    .resume = rtl8150_resume  
};
```



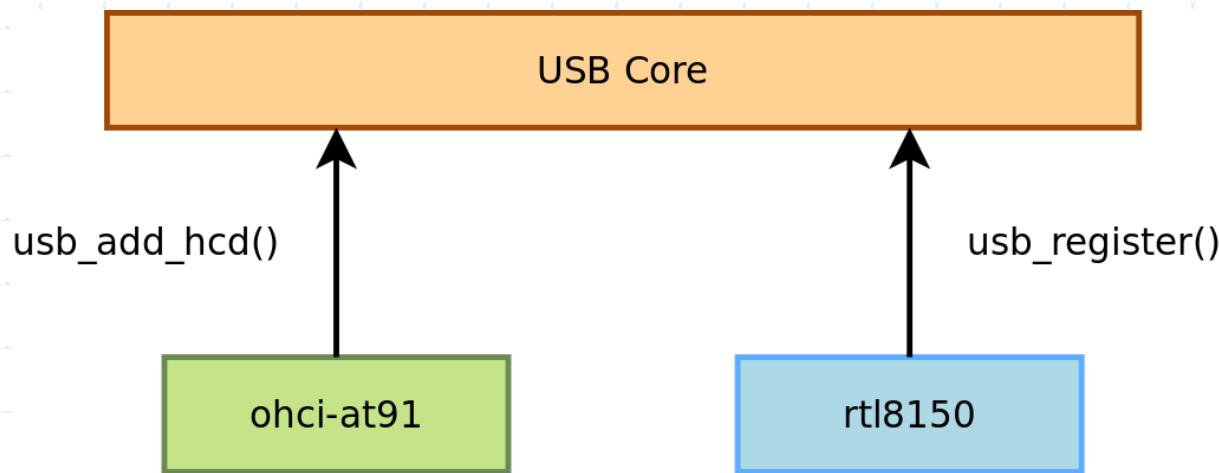
# Prijava/odjava rukovaoca

- ❖ Kada se rukovalac učita ili ukloni, neophodno je da se prijavi/odjavi USB jezgru
- ❖ To čini funkcijama `usb_register()` odnosno `usb_deregister()`, dostupnih od strane USB jezgra.
- ❖ Napomena: kod ispod je sada zamjenjen kraćim pozivom makroa `module_usb_driver()`.

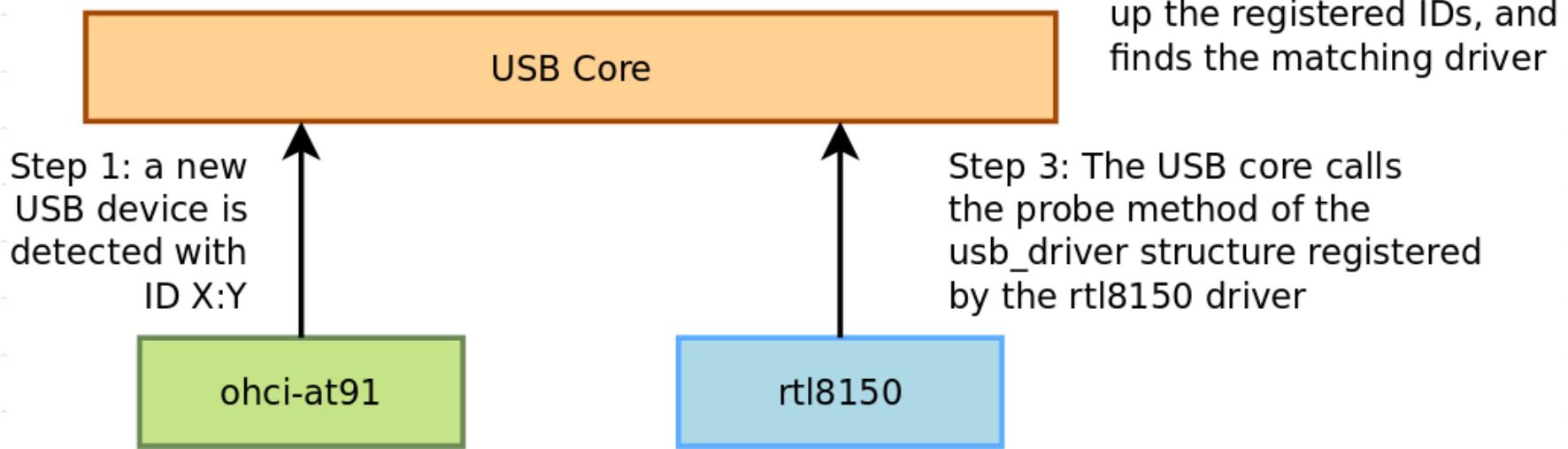
```
static int __init usb_rt18150_init(void)
{
    return usb_register(&rtl18150_driver);
}
static void __exit usb_rt18150_exit(void)
{
    usb_deregister(&rtl18150_driver);
}
module_init(usb_rt18150_init);
module_exit(usb_rt18150_exit);
```

# Inicijalizacija

- ❖ Rukovalac USB adapterom koji odgovara USB kontroleru sistema se prijavljuje USB jezgru
- ❖ **rtl8150** USB rukovalac uređaja se prijavljuje USB jezgru
- ❖ USB jegro sada zna vezu između **vendor/product IDs** od **rtl8150** i **struct usb\_driver** strukture ovog rukovaoca



# Kada je uređaj detektovan





# Probe() metoda

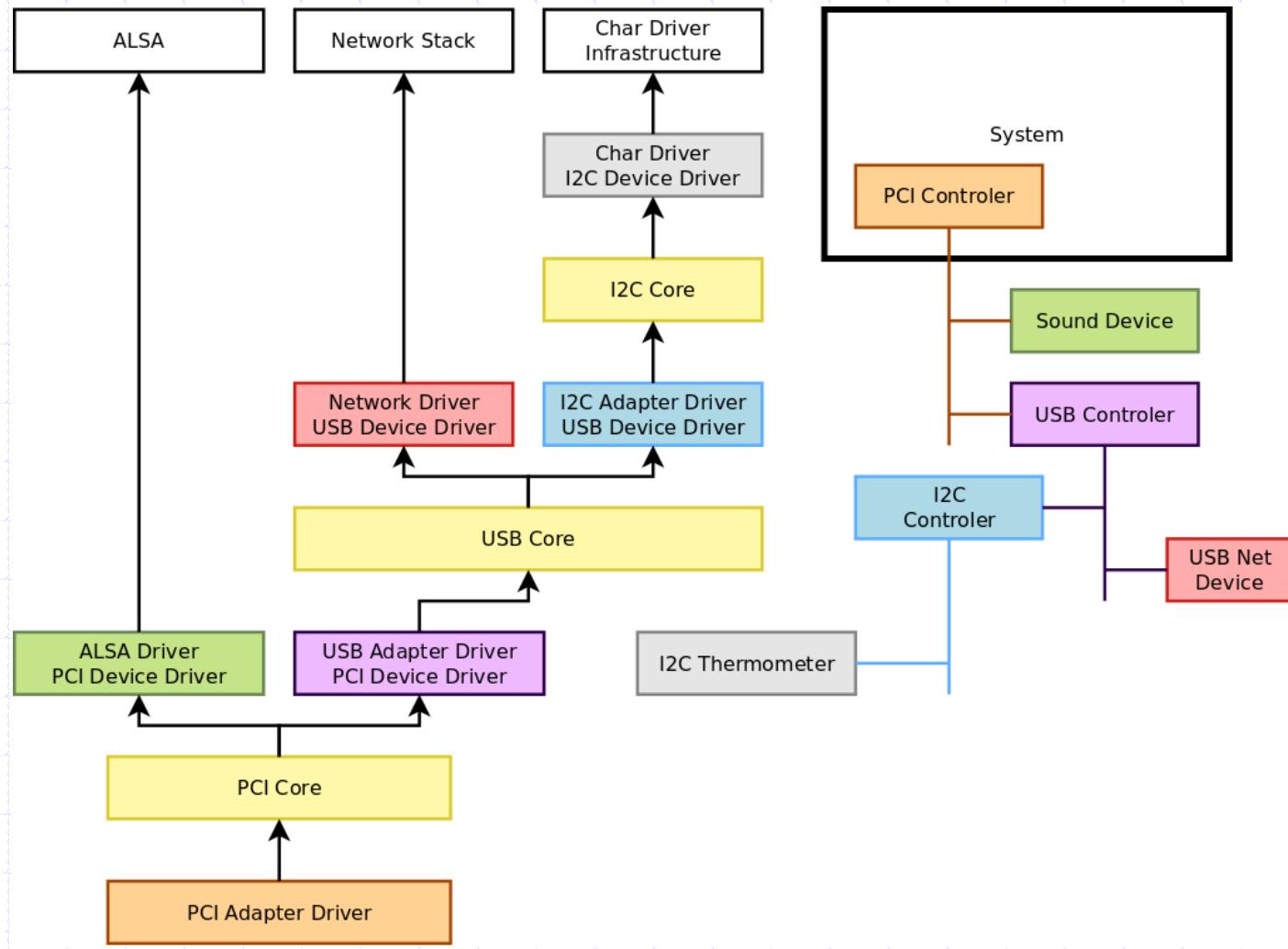
- ❖ probe() metoda prihvata kao argument strukturu koja opisuje uređaj, uglavnom definisanu od strane infrastrukture magistrale (**struct pci\_dev**, **struct usb\_interface**, itd)
- ❖ Ova funkcija je odgovorna za
  - ❖ Inicijalizaciju uređaja, mapiranje I/O memorije, registrovanje upravljanja prekida. Infrastruktura magistrale obezbeđuje metode za dobavljanje adresa, brojeva prekida i drugih informacija specifičnih za rukovaće.
  - ❖ Prijavu uređaja na odgovarajući kernel radni okvir, na primer na mrežnu infrastrukturu.



# Primer probe() metode

```
static int rtl8150_probe(struct usb_interface *intf,
                         const struct usb_device_id *id)
{
    rtl8150_t *dev;
    struct net_device *netdev;
    netdev = alloc_etherdev(sizeof(rtl8150_t));
    [...]
    dev = netdev_priv(netdev);
    tasklet_init(&dev->tl, rx_fixup, (unsigned long)dev);
    spin_lock_init(&dev->rx_pool_lock);
    [...]
    netdev->netdev_ops = &rtl8150_netdev_ops;
    alloc_all_urbs(dev);
    [...]
    usb_set_intfdata(intf, dev);
    SET_NETDEV_DEV(netdev, &intf->dev);
    register_netdev(netdev);
    return 0;
}
```

# Model je rekurzivan





Rukovaoci platforme

# **LINUX UREĐAJ I MODEL RUKOVAOCA**



## Neotkrive magistrale (1/2)

- ❖ Na ugrađenim sistemima, uređaji često nisu povezani magistralama koje podržavaju enumeraciju, hotplaging, i obezbeđivanje jedinstvenih identifikatora za uređaje.
  
- ❖ Na primer, uređaji na I2C ili SPI magistralama, ili uređaji koji su deo SOC-a.



## Neotkrive magistrale (2/2)

- ❖ Međutim, i dalje želimo da svi ovi uređaji budu deo modela uređaja.
- ❖ Takvi uređaji, umesto da budu dinamički otkriveni, moraju biti statički opisani na neki od načina:
  - ❖ U izvornom kodu kernela
  - ❖ Kroz stablo uređaja (Device tree), opisna datoteka fizičke arhitekture koja se koristi na nekim arhitekturama.



# Platformski uređaji

- ❖ Među neotkrivim magistralama, veliki deo čine uređaji koji su direktno deo SOC-a: UART kontroleri, Eternet kontroleri, SPI ili I2C kontroleri, grafčki ili audio uređaji, itd.
- ❖ U Linuks kernelu, specijalna magistrala je napravljena za rukovanje takvim uređajima i zove se platformska magistrala.
- ❖ Podržava platformske rukovaoce koji rukuju platformskim uređajima.
- ❖ Radi kao i bilo koja druga magistrala (USB, PCI), osim što su uređaji enumerisani staticki umesto da budu dinamički otkriveni.



# Instanciranje platformskih uređaja: stari način (1/3)



- ❖ Rukovalac implementira `struct platform_driver` strukturu (primer uzet iz `drivers/tty/serial/imx.c`, pojednostavljen)

```
static struct platform_driver serial_imx_driver = {  
    .probe = serial_imx_probe,  
    .remove = serial_imx_remove,  
    .id_table = imx_uart_devtype,  
    .driver = {  
        .name = "imx-uart",  
        .of_match_table = imx_uart_dt_ids,  
        .pm = &imx_serial_port_pm_ops,  
    },  
};
```

- ❖ I registruje svoj rukovalac u infrastrukturu platformskog rukovaoca

```
static int __init imx_serial_init(void) {  
    ret = platform_driver_register(&serial_imx_driver);  
}  
static void __exit imx_serial_cleanup(void) {  
    platform_driver_unregister(&serial_imx_driver);  
}
```



# Instanciranje platformskih uređaja: stari način (2/3)



- ❖ Platformski uređaji se definišu statički
  - ❖ Direktnim instanciranjem `struct platform_device` strukture, kako se radi na nekoliko starih ARM platformi. Definicija se radi ili board-specific ili SoC-specific kodom.
  - ❖ Korišćenjem stabla uređaja, kao što se radi na Power PC (i na većini ARM platformi) iz kojih se `struct platform_device` strukture kreiraju
- ❖ Primer na ARM-u, gde jeinstanciranje urađeno u `arch/arm/mach-imx/mx1ads.c`

```
static struct platform_device imx_uart1_device = {  
    .name = "imx-uart",  
    .id = 0,  
    .num_resources = ARRAY_SIZE(imx_uart1_resources),  
    .resource = imx_uart1_resources,  
    .dev = {  
        .platform_data = &uart_pdata,  
    }  
};
```



# Instanciranje platformskih uređaja: stari način (3/3)



- ❖ Uređaj je bio deo liste

```
static struct platform_device *devices[] __initdata = {  
    &cs89x0_device,  
    &imx_uart1_device,  
    &imx_uart2_device,  
};
```

- ❖ I lista uređaja se dodaje u sistem tokom inicijalizacije ploče

```
static void __init mx1ads_init(void)  
{  
    [...]  
    platform_add_devices(devices, ARRAY_SIZE(devices));  
}  
MACHINE_START(MX1ADS, "Freescale MX1ADS")  
    [...]  
    .init_machine = mx1ads_init,  
MACHINE_END
```



# Mehanizam resursa

- ❖ Rukovalac za svaki uređaj kojim upravlja tipično koristi resurse fizičke arhitekture: adrese za I/O registre, DMA kanale, IRQ linije, itd.
- ❖ Ove informacije mogu biti prikazane korišćenjem **struct resource**, i niz elemenata tipa **struct resource** je asociran sa **struct platform\_device**
- ❖ Dozvoljava rukovaocu da bude instanciran na više uređaja koji slično funkcionišu, ali sa različitim adresama, IRQ-ovima, itd.



# Deklarisanje resursa (stari način)

```
static struct resource imx_uart1_resources[] = {
    [0] = {
        .start = 0x00206000,
        .end = 0x002060FF,
        .flags = IORESOURCE_MEM,
    },
    [1] = {
        .start = (UART1_MINT_RX),
        .end = (UART1_MINT_RX),
        .flags = IORESOURCE_IRQ,
    },
};
```



# Korišćenje resursa (stari način)

- ❖ Kada je `struct platform_device` dodata u sistem korišćenjem `platform_add_device()`, pozvana je `probe()` metoda platforme
- ❖ Ova metoda je odgovorna za inicijalizaciju fizičke arhitekture, registrovanje uređaja na odgovarajući radni okvir (u našem slučaju, radni okvir rukovaoca serijskim uređajem)
- ❖ Platformski rukovalac ima prstup I/O resursima:

```
res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
base = ioremap(res->start, PAGE_SIZE);
sport->rxirq = platform_get_irq(pdev, 0);
```



# platform\_data mehanizam (stari način)

- ❖ Uz dobro definisane resurse, mnogim rukovaocima su potrebne specifične informacije za svaki platformski uređaj
- ❖ Te informacije mogu biti prosleđene korišćenjem `platform_data` polja `struct device` (koju `struct platform_device` nasleđuje)
- ❖ Pošto je `void *` pokazivač, može se koristiti za prosleđivanje bilo kojeg tipa podataka.
  - ❖ Uobičajeno, svaki rukovalac definiše strukturu podataka za prosleđivanje preko `struct platform_data`



# platform\_data primer (1/2)

- ❖ i.MX rukovalac serijskim prolazom definiše strukturu koja se prosleđuje kroz **struct platform\_data**

```
struct imxuart_platform_data {  
    int (*init)(struct platform_device *pdev);  
    void (*exit)(struct platform_device *pdev);  
    unsigned int flags;  
    void (*irda_enable)(int enable);  
    unsigned int irda_inv_rx:1;  
    unsigned int irda_inv_tx:1;  
    unsigned short transceiver_delay;  
};
```

- ❖ Kod **MX1ADS** ploče instancira takvu strukturu

```
static struct imxuart_platform_data uart1_pdata = {  
    .flags = IMXUART_HAVE_RTSCTS,  
};
```



# platform\_data primer (2/2)



- ❖ `uart_pdata` struktura je asocirana na `struct platform_device` strukturu u `MX1ADS` datoteci ploče (pravi kod je dosta komplikovaniji)

```
struct platform_device mxlads_uart1 = {
    .name = "imx-uart",
    .dev = {
        .platform_data = &uart1_pdata,
    },
    .resource = imx_uart1_resources,
    [...]
};
```

- ❖ Rukovalac može da pristupi platformskim podacima:

```
static int serial_imx_probe(struct platform_device *pdev)
{
    struct imxuart_platform_data *pdata;
    pdata = pdev->dev.platform_data;
    if (pdata && (pdata->flags & IMXUART_HAVE_RTSCTS))
        sport->have_rtscts = 1;
    [...]
};
```



# Stablo uređaja

- ❖ Na dosta ugrađenih arhitektura, ručnainstancijacija platformskih uređaja se smatrala previše opisnom i ne baš lako održivom.
- ❖ Takve arhitekture koriste, ili su počele, stablo uređaja (*Device Tree*).
- ❖ To je stablo čvorova koje modeluje hijerarhiju uređaja u sistemu, od uređaja unutar procesora do uređaja na ploči.
- ❖ Svaki čvor može da ima veći broj odlika koje opisuju odlike uređaja: adrese, prekide, radni takt, itd.
- ❖ U vremenu pokretanja, kernel dobija prevedenu verziju, Device Tree Blob, koji se parsira da bi instancirao sve uređaje opisane u stablu uređaja.
- ❖ Na ARM-u, nalaze se u [arch/arm/boot/dts](#).



# Primer stabla uređaja

```
uart0: serial@44e09000 {  
    compatible = "ti,omap3-uart";  
    ti,hwmods = "uart1";  
    clock-frequency = <48000000>;  
    reg = <0x44e09000 0x2000>;  
    interrupts = <72>;  
    status = "disabled";  
};
```

- ❖ serial@44e09000 je **ime čvora**
- ❖ uart0 je **alias**, na koji se može referencirati u drugim delovima stabla uređaja pomoću &uart0
- ❖ Ostale linije su **odlike**. Njihove vrednosti su uglavnom stringovi, liste celih brojeva ili reference na druge čvorove.



# Nasleđivanje stabla uređaja (1/2)



- ❖ Svaka platforma fizičke arhitekture ima svoje stablo uređaja.
- ❖ Međutim, nekoliko njih koriste isti procesor, a često različiti procesori iste familije dele set nekih odlika.
- ❖ Da bi se nasleđivanje dozvolilo, *datoteka stabla uređaja* može da uključi neku drugu. Stabla opisana u uključenoj datoteci imaju prednost u odnosu na stablo opisano u osnovnoj datoteci. Ovo može da se uradi:
  - ❖ Ili korišćenjem **/include/** komande unutar datoteke stabla uređaja.
  - ❖ Ili korišćenjem **#include** komande, koja zahteva poziv C predprocesora pre parsiranja stabla uređaja.
- ❖ Linuks trenutno koristi ili jednu ili drugu tehniku (Na primer, razlikuje se upotreba tehnika na različitim ARM arhitekturama).



# Nasleđivanje stabla uređaja (2/2)



Definition of the AM33xx SoC

```
/ {  
    compatible = "ti,am33xx";  
    [...]  
    ocp {  
        [...]  
        uart0: serial@44e09000 {  
            compatible = "ti,am3352-uart",  
                        "ti,omap3-uart";  
            reg = <0x44e09000 0x2000>;  
            interrupts = <72>;  
            status = "disabled";  
            [...]  
        };  
    };  
};  
am33xx.dtsi
```



Common definitions for BeagleBone boards

```
[...]  
    &uart0: serial@44e09000 {  
        pinctrl-names = "default";  
        pinctrl-0 = <&uart0_pins>;  
        status = "okay";  
    };  
am335x-bone-common.dtsi
```



Definition for BeagleBone Black

```
#include "am33xx.dtsi"  
#include "am335x-bone-common.dtsi"  
/  
    model = "TI AM335x BeagleBone Black";  
    compatible = "ti,am335x-bone-black",  
                "ti,am335x-bone",  
                "ti,am33xx";  
    [...]  
am335x-boneblack.dts
```



Compiled DTB

```
/ {  
    compatible = "ti,am335x-bone-black", "ti,am335x-bone",  
                "ti,am33xx";  
    model = "TI AMM335x BeagleBone Black";  
    [...]  
    ocp {  
        uart0: serial@44e09000 {  
            compatible = "ti,am3352-uart", "ti,omap3-uart";  
            reg = <0x44e09000 0x2000>;  
            interrupts = <72>;  
            pinctrl-names = "default";  
            pinctrl-0 = <&uart0_pins>;  
            status = "okay";  
        };  
    };  
};  
am335x-boneblack.dtb
```

Note: the real DTB is in binary format.  
Here we show the text equivalent of the  
DTB contents;



# Stablo uređaja: compatible string

- ❖ Sa stablom uređaja, uređaj je vezan za odgovarajući rukovalac preko **compatible** ključne reči.
- ❖ **of\_match\_table** polje struct device\_driver izlistava **compatible** ključne reči koje rukovalac podržava.

```
#if defined(CONFIG_OF)
static const struct of_device_id omap_serial_of_match[] = {
    { .compatible = "ti,omap2-uart" },
    { .compatible = "ti,omap3-uart" },
    { .compatible = "ti,omap4-uart" },
    { },
};

MODULE_DEVICE_TABLE(of, omap_serial_of_match);
#endif

static struct platform_driver serial_omap_driver = {
    .probe = serial_omap_probe,
    .remove = serial_omap_remove,
    .driver = {
        .name = DRIVER_NAME,
        .pm = &serial_omap_dev_pm_ops,
        .of_match_table = of_match_ptr(omap_serial_of_match),
    },
};
```



## Resursi stabla uređaja

- ❖ Rukovaoci će koristiti isti mehanizam koji smo videli ranije da dobave osnovne informacije: brojeve prekida, fizičke adrese, itd.
- ❖ Lista dostupnih resursa će se izgraditi od strane kernela u vreme pokretanja iz stabla uređaja, tako da ne moramo da nepotrebno tražimo u stablu uređaja kada učitavamo naš rukovalac.
- ❖ Sve dodatne informacije će biti specifične za rukovaoce ili za klasu kojoj pripadaju i biće definisane u **bindings**



# Stablo uređaja: veze

- ❖ **compatible** ključna reč i asocirane odlike definišu ono što nazivamo veze stabla uređaja.
- ❖ Veze su dokumentovane u [Documentation/devicetree/bindings](#)
- ❖ Pošto je stablo uređaja deo kernel ABI-ja, veze moraju biti kompatibilne tokom vremena.
  - ❖ Novi kernel mora biti sposoban da koristi staro stablo uređaja.
  - ❖ Ovo zahteva veoma pažljiv dizajn veza. One se sve pregledaju u [devicetree@vger.kernel.org](mailto:devicetree@vger.kernel.org) mailing listi.
- ❖ Veze stabla uređaja bi trebalo da sadrže **samo opis fizičke arhitekture, a ne konfiguraciju**.
  - ❖ Broj prekida može biti deo stabla uređaja sve dok opisuje fizičku arhitekturu.
  - ❖ Ali ne i da li bi trebalo da se DMA koristi za uređaj ili ne, jer je to konfiguraciona opcija.



## sysfs



- ❖ Strukture magistrala, uređaja, rukovalaca, itd. su interne u kernelu
- ❖ sysfs virutelni sistem datoteka nudi mehanizam za prikazivanje informacija u korisničkom prostoru
- ❖ Udev ih koristi za automatsko učitavanje modula, učitavanje firmvera, kreiranje datoteka uređaja, itd.
- ❖ sysfs je obično postavljen na /sys
  - ❖ /sys/bus/ sadrži listu magistrala
  - ❖ /sys/devices/ sadrži listu uređaja
  - ❖ /sys/class enumeriše uređaje po klasi (net, input, block,...), magistrale na koju su povezani
- ❖ Pregledajte /sys na vašoj radnoj stanici.



Uvod u I2C podsistem

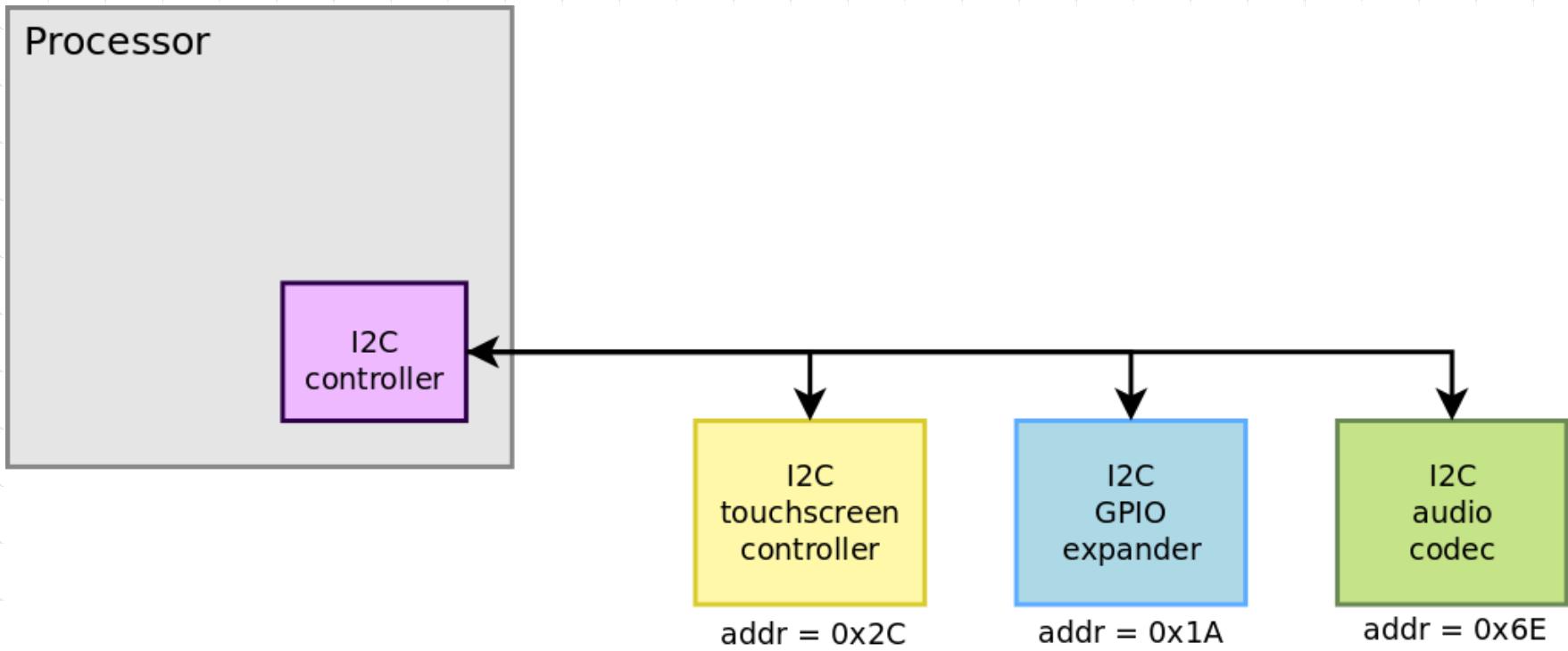
# **LINUX UREĐAJ I MODEL RUKOVAOCA**



# Šta je I2C?

- ❖ Vrlo često korišćena spora magistrala za povezivanje uređaja na procesor.
- ❖ Koristi samo dve žice: **SDA** za podatke, **SCL** za radni takt.
- ❖ To je master/slave magistrala: samo master može da inicira transakcije, a slave-ovi mogu samo da odgovore na inicirane transakcije.
- ❖ U Linuks sistemu, I2C kontroler ugrađen u procesor je tipično master, koji kontroliše magistralu.
- ❖ Svaki slave uređaj je identifikovan jedinstvenom I2C adresom. Svaki prenos iniciran od mastera sadrži ovu adresu, koji dozvoljava odgovarajućem slave-u da prepozna šta treba da odgovori na dati prenos.

# Primer I2C magistrale





## I2C podsistem (1/2)

- ❖ Kao i svi podsistemi magistrala, I2C podsistem je odgovoran za:
  - ❖ Obezbeđivanje API-ja za implementaciju I2C rukovalaca kontrolera
  - ❖ Obezbeđivanje API-ja za implementaciju I2C rukovalaca uređaja, u kernel prostoru
  - ❖ Obezbeđivanje API-ja za implementaciju I2C rukovalaca uređaja, u korisničkom prostoru
- ❖ Jezgro I2C podsistema se nalazi u **drivers/i2c**.



## I2C podsistem (2/2)

- ❖ I2C rukovaoci kontrolera se nalaze u **drivers/i2c/busses**.
- ❖ I2C rukovaoci uređaja se nalaze u **drivers/**, zavisno od tipa uređaja (primer: **drivers/input** za ulazne uređaje).



# Registrovanje I2C rukovaoca uređaja

- ❖ Kao i svi drugi podsistemi, I2C podsistem definiše **struct i2c\_driver** koji nasleđuje **struct device\_driver**, i koji mora biti instanciran i registrovan od strane svakog I2C rukovaoca uređaja.
  - ❖ Kao i obično, ova struktura pokazuje na **->probe()** i **->remove()** funkcije
  - ❖ Takođe, sadrži polje **id\_table** koje mora da pokazuje na listu ID-a uređaja (koja je lista n-torki koje sadrže string i druge privatne podatke rukovalaca). Koristi se za **probe** bez stabla uređaja I2C uređaja.
- ❖ **i2c\_add\_driver()** i **i2c\_del\_driver()** se koriste za registraciju i uklanjanje rukovaoca.
- ❖ Ako rukvoalac ne radi ništa drugo u **init()/exit()** funkcijama, savetuje se upotreba **module\_i2c\_driver()** makroa.



# Registrovanje I2C rukovaoca uredaja: primer



```
static const struct i2c_device_id <driver>_id[] = {
    { "<device-name>", 0 },
    {}
};

MODULE_DEVICE_TABLE(i2c, <driver>_id);

#ifndef CONFIG_OF
static const struct of_device_id <driver>_dt_ids[] = {
    { .compatible = "<vendor>,<device-name>", },
    {}
};

MODULE_DEVICE_TABLE(of, <driver>_dt_ids);
#endif

static struct i2c_driver <driver>_driver = {
    .probe = <driver>_probe,
    .remove = <driver>_remove,
    .id_table = <driver>_id,
    .driver = {
        .name = "<driver-name>",
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(<driver>_dt_ids),
    },
};

module_i2c_driver(<driver>_driver);
```



# Registrovanje I2C uređaja: bez stabla uređaja

- ❖ Na platformama bez stabla uređaja, `struct i2c_board_info` dozvoljava opisivanje kako je I2C uređaj povezan na ploču.
- ❖ Takve strukture se definišu `I2C_BOARD_INFO()` pomoćnim makroom.
  - ❖ Kao argument prihvata ime uređaja i slave adresu uređaja na magistrali.
- ❖ Niz takvih struktura je registrovan na bazi "po magistrali" korišćenjem `i2c_register_board_info()`, kada se inicijalizuje platforma.



# Registrovanje I2C uređaja: bez stabla uređaja: primer

```
static struct i2c_board_info <board>_i2c_devices[] __initdata = {
{
    I2C_BOARD_INFO("cs42151", 0x4a),
},
};

void board_init(void)
{
    /*
     * Here should be the registration of all devices, including
     * the I2C controller device.
     */
    i2c_register_board_info(0, <board>_i2c_devices,
                           ARRAY_SIZE(<board>_i2c_devices));
    /* More devices registered here */
}
```



# Registrovanje I2C uređaja: u stablu uređaja

- ❖ U stablu uređaja, I2C kontroler uređaj se tipično definiše u **.dtsi** datoteci koja opisuje procesor.
  - ❖ Uobičajeno definisan sa **status = "disabled"**.
- ❖ Na nivou platforme/ploče:
  - ❖ I2C kontroler uređaj je uključen (**status = "okay"**)
  - ❖ Frekvencija I2C magistrale je definisana korišćenjem **clock-frequency** odlike.
  - ❖ I2C uređaji na magistrali su opisani kao potomci čvora I2C kontroler, gde **reg** odlika daje I2C slave-u adresu na magistrali.



# Registrovanje I2C uređaja: u stablu uređaja: primer (1/2)



## ❖ Definicija I2C kontrolera, sun7i-a20.dtsi datoteka

```
i2c0: i2c@01c2ac00 {
    compatible = "allwinner,sun7i-a20-i2c",
                 "allwinner,sun4i-a10-i2c";
    reg = <0x01c2ac00 0x400>;
    interrupts = <GIC_SPI 7 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&apb1_gates 0>;
    status = "disabled";
    #address-cells = <1>;
    #size-cells = <0>;
};
```



# Registrovanje I2C uređaja: u stablu uređaja: primer (2/2)



- ❖ Definicija I2C uređaja, sun7i-a20-bananapi.dts datoteka

```
&i2c0 {  
    pinctrl-names = "default";  
    pinctrl-0 = <&i2c0_pins_a>;  
    status = "okay";  
    axp209: pmic@34 {  
        reg = <0x34>;  
        interrupt-parent = <&nmi_intc>;  
        interrupts = <0 IRQ_TYPE_LEVEL_LOW>;  
    };  
};
```



# probe() i remove()

- ❖ ->**probe()** funkcija je odgovorna za inicijalizaciju uređaja i registraciju u odgovarajući kernel radni okvir. Prihvata kao argument:
  - ❖ struct **i2c\_client** \* pokazivač, koji predstavlja sam I2C uređaj. Ova struktura nasleđuje **struct device**.
  - ❖ struct **i2c\_device\_id** \* pokazivač, koja pokazuje na I2C device ID unos koji se odosi na uređaj koji se probe-uje.
- ❖ ->**remove()** funkcija je odgovorna za uklanjanje uređaja iz kernel radnog okvira i za njegovo gašenje. Prihvata kao argument:
  - ❖ Isti struct **i2c\_client** \* pokazivač koji je prosleđen kao argument u ->probe()



# Probe/remove primer

```
static int <driver>_probe(struct i2c_client *client,
                           const struct i2c_device_id *id)
{
    /* initialize device */
    /* register to a kernel framework */
    i2c_set_clientdata(client, <private data>);
    return 0;
}
static int <driver>_remove(struct i2c_client *client)
{
    <private data> = i2c_get_clientdata(client);
    /* unregister device from kernel framework */
    /* shut down the device */
    return 0;
}
```



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Linuks uređaj i model rukovaoca  
Drugi deo



# Sadržaj



- ❖ Linuks uređaj i model rukovaoca
  - ❖ Komunikacija sa I2C uređajem
  - ❖ Uvod u pin muxing



Komunikacija sa I2C uređajem

# **LINUX UREĐAJ I MODEL RUKOVAOCA**



# Komunikacija sa I2C uređajem:

## API

- ❖ Najjednostavniji API za komunikaciju sa I2C uređajem obezbeđuje funkcije za slanje i prihvatanje podataka:
  - ❖ `int i2c_master_send(struct i2c_client *client, const char *buf, int count);`  
Šalje klijentu sadržaj iz `buf`.
  - ❖ `int i2c_master_recv(struct i2c_client *client, char *buf, int count);`  
Prihvata broj bajtova od klijenta i smešta ih u `buf`.



# Komunikacija sa I2C uređajem: transfer poruka

- ❖ API za transfer poruka dozvoljava opis **transfера** koji se sastoji iz nekoliko **poruka**, gde je svaka poruka transakcija u jednom pravcu:
  - ❖ `int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msg, int num);`
  - ❖ `struct i2c_adapter *` pokazivač može da se dobavi korišćenjem `client->adapter`
  - ❖ `struct i2c_msg` struktura definiše dužinu, lokaciju i smer poruke.



# I2C: primer transfera poruka

```
struct i2c_msg msg[2];
int error;
u8 start_reg;
u8 buf[10];

msg[0].addr = client->addr;
msg[0].flags = 0;
msg[0].len = 1;
msg[0].buf = &start_reg;
start_reg = 0x10;

msg[1].addr = client->addr;
msg[1].flags = I2C_M_RD;
msg[1].len = sizeof(buf);
msg[1].buf = buf;

error = i2c_transfer(client->adapter, msg, 2);
```



# SMBus pozivi

- ❖ SMBus je podgrupa I2C protokola.
- ❖ Definiše standardni set transakcija, na primer čitanje ili upis regista u uređaj.
- ❖ Linuks obezbeđuje SMBus funkcije koje bi trebale biti korišćene umesto postojećeg API-ja, ukoliko I2C uređaj podržava takav standardni tip transakcija. Tada, rukovalac može biti korišćen od strane SMBus-a i I2C adaptera (ne mogu da se koriste I2C komande na SMBus adapterima).
- ❖ Primer: `i2c_smbus_read_byte_data()` funkcija dozvoljava čitanje jednog bajta podataka iz regista uređaja.
  - ❖ Radi sledeće operacije: `S Addr Wr [A] Comm [A] S Addr Rd [A] [Data]`   
`NA P`
  - ❖ Što znači da prvo piše komandu dužine jednog bajta (Comm), a nakon toga čita jedan bajt podataka ([Data]).
- ❖ [Documentation/i2c/smbus-protocol](#) sadrži detaljan prikaz navedenog.



# Spisak SMBus funkcija

- ❖ Čitanje/Upis jednog bajta
  - ❖ s32 i2c\_smbus\_read\_byte(const struct i2c\_client \*client);
  - ❖ s32 i2c\_smbus\_write\_byte(const struct i2c\_client \*client, u8 value);
- ❖ Pisanje komandnog bajta i čitanje ili upis jednog bajta
  - ❖ s32 i2c\_smbus\_read\_byte\_data(const struct i2c\_client \*client, u8 command);
  - ❖ s32 i2c\_smbus\_write\_byte\_data(const struct i2c\_client \*client, u8 command, u8 value);
- ❖ Pisanje komandnog bajta i čitanje ili upis jedne reči (word)
  - ❖ s32 i2c\_smbus\_read\_word\_data(const struct i2c\_client \*client, u8 command);
  - ❖ s32 i2c\_smbus\_write\_word\_data(const struct i2c\_client \*client, u8 command, u16 value);
- ❖ Pisanje komandnog bajta i čitanje ili upis bloka podataka (najviše 32 bajta)
  - ❖ s32 i2c\_smbus\_read\_block\_data(const struct i2c\_client \*client, u8 command, u8 \*values);
  - ❖ s32 i2c\_smbus\_write\_block\_data(const struct i2c\_client \*client, u8 command, u8 length, const u8 \*values);
- ❖ Pisanje komandnog bajta i čitanje ili upis bloka podataka (bez ograničenja)
  - ❖ s32 i2c\_smbus\_read\_i2c\_block\_data(const struct i2c\_client \*client, u8 command, u8 length, u8 \*values);
  - ❖ s32 i2c\_smbus\_write\_i2c\_block\_data(const struct i2c\_client \*client, u8 command, u8 length, const u8 \*values);



# I2C funkcionalnost

- ❖ Ne podržavaju svi I2C kontroleri sve funkcionalnosti.
- ❖ Stoga, I2C rukovaoci kontrolera obaveste I2C jezgro koje funkcionalnosti podržavaju.
- ❖ I2C rukovaoci uređajem moraju da provere da li su funkcionalnosti koje su im potrebne obezbeđene od strane I2C kontrolera koji je u upotrebi na sistemu.
- ❖ `i2c_check_functionality()` funkcija dozvoljava takvu proveru.
- ❖ Primeri funkcionalnosti: `I2C_FUNC_I2C` za mogućnost korišćenja osnovnih I2C funkcija, `I2C_FUNC_SMBUS_BYTE_DATA` za mogućnost korišćenja SMBus komandi za upis komandi i čitanje/upis jednog bajta podataka.
- ❖ Pogledajte `include/uapi/linux/i2c.h` za potpun spisak postojećih funkcionalnosti.



# Reference

- ❖ <http://en.wikipedia.org/wiki/I2C>, uopštena prezentacija I2C protokola
- ❖ Documentation/i2c/, detalji o Linuks podršci za I2C
  - ❖ writing-clients, kako se pišu I2C rukovaoci uređajem
  - ❖ instantiating-devices, kako se instanciraju uređaji
  - ❖ smbus-protocol, detalji o SMBus funkcijama
  - ❖ functionality, kako mehanizam funkcionalnosti radi
  - ❖ I još mnogo drugih dokumenata
- ❖ <http://free-electrons.com/pub/video/2012/elce/elce-2012-anders-board-bringup-i2c.webm>, odlično predavanje: You, me and I2C - David Anders na ELCE 2012.



Uvod u pin muxing

# **LINUX UREĐAJ I MODEL RUKOVAOCA**

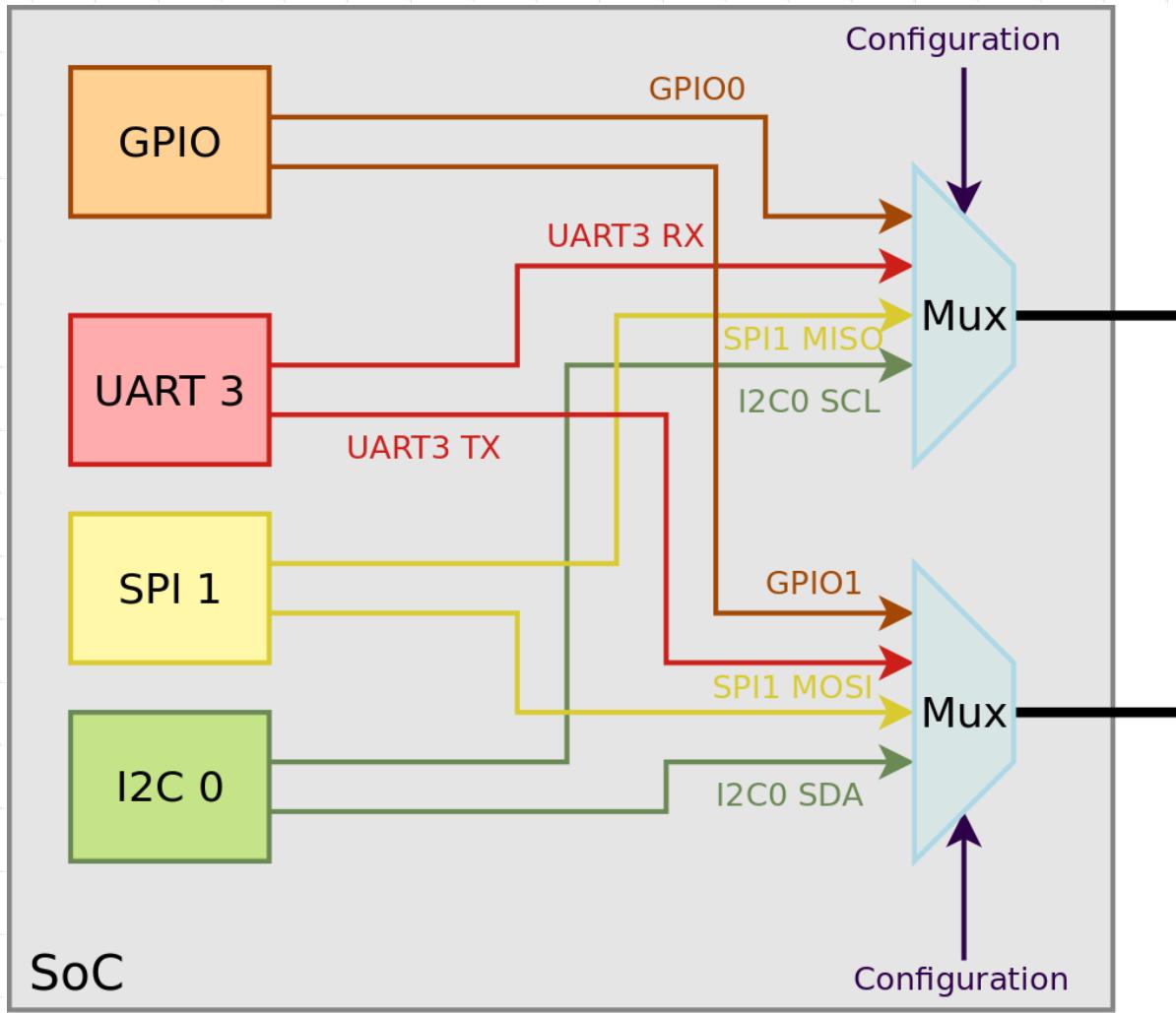


# Šta je to multipleksiranje pinova?



- ❖ Moderni SoC-ovi sadrže sve više i više blokova fizičke arhitekture, od kojih je mnogima potrebna sprega ka spoljnom svetu koju omogućuju **pinovi**.
- ❖ Međutim, fizička veličina ovih čipova ostaje mala, te je broj dostupnih pinova ograničen.
- ❖ Zbog ovog razloga nisu sve interne odlike blokova fizičke arhitekture dostupne istovremeno na pinovima.
- ❖ Ovi pinovi su **multipleksirani**: dozvoljavaju **ili** funkcionalnost bloka fizičke arhitekture A **ili** funkcionalnost bloka fizičke arhitekture B.
- ❖ **Multipleksiranje** je uglavnom konfigurabilno u programskoj podršci.

# Dijagram multipleksiranja pinova



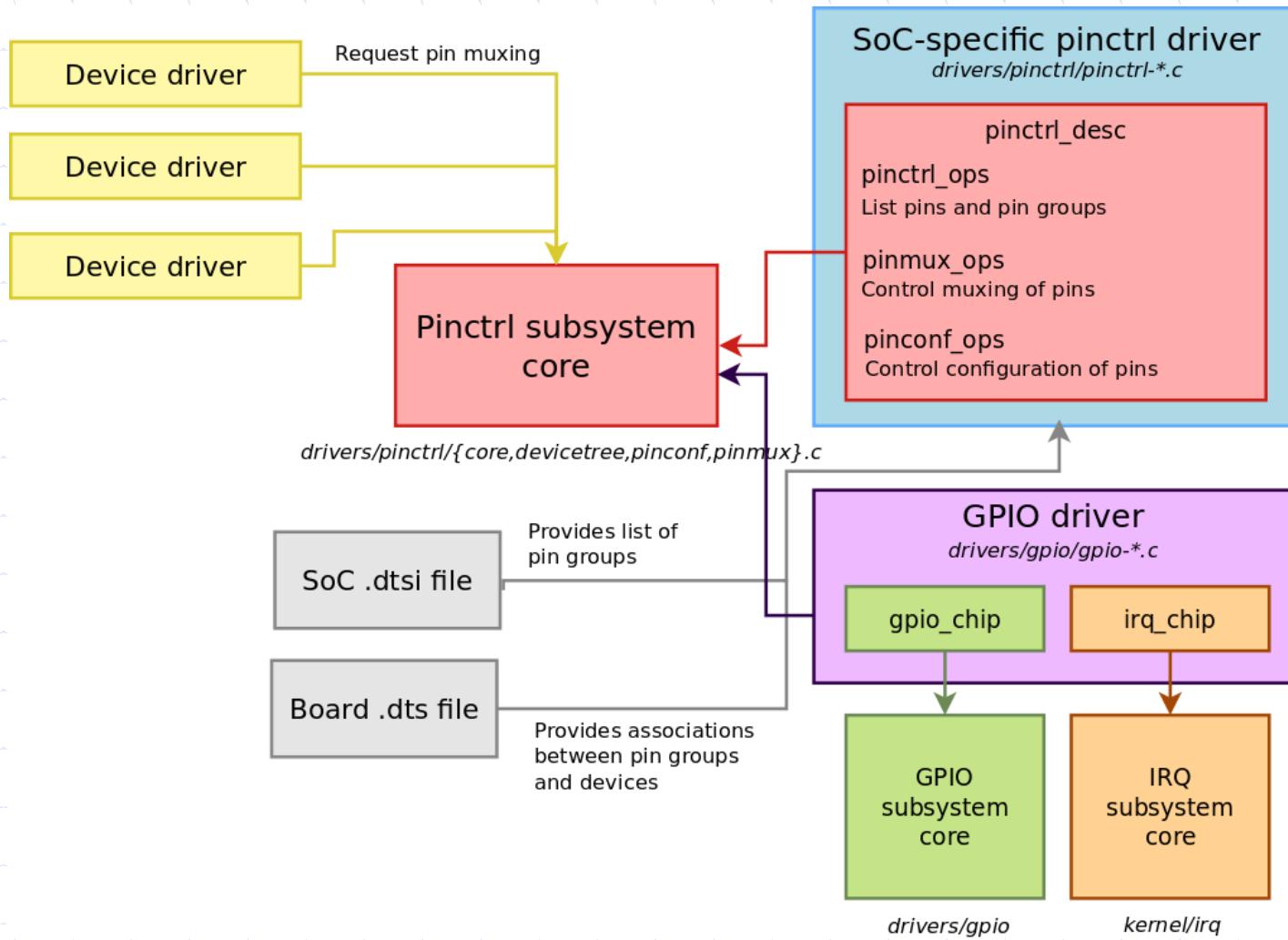


# Multipleksiranje pinova u Linuks kernelu



- ❖ U verziji Linuksa 3.2, dodat je **pinctrl** podsistem.
- ❖ Ovaj podsistem, koji se nalazi u **drivers/pinctrl** obezbeđuje generički podsistem za rukovanje multipleksiranjem pinova. Sadrži:
  - ❖ Spregu za multipleksiranje pinova za rukovaće, za implementaciju u SoC-specifičnim rukovaocima za konfiguraciju multipleksiranja.
  - ❖ Spregu za multipleksiranje pinova za potrošače, za rukovaće uređaja.
- ❖ Većina **pinctrl** rukovalaca obezbeđuje **vezu** stabla uređaja, gde multipleksiranje pinova mora biti opisano u stablu uređaja.
  - ❖ Tačna veza stabla uređaja zavisi od svakog rukovaoca. Svaka zasebna veza je opisana u **Documentation/devicetree/bindings/pinctrl**.

# Dijagram pinctrl podsistema





# Veza stabla uređaja za potrošačke uređaje (1/2)



- ❖ Uređaji koji zahtevaju da određeni pinovi budu multipleksirani će koristiti odlike stabla uređaja `pinctrl-<x>` i `pinctrl-names`.
- ❖ Odlike `pinctrl-0`, `pinctrl-1`, `pinctrl-<x>` se vezuju za konfiguraciju pina za dato stanje uređaja.
- ❖ Odlika `pinctrl-names` asocira ime sa pojedinačnim stanjem. Ime `default` je specijalno i automatski se odabira od strane rukovaoca uređajem, bez potrebe za eksplicitnim pozivom `pinctrl` funkcije.



## Veza stabla uređaja za potrošačke uređaje (2/2)

- ❖ U većini slučajeva, dovoljno je sledeće:

```
pmx_twsi0:pinctrl@0x10101010 {  
    ...  
};  
i2c@11000 {  
    pinctrl-0 = <&pmx_twsi0>;  
    pinctrl-1 = <&pmx_twsil>;  
    pinctrl-names = "default", "alternativa";  
    ...  
};
```

- ❖ Pogledajte  
<Documentation/devicetree/bindings/pinctrl/pinctrlbindings.txt> za detalje.



# Definisanje pinctrl konfiguracija

- ❖ Različite pinctrl configurations moraju biti definisane kao čvorovi potomci glavnog pinctrl uređaja (koji kontroliše multipleksiranje pinova).
- ❖ Konfiguracije mogu biti definisane na:
  - ❖ SoC nivou (.dtsi datoteka), za konfiguraciju pinova koji su često deljeni između više ploča
  - ❖ Nivou ploče (.dts datoteka) za konfiguracije koje su specifične za ploču.
- ❖ Odlika pinctrl-<x> potrošačkog uređaja pokazuje na konfiguraciju pina koja mu je potrebna kroz phandle stabla uređaja.
- ❖ Opis konfiguracija je specifičan za svaki pinctrl rukovalac. Videti u Documentation/devicetree/bindings/pinctrl za više dokumentacije o vezama stabla uređaja.



# Primer na OMAP/AM33xx

- ❖ Na OMAP/AM33xx, koristi se rukovalac **pinctrl-single**. Uobičajen je na više SoC-ova i dozvoljava konfigurisanje pinova pisanjem vrednosti u registar.
- ❖ U svakoj konfiguraciji pinova, vrednost **pinctrl-single,pins** daje listu (registrov, vrednost) parova potrebnih za konfiguraciju pinova.
- ❖ Da bi znali ispravne vrednosti, mora se koristiti dokumentacija SoC-a i ploče (datasheets).

```
am33xx_pinmux: pinmux@44e10800 {  
    i2c0_pins: pinmux_i2c0_pins {  
        pinctrl-single,pins = <  
            /* i2c0_sda.i2c0_sda */  
            0x188 (PIN_INPUT_PULLUP | MUX_MODE0)  
            /* i2c0_scl.i2c0_scl */  
            0x18c (PIN_INPUT_PULLUP | MUX_MODE0)  
        >;  
    };  
};  
  
i2c0: i2c@44e0b000 {  
    pinctrl-names = "default";  
    pinctrl-0 = <&i2c0_pins>;  
  
    status = "okay";  
    clock-frequency = <400000>;  
  
    tps: tps@2d {  
        reg = <0x2d>;  
    };  
};
```



# Primer na Allwinner SoC-u



SoC level

```
arch/arm/boot/dts/sun7i-a20.dtsi
/ {
    soc@01c00000 {
        pio: pinctrl@01c20800 {
            compatible = "allwinner,sun7i-a20-pinctrl";
            reg = <0x01c20800 0x400>;
            interrupts = <0 28 1>

            uart0_pins_a: uart0@0 { ←
                allwinner,pins = "PB22", "PB23";
                allwinner,function = "uart0";
                allwinner,drive = <0>;
                allwinner,pull = <0>;
            };
            ...
        };
    };
}
```

UART 0  
pin mux  
config

Board level

```
arch/arm/boot/dts/sun7i-a20-olinuxino-micro.dts
/ {
    soc@01c00000 {

        pio: pinctrl@01c20800 {
            LED
            pin mux
            config
            led_pins_olinuxino: led_pins@0 { ←
                allwinner,pins = "PH2";
                allwinner,function = "gpio_out";
                allwinner,drive = <1>;
                allwinner,pull = <0>;
            };
        };

        uart0: serial@01c28000 {
            pinctrl-names = "default";
            pinctrl-0 = <&uart0_pins_a>;
            status = "okay";
        };

        leds {
            Declare LED
            device and
            associate
            pin mux
            config
            compatible = "gpio-leds";
            pinctrl-names = "default";
            pinctrl-0 = <&led_pins_olinuxino>; ←
        };

        green {
            label = "a20-olinuxino-micro:green:usr";
            gpios = <&pio 7 2 0>;
            default-state = "on";
        };
    };
}
```

Enable UART0  
and associate  
pin mux  
config



# Univerzitet u Novom Sadu

Fakultet tehničkih nauka

Odsek za računarsku tehniku i  
računarske komunikacije



## Linuks u ugrađenim sistemima i razvoj rukovalaca

Radni okviri kernela za rukovaće  
uredaja



# Sadržaj



- ❖ Radni okviri kernela za rukovaće uređaja
  - ❖ Koncept radnih okvira kernla
  - ❖ Ulazni podsistem



Koncept radnih okvira kernla

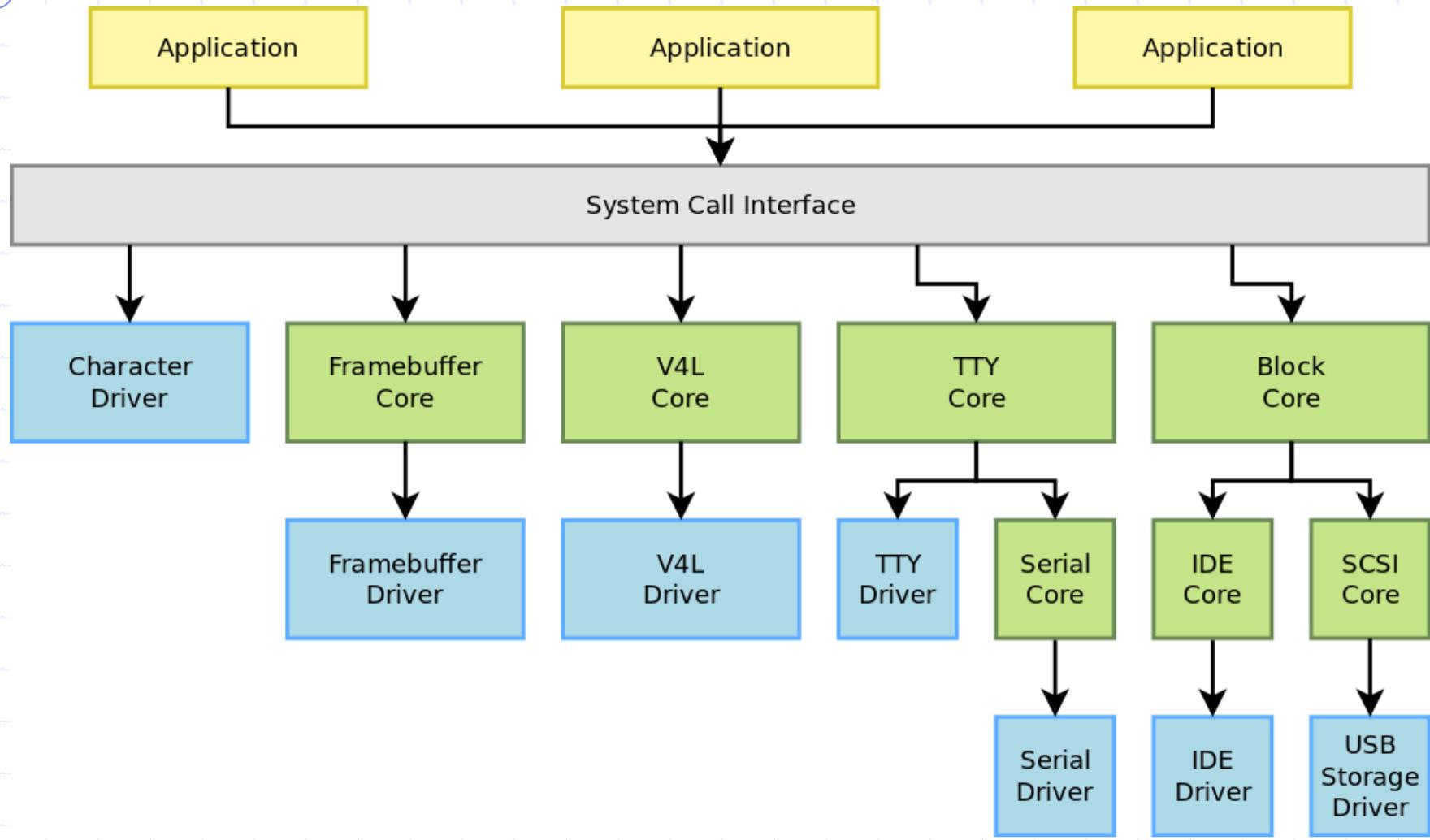
# **RADNI OKVIRI KERNELA ZA RUKOVAOCE UREĐAJA**



# Izvan slovnih rukovalaca: kernel frameworks

- ❖ Mnogi rukovaoci uređaja **nisu implementirani** kao karakterni rukovaoci
- ❖ Implementirani su pod **radnim okvirom**, specifičnim za dati tip uređaja (**framebuffer**, **V4L**, **serial**, itd)
  - ❖ Radni okvir dozvoljava razlaganje uobičajenih delova rukovalaca za isti tip uređaja
  - ❖ Iz korisničkog prostora, aplikacije ih i dalje vide kao karakterne uređaje
  - ❖ Radni okvir dozvoljava obezbeđivanje povezane sprege korisničkog prostora (**ioctl** i dr) za svaki tip uređaja, nebitno od korišćenog rukovaoca.

# Radni okviri kernela





# Primer: Framebuffer Radni okvir

- ❖ Opcija kernela CONFIG\_FB
  - ❖ menuconfig FB
    - ❖ tristate "Support for frame buffer devices"
  - ❖ Implementiran u C datotekama u **drivers/video/fbdev/core**
  - ❖ Implementira jedinstven karakterni rukovalac i definiše user/kernel API
    - ❖ Prvi deo **include/linux/fb.h**
    - ❖ Definiše set operacija koje framebuffer rukovalac mora da implementira i pomoćne funkcije za rukovaće
      - ❖ **struct fb\_ops**
      - ❖ Drugi deo **include/linux/fb.h**



# Operacije Framebuffer rukovaoca

- ❖ Navedene su operacije koje framebuffer rukovalac može ili mora da implementira i da ih definiše u **struct fb\_ops** strukturi

```
static struct fb_ops xxxfb_ops = {  
    .owner = THIS_MODULE,  
    .fb_open = xxxfb_open,  
    .fb_read = xxxfb_read,  
    .fb_write = xxxfb_write,  
    .fb_release = xxxfb_release,  
    .fb_check_var = xxxfb_check_var,  
    .fb_set_par = xxxfb_set_par,  
    .fb_setcolreg = xxxfb_setcolreg,  
    .fb_blank = xxxfb_blank,  
    .fb_pan_display = xxxfb_pan_display,  
    .fb_fillrect = xxxfb_fillrect,    /* Potreban!!! */  
    .fb_copyarea = xxxfb_copyarea,    /* Potreban!!! */  
    .fb_imageblit = xxxfb_imageblit, /* Potreban!!! */  
    .fb_cursor = xxxfb_cursor,        /* Opcioni !!! */  
    .fb_rotate = xxxfb_rotate,  
    .fb_sync = xxxfb_sync,  
    .fb_ioctl = xxxfb_ioctl,  
    .fb_mmap = xxxfb_mmap,  
};
```



# Kod Framebuffer rukovaoca

- ❖ U `probe()` funkciji, prijava framebuffer uređaja i operacija

```
static int xxxfb_probe (struct pci_dev *dev,
                       const struct pci_device_id *ent)
{
    struct fb_info *info;
    [...]
    info = framebuffer_alloc(sizeof(struct xxx_par), device);
    [...]
    info->fbops = &xxxfb_ops;
    [...]
    if (register_framebuffer(info) > 0)
        return -EINVAL;
    [...]
}
```

- ❖ `register_framebuffer()` kreira karakterni uređaj koji aplikacije korisničkog prostora mogu da koriste sa generičkim API-jem framebuffer-a



# Struktura podataka specifična za rukovaoca

- ❖ Svaki **radni okvir** definiše strukturu koju rukovalac uređaja mora da prijavi da bi bio prepoznat kao uređaj u ovom radnom okviru
  - ❖ **struct uart\_port** za serijske prolaze, **struct netdev** za mrežne uređaje, **struct fb\_info** za framebuffer-e, itd.
- ❖ Dodatno, rukovalac obično mora da čuva dodatne informacije o svom uređaju
- ❖ Ovo se uobičajeno radi
  - ❖ Postavljanjem podklase odgovarajuće strukture radnog okvira
  - ❖ Čuvanjem reference na odgovarajuću strukturu radnog okvira
  - ❖ Ili uključivanjem naših informacija u strukturu radnog okvira



# Primeri struktura podataka specifičnih za rukovače (1/2)



- ❖ i.MX serijski rukovalac: struct imx\_port je podklasa struct uart\_port

```
struct imx_port {  
    struct uart_port port;  
    struct timer_list timer;  
    unsigned int old_status;  
    int txirq, rxirq, rtsirq;  
    unsigned int have_rtscts:1;  
    [...]  
};
```

- ❖ ds1305 RTC rukovalac: struct ds1305 sadrži referencu na struct rtc\_device

```
struct ds1305 {  
    struct spi_device *spi;  
    struct rtc_device *rtc;  
    [...]  
};
```



# Primeri struktura podataka specifičnih za rukovače (2/2)



- ❖ **rtl8150 mrežni rukovalac:** struct rtl8150 sadrži referencu na struct net\_device i alociran je unutar strukture radnog okvira.

```
struct rtl8150 {  
    unsigned long flags;  
    struct usb_device *udev;  
    struct tasklet_struct tl;  
    struct net_device *netdev;  
    [...]  
};
```



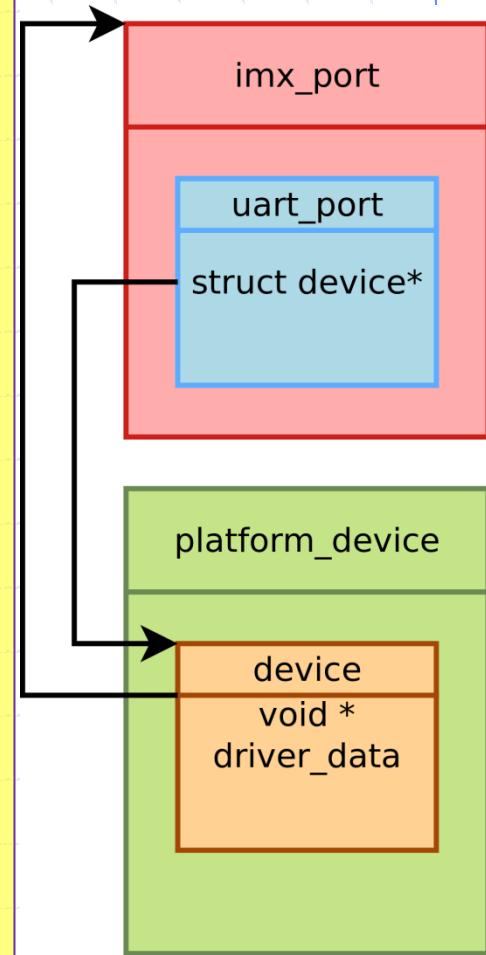
## Veza između struktura (1/4)

- ❖ Radni okvir obično sadrži **struct device \*** pokazivač gde rukovalac mora da pokazuje na odgovarajući **struct device**
  - ❖ To je veza između logičkog uređaja (na primer mrežna sprega) i fizičkog uređaja (na primer USB mrežni adapter)
- ❖ Struktura uređaja takođe sadrži **void \*** pokazivač koji rukovalac može slobodno da koristi.
  - ❖ Često se koristi da se uveže nazad uređaj ka strukturi višeg nivoa iz radnog okvira.
  - ❖ Na primer, to dozvoljava da se iz **struct platform\_device** strukture pronađe struktura koja opisuje logički uređaj.

# Veza između struktura (2/4)

```
static int serial_imx_probe(struct platform_device *pdev)
{
    struct imx_port *sport;
    [...]
    /* setup the link between uart_port and the struct
     * device inside the platform_device */
    sport->port.dev = &pdev->dev;
    [...]
    /* setup the link between the struct device inside
     * the platform device to the imx_port structure */
    platform_set_drvdata(pdev, sport);
    [...]
    uart_add_one_port(&imx_reg, &sport->port);
}

static int serial_imx_remove(struct platform_device *pdev)
{
    /* retrieve the imx_port from the platform_device */
    struct imx_port *sport = platform_get_drvdata(pdev);
    [...]
    uart_remove_one_port(&imx_reg, &sport->port);
    [...]
}
```

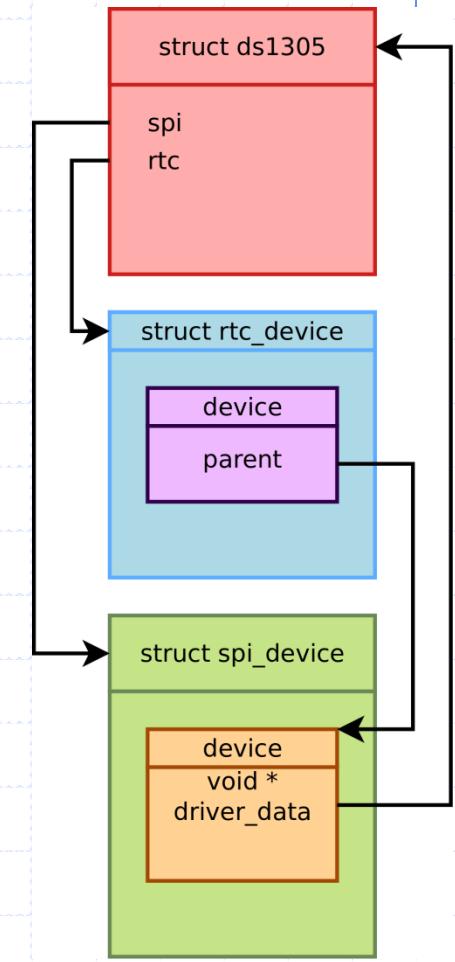


# Veza između struktura (3/4)

```

static int ds1305_probe(struct spi_device *spi)
{
    struct ds1305 *ds1305;
    [...]
    /* set up driver data */
    ds1305 = devm_kzalloc(&spi->dev, sizeof(*ds1305),
                         GFP_KERNEL);
    if (!ds1305)
        return -ENOMEM;
    ds1305->spi = spi;
    spi_set_drvdata(spi, ds1305);
    [...]
    /* register RTC ... from here on, ds1305->ctrl needs
     * locking */
    ds1305->rtc = devm_rtc_device_register(&spi->dev,
                                            "ds1305", &ds1305_ops, THIS_MODULE);
    [...]
}
static int ds1305_remove(struct spi_device *spi)
{
    struct ds1305 *ds1305 = spi_get_drvdata(spi);
    [...]
}

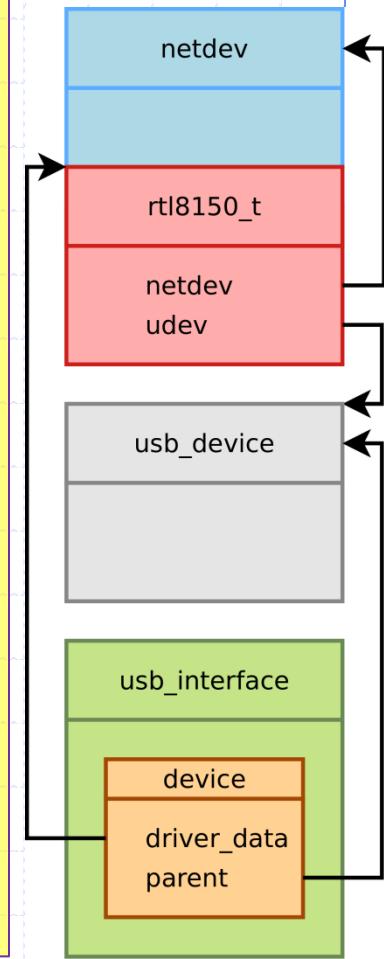
```





## Veza između struktura (4/4)

```
static int rtl8150_probe(struct usb_interface *intf,
                         const struct usb_device_id *id)
{
    struct usb_device *udev = interface_to_usbdev(intf);
    rtl8150_t *dev;
    struct net_device *netdev;
    netdev = alloc_etherdev(sizeof(rtl8150_t));
    dev = netdev_priv(netdev);
    [...]
    dev->udev = udev;
    dev->netdev = netdev;
    [...]
    usb_set_intfdata(intf, dev);
    SET_NETDEV_DEV(netdev, &intf->dev);
    [...]
}
static void rtl8150_disconnect(struct usb_interface *intf)
{
    rtl8150_t *dev = usb_get_intfdata(intf);
    [...]
}
```





Ulagni podsistem

# RADNI OKVIRI KERNELA ZA RUKOVAOCE UREĐAJA

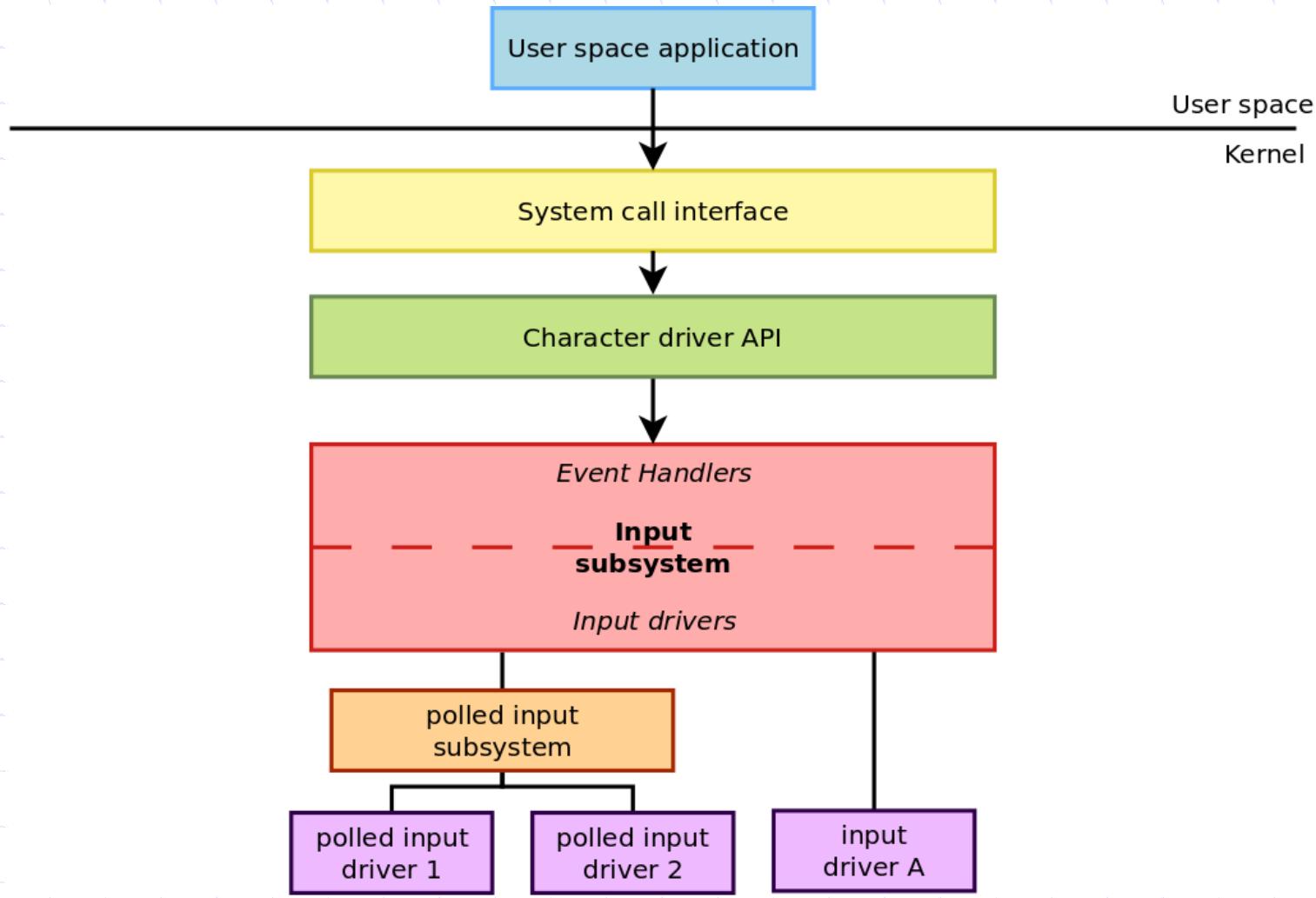


# Šta je ulazni podsistem?



- ❖ Ulagni podsistem se stara o svim ulaznim događajima koji dolaze od korisnika.
- ❖ Inicijalno je napisan za podršku USB HID (engl. Human Interface Device) uređaje, a brzo je preuzeo rukovanje svim vrstama ulaza (nezavisno od toga da li koriste USB): tastature, miševi, džojstici, ekrani osjetljivi na dodir, itd.
- ❖ Ulagni podsistem je podeljen na dva dela:
  - ❖ **Rukovaoci uređaja:** komuniciraju sa fizičkom arhitekturom (na primer preko USB-a) i prosleđuju događaje (pomeraji miša, pritisci na tastere, koordinate dodira) ulaznom jezgru
  - ❖ **Obrađivači događaja:** prihvataju događaje od rukovalaca i prosleđuju ih gde su potrebni preko raznih sprega (najčešće preko **evdev-a**)
- ❖ U korisničkom prostoru najčešće se koriste od strane nekog grafičkog steka poput **X.Org**, **Wayland** ili **Androidov InputManager**.

# Dijagram ulaznog podsistema





# Pregled ulaznog podsistema (1/2)

- ❖ Opcija kernela **CONFIG\_INPUT**
  - ❖ menuconfig INPUT
    - ❖ tristate "Generic input layer (needed for keyboard, mouse, ...)"
- ❖ Implementiran u **drivers/input/**
  - ❖ **input.c, input-polldev.c, evbug.c**



## Pregled ulaznog podsistema (2/2)

- ❖ Implementira jedinstven karakterni rukovalac i definiše user/kernel API
  - ❖ `include/uapi/linux/input.h`
- ❖ Definiše set operacija koje ulazni rukovalac mora da implementira i pomoćne funkcije za rukovače
  - ❖ `struct input_dev` za deo rukovaoca uređajem
  - ❖ `struct input_handler` za deo obradivača događaja
  - ❖ `include/linux/input.h`



# API ulaznog podsistema (1/4)

- ❖ Ulazni uređaj je opisan sa veoma velikom **struct input\_dev** strukturom, i sečak:

```
struct input_dev {  
    const char *name;  
    [...]  
    unsigned long evbit[BITS_TO_LONGS(EV_CNT)];  
    unsigned long keybit[BITS_TO_LONGS(KEY_CNT)];  
    [...]  
    int (*getkeycode)(struct input_dev *dev,  
                      struct input_keymap_entry *ke);  
    [...]  
    int (*open)(struct input_dev *dev);  
    [...]  
    int (*event)(struct input_dev *dev, unsigned int type,  
                 unsigned int code, int value);  
    [...]  
};
```

- ❖ Pre nego što se koristi, ova struktura mora biti alocirana i inicijalizovana:  
**struct input\_dev \*input\_allocate\_device(void);**
- ❖ Nakon uklanjanja **struct input\_dev**, mora biti oslobođena sa: **void  
input\_free\_device(struct input\_dev \*dev);**



## API ulaznog podsistema (2/4)

- ❖ U zavisnosti od tipa događaja koji se generiše, ulazna bit polja **evbit** i **keybit** moraju biti podešena: na primer, za dugme se generiše samo **EV\_KEY** tip događaja i od njih samo **BTN\_0** kod događaja:

```
set_bit(EV_KEY, myinput_dev.evbit);
set_bit(BTN_0, myinput_dev.keybit);
```

- ❖ **set\_bit()** je atomska operacija koja postavlja određeni bit na 1 (kasnije objašnjeno).
- ❖ Kada se **ulazni uređaj** alocira i popuni, poziva se funkcija za prijavljivanje:  
**int input\_register\_device(struct input\_dev \*);**
- ❖ Kada se ukolni rukovalac, **ulazni uređaj** se uklanja korišćenjem:  
**void input\_unregister\_device(struct input\_dev \*);**



## API ulaznog podsistema (3/4)

- ❖ Događaji se šalju od rukovaoca ka obrađivaču događaja korišćenjem `input_event(struct input_dev *dev, unsigned int type, unsigned int code, int value);`
- ❖ Tipovi događaja su dokumentovani u:  
`Documentation/input/event-codes.txt`



## API ulaznog podsistema (4/4)

- ❖ Događaj se sastoji od jedne ili više **ulaznih promena podataka** (paketi ulaznih promena podataka) poput stanja dugmeta, relativne ili absolute pozicije na nekoj osi, itd.
- ❖ Nakon slanja potencijalno višestrukih događaja, **ulazno jezgro** mora biti notifikovano pozivom funkcije: **void input\_sync(struct input\_dev \*dev):**
- ❖ Ulazni podsistem obezbeđuje omotače (wrappers) kao što su **input\_report\_key()**, **input\_report\_abs()**, ...



# Polovana ulazna podklasa

- ❖ Ulagni podsistem obezbeđuje podklasu koja podržava jednostavne ulazne uređaje koji ne podižu prekide, već moraju da se periodično skeniraju ili poluju da bi se primetile izmene u njihovom stanju.
- ❖ Polovani ulazni uređaj je opisan strukturom **struct input\_polled\_dev**:

```
struct input_polled_dev {  
    void *private;  
    void (*open)(struct input_polled_dev *dev);  
    void (*close)(struct input_polled_dev *dev);  
    void (*poll)(struct input_polled_dev *dev);  
    unsigned int poll_interval; /* msec */  
    unsigned int poll_interval_max; /* msec */  
    unsigned int poll_interval_min; /* msec */  
    struct input_dev *input;  
/* private: */  
    struct delayed_work work;  
}
```



# API polovanog ulaznog podsistema



- ❖ Alokacija/oslobađanje strukture `struct input_polled_dev` se radi korišćenjem:
  - ❖ `input_allocate_polled_device()`
  - ❖ `input_free_polled_device()`
- ❖ Samo `poll()` metoda je obavezna za strukturu `struct input_polled_dev`, ova funkcija poluje uređaj i šalje ulazne događaje.
- ❖ Polja `id`, `name`, `evkey` i `keybit` od ulaznih polja moraju biti inicijalizovana.
- ❖ Ako nijedno od `poll_interval` polja nije popunjeno, onda je podrazumevani `poll_interval` 500ms.
- ❖ Prijavljivanje/uklanjanje uređaja se radi sa:
  - ❖ `input_register_polled_device(struct input_polled_dev *dev)`.
  - ❖ `input_unregister_polled_device(struct input_polled_dev *dev)`



# evdev sprega korisničkog prostora



- ❖ Glavna sprega korisničkog prostora za ulazne uređaje je **sprega događaja**
- ❖ Svaki ulazni uređaj se predstavlja sa **/dev/input/event<X>** karakternim uređajem
- ❖ Aplikacija korisničkog prostora može da koristi blokirajuća i neblokirajuća čitanja, ali takođe i **select()** (da bude obaveštena o događajima) nakon otvaranja uređaja.
- ❖ Svako čitanje vraća strukturu **struct input\_event** sledećeg formata:

```
struct input_event {  
    struct timeval time;  
    unsigned short type;  
    unsigned short code;  
    unsigned int value;  
};
```

- ❖ Veoma korisna aplikacija za testiranje ulaznih uređaja je **evtest**, sa <http://cgit.freedesktop.org/evtest/>