PRIRODNO-MATEMATIČKI FAKULTET

Izgradnja energetski efikasnog ugrađenog računarskog sistema zasnovanog na ARM arhitekturi

DIPLOMSKI RAD

Mentor: dr Vladimir Filipović
Student: Nikola Nenadić
Broj indeksa: 26/09
Banja Luka, januar 2016.

Sadržaj

Uvod

U posljednje vrijeme na tržištu viđamo mnogo Small Business Server-a i Network Attached Storage uređaja. Potreba za ovakvom vrstom opreme proizilazi iz činjenice da je danas tehnologija postala masovno dostupna i da se koristi u skoro svim sferama života. Ipak, tehnologija dolazi po određenoj cijeni. Velike kompanije imaju mogućnosti i resurse da unaprijede svoje poslovanje i relativno lako uvedu informacioni sistem. Međutim, male i srednje kompanije moraju da se oslanjaju na jeftinija rješenja kako bi unaprijedili svoje poslovanje. To je razlog zbog kojeg su mali biznis server-i traženi na tržištu. Tržište zahtijeva proizvod koji je lako konfigurisati, proizvod koji ne troši mnogo električne energije i, ono što je najbitnije, proizvod koji je stabilan i brz u radu.

Cilj ovog diplomskog rada je prikazati proces izgradnje malog biznis servera za višestruku upotrebu baziranog na ARM arhitekturi.

Ono što čini ovaj proizvod različitim od ostalih leži u pažljivo izabranim tehnologijama i hardveru koji će obezbijediti maksimalnu brzinu po niskoj cijeni.

Postupno će biti objašnjena implementacija GNU/Linux orjentisanog sistema, kompajliranje i konfigurisanje bootloader-a, konfiguracija i kros kompajliranje kernel-a, implementacija minimalnog root-file sistema, izgradnja web interfejsa, podešavanje servisa, sastavljanje kućišta i kompletne hardverske konfiguracije.

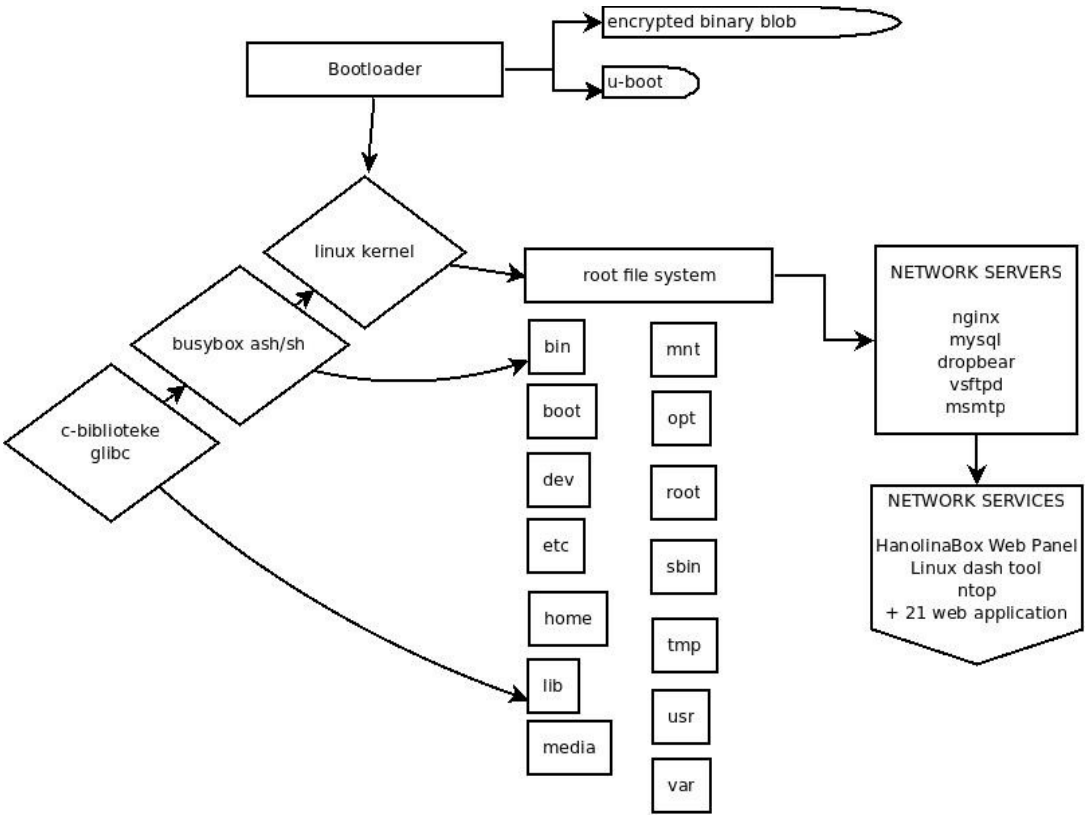
Planiranje izrade ugrađenog sistema baziranog na GNU/Linux-u.

Pri planiranju izgradnje Embedded GNU/Linux distribucije posao dijelimo u 4 etape:

• Hardware
• Bootloader
• Operativni sistem
• Root-file sistem - korisničke aplikacije

Pri razvoju sistema etape se mogu pojedinačno obrađivati. Posljedica toga je mogućnost da se svaka etapa može razvijati nezavisno jedna od druge.

Arhitektura projekta realizovanog u svrhu diplomskog rada:



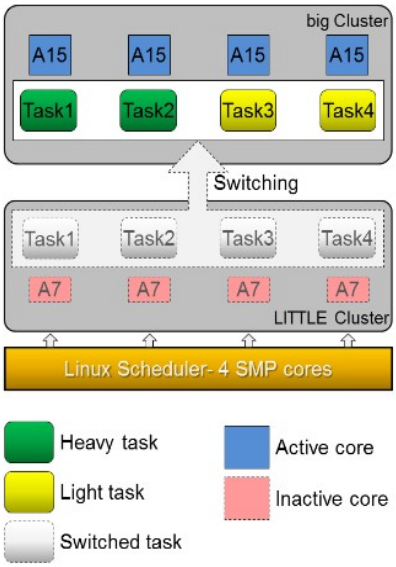
Hardware

1. Razvojna ploča

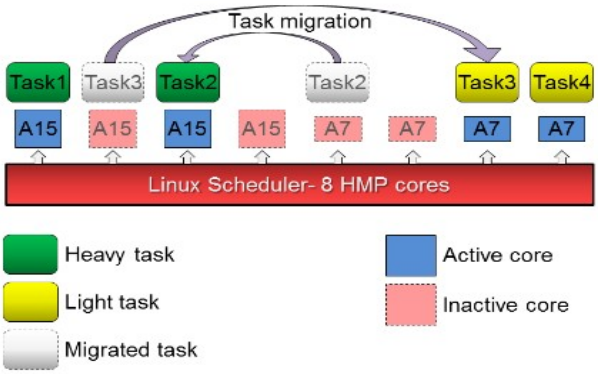
Hardware-sku osnovu sistema čini Odroid XU4 razvojna ploča. Iako malih gabarita, ova razvojna ploča posjeduje veliku procesorsku snagu. Ploča je zasnovana na porodici procesora Samsung Exynos 5422 Cortex sa podrškom za heterogeni multi procesing (HMP), kako bi se u isto vrijeme riješio problem energetske efikasnosti i nedostatka performansi u mobilnim uređajima. Samsung predstavlja novi model konstrukcije procesora nazvan big.LITTLE arhitektura.
Odroid XU4 opremljen je sa 4 velika jezgre ARM Cortex A-15 (maks.frek. 2.0 GHz) jezgre koje su zadužene za obavljanje kompleksnih operacija i 4 male jezgre ARM Cortex A7 (maks.frek 1.4GHz) koje su energetski efikasne. Ideja je da operativni sistem skaliranjem težine i kompleksnosti procesa koji dolaze na izvršavanje odluči na kojoj jezgri će zadatak biti izvršen. Procesi koji zahtijevaju malo resursa i često izvršavanje pokrenuti su na energetski efikasnim jezgrama, a procesi sa većim zahtjevima i kraćim periodom izvršavanja odlaze na velike jezgre.

big.LITTLE arhitektura ima dva načina rada (moda) na koji može da upravlja sa procesima:

- big.LITTLE Cluster Switching mode: jezgre rade u dvije grupe (dva cluster-a); u prvoj grupi nalaze se velike jezgre (cluster 1), a u drugoj grupi male jezgre (cluster 2). U trenutku kada su male jezgre dosegnule svoj maksimum i cluster 2 postao previše opterećen, dešava se Cluster switching tj. svi zadaci koji su bili na izvršavanju u cluster-u 2 prebacuju se na velike jezgre u cluster-u 1, kako bi se zadatak obavio efikasnije. Kernel posjeduje modul za kontrolu frekvencije jezgri. Samim zahtjevom zadatka za većom frekvencijom ili dosezanjem maksimalne frekvencije jezgre proces daje jasan signal da mu je potrebno više procesorske moći. Maksimalne ostvarive performanse u ovom načinu rada predstavljaju zbir procesorske moći velikih jezgri.



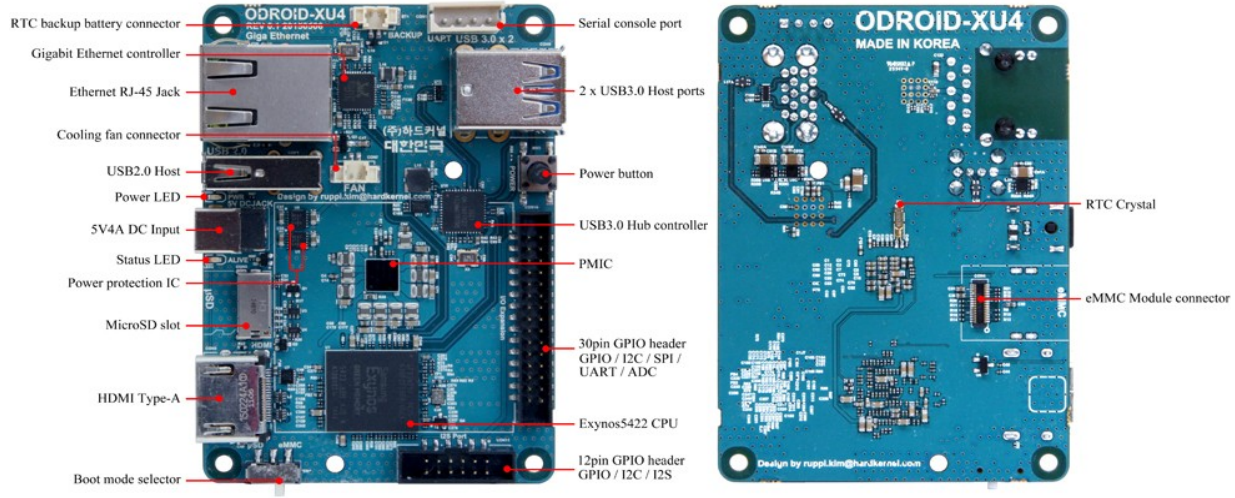
- big.LITTLE HMP Mode: zadatak se šalje na jezgru koja odgovara težini zadatka, pojam cluster-a nestaje i uvodi se heuristički pristup raspodjeli zadataka. Scheduler, poređenjem zahtjeva za resursima koji procesi upućuju subsystem-ima u kernel-u, razvrstava zadatke na "teške i lake" i na osnovu povratih informacija tačno zna koji zadatak treba da se izvršava na kojoj jezgri. Jezgra koja je neaktivna gasi se i čeka sljedeći zadatak. Kako se rad malih i velikih jezgri ne bi preklapao uvedena su dodatna pravila. Ako je za neki zadatak potrebna niska frekvencija i



malo resursa, male jezgre imaju mogućnost rada na niskim frekvencijama dok velike u HMP modu nemaju. Takođe, ovaj način rada je korišten da izvučemo maksimum iz hardware-a jer sve jezgre mogu da rade u isto vrijeme na maksimalnim frekvencijama. To dovodi do činjenice da maksimalne performanse u HMP načinu rada predstavljaju zbir procesorske moći svih jezgri.

Specifikacija razvojne ploče

Procesor	Samsung Exynos-5422 Cortex-A15 i Cortex-A7 big.LITTLE procesor sa 2Gb LPDDR3 RAM-a
Mrežni interfejs	Realtek RTL8153 10/100/1000M - gigabitni mrežni interfejs
USB 3.0 kontroler	Genesys GL3521 je dvoportni, energetski efikasan i konfigurabilan USB 3.0 SuperSpeed hub kontroler
Zaštita od preopterećenja ploče izazvana ulaznom strujom i naponom	USB Load Switch - NCP380 pruža zaštitu od pregrijavanja, preopterećena hub-a i padova napona na USB hub-u. Korištenjem MOSFET tehnologije kreirani su uređaji koji su pouzdaniji i bliži idealnim karakteristika za razliku od bipolarnih tranzistora. NCP372 zaštita od preopterećenja na razvojnoj ploči, kontroliše ulaznu struju i napon
HDMI interfejs	Standardni Type-A HDMI, podrška za 1920 x 1080 rezoluciju
PMIC	Samsung S2MPS11 regulator koji se brine o naponu i struji unutar razvojne ploče. U suštini predstavlja DC to DC konvertor kako bi se riješio problem različitih napona unutar razvojne ploče. Različiti naponi se javljaju usljed velikog broja interfejsa koje ploča posjeduje, svaki od interfejsa zahtijeva drugi napon. Naponi se kreću u rasponu od 1.8V do 5V unutar Odroid razvojne ploče
Ulazno/izlazni interfejsi	USB 3.0 x 2, USB 2.0 x 1, PWM za kontrolu kulera, UART priključak za serijsku konzolu 30 Pinova : GPIO/IRQ/SPI/ADC, 12 Pinova : GPIO/I2S/I2C
LED indikator	Plava LED dioda indikator aktivnosti operativnog sistema
Memorijski slot	Micro-SD slot eMMC 5.0 modul konektor
DC ulaz	5V / 4A input



Razvojna ploča odroid XU4 posjeduje opciju boot-ovanja sa MicroSD (class 10 i uhs1) ili eMMC kartica. Korisnik može da odluči koji od ova dva načina boot-a će koristiti. Na razvojnoj ploči postoji prekidač kojim se podešava sa kog medija da se izvrši proces boot-ovanje.

2. Memorija na koju je pohranjen operativni sistem:

eMMC (embedded Multi-Media Controller) je NAND bazirana memorija koja donosi tehnologiju korištenu u izradi SSD diskova u svijet portabilnih uređaja.

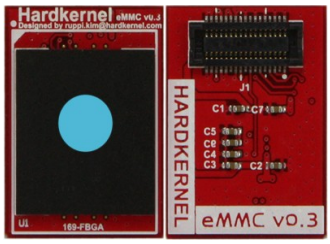
eMMC memorija se sastoji od tri komponente:

1. memorijskog interfejsa tj. vrste priključka preko kog se odvija komunikacija,
2. memorijskog prostora ili fleš memorije,
3. i memorijskog kontrolera koji se brine o adresnom prostoru i integritetu podataka.

Implementacija memorijskog kontrolera na eMMC memoriji oslobađa kernel od low-level operacija koje su mu inače potrebne za upravljanje memorijom. Kontroler sam brine o rasporedu adresa i strukturi podataka u memoriji. Ovaj pristup donosi mnogo poboljšanja u svijet embedded uređaja jer programeri nemaju potrebu da za svaki novi tip NAND memorije razvijaju driver (što je do sada bio slučaj), već može da poziva kontroler na samoj memoriji i prepusti mu organizaciju podataka. Bitno je napomenuti da eMMC daje gotovo jednake performanse kao SSD diskovi i u posljednje vrijeme često je izbor embedded programera kao medij na kom će biti pohranjena boot particija.

U nastavku biće prikazan test brzine kojim se poredi MicroSD kartica sa eMMC karticom.

Poređenje čitanja/pisanja MicroSD i eMMC kartice



Write command
dd if=/dev/zero of=test oflag=direct bs=8M count=64

Read command
dd if=test of=/dev/null iflag=direct bs=8M

Performanse memorijskih medija	SD-uhs1	eMMC 5.0
Brzina upisa podataka (Mb/s)	9.81	39.3
Brzina čitanja podataka (Mb/s)	32.1	154

3. Mrežni interfejs

Server posjeduje dva načina za pristup mreži:

- gigabitni ethernet interfejs,
- i wifi modul sa antenom (The Realtek RTL8188CUS-GR).

Mrežne uređaje treba testirati, kako bismo vidjeli koliko su pouzdani i da bismo im izmjerili brzinu. Brz protok podataka kroz mrežni interfejs je jedna od glavnih osobina koju bi trebao posjedovati server.

Testiranje mreže

Test komadna
Server mode : iperf -s
Client Mode : iperf -c [ip address] -P 10 -W 32k

Performanse mrežnog interfejsa	
Server mod (Mbit/sec)	869.0
Klijent mod (Mbit/sec)	885.0

4. Kućište

Uređaj je smješten u kutiju proizvođača CloudShell.
Kutija posjeduje externi TFT display koji komunicira preko SPI bus-a sa razvojnom pločom. Na display se ispisuju informacije o stanju sistema.
U CloudShell-u postoji mjesto za smještanje jednog SSD diska od 2.5inch. Tvrdi disk se koristi kao data storage uređaja.

USB3.0 to SATA bridge

Razvojna ploča posjeduje dva USB 3.0 SuperSpeed interfejsa.
USB 3.0 port je korišten za spajanje SSD diska sa razvojnom pločom, potrebno je izmjeriti performanse i vidjeti rezultate.

Testiranje SSD diska

Write command
dd if=/dev/zero of=test oflag=direct bs=1M count=64

Read command
dd if=test of=/dev/null iflag=direct bs=1M

Performanse USB interfejsa	Čitanje sa SSD-a (Mb/sek)	Pisanje na SSD (Mb/sek)
USB3.0 SuperSpeed	148.0	132

Desktop razvojno okruženje

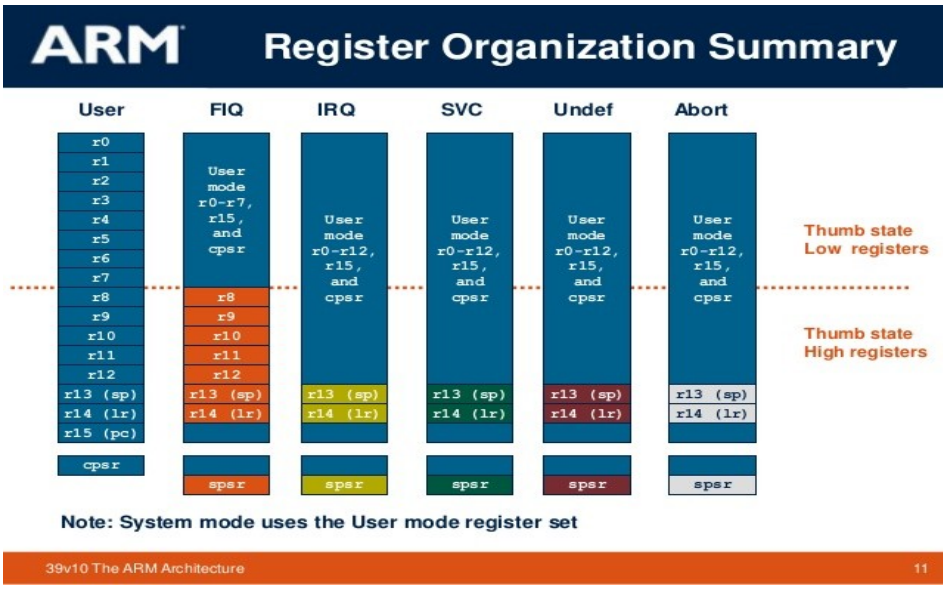
Za desktop razvojno okruženje izabrana je GNU/Linux distribucija Linux Mint 17.1 Rebbeca (linux kernel 3.13). GNU/Linux Mint i ostale tehnologije korištene za razvoj pripadaju slobodnim open-source tehnologijama koje su dostupne svima na internetu.

ARM arhitektura

ARM je 32-bitna arhitektura skupa instrukcija za procesore bazirane na RISC arhitekturi. RISC-pristup izrade računarskog dizajna znači da ARM procesori zahtijevaju mnogo manje tranzistora od tipičnih CISC procesora. Ovaj princip smanjuje cijenu, zagrijavanje i potrošnju energije. Ovo su poželjne osobine prenosnih uređaja koji rade na bateriju. Jednostavan dizajn omogućava efikasniju implementaciju multi-core procesora i jeftiniju cijenu izrade.

ARM procesori posjeduju 37 registara (mada ih često ima i više):

- 31 registar za opštu upotrebu, uključujući i program counter (PC),
- 6 statusnih registara.



ARM arhitektura posijeduje tri režima rada:

- ARM - Procesor izvršava 32-bitne ARM komande sa poravnanjem.
- Thumb 2 - U ovom režimu se izvršavaju 32-bitne i 16-bitne komande . Za zapis komande dovoljna je jedna poluriječ (halfword). Korisniku je dostupno manje registara u ovom režimu rada.
- ThumbEE - ovaj režim se najčešće koristi u svrhu hardverske virtuelizacije i optimizaciju programskih jezika visokog nivoa apstrakcije sa Just-in-Time kompajlerima kao što su Java, C# i Python.

Režim rada je definisan u registru CPSR. Upisivanjem T i J bitova mijenja se režim rada.
Kada se mijenja režim rada, treba imati u vidu da promjena režima utiče na stanje određenih registara.

Unbanked registri, od R0 do R7 - predstavljaju registre koji u svim režimima rada imaju istu fizičku reprezentaciju u hardveru. Korištenje ovih registara iz bilo kog režima rada se smatra sigurnim.

Banked registri, od R8 do R14 - stanje ovih registara je zavisno od režima rada procesora.

Promjena režima rada neće uticati na procesorske modove koji će biti spomenuti u nastavku.

ARM arhitektura podržava dva načina kodiranja memorije:

- big-endian format
- little-endian format

Procesorski modovi

Procesori bazirani na ARM arhitekturi posjeduju više modova rada. Modovi mogu biti promjenjeni ručno (programski) ili promjena moda može biti inicirana prekidom (interrupt). Većina aplikacija izvršava se u korisničkom modu (User mode). Kada se procesor nalazi u korisničkom modu, aplikacija niju u stanju da pristupi svim sistemskim resursima, ograničen joj je pristup. Mogućnost ARM arhitekture da ograničava pristup resursima u određenim modovima je preduslov za izgradnju modernog operativnog sistema.

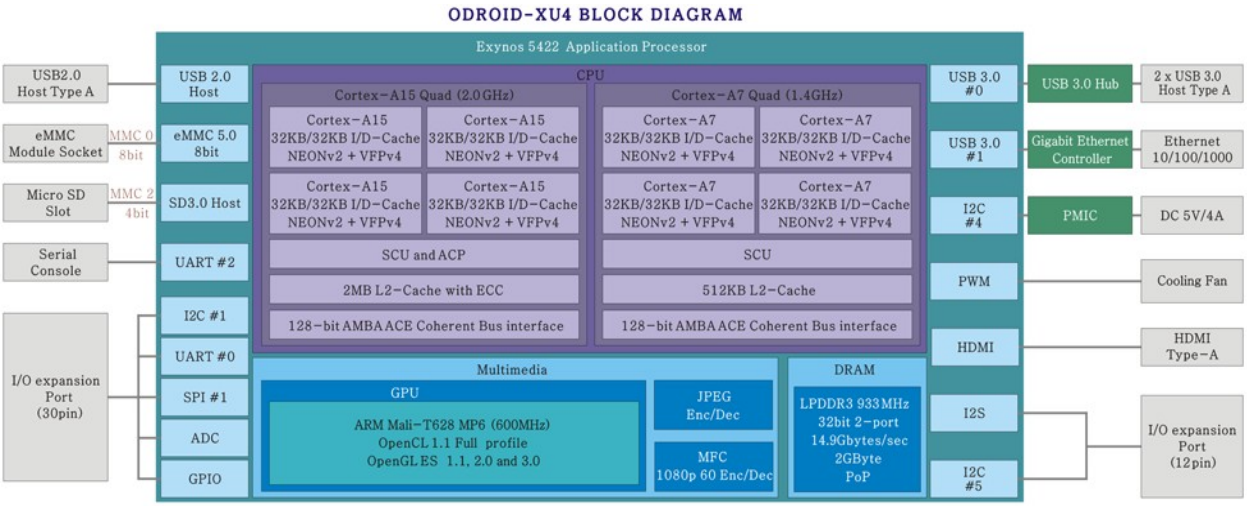
Procesorski mod		Broj moda	Opis
User	usr	0b10000	Korisnički mod
FIQ	fiq	0b10001	Podrška za high-speed data
IRQ	irq	0b10010	Kontrola prekida (interrupt handling)
Supervisor	svc	0b10011	Protected mode - korišten od strane operativnog sistema
Abort	abt	0b10111	Koristi se pri implementaciji virtuelne memorije i memorijskih barijera
Undefined	und	0b11011	Podrška za softversku emulaciju hardverskih procesora - virtuelizacija
System	szs	0b11111	Koristi se pri pokretanju privilegovanih sistemskih poziva.

Svi procesorski modovi osim korisničkog moda, spadaju i privilegovane modove. Privilegovani modovi imaju mogućnost pristupa svim sistemskim resursima. Svi privilegovani modovi smatraju se modovima izuzetka (exception mode).

Exynos 5422

U diplomskom radu korišten je SoC (System-on-Chip) Exynos 5422, gdje se pored procesora, u čipu nalaze sve ostale funkcije sistema. Jedan čip sadrži ram memoriju, gpu memoriju, podsisteme za eksterne uređaje - usb, i2c, spi,serijska komunikacija, HDMI, eMMC Module Socket...
Na SoC-u se nalaze dva procesora Cortex-A15 i Cortex-A7 koji dijele 2Gb radne memorije. Ova dva procesora su zasnovana na ARM arhitekturi.

Prikaz arhitekture čipa:



Cortex-A15 procesori koriste 40-bitne adrese za mapiranje memorije, i tako postaju jedna od rijetkih arhitektura koja se pomjera od klasičnog 32-bitnog mapiranja. Time dobijaju mogućnost korištenja do 1TB radne memorije. Pomenuti procesori koriste Thumb2 mod koji smanjuje set instrukcija potrebnih programu, bez većeg gubitka performansi.

Procesori posjeduju po jezgri, jednu L1 keš memoriju od 32kb za podatke i 32 kb za smještanje instrukcija.

Takođe procesor Cortex A-15 posjeduje 2Mb L2 keš memorije, dok procesor Cortex-A9 posjeduje 512kb L2 keš memorije.

Procesori iz serije Cortex-A15 i Cortex-A9 posjeduju 4 koherentne jezge. Pomoću big.LiTTLE arhitekture moguće je organizovati procesore u jedan heterogeni sistem (kluster). Razlog zbog kog smo izabrali ova dva procesora leži u tome da pomenuti procesori najbolje rade u heteregenom modu. Driver za ovaj mod nalazi se u linux kernel-u.

Teorijska granica za broj klustera na jednom čipu je 16, jer procesori koriste 4-bit za pamćenje CLUSTERID broja koji označava poziciju procesora u klusteru.

Procesori u ARM arhitekturi se dijele na više mikroarhitektura. Mikroarhitektura predstavlja verziju ARM jezika koji se koristi na datom procesoru i način na koji je taj jezik implementiran u hardver. Svaka od ovih grupa posjeduje karakteristične osobine.

Tabela verzija ARM mikroarhitektura:

Verzija ARM arhitekture	Dužina registra	
ARMv1	32	ARM1
ARMv2	32	ARM2, ARM250, ARM3
ARMv3	32	ARM6, ARM7

ARMv4	32	ARM8
ARMv4T	32	ARM7TDMI, ARM9TDMI, SecurCore SC100
ARMv5TE	32	ARM7EJ, ARM9E, ARM10E
ARMv6	32	ARM11
ARMv6-M	32	ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1, SecurCore SC000
ARMv7-M	32	ARM Cortex-M3, SecurCore SC300
ARMv7E-M	32	ARM Cortex-M4, ARM Cortex-M7
ARMv7-R	32	ARM Cortex-R4, ARM Cortex-R5, ARM Cortex-R7
ARMv7-A	32	ARM Cortex-A5, ARM Cortex-A7, ARM Cortex-A8, ARM Cortex-A9, ARM Cortex-A12, ARM Cortex-A15, ARM Cortex-A17
ARMv8-A	64	ARM Cortex-A35,[37] ARM Cortex-A53, ARM Cortex-A57,[38] ARM Cortex-A72[39]

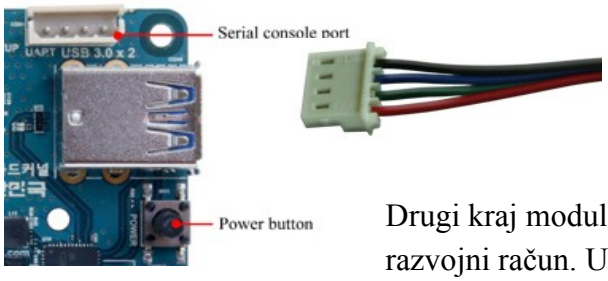
Konekcija sa hardware-om

Serijska konekcija sa razvojnom pločom se ostvaruje putem USB-to-Uart (CP2104) modula. Zadatak modula je da sve podatke koje dolaze sa serijskog porta pretvori u nama prihvatljive podatke koje može pročitati usb host. Da bi ostvarili konekciju sa nekom razvojnom pločom potrebno je podesiti parametre pristupa serijskoj konzoli:

1. *brzinu protoka podataka izraženu u bitovima po sekundi* - potrebno je da kernel i bootloader koriste isti bitrate kako ne bi došlo do rasipanja karaktera i nemogućnosti iščitavanja izlaza sa konzole. Linux kernel podržava više različitih bitrate modova (1200, 2400, 4800, 9600, 19200, 38400, 57600 i 115200 bit-a).
2. *interfejs koji "slušamo"* - nakon uključivanja usb-to-uart modula u kompjuter linux kernel automatski pronalazi driver za modul i kreira device u /dev direktoriju koji daje mogućnost za pristup podacima sa serijske konzole.
3. *kontrola protoka* - garantuje izbjegavanje pojave koja se zove overunning, tj. nemogućnosti aplikacije da prihvati i pravilno obradi sve podatke koji joj pristižu sa serijske konzole. Zbog velikih performansi desktop računara na kom se radi razvoj, možemo reći da se ova pojava neće dešavati i slobodno isključiti opcije kontrole protoka.

Serijska konekcija preko alata minicom

Kako bi serijska komunikacija bila ostvarena, potrebno je da usb-to-uart modul bude priključen na razvojnu ploču. Priključak se nalazi pored mrežnog interfejsa.



Drugi kraj modula koji sadrži usb priključak spojen je na desktop razvojni račun. U terminalu se izvršava komadna i startuje alat minicom:

```
# sudo minicom -s
```

```
+-----[configuration]------+
| Filenames and paths      |
| File transfer protocols  |
| Serial port setup        |
| Modem and dialing        |
| Screen and keyboard      |
| Save setup as dfl        |
| Save setup as..          |
| Exit                     |
| Exit from Minicom        |
+--------------------------+
```

```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyUSB0                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

Nakon konfiguracije alata minicom desktop računar je ostvario serijsku komunikaciju sa razvojnom pločom.

Tehnologije i alati korišteni u izradi:

1. Toolchain (GNU Compiler Collection) - Kompajler, Linker, Archiver, Object Dumper itd.
2. U-boot - bootloader,
3. Linux kernel - operativni sistem,
4. Busybox - minimalni rootfile sistem,
5. GNU i BSD open source alati (web server, mysql, upravljanje particijama...).

Toolchain ili cross-compiler

je skup alata za kompajliranje koda za određenju kompjutersku arhitekturu. Sa različitim vrstama toolchain-a moguće je da se kompajlira kod na jednoj arhitekturi, a izvršava na drugoj. Na primjer razvoj aplikacija za iPhone se uglavnom vrši na x86 arhitekturi ali se aplikacije izvršavaju na ARM arhitekturi.

Toolchain koji je korišten u ovom diplomskom radu sastoji se od sljedećih alata:

Ime alata	Opis alata
• arm-linux-gnueabihf-addr2line	• prevodi adrese u imena fajlova i brojeve linija
• arm-linux-gnueabihf-ar	• kreira, modifikuje i ekstraktuje arhive
• arm-linux-gnueabihf-as	• asembler
• arm-linux-gnueabihf-g++	• GNU C++ Kompajler
• arm-linux-gnueabihf-cpp	• Obrađuje i smješta u kod pretprocesorske direktive prije početka samog kompajliranja
• arm-linux-gnueabihf-elfedit	• alat koji manipuliše izvršnim (ELF) fajlovima, elfedit daje mogućnost promjene header-a u kompajliranom izvršnom fajlu. Koristi se u svrhu kontrole i promjene version control string-a.
• arm-linux-gnueabihf-gcc-4.7.3	• C kompajler
• arm-linux-gnueabihf-gcov	• testiranje u programu
• arm-linux-gnueabihf-gdb	• debugger
• arm-linux-gnueabihf-gfortran	• Fortran 95/2003/2008 za GCC
• arm-linux-gnueabihf-gprof	• alat za analizu performansi kompajlirane aplikacije
• arm-linux-gnueabihf-ld	• linker - spaja više biblioteka i izvršnih fajlova u jedan izvršni fajl, stvara početni izvršni fajl preko kog korisnici ulaze u program
• arm-linux-gnueabihf-ldd	• alat koji ispisuje sve dijeljene biblioteke potrebne izvršnom programu za rad.
• arm-linux-gnueabihf-nm	• gnu nm ispisuje listu simbola izvršnog fajla, ako su simboli ispisani malim slovima predstavljaju lokalne simbole, ako su ispisani velikim slovima predstavljaju externe ili globalne simbole koje posjeduju i neki drugi izvršni fajlovi u sistemu. Simboli ne moraju postajati u izvršnom fajlu. Oni su posljedica

	kompajliranja i većina programera ih briše iz izvršnih datoteka jer zauzimaju prostor.
• arm-linux-gnueabihf-objcopy	• gnu *objcopy* alat koji kopira sadržaj jednog izvršnog fajla u drugi izvršni fajl
• arm-linux-gnueabihf-objdump	• prikaz informacija o izvršnom fajlu, disasembliranje
• arm-linux-gnueabihf-pkg-config	• *pkg-config* alat koji pomaže pri izaboru argumenata za podešavanje kompajlera - optimizacija kompajlera
• arm-linux-gnueabihf-ranlib	• alat koji kreira i modifikuje biblioteke
• arm-linux-gnueabihf-readelf	• prikazuje sadržaj izvršnih ELF fajlova
• arm-linux-gnueabihf-size	• prikaz veličine izvršnog fajla
• arm-linux-gnueabihf-strings	• ispis stringova koji se nalaze u izvršnom fajlu
• arm-linux-gnueabihf-strip	• brisanje svih simbola nastalih pri kompajliranju

Primjer kompajliranja aplikacije hello.c sa razloženim procesom kompajliranja:

```
[root@localhost]#  arm-linux-gnueabihf-cpp  hello.c > hello.i
```

Rezultat je "hello.i" fajl koji sadrži kod sa uključenim pretprocesorskim direktivama.

1. Kompajliranje: Kompajlira kod sa pretprocesorskim direktivama u asemblerski kod namijenjenom specifičnom procesoru.

```
[root@localhost]#  arm-linux-gnueabihf-gcc-4.7.3 -S hello.i
```

2. Asembliranje: asembler pretvara asembly kod u mašinski kod i kreira objekat "hello.o".

```
[root@localhost]#  arm-linux-gnueabihf-as -o hello.o hello.s
```

3. Linker: Linkovanje objekta sa kodom iz biblioteka potrebnih za rad objekta, kako bi bio dobijen finalni izvršni fajl.

```
[root@localhost]#  arm-linux-gnueabihf-ld -o hello hello.o ...libraries...
```

Toolchain je obezbijeđen od strane proizvođača procesora. Moguće je pristupiti kreiranju vlastitog toolchain-a, ali ovaj proces se smatra nepotrebnim ako je proizvođač već obezbijedio toolchain za familiju procesora koju isporučuje na tržište.

Particije i raspored fajlova

Podešavanje particija se vrši na desktop razvojnom računaru koristeći čitač eMMC kartica i fdisk (program za manipulaciju particijama).
Kreirane su dvije particije na eMMC kartici. Za kreiranje particija korišten je alat fdisk. Kako bi se automatizovao proces kreiranja particija u budućnosti kreirana je skripta make_boot.sh koja priprema eMMC karticu za postavljanje fajlova:

make_boot.sh:

```
fdisk /dev/mmcblk0 <<EOF
n                               #kreiranje prve particije
p                               #postavljanje particije kao primarne
1                               #definisanje rednog broja particije
3072                            #particija pocinje od 2048 sektora
+129M                           #definisanje veličine particije
w                               #upis promjena na disk
EOF
fdisk /dev/mmcblk0 <<EOF
n                               #kreiranje druge particije
e                               #postavljanje particiju kao extended
2                               #definisanje rednog broja particije
266239                          #druga particije pocinje na kraju prve
15663103                        #i završava na zadnjem sektoru
w                               #upis promjena na disk
EOF

mkfs -t vfat /dev/mmcblk0p1   #postavljanje filesystem-a na prvu
                              particiju, filesystem mora da bude formata
                              FAT kako bi uređaj mogao pravilno da uradi
                              boot proces.

mkfs -t ext4 /dev/mmcblk0p2  #druga particija može da koristi bilo koji
                              filesystem koji linux može da pročita -
                              korišten je ext4 filesystem.
```

Tabela koja prikazuje pravilan raspored fajlova na razvojnoj ploči:

Area Name	Veličina	Od sektora	Do sektora	Ime fajla	Ime particije
Partition table	512	0	0		
fwbl1	15KB	1	30	fwbl1	
bl2	16KB	31	62	bl2	
u-boot	328KB	63	718	bootloader	
TrustZone SW	256KB	719	1230	tzsw	
u-boot environment	16KB	1231	1262		
FAT16 boot	129M	3072	266239		mmcblk0p1
EXT4 rootfs	Up to 64GB	266240	preostali blokovi		mmcblk0p2

Bootloader

Das U-Boot (univerzalni bootloader) je open-source bootloader koji se pretežno koristi u embedded sistemima. Prednost U-Boot-a nad ostalim bootlader-ima je podrška za mnoštvo različitih kompjuterskih arhitektura: ARM, AVR32, MicroBlaze, MIPS, x86.
Sama modularnost ovog bootloader-a daje mogućnos korisnicima da ga prilagođavaju za neke nove arhitekture i da u potpunosti kontrolišu boot proces i parametre koji se prosljeđuju kernel-u.

Postoje dvije vrste bootloader-a:

Single stage	Multi stage
Sadrže jedan fajl koji je sposoban da učita operativni sistem (npr. linux kernel) direktno u RAM i preda mu kontolu, ali on sam je učitan u potpunosti od strane BIOS-a. Single stage pristup se koristio u 16-bitnim procesorima i danas se smatra zastarjelim i teško primjenljivim na 32-bitnim procesorima, jer 32-bitne komadne zauzimaju dvostruko više memorijskog prostora. Znajući da je MBR veličine 512byte-a većina današnjih bootloader-a su multistage bootloader-i.	Zbog nedostatka biosa na ARM arhitekturi i MBR-a male veličine koristimo inkrementalni pristup boot procesu ili multistage. Bootloader se sastoji iz više dijelova. U prvih 512bytes MBR-a, upisan je samo kod koji jednostavno pokreće drugi bootloader i predaje mu kontrolu. Time je dobijeno više resursa u svakom koraku. Svaka etapa boot procesa se izvršava u drugom memorijskom kontekstu.

Odroid razvojna ploča, kao i većina procesora iz Exynos familije, posjeduje zaključan interni bootloader. Kako bi bio omogućen uspješan boot proces i učitavanje operativnog sistema u RAM korisnik mora posjedovati eksterni bootloader potpisane GPG ključem od strane samog proizvođača. Ovakav proces učitavanja operativnog sistema naziva se SecureBoot.

Secure boot tehnologija koristi se kako bi se stvorila zaštita i onemogućio nedozvoljen pristup uređaju.
Ako posjedujemo neku razvojnu ploču čija sigurnost zavisi od software-a, hakera ništa neće spriječiti da jednostavnom zamjenom software-a ili argumenata u bootloader-u preskoči login proces i pristupi sistemu tako što direktno učita shell. Većina današnjih embedded uređaja na tržištu nema nikakvu zaštitu od fizičkog pristupa, jednostavnim kopčanjem serijske konzole moguće je hakovati većinu uređaja na tržištu. Sličan proces može se koristiti na x86 arhitekturi gdje gašenjem računara iz struje i odstranjivanjem baterije koja napaja BIOS haker dobija pristup BIOS-u, a samim ti i cjelokupnom operativnom sistemu.

Sa druge strane, proizvođači često zloupotrebljavaju SecureBoot tehnologiju, čime korisnike sprječavaju da dublje izučavaju boot proces. Spriječen je pogled u samu strukturu izrade čipova, i kreiran interni bootloader koji hardware dovodi u fazu spremnu za bootanje (preuzima funkciju BIOS-a) sa nekim drugim bootloader-om koji u višim koracima boot procesa preuzima kontrolu.

Problem u SecureBoot pristupu nastaje u trenutku kada želite da testirate hardware. Na primjer, razvojna ploča neće da učita linux kernel i teško je pretpostaviti o čemu se radi, ali prirodno je da se greška prvo traži u hardware-u. Dijelove hardware-a, kao što su RAM ili L1, L2 keš, nije moguće testirati jer interni bootloader ne želi u iRam (SRAM) da učita funkcije koje nisu potpisane od strane proizvođača.

Proizvođači, takođe, žele da korisnici zavise od bootloader-a koji vam oni isporučuju i time često guše zajednice koje teže da razviju slobodne bootloader-e kao što je u-boot.

Kako ne bi dodatno zakomplikovali i proces razvoja proizvoda na hardware-u koji posjeduje secureboot, proizvođači ubacuju u bootloader-e, kao što je u-boot, već kompajlirane i potpisane izvršne fajlove (binary blob) gdje korisnik može da koristi postojeći izvršni fajl ali nema mogućnost promjena.

Uboot i Odroid razvojna ploča

Činjenica da je interni bootloader zaključan ne predstavlja veliku prepreku u ovom projektu, jer veći dio kontrole boot procesa koji je potreban pri izradi sadržan je u eksternom bootloader-u, a definisaćemo da posjedujemo ispravan hardware. Interni bootloader jednostavno učitava u-boot binary i predaje mu kontrolu.

Proces kreiranja potrebnih paketa za uspješno boot-anje sistema pristupamo:

U-boot bootloader je preuzet sa interneta u obliku tar.gz arhive. Otpakivanjem arhive dočekuje nas source kod bootloader-a. U-boot može da se kompajlira i podešava za različite arhitekture. Sve konfiguracije vezane za arhitekture nalaze se u folder-u arch/, board/ i include/configs.

Konfiguracije koje su potrebne u ovom projektu nalaze se na sljedećim putanjama:

arch/arm/cpu/armv7/exynos/ - u ovom direktorijumu se nalaze source fajlovi za boot-anje sa armv7 arhitekture; sve razvojne ploče koje koriste ovu arhitekturu mogu da iskoriste iste fajlove za boot proces.

board/samsung/smdk5422 - ovaj direktorij daje source-fajlove za opis razvojne ploče i svih uređaja koji se nalaze na njoj, podržava rad source fajlova iz direktorija koji opisuje arhitekturu, svaka razvojna ploča posjeduje specifičnu konfiguraciju. Kada razvijamo konfiguraciju za novu razvojnu ploču preporučuje se da kao polaznu tačku uzmemo ploču iste arhitekture i sličnih interfejsa koja je već podržana od strane u-boot-a i odatle nastavimo razvoj.

include/configs/odroid.h - konfiguracioni fajl za Odroid razvojnu ploču.

Cross compile-iranje bootloader-a

Unutar u-boot direktorija pokrećemo komande za kros kompajliranje bootloader-a sa odgovarajućom konfiguracijom. Komandom (sh varijablom) ARCH definišemo arhitekturu za koju se kompajlira u-boot, dok je sa komandom CROSS_COMPILE određen kros kompajler koji se u procesu kompajliranja.

```
[root@localhost]#  make ARCH=arm CROSS_COMPILE=../gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux/bin/arm-linux-gnueabihf- odroid_config
```

```
[root@localhost]#  make ARCH=arm CROSS_COMPILE=../gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux/bin/arm-linux-gnueabihf-
```

Završetkom procesa dobijena su 4 fajla (već opisana u tekstu) koja se postavljaju na prvu particiju eMMC kartice formata FAT:

- fwbl1,
- bl2,
- u-boot.bin,
- tzsw.bin.

fwbl1

Ovaj fajl CPU prvi poziva, informacije o njemu nisu dostupne jer je zaključan potpisanim ključem.

bl2

Ovo je SPL (Secondary Program Loader) , dio U-Boot-a, koji riješava sledeći problem ; On-chip ROM ili neki drugi mehanizam žele da učitaju izvršni fajl bootloader-a u iRAM(SRAM), ali kako je iRAM često jako malen po nekad manji od 10KiB u-boot ne može tu da se učita, stoga kreira manji bootloader SPL koji odgovara veličini iRAM memorije koji kasnije učitava u-boot izvršni fajl u RAM memoriju.

u-boot.bin

Ovo je U-Boot izvršni fajl

TrustZone Software

Samsung/ARM podrška za Trustzone platformu.

TrustZone tehnologija za Cortex-A procesor koristi se u svrhu pokretanja “trusted boot” ili sigurnog boot procesa tipično se koristi u mobilnim uređajima kako bi obezbjedili mehanizmi zaštite, podršku za kriptografiju u ranoj fazi boot procesa.

U trenutku pokretanja, razvojna ploča počinje da traži boot-abilni disk.

1) iROM (Kod unutar SoC-čipa) težiće da pronađe boot-abilni disk u prvih 512 bytes. Na tom memorijskom segmentu od 512 bytes nalazi se potpisani fwbl1 binary blob ubačen u u-boot. fwbl1 biće učitan u SRAM memoriju koja se nalazi unutar procesora.

2) fwbl1 će učitati bl2 (SPL) koji je dio U-Boot u RAM memoriju.

3) bl2 će učitati U-boot u RAM memoriju i predati mu kontrolu na boot procesom.

4) U-Boot će odraditi ostalo: upravljanje TrustZone, učitavanje kernel image-a ili ram disk-a.

Varijable okruženja

Kreirane su varijable okruženja za U-Boot-a kroz njih se vrši konfiguracijuabootloadera:

- boot_emmc.txt

```
ODROIDXU4-UBOOT-CONFIG

# U-Boot parametri

# Konfiguracija Mac adrese
setenv macaddr "00:1e:06:61:7a:39"

#Konfiguracija serijske konzole i memorijskog medija na kome se nalazi distribucija
setenv bootrootfs "console=tty1 console=ttySAC2,115200n8 root=/dev/mmcblk0p1 rootwait rw"

# boot komande
setenv bootcmd "ext4load mmc 0:1 0x40008000 /boot/zImage; ext4load mmc 0:1 0x44000000 /boot/dtbs/exynos5422-odroidxu4.dtb; bootz 0x40008000 - 0x44000000"

setenv bootargs "${bootrootfs} smsc95xx.macaddr=${macaddr}"

# Komanda za pokretanje boot procesa
boot
```

Kako bi u-boot bio u stanju da čita konfiguraciju potrebno je tekstualni fajl pretvoriti u konfiguracioni fajl kako bi u-boot mogao da ga razumije.

```
[root@localhost]#   mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "Odroid eMMC boot" -d boot_emmc.txt boot.scr
```

Dobijen je fajl boot.scr koji je, zapravo, korisnički konfiguracioni fajl. U-boot preko konfiguracionog fajla može da prima različite komande i izvršava naredbe u toku boot procesa, tako da korisnik nema potrebu da kompajlira u-boot iznova svaki put kada želi neku promjenu u konfiguraciji.

Nakon kreiranja bootloader-a potrebno je prebaciti fajlove na prvu particiju eMMC kartice, tako da se ne naruši poredak koji zahtijeva hardware. Svako odstupanje od pravilnog poretka fajlova može dovesti do neuspješnog boot procesa.

Smještanje bootloader-a na razvojnu ploču

Pomoću alata dd i kreirana je skripta koja će automatski završiti posao.

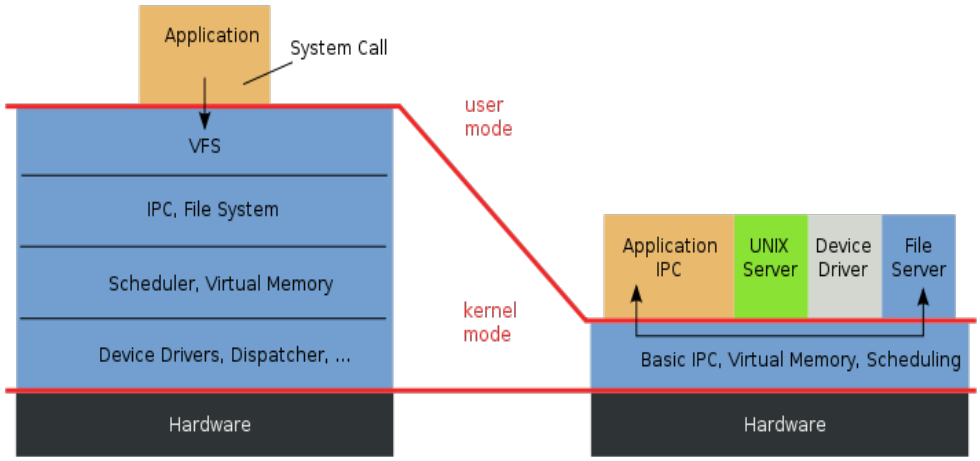
fuse.sh

```
signed_bl1_position=1
bl2_position=31
uboot_position=63
tzsw_position=719

#Kopiranje zaključanog bootloader
```

```
dd iflag=dsync oflag=dsync if=./fwbl1.bin of=$1 seek=$signed_bl1_position

#SPL verzija u-boot bootloader-a
dd iflag=dsync oflag=dsync if=./fwbl2.bin of=$1 seek=$bl2_position

#u-boot izvršni fajl
dd iflag=dsync oflag=dsync if=./u-boot.bin of=$1 seek=$uboot_position

#TrustZone
dd iflag=dsync oflag=dsync if=./tzsw.bin of=$1 seek=$tzsw_position

sync - pražnjenje sistemskih bufera
```

Linux kernel - operativni sistem

Linux je klon operativnog sistema Unix kreiran od strane Linus Torvalds-a. Linux kernel kao i svaki drugi operativni sistem upravlja hardware-om i obezbjeđuje neophodne funkcije za rad aplikativnog software-a (programi ne mogu da rade bez operativnog sistema).
Linux nastaje zbog nezadovoljstva Torvaldsa tada zastupljenim sistemom MINIX. Glavna razlika između Linux-a i MINIX-a leži u činjenici da MINIX koristi mikrokernel, a linux monolitski kernel. Razlika između ova dva pristupa u dizajnu operativnog sistema ima mnogo, čak postoje i varijante hibridnih kernela, ali osnovna razlika je u tome što linux cjelokupan proces rada obavlja u jednom adresnom prostoru (kernel-space) dok userspace ostavlja samo za korisničke aplikacije. Mikrokernel u kernel space-u drži samo basic scheduler, basic memory handling, I/O primitives function i na taj način postiže veličinu toliku da može da stane u L1 keš memoriju, dok se ostali servisi izvršavaju u userspace-u (u RAM-u).



Hardver radi brzo i korisnik ne primjećuje da operativni sistem prelazi iz kernelspace-a u userspace više puta u sekundi. Ova pojava se zove promjena konteksta i potrebno je vrijeme da se ona desi.

U nastavku je detaljnije opisana razliku izmeću monolitskog i mikrokernela.

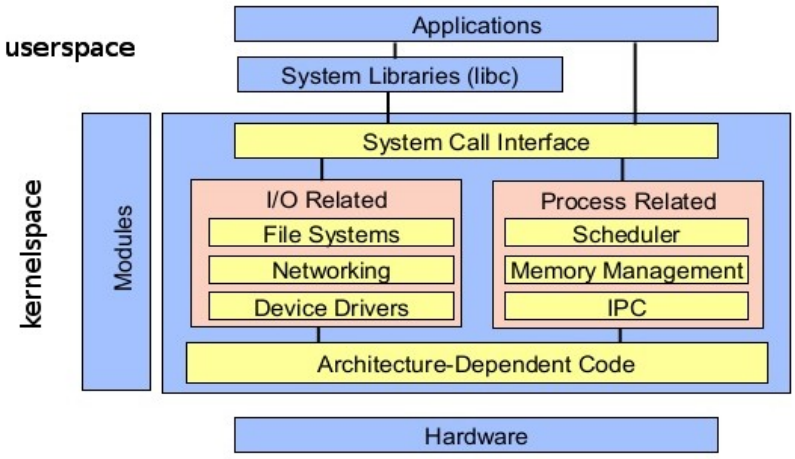
Monolitski kernel je jedan veliki proces koji se izvršava u jednom adresnom prostoru (kernelspace-u). Svi kernel servisi i moduli se takođe izvršavaju samo u kernelspace-u. Kernel može da pozove koju god funkciju želi i to direktno bez promjene konteksta. Monolitski kerneli spadaju u red najbržih kernel-a, jer imaju manju potrebu za promjenom konteksta. Iako se smatra da je linux kernel portabilan, mikrokerneli na tom polju postižu bolje rezultate. Dodavanje novih opcija u monolitski kernel, znači kompajliranje cijelog kernela. S obzirom da se device driver-i nalaze u kernelspace-u, monolitski kernel je nesigurniji, jer jedan driver može da izazove kolaps cijelog operativnog sistema. Mikrokerneli nemaju ovaj problem i zato su često korišteni u vojnim uređajima.

U mikrokernelima, kernel je podijeljen u više nezavisnih procesa, poznatih kao serveri. Neki od ovih servera se izvršavaju u kernelspace-u, dok drugi mogu da se izvršavaju u userspace-u. Svi serveri su odvojeni i koriste različit adresni prostor. Komunikacija u mikrokernelu između servera odvija se putem poruka. Servisi se pokreću tako što server pošalje poruku servisu da se upali, jer ne

može direktno da mu pristupi ako je u drugom adresnom prostoru. Serveri koji nezavisno rade daju prednost korisniku i dodatan stepen zaštite, jer ako jedan server otkaže to ne utiče na rad ostalih. Pomenuti MINIX sistem nije "čisti" mikrokernel jer se dio driver-a izvršava u kernel space-u. Nažalost, danas mikrokerneli ne nalaze široku primjenu, prije svega zbog kompleksnosti razvoja.

Komunikacija preko jednostavnih poruka izaziva mnogo problema pri debug-ovanju sistema, jer često ne znamo koji je pravi razlog pada servisa. Servis može da prestane sa radom jer mu je neki drugi servis poslao poruku. Nedefinisani su slučajevi kada više servisa pošalje poruku odjednom, jer moramo brzo da odlučimo šta je prioritet, a često ne možemo da znamo koliki je error-chain (greška je mogla nastati usljed komunikacije više servisa, teško je utvrditi gdje je ona počela).

Struktura linux kernela - monolitski kernel



Korisničke aplikacije komuniciraju sa kernel-om putem sistemskih poziva (syscalls). Njihov broj je ograničen u linux kernel-u. Sistemske pozive program može pozivati direktno ili preko biblioteka. Linux kernelu potrebne su C-biblioteke, koje proširuju funkcionalnost sistema.

Libc je standardna biblioteka koja definiše sistemske pozive i druge bazne funkcije programskog jezika C kao što su open, malloc, printf, exit.
U diplomskom radu korištišten je GNU C library, open-source implementaciju biblioteka za programski jezik C.

Linux kernel je sastavljen od 5 podsistema, device driver-a i asemblerskog koda za različite arhitekture:

1. *The Process Scheduler (SCHED)* - odgovoran je za kontrolu i izvršavanje procesa. Scheduler pokušava na pravedan način omogućiti svim procesima podjednak pristup resursima i obezbijediti što savršeniji multitasking. Linux posjeduje više algoritama za proces kontrole izvršavanja i daje mogućnost da se prekonfiguriše scheduler prema potrebama korisnika.

2. *The Memory Manager (MM)* - raspoređuje memoriju tako da svi procesi mogu u isto vrijeme da koriste jedan adresni prostor. Kako linux kernel posjeduje VFS (virtualni file system) dozvoljeno mu je da potroši više radne memorije nego što fizički posjeduje, a da potrebu za memorijom nadoknadi sa tvrdih diskova po cijeni lošijih performansi.

3. *The Virtual File System (VFS)* - podržava rad više različitih filesystema u isto vrijeme i pruža univerzalan način za pristup svim podacima sa različitih vrsta hardware-a.

4. *The Network Interface (NET)* - podrška za mrežne interfejse i TCP/IP protokole.

5. *The Inter-Process Communication (IPC)* - komunikacija između procesa putem signala.

Device drivers

Veći dio linux kernela predstavljaju device driver-i ili moduli koji daju funkcionalnost uređajima. Linux je nezavisan od ovih driver-a i može da radi bez njih. Drivere mogu da se kompajliraju tako što su kompajlirani u kernelu (static) ili kao externi moduli koji se učitavaju tokom boot procesa.

Svi driver-i u linux-u nalaze se u direktoriju linux/drivers, razvrstani su po grupama uređaja.

Architecture-dependent code

Linux posjeduje dio koda koji je zavisan samo od arhitekture na kojoj se izvršava, a source-fajlovi se nalaze u direktorijumu linux/arch. Tu se, takođe, nalaze informacije o svakoj arhitekturi koja je podržana od strane kernel-a. Svaka arhitektura ima poddirektorijume u kojim se detaljnije opisuje rad kernela na različitim arhitekturama.

Linux startup

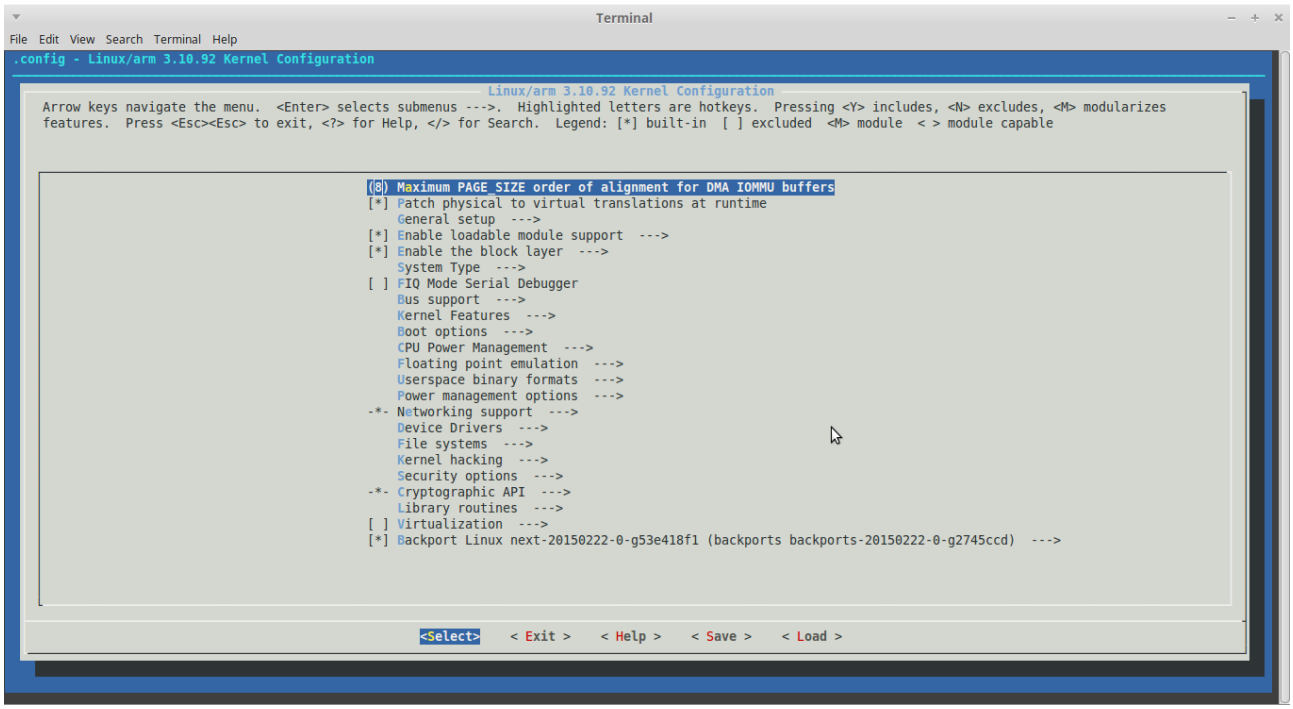
Kada bootloader završi sa radom proslijeđuje parametre kernelu. Na početku boot procesa, dio kernela se otpakuje iz arhive malim alatima koji se nalaze na početku image fajla. Kernel radi osnovu inicijalizaciju hardvera i nakon toga otpakuje ostatak kernela i smješta u radnu memoriju. Najčešće se radi o zImage kompresovanom fajlu (format image fajla u koji se pakuje linux kernel). Nakon smještanja kernela u radnu memoriju pokreće se prva userspace funkcija init(), koja nastavlja ostatak boot procesa u userspace-u.

Inicijalni koraci startup procesa u linux kernelu:

arch/arm/boot/compressed/head.S - postavlja osnovne funkcije za upravljanje hardverom
arch/arm/boot/compressed/misc.S - pravljenje stack-a i otpakivanje ostatka kernela
arch/arm/kernel/head.S - pokreće funkciju start_kernel() i čisti BSS registar
init/main.c - pokreće funkciju start_kernel()
arch/arm/kernel/process.c - pokreće init, prvi proces u userspace-u. Izvršava se funkcija cpu_idle(), prije koje je aktiviran scheduler. Nakon ovog procesa dobijena je funkcija multitasking-a.

Cross compile-iranje kernela

Alat zasnovan na ncurses biblioteci daje mogućnost konfiguracije kernela kroz grafički interfejs u terminalu.



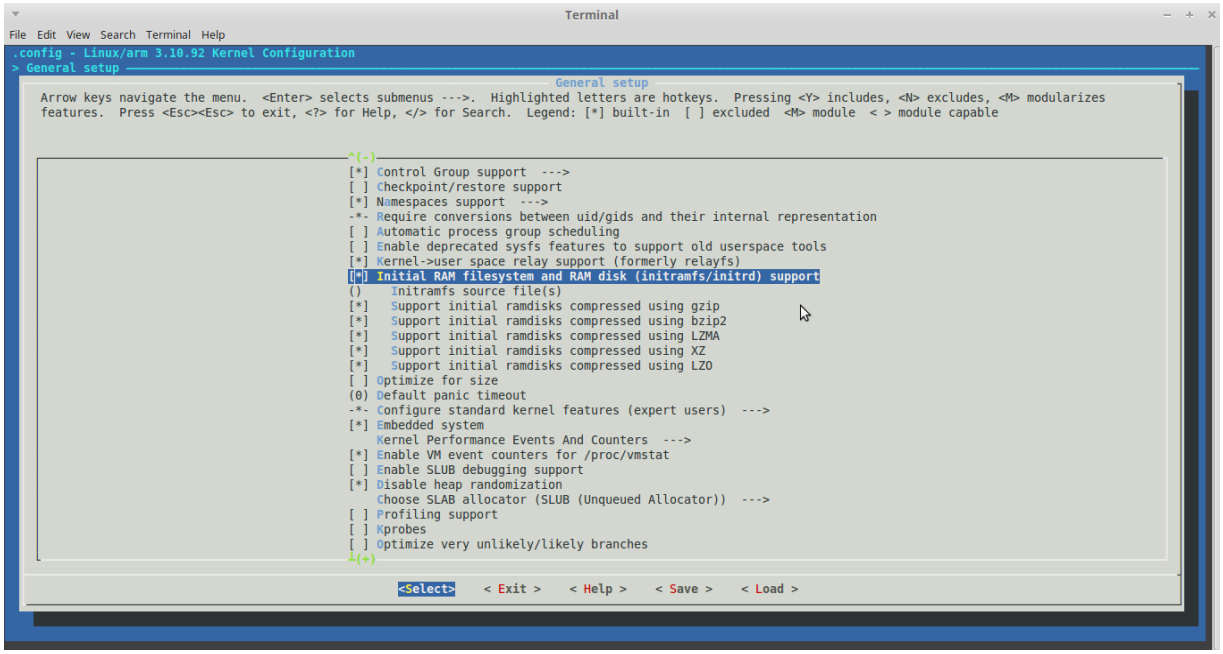
U ovom radu neće biti opisan detaljan proces konfiguracije linux kernel-a, jer je proces dugačak i izašao bi iz domena ovog rada. Ukratko, svodi se na aktiviranje željene arhitekture na kojoj se kernel izvršava i testiranja svakog drajvera pojedinačno kako bismo se uvjerili koji driver-i rade, a koji stvaraju probleme. Kada se ustanovi koji driver-i nisu u funkciju pokušavamo da ih prilagodimo datoj arhitekturi. Linux kernel posjeduje generičke driver-e za različite arhitekture, i oni se trude da na svakoj arhitekturi obezbijede minimalne funkcije za relativno jednostavan dalji nastavak rada.
Ako kernel nije u mogućnost da radi na željenoj arhitekturi, potrebno je provjeriti da li je kernel portovan na tu arhitekturu. Ako nemamo port za arhitekturu, prije kompajliranja samog kernel-a i izvršavanja bilo koje aplikacije, potrebno je prilagoditi kompajler za rad na toj arhitekturi.

Kompajliramo kernel za ARM arhitekturu:

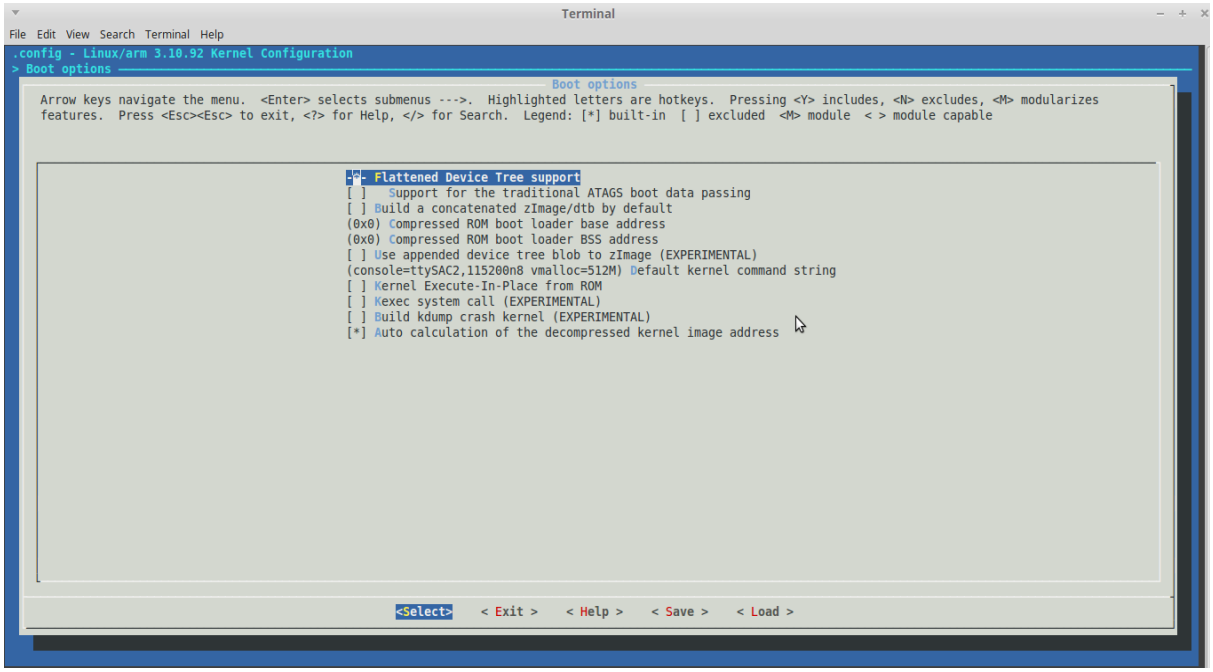
```
[root@localhost]#  export PATH=${PATH}:/opt/toolchains/arm-eabi-4.6/bin
[root@localhost]#  make ARCH=arm CROSS_COMPILE=arm-eabi odroidxu4_defconfig
```

Prikaz promjena u konfiguraciji:

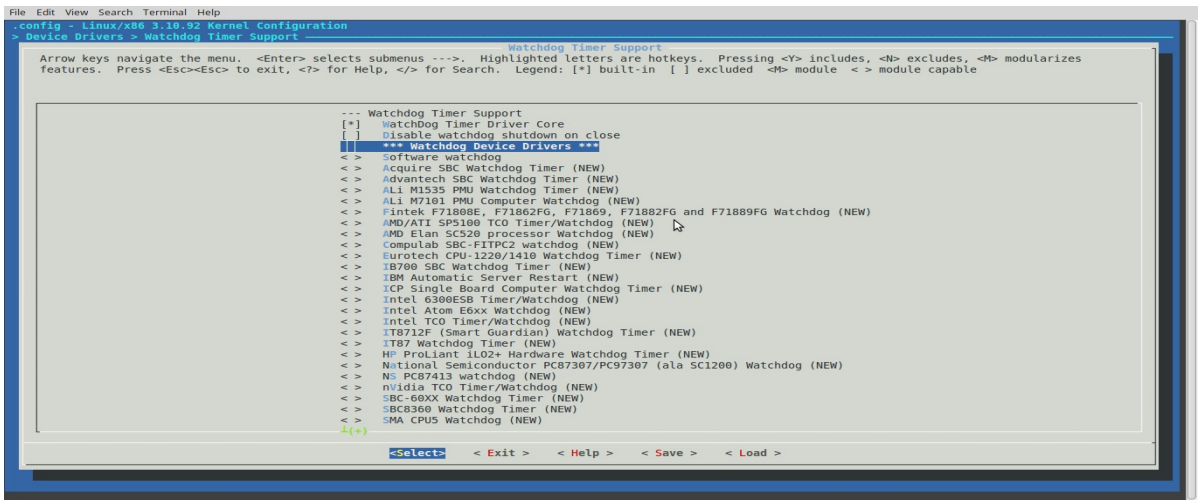
```
[root@localhost]#  make ARCH=arm CROSS_COMPILE=arm-eabi menuconfig
```



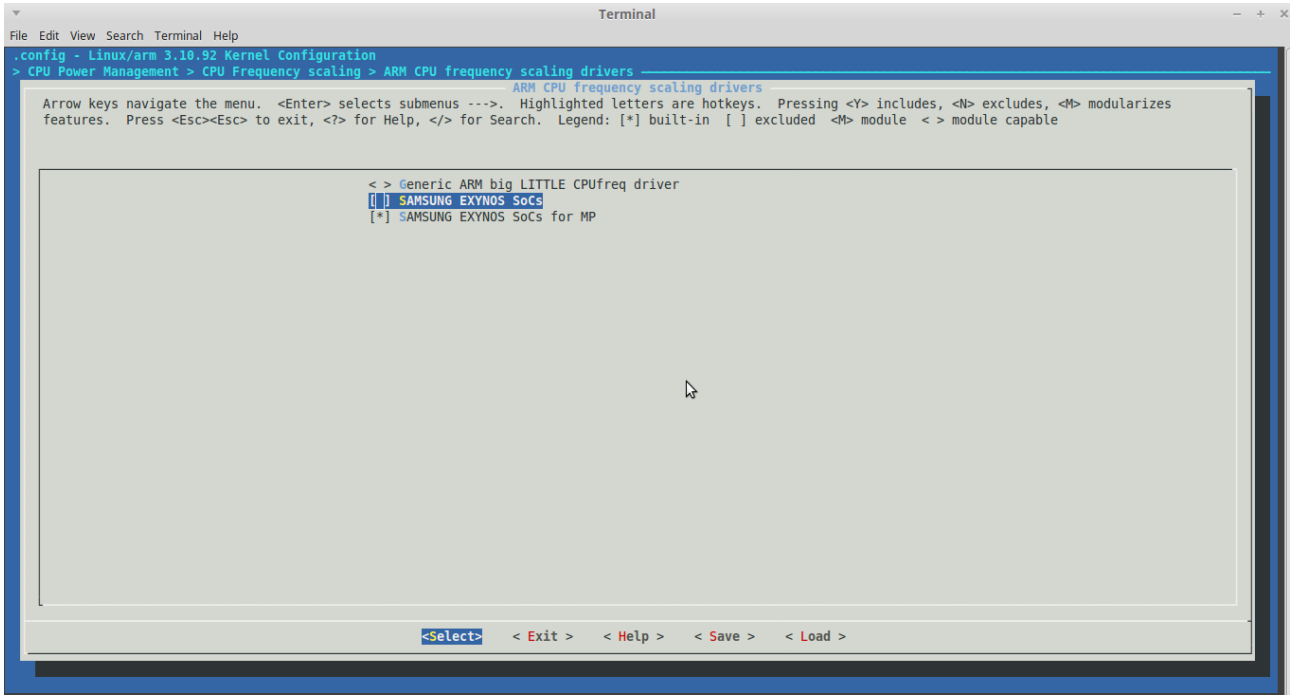
Opcija koja definiše ramdisk (ram disk će se kreirati u sledećem poglavlju).



Gašenje automatske konkatanacije Device Tree u zImage, ostavljenja mogućnost za učitavanje eksternog fajl-a.



Kreiran je watchdog timer (opisano u sljedećem odjeljku)



Konfiguracija driver-a za HMP cluster

Kompajliranje kernela, device tree-ija i modula:

```
[root@localhost]#  make ARCH=arm CROSS_COMPILE=arm-eabi zImage -j4                    #kernel image
[root@localhost]#  make ARCH=arm CROSS_COMPILE=arm-eabi exynos5422-odroidxu4.dtb  #FTD
[root@localhost]#  make ARCH=arm CROSS_COMPILE=arm-eabi modules -j4
```

Instaliranje linux-a i modula:

```
[root@localhost]#  cp arch/arm/boot/zImage arch/arm/boot/dts/exynos5422-odroidxu4.dtb /media/mmcblk0p1
[root@localhost]#  make INSTALL_MOD_PATH=/media/mmcblk0p1        #instalacija linux modula
```

Fajlovi na eMMC kartici trebaju biti raspoređeni po sljedećem rasporedu:

Particija 1:

- Kernel Image (zImage),
- exynos5422-odroidxu4.dtb,
- boot.src,
- uInitrd (ako postoji).

Particija 2:

- rootfs (a.k.a. File System) - definisano u sledećem poglavlju.

Flattened Device Tree

Predstavlja strukturu podataka koja opisuje hardware-ske konfiguracije. Ovaj princip je relativno nov u linux kernel-u . Ideja FTD-a leži u tome da određene skupine uređaja često koriste slične driver-e. Driveri za SPI bus, kao bus sistem on je standardizovan i zna se na koji način funkcioniše i koje rutine koristi. Ako imamo više TFT uređaja koji koriste SPI bus jedine razlike između njih

mogu da budu adrese koje driver koristi za čitanje i pisanje sa uređaja. Na taj način više različitih driver-a možemo da svedemo na jedan koji ima mogućnost korištenja FTD-a (u driver mora biti implementirana funkcija FTD). Jednostavnom promjenom adresa u FTD-u koristitimo jedan driver na više različitih ali arhitekturno sličnih uređaja.
Neke od funkcija koje opisuje FTD.

- broj procesora i vrsta procesora,
- osnovne adrese za inicijalizaciju RAM memorije i samu veličinu memorije,
- bus sisteme i bridge funkcije,
- eksterne uređaje,
- interrupt kontroler.

FTD može biti statički povezan sa kernelom ili učitan kao eksterni fajl.

Ovaj metod može dosta da pomogne u portovanju linux-a na nove razvojne ploče. FTD nikada ne treba predstavljati kao univerzalni pristupni interfejs preko kog se mogu konfigurisati svi uređaji. Određene skupine uređaja zbog kompleksnosti driver-a ne mogu da se opišu pomoću FTD-a.

Watchdog timer

Predstavlja timer koji detektuje kada sistem ostane u mrtvom stanju i pokušava da ga resetuje. Watchdog je elektronski timer (reprezentovan kao čip na razvojnoj ploči) koji odbrojava (unazad) kada vrijednost timer-a dosegne nulu tj. sistem prestane da osvježava brojač. Čip u kom je implementiran watchdog šalje timeout signal i resetuje operativni sistem, tako da sistem nikada ne može da završi u mrtvom stanju. Potreba za ovom vrstom timer-a se javila prvi put kada su ljudi pokušali da kontrolišu udaljeni uređaj, primjećeno je da uređaji koji zaglave u mrtvom stanju ne mogu da se resetuju sami već to korisnik mora ručno da uradi, što je u slučaju satelita često nemoguće, pa su pristupili kreiranju watchdog timer-a.

Rootfile sistem (busybox based)

Jedan tipičan GNU/Linux sistem se obično sastoji od više hiljada fajlova. Svi fajlovi (konfiguracioni, log, binarni, itd.) su rasuti po rootfs. Kako bi se korisnici i programeri lakše snalazili u ovoj hijerarhiji, kreiran je dokument koji regulište standarde u izgradnji rootfile sistema (rootfs). Ovaj dokument se zove Filesystem Hierarchy Standard (FHS).
Mnogi programi se razvijaju prateći ova pravila i podrazumijevaju da se određeni direktorijumi orjentišu po FHS fajlsistemu. Po tome je veoma moguće da neki program ne funkcioniše jer nema nekog direktorijuma u osnovnom fajlsistemu.

U nastavku će ukratko biti opisani svaki od direktorija koji treba da se nalazi u root direktoriju.

/bin - Esencijalni komandi binaries

Sadrži komande koje može koristiti administrator i korisnik; ove komande su ključne za funkcionisanje većeg dijela skripti koji se koriste u sistemu. Potrebna su korisnička prava kako bi se izvršile komande iz ovog direktorijuma.

/boot - Statički fajlovi i biblioteke potrebne bootloader-u

/dev - Device fajlovi

Linux je karakterističan sistem, jer sve uređaje i podatke vidi kao fajlove. U dev/ direktoriju nalaze se interfejsi za pristup device driver-ima, mount-ovane particije i virtuelni driveri. Linux dijeli uređaje na tri vrste:

1. character device - uređaji koji vraćaju karakter po karakter,
2. block device - uređaji koji podatke organizuju u blokove (hard diskovi),
3. network device - mrežni uređaji takođe predstavljaju jednu vrstu block-devica, sa razlikom da posjeduju drugačije metode čitanja i pisanja podataka.

/etc - sistemska konfiguracija

Konfiguracioni fajlovi za linux sistem. Većina ovih fajlova je opciona ali uglavnom svaki linux sistem posjeduje popunjen etc direktorijum. Konfiguracioni fajlovi se koriste za kontrolu rada pograma, moraju biti statički i ne smiju biti izvršni fajlovi (binary).

/lib - Esencijalni “shared library” i kernel moduli

Sadrži kernel module i shared library-ije (biblioteke) potrebne za boot-ovanje sistema, izvršavanje komandi sa terminala, i sve alate koji se nalaze u bin/ i sbin/ direktorijama. Windows ekvivalent library-iju je DLL. Ovi library-iji kao i cijelokupan sadržaj direktorija /lib je vitalan za pravilno funkcionisanje sistema.

/media - Tačka na koju se montiraju prenosivi uređaji

Ovdje se montira temporalni fajl sistem; direktorijum dobija na značaju tek kada je na sistemu prisutan neki prenosivi uređaj u sistemu, inače je prazan direktorijum.

/mnt - Tačka na koju se montiraju particije i fajl sistemi

Temporalni direktoriji za montiranje particija i fajl sistema. Programi ne bi smjeli da se montiraju na direktorij `/mnt` automatski.

/opt - Add-on aplikacije

Ovaj direktorij je rezervisan za sve programe i dodatke koji nisu pokriveni standardnom instalacijom.

/sbin - sistemski izvršni fajlovi

Za razliku od direktorija `/bin`, komande koje se nalaze u ovom direktoriju nisu namijenje običnom korisniku već administratorima. Korisnik može da ih koristi po potrebi i za većinu komandi iz ovog direktorija potrebna su root prava kako bi se izvršile. Sadrži alate ključne za boot proces, oporavak i popravku sistema.

/srv - Podaci o servisima

Ovaj direktorij je opcion i rijetko se koristi u modernim distribucijama.

/tmp - Privremeni fajlovi

Programi u ovaj direktorij smještaju privremene fajlove.

/usr - Sekundarna hijerarhija, hijerarhija korisnika

/var - Varijabilni podaci, log fajlovi, programski keš

/root - Home direktorij root korisnika

Home direktorija za root korisnika. Ovom direktorijumu po podrazumijevanim podešavanjima ne mogu pristupiti drugi korisnici sistema, mada je moguća konfiguracija koja će to dopustiti.

/home - Korisnička home direktorija

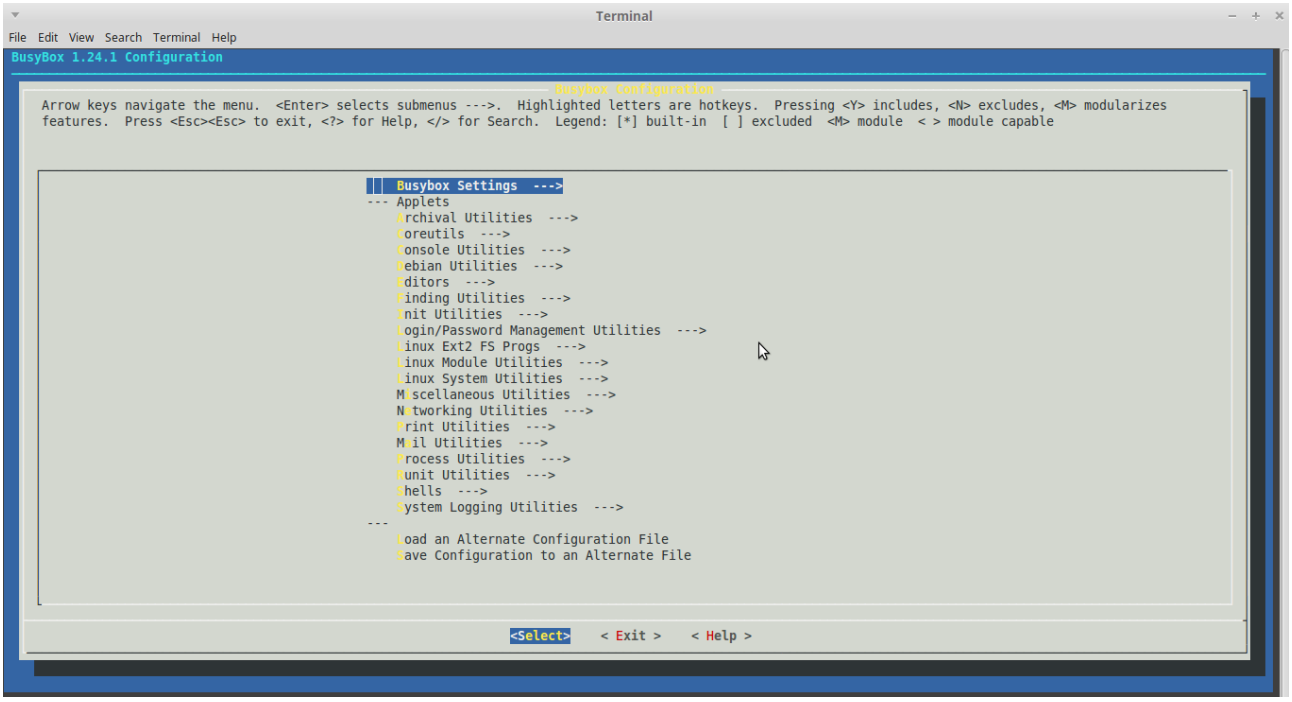
Busybox

Busybox kombinuje male verzije alata koji se često koriste u UNIX sistemima u jedan mali izvršni fajl. Busybox često nazivaju "švajcarski nož ugrađenih sistema" jer je optimizovan za upotrebu u embedded sistemima. Isporučuje sve potrebne alate za izgradnju root file sistema embedded GNU/Linux distribucije. Ako se koristi neki standardni GNU alat iz osnovnog paketa coreutils, često kompajliramo alat sa previše opcija koje nećemo koristiti u radu. Navešćemo primjer alata ls koji se koristi za ispis fajlova na konzolu. On posjeduje stotine opcija koje olakšavaju i ubrzavaju svakodnevni rad korisnicima, ali ove opcije nisu potrebne jednoj embedded distribuciji. Busybox ih odbacuje. I kreira samo listu osnovnih argumenata koje možemo da prosljeđujemo alatu (tj. najčešće korištene opcije). Možemo reći da su alati u busybox-u minimalne verzije standaradnih alata u GNU/Linux-u.

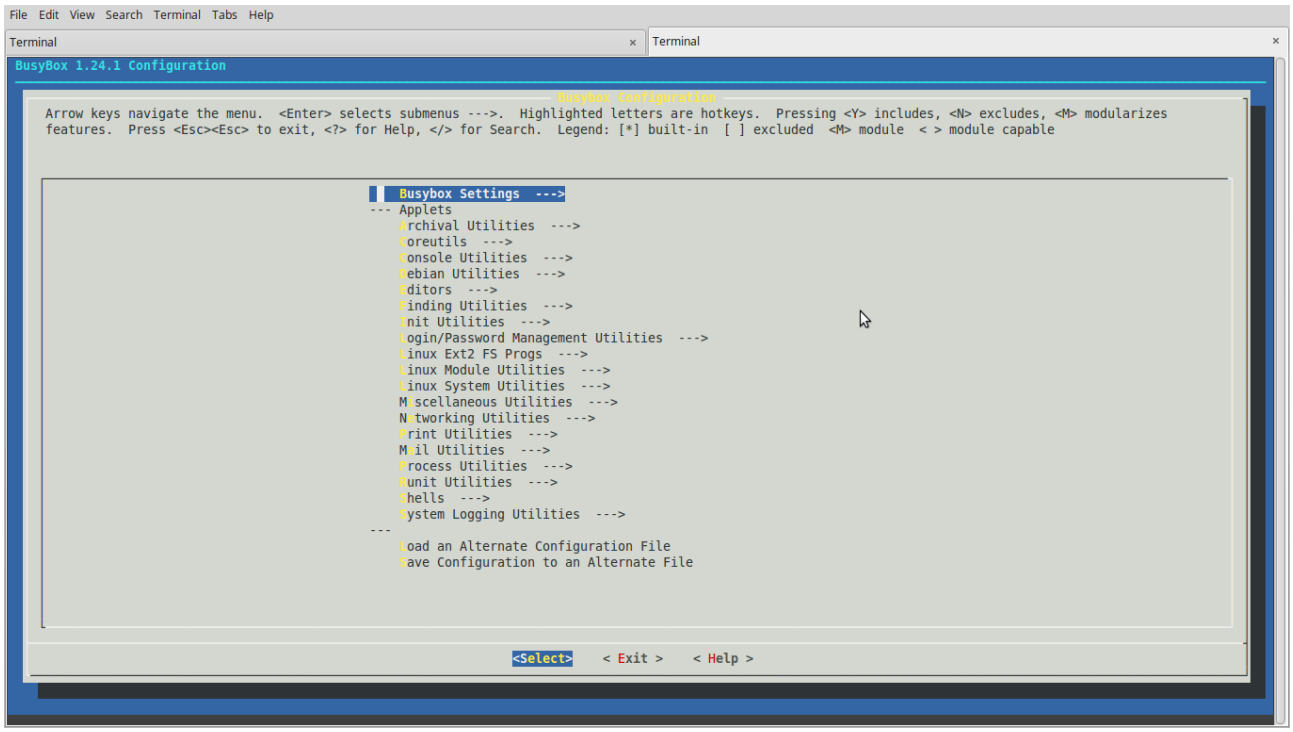
Glavna razlika između GNU alata i busybox je u tome što busybox ne prati nikakav standard kada je izrada alata u pitanju, tako da sisteme koje sa njim izrađujemo ne možemo posmatrati kao POSIX orijentisane sisteme.

Slično kao i linux kernel i busybox možemo podešavati kroz grafički interfejs.

Cross compile-iranje alata busybox




```
make ARCH=arm menuconfig
```



Nakon odabira alata potrebnih u root file sistemu započinje se proces kros kompajliranja.

```
[root@localhost]#  make ARCH=arm CROSS_COMPILE=/opt/toolchains/arm-eabi-4.6/ -j2
[root@localhost]#  make install
```

Busybox kreira osnovnu strukuturu root file sistema. Nakon završetka procesa kompajliranja izvršni fajl se nalazi u busybox direktoriju _install, koji se kreirao nakon izvršne komande make install:

bin	Svi alati u folderu bin/ izraženi su preko soft linka prema busybox izvršnom fajlu (prečica prema funkciji u biblioteci) - osnovni alati Primjer: busybox <ime_binary_fajla> -> busybox mkdir -> busybox
linuxrc	modul za integrisanje device driver-a u rane faze boot procesa
sbin	sistemski alati generisani od strane busybox-a
usr	korisnički alati generisani od strane busybox-a

Konstrukcija ostatka root file sistema

Da bismo imali funkcionalan operativni sistem zajedno sa aplikacijama ne smijemo da se oslonimo samo na fajlove koji su generisani od strane busybox-a, već moramo da popunimo naš sistem odgovarajućim konfiguracionim fajlovima i da uzdignemo strukturu rootfs-a.

Prvi korak: kreiranje direktorija koji nedostaju za osnovnu strukturu rootfs po FHS standardu:

```
[root@localhost]#  mkdir dev
[root@localhost]#  mkdir dev/pts
[root@localhost]#  mkdir etc
[root@localhost]#  mkdir etc/init.d
[root@localhost]#  mkdir lib
[root@localhost]#  mkdir mnt
[root@localhost]#  mkdir opt
[root@localhost]#  mkdir proc
[root@localhost]#  mkdir root
[root@localhost]#  mkdir sys
[root@localhost]#  mkdir tmp
[root@localhost]#  mkdir var
[root@localhost]#  mkdir var/log
```

Linux komunicira sa hardverom pomoću device nodes-a. Kernel može sam da kreira ove node-ove, ali preporuka je da se oni ručno kreiraju tj. one osnovne bez kojih kernel ne može da funkcioniše.

Komanda koja kreira device nodes uz pomoć alata mknod:

```
[root@localhost]#  mknod dev/console c 5 1
[root@localhost]#  mknod -m 666 /dev/null c 1 3
[root@localhost]#  mknod -m 666 /dev/zero c 1 5
[root@localhost]#  mknod -m 666 /dev/ptmx c 5 2
[root@localhost]#  mknod -m 666 /dev/tty c 5 0
[root@localhost]#  mknod -m 444 /dev/random c 1 8
[root@localhost]#  mknod -m 444 /dev/urandom c 1 9
[root@localhost]#  chown -v root:tty /dev/{console,ptmx,tty}
```

dev	Direktorij se popunjava u trenutku boot-a; kada se montira devtmpfs i pronađe ostatak device-a koji su prisutni u sistemu. Dio uređaja potrebnih za rad kernela definisan je ručno u prethodnom pasusu, a ostatak će se pojaviti u toku startup procesa. Ručno kreiranim device fajlovima ubrzava se startup proces. Direktorij se popunjava incijalizaciom runlevel-a uz pomoć komande koja se nalazi u /etc/init.d/rcS: mount -n -t tmpfs mdev /dev mount -t devpts devpts /dev/pts echo /sbin/mdev > /proc/sys/kernel/hotplug

proc	Specijalni file sistem u Linux orjentisanim operativnim sistemima koji daje podatke o procesima i drugim parametrima sistema. Organizovan je kao struktura fajlova koja pruža jednu vrstu interfejsa koja obezbjeđuje direktan pristup internim strukturama unutar kernel-a. /proc filesystem je montiran na direktorij /proc koji se nalazi u osnovi root file sistema. Direktorij se popunjava incijalizaciom runlevel-a uz pomoć komande koja se nalazi u /etc/init.d/rcS: mount -t proc proc /proc
sys	Virtuelni fajl sistem koji eksportuje informacije o podsistemima i device driverima unutar kernela. Kreira virtuelne fajlove koji služe za razmjenu informacija između userspace i kernelspace-a Daje mogućnost da iz userspace-a kontroliše hardware. Fajlovi koji su zapisani u sysfs ujedno se koriste i kao konfiguracije uređaja pri startup-u. Direktorij se popunjava incijalizaciom runlevel-a uz pomoć komande koja se nalazi u /etc/init.d/rcS: mount -t sys sys /sys

Sistem init

Kada kernel završi sa boot procesom, predaje kontrolu alatu init, koji se nalazi u /sbin/init direktoriju. Init određuje runlevel u kom će linux kernel da se izvršava i definiše određenja stanja sistema.

Konfiguracija za init alat se nalazi u /etc/inittab fajlu.

Konfiguraciju korisnik piše ručno: /etc/inittab

```
::sysinit:/etc/init.d/rcS

#Serijska konzola
ttyO2::respawn:/sbin/getty -L ttyO2 115200 vt100

#Definisanje više konzola
#tty2::askfirst:-/bin/sh
#tty3::askfirst:-/bin/sh
#tty4::askfirst:-/bin/sh

#/sbin/getty poziv za izabrane ttys terminale
#tty4::respawn:/sbin/getty 38400 tty5
#tty5::respawn:/sbin/getty 38400 tty6
::respawn:/sbin/getty -L tty02 115200 vt100

#Primjer kako postaviti serijsku konzolu uz pomoć alata getty (u svrhu terminala)
#::respawn:/sbin/getty -L ttyS0 9600 vt100
```

```
#::respawn:/sbin/getty -L ttyS1 9600 vt100
#Komanda koja restartuje init
::restart:/sbin/init

#Stvari koje se izvršavaju prije reseta
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
::shutdown:/sbin/swapoff -a
```

Runlevel

Linux mora imati jedan runlevel, a može da ih ima više. Prema standardu postoje 6 runlevel-a, mada u embedded uređajima često je prisutan samo jedan runlevel.

Po alatu System-V-Init postoji 6 radnih nivoa - režima (runlevel).

- Runlevel 0 = Gašenje
- Runlevel 1 = Single-user
- Runlevel 2 = Multi-user bez mreže
- Runlevel 3 = Multi-user sa mrežom
- Runlevel 4 = Korisnički definisani runlevel
- Runlevel 5 = Multi-user sa mrežom i automatskim X-startom
- Runlevel 6 = Reboot

Konfigurisan je samo jedan runlevel definisan u fajlu /etc/init.d/rcS.
Sadržaj fajla definiše korisnik: /etc/init.d/rcS

```
#!/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel
#Mount procfs,sysfs and pts devices

echo 'mount procfs'
mount -t proc proc /proc
echo 'mount sysfs'
mount -t sysfs sysfs /sys
echo 'mount tmpfs'
mount -n -t tmpfs mdev /dev
echo 'mount pts devices'
mount -t devpts devpts /dev/pts
echo 'enable hotplug'
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
echo 'set loopback device'
ifconfig  lo 127.0.0.1 netmask 255.0.0.0
trap : INT QUIT TSTP
echo 'mount sd partitions'
mount -t auto /dev/mmcblk0p1 /media/sd1/
mount -t auto /dev/mmcblk0p2 /media/sd2/
```

```
mount > /etc/mtab
echo 'Start telnetd'
telnetd
echo ''
echo 'START UP is DONE'
sync
```

/etc/fstab

Definisanje kako se block device (diskovi, virtuelni fajl sistemi) montiraju i konfigurišu.

U jednoj liniji je opisan jedan fajl sistem.

```
# /etc/fstab
# device            directory       type        options
#/dev/nfs           /               nfs         defaults
/dev/proc           /proc           proc        defaults        0 0
#none               /dev/pts        devpts      mode=0622       0 0
/dev/sys            /sys            sysfs       defaults        0 0
none                /dev/shm        tmpfs       defaults,size=100M  0 0
```

/etc/group - definisane su korisničke grupe.

```
root:x:1000
tty:x:1:
sshd:x:4:
mysql:x:2:
nginx:x:1
```

/etc/hosts - konfiguracija hostova

```
127.0.0.1       localhost
```

/etc/hostname - ime na mreži (proizvoljno)

Ostali fajlovi u etc/ direktoriju

etc/	
fstab	Fajl se koristi za definisanje načina montiranja diskova, block devices i udaljenih filesystema.
hosts	Informacije o hostovima koji se pojavljuju na mreži; konfigurisanjem ovog fajl-a može se izbjegnuti potreba za DNS serverom u malim mrežama.
host.conf	Resolver
init.d	Direktorij u kojem se pohranjuju konfiguracione skripe od runlevel-a.
nsswitch.conf	The Name Service Switch (NSS)
shadow	Sadrži enkriptovane lozinke i druge informacije o korisnicima, kao što je vremenska određenost lozinke podaci o nalogu; /etc/shadow je čitljiv jedino od strane root korisnika.
group	Fajl koji opisuje korisničke grupe u sistemu.
hosts.allow	Kontrola pristupa hostu - dozvoljeni hostovi.

hosts.deny	Kontrola pristupa hostu - zabranjeni hostovi.
inittab	Konfiguracija za init startup deamon.
issue	Poruka koja se ispisuje prije log-in procesa.
passwd	Pristupne lozinke od korisnika u kriptovanom obliku.
shells	Putanje do dostupnih shell alata.
gshadow	Enkriptovani pristupni podaci o korisnicima, vlasnička prava nad grupama. Ovaj fajl može čitati samo root korisnik.
modules	Imena kernel modula koji treba da se učitaju u toku boot procesa.
resolv.conf	Konfiguracioni fajl za DNS resollver, koji omogućava C bibliotekama pristup dns serveru.
ld.so.cache	Putanja na kojoj dijeljene biblioteke smještaju svoj keš.
motd	Poruka koje se ispisuje nakon pristupa sistemu.
hostname	ime na mreži
ld.so.conf	Dijeljene biblioteke imaju svoju standardnu putanju u linux kernel-u, kroz ovaj fajl određujemo dodatne putanje na koje možemo da smještamo biblioteke.
mtab	Informacije o fajlsistemima.
services	Opis mrežnih servisa i portova koje koriste.

Kreiranje VAR direktorija

```
[root@localhost]#  mkdir var
[root@localhost]#  mkdir var/backups
[root@localhost]#  mkdir var/cache
[root@localhost]#  mkdir var/lib
[root@localhost]#  mkdir var/lock
[root@localhost]#  mkdir var/log
[root@localhost]#  mkdir var/spool
[root@localhost]#  mkdir var/tmp
[root@localhost]#  mkdir var/www
[root@localhost]#  mkdir var/www/html
```

var/	
backups	rezervne kopije
cache	procesorski keš
lib	biblioteke od web servisa
lock	programsko zaključavanje uređaja i drugih resursa koji se dijele između više aplikacija
log	log fajlovi
spool	podaci koji će kasnije doći na obradu
tmp	temporalni fajlovi od web servisa
www	virtuelni hostovi od web server-a

www/html	virtuelni hostovi od web server-a

Kreiranje home, lib, mnt, opt, root direktorija

```
[root@localhost]#  mkdir home
[root@localhost]#  mkdir lib
[root@localhost]#  mkdir mnt
[root@localhost]#  mkdir opt
[root@localhost]#  mkdir root
```

home	home direktorij od svih user-a u sistemu
lib	sistemske biblioteke
mnt	montirane particije
opt	Ovaj direktorij je rezervisan za sve programe i dodatke koji nisu pokriveni standardnom instalicijom.
root	home direktorija od root korisnika

InitramFS

Initramfs predstavlja tehniku pomoću koje je moguće zapakovati minimalni root file sistem zajedno sa kernelom u jedan Image fajl i reći bootloader-u da ga pročita sa eksterne memorije i smjesti direktno u RAM. Postoje dvije vrste ove tehnike: prva definiše ramdisk fiksne veličine i smješta ga u radnu memoriju, a u drugoj vrsti definišemo initramfs tako da bude podesiv i može da uzima onoliko prostora u ramu koliko mu je potrebno.

Postoji više razloga za korištenje ramdiskova:

- moguće je izvršavanje operacija prije nego što se montira glavna root particija,
- samim tim što se sistem nalazi u RAM-u povećavaju se performanse,
- distribucije koje su zapakovane na ovaj način mogu da funkcionišu i ako tvrdi disk nije prisutan zato je initramfs tehnika koju koriste sve live distribucije koje se izvršavaju sa prenosivih medija,
- nalazi čestu primjenu u embedded uređajima jer je moguće brzo da testiramo interfejse i vršimo debug-ing u sistemu.

Treba biti oprezan sa upotrebom ramdiskova, jer sve konfiguracije koje se promijene u toku rada u live distribuciji ne ostaju zapamćene i potrebna je redudancija u vidu tvrdog diska kako bi se mogle zapamtiti promjene.
U ram disk se po pravilu ne stavljaju sistemski servisi, demoni i aplikacije sa visokim nivoom apstrakcije. Cilj ramdiska je da ostane što manje veličine kako se ne bi usporio startup sistema. Ostale korisničke aplikacije biće jednostavno dostupne montiranjem particije na kojoj se one nalaze (nisu potrebne prilikom startup-a sistema).

Proces kreiranja initramfs-a

```
[root@localhost]#  cd busybox/install/
[root@localhost]#  find . | cpio -H newc -o > ../initramfs.cpio
[root@localhost]#  cd ..
[root@localhost]#  cat initramfs.cpio | gzip > initramfs.cpio.gz
[root@localhost]#  mkimage -n 'Ramdisk Image'  -A arm -O linux -T ramdisk -C gzip -d
initramfs.cpio.gz initramfs.uImage
```

Kopiranje biblioteka u root file sistema

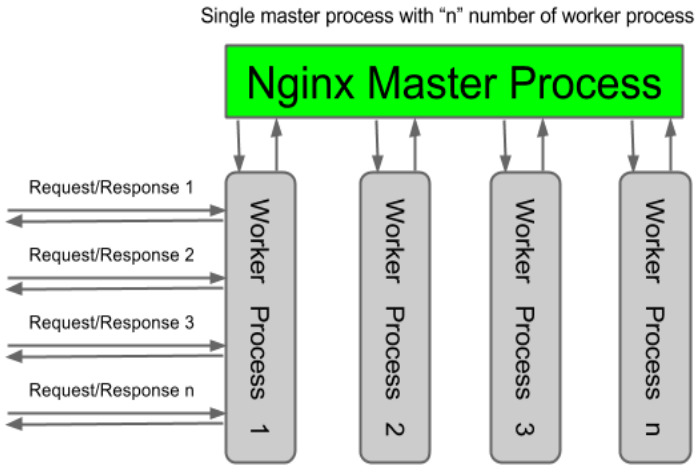
Sve biblioteke koje su se koristile za kompajliranje aplikacija u toku ovog procesa, treba da se nalaze u root fajl sistemu. Ove biblioteke se zovu glibc i o njima smo govorili u poglavlju o linux kernel-u. Biblioteke se kopiraju iz toolchain-a u rootfile sistem koji je kreiran.

```
[root@localhost]#  cp –r /opt/toolchains/arm-eabi /libc/lib/* . #kopiranje biblioteka
[root@localhost]#  arm-eabi-strip * #brisanje simbola nastalih u toku kompajliranja biblioteka, ovi simboli su često
potrebni programeru u toku razovoja, ali u produkciji oni se brišu
```

Dodatne aplikacije

Aplikacije koje je potrebno kros kompajlirati za ARM arhitekturu:

- **Nginx,**
- **PHP5,**
- **Mysql,**
- **Dropbear,**
- **Vsftpd.**



Nginx je web server i reversni proksi u jednom, sa podrškom za HTTP, HTTPS, SMTP, POP3 i IMAP protokole.
Karakterišu ga velike performanse i mala potrošnja memorije naročito pri velikom opterećenju. Nginx ostvaruje velike performanse zbog arhitekture samog servera, za razliku od Apache servera koji koristi process-driver pristup pri obradi zahtijeva. Nginx koristi asinhroni event-driven pristup.
Razlika između ova dva pristupa u dizajnu web servera je u tome što za svaki novi zahtjev Apache server stvara novi proces, dok Nginx koristi jedan glavni proces koji kontroliše broj radnika koji ga obrađuju (broj radnika ili potprocesa definiše korisnik).

Kako nginx nema mogućnost da stvara nove procese, svi zahtjevi se usmjeravaju na radnika koji je najmanje zauzet u datom trenutku. Asinhrono znači da više thread-ova ili radnika može da se izvršava paralelno bez blokiranja resursa koje dijele među sobom i koji su dodijeljeni serveru (ovo nije moguće u Apache server-u jer proces treba da čeka da se resurs oslobodi da bi ga neki drugi proces mogao koristiti).
Zahtjevi od više klijenata mogu da budu obrađeni od strane jednog thread-a ili radnika. Svi zahtjevi se obrađuju u event-loop-u koji se brine da svaki proces dobije onoliko procesorskog vremena koliko mu je potrebno da bi što brže obradio nadolazeće zahtjeve.

Cross compile proces nginx server-a:

Potrebne biblioteke:

libssl0.9.8 libssl-dev zlib1g zlib1g-dev lsb-base openssl libssl-dev libperl-dev libgd2-xpm-dev libatomic-ops-dev libxml2-dev libxslt1-dev python-dev

Nećemo prikazivati proces kros kompajliranja svih potrebnih biblioteka zbog same dužine koda, ali moramo biti svjesni postojanja tih biblioteka.

Konfigurišemo samo nginx server:

```
[root@localhost]#  ./configure --user=nginx --group=nginx --without-pcre --without-http_rewrite_module
[root@localhost]#  make
```

Nginx nije pogodan za kros kompajliranje, jer nema standardnu autoconf funkciju za ARM arhitekturu, zato je potrebno definisati kompajler, putanje do biblioteka i cross compiler.

```
[root@localhost]# vi objs/Makefile

TOP := $(dir $(lastword $(MAKEFILE_LIST)))
CC =   arm-eabi
CFLAGS =  -pipe  -O -W -Wall -Wpointer-arith -Wno-unused-parameter -Wunused-function -Wunused-variable
-Wunused-value
CFLAGS += arm-eabi/libray_dependency/include      #putanja do library-ija potrebnih nginx server-u za kompajliranje
```

Čišćenje starih objektnih fajlova i stvaranje novih prema novoj konfiguraciji:

```
[root@localhost]# find . -name "*.o" | xargs rm -f
[root@localhost]# make.    #Izvršni fajl se nalazi na lokaciji objs/nginx
```

Korisnička konfiguracija.

```
[root@localhost]#  mkdir -p /var/www/test/public_html
[root@localhost]#  chown -R www-data:www-data /var/www/test/public_html
[root@localhost]#  chmod 755 /var/www
[root@localhost]#  cp /etc/nginx/sites-available/default /etc/nginx/sites-available/test.com
[root@localhost]#  nano /etc/nginx/sites-available/test.com
```

```
server {
      listen   80;
      #listen   [::]:80 default ipv6only=on;
      root /var/www/example.com/public_html;
      index index.html index.htm;
      server_name example.com;
}
```

```
[root@localhost]#  ln -s /etc/nginx/sites-available/test.com /etc/nginx/sites-enabled/test.com
```

Cross compile proces programskog jezika PHP5:

Potrebne biblioteke:
libxml2

Kros kompajliranje libxml2 biblioteke, ova biblioteka je potreba kako bi se mogao kompajlirati php za arm arhitekturu.

```
[root@localhost]#  export ARCH=arm
[root@localhost]#  export CC=arm-eabi-gcc
[root@localhost]#  export AR=arm-eabi-ar
[root@localhost]#  export LD=arm-eabi-ld
[root@localhost]#  export RANLIB=arm-eabi-ranlib
[root@localhost]#  ./configure --host=arm-none-linux-gnueabi --without-python --without-iconv --without-zlib
[root@localhost]#  make
[root@localhost]#  make install
```

Kako nginx server nije modularan i nema mogućnost učitavanja php modula, naknadno se kompajlira programski jezik php i povezuje sa nginx server-om.

```
[root@localhost]# make clean
[root@localhost]# export  CC=arm-eabi-gcc
[root@localhost]# export CXX=arm-eabi-g++
[root@localhost]# export  AR=arm-eabi-ar
[root@localhost]# export  LD=arm-eabi-ld
[root@localhost]# export RANLIB=arm-eabi-ranlib
.[root@localhost]# /configure  --host=arm-none-linux-gnueabi --target=arm --without-sqlite3 --without-pdo-sqlite
--without-pear --enable-simplexml --disable-mbregex --enable-sockets --enable-fpm --disable-opcache --enable-libxml
--without-zlib --with-libxmldir=<putanja do libxml-a> --disable-all
[root@localhost]# make
[root@localhost]# make install
```

Povezivanje programskog jezika PHP i nginx server-a:

Nakon instaliranja nginx server-a, na putanji /etc/nginx/sites-available/default nalazi se podrazumijevano podešavanje za svaki serverski blok. Svaki blok treba da ima podešavanja za korištenje php-a. U teksu je prikazan primjer konfiguracije koja koristi samo statični html i konfiguracije koja ima mogućnost korištenja php-a:

server blok konfiguracija bez programskog jezika php:

```
server {
   listen 80 default_server;
   listen [::]:80 default_server ipv6only=on;
   root /usr/share/nginx/html;
   index index.html index.htm;
   server_name localhost;
   location / {
      try_files $uri $uri/ =404;
   }
}
```

server blok konfiguracija sa programskim jezikom php:

```
server {
   listen 80 default_server;
   listen [::]:80 default_server ipv6only=on;
   root /usr/share/nginx/html;
   index index.php index.html index.htm;
   server_name server_domain_name_or_IP;
   location / {
      try_files $uri $uri/ =404;
   }
   error_page 404 /404.html;
   error_page 500 502 503 504 /50x.html;
   location = /50x.html {
      root /usr/share/nginx/html;
   }
   location ~ \.php$ {
      try_files $uri =404;
      fastcgi_split_path_info ^(.+\.php)(/.+)$;
      fastcgi_pass unix:/var/run/php5-fpm.sock;
      fastcgi_index index.php;
      fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
      include fastcgi_params;
   }
}
```

MySQL server

Za baze podataka koristi se standardna relaciona baza podataka MySQL.

Cross compile proces MySQL servera

Potrebni library: libncurses potreban za kompajliranje mysql servera.

```
[root@localhost]#  tar xvf ncurses-5.7.tar.gz
[root@localhost]#  cd ncurses-5.7
[root@localhost]#  CC=arm-linux-gcc CXX=arm-linux-g++ ./configure --host=arm-linux-gnu
--prefix=/usr
[root@localhost]#  make
[root@localhost]#  make install DESTDIR=/home/projects/libncurses-compiled
```

#Kreiraju se dvije kopije source koda

```
[root@localhost]#  mkdir mysql-src-x86 mysql-src-arm
[root@localhost]#  tar xvf mysql-5.5.37.tar.gz
[root@localhost]#  mv mysql-5.5.37 mysql-src-x86
[root@localhost]#  tar xvf mysql-5.5.37.tar.gz
[root@localhost]#  mv mysql-5.5.37 mysql-src-arm
```

```
[root@localhost]#  cd mysql-src-x86/BUILD
[root@localhost]#  ./compile-amd64-max
```

Kreiranje CMake toolchain fajl *mysql-src-arm/build/mysql_odroid.cmake*:

```
SET(CMAKE_SYSTEM_NAME Linux)
SET(CMAKE_SYSTEM_VERSION 1)
SET(CMAKE_C_COMPILER /opt/toolchains/usr/bin/arm-eabi-gcc)
SET(CMAKE_CXX_COMPILER /opt/toolchains/usr/bin/arm-eabi-g++)
SET(CMAKE_FIND_ROOT_PATH /opt/buildroot/usr/arm-linux/sysroot /home/projects/rootfs)
SET(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
SET(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
SET(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
SET(CMAKE_LIBRARY_PATH ${CMAKE_LIBRARY_PATH} /home/projects/libncurses-
compiled/usr)
SET(STACK_DIRECTION 1)
SET(WITH_UNIT_TESTS OFF)
```

Skripta 1: *mysql-src-arm/BUILD/doconfig*:

```
#!/bin/bash
rm CMakeCache.txt
#path=`dirname $0`
cmake -DCMAKE_TOOLCHAIN_FILE=mysql_at91sam9260.cmake ..
```

Skripta 2: *mysql-src-arm/BUILD/domake*:

```
#!/bin/bash
export PATH=$PATH:`pwd`/extra
export PATH=$PATH:`pwd`/../scripts
# These scripts are needed to be copied from the x86 dir every time before make
# runs because they will be overwritten with ARM versions after a while.
cp ../../mysql-5.5.37-x86/sql/gen_lex_hash sql/gen_lex_hash
export PATH=$PATH:`pwd`/sql
cp ../../mysql-5.5.37-x86/storage/perfschema/gen_pfs_lex_token
storage/perfschema/gen_pfs_lex_token
export PATH=$PATH:`pwd`/storage/perfschema/
```

Sa terminala se izvršavaju sljedeće opcije i nastavljamo proces kros kompajliranja.

```
[root@localhost]#  make
[root@localhost]#  domake
[root@localhost]#  make install DESTDIR=$ROOTFS
[root@localhost]#  cd /usr/local/
[root@localhost]#  ./scripts/mysql_install_db
[root@localhost]#  chown -R mysql:mysql data /var/lib/mysql
```

Dropbear

Mali SSH server i klijent, namijenjem embedded uređajima i svim sistemima koji su POSIX orjentisani. Daje mogućnost korisniku da sa udaljene lokacije pristupa i kontroliše sistem. Dropbear je procesu kompajliranja moguće smanjiti do 200kb veličine što ga čini jednim od najmanjih ssh alata u industriji, otvorenog je koda i svako može da se priključi razvoju.
.

Cross compile proces Dropbear

```
[root@localhost]#  export CC=arm-linux-gnueabi-gcc
[root@localhost]#  export CPP=arm-linux-gnueabi-cpp
[root@localhost]#  ./configure --host=arm-linux --disable-zlib
[root@localhost]#  make -j2
[root@localhost]#  make scp -j2
```

Vsftp

FTP server za razmjenu fajlova sa serverom i kao podrška ostalim servisima u sistemu koji zahtjevaju FTP pristup, kao što su CMS sistemi. VSFTP je jedan od najmanjih, najsigurnijih i najbržih web servera koji se koriste na ugrađenim sistemima, prije svega zbog jednostavnog procesa kros kompajliranja i mogućnosti da se izvršava na mnogo arhitektura. Napisan je u čistom C jeziku.

Cross compile proces vsftpd

```
[root@localhost]#  make CC=arm-linux-gnueabi-gcc -j2
[root@localhost]#  make install
```

Msmtp

E mail alat sa trivijalnim funkcija, daje mogućnost programskom jeziku PHP da šalje mail-ove preko smtp protokola. Ova opcija je često potrebna web aplikacijama. Sam alat je veoma malen i ne prelazi veličinu od 100kb. U rootfile systemu, potrebno je definisati u php.ini fajlu koji mail alat da koristi. Konfiguracija se nalazi na putanji /etc/php5/php.ini, gdje postavkom varijable sendmail, koja je već definisana unutar konfiguracije, postavljamo podrazumijevani mail alat.

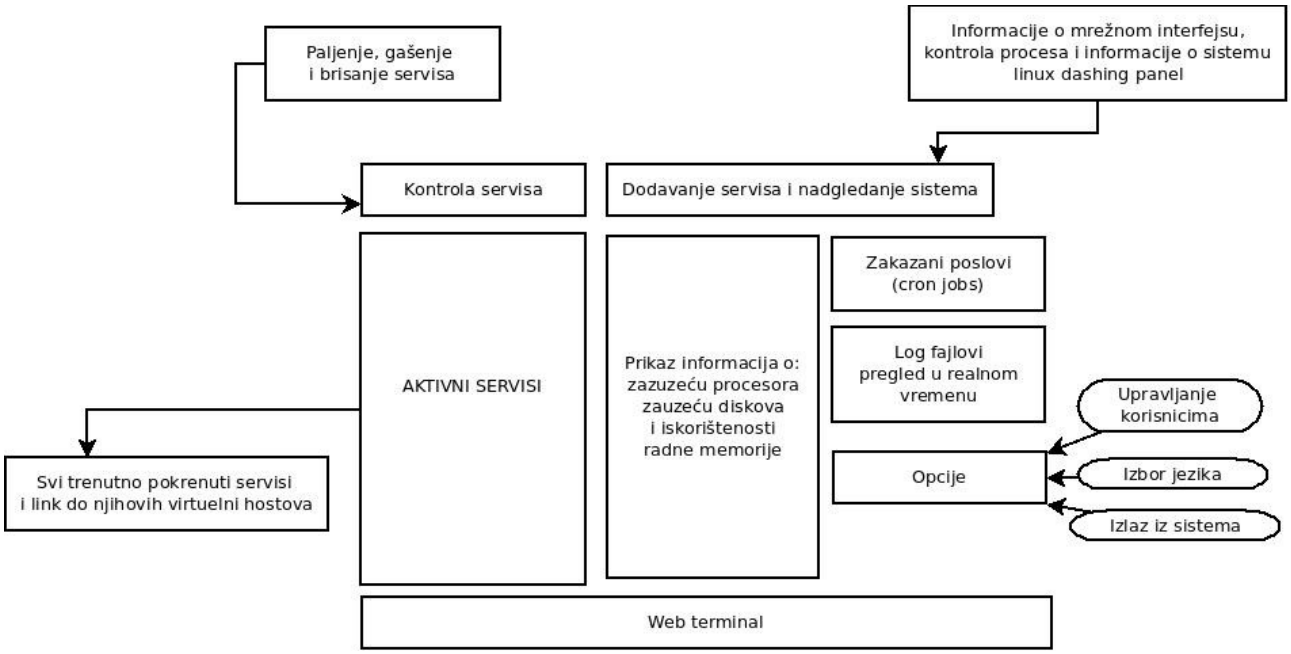
Cross compile proces msmtp

```
[root@localhost]#  export CC=arm-linux-gnueabi-gcc-4.7
[root@localhost]#  export CXX=arm-linux-gnueabi-g++
[root@localhost]#  export LD=arm-linxu-gnueabi-ld
[root@localhost]#  ./configure --host=arm-linux --disable-ssl --disable-gsasl --disable-nls -without-libidn --with-ssl=no
[root@localhost]#  make -j2
[root@localhost]#  mkdir output
[root@localhost]#  make DESTDIR=/home/spatifilum/Desktop/diploma/rootfs-work/output install
```

Custom Web panel

Zbog potrebe za kontrolom servera sa udaljenje lokacije pored SSH server-a koji obezbjećuje mogućnost kontrole preko terminala dizajniran je Web panel nazvan HanolinaBox.
Web panel je razvijen uz pomoć programskih jezika PHP, skriptnih jezika JavaScript, Bash-a i opisnih jezika kao što su HTML i CSS. Za rad web panela nije potrebna baza podataka (sam panel je nezavisan od baze ali servisi koje on učitava mogu zahtijevati bazu podatka). Većina današnjih remote web panel-a oslanjaju se na upotrebu CMS sistema i Mysql baze podataka. Kao programer izabrao sam drugi pristup jer smatram da ovakav način dizajna sprječava korisnike da koriste web panel na malim ugrađenim sistemima, koji nemaju dovoljno resursa za pokretanje MySQL servera i nepostojanja sistema za baze podataka koji je dovoljno malen da nađe primjenu u ugrađenim sistemima. HanolinaBox panel koristi tekstualne datoteke kao zamjenu za bazu podataka.

Dijagram arhitekture grafičkog interfejsa HanolinaBox web panel-a:



HanolinaBox ima mogućnost prikaza log fajlova u realnom vremenu. Takođe na sličan način ispisuje informacije o sistemu koje korisniku daju uvid koliko je zauzeće procesora, radne memorije, tvrdih diskova.
Kako bi se omogućio prikaz ovih informacija korišteni su ajax pozivi, php i bash skripte.
Preko jquery funkcije poziva se php fajl (fajl se poziva svake sekunde), u php fajlu se nalaze funkcije koje preko funkcije shell_exec() izvršavaju komande na terminalu. Kada php dobije tražene vrijednosti on kao request back vraća html kod u kom se nalaze date vrijednosti i ispravno podešen html tag koji se prikazuje u browser-u.

Prikaz koda opcije ram monitoring, koja prikazuje zauzeće radne memorije u realnom vremenu.

ajax poziv prema php fajlu, poziv se osvježava svake sekunde

```
<script type="text/javascript">
var auto_refresh = setInterval(
function (){
$('#load_ram_meter').load('ram_meter.php').fadeIn("slow");
}, 1000);
</script>
<div id="load_ram_meter"> </div>
```

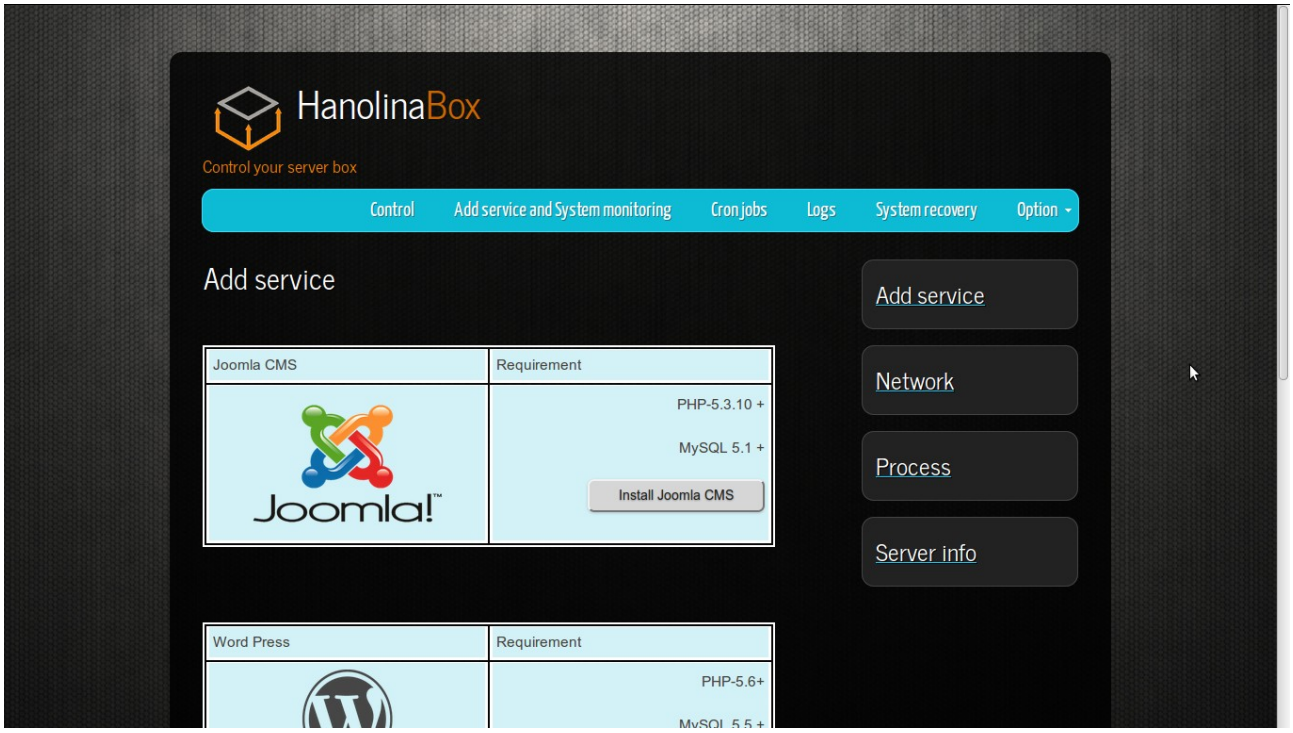
request

request back

php fajl uz pomoć funkcije shell_exec vraća rezultate i ispisuje ih na ekran

```
<?php
$mem_free=shell_exec("cat /proc/meminfo | grep MemFree | grep -o '[0-9,.]*'");
$mem_total=shell_exec("cat /proc/meminfo | grep MemTotal | grep -o '[0-9,.]*'");
$mem_used=shell_exec("busybox top -b n 1 | grep  Mem | grep -o [0-9]* | sed -n '1 p'");
echo '<meter value=', $mem_used  ,'max=' , $mem_total, ' min="0"> </meter> ';
print_r($mem_used);
echo "/ " , $mem_total , "kb";
?>
```

Cilj web panela je da omogući korisnicima koji nisu IT tehničari da na jednostavan način konfigurišu njima potrebne servise i održavaju server sa udaljene lokacije.



Integrisani Web servisi

HanolinaBox panel daje mogućnost instaliranja određenih web aplikacija putem grafičkog interfejsa gdje pritiskom na dugme bivamo proslijeđeni na instalaciju web servisa. Podržani web servisi se nalaze u tabeli:

Cloud	Own Cloud
Web terminal	ShellinaBox
Webmail	RoundCube
Content Managament System	Joomla, Word Press, Drupal, Wolf
Version Control System	gitphp, gitlist, websvn
Learning Managament System	Moodle
Wiki	Doku Wiki, MediaWiki
E-commerce	Drupal-kickstart, Magento, Opencart
Utility	linux dash panel, ntop1, phpcpu,
Forum	phpbb, punbb

Zaključak i pravci daljeg rada

Vidjeli smo na koji način radi i kako utiče ARM arhitektura na svijet prenosnih i energetski efikasnih uređaja. Nove mogućnosti rada procesora u klasterima i konfiguracija različitih procesora povećavaju iskoristivost starih jezgri i na ekološki način kombinuju staro i novo. Nove mogućnosti ARM procesora kao što je 40-bitna memorijska alokacija proširuju mogućnosti hardvera i donose nova pravila u svijet virtuelizacije na ARM arhitekturi.
Linux kao jedan od primarnih operativnih sistema koji pokriva najveći broj arhitektura je ujedno i nakorišteniji sistem na ARM arhitekturi. Kroz praktične primjere je pokazano kako se gradi GNU/Linux orijentisani embedded sistem. Konfiguracija sistema, kros kompajliranje i prilagođavanje software-a novoj razvojnoj ploči. Razvojem open-source alata razvija se i svijet embedded tehnologija. Kombinovanjem Linux-a i ARM arhitekture moguće je izgradi mnoge vrste jeftinih proizvoda. Server koji je izgrađen u svrhu diplomskog rada košta oko 150$ + ssd disk.
Jedini problem koji je trajno prisutan u embedded svijetu je nedostatak framework-a za razvoj aplikacija na ARM arhitekturi. Takođe na ovoj arhitekturi problem predstavlja prikaz grafike, gdje je većina rješenja komercijalnog karaktera. Iz tog razloga većina programera isporučuje embedded aplikacije u vidu web servisa. Linux koliko god bio portabilan i dobro dokumentovan i dalje je kompleksan sistem za korištenje i prilagođavanje.
Nakon prilagođavanja Linux na razvojnu ploču i definisanja FTD-a, prikazan je proces izgradnje rootfile sistema. Rootfile sistem je hijerahija fajlova koju koriste Linux orijentisani sistemi. Potrebno je posvetiti vrijeme izgradnji rootfile sistema, jer je on osnova kvalitetnog korisničkog interfejsa.

Sve aplikacije korištene u embedded uređajima potrebno je kros kompajlirati za datu arhitekturu. Od aplikacije zavisi koliko je kompleksan proces kros kompajliranja, neke aplikacije su prilagođene ovom procesu, a neke nisu. Konfiguraciom aplikacija stvoren je funkcionalan embedded sistem.

Embedded sistem koji je izgrađen predstavlja Small Business Server, riješenje namjenjeno malim preduzećima i kancelarijama koje imaju potrebu za ekonomičnim uređajima.
Cilj diplomskog rada ne predstavlja samo prikaz izgradnje servera, već pokazuje da tehnologija može da se razvija bilo gdje na svijetu. Produkcija ovakvih proizvoda je ekonomična i jednostavna za održavanje što je čini idealnom za implementaciju na lokalnim tržištima.
Zanimljiva činjenica je da danas hardver počinje da se prilagođava softveru. Ovo se dešava zbog manje mogućnosti greške pri dizajniranju hardvera i kompleksnosti razvoja softvera za nove arhitekture.

Dolaskom ARM64 arhitekture ostavljena je mogućnost proširenja performansi novih uređaja baziranih na ovoj arhitekturi. Postepeno će se povećavati učešće ARM baziranih procesora u serverskim poslovima.

LITERATURA

Karim Yaghmour , *Building Embedded Linux Systems,* 3rd edition, O'Reilly Media, 2008

Christopher Hallinan, *EmbeddedLinuxPrimer*, 2nd edition, Prentice Hall, 2006

Greg Kroah-Hartman, *Linux Kernel in a Nutshell*, O'Reilly Media, 2006

Jonathan Corbet, Alessandro Rubini i Greg Kroah-Hartman, *Linux Device Drivers*, 3th, O'Reilly Media, 2005

Thomas Petazzoni, *Porting a linux kernel to a ARM board,* Free Electrons, 2010
,
Gerard Beekmans , Matthew Burgess i Bruce Dubbs, *Linux From Scratch*, verzija 7.3, 2013, Community edition

Rusty Russell , Daniel Quinlan i Christopher Yeoh, *Filesystem Hierarchy Standard, Filesystem Hierarchy Standard Group ,*2004

Adam Haeder, Stephen Addison Schneiter, Bruno Gomes Pessanba i James Stanger, *LPI Linux Certification in Nutshell,* O'Reilly Media*,* 2010

ARM developers team, *ARM Architecture Reference Manual*, ARM Holdings, 2005

ARM developers team, *Cortex -A15 Technical Reference Manual,* Revision: r2p1*,* ARM Holdings, 2011

ARM developers team, *Cortex -A7 MPCore Technical Reference Manual*, Revision: r0p3, ARM Holdings, 2012

The Linux Kernel Organization, Linux kernel
https://www.kernel.org/

Free Software Fondation, Wolfgang Denk
http://www.denx.de

Free Software Fondation, Bruce Perens, Denys Vlasenko, Rob Landley
https://www.busybox.net/

Nginx Inc. Igor Sysoev
http://nginx.org

MySQL AB, Oracle Corporation
https://mysql.com

Free Software Foundation, Martin Lambers
http://msmtp.sourceforge.net/

Free Software Fondation, Chris Evans
https://security.appspot.com/vsftpd.html

Odroid developer team, Hardkernel

http://odroid.com

Free Software Fondation, Matt Johnston
https://matt.ucc.asn.au/dropbear/dropbear.html

Zend Technologies, Rasmus Lerdorf
https://.php.net/