УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Јурица Станојковић

КРЕИРАЊЕ И ВЕРИФИКАЦИЈА ОПЕРАТИВНОГ СИСТЕМА ЛИНУКС – СТУДИЈА СЛУЧАЈА НА ПРИМЕРУ АРХИТЕКТУРА MIPS32 И MIPS64

МАСТЕР РАД

Нови Сад, 2015

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Дипломски – мастер рад	
Аутор, **АУ**:		Јурица Станојковић	
Ментор, **МН**:		Проф. др Никола Теслић	
Наслов рада, **НР**:		Креирање и верификација оперативног система Линукс – студија случаја на примеру архитектура MIPS32 и MIPS64	
Језик публикације, **ЈП**:		Српски / латиница	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2015	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		8/88/1/8/14/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кqучне речи, **ПО**:		Линукс, креирање и верификација, Debian, MeeGo, аутоматизовано превођење пакета, MIPS32, MIPS64	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		Анализа процеса креирања и верификације система базираних на језгру оперативног система Линукс. Преглед практичних искуства стечених у раду са новим архитектурама MIPS32 и MIPS64. Области од интереса приликом првог превођења система за нове архитектуре. Верификација дистрибуција Линукс-а.	
Датум прихватања теме, **ДП**:		16.9.2014.	
Датум одбране, **ДО**:		16.1.2015.	
Чланови комисије, **КО**:	Председник:	Doc. dr Jelena Kovačević	
	Члан:	Prof. dr Nikola Čelanović	Потпис ментора
	Члан, ментор:	Prof. dr Nikola Teslić	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Master Thesis	
Author, **AU**:		Jurica Stanojković	
Mentor, **MN**:		Nikola Teslić Ph.D.	
Title, **TI**:		Building Linux distributions and verification - case study for MIPS32 and MIPS64 architectures	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2015	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		8/88/1/8/14/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		Linux , building and verification, Debian, MeeGo, automated package building, MIPS32 , MIPS64	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		Analysis of the process of creating and verifying systems based on the Linux Kernel. Overview of practical experience gained in working with new architectures MIPS32 and MIPS64. Areas of interest during the first system build for a new architecture. Verification of Linux distributions.	
Accepted by the Scientific Board on, **ASB**:		16.9.2014.	
Defended on, **DE**:		16.1.2015.	
Defended Board, **DB**:	President:	Jelena Kovačević Ph.D.	
	Member:	Nikola Čelanović Ph.D.	Menthor's sign
	Member, Mentor:	Nikola Teslić Ph.D.	

	УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА МАСТЕР РАД	Датум:

(Податке уноси предметни наставник - ментор)

СТУДИЈСКИ ПРОГРАМ:	Računarstvo i automatika
РУКОВОДИЛАЦ СТУДИЈСКОГ ПРОГРАМА:	Prof. dr Nikola Jorgovanović

Студент:	Jurica Stanojković	Број индекса:	E10558
Област:	Računarska tehnika i računarske komunikacije		
Ментор:	Prof. dr Nikola Teslić		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА МАСТЕР РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;

НАСЛОВ МАСТЕР РАДА:

Kreiranje i verifikacija operativnog sistem Linuks – studija slučaja na primeru arhitektura MIPS32 i MIPS64

ТЕКСТ ЗАДАТКА:

Zadatak rada je da uradi analizu procesa kreiranja i verifikacije operativnih sistema baziranih na kernelu Linuks. U okviru rada, a na bazi teorijske pripreme nad postojećim znanim procesima i procedurama za odabrane arhitekture, potrebno je praktično primeniti uočene koncepte za nove arhitekture MIPS32 i MIPS64. Ova dodatna praktična iskustva treba da obogate, utvrde i verifikuju postavljena znanja, kao i da ukažu na oblasti od interesa kod prvog prevođenja sistema za nove arhitekture, te kod verifikacije postojećih i budućih distribucija

Руководилац студијског програма:	Ментор рада:

Примерак за: o - Студента; o - Ментора

Zahvalnost

Kolegama koji su stručnim savetima pomogli izradu ovog rada:

Dejan Latinović
Dragoslav Šićarov
Petar Jovanović

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

APT - *Advanced Packaging Tool*

BSP - *Board Support Package*

CD - *Compact Disc*

CD-ROM - *Compact Disc Read Only Memory*

CPU - *Central Procesor Unit*

DEB - *binarni paketi sa oznakom tipa datoteke .deb*

FP - *Floating Point*

FPU - *Floating Point Unit*

GNU - *GNU's Not Unix*

GPL - *General Public License*

HTTP - *Hypertext Transfer Protocol*

LFS - *Linux From Scratch*

LL - *Load Linked*

OBS - *Open(Suse) Build Service*

OE - *OpenEmbedded*

OE-Core - *OpenEmbedded-Core*

OS - *Operating System*

RMW - *Read Modify Write*

ROM - *Read Only Memory*

RPM - *binarni paketi sa oznakom tipa datoteke .rpm*

SC - *Store Conditional*

SSL - *Secure Sockets Layer*

USB - *Universal Serial Bus*

XML - *Extensible Markup Language*

XML-RPC - *XML-Remote Procedure Call*

Yamon - *Yet Another MONitor* - ROM monitor for MIPS

1. Uvod

U ovom radu je data analiza procesa kreiranja i verifikacije operativnog sistema Linuks za ugrađene sisteme MIPS sa posebnim osvrtom na platforme za verifikovanje distribucije. Rad daje prikaz procesa stvaranja nove distribucije, kao i tipskih izazova u razvoju i verifikaciji ovakvih rešenja. Nastao je na osnovu iskustava stečenih u procesu prevođenja i verifikacije distribucije MeeGo i distribucije Debian za izvedbe MIPS32 i MIPS64 arhitekture MIPS. Ovaj rad pritom ne predstavlja korak po korak uputstvo za prevođenje distribucija MeeGo i Debian za arhitekturu MIPS, već pruža okvir koji bi mogao da posluži za kreiranje i verifikaciju i drugih distribucija operativnog sistema Linuks za željene arhitekture.

1.1 Linuks

Naziv „Linux“ se inicijalno pojavljuje kao ime za kernel (jezgro za Linuks) koji je napisao Linus Torvalds. Kako se jezgro operativnog sistema Linuks često koristilo u kombinaciji sa drugim programskim rešenjima, naročito sa onima iz projekta GNU, termin Linuks postaje šire poznat kao naziv za kombinaciju jezgra operativnog sistema Linuks i programske podrške GNU. Prvi put se koristi 1992. godine u kombinaciji sa GNU i vrlo brzo postaje najpopularnije programsko rešenje u okviru zajednice GNU. U junu 1994. godine u biltenu GNU, Linuks je spomenut kao besplatni klon UNIX-a, a projekat Debian počeo je da svoje rešenje naziva „Debian GNU/Linux“. U maju 1996. pojavio se naziv „Lignux“, predložio ga je Richard Stallmana kao ime za tip sistema, ali se ovaj termin nije ustalio. Danas se za kombinaciju jezgra operativnog sistema Linuks i programske podrške GNU koristi GNU/Linux ili samo Linux.

1.2 Linuks pod GNU GPL

Programsko rešenje koje se koristilo sa jezgrom operativnog sistema Linuks je bila programska podrška (softver) razvijena kao deo projekta GNU licencirana GNU GPL (ili kraće GPL) licencom, što znači da je programska podrška besplatna za upotrebu. Prvo izdanje jezgra operativnog sistema Linuks (Linux 0.01) u sastavu je imalo programsko rešenje bash (bash shell).

Linus Torvalds u svojim beleškama za Linuks 0.01[1] navodi da je GNU potreban da bi se Linuks koristio: „Na žalost, sam kernel ne vodi nikuda. Da bi sistem radio, potrebni su

školjka, prevodioci, biblioteke i druge stvari. Ovi različiti delovi mogu biti sa strožijom licencom (ili manje strogom). Veći deo alata koji se koriste su rešenja GNU i nalaze se pod licencom GNU copyleft. Ovi alati se ne nalaze u distribuciji.“

1992. godine predlaže da se kernel objavi pod GNU General Public License. Ova odluka je objavljena u beleškama verzije 0.12. Decembra 1992. godine verzija 0.99 je objavljena pod ovom licencom (GNU GPL).

Iako je Linuks otvorenog tipa, postoje kompanije koje ostvaruju profit od Linuksa. Najveći deo njih su članovi Open Source Development Lab (razvojna laboratorija za programska rešenja otvorenog tipa) i ulažu značajna sredstva u razvoj i poboljšanje Linuksa kako bi ga prilagodili različitim potrebama. Ovo podrazumeva donacije fizičkih uređaja za razvoj rukovaoca (eng. driver), novčane donacije ljudima koji pomažu razvoj programskih rešenja za Linuks i zapošljavanje programera za Linuks. Neki primeri su IBM i HP koji koriste Linuks na sopstvenim serverima i Red Hat koji održava i sopstvenu distribuciju.

1.3 Distribucije bazirane na Linuksu

Distribucija Linuks je član porodice operativnih sistema zasnovanih na jezgru operativnog sistema Linuks. Takve distribucije su operativni sistemi koji uključuju veliku kolekciju programskih rešenja poput procesora teksta, tabela, multimedijalne aplikacije i aplikacije zasnovane na bazama podataka. Oni se sastoje od jezgra operativnog sistema Linuks i skupa biblioteka i alata projekta GNU, sa grafikom zasnovanom na sistemu prozora X. Distribucije koje imaju malu veličinu ne sadrže X i obično sadrže kompaktnije alternative alata GNU, kao što su BusyBox, uClibc ili dietlibc.

Danas postoji više od šest stotina distribucija operativnog sistema Linuks. Preko tri stotine se aktivno unapređuje.

Budući da su jezgro operativnog sistema Linuks i većina paketa besplatni i otvorenog koda, distribucije imaju različite forme – za stone računare, prenosne računare, poslužioce (servere), mobilne telefone, tablet računare ili ugrađene sisteme.

Distribucije se mogu deliti na komercijalno podržane (poput Fedora-e (Red Hat), openSUSE (SUSE), Ubuntu (Canonical Ltd.) i Mandriva Linuks (Mandriva)) i distribucije kreirane i održavane od strane zajednice (Debian, Mageia i Gentoo).

Distribucije Linuksa mogu biti:

- Komercijalne ili nekomercijalne
- Namenjene za kućnu upotrebu, poslovnu upotrebu ili iskusne korisnike
- Za više platformi ili za specifičnu platformu
- Stvorene za poslužioce, kućne računare ili razne uređaje
- Za generalnu upotrebu ili visoko specijalizovane
- Za određenu grupu korisnika
- Stvorene primarno za sigurnost, upotrebnu vrednost, prenosivost ili sveobuhvatnost.

Različitost distribucija Linuks je prisutna zbog tehničkih, organizacionih i filozofskih razlika između prodavaca (ponuđača) i krajnjih korisnika. Svaki korisnik odgovarajućeg znanja, uz malo interesovanja može napraviti distribuciju koja podmiruje potrebe tog korisnika.

Neke od veoma poznatih distribucija operativnog sistema Linuks su:

- ArchLlinux – minimalistička distribucija čiji su krajnji korisnici iskusni ljudi, distribuciju održava zajednica okupljena oko nje.
- Debian – nekomercijalna distribucija održavana od strane zajednice, sa velikom privrženošću besplatnom sadržaju

- Knoppix – prva live CD distribucija (pokreće se sa prenosivih medija, bez instalacije), nastala od Debian-a
- Linux Mint Debian Edition (LMDE) bazirana na verziji Debian distribucije u fazi ispitivanja (testing, not stable)
- Ubuntu – popularna desktop i server distribucija, nastala od Debian-a, koju održava Canonical Ltd.
    - BackTrack – baziran na Ubuntu. Koristi se za digitalnu forenziku i ispitivanje upada.
    - Kubuntu – KDE verzija Ubuntu-a
    - Linux Mint – distribucija bazirana i kompatibilna sa Ubuntu-om, koristi školjku Gnome (Cinnamon)
    - Xubuntu – Xfce verzija Ubuntu-a
    - Lubuntu – LXDE verzija Ubuntu-a, nezahtevna verzija Ubuntu-a
- Fedora – distribucija koju je sponzoriše Red Hat
    - Red Hat Enterprise Linux – komercijalna distribucija
        - CentOS – distribucija izvedena od istih resursa koje koristi Red Hat
        - Oracle Enterprise Linux – izveden od Red Hat Enterprise Linux-a, održava ga i komercijalno podržava Oracle.
    - Mandriva – održavana od strane Mandriva-e
        - PCLinuxOS – desktop distribucija
- Gentoo – distribucija kojoj su krajnji korisnici iskusni ljudi
- Slackware – jedna od prvih distribucija operativnog sistema Linuks, napravljena 1993. i aktivno održavana od strane Patrika Volkerdinga
- openSUSE – distribucija održavana od strane Novell-a
    - SUSE Linux Enterprise

Postoje distribucije koje imaju specifične mete, kao što je distribucija OpenWrt namenjena ruterima, distribucije za bioinformatiku, Edubuntu za edukaciju i druge.

2. Kreiranje distribucija

2.1 Alati za kreiranje distribucija

Postoji više alata, koji na osnovu već gotovih distribucija, omogućavaju korisnicima izgradnju distribucija Linuksa prema sopstvenim potrebama. Razlikuju se, kako od toga koja se distribucija koristi kao polazna, tako i prema tome da li prave „sliku“ već prekonfigurisanog sistema ili čarobnjak vodi kroz proces odabira funkcionalnosti, a zatim se distribucija gradi na osnovu prikupljenih podataka uz povlačenje paketa sa udaljenog skladišta.

Pobrojaćemo neke od alata:

- Remastersys – pravi potpunu sliku sa svim ličnim podacima, ili stvara kopiju distribucije bez ikakvih ličnih podataka na njoj.
- Linux Live Scripts – omogućava stvaranje distribucije pomoću postojeće instalacije.
- Live – Magic – alat koji omogućava lako i jednostavno generisanje distribucije Debian pomoću live – helper alata koji je deo projekta Debian Live.
- Revisor – omogućava prilagođavanje ili stvaranje sopstvene distribucije bazirane na Fedori.
- Instalinux.com – pomoću čarobnjaka se odabere distribucija i njena verzija i programska rešenja koja treba da budu uključena. Krajnji proizvod će automatski stvoriti sistem pomoću paketa koje pribavlja sa mrežnog uređaja.
- SUSE Studio
- Ubuntu Customisation Kit – alat koji pomaže prilikom prilagođavanja Ubuntu distribucije.
- Reconstructor – stvara prilagođene verzije GNU/Linuks distribucija (Debian i Ubuntu)
- Pungi – alat otvorenog koda za stvaranje Fedora distribucija. Poseduje skup python biblioteka za stvaranje različitih alata.
- Builder – jednostavan alat koji omogućava preuzimanje, izdavanje i prilagođavanje, kao i ponovno stvaranje distribucija Ubuntu.
- Linux-Live – skup shell skriptova koji omogućavaju stvaranje distribucije Linuksa sa već instalirane distribucije. Kreirana distribucija će moći da se pokrene sa CD-ROM-a, memorijskog USB i drugih uređaja.

- MySlax Creator – omogućava stvaranje distribucije na računaru na kome je pokrenut operativni sistem Microsoft Windows.

Većina pobrojanih alata pruža mogućnost izgradnje distribucije na osnovu odabira funkcionalnosti već dostupnih distribucija za podržane arhitekture (platforme), međutim postavlja se pitanje šta da se radi kada je potrebno napraviti distibuciju za arhitekturu koja još uvek nije podržana, ili nije u potpunosti podržana, tom distribucijom, tj. kada ne postoje binarni paketi za željenu arhitekturu?

Tada na scenu stupaju drugačiji alati koji omogućavaju izgradnju distribucije za željenu arhitekturu na drugoj već podržanoj arhitekturi.

Kako bi krenuli u prevođenje željene distribucije za željenu arhitekturu potrebni su nam paketi izvornog koda namenjeni za izgradnju željene distribucije, kao i skup alata koji ćemo koristiti za prevođenje paketa.

Paketi izvornog koda se razlikuju prema tome da li su i kojoj distribuciji namenjeni. Pored osnovne verzije paketa izvornog koda, postoje i paketi prilagođeni distribucijama, čijim se prevođenjem dobijaju binarni paketi i to:

- rpm paketi - namenjeni Red Hat, Mandrake, SUSE, Fedora i
- deb paketi - namenjeni Debian i Ubuntu distribucijama.

Razlikuju se prema organizaciji datoteka i načinu prevođenja. Osnovni paket izvornog koda je u suštini kompresovani skup datoteka koji se raspakuje i zatim komandama configure, make, make install, prevodi i instalira na ciljnoj platformi. Rpm paketi pored izvornog koda poseduju između ostalih i ime_paketa.spec datoteku (.spec file) koja definiše način prevođenja. Deb paketi izvornog koda se sastoje najčešće od 3 celine, paketa izvornog koda kompresovana ime_paketa.orig datoteka, kompresovana ime_paketa.diff datoteka koja se prilikom prevođenja primenjuje, raspakuje na originalni paket i ime_paketa.dsc datoteka koja definiše kontrolne sume i zavisnosti potrebne za izgradnju paketa. Za razliku od paketa sa nastavkom ".rpm" gde se u ".spec" datoteci definiše način prevođenja, kod paketa sa nastavkom ".deb" se datoteka „rules“ najčešće nalazi u zapakovanoj datoteci sa nastavkom ".diff" (postoje i izuzeci od ovog pravila). Ove teme ćemo se ponovo dotaći kada budemo razmatrali sisteme za automatsko prevođenje paketa.

Kako bi preveli pakete za neku dotad nepodržanu arthitekturu možemo koristiti neke od sledećih alata i uputstava za izgradnju sistema:

- LFS (Linux From Scratch) - Korak po korak uputstvo za izgradnju distribucije od paketa izvornog koda.
- Yocto Project - projekat namenjen izgradnji Linuks distribucije za više ugrađenih arhitektura.
- Open Embedded - sistem za izgradnju distribucija za ugrađene arhitekture od izvornog koda.

Kako smo se susreli sa izazovom izgradnje sistema za dotad nepodržanu distribuciju, izložićemo jedno od mogućih rešenja (LFS) uz osvrt i na druge moguće načine kreiranja željene distribucije.

2.1.1 LFS - Linux From Scratch

Dobra polazna tačka je projekat LFS (Linux From Scratch). Projekat LFS[2] vodi korak po korak kroz proces izrade sopstvene distribucije od izvornog koda, daje instrukcije neophodne za dizajn i izradu sopstvenog sistema. Pored toga što pruža uputstva nudi i pomoć prilikom izrade sistema, kako za korisnike koji doslovce prate uputstvo, tako i za one koji odstupaju od uputstva i sistem grade prema sopstvenim specifikacijama.

Namena LFS je da se izgradi kompletan i upotrebljiv osnovni minimalni sistem. Ovo uključuje sve pakete potrebne da se od izvornog koda izgradi skoro minimalna baza iz koje se može izgraditi kompletni sistem u odnosu na potrebe korisnika.

LFS nije nužno najmanji mogući sistem. Uključeni paketi su bitni ali ne i esencijalni u odnosu na izgradnju sistema.

Sledeći paketi su uključeni u jegro LFS-a:

Bash, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Tar, Util-linux, Zlib.

Na osnovu grupe paketa koji se koriste u LFS-u može se uz izmene, shodno potrebama, izgraditi osnovni skup paketa kako bi se krenulo u dalje prevođenje paketa za željenu platformu. Osnovni skup paketa, korišćen u ovom radu, nakon prevođenja sadrži približno dve stotine binarnih paketa. Iz jednog paketa sa izvornim kodom dobija se jedan ili više binarnih paketa.

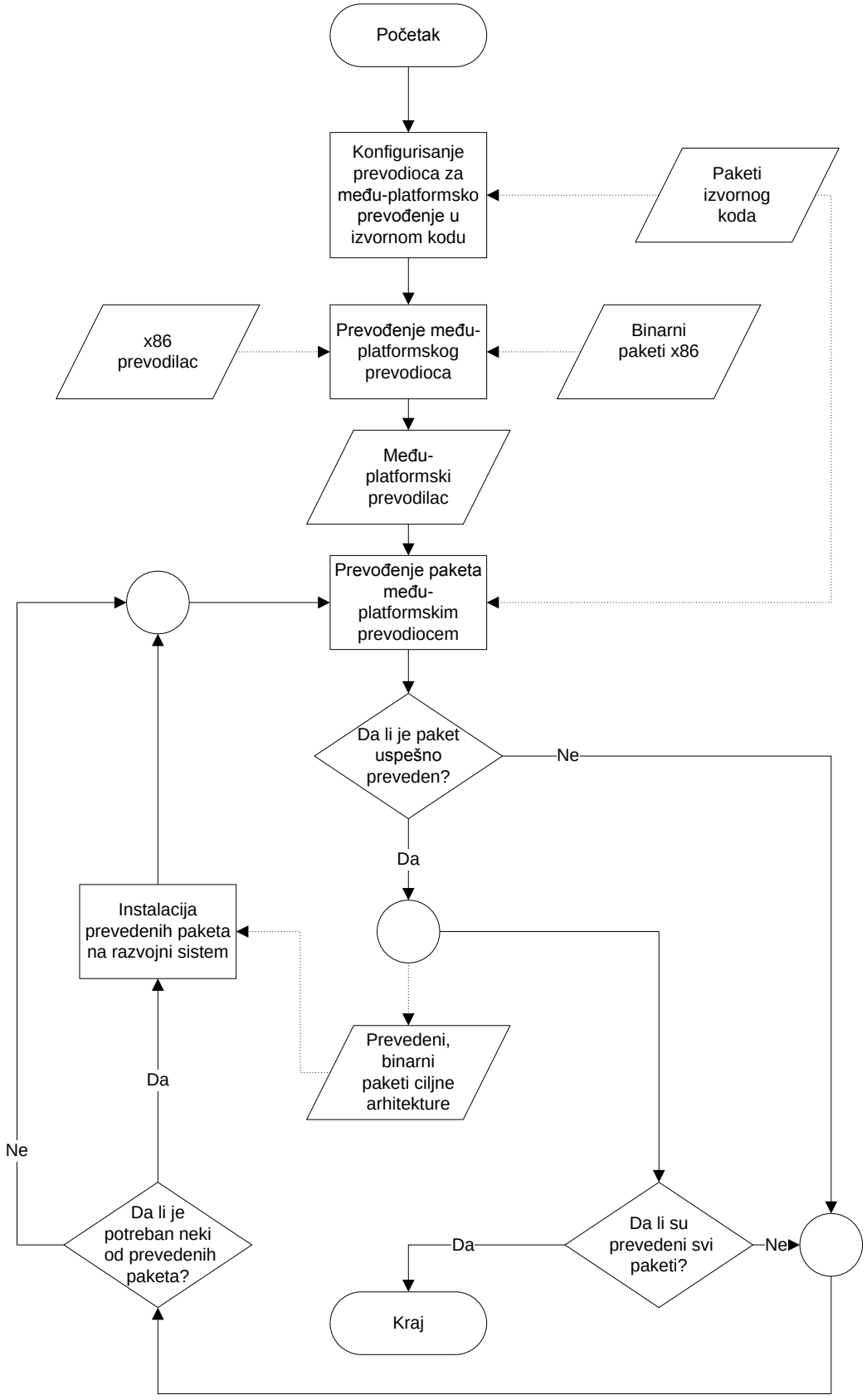
Kako bi prevođenje osnovnog skupa paketa za željenu arhitekturu bilo moguće, potreban je međuplatformski prevodilac (cross-compiler) ili skup alata (toolchain) poznat i kao međuplatformski skup alata (cross-toolchain).

Najjednostavnije rešenje je nabavka već pripremljenog prevodioca i skupa alata od proizvođača fizičke arhitekture (hardvera), platforme, za koju će se distribucija Linuksa prevoditi. A zatim isti prevodilac koristiti za prevođenje osnovnog skupa paketa, tako što se prvo prevode paketi koji su samostalni (samodovoljni), a zatim paketi koji zavise od paketa koji su već prevedeni.

S druge strane skup alata možemo izgraditi i samostalno koristeći gcc core verziju, glibc, gcc, binutils i zaglavlja jezgra operativnog sistema Linux (kernel headers). Najpre je potrebno prevesti glibc koristeći osnovnu (core) verziju gcc-a da bismo kasnije bili u mogućnosti da prevedemo gcc. Potrebno je uskladiti verzije paketa tako da međusobno budu u saglasnosti (kompatibilni), kako između sebe tako i sa arhitekturom za koju su namenjeni kako bi se uopšte preveli, bez garancija da će se i ako se prevedu biti ispravni i upotrebljivi. U radu se susreće i sa potrebom da samostalno prevedemo i konfigurišemo skup alata od izvornog koda.

Za ahritekturu x86 su nam dostupni svi potrebni binarni paketi, dok za ciljanu arhitekturu nema prevedenih binarnih paketa, uključujući tu i najosnovnije alate. Najpre se prevodiocem za Intel, na Intel računaru, od izvornog koda gcc-a, prevodi međuplatformski prevodilac konfigurisan tako da prevodi pakete na računaru baziranom na x86 (Intel) arhitekturi, za ciljanu arhitekturu. Kada se dobije međuplatformski prevodilac prelazi se na prevođenje osnovnog skupa paketa sa ciljem da binarni paketi, koji se dobiju prevođenjem, omoguće izgradnju korenskog sistema datoteka.

U procesu prevođenja, prikazanom na slici 2.1, paketi se nakon prevođenja međuplatformskim prevodiocem instaliraju na platformu na kojoj se vrši prevođenje, na određenu alternativnu putanju. Ovo se radi kako bi zavisnosti paketa koji se prevodi tj. potrebne datoteke i biblioteke za ciljani sistem bile dostupne na razvojnom sistemu i omogućile dalje prevođenje paketa za ciljnu arhitekturu. Prilikom daljeg prevođenja paketa među-platformskim prevodiocem treba u pozivnu liniju uključiti i informaciju o alternativnoj putanji na kojoj se nalaze potrebne datoteke i biblioteke za proces prevođenja, kako se ne bi koristile datoteke i biblioteke namenjene x86 arhitekturi.



2.1 - prevođenje paketa međuplatformskim prevodiocem na x86

Jednom prevedeni osnovni skup paketa može se iskoristiti za izgradnju minimalnog korenskog sistema datoteka koji se dalje može koristiti za prevođenje preostalih paketa potrebnih za izgradnju odgovarajuće distribucije.

Broj i međusobne zavisnosti paketa predstavljaju ozbiljan izazov u procesu izgradnje distribucije. Manuelno prevođenje potrebnih paketa, jedan po jedan uz vođenje računa o redosledu prevođenja paketa zahtevalo bi veliki utrošak resursa i zbog toga se pojavila potreba za alatima koji se mogu koristiti da automatizuju proces prevođenja. Takvi alati su recimo OBS (Open Build Service), DebianWannaBuild, Koji (Fedora).

2.1.2 Projekat Yocto

Projekat Yocto[3] je projekat saradnje otvorenog koda koji pruža šablone (template), alate i metode kako bi se kreirala željena proizvoljna distribucija Linuks za ugrađene sisteme, bez obzira na fizičku arhitekturu. Pokrenut je 2010. godine kao saradnja između više proizvođača fizičkih arhitektura, dobavljača operativnih sistema otvorenog koda, i kompanija koje se bave elektronikom, s namerom da se uvede red u haos koji je do tada postojao na polju razvoja distribucija Linux za ugrađene sisteme.

Kao projekat otvorenog koda, projekat Yocto, vođen je glavnim arhitektom Ričardom Pardijem (Richard Purdie) i funkcioniše po hijerarhijskom uređenju u skladu sa zaslugama u odnosu na projekat. Tehnički saradnici se biraju na osnovu kvaliteta i kvantiteta njihovog doprinosa projektu Yocto na relevantnim poljima.

Ovakav način funkcionisanja omogućava projektu da zadrži nivo slobode i samostalnosti spram svojih organizacija-članica, koje učestvuju na razne načine u cilju obezbeđivanja potrebnih resursa za sam projekat.

Projekat Yocto predstavlja kompletno okruženje za razvoj distribucija operativnog sistema Linuks za ugrađene sisteme, sa alatima, podacima o korištenim podacima (meta podaci) i dokumentacijom. Alati su besplatini za upotrebu i laki za korišćenje, između ostalih mogućnosti nude i emulatore, okruženja za ispitivanje otkaza (eng. debugers), alate za jednostavniju izradu programskih rešenja i omogućuju da projekat napreduje bez da se moraju otpisati resursi utrošeni u fazi izrade prototipa projekta. Korišćenje tehnologije otvorenog koda projekta Yocto omogućava korisnicima da se skoncentrišu na razvoj rešenja za svoje specifične potrebe, bez brige o samom okruženju.

Projekat Yocto nudi resurse i informacije pogodne i za nove i za iskusne korisnike i uključuje recepte za izgradnju osnovnih komponenti sistema pruženih projektom OpenEmbedded (OE). Recepti su osnovne komponente projekata OE i Yocto. Svaka programska komponenta dostupna u okviru projekata definisana je receptom koji je opisuje. Unutar recepta nalazi se potpuno opisan postupak prevođenja određene programske komponente. Opisani su postupak pribavljanja izvornog koda sa određene adrese, potrebne zavisnosti paketa, način raspakovanja izvornog koda, licencne informacije, podešavanje, proces prevođenja, pakovanje rešenja i ispitivanje dobijenog rešenja.

Projekat Yocto upućuje i na primere prevedenog koda koji demonstriraju njegove mogućnosti. Nudi i gotove slike sistema koje su ispitali razvojni timovi i korisnici okupljeni oko projekta Yocto. Pored korenskog sistema datoteka, slike sistema uključuju i jezgro operativnog sistema Linuks i pokrivaju nekoliko profila namenjenih prevođenju za više fizičkih arhitektura uključujući ARM, PPC, MIPS, x86 i x86-64 . Podrška za određenu platformu je u formi paketa BSP (Board Support Packages), tj paketa podrške za određenu izvedbu fizičke arhitekture, za šta je definisan i razvijen standardni format.

Kod ugrađenih sistema, paket podrške za neku od izvedbi fizičke arhitekture ili paket BSP predstavlja implementaciju specifičnog koda za izabrani uređaj koji odgovara određenom operativnom sistemu. Najčešće uključuje program „bootloader“, program je namenjen za inicijalno podizanje programskog rešenja na uređaju. Ovaj program sadrži minimalnu podršku za uređaj kako bi se na njemu startovao operativni sistem i podršku za sve ugrađene uređaje na toj verziji fizičke arhitekture.

Projekat Yocto je krovni projekat koji sadrži više različitih internih projekata:

- OpenEmbedded Core – sadrži uputstva baznog nivoa, klase i povezane datoteke koje su zajedničke između više različitih sistema izvedenih iz kostura sistema OpenEmbedded.
- Swabber – Alat zadužen da otkrije nedozvoljeno pristupanje nekom od parametara sistema, na sistemu na kome se vrši prevođenje ili izvršavanje (HOST) koda namenjenog ciljnom sistemu (TARGET), koji bi mogao da utiče na izvršavanje.
- ADT – Aplication Developmet Toolkit – razvojni alat čija je uloga da obezbedi mogućnost razvoja programskih rešenja koja mogu uz korišćenje dostupnog skupa programskih rešenja projekta da se prevode, izvršavaju, ispitiju i otklanjaju greške (debug), itd.
- AutoBuilder – projekat namenjen automatizaciji prevođenja, ispitivanja i kontrole kvaliteta.
- BitBake – zadužen za prevođenje uz pomoć recepata prateći recepte u odgovarajućem formatu u cilju izvršenja skupa zadataka
- Build Appliance – virtualna mašina koja omogućava da se prevodi i pokreće slika operativnog sistema Linux namenjena ugrađenim sistemima projekta Yocta na sistemima koji ne rade na operativnom sistemu Linuks.
- Cross-Prelink – pre-uvezivanje je proces prethodnog izračunavanja adresa učitavanja i tabela uvezivanja uz pomoć dinamičkog uvezivača, inače se ovo radi prilikom pokretanja izvršnog koda. Ovakvim radom dobija se na vremenu i performansi prilikom izvršavanja programskog rešenja – aplikacije.
- Eclipse IDE Plug-in - Integracija razvojnog alata ADT projekta Yocto i skup alata u integrisanu razvojnu platformu Eclipse (Eclipse IDE)
- EGLIBC – Izdanje GNU C biblioteke namenjeno ugrađenim sistemima
- Hob – grafička korisnička međuveza (interfejs) za BibBake
- Matchbox – osnovno okruženje otvorenog koda za X Window System (sistem prozora X) koji se izvršava na ugrađenim ne-stonim fizičkim uređajima kakvi su prenosni uređaji, „set-top box“ i kiosci, kao i svi uređaji gde su prostor na ekranu, mehanizam korisničke interakcije (input) ili resursi sistema ograničeni.
- Poky – referentni sistem projekta Yocto – kolekcija alata i meta podataka koji se koristi kao skup radnih primera. Poky se može posmatrati i kao referentna distribucija koja koristi tehnologiju projekta Yocto. Poky sadrži komponente koje zajedno oblikuju sistem za prevođenje paketa. Podržala ga je fondacija Linuks.
- Pseudo – program koji se koristi u situacijama kada su potrebna administratorska prava a korisnik ih ne poseduje, omogućava da željeni procesi uspeju kao da korisnik poseduje administratorska prava.
- Toaster – mrežno bazirana programska sprega namenjene pisanju programske podrške (mrežno baziran API) koja sakuplja i prezentuje informacije o prevođenjima vezano za određenog korisnika. Mogu se pregledati i pretraživati uz pomoć Web čitača.

2.1.2.1 Poky

U okviru projekta Yocto, Poky[4] obezbeđuje alat otvorenog koda za izgradnju pune platforme bazirane na operativnom sistemu Linuks i mobilnim tehnologijama GNOME (GNOME Mobile technologies). Poky je primarno alat za izgradnju platforme koji generiše slike korenskog sistema datoteka bazirane na programskim rešenjima otvorenog koda. Iako slike

sistema mogu biti generisane za mnoge uređaje standardni primeri su namenjeni za izvršavanje kompletnog sistema u okviru emulatora QEMU i to x86, ARM, MIPS i PowerPC i na realnom referentnom primerku fizičke arhitekture za svaku od tih arhitektura. Mogućnost sistema Poky da se pokrene u okviru emulatora QEMU, čini ovaj sistem pogodnim da se koristi kao ispitna platforma namenjena razvoju programskih rešenja namenjenih ugrađenim sistemima.

2.1.2.2 Ispitivanje kvaliteta projekta Yocto

Interesantna osobina projekta Yocto je način ispitivanja i osiguravanja kvaliteta različitih izdanja. Za proces ispitivanja i verifikacije koristi se LTP[5] (Linux Test Project) kako na sistemu koji se izvršava na emulatoru fizičke arhitekture – QEMU (npr. za arhitekturu MIPS[6]), tako i na dostupnim instancama fizičkih arhitektura, recimo beagleboard[7] za arhitekturu ARM ili routerstation[8] za arhitekturu MIPS. Ovi podaci su u velikoj meri pomogli u procesu istraživanja i identifikovanja uzroka otkaza na sistemima koje smo razvijali za fizičku arhitekturu MIPS i potom verifikovali korišćenjem dostupnih rešenja iz projekta LTP namenjenih ispitivanju. Ove teme ćemo se ponovo dotaći prilikom razmatranja verifikacije sistema.

2.1.3 OpenEmbedded

OpenEmbedded[9] je radni okvir namenjen izgradnji sistema (eng. build framework) za ugrađene urađaje koji koriste operativni sistem Linuks. OpenEmbedded nudi okruženje za međuplatformsko prevođenje, omogućava razvoj i kreiranje celokupnih distribucija operativnog sistema Linuks za ugrađene sisteme. Neke prednosti projekta OpenEmbedded su sledeće:

- Podržava različite fizičke arhitekture
- Više različitih izdanja za podržane arhitekture
- Lako podesiv
- Može se koristiti na bilo kojoj distribuciji operativnog sistema Linuks
- Međuplatformski prevodi hiljade paketa uključujući GTK+, Qt, X Windows sistem, Mono, Javu,..

2.1.3.1 OpenEmbedded-Core

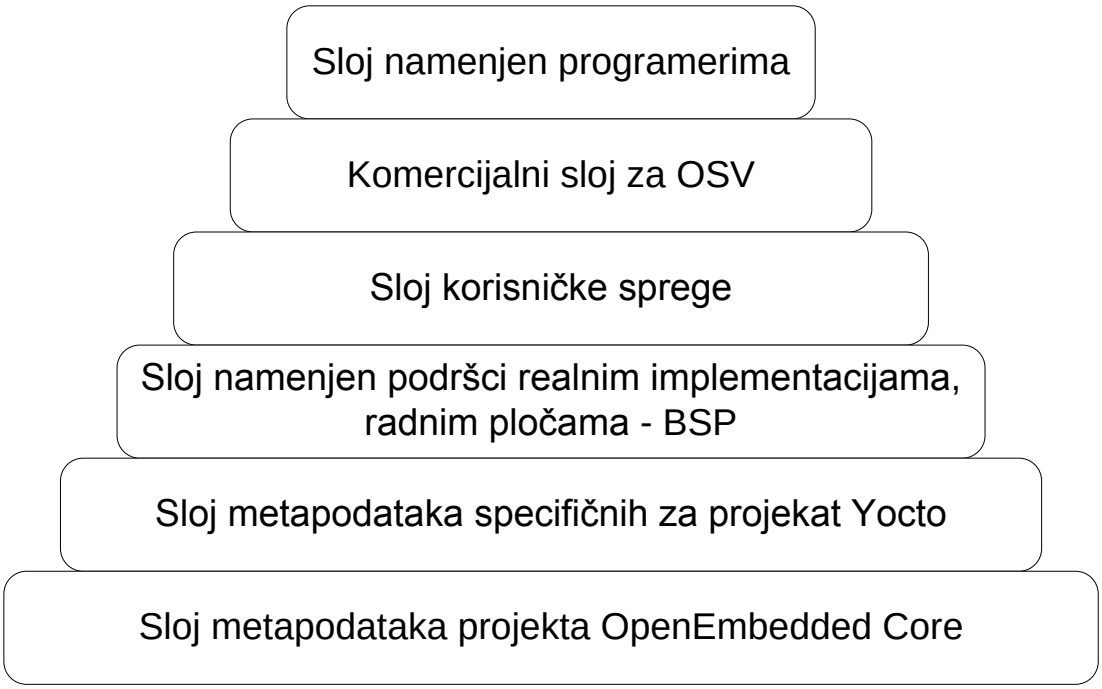
Nova izdanja sistema OpenEbedded zasnovana su na OpenEmbedded-Core[10] (OE-Core). OE-Core je evoluirao kroz proces međusobne saradnje sa projektom Yocto i uočavanjem i prihvatanjem činjenice da je prethodni model koji se koristio za OE postao neodrživ.

Originalni repozitorijum OE je porastao tokom godina na preko 7500 recepata, pokrivajući približno 300 izvedbi raznih fizičkih arhitektura i 20 distribucija. Pokušaji održavanja ovog obima i količine metapodataka (metadata) vremenom su postali komercijalno neodrživi. Poky je 2006. godine uveden kao čistije i lakše za održavanje izdanje OE-a. Poky se danas održava kao referenta distribucija u okviru projekta Yocto uz podršku fondacije Linuks. OE-Core je izdvojen iz distribucije Poky 2011. godine kako bi se obezbedila saradnja više

učesnika (kolaboracija) oko relativno malog skupa (baze), lakog za održavanje, sa isključenim realnim izvedbama fizičke arhitekture i distribucijama iz OE-Core sistema.

Od 2011. godine OE se više ne održava. Nova rešenja se grade na OE-Core osnovi. Organizovan je sada u više nivoa, za razliku od običnog OE gde je sve bilo u jednom nivou. Na osnovnom nivou se nalazi OE-Core. Zatim slede nivoi za konkrente izvedbe arhitektura (mašine), aplikacije, distribucije. Korisnici mogu da odaberu koju aplikaciju ili platformu žele da podrže u konfiguraciji. Specifična podešavanja za distribucije ili platforme nisu više raširena preko svih recepata. Podešavanja za specifične platforme se nalaze sada u sloju podrške realnim implementacija (machine layer). Ovaj nivo se često pominje kao BSP (Board Support Package) ili paket podrške konkretnoj izvedbi fizičke arhitekture.

Sledeća slika 2.2 ilustruje nov sistem zasnovan na organizaciji u više nivoa. Ovakva organizacija sistema[11] je još uvek u fazi razvoja. Većina nivoa se može dodatno konfigurisati ili kompletno izostaviti u odnosu na potrebe sistema koji se razvija.



2.2 - slojevita oraganizacija OE-Core

Organizacija u više nivoa obuhata:

- OpenEmbedded Core sloj - meta-podaci projekta OpenEmbedded Core
- Yocto sloj - metapodaci projekta Yocto
- BSP sloj - namenjen podršci za realne implementacije, radne ploče
- Sloj za korisničku spregu - meta podaci o sistemima za spregu sa korisnicima, npr. meta podaci za različite sisteme prozora itd.
- Komercijalni sloj - namenjen za smeštanje paketa, dodatnih modula i podešavanja dobavljenih od distributera programa otvorenog koda.
- Sloj namenjen programerima, korisnicima - u ovaj sloj ulaze recepti korisnika koji se po svojoj nameni i sadržaju ne mogu svrstati u neki od opštijih slojeva.

OpenEmbedded-Core sfera obuhvata:

- Podršku za 5 arhitektura: ARM, x86, x86-64, PowerPC i MIPS
- Samo uz pomoć emulatora QEMU, imitirane mašine.

- Bez opredeljanja za određenu distribuciju (DISTRO = " ")
- Samo grafičko okruženje Sato, bazirano na X sistemu prozora
- Samo recepte koji su potrebni gotovo svima ili su potrebni za ispitivanje drugih delova OE-Core.
- Pokušaj da se čuva samo jedan primerak svakog recepta sem GPLv2 / v3 izdanja.

Sve što se ne nalazi u OE-Core može se lako dodati koristeći više nivoe podrške. Ovi viši nivoi se takođe mogu koristiti kako bi se uvela starija izdanja koja su isključena iz OE-Core (uz poštovanje dogovorene strukture).

2.2 Alati za automatsko prevođenje paketa

Poznati alati za automatsko prevođenje paketa su OBS, Debian Autobuilder network i Koji. Ovde ćemo detaljnije obraditi dva koja su korišćena tokom izrade rada:

- OBS – Open(Suse) Build Service
- Debian Autobuilder network

2.2.1 OBS – Open(Suse) Build Service

Kako bi ubrzali prevođenje paketa i to uradili na jasan i ponovljiv način odlučili smo da za automatsko prevođenje paketa koristimo OBS[12].

OBS je sistem za prevođenje i distribuciju paketa izvornog koda na automatski, konzistentan i ponovljiv način. Sadrži sve potrebne alate da omogući timski rad više ljudi na projektima za koje se koristi. Nudi mogućnost finog definisanja prava pristupa, grananja koda, zahteva za spajanje više grana i pregleda dnevnika izmena.

OBS je besplatan, licenciran GNU GPL licencom. Dostupan je u obliku izvornog koda, zatim kao skup već prevedenih paketa, ali i kao programsko rešenje sa čarobnjakom za instalaciju. Slobodno se može koristiti, menjati, proširivati, popravljati i koristiti kao deo nekog drugog programskog rešenja.

OBS omogućava definisanje ulaznih repozitorijuma, koji se mogu koristiti u okviru jednog ili više projekata. OBS daje mogućnost kontrole, kako na nivou projekata, tako i na nivou paketa, pa se za svaki projekat može definisati za koje će se fizičke arhitekture izvorni kod prevoditi, ali se i na nivou projekta mogu odabrati paketi koji će biti uključeni ili isključeni iz trenutnog procesa prevođenja. Može se pratiti proces prevođenja svakog paketa pojedinačno, u toku samog prevođenja, ili pogledati kasnije izveštaj poslednje uspešnog ili neuspešnog prevođenja određenog paketa u okviru odabranog projekta za željenu arhitekturu. Kroz ulazne repozitorijume OBS-u se sem izvornog koda paketa može ponuditi i već prevedeni osnovni skup paketa za određenu arhitekturu. Koristeći ovu mogućnost OBS-u dajemo prevedeni osnovni skup paketa i time omogućavamo dalje prevođenje paketa, zavisnih od osnovnog skupa, planiranih za potrebe izrade naše distribucije Linuksa.

Za sam proces prevođenja potrebno je obezbediti namenske računare na kojima će se paketi prevoditi. Prednost OBS-a je što se za ove potrebe mogu koristiti udaljeni računari, različitih fizičkih arhitektura, pa se tako paketi uz pomoć OBS-a mogu prevoditi kako na uređajima (računarima) same fizičke arhitekture za koju se distribucija prevodi, tako i na računarima druge fizičke arhitekture. OBS svaki put kada prevodi određeni paket raspakuje sve potrebne zavisne pakete i gradi u okviru novog direktorijuma sistem dovoljan za prevođenje paketa. OBS sam razrešava među-zavisnosti paketa i redosled prevođenja usklađuje u odnosu na

zavisnost paketa od drugih, pa će paket za čije prevođenje nisu zadovoljeni svi preduslovi biti blokiran, na čekanju, sve dok se potrebni preduslovi ne zadovolje. Kada uspešno prevede neki paket OBS će sam pokrenuti prevođenje paketa koji od njega zavise a zadovoljene su im preostale zavisnosti od drugih paketa. Na žalost sistem nije savršen i mogu se pojaviti paketi sa kružnim zavisnostima, odnosno više paketa koji su međusobno zavisni, tj. blokirajući pa će u tom slučaju biti potrebno asistrirati OBS sistemu i problem razrešiti samostalno. Kada se sretnemo sa ovakvim problemom najčešće rešenje je prevod jednog od paketa iz grupe međusobno zavisnih uz isključenje prevođenja dela tog paketa koji zavisi od drugog iz grupe. Kada bismo tako preveli taj paket, preveli bismo drugi koji od njega zavisi, a zatim, nakon uspešnog prevođenja drugog paketa iz grupe ponovo preveli prvi ali potpuno uz sve zavisnosti.

Prilikom procesa prevođenja distribucija sreli smo se i sa potrebom da pakete prevodimo kako na arhitekturi za koju su i namenjeni tako i na drugim šire dostupnim arhitekturama koje su nam bile dostupne kao resurs i mogle da pruže veći broj prevedenih paketa u kraćem vremenskom intervalu. Recimo prevođenje distribucije za arhitekturu MIPS koristeći dostupne računare bazirane na arhitekturi Intel x86.

Da bismo prevodili pakete namenjene drugoj fizičkoj arhitekturi na računarima baziranim na arhitekturi Intel x86, potrebno je omogućiti da jezgro operativnog sistema Linuks, na računarima baziranim na arhitekturi x86, pozove odgovarajući emulator fizičke arhitekture kada detektuje pokušaj izvršavanja koda koji nije namenjen izvršavanju na x86 arhitekturi kako bi posredovao u procesu prevođenja (npr. paketa namenjenog arhitekturi MIPS na računaru baziranom na arhitekturi Intel x86). Ovo se postiže dodavanjem magičnog ključa za odgovarajuću arhitekturu u /proc/sys/fs/binfmt_misc/ direktorijum. Kao emulator koristili smo Qemu (Quick EMUlator). Qemu je besplatni emulator otvorenog koda koji može pokrenuti operativne sisteme (OS) i programe pisane za jednu arhitekturu, npr. MIPS, na drugoj arhitekturi, npr. x86.

Primer podešavanja magičnog ključa za MIPS:

```
echo
':mips:M::\x7fELF\x01\x02\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00
\x08:\xff\xff\xff\xff\xff\xff\xff\x00\xff\xff\xff\xff\xff\xff\xff\xff\xfe
\xff\xff:/usr/bin/qemu-mips:' > /proc/sys/fs/binfmt_misc/register
```

OBS je prvenstveno namenjen prevođenju rpm paketa. Prevođenje deb paketa je takođe moguće uz određena prilagođavanja. Izlazni repozitorijum prevedenih paketa je uređen prema strukturi karakterističnoj za sisteme koji se oslanjaju na rpm pakete, tako da se takav ne može koristiti za izgradnju korenskog sistema datoteka od deb paketa, korišćenjem standardnih alata, bez prethodnih izmena. Ovoga ćemo se značajnije dotaći kod sistema za izgradnju korenskog sistema datoteka.

Još jedna prednost u korišćenju OBS-a za prevođenje rpm paketa u odnosu na prevođenje deb paketa proističe iz same organizacije sistema pakovanja ovih paketa. Dok se kod rpm paketa pravila za prevođenje paketa nalaze u ".spec" datoteci, kod deb paketa se u ".dsc" datoteci nalaze uglavnom pobrojani potrebni paketi za prevođenje željenog paketa i opisane arhitekture za koje je paket namenjen, ali bez pravila za prevođenje koja se nalaze uglavnom u .diff.gz zapakovanoj datoteci u okviru debian direktorijuma. Zbog ovakve organizacije, jednostavnije promene nad paketom se u slučaju prevođenja rpm organizovanog paketa mogu izvršiti iz grafičkog „web“ okruženja koje pruža OBS. Međutim za bilo koju imalo složeniju intervenciju nad deb paketom, isti se mora raspakovati, zatim se unose promene, pa se paket pakuje i potpisuje i zatim ponovo podiže na ulazni repozitorijum OBS-a.

Bez obzira na ove nedostatke, sistem se pokazao kao pouzdan, jednostavan za korišćenje i vrlo upotrebljiv u procesu prevođenja paketa distribucije operativnog sistema Debian, kao i koordiniranju rada više ljudi na istom projektu, kako u procesu prevođenja paketa uz pomoć namenskih računara uz korišćenje alata QEMU tako i procesu prevođenja paketa na realnim implementacijama (radnim pločama) sa arhitekturom MIPS.

2.2.2 Debian Autobuilder network – mreža za automatsko prevođenje paketa

Nakon što se izgradi željena distribucija operativnog sistema Linuks, za arhitekturu dotad nepodržanu tom distribucijom, bilo bi poželjno tu arhitekturu nekako uključiti u skup arhitektura zvanično podržan tom distribucijom. Ovako nešto, pored zainteresovanosti ljudi okupljenih oko razvoja distribucije operativnog sistema Linuks, zahteva i primerke fizičke arhitekture koji bi se koristili za dalje održavanje i razvoj distribucije za tu fizičku arhitekturu, ali i korišćenje alata zvanično podržanih tom distribucijom (paketi distribucije Debian za ciljnu fizičku arhitekture se prevode na primercima te iste fizičke arhitekture kojoj su namenjeni, na x86 za x86, na MIPS za MIPS). Pakete dobijene međuplatformskim prevodiocem treba podići na repozitorijum distribucije Debian. Ovi paketi moraju biti prevodivi na primerku fizičke arhitetkture za koju su namenjeni.

U procesu pružanja podrške za arhitekture mips i mipsel (skup instrukcija mips2) na zvaničnom Debian repozitorijumu za izdanje „jessie“ distribucije Debian susreli smo se sa potrebom korišćenja zvanično podržanih alata projektom Debian kako bismo mogli da u procesu razrešavanja problema damo precizne i uporedive podatke u odnosu na zvanični proces prevođenja paketa za distribuciju Debian.

Debian Autobuilder network[13] ili mreža (sistem) za automatsko prevođenje paketa Debian je mreža namenjena razvoju operativnog sistema Debian. Upravlja prevođenjem paketa i to za sve arhitekture koje operativni sistem Debian trenutno podržava.

Mreža se sastoji od više mašina različitih fizičkih arhitektura koje uz pomoć posebno razvijenog programskog rešenja „buildd“ kupe pakete izvornog koda iz arhive operativnog sistema Debian i prevode ih za željenu fizičku arhitekturu.

Za prevođenje paketa namenjenih operativnom sistemu Debian ne koristi se međuplatformski prevodilac već se paketi prevode na primercima fizičke arhitekture za koje su i namenjeni, uz korišćenje programskih alata namenjenih toj arhitekturi.

Operativni sistem Debian je namenjen za više različitih fizičkih arhitektura[14].

Autori paketa najčešće prevode u binarni (izvršni) oblik samo za jednu fizičku arhitekturu koja im je dostupna, najčešće i386 ili amd64. Za ostale fizičke arhitekture koje podržava projekat Debian, paketi se prevode automatski i pritom se vodi računa da se svaki paket prevede samo jednom za svaku arhitekturu. Stanje prevođenja paketa se prati uz pomoć baze podataka sistema za automatsko prevođenje paketa.

Istorijski, prva arhitektura podžana uz Intel arhitekturu bila je m68k. Kada se krenulo sa davanjem podrške za novu arhitekturu, programeri (developeri) zaduženi za podršku operativnog sistema Debian namenjenog fizičkoj arhitekturi m68k morali su sami da prate ciklus izdavanja novih verzija paketa izvornog koda i da ih sami prevode za ciljanu arhitekturu, ako su želeli da isprate napredak operativnog sistema Debian namenjenog fizičkoj arhitekturi „Intel“. Sve ovo je rađeno manuelno, tj. programeri su pratili listu prepiske elektronske pošte na kojoj su se objavljivale informacije o dodatim paketima kako bi uzeli iste i preveli ih za arhitekturu m68k.

Da bi se izbeglo višestruko prevođenje paketa, od strane različitih ljudi, objavljivanje rada na određenom paketu vršeno je putem te iste liste na kojoj se objavljivala prepiska elektronske pošte veza za ovu temu.

Ova metodologija rada zahteva dosta vremena za praćenje rada i napretka a pored toga. podložna je i greškama. Bez obzira na nesavršenost ovakve metode, ona je dugo bila upotrebljavana za prevođenje paketa koji nisu bili namenjeni arhitekturi i386.

Mreža, sistem za automatsko prevođenje paketa distibucije Debian, automatizuje veći deo posla. Sastoji se iz niza skripti (najčešće pisanih u programskim jezicima Perl i Python) koje su kroz niz izmena napredovale pomažući u različitim zadacima. Ove skripte su vremenom prerasle

u sistem koji je u stanju da održava distribuciju operativnog sistema Debian ažurnom gotovo automatski.

Sigurnosne zakrpe se prevode na istom skupu fizičkih arhitektura kako bi se omogućila njihova pravovremena dostupnost.

Programsko rešenje buildd je u stvari ime dato grupi programskih rešenja koje koristi mreža za automatsko prevođenje podataka.

Ono se u stvari sastoji iz više različitih delova:

- wanna-build
- buildd
- sbuild

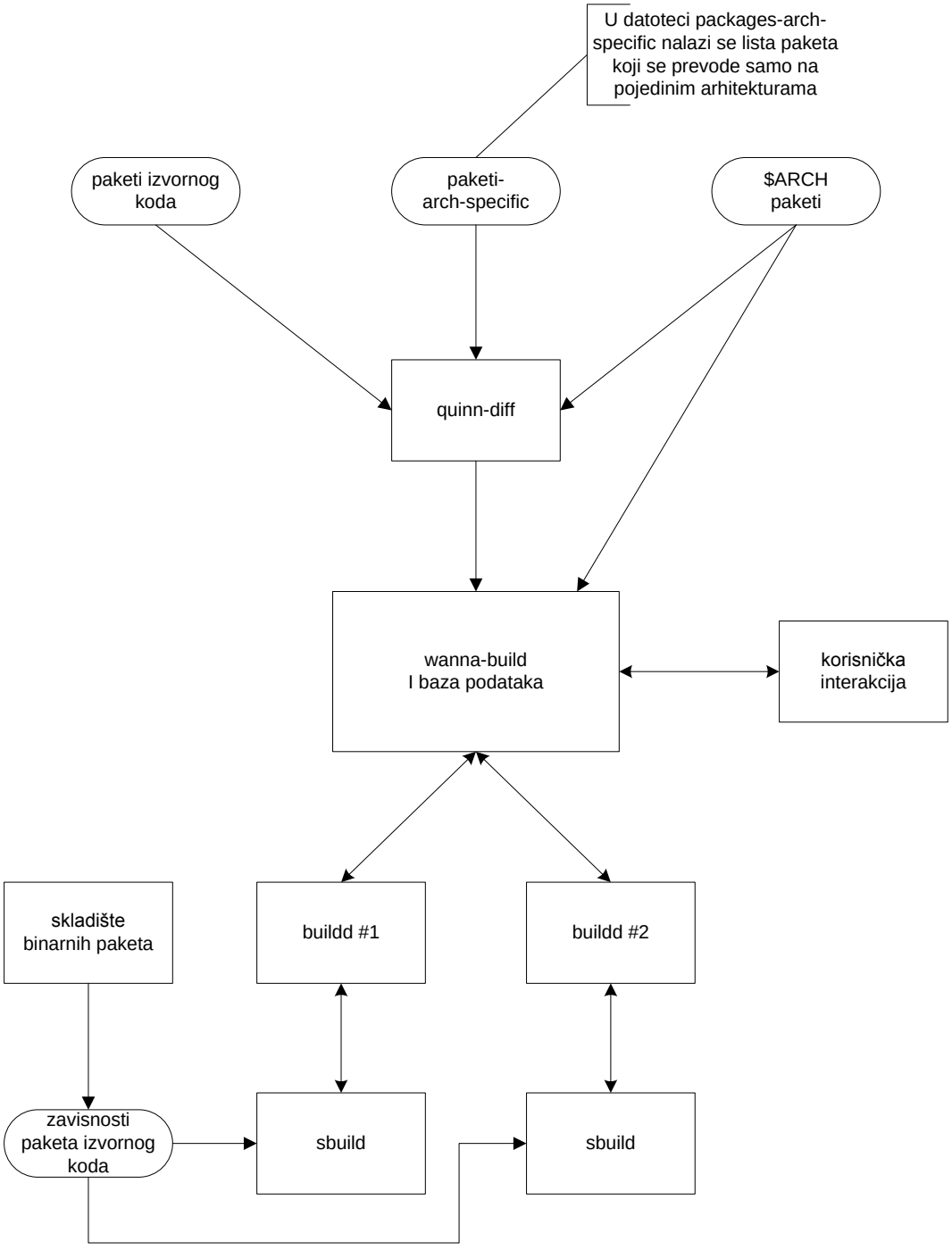
Wanna-build je alat koji pomaže koordiniramje procesa prevođenja i ponovnog prevođenja paketa uz pomoć baze podataka koja sadrži listu paketa i njihov status. Postoji jedna centralna baza podataka po podržanoj arhitekturi koja čuva podatke o statusu paketa, njihovim verzijama i pratećim informacijama. Popunjava se paketima izvornog koda i paketima iz različitih arhiva distribucije Debian (ftp-master, security-master).

Buildd je proces koji povremeno proverava bazu podataka održavanu od strane alata wanna-build i poziva alat sbuild da prevodi pakete. Nakon što administrator prihvati izveštaj prevođenja, buildd proces smešta paket u odgovarajuću arhivu.

Sbuild je alat zadužen za sam proces prevođenja paketa. Prevođenje svakog paketa vrši se u zasebnom, izolovanom korenskom sistemu datoteka (eng. chroot). Pritom alat sbuild osigurava da su sve zavisnosti potrebne za prevođenje određenog paketa od izvornog koda instalirane u korenski sistem datoteka pre samog procesa prevođenja paketa. Zatim poziva standardne alate distribucije Debian da započnu proces prevođenja. Izveštaji nastali prilikom procesa prevođenja šalju se na bazu podataka zaduženu za čuvanje istih.

Sva ova rešenja funkcionišu zajedno kako bi omogućila rad sistema za automatsko prevođenje paketa.

Uz pomoć sledećeg grafika 2.3 detaljnije ćemo objasniti način funkcionisanja sistema za automatsko prevođenje paketa.



2.3 - funkcionisanje sistema za automatsko prevođenje paketa

Paketi se, pored toga što se prevode automatski, mogu prevoditi i samostalno u slučajevima kada automatsko prevođenje iz nekog razloga nije moguće. (Ispitivanje, popravka paketa, nedostajuće zavisnosti).

Ukoliko se proces prevođenja paketa izvornog koda završi uspešno, prevedeni paket kasnije može preći u stanje „Uploaded“. Iz ovog stanja je moguće instaliranje binarnog paketa u arhivu operativnog sistema Debian te će se paket onda pojaviti na listi novih paketa za ciljanu arhitekturu. Nakon toga ta lista biće dodata u bazu podataka pa će paketi preći u stanje „Installed“, što znači da je paket uspešno instaliran u distribuciju operativnog sistema Debian. Paket ostaje u stanju instaliran sve dok se ne pojavi nova verzija izvornog koda za taj binarni paket.

Ukoliko paket u procesu prevođenja nije uspešno preveden usled nedostataka u izvornom kodu i očekuje se da će nedostaci biti otklonjeni u narednoj verziji (nakon prijavljenog problema) paket prelazi u stanje „Failed“ tj. neuspeo. Ovo stanje omogućava da, pri pojavi nove verzije izvornog koda paketa koji se nalazi u stanju neuspeo, nova verzija tog paketa bude automatski u stanju „Needs-Build“ uz napomenu da u prethodnoj verziji nešto nije bilo kako treba. Uz stanje neuspeo tj. „Failed“ čuvaju se i informacije o problemu koji je nastao prilikom prevođenja.

Stanje „Dep-Wait“ je stanje u koje paket dolazi kada su za njegovo prevođenje potrebni paketi koji još uvek nisu prevedeni. Ovo stanje čuva listu paketa i njihovih verzija koji su potrebni kako bi paket izašao iz stanja čekanja potrebnih zavisnosti (paketa koji su preduslov za njegovo prevođenje). Kada svi neophodni paketi budu prevedeni, odnosno dostupni, paket će preći u stanje „Needs-Build“ tj. potrebno prevođenje.

Proces (eng. daemon) zadužen za prevođenje paketa uzima pakete sa baze podataka i prevodi ih. Za sam proces prevođenja koristi alat sbuild i pritom se za svaki proces prevođenja, rezultujući izveštaj procesa prevođenja (eng. build log), šalje nadzorniku (eng. maintainer) procesa prevođenja. Nadzornik pregleda izveštaje i odlučuje da li da podigne paket (eng. Upload), proglasi ga neuspelim (eng. Failed), da ga stavi u stanje čekanja na potrebne pakete od kojih zavisi (eng. Dep-Wait) ili da pošalje paket na ponovno prevođenje (eng. Needs-Build). Ukoliko se dobije odobrenje nadzornika (potpisana datoteka .changes), proces zadužen za prevođenje paketa će premestiti paket u direktorijum namenjen za podizanje paketa. Odavde se svi paketi podižu uz pomoć zakazanog cron procesa koji se periodično izvršava.

Pregled izveštaja je jedini deo procesa koji zahteva aktivno učešće čoveka u celom procesu, pod uslovom da je prevođenje uspešno. U procesu prevođenja mogu se pojaviti rezultati koji su, iako je prevođenje neuspelo iz različitih razloga, nepoznatih procesu zaduženom za prevođenje, samo vratili vrednost uspešan, te je učešće čoveka neophodno. Pored toga postoji još jedan razlog da kompletan proces ne bude u potpunosti automatizovan: za podizanje (upload) paketa, potrebno je da isti bude potpisan, a automatsko potpisivanje bez lozinke bi pretstavljalo ozbiljan sigurnosti propust.

Alat sbuild, koji je u formi skripte, je zadužen za pozivanje standardnih alata distribucije Debian kako bi se preveo izvorni kod paketa. Uz to ovaj alat zadužen je i za automatsko instaliranje paketa koji su neophodni u procesu prevođenja paketa, tj. instaliranje zavisnosti potrebnih za prevođenje željenih paketa. U procesu pružanja podrške za arhitekture mips i mipsel koristili smo ovaj zvanično podržani alat za prevođenje paketa distribucije Debian u cilju identifikovanja i otklanjanja grešaka. Alternativno se može koristiti i alat pbuilder. Kasnije ćemo reći nešto više o ovim alatima i korišćenju istih.

2.2.2.1 Dodavanje paketa u distribuciju Debian i proces prevođenja

Sa strane programera koji razvija paket namenjen distribuciji Debian dovoljno je da paket izvornog koda, koji je preveden (binarni paket) za neku od arhitektura distribucije Debian, bude podignut (eng. Uploaded) odnosno dodat u arhivu. Paket će biti automatski dodat u bazu podataka za sve arhitekture u stanju Needs-Build. Kako automatski procesi za prevođenje paketa šalju upite za pakete u stanju Needs-Build, paket će biti preuzet i preći u stanje „Building“ u odnosu na trenutnu arhitekturu na kojoj se prevođenje vrši. Novi paketi se najpre dodaju u eksperimentalno izdanje distribucije Debian – experimental.

Lista paketa koje je potrebno prevesti „Needs-Build“ je uređena u skladu sa prioritetima paketa, u odnosu na prethodno stanje prevođenja: da li je paket već ranije prevođen i to prevođenje je neuspešno ili paket do sada nije bio prevođen (out-of-date, uncompiled), zatim prema zadatatom prioritetu i imenima paketa. Kako ne bi došlo do nepredvidivo dugog čekanja paketa na listi, kao posledice dodavanja novih paketa na listu (ili vraćanja već jednom prevođenih paketa u nju) na prioritet još utiče i vreme provedeno u čekanju na listi.

Ako prevođenje paketa bude uspešno na svim podržanim arhitekturama, (nadzornik neće morati da uradi ništa) prevedeni binarni paketi će automatski biti podignuti u odgovarajuće arhive. Ukoliko je prevođenje neuspešno paket će preći u neko od sledećih stanja:

- Build-Attempted ako se radi o grešci prilikom prevođenja koja još uvek nije razmatrana,
- Failed za razmatranu i prijavljenu grešku u paketu i
- Dep-Wait ako zavisi od drugih paketa koji trenutno nisu dostupni.

Sa tim da se stanje prevođenja nekog paketa može razlikovati na različitim arhitekturama.

Administratori sistema za automatsko prevođenje paketa će nakon razmatranja izveštaja prevođenja, rezultate proslediti osobi koja brine o paketu. U slučaju da se radi o izveštajima neuspelih prevođenja, rezultate će proslediti tako što će prijaviti novu grešku na sistemu za praćenje grešaka.

Ponekad je za prevođenje određenih paketa potrebno znatno više vremena na jednoj nego na drugoj arhitekturi. Ovo može zadržati paket da ne uđe na vreme u distribuciju testing (sid). Ovakvi slučajevi se mogu posebno razmatrati od strane tima zaduženog za izdanje Distribucije (Release team).

Stanje pokušaja prevođenja paketa se može ustanoviti pregledom izveštaja prevođenja. Ovi izveštaji su povezani sa pregledom paketa dodeljenih određenom pojedincu na održavanje.

2.2.2.2 Stanja paketa – Wanna build stanja

Kako bismo bolje razumeli proces prevođenja paketa daćemo pregled mogućih stanja paketa i pojasniti šta se dalje dešava kada se paket nađe u određenom stanju[15].

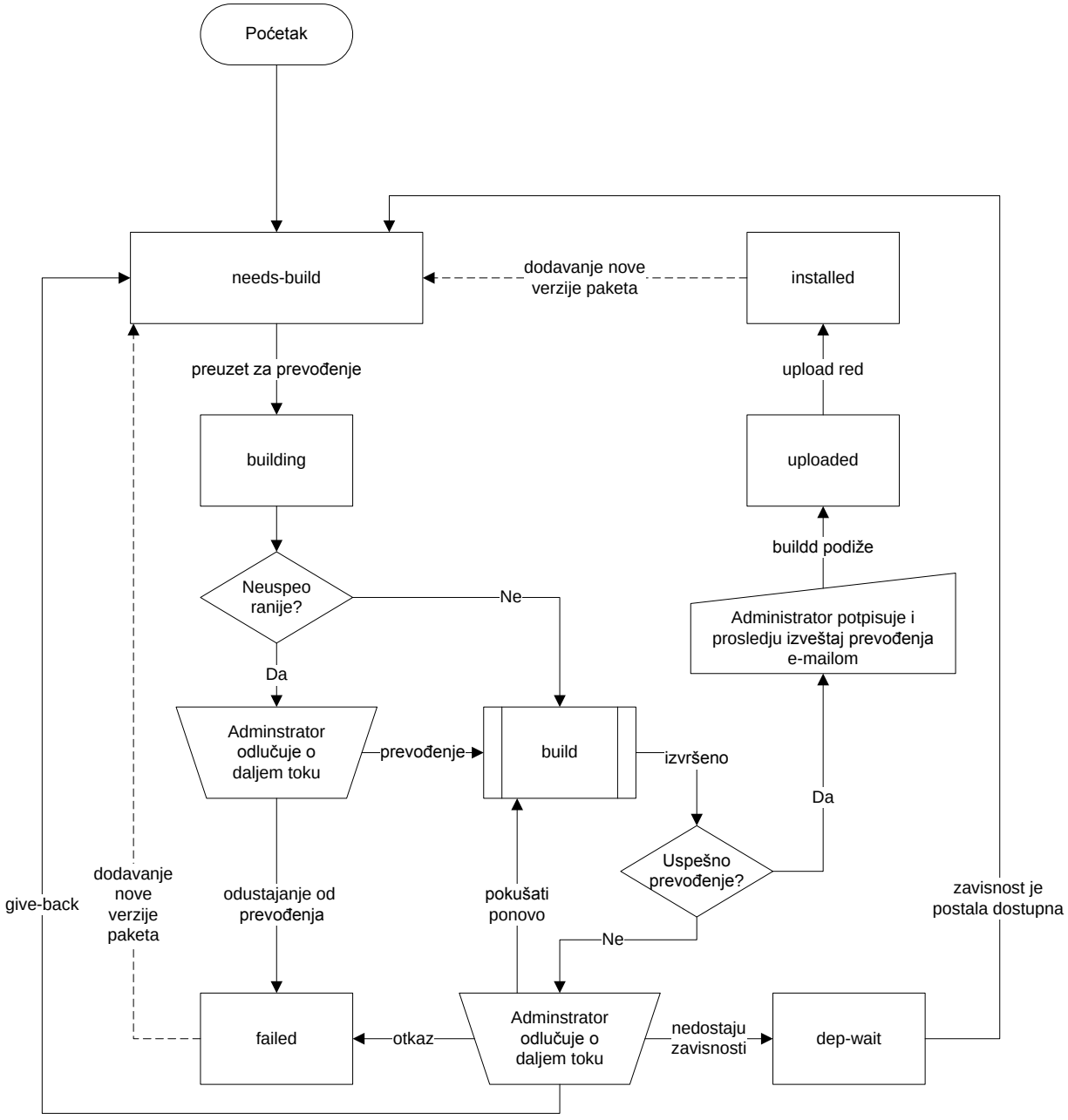
Pored toga pojasnićemo i stanje paketa u odnosu na proces i izveštaj prevođenja.

Za svaku podržanu arhitekturu distribucije Debian postoji baza podataka instalirana na buildd.debian.org sa svim paketima i trenutnim stanjima u kojima se paketi (trenutno) nalaze.

Postoji 8 stanja:

- Needs-Build
- Building
- Uploaded
- Dep-Wait
- BD-Uninstallable
- Failed
- Not-for-us
- Installed

Sledeći grafik 2.4 daje uvid u moguće akcije i prelaze u odnosu na trenutno stanje u kome se paket nalazi:



2.4 - prelazi stanja paketa

Needs-Build

Paket koji se nalazi u stanju „Needs-Build“ je podignut po prvi put ili podignut na novu verziju od strane osobe zadužene za održavanje paketa ali za neku drugu arhitekturu u odnosu na bazu podataka u kojoj se sada pojavljuje u stanju „Needs-Build“ i zbog toga je potrebno (ponovo) prevesti paket. Ako se paket nalazi u ovom stanju, sistem za automatsko prevođenje paketa će tek preuzeti ovaj paket na prevođenje. Može se prostije reći da je ovaj paket zakazan za i čeka na prevođenje. Red čekanja nije prost red čekanja gde paketi izlaze jedan za drugim u onom redosledu u kom su pristigli već se radi o redu koji se uređuje u skladu sa sledećim kriterijumima:

- Prema prethodnim stanjima u kojima su se paketi nalazili – paketi koji su već ranije prevođeni naći će se na listi ispred paketa koji do sada nisu prevođeni.
- Prioriteti – paketi sa prioritetom required (potreban) će biti prevedeni pre paketa sa prioritetom extra (dodatno).

- Prema sekciji u kojoj se paketi nalaze, odnosno prema tome koji se paketi smatraju bitnijim u odnosu na proces prevođenja – tako će se paket iz sekcije games prevoditi nakon paketa iz sekcije base, a sekcija libs će se prevoditi pre sekcije devel.
- Prema ASCII imenu paketa, nalik na abecedni redosled ali u odnosu na ASCII tabelu.

Pored svih ovih pravila, može doći do odstupanja. Tako se za prevođenje ne mora uzeti paket koji je prvi na listi ukoliko, recimo, proces buildd ne može u datom momentu da pronađe izvorni kod datog paketa. U tom slučaju pokušaće da uzme sledeći sa liste a paket koji je trenutno na vrhu liste tu će i ostati, ali sledećih nekoliko sati neće biti razmatran kao kandidat za prevođenje.

Može se desiti da za određenu arhitekturu postoji više procesa za automatsko provođenje paketa, uređenih tako da su procesi koji su „brži“, tj. rade na snažnijoj fizičkoj izvedbi određene arhitekture, budu zaduženi za zahtevnije pakete, dok će oni procesi koji imaju manje fizičkih resursa raditi na prevođenju paketa koji su manje zahtevni (ovo je najviše izraženo kod prevođenja paketa distribucije Debian namenjene ugrađenim sistemima).

Pored toga buildd, u teoriji, može zahtevati da prioritet sekcija bude drugačije organizovan ali to je jako retka pojava.

Building

Paket prelazi u stanje „Building“ (stanje prevođenja) kada ga sistem za automatsko prevođenje paketa pokupi sa reda čekanja liste wanna-build. Kako se paketi ne kupe jedan po jedan za prevođenje, paket će biti označen sa „Building“ i pre nego što započne proces samog prevođenja. Paketi u ovom stanju čekaju dalje u FIFO redu na prevođenje. Paketi ostaju u ovom stanju i nakon završenog procesa prevođenja sve dok administrator sistema za automatsko prevođenje paketa ne odreaguje na izveštaj kreiran prilikom procesa prevođenja.

Uploaded

Nakon uspešnog pokušaja prevođenja paketa izvornog koda izveštaj prevođenja se prosleđuje administratoru sistema za automatsko prevođenje paketa i na buildd.debian.org. Zatim administrator pregleda izveštaj prevođenja i ako je istim zadovoljan potpisuje .changes datoteku koja je uključena u izveštaj prevođenja (build log) i prosleđuje sistemu za automatsko prevođenje paketa. Sistem za automatsko prevođenje paketa reaguje tako što podiže (upload) prevedeni paket i prevodi ga u stanje „Uploaded“. Sistem za automatsko prevođenje paketa neće više raditi sa ovim paketom sve dok je on u stanju „Uploaded“.

Dep-Wait

U slučaju da je prevođenje paketa neuspešno usled nedostajućih zavisnosti paketa koji se prevodi, administrator sistema za automatsko prevođenje paketa proslediće poruku sistemu za automatsko prevođenje paketa kojom će izdati naredbu za uklanjane izvornog koda paketa, čije je prevođenje neuspešno, i prevođenje paketa u stanje „Dep-Wait“ u odnosu na nedostajuće pakete za njegovo prevođenje. Paket u ovom stanju će automatski, bez ikakve spoljne intervencije, preći u stanje „Needs-Build“ jednom kada sve njegove zavisnosti postanu dostupne. Istorijski, određeni paket je prvo morao da bude poslat na prevođenje pre nego bi došao u „Dep-Wait“ stanje. Izmenama koje su načinjene na kodu baze podataka wanna-build omogućeno je da paket direktno iz stanja „Installed“ pređe u stanje „Dep-Wait“ ako je to potrebno.

Postoji mogućnost da paket zauvek bude u stanju „Dep-Wait“ i to u dva slučaja:

- Kada je došlo do greške u kucanju prilikom definisanja „Dep-Wait“ liste paketa za paket koji se prevodi,

- i kada se neki od paketa,od kojih je paket koji prevodimo zavisan, nalazi u stanju „Not-For-Us“ ili se nalazi na listi paketa koji su specijalno namenjeni za neku od ostalih podržanih arhitektura (arch-specific).

Recimo da posmatramo tri paketa od kojih prvi postoji samo za i386, drugi postoji samo za MIPS i uopšteno gledano radi isto što i prvi paket ali na arhitekturi MIPS a treći paket se prevodi uz pomoć prvog ili drugog paketa. Ukoliko se desi da osoba zadužena za održavanje trećeg paketa zaboravi da doda drugi paket na (Build-Depends) listu paketa potrebnih za prevođenje trećeg paketa, kada i kasnije bude primećeno da se treći paket nalazi u stanju „Dep-Wait“ u odnosu na nepostojeći prvi paket za arhitekturu mips i bude mu ispravljena lista paketa od kojih zavisi, paket neće moći da iz „Dep-Wait“ stanja izađe automatski već će ovo stanje napustiti tek kada osobe zadužene za rad na izdanju distribucije Debian za MIPS intervenišu kako bi promenili stanje u kom se nalazi.

BD-Uninstallable

Tokom konferencije debconf9, Joachim Breitner je izneo ideju o mogućnosti korišćenja alata edos-debcheck za proveru mogućnosti instaliranja paketa potrebnih za prevođenje paketa koji bi dotadašnjim kriterijumima prešao u stanje „Needs-Build“. U tom momentu wanna-build je posedovao mogućnost da proveri trenutnu dostupnost paketa potrebnih za prevođenje, ali nije mogao da utvrdi da li paket koji je preduslov za prevođenje željenog paketa može biti instaliran ukoliko on zavisi od drugog paketa koji zavisi od trećeg paketa, a treći paket je dostupan ali u verziji, recimo, 4.6.3 dok je za drugi paket potrebna verzija >=4.7.2. Pa bi proces prevođenja vrlo brzo došao u stanje otkaza usled nemogućnosti da instalira potrebne pakete. Proces rešavanja ovakvih problema je zahtevao uključivanje administratora kako bi problem istražio, pa je ceo proces trajao dugo.

Pojavom izmene (zakrpe) za „BD-Uninstallable“, stanje se dosta poboljšalo. Sada kada je paket u stanju „BD-Uninstallable“ to znači da se neki od njemu potrebnih paketa ne može instalirati ili zbog toga što je nedostupan ili zato što je neki paket koji se nalazi na listi njemu potrebnih paketa nedostupan, tj. nedostaje neki paket iz stabla potrebnih paketa za prevođenje željenog paketa.

Na žalost ovo proširenje ne obezbeđuje i informacije o tome koji su paketi tačno nedostupni, pa je potrebno koristiti alat edos-debcheck da bi se ti paketi izdvojili. Ipak paket će onog momenta, kada njemu potrebni paketi iz stabla paketa od kojih zavisi postanu dostupni, automatski preći u stanje „Needs-Build“.

Failed

Ako se pokušaj prevođenja završio neuspešno, a administrator instance sistema za automatsko prevođenje paketa (eng. autobuilder maintainer) utvrdi da se zaista radi o otkazu i da ne treba ponovo pokušati prevesti paket, paket prelazi u stanje „Failed“. Paket neće napustiti ovo stanje sve dok osoba koja se bavi prevođem za željenu arhitekturu ne odluči kako će rešavati nastali problem ili dok se ne pojavi novija verzija paketa. Ukoliko se pojavi nova verzije paketa koji je u stanju „Failed“ neće doći do automatskog prevođenja paketa u drugo stanje, već će sistem za automatsko prevođenje paketa proslediti zahtev administratoru da odluči treba li pokušati prevođenje nove verzije paketa. Ova provera je potrebna kako bi se izbeglo da paketi ,koji će neminovno „pasti“ u toku procesa prevođenja, bespotrebno okupiraju resurse potrebne za prevođenje paketa. Iako proglašavanje paketa za neuspešan i prevođenje u stanje „Failed“ bez prethodnog pokušaja prevođenja ne predstavlja dobru praksu, ova mogućnost stoji na raspolaganju administratoru sistema za automatsko prevođenje paketa.

Bitno je primetiti da paket ni u kom slučaju neće biti proglašen za „Failed“ od strane sistema za automatsko prevođenje paketa, već će to uvek biti urađeno kada tako odluči čovek, tj. administrator zadužen za to.

Not-for-us

Pojedini paketi su namenjeni samo nekim od podržanih arhitektura, recimo paket lilo (boot loader) koji je namenjen za i386, ali ne bi trebalo da se prevodi recimo na s390. Kako wanna-build ne pregleda control datoteku paketa izvornog koda kada kreira bazu podataka nego samo ime paketa, sekciju (grupu) kojoj paket pripada, prethodni rezultat tj. stanje prevođenja i prioritet paketa, novododati paket koji je specifičan za određenu arhitekturu bio bi upućen na prevođenje za sve podržane arhitekture operativnog sistema Debian, ali bi se prevođenje vrlo brzo zaustavilo još pre pribavljanja ili instalacije paketa potrebnih za prevođenje željenog paketa.

Kako sistemi za automatsko prevođenje paketa ne bi gubili vreme na prevođenje paketa koji nisu namenjeni arhitekturi za koju i na kojoj prevode pakete, bilo je potrebno pronaći rešenje koje bi rešilo ovaj problem.

Prvobitno rešenje je bilo uvođenje stanja „Not-for-us“. Paket bi bio proglašen, stavljen, u stanje „Not-for-us“ za određenu arhitekturu pa sistem za automatsko prevođenje paketa ne bi uzimao ovakve pakete u obzir prilikom procesa prevođenja paketa. Ovakav način rešavanja problema se pokazao kao komplikovan za održavanje, tako da se ovaj sistem danas smatra zastarelim.

Umesto „Not-for-us“ stanja, administratori sistema za automatsko prevođenje paketa, trebali bi da koriste liste na kojima se nalaze paketi specijalno namenjeni za jednu ili više arhitektura (ali ne za sve).

Paket koji se nalazi u stanju „Not-for-us“ ili je specijalno namenjen za određenu arhitekturu, neće automatski napustiti stanje u kom se nalazi. Ukoliko je za neki paket, koji je za neku od arhitektura prethodno isključen u datoteci control tog paketa, u međuvremenu dodata podrška za neku od ranije nepodržanih arhitektura, u njegovu datoteku control, paket neće biti automatski prevođen za tu arhitekturu, već će biti potrebna intervencija kako bi se paket onda prevodio za novopodržanu arhitekturu, tj. paket se mora ručno dodati na listu paketa koji čekaju na prevođenje.

Ukoliko je ovako nešto potrebno zahtevati, treba poslati zahtev relevantnim osobama zaduženim za održavanje sistema za automatsko prevođenje paketa.

Installed

Paket koji se nalazi u stanju „Installed“ u odnosu na određenu arhitekturu i bazu podataka wanna-build za tu arhitekturu, je preveden za tu arhitekturu.

Do pojave distribucije operativnog sistema Debian pod kodnim imenom „woody“, stanje paketa se menjalo iz „Uploaded“ u „Installed“ jednom dnevno (kada bi radio proces katie). Sada paket prelazi iz stanja „Uploaded“ u stanje „Installed“ kada je prihvaćen u arhivu. Najčešće je potrebno svega 15 minuta da paket pređe u stanje „Installed“.

Removed

Pored pobrojanih stanja, postoje i dva stanja sa nastavkom „-removed“. To su „Dep-Wait-removed“ i „Failed-removed“.

Kada se paketi koji su u stanju „Failed“ ili „Dep-Wait“ ne pojave na novoj listi paketa u datoteci koja se prosleđuje bazi podataka wanna-build, dotadašnji podaci o tim paketima se ne odbacuju, je se može desiti da je nepojavljivanje određenih paketa u datoteci samo trenutni propust koji može biti kasnije otklonjen ili je paket samo privremeno uklonjen iz nekog razloga a pojaviće se ponovo nakon određenog vremenskog intervala. U takvim slučajevima paketi prelaze u jedno od „-removed“ stanja, kako bi se, u slučaju da postoji potreba, informacija o tome zašto je paket otkazao, ili zbog čega se paket nalazi u stanju čekanja, mogla pružiti na uvid. Ukoliko se paket označen kao „Failed-removed“ ili „Dep-Wait-removed“ ponovo pojavi na listi paketa koja se prosleđuje bazi podataka wanna-build, paket će pre bilo kakve dalje obrade biti vraćen u prethodno stanje u kome se nalazio pre uklanjanja sa liste paketa, tj. iz stanja „Failed-removed“ preći će u stanje „Failed“, ili će iz stanja „Dep-Wait-removed“ preći u stanje „Dep-Wait“.

Samoj bazi podataka wanna-build nije moguće pristupiti direktno. Baza podataka wanna-build se nalazi na ftp-master.debian.org koji je sistem sa ograničenim pravima pristupa. Samo sistemi zaduženi za automatsko prevođenje paketa imaju SSH ključ uz pomoć koga pristupaju bazi podataka wanna-build za arhitekturu kojoj pripadaju i prevode pakete namenjene toj arhitekturi. Sistem namenjen automatskom prevođenju paketa za arhitekturu MIPS ne može da pristupi bazi podataka wanna-build namenjenoj arhitekturi i386.

Trenutno stanje u kojem se određeni paket nalazi može se proveriti na adresi buildd.debian.org.

2.2.2.3 Zabeleške rezultata prevođenja paketa (build log results)

Prilikom prevođenja paketa korišćenjem alata sbuild (komponente buildd-a koja je zadužena za sam proces prevođenja), po završenom pokušaju prevođenja biće poslat izveštaj sa rezultatima prevođenja putem elektronske pošte administratoru sistema za automatsko prevođenje paketa i na adresu logs@buildd.debian.org (kako bi rezultati bili dostupni na http://buildd.debian.org). Izveštaj sadrži informaciju o procesu prevođenja i može biti označen na jedan od sledećih načina:

- successful,
- attempted (ranije poznat kao failed),
- given-back,
- skipped.

Na stranici koja nudi pregled izveštaja prevođenja na ove oznake se dodaje reč „maybe-“ (možda) te se njome sugeriše da izveštaj koji je označen kao neuspeo ili uspeo može biti tu iako je došlo do neuspelog prevođenja paketa usled stvari koje nisu uslovljene nekorektnošću samog paketa koji je predmet prevođenja ili je sa druge strane paket navodno uspešno preveden a ustvari paket nije korektno preveden i potrebno ga je ponovo prevesti.

Značenje oznaka koje se koriste za označavanje izveštaja prevođenja je sledeće:

Successful

Proces prevođenja je uspešno izvršen. Kada osoba zadužena za održavanje sistema za automatsko prevođenje paketa dobije ovaj izveštaj (zabelešku – eng. log) (ukoliko se pregledom ustanovi da log nije samo prividno uspešan) izvući će uključenu .changes datoteku, potpisati je i poslati je nazad sistemu za automatsko prevođenje paketa, što će za rezultat imati da paket bude podignut („uploaded“).

Attempted (ranije označavan failed)

Proces prevođenja se završio rezultatom koji je različit od nule, čime se ukazuje da se najverovatnije završio neuspehom. Kako neupeh prilikom prevođenja može da se javi usled različitih faktora, pokušaj da se za svaki mogući razlog otkaza dodeli odgovarajući broj kako bi se klasifikovali svi mogući otkazi bio bi vrlo zahtevan, tako da se nije ni pokušavalo klasifikovanje mogućih razloga otkaza. Ako je paket označen kao (maybe-)failed, biće potrebno pogledati izveštaj prevođenja i proveriti trenutno (wanna-build) stanje.

Given-back

Prevođenje je neuspešno usled pojave privremenog problema sa sistemom za automatsko prevođenje paketa, npr. problem sa mrežom, trenutna nedostupnost izvornog koda paketa u odnosu na trenutnu listu (sources.list), nedostatak slobodnog prostora na fizičkom disku za čuvanje podataka (hdd) i drugo.

Paket koji je dat, vraćen nazad na prevođenje (eng. given-back) stavlja se ponovo u stanje „Needs-Build“ kako bi ponovo bio automatski pokupljen od strane nekog drugog **(od)** sistema za automatsko prevođenje paketa kada je neki od njih slobodan.

Skipped

Paket će biti ovako označen ako se za vreme koje je paket čekao u stanju Building, nakon što je pokupljen od strane sistema za automatsko prevođenje paketa do momenta kada bi prevođenje paketa zaista i bilo pokušano (eng. build attempt), pojavila nova verzija istog paketa, ili je osoba zadužena za prevođenje paketa za trenutnu arhitekturu izmenila (wanna-build) stanje paketa. Kada se ovo dogodi, prosleđuje se poruka sistemu za automatsko prevođenje paketa, koji će označiti paket kao neželjen za prevođenje. Sbuild reaguje na ovo i preskače prevođenje, ali se ipak šalje izveštaj prevođenja u kome se opisuje da se dogodilo odustajanje, preskakanje prevođenja.

Razumevanje i korišćenje zvanično podržanih alata je preporučljivo jer daje uporedive rezultate sa zvaničnim repozitorijumom distribucije Debian, te daje dobru podlogu za unošenje promena i podrške za željene pakete u okviru distribucije Debian. Sa stanovištva kasnijeg održavanja daleko je praktičnije ući na listu zvanično podržanih paketa nego svaki put, kada se pojavi novo izdanje paketa, prilagođavati izmene u odnosu na novo izdanje. Promene koje jednom uđu u zvanični repozitorijum biće testirane i u odnosu na najnoviju verziju paketa. Takođe su dostupne daleko većoj ciljnoj grupi, te je broj osoba potencijalno zainteresovanih za održavanje tih promena daleko veći.

2.2.2.4 Korišteni alati u procesu pružanju podrške distribucije Debian sid

Prilikom pružanja podrške za distribuciju Debian sid (treba da bude izdata pod oznakom jessie) korišteni su sledeći alati:

- sbuild
- pbuilder

Oba alata se mogu koristiti za prevođenje paketa od izvornog koda namenjenog distribuciji Debian i predstavljaju alternativu jedan drugom.

Zvanično korišten alat u procesu prevođenja je alat sbuild.

Oba alata se odlikuju osobinom da prevođenje vrše u zasebnom korenskom sistemu datoteka u koji instaliraju sve potrebne zavisnosti. Na ovaj način se prevođenje paketa svaki put vrši u „čistom“ korenskom sistemu datoteka bez spoljnih uticaja, a ovim se i čuva sistem na kojem se obavlja prevođenje jer se na njega ne instaliraju direktno, potencijalno nestabilni paketi.

2.2.2.4.1 Sbuild

Alat sbuild[16] se koristi na zvaničnoj mreži za prevođenje paketa distribucije Debian buildd za sve podžane arhitekture kako bi se dobili prevedeni - binarni paketi. Alternativno se mogu koristiti alati cowbuilder i pbuilder. Alat sbuild se može koristiti i od strane pojedinca za proces pripreme paketa za prevođenje na zvaničnom repozitorijumu ili za samostalno prevođenje paketa.

Kako bi se alat koristio potrebno ga je dobaviti i podesiti a zatim se može krenuti u prevođenje željenog paketa:

```
1. sudo apt-get install sbuild
```

```
2. sudo sbuild-update –keygen
3. sudo sbuild-adduser $IMEKORISNIKA
4. isključiti se i zatim opet uključuti
5. sudo sbuild-createchroot –make-sbuild-tarball=/var/lib/sbuild/sid-
   mips.tar.gz sid ‘mktemp –d’ http://ftp.debian.org/debian
```

Prva linija dobavlja i instalira paket sbuild. Druga linija generiše ključ apt koji se koristi od strane alata sbuild. Treća linija će dodati korisničko ime, tj. omogućiti tom korisniku da koristi komandu sbuild. Četvrta linija, se izvšava kako bi se grupa korisnika koji mogu koristiti sbuild osvežila. Peta linija kreira korenski sistem datoteka koji će biti korišćen od strane alata sbuild za prevođenje paketa izdanja sid distribucije Debian. Udaljeni repozitorijum apt koji će se koristiti je takođe naveden u okviru pete linije i nalazi se na adresi http://ftp.debian.org/debian. U okviru ove linije može se proslediti i parameter –arch kako bi se kreirao korenski sistem datoteka za drugu arhitekturu. Ovo je korisno kada, recimo, prevodimo pakete za i386 na amd64 pa ćemo proslediti parameter –arch=i386. Takođe ako prevodimo pakete namenjene arhitekturi mips ili mipsel na arhitekturi mips64 odnosno mips64ei, tada će u prvom slučaju biti –arch=mips, a u drugom –arch=mipsel. Ukoliko se ovaj parametar ne navede biće podešen da odgovara arhitekturi na kojoj se i vrši prevođenje.

Prevođenje se poziva naredbom:

- `sbuild ime_paketa.dsc` - ako se paket prevodi u lokalu uz pomoć njegove dsc datoteke – u ovom slučaju potrebno je samostalno dobaviti paket za prevođenje.
- `sbuild –d sid ime paketa verzija paketa` - ako se paket prevodi sa repozitorijuma apt. U ovom slučaju dobavlja se odgovarajuća verzija paketa sa repozitorijuma. Paket izvornog koda distribucije Debian će pritom ostati u korenskom sistemu datoteka u kome je vršeno njegovo prevođenje.

Izdanje distribucije za koju se prevode paketi je prema osnovnim podešavanjima izdanje sid. Ako se želi promeniti izdanje za koje se paketi prevode neophodno je ovo specificirati putem parametra „–d izdanje“, naredbe komandne linije. Takođe će se prema osnovnim podešavanjima prevoditi samo paketi zavisni od arhitekture tj. namenjeni određenoj arhitekturi. Paketi koji su nezavisni od arhitekture tj. koji su namenjeni svim arhitekturama se neće prevoditi. Ako želimo da i njih prevodimo, možemo dodati u komandnu liniju za prevođenje parametar –A. Osnovna podešavanja su uslovljena korišćenjem ovog alata u okviru sistema za automatsko prevođenje paketa, pa pakete koji su nezavisni u odnosu na arhitekturu ne bi imalo smisla prevoditi na svim dostupnim arhitekturama. Ovo se odnosi kako na kompletne pakete izvornog koda tako i na delove paketa izvornog koda.

Alat sbuild će se pritom postarati da sve potrebne zavisnosti budu dobavljene i instalirane u okviru korenskog sistema datoteka u kome će se vršiti prevođenje kako bi se krenulo u sam postupak prevođenja paketa.

Nakon svakog pokušaja prevođenja paketa u direktorijumu odakle je prevođenje paketa i pokrenuto ostaće trag, izveštaj procesa prevođenja. Takođe se korenski sistem datoteka u kome je vršen proces prevođenja može po želji sačuvati ili ukloniti, u odnosu na parametre komandne linije. Ukoliko nam je korenski sistem datoteka neophodan za dalje istraživanje problema sačuvaćemo ga kako bismo mogli da krenemo u rešavanje problema.

Ukoliko je paket uspešno preveden, binarni paketi će se naći u direktorijumu u kome se nalazi i izveštaj procesa prevođenja.

2.2.2.4.2 Pbuilder

Kako smo prilikom procesa verifikacije promena na paketima za koje smo pružali podršku dobijali, od drugih zainteresovanih učesnika za proces pružanja podrške, povratne

informacije u obliku izveštaja prevođenja paketa dobijenih uz pomoć alata pbuilder, ovaj alat se nametnuo kao neophodan za korišćenje. Alat smo koristili za prevođenje paketa kako bi dobijeni izveštaji bili potpuno uporedivi sa rezultatima koje smo dobijali od ostalih učesnika u procesu pružanja podrške za distribuciju Debian sid namenjenu arhitekturama mips i mipsel.

Alat pbuilder[17][18] kao i alat sbuild omogućava da se vrši prevođenje paketa u zasebnom „čistom“ korenskom sistemu datoteka. Ovaj alat kao i prethodno razmatrani sbuild sam dobavlja i instalira potrebne zavisnosti u čist korenski sistem datoteka (na osnovu .dsc datoteke) pre nego započne prevođenje željenog paketa.

Kako bi se alat koristio potrebno ga je instalirati i kreirati korenski sistem datoteka u kome će se vršiti prevođenje paketa:

- `sudo apt-get install pbuilder` – dobavlja i instalira paket pbuilder.
- `sudo pbuilder create` – naredba kreira osnovni korenski sistem datoteka koji će se korisititi u procesu prevođenja paketa.

Ukoliko želimo da prevodimo recimo i386 pakete na amd64, potrebno je da prilikom kreiranja korenskog sistema datoteka koristimo parametar arch, u ovom slučaju –arch=i386.

Nakon toga potrebno je pribaviti izvorni kod paketa distribucije Debian koji želimo da prevodimo uz pomoć naredbe „apt-get source ime_paketa“.

Zatim se komandom „sudo pbuilder build ime_paketa.dsc“ pokreće proces prevođenja paketa. Ukoliko je paket uspešno preveden, binarni paketi će po završetku prevođenja biti smešteni na lokaciji /var/cache/pbuilder/results.

Ukoliko želimo da dobijemo izveštaj prevođenja paketa u obliku datoteke, koju možemo po potrebi pregledati, potrebno je da komandom kojom se poziva prevođenje paketa prosledimo parametar –logfile putanja_do/ime_željenog_izveštaja u sledećem redosledu:

```
$ sudo pbuilder build –logfile putanja_do/ime_željenog_izveštaja
ime_paketa.dsc
```

Jedna od razlika alata sbuild i pbuilder je ta da alati, ako im drugačije nije rečeno kroz parametar pozivne linije, prevode paket korišćnjem sledećih naredbi:

- `build`, u slučaju alata pbuilder,
- `build-arch`, u slučaju alata sbuild.

Tako će alat pbuilder prilikom prevođenja paketa dati sve binarne pakete na osnovu ulaznog izvornog koda koji se mogu izgraditi, dok će alat sbuild prevođenjem dati samo one pakete koji su zavisni od arhitekture a binarne pakete, koji su isti na svim arhitekturama (arch all, recimo dokumentacija), neće prevoditi od dostupnog izvornog koda. Ovo je tako zbog specifične namene koju alat sbuild ima u grupi alata sistema za automatsko prevođenje paketa distribucije Debian. Ovu osobinu alata treba imati u vidu u slučaju da se porede rezultati prevođenja paketa dobijeni uz pomoć jednog i drugog alata.

2.2.3 Koji

Koji je sitem za prevođenje RPM paketa. Projekat Fedora koristi Koji[19] kao sopstveni sistem za prevođenje i izgradnju paketa. Namena sistema Koji je da obezbedi fleksibilan, siguran i ponovljiv način za igradnju programskih komponenti. Prilikom procesa prevođenja Koji koristi alat Mock da izgradi korenski sistem datoteka potreban za prevođenje paketa. Alat je zadužen da na pouzdan način izgradi korenski sistem datoteka i pokuša prevođenje paketa u okviru istog.

Koji obezbeđuje novi korenski sistem datoteka za svako prevođenje paketa, koristi XML-RPC programsko sučelje za lakšu integraciju sa ostalim alatima. Poseduje Web spregu (interfejs) sa protokolom SSL i protokolom Kerberos za potvrdu identiteta. Poseduje mali, prenosivi klijent program za rad iz komandne linije tj. školjke. Korisnici mogu da kreiraju lokalne korenske

sisteme datoteka (buildroot) za prevođenje paketa. Podaci o korenskim sistemima datoteka korišćenim za prevođenje paketa se čuvaju u bazi podataka.

Koji se sastoji od nekoliko ključnih komponenti:

- Koji-Hub - poslužilac
- Kojid - servis sistema Koji namenjen prevođenju.
- Koji-Web - web klijent za Koji
- Koji-client - klijent za Koji
- Kojira – proces (eng. daemon) koji je zadužen da održava korenski sitem datoteka namenjen prevođenju (eng. build root) ažurnim.

Koji-Hub je centar svih koji operaticija. To je poslužilac prima pozive (XML-RPC) i oslanja se na procese zadužene za prevođenje i ostale komponente da započnu komunikaciju. Sam je pasivan i ne započinje proces komunikacije. Jedina je komponenta koja ima pristup bazi podataka i jedna od dve komponente koja imaju mogućnost pisanja na nivou sistema datoteka.

Kojid je proces koji vrši prevođenje koje se izvršava na svakoj od mašina namenjenih prevođenju. Odgovoran je primarno za organizovanje resursa i obrađivanje dolazećih zahteva za prevođenje paketa. U suštini Kojid pita Koji-Hub da li postoje zadaci koje bi Kojid trebalo da obradi. Koji podržava i drugačije zadatke od prevođenja paketa, npr. kreiranje i instalaciju slika sistema, te zadatke takođe obavlja Kojid. Za potrebe prevođenja Kojid koristi alat Mock. Takođe kreira nov korenski sistem datoteka namenjen prevođenju paketa prilikom svakog prevođenja. Napisan je u programskom jeziku Python. Komunikaciju sa Koji-Hub poslužiocem obavlja koristeći XML-RPC.

Koji-Web je skup skripti koje obezbeđuju mrežnu spregu za Koji. Ponaša se kao klijent prema Koji-Hub komponenti i obezbeđuje vizuelnu spregu za izvršavanje određene grupe akcija. Koji-Web obezbeđuje pregled velike količine informacija kao i određeni broj akcija i operacija kao npr. otkazivanje procesa prevođenja.

Koji-client je klijent napisan korišćenjem programskog jezika Python. Nudi mogućnost povezivanja i dobavljanje informacija korisniku kroz upite ali i obavljanje aktivnosti kao što su iniciranje zahteva za prevođenje ili dodavanje novih korisnika sistema.

Kojira je proces koji obezbeđuje ažurnost podataka u korenskim sistemima datoteka namenjenim prevođenju paketa. Zadužen je i za uklanjanje redundantnih korenskih sistema datoteka u kojima se prevode paketi kao i za čišćenje, uklanjanje, korenskog sistema datoteka nakon završenog procesa prevođenja paketa.

2.3 Alati za pakovanje distribucija

Za pakovanje distribucije iz prevedenih paketa koriste se, u zavisnosti od ciljne distribucije, različiti alati, recimo: MIC2 Mic-Image-Creator za distribuciju MeeGo, debootstrap za distribuciju Debian, febootstrap i Mock za distribuciju Fedora itd. U procesu pakovanja korišten je i alat reprepro čiji je zadatak izgradnja skladišta apt od binarnih paketa.

Kako su paketi, prevedeni uz pomoć programskog rešenja OBS, organizovani prema uređenju namenjenom za .rpm (OpenSuse, Fedora) pakete, dobijeni izlazni repozitorijum OBS-a je u slučaju distribucije MeeGoo (distribucija koristi .rpm pakete) bio pogodan za izgradnju korenskog sitema datoteka. Dok je u slučaju automatskog prevođenja paketa distribucije Debian uz pomoć OBS-a bilo potrebno prevedene pakete nekako organizovati u skladu sa zahtevima izgleda repozitorijuma koji sadrži .deb pakete i pogodan je za izgradnju korenskog sistema datoteka. Podsetićemo da je i OBS prvenstveno namenjen prevođenju .rpm paketa. Ali je

korišten kao sistem koji nam je već bio dostupan i poznat i za prevođenje .deb paketa što nije njegova inicijalna namena.

Kao rešenje za problem neadekvatno uređenog izlaznog skladišta OBS-a u odnosu na potrebe rada sa deb paketima, pojavio se alat „reprepro“ koji je korišten za izgradnju odgovarajućeg skladišta deb paketa iz koga ćemo kasnije izgraditi korenski sistem datoteka. Dodatna pogodnost je što nije bilo potrebe za korišćenjem izlaznog repozitorijuma OBS-a koji je uređen prema pravilima uređivanja za .rpm pakete, već je korišćen direktorijum u koji OBS smešta pakete nakon što se završi njihovo prevođenje i pre nego ih smesti u odgovarajuće izlazno skladište. Ovi paketi ostaju u tom direktorijumu i nakon njihovog smeštanja na izlazno skladište OBS-a.

2.3.1 Reprepro

Reprepro[20] je alat za upravljanje skladištem za pakete OS Debian. On čuva datoteke, koje se mogu ili uneti ručno ili dobaviti iz nekog drugog skladišta, preslikane u odgovarajuću hijerarhiju. Paketi i datoteke se čuvaju uz pomoć baze podataka Berkley, tako da poslužilac baze podataka nije potreban. Podržana je kako provera potpisa preslikanih skladišta, tako i generisanje novih potpisa.

Reprepro je alat namenjen kreiranju skladišta APT sa strukturom kakvu koriste i zvanične replikacije skladišta OS Debian. Prevedeni binarni paketi se nalaze u direktorijumu „pool“.

Samo skladište može sadržati pakete za više različitih izdanja distribucije operativnog sistema Debian: Stable, Unstable, Testing itd. Može sadržati pakete za više različith arhitektura, recimo x86, sparc, mips, all, itd.

U postupku kreiranja skladišta najpre je potrebno instalirati alat reprepro. Jednom kada je alat instaliran može se krenuti u proces izgradnje skladišta koje je u saglasnosti sa zvaničnim načinom kreiranja skladišta namenjenih operativnom sistemu Debian u sledećih nekoliko koraka:

- Odlučiti gde se želi da arhiva bude locirana
- Napraviti konfiguracionu datoteku
- Uvesti prvi paket

Skladište može biti kako na lokalnoj mašini tako i na udaljenom računaru.

Potrebno je napraviti odgovarajuću struktruru direktorijuma u koju će se smeštati paketi i konfiguraciona datoteka. Treba napraviti sledeće direktorijume:

```
mkdir -p /putanja_do_skladišta/apt
mkdir -p /putanja_do_skladišta/apt/conf
mkdir -p /putanja_do_skladišta/incoming
```

Sada kada imamo direktorijum koji sadrži naše skladište možemo pristupiti kreiranju datoteke sa podešavanjima. Ova datoteka će odrediti za koje izdanje OS Debian je skladište namenjeno (squeeze, wheezy, sid). Skladište može biti podešeno tako da sadrži pakete više različitih izdanja distribucije Debian. Paketi mogu biti namenjeni za jednu, više ili sve (podržane) arhitekture.

Kada se podešavanja sačuvaju na lokaciji `conf/distributions` može se pristupiti procesu uvođenja paketa u skladište. Paketi se u skladište mogu uneti koristeći u formatu naredbe .changes ili .deb datoteke. Naredba ima sledeći oblik:

```
reprepro -Vb . include ime_paketa putanja_i_ime_datoteke
```

Pakete je moguće i ukloniti sa skladišta uz pomoć sledeće komande:

```
reprepro -b . remove ime_paketa reprepro
```

Prilikom dodavanja novijeg izdanja (verzije) paketa nije potrebno uklanjati starije izdanje, alat će sam izvršiti zamenu starijeg paketa novijim.

Alat poseduje i mogućnost automatizovanog podizanja paketa i izgradnje skladišta korišćenjem alata cron i jednostavnih skripti. Paketi se šalju na lokaciju gde je podešen alat reprepro u direktorijum „incoming“. Potrebno je još pomoću alata cron podesiti izvršavanje skripte koja u zakazano vreme pretražuje ulazni direktorijum „incoming“ i pretraži datoteke sa nastavkom ".changes" pa vrši uvođenje pronađenih paketa u skladište. Prilikom uvođenja paketa u skladište važno je brisati ono što smo obradili iz direktorijuma „incoming“ kako ne bi dolazilo do višestrukog uvođenja identičnog paketa u skladište.

Alat reprepro olakšava proces izrade repozitorijuma i izradu posebnih datoteka u saglasnosti sa programom apt, kao i dodavanje i uklanjanje paketa u repozitorijum.

2.3.2 Debootstrap

Debootstrap[21] je alat koji instalira sistem baziran na distribuciji Debian u direktorijum drugog već pokrenutog sistema. Potrebno je alatu definisati adresu ulaznog repozitorijuma sa prevedenim binarnim paketima. Zatim se korišćenjem komande debootstrap sa odgovarajućim parametrima kreira korenski sistem datoteka. Alatu se može navesti lista paketa koje treba da uključi prilikom izgradnje korenskog sistema datoteka. Pritom će alat debootstrap sam instalirati i sve propratne zavisnosti, tj. ukoliko je neki paket naveden probaće da uključi i sve pakete od kojih taj navedeni paket zavisi, pri čemu će i upozoriti ako naiđe na ovakvu situaciju. Ovo razrešavanje zavisnosti nije toliko efikasno u smislu da se može porediti sa funkcionalnošću apt ili dpkg komande pa se na njega ne može uvek osloniti. Zato je sigurnije pobrojati sve pojedinačne pakete koje želimo da budu uključeni u korenski sistem datoteka koji gradimo.

Komanda debootstrap ima oblik:

```
debootstrap [OPTION...] SUITE TARGET [MIRROR [SCRIPT]]
```

Alat debootstrap sam podiže osnovni sistem Debian od izdanja SUITE (lenny, squeeze, wheezy, sid) u direktorijumu TARGET sa lokacije MIRROR izvršavajući skriptu SCRIPT. Lokacija MIRROR može biti naveden u nekom od sledećih oblika

- `http:// URL adresa,`
- `file:/// URL adresa,`
- `ssh:/// URL adresa.`

Treba spomenuti da se file:/ prevodi na file:/// a da korišćenje file:// nije moguće.

Alat debootstrap ćemo kasnije ponovo pomenuti. Primer debootstrap komande korištene za pakovanje sistema Debian dat je u okviru dodatka u poglavlju 7.3.

2.3.3 Mic-Image-Creator

Za izgradnju korenskog sistema datoteka za distribuciju MeeGo koristi se alat mic-image-creator. Ovaj alat se nalazi u skupu alata pod nazivom MIC2[22].

MIC2 je skup alata namenjen za rad na distribuciji MeeGo. Ime MIC2 je dato kako bi se napravila razlika u odnosu na MIC ili Moblin Image Creator alat koji je zastareo i sastoji se od serija alata namenjenih kreiranju slika distribucije, konvertovanju slika distribucije, pravljenju korenskog sistema datoteka itd.

MIC2 je baziran na livecd-tools i appliance-tools alatima distribucije Fedora. Sa alatima MIC2 korisnici mogu stvarati različite tipove slika sistema za različite namene, uključujući slike sistema namenjene pokretanju direktno sa CD-a ili USB-a (live CD/USB images), zatim slike

sistema namenjene radu sa KVM, Vmware, VirtualBox, N900, slike sistema namenjene MeeGo razvojnim timovima (developers) itd. Moguće je manipulisati i transformisati slike sistema iz jedne u drugu vrstu slike.

MIC2 nudi sledeće alate:

- mic-image-creator - kreira slike sistema
- mic-image-convertor - vrši prevođenje iz jednog oblika u drugi.
- mic-chroot - obezbeđuje MeeGo okruženje za potrebe razvoja od „live/loop“ slike sistema. Može i da prevede „chroot“ u „live“ sliku sistema.
- mic-image-writer - koristi se za upis MeeGo slika na USB memorijski disk.

Treba naglasiti da MIC2 na PC računarima radi samo sa Intel procesorima koji u sebi imaju podršku za SSSE3 instrukcije.

Od interesa za izgradnju korenskog sistema datoteka je mic-image-creator koji koristi skripte sa nastavkom ".ks". Koristeći kickstart skripte korisnici mogu da navedu koje repozitorijume treba koristiti, koje pakete treba instalirati prilikom izgradnje korenskog sistema datoteka i mogu proslediti osnovne direktive za konfiguraciju sistema.

Primer korišćenja mic-image-creator naredbe:

```
sudo mic-image-creator --config=default.ks --format=loop  \
--cache=mycache
```

Moguće je proslediti i parametar `--arch=ciljna_arhitektura` kojim se definiše arhitektura za koju je korenski sistem datoteka namenjen.

Kako je kao sistem za prevođenje paketa i u slučaju distribucije MeeGo korišten OBS nije bilo potrebno dodatno prilagođavati izlazni repozitorijum paketa OBS prilikom izgradnje korenskog sistema datoteka distribucije MeeGo. Mic-image-creator je namenjen radu sa paketima sa nastavkom ".rpm" pa ovde nije bilo potrebno da se koristite posebni alati za organizovanje skladišta paketa pre same izgradnje korenskog sistema datoteka.

2.3.4 Febootstrap

Febootstrap[23] je alat izrađen po ugledu na alat debootstrap. Namenjen je radu sa distribucijom Fedora, ali je vremenom prerastao u alat koji se koristi i za druge distribucije Linuksa za izradu malih distribucija veličine oko stotinu KB. Treba napomenuti da je poželjno ne koristiti ga sa administratorskim pravima jer se prilikom rada (sa root pravima) može desiti da zaustaviti neke sistemske procese. Febootstrap može da se koristi za izgradnju korenskih sistema datoteka ali uz znatna ograničenja. Tako se uz pomoć ovog alata može izgraditi korenski sistem datoteka na željene distribucije, ali samo ako je alat pokrenut na toj istoj distribuciji (nije moguće recimo graditi korenski sistem distribucije Debina a pokretati alat na distribuciji SUSE). Takođe alat se može koristiti samo za izgradnju korenskih sistema datoteka za onu arhitekturu na kojoj je i pokrenut, što znači da je nemoguće graditi, recimo, korenski sistem datoteka za arhitekturu MIPS na operativnom sistemu koji se izvršava na i386 arhitekturi. Ukratko alat Febootstrap se ne može koristiti za izgradnju jednog sistema na drugom sistemu.

2.3.5 Mock

Mock[24] je alat namenjen radu sa Fedora distribucijom. Prvenstvena namena ovog alata je kreiranje korenskog sistema datoteka i prevođenja paketa u njemu. Iako se delom može koristi za prevođenje distribucije namenjene jednom sistemu na drugoj arhitekturi, ovo ne postoji kao mogućnost prema zvaničnim podatacima u dokumentaciji.

3. Verifikacija

Nakon što smo prevođenjem paketa dobili distribuciju za željenu platformu, suočili smo se sa potrebom verifikacije distribucije, dobijene prevođenjem. Kako potvrditi da se sistem ponaša predvidivo i u granicama očekivanog?

Postoji veliki broj rešenja namenjenih ispitivanju pojedinih aspekata sistema zasnovanih na Linuksu, međutim veliki broj njih je namenjen proveri stabilnosti elemenata fizičke arhitekture ili merenju performansi određenih operacija na ciljnom sistemu. Sistema koji testiraju ispravnost operativnog sistema koji se razvija nema puno.

Ovde ćemo obrazložiti dva sistema i dati detaljniji opis postupaka i programskih rešenja koje smo koristili u procesu verifikacije distribucija koje smo prevodili:

- Android CTS - Android Compatibility Test Suite
- LTP - Linux Test Project

3.1 Android CTS

Android Compatibility Test Suite[25] ili program ispitivanja saglasnosti za Android je zamišljen kao sredstvo koje treba da olakša proizvođačima uređaja zasnovanih na operativnom sistemu Android, razvoj uređaja u saglasnosti sa opeterativnim sistemom Android.

Cilj programa saglasnosti za Android je da radi u korist svih zainteresovanih strana, uključujući korisnike, programere i proizvođače uređaja.

Sve tri interesne grupe su međusobno povezane i zavisne. Korisnici uređaja žele široku ponudu raznovrsnih uređaja i aplikacija. Aplikacije razvijaju programeri motivisani mogućnostima plasiranjem programa na tržišta sa visokim brojem potencijanih korisnika. Proizvođači uređaja zasnovanih na operativnom sistemu Android imaju koristi od velikog broja dostupnih aplikacija jer one podižu vrednost uređaja zasnovanih na operativnom sistemu Android u očima korisnika.

Ideja CTS-a je da ponudi sledeće prednosti od kojih će sve tri grupe imati koristi:

1. Da obezbedi postojana, održiva programska rešenja i elementne fizičke arhitekture. Bez utvrđenog standarda saglasnosti (kompatibilnosti), uređaji bi mogli da variraju u toj meri da programeri moraju da dizajniraju različite verzije softvera namenjene različitim uređajima. Obrazac saglasnosti precizno definiše šta programeri mogu da očekuju od saglasnog uređaja, kako u pogledu sučelja namenjenog pisanju programa, tako i u podgledu mogućnosti usklađenih uređaja.

Programeri mogu koristiti ove informacije i prilikom dizajniranja programskih rešenja i biti sigurni da će njihovo rešenje raditi na saglasnim uređajima kojima je namenjeno.

2. Da korisnicima uređaja obezbedi postojano okruženje u radu sa programskim rešenjima. Ukoliko jedno programsko rešenje radi na jednom uređaju saglasnom sa izdanjem operativnog sistema Android, trebalo bi da podjednako dobro radi i na drugom, sličnom, uređaju saglasnom sa istom verzijom operativnog sistema Android. Kako se uređaji bazirani na operativnom sistemu Android razlikuju po mogućnostima fizičke arhitekture, tako i prema podržanom programskom rešenju, program saglasnosti omogućava izgradnje jednostavnog sistema filtera za programska rešenja poput „Google play“, koje na osnovu tabele saglasnosti uređaja, korisniku nudi samo ona programska rešenja koja su u saglasnosti sa njegovim tipom uređaja.
3. Da proizvođačima uređaja zasnovanih na programskom rešenju tj. operativnom sistemu Android omogući da ponude različite uređaje korisnicima i time budu drugačiji od ostalih proizvođača, a da pritom uređaji budu saglasni sa odgovarajućom verzijom operativnog sistema Android. Program saglasnosti za operativni sistem Android se fokusira na aspekte potrebne za pokretanje programa od strane trećih lica (ni proizvođača uređaja, ni razvojnog tima operativnog sistema Android) i ostavlja proizvođačima uređaja prostor da kreiraju jedinstvene uređaje a da ostanu saglasni (kompatibilni) sa programskim rešenjima namenjenim operativnom sistemu Android.
4. Da smanji troškove i potencijalni dodatni rad kako bi se postigla saglasnost uređaja. Kako bi se proizvođačima uređaja omogućilo da na jednostavan i jeftin način saglasnost postignu ponuđen je alat za ipitivanje saglasnosti - CTS (compatibility testing suite). Alat za ispitivanje saglasnosti - CTS je besplatno programsko rešenje otvorenog koda dostupno za preuzimanje i korišćenje. Zamišljeno je kao alat za stalno samoispitivanje u procesu izrade uređaja zasnovanog na operativnom sistemu Android, kako bi se izbegla potreba da se proces više puta ponavlja i kako bi se korišćenjem CTS-a izbegla potreba slanja uređaja na ispitivanje i sertifikaciju kod trećih lica.

Program saglasnosti Android se sastoji od sledećih ključnih komponenti:

- Izvornog koda operativnog sistema Android (Andorid OS)
- CDD (Compatibility Definition Document) - dokumenta kojim se definišu uslovi saglasnosti.
- CTS (Compatibility Test Suite) - mehanizma za ostvarivanje saglasnosti.

Kako za različite verzije operativnog sistema Android postoje i zasebna grananja izvornog koda tako i za svaku verziju operativnog sistema Android postoji i poseban alat za ispitivanje saglasnosti CTS i dokument kojim se definišu uslovi saglasnosti CDD. Pored uređaja od interesa i specifičnih programskih rešenja, sve što je potrebno za izradu saglasnog uređaja sa operativnim sistemom Android su izvorni kod, alat CTS, dokument CDD.

3.1.1 CDD (Compatibility Definition Document)

Detaljni dokument koji definiše uslove saglasnosti - CDD daje se za svako izdanje operativnog sistema Android.

Nijedno ispitivanje, uključujući i CTS koji ćemo kasnije razmatrati, ne može biti sveobuhvatno, npr. CTS sadrži ispitni scenario koji ispituje OpenGL grafičku programsku spregu namenjenu pisanju programske podrške (OpenGL grafički API) ali ispitni scenario sam po sebi

ne može potvrditi da se je grafika ispravno prikazala na uređaju i ne postoji način da ispitni scenario potvrdi da li postoje, tj. da li su ugrađene, kamera ili tastatura na uređaju koji se ispituje.

Uloga dokumenta CDD je da definiše i pojasni zahteve i eliminiše sve dvosmislenosti i nedoumice vezane za saglasnost uređaja. Dokument CDD nije zamišljen tako da bude sveobuhvatan jer operativni sistem Android je programsko rešenje otvorenog koda pa je sam po sebi specifikacija platforme. Dokument CDD daje smernice i uputstva kao i reference na druge sadržaje koji pružaju okvir u kojem se izvorni kod Androida može koristiti tako da krajnji rezultat bude saglasni sistem.

3.1.2 CTS (Compatibility Test Suite)

Programsko rešenje za ispitivanje saglasnosti, CTS, je besplatno, standardizovano programsko rešenje koje je dostupno za slobodno preuzimanje. Predstavlja mehanizam za postizanje saglasnosti. Izvršava se na stonom računaru a testove pokreće ili na uređaju povezanom sa računarem ili na emulatoru. Programsko rešenje CTS je skup jediničnih scenarija ispitivanja (unit testova) dizajniranih tako da budu kontinuirano korišćeni od strane inženjera u svakodnevnom radu, na razvoju uređaja. Ispitivanja doprinose ranom otkrivanju nesaglasnosti programskih rešenja i osiguravaju da programska rešenja ostanu saglasna sa specifikacijom kroz ceo razvojni proces.

Alat CTS je automatizovano rešenje za ispitivanje koje se sastoji iz dve bitne programske komponente:

- Okruženja koje se izvršava na stonom računaru i upravlja ispitivanjem priključenog uređaja.
- Individualnih ispitnih scenarija koji se izvršavaju na priključenom mobilnom uređaju ili emulatoru. Ispitni scenariji su pisani na programskom jeziku Java kao JUnit ispitivanja i zapakovani kao datoteke sa nastavkom ".apk" operativnog sistema Android kako bi se izvšavali na ciljnom uređaju ili emulatoru.

Za pokretanje ispitivanja potreban je barem jedan uređaj ili jedan emulator. Po pokretanju programskog rešenja za ispitivanje saglasnosti, ispitno okruženje najpre kopira datoteke sa nastavkom ".apk" na priključene uređaje ili emulatore pa pokreće ispitivanje i beleži rezultate. Samo ispitivanje se vrši na priključenim uređajima ili emulatorima. Nakon izvršenih ispitivanja i prikupljanja rezultata programsko rešenje zaduženo za ispitivanje uklanja .apk datoteke sa priključenih uređaja ili emulatora.

Alat CTS sadrži sledeće tipove ispitnih scenarija:

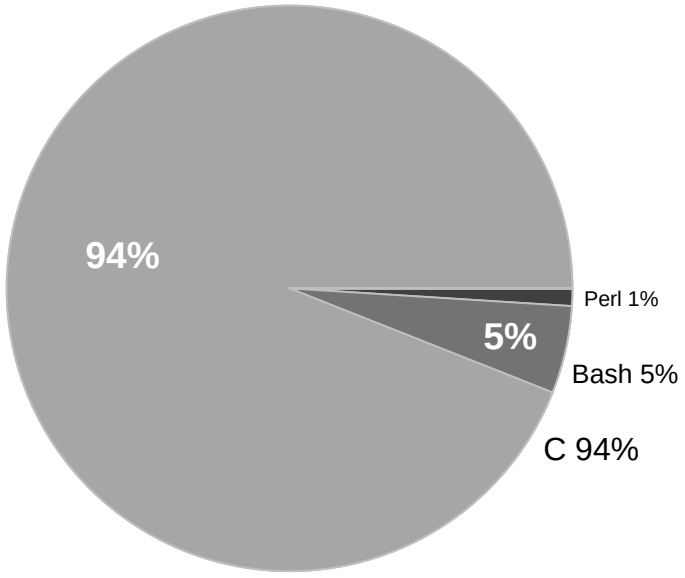
- Trenutno dostupni:
  - Ispitivanja Junit - ispitivanje atomskih jedinica koda u okviru operativnog sistema Android, kao što je npr. java.util.HashMap
  - Ispitivanja funkionalnosti - ispitivanje programskih spregi namenjenih pisanju programa zajedno sa naprednijim scenarijima upotrebe.
  - Referentna ispitivanja programa - ispitivanje kompletnog uzorka softverskog rešenja sa izvršavanjem punog skupa programskih spregi namenjenih pisanju programa i servisa koji se koriste prilikom izvršavanja programa na operativnom sistemu Android.
- Planirani za buduće verzije CTS-a:
  - Ispitivanje robustnosti - ispitivanje izdržljvosti sistema prilikom stresa.
  - Ispitivanje performanse - ispitivanje performanse sistema pomoću programa za referentno ispitivanje npr. broj obrađenih i iscrtanih slika u sekundi (fps)

3.2 LTP - Linux Test Project

Prilikom procesa izrade distribucija suočili smo se sa potrebom da verifikujemo distribucije koje razvijamo. U potrazi za alatom koji bismo mogli da upotrebimo za automatsko testiranje susreli smo se sa LTP-om[26] i odabrali za početak verziju koja dolazi uz distribuciju Debian squeeze, kako bi razmotrili mogućnosti LTP-a[27].

Kao dobra polazna tačka za razumevanje LTP-a uzet je rad Paula Larsona objavljen 2002 godine - „*Testing Linux® with the Linux Test Project"*[28].

Linux Test Project (kraće LTP) je projekat otvorenog koda sa ciljem da stavi na raspolaganje ispitne scenarije otvorenog koda koji služe za verifikaciju pouzdanosti, robusnosti i stabilnosti Linuksa. LTP ispitno okruženje je skup automatizovanih i poluautomatizovanih ispitivanja namenjenih ispitivanju različitih aspekata operativnog sistema Linuks. Većina ispitnih scenarija (94%) je napisana u programskom jeziku C, zatim 5% Bash skripti i manje od 1% Perl koda, slika 3.1.



3.1 - struktura izvornog koda u LTP-u

Istorijski, ispitivanje Linuksa je često rađeno površno i ad-hok metodom. Korisnici bi prilikom korišćenja Linuksa nailazili na probleme i iste prijavljivali. Nije bilo mnogo mogućnosti za organizovano ispitivanje Linuksa.

Pojavom LTP maja 2000. godine, kada Silicon Graphic Inc. objavljuje prvu verziju LTP-a stvari se menjaju na bolje.

LTP sadrži ispitivanja koja se mogu zasebno pozivati, predefinisane skripte koje izvršavaju više ispitnih scenarija jedan za drugim i upravljački program za postupak ispitivanja (pan) koji parsira datoteku sa spiskom ispitivanja koje treba izvršiti. Izvršava ispitivanja i izlazi sa 0 ako su sva ispitivanja uspešno izvršena, odnosno sa brojem koji pokazuje koliko ispitivanja nije uspešno izvršeno.

LTP ispitivanja su svrstana u nekoliko grupa:

- Kernel – Ispitni scenariiji za ispitivanje jezgra operativnog sistema Linuks kao filesystems io, ipc, memory managment, scheduler, i system calls.
- Network – Ispitni scenariji za ipv6, multicast, nfs, rpc, sctp, i komande namenjene radu sa mrežom.
- Command – Ispitni scenariji komandi korisničkog nivoa koje se često koriste u razvoju programskih rešenja kao što su ar, ld, ldd, nm, objdump i size.
- Misc - Razni ispitni scenariji koji se ne mogu svrstati u neku od prethodno navedenih grupa kao ispitni scenario „crashme“ i matematička ispitivanja (floating point).

Kao što smo već ranije rekli, LTP nudi i nekoliko ispitnih skripti sa predefinisanim nizom ispitnih scenarija koji se izvršavaju jedan za drugim:

- runalltests.sh - pokreće skup svih automatizovanih ispitivanja za ispitivanje jezgra operativnog sistema Linuks, koje izvršava sekvencijalno
- network.sh - pokreće sva automatizovana mrežna ispitivanja u sekvencijalnom redosledu.
- diskio.sh - pokreće stress_floppy i stress_cdrom ispitivanja.

Iako runalltest.sh svojim imenom sugeriše izvšavanje svih ispitivanja, ova skripta ipak ne poziva sva raspoloživa ispitivanja u okviru LTP-a. Ona pokreće sva nedestruktivna ispitivanja koja su zamišljena da se izvršavaju samostalno, automatski, bez potrebe da korisnik svojeručno podešava ispitne scenarije.

Kako samo ime sugeriše, network.sh skripta sadrži većinu mrežnih ispitivanja. Ova ispitivanja su izdvojena u zasebnu grupu zato što je za njihovo pokretanje potrebno ispuniti dodatne uslove kako bi funkcionisali korektno. Potrebne su dve mašine da bi se izvršila sva mrežna ispitivanja. Obe mašine moraju imati prevedenu istu verziju LTP-a i ona mora biti instalirana na istoj apsolutnoj putanji na obe mašine. Mašina (računar) koja ima ulogu klijenta je zadužena za pokretanje skripte network. Na mašini koja igra ulogu poslužioca .rhost se mora podesiti za root korisnika kako bi bilo dozvoljeno ostvarivanje komunikacije sa klijent mašinom.

Poslednja je skripta diskio. Ona predstavlja mali skup ispitivanja koji pokreće dva intenzivna ulaz-izlaz (IO) ispitna scenarija. Jedan je namenjen ispitivanju rada uređaja za čitanje optičkih diskova (cd-rom), a drugi ispitivanju rada disketne jedinice. Za izvšavanje ovih testova potrebno je u cd-rom ubaciti cd sa podacima, dok je za izvršavanje floppy testa, neophodna prazna formatirana disketa unutar disketne jedinice.

LTP upravljački program za ispitivanje, koji je zadužen za izvršavanje ispitivanja i izveštavanja o uspešnosti ispitivanja nosi naziv ltp-pan. Ltp-pan parsira datoteku u kojoj je navedena lista ispitivanja koje treba izvršiti, izvršava ih i vraća vrednost 0 ukoliko su sva ispitivanja uspešno izvršena ili broj različit od nule koji predstavlja broj ispitivanja koja se nisu uspešno izvršila. Oslanjanjem na ltp-pan mogu se odabrati željenja ispitivanja i prepustiti izvršavanje ispitivanja i izveštavanje ltp-pan-u kako bismo kasnije samo pregledali rezultate ispitivanja koji će biti smešteni u datoteku. Ispitivanja se mogu izvršavati uz upotrebu upravljačkog programa ltp-pan ali i samostalno, ukoliko se ukaže potreba za tim.

Ispitivanja u odnosu na uspešnost izvršavanja mogu vratiti neku od sledećih poruka:

- TPASS – ispitivanje se izvršilo uspešno i vratilo očekivane vrednosti
- TFAIL – ispitivanje je dalo neočekivane vrednosti i nije se uspešno izvršilo.

- TBROK – ukazuje da su preostala ispitivanja iz tekućeg scenarija ne mogu izršiti ispravno jer neki preduslov nije ispunjen, npr. resurs nije dostupan.
- TCONF – test slučaj nije napisan da se izvršava na trenutnom hardveru ili konfiguraciji softvera ili nije moguće izvšiti to ispitivanje na jezgru operativnog sistema Linuks koje se koristi.
- TRETR – ispitni slučaj je povučen i više se ne preporučuje za upotrebu.
- TWARN – prilikom izvršavanja ispitnog slučaja došlo je do neočekivanog ili nepoželjnog ponašanja koje ne bi trebalo da na bilo koji način utiče na izvršavanje samog ispitivanja (npr. nemogućnost oslobađanja zauzetih resursa nakon završetka ispitivanja, brisanje privremenih datoteka).
- TINFO – daje korisne informacije o statusu ispitivanja. Ne utiče na rezultate ispitivanja i ne ukazuje na problem prilikom ispitivanja.

LTP pruža mogućnost odabira željenih ispitivanja uz upotrebu programa ltpmenu (koji ne mora biti uključen u svaku verziju LTP-a), koji je u obliku čarobnjaka sa mogućnošću odabira željenih ispitivanja i vremena izvršavanja ispitivanja (broj prolaza ili vremensko ograničenje) kao i odabira željenog mesta za smeštanje izveštaja ispitivanja. Sve ovo se može postići i iz komandne linije uz korišćenje odgovarajućih parametara.

Uvođenje ispitivanja LTP u redovnu upotrebu omogućilo nam je da kvalitativno uporedimo distribucije koje prevodimo i pomoglo da identifikujemo nedostatke. Kao primer otkrivenih nedostataka navešćemo paket libaio koji nije podržavao arhitekturu MIPS u punoj meri. Kada bi prilikom otkaza libaio trebao da vrati numeričku vrednost koda greške (errno), za arhitekturu MIPS, libaio bi uvek vraćao vrednost -1. Broj -1 ukazuje na grešku, ali ne pruža nikakve dodatne podatke o nastaloj grešci. Uz dato saznanje bili smo u mogućnosti da problem nakon istraživanja otklonimo.

4. Linuks za MIPS32 i MIPS64

Ovaj rad je nastao na iskustvima stečenim u procesu prevođenja distibucije MeeGo i distribucije Debian operativnog sistema Linuks za arhitekturu MIPS.

4.1 Prevođenje distribucije MeeGo

4.1.1 Distribucija MeeGo

Distribucija MeeGo[29] je nastala udruživanjem izvornog koda operativnog sistema Moblin kompanije Intel i operativnog sistema Maemo kompanije Nokia[30]. Zamišljena je kao operativni sistem otvorenog koda namenjenog za upotrebu u prenosnim uređajima, pametnim telefonima, potrošačkoj elektronici, auto elektronici za pružanje informativnih i zabavnih sadržaja, pametnim televizorima i drugim ugrađenim sistemima. Ovaj operativni sistem je trebalo da bude platforma koja će se izvršavati na više različitih fizičkih arhitektura i skratiti vreme potrebno za razvoj programskih rešenja za ugrađene uređaje. Projekat je najavljen februara 2010. na događaju Mobile World Congress a zvanično je otkazan u septembru 2011. Prvo izdanje je objavljeno maja 2010. a poslednje jula 2012.

Nakon što su Intel i Nokia napustili projekat MeeGo, rešenja projekta preuzeta su u projekat Mer[31]. Ovaj projekat razvija zajednica nezavisnih programera ranije okupljena oko projekta MeeGo. Na osnovu projekta Mer, Finska kompanija Jolla je razvila operativni sistem Sailfish OS koji se nalazi na pametnom telefonu Jolla Phone.

Operativni sistem MeeGo koristi programske pakete RPM, nastao je spajanjem dve izvedene distribucije od kojih je Moblin izveden od distribucije Fedora a distribucija Maemo izvedena od distribucije Debian. Koristi QT okvir (framework) a korisnička grafička sprega je optimizovana za prenosne uređaje. Za prevođenje paketa projekat MeeGo se oslanja na OBS.

Operativni sistem MeeGo je dostupan za arhitekture ARM i Intel x86.

4.1.2 Postupak prevođenja distribucije MeeGo

U ovom radu će biti obrazložen postupak prevođenja operativnog sistema MeeGo za arhitekturu MIPS. Operativni sistem MeeGo je preveden za procesna jezgra MIPS sa sledećim instrukcijskim skupovima:

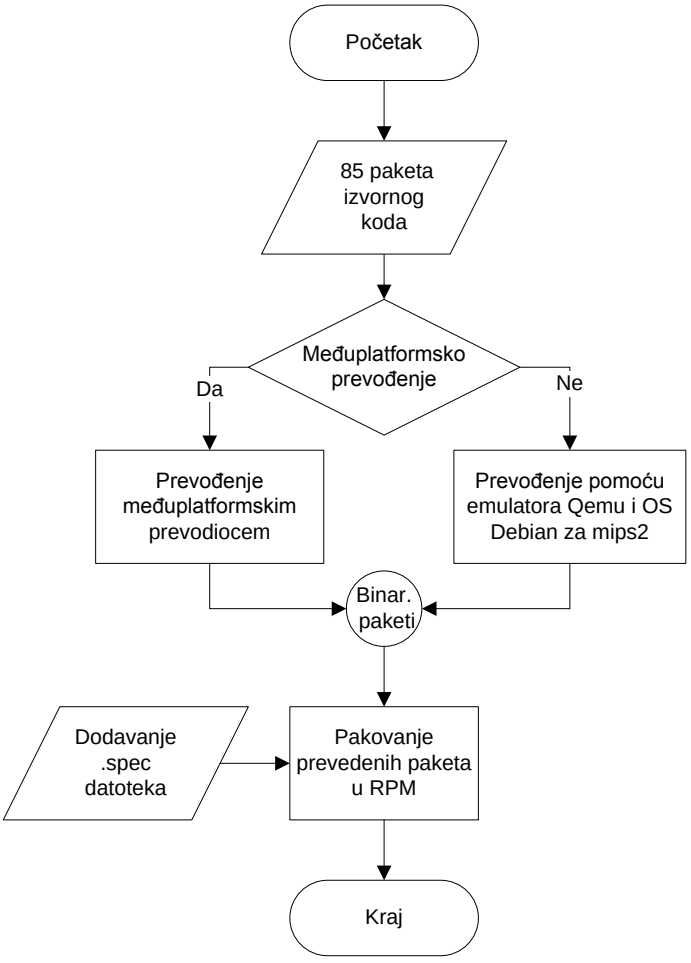
- mips32elr2 – MIPS32 revizija 2, little endian
- mips32elr2 fp64 – MIPS32 revizija 2, little endian, floating point unit 64 bit.

Distribucija MeeGo ne podržava ni jednu fizičku izvedbu arhitekture MIPS. Proces prevođenja je započinjao od paketa izvornog koda distribucije MeeGo koji su dobavljeni na ciljne računare na kojima je vršeno prevođenje. Osnovna grupa paketa koju je trebalo prevesti sastojala se od 85 paketa. Kompletan spisak prevođenih paketa dat je u dodatku 7.1.

Paketi su od izvornog koda paketa distribucije MeeGo prevođeni na 2 načina (slika 4.1):

- Prvi način je podrazumevao korišćenje među-platformskog prevodioca za prevođenje paketa. Lanac alata koji je pritom korišten je Code Sourcery, koji se sada može naći pod nazivom Mentor Graphics[32].
- Drugi način prevođenja je podrazumevao korišćenje alata QEMU u sistemskom režimu rada. Uz pomoć ovog alata je na arhitekturi Intel podignut operativni sistem Debian namenjen arhitekturi MIPS mipsel (mips little endian) sa istrukcijskim skupom mips2.

U oba slučaja paketi su prevođeni uz pomoć naredbi „configure“ i „make“ čime su dobijani binarni paketi koje je bilo potrebno zatim spakovati za distribuciju MeeGo.

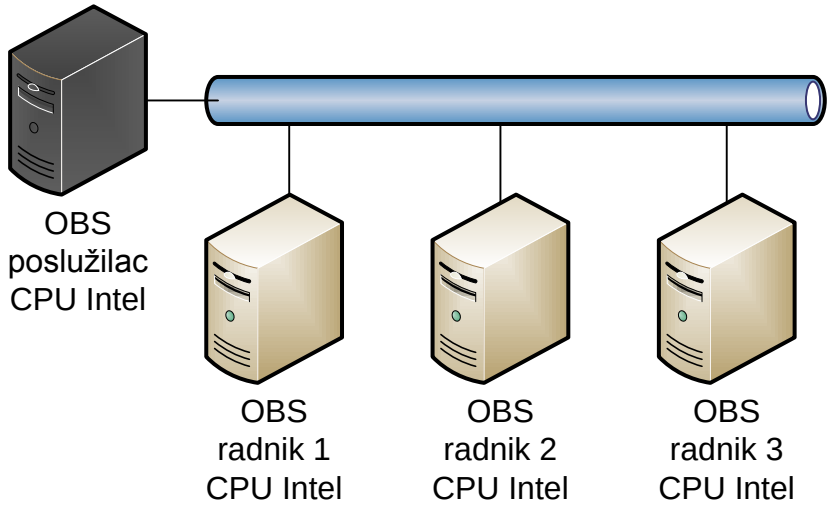


4.1 - distribucija MeeGo prevođenje osnovnog skupa paketa

Distribucija MeeGo pripada grupi distribucija koje koriste pakete RPM, te je bilo potrebno prevedene pakete spakovati u ovaj oblik. To je postignuto uz pomoć alata rpmbuild.

Kada je preveden osnovni skup od 85 paketa, taj skup dat je alatu OBS kao ulazni repozitorijum. Ovi paketi su potrebni i dovoljni da se izgradi minimalni sistem koji će poslužiti za dalje prevođenje paketa. Ta osnovna baza paketa je poslužila da se ti isti paketi prevedu na OBS-u.

OBS je za prevođenje paketa organizovan uz pomoć više računara sa procesorom Intel (slika 4.2). Na jednom od računara se nalazio poslužilac kome su kao resurs dodeljeni radnici tj. drugi računari sa CPU Intel. Jedan računar mogao je imati jednog ili više radnika. Broj radnika je usklađen sa brojem procesorskih jezgara koje poseduje računar.



4.2 - organizacija OBS-a za prevođenje distribucije MeeGo

Prevođenje paketa za arhitekturu MIPS na računarima sa procesnim jezgrima Intel je omogućeno uz pomoć datoteke binfmt. Tako je OBS konfigurisan da kada dođe do izvršavanja koda, koji nije namenjen arhitekturi na kojoj se vrši prevođenje na osnovu magičnog ključa, bude pozvan odgovarajući emulator, tj. da kod bude izvšren korišćenjem alata QEMU u korisničkom modelu rada.

Od prevedenih paketa distribucije MeeGo izgrađen je korenski sistem datoteka koji je predstavljao distribuciju MeeGo namenjenu arhitekturi MIPS.

Prevođenje paketa namenjenih arhitekturi MIPS32 za procesna jezgra sa instrukcijskim skupom mips32r2 little endian, bilo je olakšano činjenicom da prevodilac koji se isporučuje uz zvaničnu Debian distribuciju namenjenu arhitekturi MIPS može biti uz jednostavno prosleđivanje opcionog parametra podešen da prevodi pakete umesto za MIPS2 ISA za MIPS32R2 ISA:

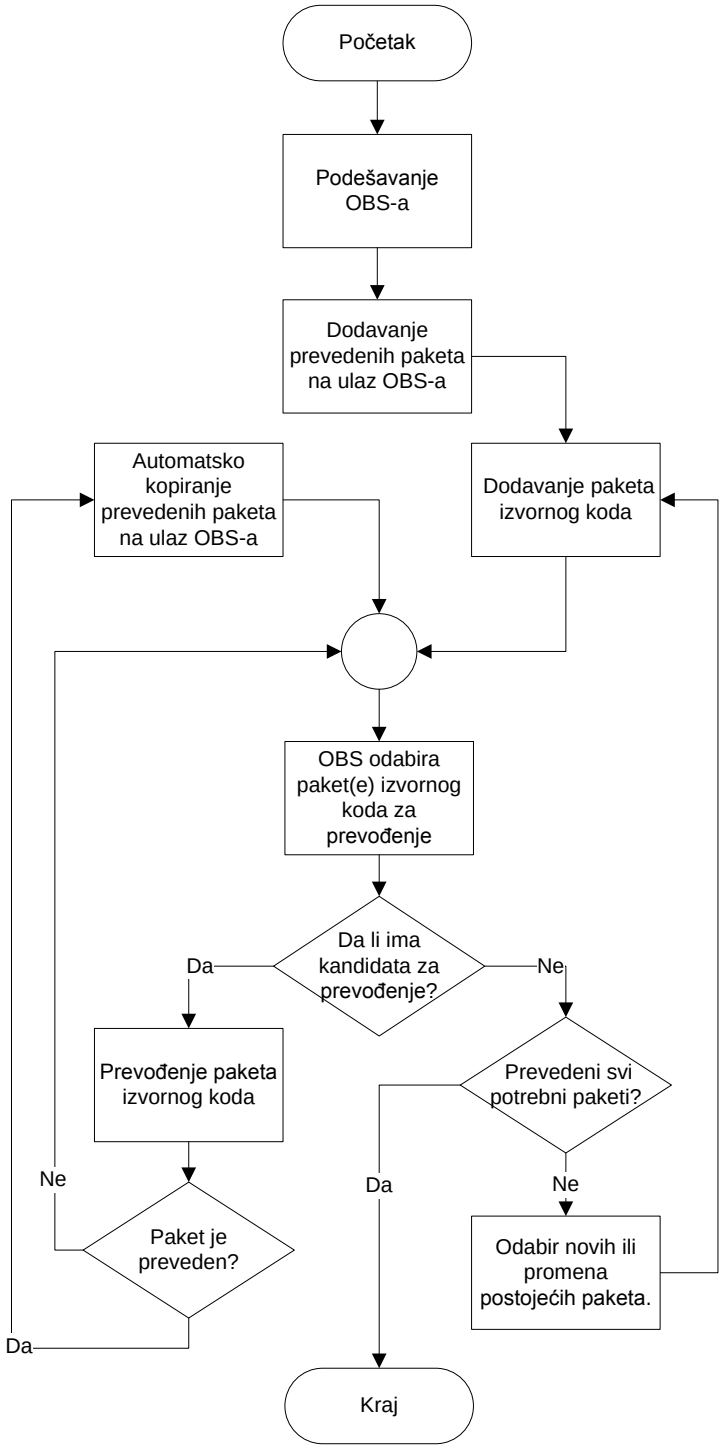
```
gcc -march=arch
arch: ‘mips1’, ‘mips2’, ‘mips3’, ‘mips4’, ‘mips32’, ‘mips32r2’, ‘mips32r3’, ‘mips32r5’, ‘mips64’, ‘mips64r2’, ‘mips64r3’ and ‘mips64r5’
```

Ukoliko postoji potreba, paket može biti uvezen uz pomoć biblioteka paketa koji su prevedeni za MIPS2 ISA. Ovo je moguće jer instrukcijski skup mips32r2 predstavlja proširenje instrukcijskog skupa mips2. To je u velikoj meri olakšalo obezbeđivanje nedostajućih zavisnosti prilokom prevođenja paketa.

Binarni paketi dobijeni ovakvim kombinovanjem više različitih ISA će prilikom provere vratiti informaciju o najkompleksnijoj od svih ISA koja je korišćena prilikom formiranja paketa.

Jednom dobijeni osnovni skup paketa se koristiti najpre za ponovno prevođenje tih istih paketa na OBS-u (slika 4.3), ali sada uz korišćenje prevodioca koji je tako podešen i preveden da pakete prevodi na arhitekturi za koju se paketi i prevode tj. na MISP32 za MIPS32. Ovo se radi

kako bi se izbegla mogućnost da određeni delovi koda budu optimizovani za instrukcijski skup mips2 i time ne koriste prednosti koje donosi proširenи instrukcijski skup mips32r2.



4.3 - proces prevođenja paketa na OBS-u

Sledeći korak je prevođenje proširene razvojne baze koja sadržali 326 paketa. Nakon ovoga pristupa se prevođenju preostalih paketa distribucije MeeGo. Proces prevođenja se zaustavlja nakon što je prevedeno preko 1000 paketa izvornog koda distribucije MeeGo za arhitekturu MIPS32.

Prevođenje paketa uz pomoć alata OBS je pojednostavljeno mogućnošću da se na nivou projekta definišu opcioni parametri prevodioca i uvezivača i na taj način olakša proces

prevođenja paketa. Sintaksa koja se koristi za podešavanje kofiguracione datoteke projekta je vrlo slična sintaksi koja se koristi u datotekama sa nastavkom ".spec" RPM paketa.

Sledeći primer ilustruje prosleđivanje parametara prevodiocu kroz konfiguracionu datoteku projekta, za repozitorijum „standard“ projekta MeeGo namenjenog za mips32elr2

```
%if "%_repository" == "standard"

# optflags for mipsel
# added by lade
################################################################
Optflags: mipsel %{__global_cflags} -O2 -g -mips32r2 -EL -mhard-float
################################################################

%endif
```

Specifičnost prevođenja distribucije MeeGo za arhitekturu MIPS32 sa instrukcijskim skupom mips32elr2 fp64 se ogledala u sledećem:

- Nije postojao dostupan međuplatformski prevodilac koji bi kao rezultat prevođenja dao pakete namenjene za arhitekture zasnovane na ovom instrukcijskom skupu, pa je isti najpre bilo potrebno konfigurisati i prevesti.
- Paketi dostupni za MIPS2 ISA nisu se mogli koristiti za uvezivanje sa paketima namenjenim za mips32elr2 fp64 pa je proces prevođenja osnovnog skupa paketa bio zahtevniji.

4.2 Prevođenje distribucije Debian

4.2.1 Distribucija Debian

Distribucija Debian[33] operativnog sistema Linuks je poznata po obilju opcija. Svako izdanje (eng. release) ima više od 29.000 paketa za 11 različitih arhitektura koje koriste jezgro operativnog sistema Linuks. Postoje paketi za arhitekture koje koriste FreeBSD jezgro operativnog sistema Linuks (kfreebsd-i386 i kfree-amd64) ili Hurd jezgro operativnog sistema Linuks čineći Debian jedinim operativnim sistemom koji podržava 3 različita jezgra operativnog sistema Linuks. Podržane arhitekture se kreću od Intel/AMD 32-bitnih i 64-bitnih do ARM, MIPS arhitektura i IBM eServer zSerija.

Debian je 16. avgusta 1993. prvi put objavio Ian Murdok (Ian Murdock). On je sistem nazvao Debian Linux Release. Prethodnik distribucije Debian bila je distribucija Softlanding Linux System (SLS), distribucija sastavljena od različitih paketa koja je bila popularna osnova za distribucije 1993-1994. Loše održavanje i previše grešaka u SLS motivisali su Murdoka da kreira novu distribuciju.

1993. godine objavljen je proglas (manifest) Debian kojim se pozivalo na otvorenost u duhu Linuksa i GNU-a.

Projekat Debian je polako rastao i prve 0.9x verzije su proizišle 1994. i 1995. godine. U to vreme projekat je sponzorisao Free Software Foundations GNU Project–a. Prva prevođenja za druge, ne i386 arhitekture, počele su 1995. i prva 1.x verzija Debiana je izdata 1996. godine.

Trenutno aktuelno izdanje distribucije Debian je Debian 7.0 „wheezy“.

4.2.2 Postupak prevođenja distribucije Debian

Za arhitekturu MIPS prevodili smo dva izdanja distribucije Debian:

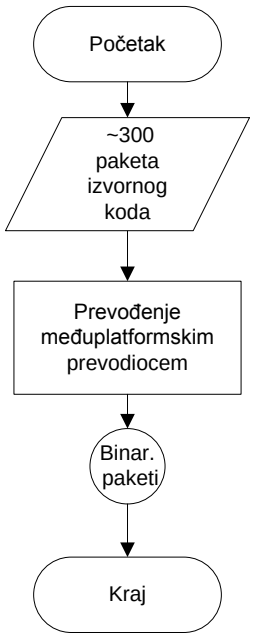
- Izdanje Debian 6.0 squeeze za procesorska jezgra sa instrukcijskim skupom:
  - mips32el – MIPS32, little endian
  - mips32elr2 – MIPS32 revizija 2, little endian
  - mips32elr2 fp64 – MIPS32 revizija 2, little endian, floating point unit 64 bit.
  - mips64elr2 – MIPS64 revizija 2, little endian
- Izdanje Debian 7.0 wheeze za procesorska jezgra sa instrukcijskim skupom:
  - mips32r2 – MIPS32 revizija 2, big endian
  - mips32elr2 – MIPS32 revizija 2, little endian
  - mips64r2 – MIPS64 revizija 2, big endian

Na prevođenje paketa distribucije Debian za arhitekturu MIPS uticala su iskustva stečena u procesu prevođenja distribucije MeeGo za arhitekturu MIPS.

Osnovni skup paketa koji je ručno prevođen za ciljanu arhitekturu je proširen najpre na oko 100 paketa a kasnije na približno 300 paketa. Broj paketa je proširivan kako bi se što pre krenulo u prevođenje većeg broja paketa uz pomoć OBS-a i što bolje uposlili dostupni resursi. Prošireni skup paketa je dobijen praćenjem međusobnih zavisnosti paketa. Kompletan spisak paketa dat je u dodatku 7.2

Za ručno prevođenje paketa je korišten među-platformski prevodilac (slika 4.4) i to za:

- MIPS32 preuzet sa Code Sourcery-a,
- MIPS32 fp64 je morao biti dodatno konfigurisan i preveden
- MIPS64 je morao biti dodatno konfigurisan i preveden



4.4 - distribucija Debian prevođenje osnovnog skupa paketa

Osnova od koje je započinjalo prevođenje paketa je izvorni kod paketa distribucije Debian. Prevođenje je vršeno uz pomoć među-platformskog prevodioca na računarima sa CPU Intel uz korištenje programskog alata dpkg-buildpackage koji je zadužen za automatizovanje procesa prevođenja paketa distribucije Debian i alata dpkg-cross koji olakšava međuplatformsko prevođenje paketa. Alat dpkg-cross prevodi paket tako da paket namenjen za određenu

arhitekturu može biti instaliran na bilo kojoj drugoj arhitekturi ali na alternativnoj putanji. Na ovaj način obezbeđuju se potrebne zavisnosti koje će se dalje koristiti za međuplatformsko prevođenje paketa.

Primer instaliranja paketa prevedenog za mips64el na arhitekturi x86:

```
mkdir ../cross-debs/
cd ../cross-debs/
dpkg-cross -a mips64el -b ../*.deb
dpkg -i --force all *.deb
```

Zatim je bilo potrebno obezbediti prevodilac koji će prevoditi pakete namenjene ciljanoj arhitekturi na toj istoj arhitekturi. Pokretanjem samog prevođenja paketa na OBS-u, uočeno je da source paketi distribucije Debian neretko menjaju parametre prevodioca i uvezivača tako što, umesto da ih proširuju dodatnim podešavanjima npr. CC += "", menjaju svojim podešavanjima CC= "". Ovo je obesmislilo podešavanje ovih parametara na nivou projekta. Ukoliko bismo se odlučili da zadržimo pristup podešavanja ovih parametara uz pomoć konfiguracione datoteke projekta morali bismo da menjamo veliki broj paketa izvornog koda.

Rešenje ovog problema pronađeno je u obezbeđivanju namenskog prevodioca koji je specijalno konfigurisan da prevodi pakete za ciljani instrukcijski skup određene arhitekture a da mu u toku prevođenja nije potrebno proslediti opcione parametre.

Kako je i distribucija Debian prevođena na OBS-u (slika 4.3), ali za više različitih izvedbi arhitekture MIPS, i kako je za svaku od izvedbi bilo potrebno obezbediti namenski prevodilac, odlučeno je da se iz osnovnog projekta, gde su se nalazili paketi izvornog koda koji su identični za sve arhitekture izdvoji grupa paketa koja će zbog sprecifičnosti prevodioca biti drugačija za arhitekture MIPS sa npr. instrukcijskim skupom mips32r2 i instrukcijskim skupom mips64elr2. Ova grupa je organizovana kao podprojekat pa je projekat prevođenja distribucije Debian na OBS-u imao onoliko podprojekata koliko je različitih prevodilaca bilo potrebno obzbediti, tj. za koliko se je različith izvedbi arhitekture MIPS distribucija Debian prevodila. Ovu grupu činili su sledeći paketi:

- gcc – prevodilac, jedno ili više različitih izdanja
- binutils – kolekcija binarnih alata od kojih su najrelevantniji:
  - ld – uvezivač
  - as – asembler
- eglibc – izdanje biblioteke C namenjeno ugrađenim sistemima, odlikuje se boljom podrškom za međuplatformsko prevođenje.
  - Dalji razvoj ovog alata je zaustavljen, a ciljevi postavljeni pred ovaj alat su sada preneseni na sam GLIBC tj. C biblioteku.

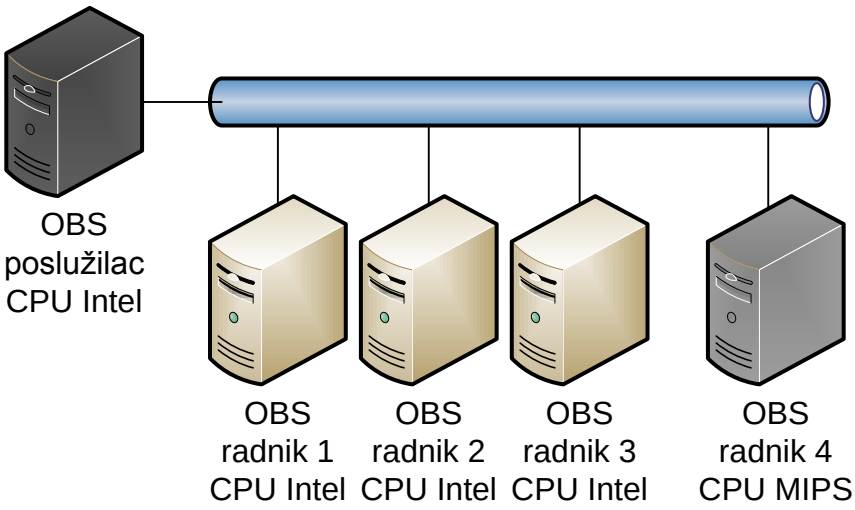
Paketi koji su dati OBS-u kao ulaz su zatim prevođeni na OBS-u uz pomoć specijalno podešenog prevodioca koji prevodi pakete za ciljanu arhitekturu na samoj toj arhitekturi bez potrebe da mu se prosleđuju parametri.

Kako bismo mogli organizovati prevođenje paketa za željenu izvedbu arhitekture OBS-om uz korišćenje računara sa procesorima arhitekture Intel bilo je potrebno podesiti da kada se detektuje izvšavanje stranog koda on bude izvšen korišćenjem emulatora QEMU u korisničkom modelu rada. Ovo je kao i slučaju prevođenja distibucije MeeGo postignuto korišćenjem datoteke binfmt i magičnog ključa.

Korščenje emulatora QEMU je bilo moguće za sve izvedbe arhitekture MIPS32 dok za arhitekturu MIPS64 u momentu prevođenja distribucije Debian emulator QEMU sa sistemskim modelom rada nije postojao, dok je korisnički model rada bio problematičan.

Kako nam je u tom momentu bila dostupna razvojna platforma za MIPS64 odlučeno je da se za MIPS64 prevođenje uz pomoć OBS-a vrši bez posredovanja emulatora i da se osnovni skup paketa koji je dat OBS-u kao ulazni iskoristi kao minimalni sistem koji će biti podignut na ovoj razvojnoj platformi. Zatim je OBS poslužiocu dodeljena razvojna platforma MIPS64 (slika 4.5) i na njoj se vršilo prevođenje paketa za MIPS64. Na OBS-u je bilo potrebno još onemogućiti

korišćenje radnika na Intel baziranim računarima za prevođenje paketa namenjenih arhitekturi MIPS64.



4.5 - organizacija OBS-a za prevođenje distribucije Debian

Minimalni skup paketa koji je podignut na ovoj razvojnoj platformi je prvi operativni sistem za arhitekturu MIPS64.

Nakon završenog prevođenja osnovnog skupa paketa trebalo je preći na prevođenje proširenog skupa paketa. Kako distribucija Debian sadži preko 29000 paketa bilo je potrebno na osnovu nekog kriterijuma odabrati pakete koju su relevantni i njih prevesti. Za ovo smo se oslonili na distribuciju Emdebian. Uz pomoć ove distribucije izdvojili smo skup od preko dve hiljade paketa. Izdvojeni skup paketa je dalje prevođen uz korišćenje alata OBS.

4.2.3 Distribucija Emdebian

Distribucija Emdebian[34] je mala distribucija zasnovana na distribuciji Debian i u saglasnosti je sa distribucijom Debian. Ova mala distribucija je podprojekat distribucije Debian i deo je projekta Embedded Debian.

Sama distribucija Debian se odlikuje prednostima koje je kvalifikuju za kandidata za primenu na različitim sistemima:

- namenjena za više različitih arhitektura,
- projekat je otvorenog koda,
- poseduje veliku programsku bazu

Ali je distribucija Debian primarno namenjena sistemima koji ne oskudevaju u resursima, poseduju veliku količinu memorijskog i skladišnog prostora.

Embedded Debian ili Emdebian je sistem dobijen od distribucije Debian sa smanjenim zahtevima za memorijom, a da su pritom osnovne funckionalnosti operativnog sistema ostale dostupne. Podržan je manji broj arhitektura u odnosu na distribuciju Debian, 7 u odnosu na 12.

Kako je OS Emdebian u saglasnosti sa OS Debian, određeni paketi koji se ne nalaze u distribuciji Emdebian po potrebi se mogu preuzeti od distribucije Debian i instalirati na sistem. Alat razvijen za ove potrebe je apt-grip. Paketi distribucije Emdebian su u suštini paketi distribucije Debian oslobođeni delova koji nisu neophodni u odnosu na funkcionisanje sistema, poput dokumentacije. Stoga se isti paketi mogu dobiti od paketa distribucije Debian, za šta je razvijen alat emdebian-grip.

Paketi distribucije Emdebian se mogu naći i pod terminom Grip ili Emdebian Grip.

U odnosu na distribuciju Debian, Emdebian ima gotovo deset puta manje paketa u svom sastavu. Oko tri hiljade paketa u distiribuciji Emdebian naspram preko dvadeset devet hiljada paketa distribucije Debian.

Razvoj distribucije Emdebian je prekinut jula 2014. Poslednje dostupno izdanje je Emdebian Grip 3.1 izvedeno od izdanja Wheezy 7.1 distribucije Debian.

Nedostatak memorijskih resursa kod ugrađenih arhitektura predstavljao je osnovni razlog za započinjanje projekta Emdebian. Razvoj memorijskih uređaja i široka dostupnost memorije i na ugrađenim uređajima dovela je do prestanka potrebe za ovakvom distribucijom. Većina uređaja danas poseduje mogućnost proširivanja dostupne memorije, recimo putem memorijskih SD kartica sa više GB memorijskog prostora.

4.3 Problemi u procesu prevođenja paketa

U procesu prevođenju paketa za arhitekture MIPS32 i MIPS64 susreli smo se sa nekoliko tipičnih problema koje ćemo ovde izdvojiti:

- Paket nema podršku za željenu arhitekturu.
- Paket je preveden ali je rezultat prevođenja nekorektan.
- Paketi koji čekaju na prevođenje su međusobno zavisni.
- Problemi tipični za big-endian sisteme.
- Problemi sa neporavnatim pristupom memoriji
- Neprekidne operacije nad 64-bitnim vrednostima na arhitekturi MIPS32.

Nedostatak podrške za željenu arhitekturu:

Može se ogledati u tome da u datotekama u kojima su pobrojane podržane arhitekture nije pobrojana i arhitektura za koju se želi prevesti paket. Stoga, da bi prevođenje standardnim alatima uopšte započelo potrebno je u listu podržanih arhitektura u odgovarajućim datotekama dodati i onu arhitekturu za koju se paket želi prevesti. Na primer kod distribucije Debian to su odgovarajuća control datoteka i ime_paketa.dsc datoteka.

Sam nedostatak podrške može se ogledati i u nepostajanju odgovarajućih programskih sekvenci specijalizovanih za ciljnu arhitekturu. Recimo neko ponašanje opisano na nivou mašinskog koda. Ovakvi problemi se razmatraju i rešavaju ponaosob, dodavanjem odgovarajućeg programskog koda, izuzimanjem dela paketa ili izuzimanjem kompletnog paketa iz daljeg prevođenja.

Nekorektno prevedeni paketi:

Često, paketi izvornog koda sadrže i testove. Ovi testovi će prilikom prevođenja biti izvršeni kako bi se ispitivanjem paketa dobijenog prevođenjem potvdila ispravnost dobijenog rezultata po unapred određenim kriterijumima. Ukoliko se ispitivanje uspešno izvrši paket će biti zapakovan u binarni paket, tj svoj izvršni oblik.

U samom postupku prevođenja paketa za novu arhitekturu, kako smo ranije objasnili, koriste se emulator i/ili među-platformski skup alata. Prevođenje paketa izvornog koda uz pomoć ovih alata unosi još jedan nivo nesigurnosti u sam proces prevođenja, tj. otkaz u procesu prevođenja može biti uslovljen kako određenom nekorektošću u samom paketu, tako i nekorektonošću u radu alata koji se koriste za prevođenje.

U praksi se dešava da se sam paket prevodi bez ikakve poruke o grešci a da u procesu ispitivanja paketa njegovim ugrađenim testovima bude prijavljena greška. U idealnom slučaju ove greške bi morale biti otklonjene pre nego što paket bude korišten dalje za izgradnju sistema.

Međutim, kada se želi podržati nova arhitektura, za koju nemamo binarne pakete, tj. nema ni prevedenih ranijih izdanja paketa, ponekad se moraju isključiti ugrađeni testovi kako bi se krenulo u dalje prevođenje paketa i došlo do minimalnog sistema koji bi se mogao pokrenuti

na ciljanoj arhitekturi, a zatim se na samom razvojnom sistemu ispitali mogući nedostaci. Razlozi za loše rezultate ispitivanja mogu biti:

- neispravnost paketa,
- neispravnost među-platformskog prevodioca,
- upotreba emulatora,
- neprilagođenost ispitivanja sa ciljanom arhitekturom, itd.

Ovakvi paketi se nakon što su jednom prevedeni bez ispitivanja kasnije ponovo prevode sistemom za automatsko prevođenje, pri čemu se sada vrši i ispitivanje ugrađenim test scenarijima u toku prevođenja paketa. Ukoliko bude otkriven otkaz, koji dovede dalje do otklanjanja odgovarajuće greške, potrebno je da paketi koji se u procesu prevođenja oslanjaju na paket kod koga je otklonjena greška budu ponovo prevedeni kako bi se isključila mogućnost da se neispravnost tog jednog paketa dalje odrazi na druge pakete.

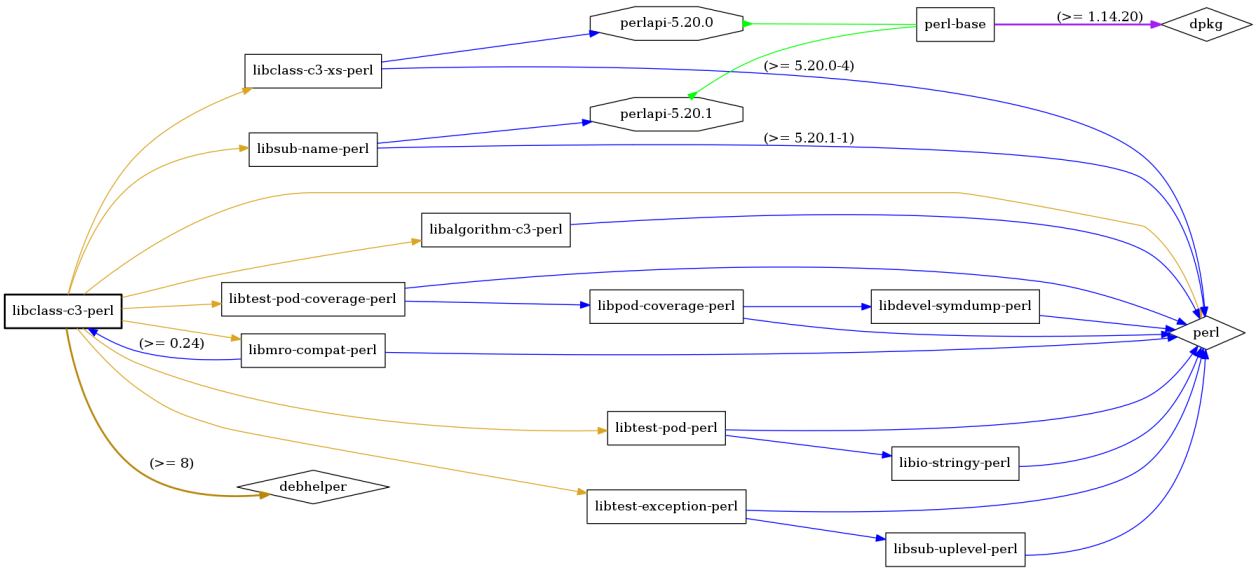
Međusobno zavisni paketi:

U procesu prevođenja pojavili su se paketi za čije su prevođenje bili neophodni neki drugi paketi koji bi opet direktno ili preko nekih trećih paketa zahtevali one pakete od kojih smo i krenuli u proces razrešavanja zavisnosti.

Daćemo najpre nekoliko primera kružnih zavisnosti a zatim ćemo obrazložiti načine na koje je ovo rešavano i zašto problem kružnih zavisnosti nije izražen na zvaničnom Debian sistemu za prevođenje paketa za podržane arhitekture.

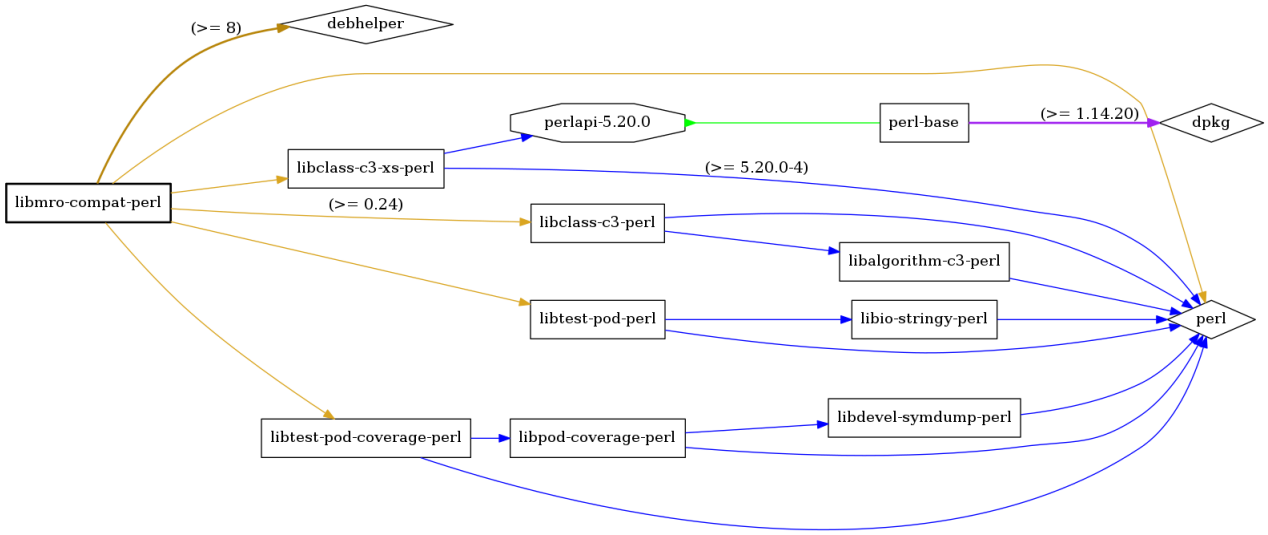
Kako bismo pojasnili ovaj problem pogledaćemo pakete libclass-c3-perl i libmro-compat-perl. Koristićemo grafike zavisnosti dobijene pomoću alata debtree[35].

Ukoliko potražimo zavisnosti paketa libclass-c3-perl (0.26-1) primetićemo da je on zavisan od paketa libmro-compat-perl a zatim i da paket libmro-compat-perl (0.12-1) zavisi od paketa libclass-c3-perl izdanja 0.24 ili novijeg (slika 4.6).



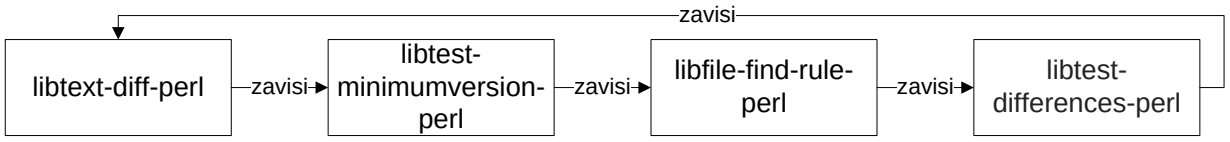
4.6 - zavisnosti paketa libclass-c3-perl

Sledeća slika 4.7 prikazuje zavisnosti paketa libmro-compat-perl (0.12-1). Na slici možemo videti da paket zavisi od paketa libclass-c3-perl izdanja 0.24 ili novijeg.



4.7 - zavisnosti paket libmro-compat-perl

Kružne zavisnosti ne moraju biti direktne kao u prethodnom primeru već krug kružne zavisnosti može biti duži i složeniji, jedan pojednostavljen primer je prikazan na sledećoj slici 4.8:



4.8 - kružna zavisnost paketa libtext-diff-perl

Na priloženoj slici se vidi da paket libtext-diff-perl(1.41-1) zavisi od paketa libtest-minimumversion-perl. Paket libtest-minimumversion-perl(0.101081-1) dalje zavisi od paketa libfile-find-rule-perl. Dalje paket libfind-rule-perl(0.33-1) zavisi od paketa libtest-differences-perl, a na kraju paket libtest-differences-perl(0.62-1) zavisi od paketa kojim je započeo krug zavisnosti, libtext-diff-perl.

OBS ove kružne zavisnosti nije mogao sam da razreši. Već su ručno razrešavane, tako što bi se jedan od paketa iz lanca kružno zavisnih paketa preveo i dodao na ulazni repozitorijum (skladište) OBS-a.

Rešenje prvog primera je prevođenje paketa libclass-c3-perl bez paketa libmro-compat-perl, zatim kopiranje prevedenog binarnog paketa na ulazno skladište OBS-a i zatim dalje prevođenje paketa na OBS-u, najpre paketa libmro-compat-perl pa onda paketa libclass-c3-perl. Treba podsetiti da se binarni paketi prevedeni na OBS-u, automatski kopitaju na ulazno skladište OBS-a.

Vrlo slično je i rešenje drugog primera. Prevođenje paketa libtext-diff-perl najpre je obavljeno bez paketa libtest-minimumversion-perl. Kopiran je prevedeni binarni paket na ulazno skladište OBS-a a zatim su redom prevođeni paketi, iz lanca kružne zavisnosti. Na kraju je ponovo preveden paket libtext-diff-perl.

Potrebno je naglasiti da nije uvek uspešno rešenje izabrati „prvi“ paket iz lanca već je potrebno pronaći odgovarajući paket u lancu koji se može prevesti bez neke svoje zavisnosti.

Da bismo razumeli razloge za javljanje problema kružnih zavisnosti objasnićemo razlike u prevođenju paketa distribucije Debian u radu sa OBS-om u odnosu na sistem za automatsko prevođenje paketa distribucije Debian i izložiti specifičnosti prevođenja paketa za arhitekturu za koju trenutno ne postoji podrška.

OBS prevodi sve pakete OS Debian naredbom `debian/rules build`.

Debianov sistem za prevođenje paketa većinu paketa prevodi naredbom `debian/rules build-arch`.

Ova razlika je proistekla iz toga što se paketi operativnog sistema Debian prevode za više arhitektura, pa su paketi po arhitekturi podeljeni prema ključnoj reči u debian/control i *.dsc datoteci. Nakon ključne reči „Architecture“, sledi lista konkretno podržanih arhitektura ili ključne reči „`any`“ ili „`all`“.

Ključna reč „`any`“ označava da paket može biti preveden za bilo koju (Debian) arhitekturu, tj. kod je napisan tako da je prenosiv i da se može uspešno prevesti na različitim arhitekturama. Binarni paket za odereñenu arhitekturu se dobija prevođenjem izvornog koda paketa na toj arhitekturi.

Ključna reč „`all`“ označava da će isti binarni paket raditi na svim (Debian) arihetkturama, a da ne mora biti preveden za svaku od njih posebno. Tj. dovoljno je paket prevesti na jednoj arhitekturi a zatim binarni paket koristiti na svim ostalim arhitekturama. Na primer „`all`“ paket bi bio onaj paket koji se sastoji samo od skripti, npr. shell scripti. Ove skripte rade svuda na isti način i nije ih potrebno prevoditi.

Ako pogledamo pakete koji učestvuju u kružnim zavisnostima u prethodna dva primera, primetićemo da paketi pripadaju „`all`“ grupi. Takođe treba primetiti da paket libclass-c3-perl (0.26-1) zahteva paket libmro-compat-perl (0.12-1) a zatim paket libmro-compat-perl zahteva paket libclass-c3-perl (0.24-1) ili noviji. Time je za arhitekture zvanično podržane distribucijom Debian znatno olakšano prevođenje ovih paketa, jer se za jednu arhitekturu mogu koristiti binarni paketi iz grupe „`all`“ prevedeni na drugoj arhitekturi, ili se za prevođenje paketa može koristiti paket iz prethodnog izdanja distribucije Debian.

Kako prilikom prevođenja paketa za dotad nepodržanu arhitekturu, npr. MIPS64, nemamo ni osnovni skup alata i paketa preveden i u dovoljnoj meri ispitan, prevodili smo sve pakete izvornog koda, uključujući tu i „`all`“ pakete. To znači da nismo preuzeli gotove pakete iz grupe „`all`“ sa zvaničnog skladišta binarnih paketa distribucije Debian. Preuzimanje ovih paketa bi ubrzalo proces prevođenja, ali ne postoji garancija da time ne bismo uneli još jedan stepen nesigurnosti u sam proces izgradnje sistema. Jednom uspostavljen sistem za dotad ne podržanu arhitekturu može se kasnije iskoristiti za znatno brže i jednostavnije prevođenje sledećeg izdanja željenog sistema.

U realnom radu, primećeni su slučajevi da paket iz grupe „`all`“ ne može da se prevede na arhitekturi MIPS. Primer su paketi x11proto-core i xorg-docs paketi izvornog koda prevođeni za izdanje wheezy distribucije Debian. Binarni paketi koji se dobijaju prevođenjem ovih paketa su x11proto-core-dev (7.0.23-1) i xorg-docs (1:1.6-1) i pripadaju grupi „`all`“. Prilikom prevođenja ovih paketa uočen je problem sa paketom fop, koji je takođe „`all`“ paket. A dalje ispitivanje je dovelo do pronalaženja uzroka otkaza u okviru openjdk-6 paketa.

Da su paketi x11proto-core-dev, xorg-docs i fop, bili samo preuzeti kao gotovi binarni paketi i dodati na ulazno skladište OBS-a, otkaz bi promakao neprimećen ili bi eventualno bio primećen u krugu ponovnog prevođenja paketa, ukoliko i u okviru ponovnog prevođenja ne bi odlučili da ne prevodimo grupu paketa „`all`“.

Problemi u prevođenju paketa za sisteme sa big-endian redosledom upisa bajtova:

Arhitekture MIPS32 i MIPS64 imaju svoje izvedbe big-endian i little-endian[36]. Prilikom prevođenja paketa za arhitekture sa izvedbom big-endian uočeni su problemi koji se nisu javljali prilikom prevođenja paketa za izvedbe little-endian. Problem se javljao prilikom prevođenja paketa koji sadrže kod u kome dolazi do pretvaranja pokazivača na jedan tip podataka u pokazivač na drugi tip podataka.

Izraz endian se odnosi na konvenciju kojom se opisuje način na koji se interpretiraju bajtovi u okviru memorijske reči, kada se ti bajtovi nalaze u memoriji računara. Svaki bajt u memoriji računara se nalazi na odreñenoj adresi.

	big-endian			
Adresa	1000	1001	1002	1003
Zapis	12	34	56	78

	little-endian			
Adresa	1000	1001	1002	1003
Zapis	78	56	34	12

4.9 - zapis broja u memoriji u odnosu na endian

Pokazivač na određeni tip, pokazuje na određenu memorijsku lokaciju i obuhvata, u odnosu na tip na koji pokazuje, određeni broj bajtova počev od memorijske lokacije na koju pokazuje. Tako će pokazivač tipa int pokazivati na 4 bajta počev od zadate memorijske lokacije. Ako uznemo za primer broj 0x12345678 pokazivač će na big-endian-u pokazivati na bajt sa vrednošću 0x12 a na little-endian-u na bajt sa vrednošću 0x78 (slika 4.9).

Sledeći primer koda će nam poslužiti da objasnimo problem koji nastaje prilikom pretvaranja pokazivača:

```
unsigned int x = 0x12345678;
unsigned short y;
unsigned char z;

y = *((unsigned short*)&x);
z = *((unsigned char*)&x);

printf("y = %x\n",y);
printf("z = %x\n",z);
```

Rezultat izvršavanja ovog dela koda će biti različit na različitim endian-ima. Na big-endian-u ćemo imati sledeći ispis:

```
y = 0x1234
z = 0x12
```

Dok će na Little-endian-u biti ispisano sledeće:

```
y = 0x5678
z = 0x78
```

Samo pisanje koda sa pretvaranjem pokazivača nije nezavisno od endian-a i daće različite rezultate. Takav kod nije u saglasnosti sa standardom ANSI C. Primeri ovakvog koda su relativno česti pa je ovo jedan od problema sa kojim smo često dolazili u susret.

Jedno od jednostavnih rešenja je korišćenje le16toh, le32toh, le64toh, htole16, htole32, htole64, be16toh, be32toh, be64toh, htobe16, htobe32, htobe64 funkcija koje konvertuju vrednost između vrednosti endian-a mašine (host) na kojoj se vrši izvršavanje koda i little/big-endian-a. Ukoliko se recimo primeni funkcija leXXtoh na mašini koja koristi little-endian neće se desiti nikakva promena, dok će ista funkcija na big-endian mašini odraditi zamenu endian-a tj. redosleda bajtova.

Ukoliko bismo želeli da za prethodni primer 0x12345678 dobijemo na big-endian-u identične vrednosti kao i na Little-endian-u, bilo bi potrebno da redosled bajtova u x pre pretvaranja pokazivača okrenemo sa x=htole32(x) ali da i posle pretvaranja pokazivača okrenemo redosled bajtova u y sa y=le16toh(y) i zatim vratimo x na originalnu vrednost sa x=le32toh(x).

Ukoliko y ne bismo okrenuli nakon pretvaranja pokazivača vrednost bi bila y=0x7856, dok promenljivu z nema potrebe okretati dodatno jer je dužine jedan bajt. Tako bi kod koji bi dao identične rezultate na oba endian-a za prethodni primer sa pretvaranjem pokazivača izgledao ovako:

```
unsigned int x = 0x12345678;
unsigned short y;
unsigned char z;
```

```
x = htole32(x);
y = *((unsigned short*)&x);
y = le16toh(y);
z = *((unsigned char*)&x);
x = le32toh(x);

printf("y = %x\n",y);
printf("z = %x\n",z);
```

Krajnji rezultat na oba endian-a će biti identičan:

```
y = 0x5678
z = 0x78
```

Problem se javlja i u radu sa unijom. Unija je specijalni tip podataka u programskom jeziku C koji dozvoljava da se različiti tipovi podataka smeštaju na istu lokaciju u memoriji. Prilikom deklarisanja unije može se navesti proizvoljan broj članova različitog tipa, ali će u istom vremenskom momentu samo jedan član unije imati dodeljenu vrednost. Dodeljena vrednost nekom članu unije se smešta na zadatu memorijsku lokaciju, a ukoliko probamo da upišemo drugi element doći će do prepisivanja te iste memorijske lokacije.

Sledeća dva primera će pokazati ponašanje unije na big i little-endian-u.

Primer 1:

```
typedef union unija
{
      short a;
      char b[2];
}Unija;

int main()  {
      Unija u;
      u.a = 1;
      printf("a = %d \n",u.a);
      printf("b = %d,%d\n",u.b[0],u.b[1]);

}
```

Izvršavanje koda će na big-endianu dati rezultat:

```
a = 1
b = 0,1
```

a na little-endianu:

```
a = 1
b = 1,0
```

Primer 2:

```
typedef union unija
{
      int a;
      char b[2];
      short c;
}Unija;

int main()  {
      Unija u;
      u.a = 1;
      printf("a = %d\n",u.a);
      printf("b = %d,%d\n",u.b[0],u.b[1]);
      printf("c = %d\n",u.c);
      return 0;
}
```

Izvršavanje prethodnog koda će na big-endianu dati rezultat:

```
a = 1
b = 0,0
c = 0
```

a na little-endianu:

```
a = 1
b = 1,0
c = 1
```

I ovo ponašanje je posledica različitog redosleda upisa bajtova na određenu memorijsku lokaciju na koju se smešta unija. Kao što smo već pomenuli, najmanje značajan bajt se kod sistema little-endian smešta prvi pa će kada se ta memorijska lokacija pročita kao podatak drugog tipa, dobiti prvo manje značajni bajtovi, dok ćemo kod big-endian sistema dobiti prvo najznačajnije bajtove. Ovoga treba biti svestan prilikom pisanja koda kako bi napisani kod bio nezavisan od endian-a. Ukoliko je ipak potrebno upisati jedan tip u uniju a pročitati drugi, potrebno je koristiti ranije spomenute leXXtoh i htoleXX funkcije, kako bi rezultat na oba endian-a bio identičan.

Problem je uočen i kod paketa koji su koristili gotove binarne datoteke isporučene u okviru paketa izvornog koda a izgrađene na little-endian sistemu i njemu i namenjene.

Ili kod paketa koji pišu u određenu vrstu datoteka koja će kasnije biti korištena za čitanje, a ne vode računa o tome da programski kod kojim se vrši čitanje i pisanje podataka bude nezavisan od endian-a, ili da je sam zapis nezavisan od endian uređenja mašine na kojoj se programski kod izvršava, kako bi ista datoteka mogla da se koristi i na sistemu sa drugim endian uređenjem. Rešenje za ovakve pakete je da se podaci pre upisa u datoteku prebace u neki od endian-a, i da se taj endian proglasi standardom za upis u datoteku, a da se prilikom čitanja datoteke zapisi prebace u prirodni endian fizičke arhitekture na kojoj se program izvršava.

Problemi uočeni na big-endian sistemima su često uslovljeni pogrešnom pretpostavkom da je redosled bajtova u reči uređen uvek kao kod little-endian sistema i da se delu nekog podatka može bezbedno pristupiti preko određene memorijske lokacije.

Neporavnat pristup memoriji:

Jedan od čestih problema na arhitekturi MIPS32 je neporavnat pristup double vrednosti u memoriji. Ovo se najčešće događa kada se u kodu pojavi pretvaranje pokazivača na proizvoljni tip u pokazivač na double vrednost. Kako bi razumeli problem najpre ćemo pogledati standardne tipove i koliko mesta u memoriji oni zauzimaju na arhitekturi MIPS.

Veličina osnovnih tipova:

Tip programskog jezika C	Ime u mašinskom kodu	Veličina tipa u bajtima
Char	byte	1
Short	half (half word)	2
Int	word	4
long long	dword (double word)	8
Float	word	4
Double	dword	8

4-1 - veličina osnovnih tipova, arhitektura MIPS

Kada pogledamo pobrojane tipove primetićemo da nedostaje tip long i tip pokazivača. Izostavljeni su sa razlogom. Veličina tipa long se razlikuje na arhitekturama MIPS32 i MIPS64. A tip pokazivača je na arhitekturi MIPS implementiran kao unsigned long. Tako da će i veličina tipa pokazivača u memoriji biti različita na MIPS32 i MIPS64 sistemima.

Tip programskog jezika C	Oznaka u mašinskom kodu	Veličina u bajtima
MIPS32		
Long	word	4
tip* (tip pokazivača)	word	4
MIPS64		
Long	dword	8
tip* (tip pokazivača)	dword	8

4-2 - veličina tipova long tipa i tipa pokazivača, arhitektura MIPS

Svi ovi tipovi se mogu koristiti u kombinaciji sa standardnim instrukcijama arhitekture MIPS ako su poravnati na odgovarajuću adresu tako da adresa na kojoj se podatak nalazi podeljena sa brojem bajtova koju podatak odgovarajućeg tipa zauzima daje ostatak nula. Na primer za podatak tipa int, adresa mora biti deljiva sa 4 bez ostatka.

Prevodioci će najčešće, ako im nije drugačije naloženo, posložiti podatke tako da je poravnanje prirodno, tj. da su adrese promenljivih u memoriji deljive bez ostatka sa brojem bajtova koji zauzima tip. Ovo će se postići na taj način što će prazni bajtovi biti ostavljeni između dve uzastopna podatka, ukoliko je potrebno, kako bi se očuvalo prirodno poravnanje podataka.

Ukoliko se ukaže potreba, na arhitekturi MIPS se mogu koristiti i neporavnati podaci u memoriji. Rad sa ovakvim podacima je sporiji. Radi tako što se umesto jedne koriste dve instrukcije za čitanje ili pisanje, kako bi se omogućio rad sa neporavnatim adresama u memoriji, za šta postoje specijalne instrukcije.

Izuzetak je rad sa float i double tipovima, tj. tipovi sa kojima radi jedinica za rad sa vrednostima u pokretnom zarezu (FPU) i ovde se mora poštovati prirodno poravnanje. Instrukcije za rad sa FP vrednostima arhitekture MIPS:

Instrukcija	Zapis	Rezultat	Alternativna instrukcija
l.d–load FP double	l.d fd, addr	fd=(double)*addr	ldc1–load FP double to FP register
s.d–FP store double	s.d fs, addr	(double)*addr=fs	sdc1–store FP double register to memory
l.s–load FP single	l.s fd, addr	fd=(float)*addr	lwc1–load FP single to FP register
s.s–FP store single	s.s fs, addr	(float)*addr=fs	swc1–store FP single register to memory

4-3 - FP čitaj piši instrukcije, arhitektura MIPS

Instrukcije za čitanje i pisanje vrednosti u pokretnom zarezu (FP) zahtevaju da adrese sa kojih se čita ili u koje se upisuje budu prirodno poravnate. Tj. da budu deljive sa 4 bez ostatka za float vrednosti i deljive sa 8 bez ostatka za double vrednosti.

Kako smo ranije izneli veličina pokazivača je jednaka veličini tipa unsigned long. Veličina pokazivača na arhitekturi MIPS32 biće 4B, jer je long veličine 4B na ovoj arhitekturi, pa će pokazivač biti prirodno poravnat na adresu deljivu sa 4. S druge strane double vrednosti se prirodno poravnavaju na adrese koje su deljive sa 8. Sledi da će prilikom pretvaranja proizvoljnog pokazivača u pokazivač na double vrednost postojati 50% šanse da dođe do neporavnatog pristupa memoriji.

Ovo je najčešći razlog za neporavnat pristup memoriji i otkaz programa usled njega. Može se rešiti na više načina, od zamene jedne double instrukcije sa dve single instrukcije, do ispravljanja pogrešne pretpostavke u kodu – što je najbolje rešenje.

Ovaj problem je toliko čest da se predlagalo uvođenje specijalne opcije za prevodilac, kojom bi se prevodiocu sugerisalo, da u procesu prevođenja umesto jedne instrukcije l.d ili s.d emituje dve instrukcije l.s ili s.s.

Međutim ovo nije jedini izvor neporavnatog pristupa memoriji kod arhitekture MIPS. Do neporavnatog pristupa može se doći i ako se prevodiocu da zahtev da podatke spakuje tj. da ne umeće prazne bajtove kojima se održava prirodno poravnanje podataka a da se pritom među podacima nalaze i podaci FP tipa. Isti rezultat ćemo dobiti i ako prevodiocu naložimo da FP podatke poravnava na vrednost koja je manja od one koja odgovara prirodnom poravnanju za ove podatke (4 ili 8 bajta). Takođe korišćenje nekog zasebno razvijenog mehanizma za zauzimanje memorije, koji neće voditi računa o poravnanjima može dovesti do otkaza. Uzrok svih ovih otkaza je nedovoljna saglasnost koda sa specijalnim zahtevima različitih arhitektura na kojima se kod izvšava, u ovom slučaju arhitekturom MIPS.

Neprekidne operacije:

Specifičnost arhitekture MIPS32 je da nema podršku za neprekidne, nedeljive (atomske) 64-bitne operacije[36] te su se u radu, kada određeni paket zahteva neprekidne 64-bitne operacije na arhitekturi MIPS32, javljali problemi u prevođenju paketa.

Da bi ovo pojasnili najpre ćemo objasniti mehanizam kojim se kod arhitekture MIPS obezbeđuje neprekidnost operacija a zatim izložiti ugrađene funkcije prevodioca za neprekidni pristup memoriji.

Arhitektura MIPS poseduje par instrukcija load-linked, store-conditional (dalje u tekstu LL i SC). Koriste se da bi se implementirala proizvoljna read-modify-write (čitaj-promeni-upiši, dalje RMW) sekvenca nad promenljivom. Čitanje se vrši instrukcijom LL a pisanje odgovarajućom instrukcijiom SC. Princip rada je takav da sama sekvenca po sebi nije neprekidna, ali instrukcija SC neće odraditi upis ukoliko ne postoji sigurnost da je izvršavanje sekcije bilo neprekidno. U zavisnosti od toga da li je blok prilikom izvršen bez prekida ili ne instrukcija SC će vratiti vrednost koja se može programski uporediti i na osnovu nje ponoviti RMW sekcija ako je prethodna bila neuspešna.

Ove instrukcije su osmišljene na ovaj način zbog prednosti koje pokazuju u radu na sistemima sa više procesorskih jezgara, kao alternativa za garantovane nedeljive RMW operacije. Prednost im je što ne moraju da zaustave kompletan pristup memoriji, ili njenom određenom bloku kako bi garantovale neprekidnost operacije i obezbedile da niko ne pristupi podacima od interesa sa kompletnog sistema.

Sama izvedba je prilično jednostavna i ogleda se u tome da instrukcija LL postavlja odgovarajući „link bit“ i beleži se adresa učitavanja, kako bi CPU mogao da prati pristup toj adresi, a instrukcija SC će uspeti (izvršiti upis) i vratiti vrednost 1 samo ako je „link bit“ i dalje nepromenjen.

Primer funkcije `atomic_inc(&var)` koja uvećava vrednost promenljive var:

```
atomic_inc:
ll v0, 0(a0)       # registar a0 sadrži pokazivač na var
addu v0, 1
sc v0, 0(a0)
beq v0, zero, atomic_inc
 nop         # delay slot
jr ra
 nop         # delay slot
```

Po izlazu iz ove funkcije vrednost promenljive var je uvećana za 1. Moguće je da će se petlja izvršiti nekoliko puta dok se ne postigne neprekidno izvršavanje, ali pošto je u petlji mala sekvenca instrukcija, petlja se najverovatnije neće izvršiti više od tri puta.

Na osnovu ovoga može se primetiti da LL i SC par instrukcija nije namenjen za veoma komplikovane operacije gde su instrukcije LL i SC veoma udaljene jedna od druge.

Postoje dva razloga zbog kojih izvršavanje instrukcije SC može biti neuspešno. Prvi je da se desio neki izuzetak između izvršavanja instrukcija LL i SC. Obrađivač izuzetka ili proces okinut izuzetkom su mogli da ugroze neprekidnost operacije.

Drugi razlog za neuspelo izvšavanje instrukcije SC javlja se samo kod izvedbi sa više procesorskih jezgara. Dešava se kada drugi CPU upiše u memorijsku lokaciju za koju treba garantovati neprekidnost ili upiše u lokaciju koja je dovoljno blizu ovoj lokaciji. Šta znači dovoljno blizu zavisi od konkretne implementacije načina na koji se na konkretnoj platformi vodi računa o memoriji. Zato neuspeh izvršavanja SC ne garantuje da je sekciji od interesa ugrožena neprekidnost operacije, već samo postoji određena sumnja da bi mogla biti ugrožena.

Konkretni nazivi instrukcija na arhitekturi MIPS su ll, sc, lld, scd.

Instrukcije su prikazane su u sledećoj tabeli 4-4:

Instrukcija	Zapis	Rezultat
ll–load linked	ll t, addr	Load linked 32 bits
sc-store conditional	sc t, addr	Store word condtional
lld–load linked double	lld t, addr	Load linked 64 bits
scd–store conditional double	scd t, addr	Store double conditional

4-4 - Instrukcije koje se koriste za neprekidne operacije, arhitektura MIPS

Na arhitekturi MIPS32 su implementirane samo instrukcije ll i sc. Aritektura MIPS64 ima implementirane sve četri mašinske instrukcije ll, sc, lld i scd.

Ugrađene funkcije prevodioca za neprekidni pristup memoriji:

U izdanju squeeze distribucije Debian podržana verzija prevodioca gcc je 4.4, dok je u izdanju wheeze bila verzija 4.6 da bi na kraju bila uključena verzija 4.7

Prevodilac je do verzije 4.7[37] (prema dokumentaciji) podržavao samo nedeljive operacije koje započinju rečju __sync, navešćemo neke od njih. Ideja je da se na bilo kojoj arhitekturi pozivanjem nedeljive operacije na istovetan način obezbedi identičan rezultat, a sam postupak obezbeđivanja neprekidnosti je implementiran ispod, sakriven od samog korisnika. Za arhitekturu MIPS se to postiže kao što smo ranije videli korišćenjem load-linked i store-conditional instrukcija.

Navešćemo neke od podržanih operacija:

```
type __sync_fetch_and_add (type *ptr, type value, ...)
type __sync_fetch_and_sub (type *ptr, type value, ...)
type __sync_fetch_and_or (type *ptr, type value, ...)
type __sync_fetch_and_and (type *ptr, type value, ...)
type __sync_fetch_and_xor (type *ptr, type value, ...)
type   sync fetch and nand (type *ptr, type value, ...)
```

Operacije se ponašaju u skladu sa njihovim nazivom i vraćaju vrednost koja je prethodno bila u memoriji:

```
{ tmp = *ptr; *ptr op= value; return tmp; }
{ tmp = *ptr; *ptr = ~(tmp & value); return tmp; } // nand

type __sync_add_and_fetch (type *ptr, type value, ...)
type __sync_sub_and_fetch (type *ptr, type value, ...)
type __sync_or_and_fetch (type *ptr, type value, ...)
type __sync_and_and_fetch (type *ptr, type value, ...)
type __sync_xor_and_fetch (type *ptr, type value, ...)
type __sync_nand_and_fetch (type *ptr, type value, ...)
```

Operacije se ponašaju u skladu sa njihovim nazivom i vraćaju vrednost koja je prethodno bila u memoriji:

```
{ *ptr op= value; return *ptr; }
{ *ptr = ~(*ptr & value); return *ptr; } // nand
```

Nisu sve operacije podržane od svih arhitektura. Ako neka operacija ne može biti podržana na ciljnom procesoru generiše se upozorenje i poziv na spoljnu funkciju. Spoljna

funkcija će imati isto ime kao ugrađena funkcija sa dodatnim sufiksom „_n“ gde će n predstavljati broj bajtova. Broj bajtova može biti 1, 2, 4 ili 8.

Za arhitekturu MIPS32 broj koji će se pojaviti kao sufiks nepodržanih ugrađenih funkcija je 8 i nepodržane su sve ugrađene funkcije sa sufiksom _8. Ovom problemu se do pojave prevodioca gcc-4.8 moglo pristupiti na jedan od sledećih načina u zavisnosti od važnosti paketa na kome je problem uočen:

- Izbacivanjem paketa iz grupe paketa koje prevodimo za ciljanu MIPS32 arhitekturu.
- Isključivanjem podrške za 64-bitne nedeljive operacije u određenom paketu za arhitekturu MIPS32, u paketu ostaje podrška za 32-bitne nedeljive operacije.
- Pisanjem sekcije programskog koda koja će obezbediti željenu neprekidnost. Ovo se može postići korišćenjem nekog programskog mehanizma zaključavanja koji će obezbediti neprekidno izvršavanje koda (mutex, __sync_lock_test_ant_set(type *ptr, type value, ...) – __sync_lock_release(type *ptr, ...), itd.)

Sa pojavom prevodioca koji ugrađene nedeljive operacije prilagođava C++11 standardu memorijskog modela pojavljuju se nove ugrađene funkcije[38] koje imaju prefix __atomic i svojim nazivom i funkcionalnošću podsećeju na one sa prefiksom __sync ali za razliku od njih primaju još dodatni parametar koji se odnosi na memorijski model. Ovo se prvi put spominje u prevodiocu gcc 4.7 a u funkciji je u prevodiocu gcc 4.8

Zamene za prethodno navedene __sync ugrađene funkcije su:

```
type __atomic_add_fetch (type *ptr, type val, int memmodel)
type __atomic_sub_fetch (type *ptr, type val, int memmodel)
type __atomic_and_fetch (type *ptr, type val, int memmodel)
type __atomic_xor_fetch (type *ptr, type val, int memmodel)
type __atomic_or_fetch (type *ptr, type val, int memmodel)
type   atomic nand_fetch (type *ptr, type val, int memmodel)
```

Rezultat operacija je:

```
{ *ptr op= val; return *ptr; }
```

Svi memorijski modeli su dozvoljeni.

```
type __atomic_fetch_add (type *ptr, type val, int memmodel)
type __atomic_fetch_sub (type *ptr, type val, int memmodel)
type __atomic_fetch_and (type *ptr, type val, int memmodel)
type __atomic_fetch_xor (type *ptr, type val, int memmodel)
type __atomic_fetch_or (type *ptr, type val, int memmodel)
type   atomic fetch_nand (type *ptr, type val, int memmodel)
```

Rezulat operacija je:

```
{ tmp = *ptr; *ptr op= val; return tmp; }
```

Svi memorijski modeli su dozvoljeni

Mogući memorijski modeli:

- __ATOMIC_RELAXED
- __ATOMIC_CONSUME
- __ATOMIC_ACQUIRE
- __ATOMIC_RELEASE
- __ATOMIC_ACQ_REL
- ATOMIC SEQ CST

Razlikuju se prema nivou memorijske barijere i nivou sinhronizacije. Prvi je najmanje stritktan dok je poslednji najzahtevniji po ovom pitanju. Ukoliko se koristi poslednji memorijski model __ATOMIC_SEQ_CST __atomic ugrađene funkcije imaju istu funkcionalnost kao __sync ugrađene funkcije.

Operacije sa prefiksom __atomic u odnosu na operacije sa prefiksom __sync se razlikuju po tome što su ciljne arhitekture dobrodošle da obezbede svoju implementaciju za bilo koju od ugrađenih funkcija. Ako ne postoji implementacija za ciljnu arhitekturu, originalne __sync operacije će biti pozvane sa svim potrebnim preduslovima (sinhronizaciona ograničenja) da se dobije ponašanje koje odgovara ponašanju pozvane __atomic operacije. Ako nema pravila ili

mehanizma da se obezbedi instrukcijska sekvenca bez zaključavanja (nema ni odgovarajuće __sync operacije) biće napravljen poziv ka spoljnoj biblioteci sa parametrima koji će biti razrešeni za vreme izvšavanja. Spoljna biblioteka kojoj se pozivi upućuju se zove libatomic. Ideja ove biblioteke je da se obezbedi standardna implementacija nedeljive operacija kako bi različiti prevodioci mogli neometano da komuniciraju.

Neki od razloga za korišćenje ove biblioteke su razrešavanje sledećih problema:

- Arhitektura nema implementiran compare_and_swap ili drugačije ne podržava nedeljive operacije (Većina jediničnih operacija se može izgraditi pomoću compare_and_swap, ali je potrebna barem ta operacija).
- Veličina objekta ne može da se mapira na podržane veličine nedeljivih operacija. Problem 64 bitne operacije na arhitekturi MIPS32.
- Potrebno je da program komunicira sa drugim arhitekturama koje nemaju podršku za nedeljive operacije.

Kada se koristi prevodilac verzije 4.8 ili noviji, moguće je za arhitekturu MIPS32 koristiti __atomic umesto nedostajućih __sync ugrađenih funkcija. Ukoliko ne postoji implementacija __atomic funkcija za ciljanu arhitekturu biće pozvane odgovarajuće __sync, a tamo gde one nedostaju biće korištene funkcije spoljne biblioteke.

U momentu kada smo prevodili pakete za izdanja „squeeze“ i „wheeze“ distribucije Debian ova funkcionalnost nije bila dostupna. U kasnijem procesu pružanja podrške novijim izdanjima distribucije Debian korištena je mogućnost zamene __sync operacija ugrađenim __atomic operacijama. Pored zamena operacija odgovarajućim __atomic operacijama potrebno je uključiti i biblioteku libatomic kako bi paket bio uspešno preveden.

4.4 Pakovanje distribucija

Nakon što dobijemo prevedene binarne pakete za određenu distribuciju, potrebno ih je organizovati u sistem koji se može koristiti kao operativni sistem na platformama zasnovanim na arhitekturi MIPS.

Kako je u procesu prevođenja paketa korišten OBS, kako za prevođenje RPM, tako i za prevođenje DEB paketa, potrebno je prevedene binarne pakete organizovati tako da možemo da upotrebimo odgovarajuće programsko rešenje za izgradnju korenskog sistema datoteka

Paketi prevedeni uz pomoć programskog rešenja OBS organizovani su u okviru izlaznog repozitorijuma OBS-a u skladu sa uređenjem skladišta namenjenog radu sa RPM paketima (OpenSuse, Fedora).

Distirbucija MeeGo koristi pakete sa nastavkom ".rpm" a alat mic-image-creator namenjen je izgradnji sistema od paketa iz skladišta RPM paketa, pa je pakovanje distribucije MeeGo odrađeno uz pomoć alata mic-image-creator od izlaznog repozitorijuma OBS-a na računaru sa procesorskim jezgrom Intel.

Distribucija Debian koristi pakete sa nastavkom ".deb". Izlazni repozitorijum iz OBS-a je po organizaciji RPM prilagođen i ne odgovara ulaznom repozitorijumu debootstrap-a pa je potrebno rezultujući repozitorijum OBS-a preurediti. Za ove potrebe koristili smo alat reprepro.

Dodatna pogodnost je proizašla iz načina funkicionisanja OBS-a. OBS najpre binarne pakete dobijene preveođenjem smešta u direktorijum namenjen za to, kao među korak pre prebacivanja paketa na izlazno skladište. Zatim binarne pakete iz tog direktorijuma dalje smešta u odgovarajuće izlazno skladište na taj način da skladište bude uređeno u skladu sa standardom za rad sa RPM paketima. Prevedeni paketi ostaju u direktorijumu i nakon njihovog smeštanja na izlazno skladište OBS-a.

Alat reprepro olakšava proces izrade repozitorijuma i izradu posebnih datoteka u saglasnosti sa apt programskim rešenjem, kao i dodavanje i uklanjanje paketa u/iz skladišta. Uz pomoć alata reprepro i korišćenjem direktorijuma u koji OBS smešta prevedene pakete pre nego ih prebaci na izlazno skladište izgrađeno je skladište paketa DEB koje je zatim korišćeno kao ulaz alata debootstrap u procesu izgradnje korenskog sistema datoteka distribucije Debian.

Iz dobijenog skladišta paketa DEB se može korišćenjem alata debootstrap izgraditi sistem Debian. Komanda ima sledeci oblik:

```
$ /usr/sbin/debootstrap --exclude="paketX, paketY, ..."
--include="paketA, paketB, paketC, paketD, ..." --arch arhitektura
izdanje_distribucije_Debian ciljna_putanja lokacija_ulaznog_skladišta
```

U prethodnoj komandi se vidi lista paketa koje ne treba uključiti u sistem (paketX, paketY), zatim lista paketa koje treba koristiti prilikom izgradnje sistema (paketA, paketB, paketC, paketD). Potom se navodi kojoj je arhitekturi namenjen sistem koji se gradi i koje se izdanje distribucije Debian izgrađuje. Sledi putanja na koju treba smestiti izgrađeni sistem. Na samom kraju naredbe se nalazi putanja do skladišta DEB paketa, organizovanog u skladu sa standardom alata apt, sa koga se vrši povlačenje paketa koji će biti instalirani na sistem.

Konkretan primer komande je dat u dodatku u okviru poglavlja 7.3.

Treba napomenuti da alat debootstrap ne može da koristi više skladišta paketa istovremeno, pa je neophodno sve pakete potrebne za izgradnju sistema smestiti u jedno skladište. Postoje i alati koji dozvoljavaju izgradnju sistema iz više različitih skladišta, npr. multistrap.

5. Ispitivanje distribucija

Kako bi rezultati koje dobijamo bili uporedivi razradili smo postupak koji je obuhvatao ispitivanje i čuvanje rezultata dobijenih ispitivanjem. Ovakav postupak omogućio nam je da pored provere samog sistema pre isporuke, možemo da u budućnosti prilikom ispitivanja drugog korenskog sistema datoteka te iste podatke uporedimo i identifikujemo eventualne nedostatke ili prednosti novog korenskog sistema datoteka u odnosu na ranije isporučene sisteme. Time smo dobili način da izmerimo kvalitet rešenja koje se spremamo da isporučimo. Da bi obezbedili što viši kvalitet rešenja koje se razvija, za poređenje je korišten sistem „timesys“ koji je dolazio uz razvojne ploče arhitekture MIPS namenjen za arhitekturu MIPS32 ali sa siromašnijim instrukcijskim skupom „mips2“. Po potrebi sistem koji je razvijan poređen je i sa zvaničnim sistemom Debian za arhitekturu MIPS32 instrukcijski skup mips2. Razvijani sistem morao je da pokaže iste ili bolje rezultate kako bi odlučili da je rad na istom završen.

Rezultate dobijene ispitivanjem možemo zatim iskoristiti da unapredimo i ranije razvijene sisteme u skladu sa novim saznanjima.

Treba napomenuti i da većina paketa namenjenih distribuciji Debian u svom sastavu ima ispitne scenarije namenjene proveri validnosti paketa u toku prevođenja. Ukoliko se detektuje problem na ovom nivou, on je razrešavan u procesu prevođenja paketa. Upoređivani su rezultati ovih ispitivanja sa rezultatima dobijenim na zvaničnom sistemu za prevođenje paketa distribucije Debian. Samo ispitivanje po procesu izgradnje korenskog sistema datoteka je višeg nivoa i nije namenjeno pojedinačnoj proveri paketa, paket po paket, uključenih u korenski sistem datoteka.

Međutim, ovo ne znači da se u ovoj fazi nećemo vraćati na nivo ispitivanja pojedinačnog paketa. Ukoliko nas dobijeni rezultati navedu na sumnju prema određenom paketu ili grupi paketa, iste ćemo ponovo razmotriti i ukoliko bude potrebe ponovo prevesti sa izmenama i zatim uvrstiti u sistem.

Ispitivanja na nivou paketa mogu se po potrebi i isključiti, ukoliko nam je neki paket preko potreban a u procesu prevođenja se dešavaju otkazi u fazi ispitivanja paketa. Razlog za to može biti sumnja na otkaz koji je uslovljen korišćenjem emulatora recimo. Kasnije ispitivanje može pokazati da je pretpostavka bila pogrešna i da se mora opet vratiti na ispitivanje tog pojedinačnog paketa.

Kako bi rezultati, koje smo dobijali ispitivanjem distibucija, bili uporedivi odlučeno je da se koristi sledeći način beleženja podataka o korenskom sistemu datoteka koji se ispituje:

Operativni sistem koji se ispituje
- Podaci o prevodiocu (gcc)
- Podaci o jezgru operativnog sistema
  - Podatak o nazivu i verziji jezgra operativnog sistem
  - Kompletna konfiguraciona datoteka jezgra operativnog sistema (kernel config file)
- Poznati problemi
  - Spisak poznatih problema i njihov opis
- Rezultati ispitivanja programskim rešenjem LTP
  - Prvi krug ispitivanja - datum ispitivanja
    - Podaci o procesu ispitivanja
    - Rezultati dobijeni ispitivanjem
      - Kompletni izveštaj ispitivanja
      - Spisak nesupelih ispitivanja
  - Drugi krug ispitivanja - datum ispitivanja
    - Podaci o procesu ispitivanja
    - Rezultati dobijeni ispitivanjem
      - Kompletni izveštaj ispitivanja
      - Spisak nesupelih ispitivanja
- Informacije o operativnom sistemu
- Informacije o platformi na kojoj je izvršeno ispitivanje sistema
- Sačuvani (boot-log) ispis informacija dobijen prilikom podizanja sistema u okviru konzole programskog rešenja Yamon

Metologija ispitivanja je formirana na osnovu iskustva stečenih kroz proces razvoja i ispitivanja dobijenih rešenja.

Podaci o prevodiocu beleženi su nakon što smo uočili da postoji mogućnost potkradanja greške, da prevodilac prevede paket za opštijeg pripadnika arhitektura MIPS, recimo mips32 (mips2) a da uvezivač (linker) uveže program koji je predmet prevođenja za više sprecijalizovanu arhitekturu MIPS, recimo mips32r2. Tako preveden i uvezan program, spolja posmatrano bi izgledao kao da je preveden za mips32r2 ali ne bi koristio sve prednosti ove arhitekture, tj. propust ne bi bio očigledan a performanse bi bile umanjene jer bi se prevedeni program bazirao na opštijem, siromašnijem instrukcijskom skupu. Kako bi se ova mogućnost predupredila, proveravili smo i beležili mogućnosti prevodioca.

Kako ćemo kasnije videti izbor izdanja jezgra operativnog sistema Linuks može da utiče na osobine i funkcionalnost sistema, pa je podatak o korišćenoj verziji beležen. Takođe je čuvana i konfiguraciona datoteka, kako bi se u slučaju potrebe moglo na istovetan (ili barem vrlo blizak) način konfigurisati jezgro operativnog sistema Linuks, kako za korišćenu tako i neku drugu ispitnu platformu. Radi uporedivosti rezultata, od značaja je korišćenje istog ili vrlo sličnog jezgra operativnog sistema Linuks.

Poznati problemi su beleženi kako bismo, ukoliko se kasnije susretnemo ponovo sa njima ili nađemo rešenje za njih, imali u vidu da su i ranije postojali i pod kojim uslovima. I kako bismo ih ako se ukaže mogućnost otklonili i u ranijim izdanjima.

Čuvani su i izveštaji ispitivanja sprovedenog programskim rešenjem LTP, uključujući izveštaje za oba sprovedena kruga ispitivanja. Metodologiju ispitivanja korščenjem programskog rešenja LTP ćemo kasnije detaljnije obrazložiti.

Beležene su informacije o izdanju operativnog sistema Debian koji je predmet ispitivanja.

Čuvane su informacije o izvedbi fizičke arhitekture na kojoj je izvršeno ispitivanje sistema. U zavisnosti od korišćene platforme mogu se pojaviti otkazi koji ne postoje na drugoj platformi. Ukoliko se ovakvi otkazi identifikuju, potrebno je ispitati njihov uzrok, te sa sigurnošću odrediti čime je otkaz izazvan: da li se radi o specifičnosti jezgra operativnog sistema Linuks namenjenog toj platformi ili je u pitanju otkaz uslovljen nekom drugom specifičnošću

platforme. Ispitivanjem uz pomoć LTP je recimo uočeno da prozivanjem memorijskog kontrolera na platformi Netlogic XLP dolazi do katastrofalnog otkaza sistema. Ovo se, recimo, događa ako se koristi naredba „find“ nad sistemskim direktorijumom proc ili njemu nadređenom diretkorijumu „ / “ (root). Time smo utvrdili uzrok povremenog otkaza ove platforme koja je korišćena od velikog broja korisnika istovremeno. U takvom okruženju prilikom otkaza nije bilo jednostavno utvrditi ko i pod kojim uslovima je izazvao otkaz. Korišćenjem LTP-a se već nakon dva ciklusa izvršavanja moglo na osnovu izveštaja izvšavanja ispitivanja utvrditi koja je grupa ispitnih scenarija i koji konkretno ispitni scenario poslednji izvršen pre otkaza. Na taj način lociran je uzrok nepredviđenih katastrofalnih otkaza sistema.

Sačuvani ispisi informacija dobijeni prilikom podizanja sistema u okviru konzole programskog rešenja Yamon čuvani su kako bi se, ukoliko se ukaže potreba, isti mogli pregledati i uporediti. Sami ispisi bi mogli da sugerišu neki nedostatak u samom sistemu ili u procesu podizanja sistema.

5.1 Postupak ispitivanja LTP - om

Ispitivanje korišćenjem projekta za ispitivanje Linuks, LTP, sprovođeno je korišćenjem sledećih ispitnih grupa:

admin-tools, cap-bounds, commands, dio, fcntl-locktests, filecaps, fs, hugetlb, hyperthreading, io, ipc, math, mm, nptl, numa, power_managment_tests, pty, sched, syscalls, timers.

Dinamika razvijanja rešenja bila je unapred utvrđena pa je ispitna metodologija morala da bude u skladu sa unapred definisanim zahtevima. Rešenje je realizovano u dve etape.

Najpre je realizovan sistem bez grafičkog okruženja zasnovanog na sistemu prozora X i sa manjim skupom prevedenih paketa. Korisnik je u okviru ovog korenskog sistema datoteka mogao da koristi za komunikaciju sa sistemom komandnu liniju tj. komandnu školjku sh ili napredniju komandnu školjku bash. Prilikom prevođenja paketa uložen je dodatni trud da u okviru prve isporuke budu dostupni prevodilac (gcc), povezivač (linker), alat za detekciju i otklanjanje grešaka iz programskih rešenja (debuger – gdb), kao i neki češće korišćeni alati, apt, tar, vim, wget, itd, kako bi već nakon prvog dostupnog sistema bilo moguće razvijati programska rešenja za željenu arhitekturu koristeći razvijeni korenski sistem datoteka i prateće alate.

U okviru druge etape pripreman je i zatim realizovan sistem sa proširenim skupom paketa uključujući i grafički sistem zasnovan na sistemu prozora X. U sam korenski sistem datoteka uključena su najčešće korišćena programska rešenja. Ostatak prevedenih paketa je bio dostupan u formi prevedenih paketa distribucije Debian (binarni deb paketi). Svi nezavisno isporučeni paketi mogli su biti dodati u korenski sistem paketa, tj. za svaki prevedeni paket distribucije Debian (binarni paket) dostupni su i svi njemu potrebni paketi (zavisnosti) kako bi isti mogao biti uključen u korenski sistem datoteka tj. instaliran na sistem.

Po završetku svake od ovih etapa, pre isporuke, vršeno je ispitivanje korenskog sistema datoteka. Rezultati su pribeleženi i istraženi i izvršeno je upoređivanje rezultata sa prethodnim isporukama. Sve dok ne bismo bili zadovoljni rezultatima ponavljali bismo sledeću sekvencu akcija:

- jezgro operativnog sistema Linuks i korenski sistem datoteka su rekonfigurisani i ponovo ispitani uz pomoć LTP-a.
- rezultati su pregledani, proučeni i upoređeni sa prethodnim

Kada smo zadovljni dobijenim rezultatima isti su pribeleženi uz napomene o problemima i rešenjima sa kojima smo se susreli.

Rezultati dobijeni ispitivanjem, koje se obavlja nakon druge etape prevođenja, upoređuju se sa razultatima dobijenim ispitivanjem ranije razvijenih rešenja. Posebno se vodi računa da

sistem koji se isporučuje nakon druge etape ne uvodi nove otkaze i neželjena ponašanja u odnosu na sistem koji je dobijen nakon prve etape.

Prilikom istraživanja uzroka otkaza od velike pomoći je bio projekat Yocto. Kako smo ranije napomenuli za ispitivanje i osiguranje kvaliteta (QA) projekta Yocto kortisti se LTP[39]. Rezultati ispitivanja su sortirani u skladu sa platformama koje su ispitavane izvršavanjem LTP ispitnih scenarija. Otkazi su pribeleženi i dokumentovani sa datim uzrokom otkaza tamo gde je uzrok otkaza poznat. Rezultati ispitivanja su dati tabelarno za svaku platformu na kojoj je vršeno ispitivanje i svako izdanje koje je ispitivano. Stoga su nam u cilju poređenja i razumevanja uzroka otkaza ispitivane distribucije Debian operativnog sistema Linuks za fizičku arhitekturu MIPS bili dostupni rezultati ispitivanja sa primerka fizičke izvedbe bazirane na MIPS arhitekturi – koju projekat Yocto koristi kao referentnu platformu - routerstation i rezultati ipitivanja dobijeni korśćenjem emulatora QEMU.

Sledeća tabela 5-1 prikazuje uporedno ispitne scenarije koji se koriste za ispitivanje trenutno dostupnog izdanja projekta Yocto korišćenjem alata Qemu i referentnih primeraka izvedbi fizičkih arhitektura kao i listu ispitnih scenarija koje smo koristili za verifikaciju sistema distribucije Debian koje smo razvili.

Qemu MIPS – projekat Yocto	Izvedba fizičke arhitekture – routerstation – projekat Yocto MIPS arhitektura	Izvedba fizičke arhitekture – preveden OS Debian namenjen MIPS arhitekturi
admin-tools	admin-tools	admin-tools
---	---	cap-bounds
commands	commands	commands
---	dio	dio
---	---	fcntl-locktests
---	---	filecaps
---	fs	fs
---	fsx	---
---	---	hugetlb
---	---	hyperthreading
---	io	io
ipc	ipc	ipc
math	math	math
mm	mm	mm
nptl	nptl	nptl
---	---	numa
---	---	power_managment_tests
pty	pty	pty
sched	sched	sched
syscalls	syscalls	syscalls
---	timmers	timmers
9 grupa ispitnih scenarija	14 grupa ispitnih scenarija	20 grupa ispitnih scenarija

5-1 - poređenje korišećenih ispitnih scenarija

Tabela 5-1 je data za projekat Yocto 1.7 sa korišćenim izdanjem ltp-20140422 LTP-a. Iz priložene tabele se može videti da je poklapanje korišćenih grupa ispitnih scenarija u odnosu na one koje smo mi korisitili veliko.

Rezultate dobijene ispitivanjem korenskih sistema datoteka izgrađenih od paketa distribucije Debian prevedenih za željenu arhitekturu MIPS, poredili smo kako sa ranije tazvijenim sistemima tako i sa sistemom timesys koji je bio dostupan na realnim implementacijama (radnim pločama) arhitekture MIPS. Za ispitivanje smo koristili primerke „računara“ baziranih na arhitekturi MIPS. Za ispitivanje nismo koristili QEMU već su svi sistemi ispitivani na platformama kojima su bili i namenjeni u realnom radu.

Sledeće tabele 5-2, 5-3 i 5-4 ilustruju uporedno rezultate dobijene ispitivanjem više različitih korenskih sistema datoteka.

	Ispitivanja korenskih sistema datoteka – arhitektura MIPS				
Operativni sistem	I etapa Debian 7.0	II etapa Debian 7.0	Zvanični Debian 7.0	Timesys	Timesys
MIPS Arhitektura	mips32r2	mips32r2	mips32 (mips2)	mips32 (mips2)	mips32 (mips2)
Izdanje jezgra OS Linuks	3.2.40-rtrk	3.2.39-rtrk	3.6.0-rtrk	3.2.39-rtrk	2.6.35.3-rtrk
Zamenska, virtuelna (swap) memorija	256MB	~ 4GB	256MB	~ 4GB	~ 4GB

5-2 - Ispitivanje sistema – podaci o sistemu

	Ispitivanja korenskih sistema datoteka – arhitektura MIPS				
Operativni sistem	I etapa Debian 7.0	II etapa Debian 7.0	Zvanični Debian 7.0	Timesys	Timesys
Ime ispitnog scenarrija	Kodovi otkaza				
su01	1	1	127	1	1
cron02	-	-	-	1	1
cron_deny01	-	-	-	1	1
cron_allow01	1	1	1	1	1
cron_dirs_cheks01	-	-	-	1	1
at_deny01	-	-	-	1	1
at_allow01	1	1	1	1	1
acl_test01	1	1	1	1	1
cap_bounds	1	1	1	1	1
Ar	1	-	-	-	-
Ld	1	1	1	-	-

	Ispitivanja korenskih sistema datoteka – arhitektura MIPS				
Operativni sistem	I etapa Debian 7.0	II etapa Debian 7.0	Zvanični Debian 7.0	Timesys	Timesys
Ime ispitnog scenarrija	Kodovi otkaza				
File	2	3	2	3	3
Cron	4	4	4	4	1
Logrotate	1	1	1	1	1
Mail	-	5	-	-	-
unzip01	-	-	-	1	1
rwtest01	127	127	127	-	-
rwtest02	127	127	127	-	-
rwtest03	127	127	127	-	-
rwtest04	127	127	127	-	-
rwtest05	127	127	127	-	-
ftest03	6	6	6	-	-
ftest07	6	6	6	-	-
hugemmap01	1	1	1	1	1
hugemmap02	2	2	2	2	2
hugemmap04	2	2	2	2	2
hugeshmat01	2	2	2	2	2
hugeshmat02	2	2	2	2	2
hugeshmat03	2	2	2	2	2
hugeshmctl01	2	2	2	2	2
hugeshmctl02	2	2	2	2	2
hugeshmctl03	2	2	2	2	2
hugeshmdt01	2	2	2	2	2
hugeshmget01	2	2	2	2	2
hugeshmget02	2	2	2	2	2
hugeshmget05	2	2	2	2	2
mtest06	255	-	-	-	-
mtest06_02	255	-	255	-	-
pipe_test_01	-	-	-	255	255
numa_testcases	-	-	1	1	1
chdir01A	-	-	-	1	1
fallocate01	1	1	1	1	1

	Ispitivanja korenskih sistema datoteka – arhitektura MIPS				
Operativni sistem	I etapa Debian 7.0	II etapa Debian 7.0	Zvanični Debian 7.0	Timesys	Timesys
Ime ispitnog scenarrija	Kodovi otkaza				
fallocate02	1	1	1	1	1
fallocate03	1	1	1	1	1
get_robust_list01	1	1	1	-	1
ioctl03	2	-	2	2	-
io_cancel01	-	-	1	1	1
io_destroy01	-	-	1	1	1
io_getevents01	-	-	1	1	1
io_setup01	-	-	1	1	1
io_submit01	-	-	1	1	1
shmat01	-	-	-	3	-
libevent01	127	127	127	-	-
lseek03	1	1	1	-	1
rt_sigqueueinfo01	17	17	17	-	-
send01	-	-	1	-	-
sendmsg01	-	-	1	-	-
Sendto01	-	-	1	-	-
signalfd4_01	1	1	1	1	1
signalfd4_02	1	1	1	1	1
sockioctl01	1	1	1	-	1
ssetmask01	-	-	1	-	-
sync_file_range01	1	1	1	1	1
sysctl03	1	1	1	1	1
syslog01	-	-	-	1	1
syslog02	-	-	-	1	1
syslog03	-	-	-	1	1
syslog04	-	-	-	1	1
syslog05	-	-	-	1	1
syslog06	-	-	-	1	1
syslog07	-	-	-	1	1
syslog08	-	-	-	1	1
syslog09	-	-	-	1	1

	Ispitivanja korenskih sistema datoteka – arhitektura MIPS				
Operativni sistem	I etapa Debian 7.0	II etapa Debian 7.0	Zvanični Debian 7.0	Timesys	Timesys
Ime ispitnog scenarrija	Kodovi otkaza				
syslog10	-	-	-	1	1
utimentsat01	-	-	-	1	1
clock_gettime03	1	1	1	1	1
clock_settime03	1	1	1	-	1
timer_create04	1	1	1	1	1

5-3 - Ispitivanje sistema – prikaz ispitivanja koja nisu uspešno izvršena

	Ispitivanja korenskih sistema datoteka – arhitektura MIPS – rezultati				
Operativni sistem	I etapa Debian 7.0	II etapa Debian 7.0	Zvanični Debian 7.0	Timesys	Timesys
Ukupan broj ispitnih scenarija	1212	1212	1182	1212	1212
Broj neuspelih scenarija	49	46	57	57	59

5-4 - Ispitivanje korenskih sistema datoteka – rezultati

Korišteno izdanje LTP-a smo prethodno koristili i u procesu validacije korenskih sistema datoteka prevedenih od paketa izvornog koda uključenih u izdanje 6.0 OS Debian.

Pojašnjenje uočenih otkaza:

- ld – ispitivanje neuspelo usled neusaglašenosti LTP-a sa promenama u izveštavanju o greškama koje je uvelo izdanje 4.6 prevodioca gcc. Ispitni scenario daje korektne rezultate sa ranijim izdanjima prevodioca (npr. 4.4, 4.5). Novija Izdanja LTP-a rešila su ovaj nedostatak.
- mail – ovo ispitivanje prilikom prvog kruga ispitivanja nije prijavljeno u grupi otkaza, dok se u drugom krugu prikazuje kao neuspeo. Prilikom prvog kruga ispitivanja paket je samo prividno prošao, kao posledica neprisutnosti paketa u korenskom sistemu datoteka za ispitivanje ove funkcionalnosti. Ispitni scenario je zasnovan na ranijem izdanju mailx programskog rešenja. Otkaz je istražen i pretstavlja posledicu neusklađenosti ispitnog scenarija i sistema koji je predmet ispitivanja.
- rwtest – ispitivanje neuspelo usled nedostatka skripte rwtest.sh. Ova skripta se ne dobije prilikom prevođenja paketa koji je dostupan u okviru „squeeze“ izdanja distribucije Debian.
- sockioctl01 – otkaz usled neusklađenosti LTP-a sa promenama na jezgru operativnog sistema Linuks. U jezgru operativnog sistema Linux od izdanja 2.6.39 promenjen je kod greške koji se vraća prilikom pozivanja ioctl() sistemskog poziva (ioctl sistemski poziv najčešće korišćen za kontolu fizičkih uređaja, ulaz-izlaz i druge operacije koje se ne mogu predstaviti standardnim sistemskim pozivima). Poziv sistemskog poziva o utičnicu (eng. against socket)

korišćenjem nevalidne ioctl operacije je u ranijim verzijama vraćao EINVAL (neispravan argument) umesto ENOTTY. U skladu sa time LTP u korišćenom izdanju očekuje EINVAL i prijavljuje grešku usled dobijanja ENOTTY. Ovo je ispravljeno kasnije u okviru LTP-a, pa se u skladu sa otkrivenim (detektovanim) izdanjem jezgra operativnog sistema Linuks, očekuje jedan ili drugi kod greške.

- libevent01 – ispitivanje neuspelo usled nedostatka skripte run_libevent.sh. Ova skripta se ne dobija prilikom prevođenja paketa koji je dostupan u okviru „squeeze“ izdanja distribucije Debian
- lseek03 – otkaz usled neusklađenosti LTP-a sa promenama na jezgru operativnog sistema Linuks. Od izdanja 3.1 jezgra korenskog sistema datoteka dodate su nove funkcionalnosti sistemskom pozivu lseek. LTP prilikom pozivanja sistemskog poziva, za vrednost trećeg parametra poziva, prosleđuje vrednost van opsega. Od izdanja 3.1 jezgra operativnog sistema Linuks prosleđena vrednost je namenjena novododatoj funkcionalnosti. Povratna vrednost za ovakav poziv nije odgovarala očekivanoj EINVAL (neispravan argument) već se vrednost dobijala u skladu sa proširenom funkcionalnošću ENXIO (ne postoji takav uređaj ili adresa).
- ftest03, ftest07 – poznat problem prelivanja međumemorije (bafer memorije, buffer overflow) kada se ispitivanja pozivaju u okviru veće grupe ispitivanja. Samostalno pozvana ova ispitivanja se izvrše bez otkaza. U zavisnosti o toga kako se ova ispitivanja pozivaju, kroz grupu LTP ispitivanja ili samostalno, zauzimaju drugačije prostor u sistemskom direktorijumu /tmp.
- rtsigqueueinfo01 – u okviru korišćene verzije LTP-a korišćena je numerička umesto simboličke vrednosti za vrednost signala – broj 17. Ova vrednost na arhitekturama x86, arm i powerpc odgovara simboličkoj vrednosti SIGCHILD. Na arhitekturi MIPS ova vrednost odgovara simboličkoj vrednosti SIGUSR2 dok simbolička vrednost signala SIGCHILD odgovara numeričkoj vrednosti 18. Problem je korigovan korišćenjem simboličke (SIGCHILD) umesto numeričke vrednosti (17) u novijim izdanjima LTP-a.
- get_robust_list01 – otkaz usled neusklađenosti LTP-a sa promenama na jezgru operativnog sistema Linuks. Promene predstavljaju preventivnu sigurnosnu zakrpu. Novija izdanja LTP-a usklađena su sa ovom promenom.

Operativni sistem Debian je aktivan projekat koji raste i razvija se dodavanjem novih i promenom već postojećih paketa, jezgro operativnog sistema Linuks takođe dobija nove mogućnosti i menja ili proširuje već postojeće funkcionalnosti. Ove promene uslovljavaju i prilagođavanje sistema namenjenih ispitivanju koji mora da prati promene i prilagodi se njima u cilju davanja pouzdanih rezultata ispitivanja.

Za ispitivanje smo koristili paket LTP-a (ltp20091231) koji je dostupan u okviru izdanja „squeeze“ distribucije Debian operativnog sistema Linuks. Odlučili smo se za ovo izdanje paketa koje je već bilo uključeno u okviru izdanja distribucije Debian koje smo među prvima prevodili za ciljane arhitekture MIPS. Kako bi rezultati što duže bili uporedivi u odnosu na ranije isporuke (a i kako se u okviru distribucije Debian nije pojavljivalo novije izdanje paketa) držali smo se istog izdanja paketa. Sa napredkom distribucije Debian pojavila se potreba i zahtev za novim ciklusom prevođenja sada novog, aktuelnog izdanja distribucije Debian za arhitekture MIPS za koje smo ranije prevodili pakete.

To ilustruju tabele kao i „lažni“ otkazi otkriveni zastarelim izdanjem LTP-a primenjenim za ispitivanje korenskog sistema datoteka zasnovanog na paketima koji po izdanju odgovaraju izdanju „wheeze“ distribucije Debian uz korišćenje izdanja jezgra operativnog sistema Linuks 3.2.xx

Kako arhitekture MIPS za koje smo prevodili pakete nisu bile u okviru zvanično podržanih arhitektura od strane projekta Debian morali smo da iznova prevodimo novija izdanja paketa kako bismo dobili bazu za izgradnju sistema distribucije Debian za željene arhitekture.

Novije izdanje distribucije Debian je pratila i potreba za novijim izdanjem jezgra operativnog sistema Linuks. U pokušaju da rezultati koje smo dobijali ispitivanjem budu uporedivi sa prethodno isporučivim verzijama koristili smo ranije dostupno rešenje LTP-a. Suočili smo se sa novim otkazima koji su uslovljeni neusklađenošću jezgra operativnog sistema i izdanja distribucije Debian sa izdanjem LTP-a. Kako sama distribucija Debian nije nudila novije izdanje LTP-a, odlučili smo se za korišćenje klasičnog paketa izvornog koda koji bismo konfigurisali i prevodili na ciljnom sistemu za koji su prethodno prevedeni paketi. Drugim rečima, sam paket LTP nije prevođen zajedno sa paketima distribucije Debian, već je naknadno prevođen što je uvelo još jednu nesigurnost u pogledu konfiguracije LTP-a koja zavisi od trenutno instaliranih paketa u samom korenskom sistemu datoteka, a još zahtevalo i čuvanje izveštaja konfiguracije LTP-a pre započetog procesa ispitivanja.

U procesu ispitivanja koristili smo primerke platforme MIPS namenjene razvoju. Najčešći način podizanja sistema na ovim platformama je podizanje jezgra operativnog sistema Linuks i korenskog sistema datoteka distribucije Debian sa mrežnog sistema datoteka NFS putem komandne linije Yamon-a. Prilikom ovakvog načina podizanja sistema jezgro operativnog sistema Linuks i korenski sistem datoteka se nalaze na udaljenom skladištu na mreži. Skladište funkcioniše kao poslužilac mrežnog sistema datoteka (NFS server). U ovakvom radu sve akcije na korenskom sistemu datoteka, otvaranje, čitanje, pisanje, zatvaranje, vrše se na udaljenom resursu koji podrazumeva drugačiji način rada sa datotekama. U radu sa datotekama u lokalnom korenskom sistemu datoteka moguće je samo pozvati komandu unlink(), dok je u radu sa korenskim sistemu datoteka sa mrežnog sistetema datoteka potrebno zatvoriti datoteku naredbom close() pre korišćenja naredbe unlink().

Ovo je uslovnjeno specifičnošću u načinu rada u mrežnom sistemu datoteka. Kada se ne koristi mrežni sistem datoteka, naredba unlink() briše ime iz sistema datoteka (filesystem), ako je to ime pretstavljalo poslednju vezu (link) do datoteke i ne postoje procesi koji tu datoteku drže otvorenom, datoteka će biti obrisana i prostor oslobođen. Ako je ime koje je obrisano poslednja veza do datoteke, a neki od procesa i dalje drži otvoreno datoteku, datoteka će ostati otvorena sve dok i poslednji opisivač datoteke koji upućuje na nju (file descriptor) nije zatvoren. Ovo je mogućnost koja je pogodna za korišćenje u radu sa privremenim (tmp) datotekama, ako se izvrši komanda unlink(), a zatim proces nastavi da radi sa datotekom, ukoliko bi isti proces neočekivano završio ista bi mogla biti obrisana. Ovakav način rada je jako teško ostvariv u radu sa datotekama na mrežnom sistem datoteka, i vezan je za način na koji radi mrežni sistem datoteka.

Kada se naredba unlink() izvrši nad datotekom u mrežnom sistemu datoteka, sama datoteka će biti preimenovana u nfs<xxxxx> datoteku ukoliko i jedan proces ima ovu datoteku otvorenu. Ovakvim preimenovanjem obezbeđuje se putanja do datoteke za klijenta na mrežnom sistemu datoteka. Datoteka je primenom unlink() komande ostala bez imena ali je i dalje dostupna te je potrebno nekako pružiti klijentu mogućnost da pristupi sadržaju. Ova datoteka ne može biti obrisana sve dok ne bude zatvorena.

Kako u jednom broju ispitivanja prilikom pisanja ispitnih scenarija ovo nije uzeto u obzir isti su prijavljivali grešku, koja se daljim uvidom u rezultate izvršavanja ispitivanja manifestovala kao upozorenje da je objekat prilikom pokušanja uklanjanja nemoguće ukloniti sa greškom resurs ili uređaji su zauzeti. Uočeno je da je i sve otvorene opise-datoteka (file descriptors) potrebno, kada nam više nisu potrebni, zatvoriti, kada se radi sa korenskim sistemom datoteka sa mrežnog resursa. Ovi otkazi se nisu manifestovali kada bi se korenski sistem datoteka pokretao sa lokalnog skladišta. Kao mogući načini izbegavanja negativnih uticaja ovog problema izdvojilo se nekoliko mogućih pristupa:

- Korišćenje lokalnog skladišta za korenski sistem datoteka.
- Korišćenje lokalnog skladišta za direktorijum /tmp u koji se gotovo uvek izvršavaju ispitivanja vezana za rad sa datotekama.
- Istraživanje i ispravljanje svih ispitnih scenarija kod kojih se manifestovao sličan problem.

- Korišćenje ažurnijeg izdanja projekta LTP kod koga su ovi problemi u velikoj meri otklonjeni.

Svaki pristup ima svoje prednosti i mane.

Korišćenje lokalnog skladišta za korenski sistem datoteka zahteva prisustvo lokalnog diska (HDD), za šta kod većeg broja razvojnih platformi postoje uslovi, ali i vreme potrebno da se korenski sistem datoteka premesti na lokalni disk i sistem podesi tako da koristi lokalnu sliku. Kako je na raspolaganju ograničen broj razvojnih platformi, koje se koriste u okviru različitih projekata, a korenski sistemi datoteka, koji se ispituju, bivaju menjani i ispravljani u procesu validacije, ekskluzivno vezivanje za jednu od razvojnih ploča ili pokretanje sistema koji se ispituje sa lokalnog diska uz neophodno kopiranje i podešavanje predstavljali bi ograničavajući faktor u procesu razvoja i ispitivanja sistema.

Korišćenje lokalnog skladišta, diska za direktorijum /tmp nametnulo se kao najjednostavnije rešenje tamo gde je za to postojala mogućnost. Ovakav način ispitivanja je bio moguć jer je veliki broj platformi korišćenih za ispitivanje već imao lokalni disk koji se između ostalog koristio kao dodatni izvor virtuelne (swap) memorije u cilju prevazilaženja nedostatka veće količine radne memorije**.** Nedostatak radne memorije javlja se kao posledica korišćenja ugrađenih sistema za proces ispitivanja distribucije. Uz to ovaj pristup je samo zahtevao prevezivanje tmp direktorijuma sistema, koji se ispituje, na direktorijum na lokalnom disku, što se postiže sa par komandi nakon podizanja sistema sa udaljenog mrežnog sistema datoteka. Na primer:

```
mkdir /mnt/localhdd
mount /dev/sda12 /mnt/localhdd
mount --bind /mnt/localhdd/tmp /tmp
```

Istraživanje i ispravljane svih ispitnih scenarija za koje se utvrdi da predstavljaju otkaz usred korišćenja mrežnog sistema datoteka zahtevalo bi znatno vreme za istraživanje i ispitivanje svakog pojedinačnog otkaza koji bismo uočili. Ovakav pristup bi zahtevao i dodatno vreme za validaciju ispravki kako bi se utvrdilo da učinjene promene na sistemu nisu prouzrokovale neke neželjene promene u ponašanju LTP-a. Iako bi ovakav pristup ponudio daleko čistiji sistem nakon učinjenih izmena, ovaj pristup zahteva i najveće angažovanje za njegovu realizaciju.

Korišćenje ažurnijeg izdanja projekta LTP kod koga su ovi i drugi problemi u velikoj meri prečišćeni je dobro rešenje uz nekoliko ograničenja. Ukoliko bismo koristili sledeće izdanje izgubili bismo mogućnost poređenja jedan na jedan sa ranije dobijenim rezultatima ispitivanja korenskih sistema datoteka. Novo izdanje donosi i novi ispravljeni i proširen skup ispitnih scenarija, ovo uvodi kako ispravljene scenarije, tako i nove dotad nepoznate i u praksi neispitane ispitne scenarije. Novo izdanje često je prilagođeno novijim izdanjima paketa ili jezgra operativnog sistema Linuks. Sve ovo unosi dodatnu dozu nesigurnosti u proces ispitivanja sistema. Da bi se potvrdio nivo kvaliteta, morala bi se sva prethodna rešenja zasnovana na istoj bazi ispitati novijim izdanjem LTP-a.

Stoga smo se odlučili za pristup sa smeštanjem direktorijuma tmp na lokalni tvrdi disk za ispitivanje svih sistema zasnovanih na paketima dostupnim u okviru izdanja „squeeze“ distribucije Debian (Debian 6.0), i jezgra operativnog sistema Linuks 2.6.35

Prelaskom na izdanje „wheeze“ distribucije Debian (7.0) i jezgro operativnog sistema Linuks 3.2.39, izdanje LTP-a koje smo dotad standardno koristili se prilikom istraživanja pokazalo kao neadekvatno za ispitivanje sistema organizovanog oko aktuelnih rešenja i u neskladu sa njim. Tako se momenat prelaska na razvoj izdanja „wheezy“ distribucije Debian za MIPS arhitekture pokazao kao pogodan za razmatranje i realizovanje prelaska na ažurnije izdanje projekta LTP.

Projekat LTP nam je pružio mogućnost verifikacije rešenja koja smo razvijali. LTP nam je omogućio da identifikujemo određene otkaze. Pregledom rezultata izvršavanja i zatim njihovim daljim istraživanjem bili smo u mogućnosti da identifikujemo i otklonimo greške na nivou pojedinačnih paketa ali i jezgra operativnog sistema Linuks.

Primer paketa je paket libaio koji je vraćao neispravne kodove grešaka prilikom korišćenja sistemskih poziva u okviru paketa libaio. Prilikom greške umesto konkretnog koda greške na MIPS arhitekturi se uvek dobijala vrednost -1.

Primer koda:

```
#define io_syscall1(type,fname,sname,atype,a) \
type fname(atype a) \
{ \
  register unsigned long __a0 asm("$4") = (unsigned long) a; \
  register unsigned long __a3 asm("$7"); \
  unsigned long __v0; \
  \
  __asm__ volatile ( \
  ".set\tnoreorder\n\t" \
  "li\t$2, %3\t\t\t# " #fname "\n\t" \
  "syscall\n\t" \
  "move\t%0, $2\n\t" \
  ".set\treorder" \
  : "=&r" (__v0), "=r" (__a3) \
  : "r" (__a0), "i" (__NR_##sname) \
  : "$2", "$8", "$9", "$10", "$11", "$12", "$13", "$14", "$15", "$24",\
    "memory"); \
  \
  if (__a3 == 0) \
          return (type) __v0; \
  return (type) -1; \
}
```

Od interesa je predposlednja linija `return (type) -1; \`

Rešenje je zamena te linije sledećom:

```
  return (type) 0 -   v0; \
```

Kao primer za uočeni otkaz na jezgru operativnog sistema Linuks izdvojićemo pojavu da se na jednom tipu razvojne ploče za arhitekturu MIPS32 memcpy implementiran na nivou jezgra operativnog sistema ponašao tako da je vraćao pogrešan broj trenutno upisanih karaktera u slučaju da dođe do prekida upisa usled nekog otkaza. Ovaj otkaz je bio posebno interesantan jer se nije dešavao uvek, već samo kada je broj dotad upisanih karakera bio deljiv sa 4. Kako je ovaj otkaz bio prisutan samo pod uslovom da je došlo do otkaza i da je pritom upoređivan proveravan broj dotad upisanih karaktera i da je on bio deljiv sa 4, uočavanje ovakvog otkaza, koji je povremen i nije kritičan u odnosu na ispravan rad sistema vrlo verovatno bi ostalo neprimećeno i neotklonjeno da se nije korišten LTP za proveru sistema.

Pored ovih ispitivanja u zavisnosti od mogućnosti dostupnih sistema ispitivali smo i korisničku interakciju sa sistemom kroz korišćenje grafičke korisničke međuveze i standarnih ulaznih uređaja (miš i tastatura). Ova vrsta ispitivanja je obuhvatala neke standardne korisničke akcije i pokretanje programa i alata koji su uključeni u isporučivano grafičko korisničko okruženje uključeno u operativni sistem Debian.

6. Zaključak

Ovaj rad daje uvid u jedan od mogućih pristupa kreiranju i verifikaciji distribucija operativnog sistema Linuks. Rad je nastao na osnovu metoda primenjenih u procesu prevođenja i verifikacije izdanja „squeeze“ i „wheeze“ distribucije Debian za izvedbe arhitektura MIPS koje nisu zvanično podržane izdanjima distribucije Debian, kao i izdanja distribucije MeeGo za izvedbe arhitekture MIPS. U radu se osvrće i na proces pružanja podrške, kako jednom otkriveni nedostaci ne bi propagirali u kasnija izdanja distribucija.

Izložene metode i primenjeni postupci pretstavljaju jedno od mogućih rešenja i formirane su u odnosu na potrebe projekta i skladu sa tada trenutno dostupnim resursima. U radu se pored korištenih metoda daje uvid i u alternativne alate i postupke.

U zavisnosti od postavljenih ciljeva i dostupnih resursa alati i postupci mogu biti drugačiji od onih izloženih u ovom radu.

Mogući pravac unapređenja dobijenih rešenja bi bilo uključivanje dobijenih rešenja među zvanično podržana, kako bi bila dostupna većem broju korisnika. Samo izlaganje rešenja širem auditorijumu dalo bi precizniji uvid u prednosti i nedostatke razvijenog sistema u realnom radu i omogućilo da sistem bude održavan od šireg broja zainteresovanih učesnika.

7. Dodatak

7.1 Osnovni skup paketa - distribucija MeeGo

Osnovnu grupu paketa koja se je trebala prevesti broji 85 paketa. Kompletan spisak prevođenih paketa prikazan je na sledećoj listi:

1. acl
2. attr
3. autoconf
4. automake
5. automake14
6. basesystem
7. bash
8. binutils
9. bison
10. build
11. build-compare
12. bzip2
13. cloog
14. coreutils
15. cpio
16. cvs
17. db4
18. diffutils
19. e2fsprogs
20. ed
21. elfutils
22. expat
23. fdupes
24. file
25. filesystem
26. findutils
27. flex
28. gawk
29. gcc
30. gdbm
31. gettext
32. glibc
33. gmp
34. grep
35. groff
36. gzip
37. iso-codes
38. kernel-headers
39. libcap
40. libffi
41. libtool
42. libxml2
43. lua
44. m4
45. make
46. meego-accelerator
47. meego-release
48. meego-rpm-config
49. mpc
50. mpfr
51. nano
52. ncurses
53. net-tools
54. nspr
55. nss
56. openssl
57. pam
58. patch
59. patchelf
60. pcre
61. perl
62. perl-TimeDate
63. pkgconfig
64. popt
65. ppl
66. psmisc
67. python
68. readline
69. rpm
70. rpmlint-Moblin
71. sed
72. setup
73. sgml-common
74. shadow-utils
75. sqlite
76. tar
77. tcl
78. tcsh
79. texinfo
80. tzdata
81. unzip

82. util-linux-ng
83. xz
84. zip
85. zlib

7.2 Osnovni skup paketa - distribucija Debian

Osnovni skup paketa koji je ručno prevođen za ciljanu arhitekturu je proširen najpre na oko 100 paketa, a zatim na približno 300 paketa. Kompletan spisak prevođenih paketa prikazan je na sledećoj listi:

1. acl
2. adduser
3. apr
4. apr-util
5. apt
6. aptitude
7. apt-listchanges
8. at
9. attr
10. autoconf
11. automake
12. automake1.11
13. autotools-dev
14. base-files
15. base-passwd
16. bash
17. bash-completion
18. bc
19. bind9
20. binutils
21. bison
22. boost1.42
23. bsd-mailx
24. bsdmainutils
25. build-essential
26. busybox
27. bzip2
28. ca-certificates
29. cdrkit
30. chrpath
31. console-common
32. console-data
33. console-setup
34. coreutils
35. cpio
36. crack
37. cron
38. curl
39. cvs
40. cwidget
41. cyrus-sasl2
42. dash
43. db4.6
44. db4.7
45. db4.8
46. debconf
47. debian-archive-keyring
48. debian-faq
49. debianutils
50. dialog
51. diffutils
52. doc-debian
53. doc-linux
54. dosfstools
55. dpkg
56. e2fsprogs
57. ecj
58. eglibc
59. exim4
60. expat
61. fakeroot
62. file
63. findutils
64. flex
65. fuse
66. gawk
67. gcc-4.4
68. gcc-defaults
69. gcj-4.4
70. gconf
71. gdbm
72. geoip
73. gettext
74. ghostscript
75. git
76. glib2.0
77. gmp
78. gnupg
79. gnutls26
80. gpgme1.0
81. gpm
82. grep
83. groff
84. gzip
85. hostname
86. ifupdown
87. initramfs-tools
88. insserv
89. intltool-debian
90. iproute
91. iptables
92. iputils
93. isc-dhcp
94. iso-codes
95. jade
96. java-common
97. kbd
98. keyutils
99. klibc
100. krb5
101. less
102. libaio
103. libbsd
104. libcap2
105. libcroco
106. libcurses-perl
107. libcurses-ui-perl
108. libedit
109. libept
110. liberror-perl
111. libevent
112. libffi
113. libfile-copy-recursive-perl
114. libgc

115. libgcrypt11
116. libgpg-error
117. libgssglue
118. libhtml-parser-perl
119. libhtml-tagset-perl
120. libhtml-tree-perl
121. libice
122. libidn
123. libjpeg6b
124. liblist-moreutils-perl
125. liblocale-gettext-perl
126. liblockfile
127. libnfnetlink
128. libnfsidmap
129. librpcsecgss
130. libselinux
131. libsepol
132. libsgmls-perl
133. libsigc++-2.0
134. libtasn1-3
135. libterm-readkey-perl
136. libtext-charwidth-perl
137. libtext-iconv-perl
138. libtext-wrapi18n-perl
139. libtimedate-perl
140. libtool
141. libunistring
142. liburi-perl
143. libusb
144. libuuid-perl
145. libwww-perl
146. libx11
147. libxau
148. libxcb
149. libxdmcp
150. libxext
151. libxinerama
152. libxml2
153. libxmu
154. libxpm
155. libxt
156. linux-2.6
157. live-boot
158. live-config
159. logrotate
160. lsb
161. lsof
162. ltp
163. lvm2
164. lzma
165. m4
166. make-dfsg
167. man-db
168. manpages
169. mawk
170. mercurial
171. mime-support
172. mlocate
173. module-init-tools
174. mpfr4
175. mtd-utils
176. mtools
177. mutt
178. nano
179. ncurses
180. neon27
181. netbase
182. netcat
183. netkit-ftp
184. netkit-telnet
185. net-tools
186. newt
187. nfs-utils
188. numactl
189. openjdk-6
190. openldap
191. openssh
192. openssh-blacklist
193. openssl
194. original-awk
195. pam
196. patch
197. pciutils
198. pcre3
199. perl
200. pkg-config
201. po4a
202. po-debconf
203. popt
204. portmap
205. ppl
206. procmail
207. procps
208. psmisc
209. pth
210. python2.5
211. python2.6
212. python3.1
213. python-apt
214. python-central
215. python-defaults
216. python-support
217. rcs
218. readline5
219. readline6
220. reportbug
221. rsync
222. rsyslog
223. scons
224. scowl
225. sed
226. sensible-utils
227. serf
228. sgml-base
229. shadow
230. slang2
231. sqlite3
232. sshfs-fuse
233. stracef
234. subversion
235. sudo
236. swi-prolog
237. sysvinit
238. tar
239. tasksel
240. tcp-wrappers
241. tex-common
242. texinfo
243. texlive-base
244. texlive-bin
245. time
246. tokyocabinet
247. traceroute
248. tzdata
249. ucf
250. udev
251. unixodbc
252. unzip
253. update-inetd
254. user-setup
255. util-linux
256. vim
257. w3m
258. wget
259. whois
260. x11proto-core
261. xapian-core
262. xauth
263. xfonts-terminus
264. xft
265. xkeyboard-config
266. xml-core
267. xutils-dev
268. xz-utils
269. zlib

7.3 Primer debootstrap komande

Za kreiranje sistema Debian iz skladišta binarnih paketa koristi se alat debootstrap. Sledeća primer prikazuje komandu sa kompletnom listom paketa korištenih za izgradnju sistema sa skladišta na lokalnoj mreži:

```
$ /usr/sbin/debootstrap --verbose --exclude="ept-cache, libept-dev"
--include="g++, gdb, git, gettext-base, console-setup, doc-linux-text,
keyboard-configuration, liblzo2-2, libserf1, zip, unzip, vim, bzip2,
subversion, adduser, apt, apt-utils, aptitude, at, autoconf, automake,
automake1.4, autotools-dev, base-files, base-passwd, bash, bash-completion,
bc, bind9-host, binutils, bsdmainutils, bsdutils, busybox, ca-certificates,
console-common, console-data, coreutils, cpio, cpp, cpp-4.7, cron, curl, dc,
debconf, debconf-doc, debconf-utils, debian-archive-keyring, debian-faq,
debianutils, diffutils, dmsetup, dnsutils, doc-debian, dpkg, e2fslibs,
e2fsprogs, file, findutils, ftp, g++-4.7, gawk, gcc-4.7, gcc-4.7-base, gnupg,
gpgv, grep, groff-base, gzip, host, hostname, ifupdown, info, initramfs-
tools, initscripts, insserv, install-info, iproute, iptables, iputils-ping,
isc-dhcp-client, isc-dhcp-common, iso-codes, kbd, klibc-utils, less, libacl1,
libattr1, libbind9-80, libblkid1, libboost-iostreams1.49.0, libbsd0, libbz2-
1.0, libc-bin, libc-dev-bin, libc6, libc6-dev, libc6-pic, libc6-prof,
libcap2, libcomerr2, libcwidget3, libdb5.1, libdevmapper1.02.1, libdns88,
libedit2, libept1.4.12, liberror-perl, libevent-2.0-5, libexpat1, libgc1c2,
libgcc1, libgcrypt11, libgdbm3, libgeoip1, libgmp10, libgnutls26, libgomp1,
libgpg-error0, libgpgme11, libgpm2, libgssapi-krb5-2, libgssglue1,
libgssrpc4, libidn11, libisc84, libisccc80, libisccfg82, libk5crypto3,
libkadm5clnt-mit8, libkadm5srv-mit8, libkdb5-6, libkeyutils1, libklibc,
libkrb5-3, libkrb5support0, libldap-2.4-2, liblocale-gettext-perl,
liblockfile1, liblwres80, liblzma5, libmagic1, libmpfr4, libncurses5,
libncursesw5, libnewt0.52, libnfnetlink0, libnfsidmap2, libpam-modules,
libpam-runtime, libpam0g, libpci3, libpcre3, libpopt0, libpth20,
libreadline6, librpcsecgss3, libsasl2-2, libsasl2-modules, libselinux1,
libsepol1, libsigc++-2.0-0c2a, libslang2, libsqlite3-0, libss2, libssl1.0.0,
libstdc++6, libstdc++6-4.7-dev, libtasn1-3, libtext-charwidth-perl, libtext-
iconv-perl, libtext-wrapi18n-perl, libtokyocabinet9, libtool, libudev0,
libusb-0.1-4, libuuid-perl, libuuid1, libwrap0, libx11-6, libx11-data,
libxapian22, libxau6, libxcb1, libxdmcp6, libxext6, libxml2, libxmuu1, linux-
libc-dev, locales, login, logrotate, lsb-base, lsb-release, lsof, m4, make,
man-db, manpages, mawk, mime-support, mlocate, module-init-tools, mount,
mutt, nano, ncurses-base, ncurses-bin, ncurses-term, net-tools, netbase,
netcat-traditional, nfs-common, openssh-blacklist, openssh-blacklist-extra,
openssh-client, openssh-server, openssl, passwd, patch, pciutils, perl, perl-
base, perl-modules, pkg-config, procmail, procps, psmisc,python,python-
apt,python-apt-common,python-central, python-minimal, python-reportbug,
python-support, python2.6, python2.6-minimal, readline-common, reportbug,
rsync, rsyslog, scons, sed, sensible-utils, sgml-base, sudo,sysv-rc,
sysvinit, sysvinit-utils, tar, tasksel, tasksel-data, tcpd, telnet, texinfo,
time, traceroute, tzdata, ucf, udev, util-linux, uuid-runtime, vim-common,
vim-tiny, w3m, wamerican, wget, whiptail, whois, xauth, xkb-data, xml-core,
xz-utils, zlib1g" --arch mips wheezy . http://192.168.236.102:82/debian-
standard/wheezy/mips32r2-hf-grip/
```

U prethodnoj komandi se vidi lista paketa koje ne treba uključiti u sistem, kao i lista paketa koje treba koristiti prilikom izgradnje sistema. Zatim se navodi kojoj je arhitekturi namenjen sistem koji se gradi i koje se izdanje distribucije Debian izgrađuje. Na samom kraju naredbe se nalazi putanja do skladišta binarnih paketa DEB, organizovanog u skladu sa standardom alata apt, sa koga se vrši povlačenje paketa koji će biti instalirani na sistem.

8. Literatura

[1] Torvalds Linus: Notes for linux release 0,01 kernel.ort, 1991 - https://www.kernel.org/pub/linux/kernel/Historic/old-versions/RELNOTES-0.01
[2] http://www.linuxfromscratch.org/
[3] https://www.yoctoproject.org/
[4] https://www.yoctoproject.org/tools-resources/projects/poky
[5] https://wiki.yoctoproject.org/wiki/LTP_result#Test_Configuration_and_Environment
[6] https://wiki.yoctoproject.org/wiki/Qemumips-ltp
[7] https://wiki.yoctoproject.org/wiki/Beagleboard-ltp
[8] https://wiki.yoctoproject.org/wiki/Routerstation-ltp
[9] http://www.openembedded.org/wiki/Main_Page
[10] http://www.openembedded.org/wiki/OpenEmbedded-Core
[11] http://en.wikipedia.org/wiki/OpenEmbedded#Layer_organisation
[12] http://openbuildservice.org/
[13] https://www.debian.org/devel/buildd/
[14] https://www.debian.org/ports/
[15] https://www.debian.org/devel/buildd/wanna-build-states
[16] https://wiki.debian.org/sbuild
[17] http://pbuilder.alioth.debian.org/
[18] https://wiki.ubuntu.com/PbuilderHowto
[19] http://fedoraproject.org/wiki/Koji
[20] http://mirrorer.alioth.debian.org/
[21] https://wiki.debian.org/Debootstrap
[22] https://wiki.merproject.org/wiki/Image_Creation_For_Beginners
[23] http://linux.die.net/man/8/febootstrap
[24] http://fedoraproject.org/wiki/Projects/Mock
[25] https://source.android.com/compatibility/cts-intro.html
[26] http://linux-test-project.github.io/
[27] https://packages.debian.org/squeeze/ltp
[28] Paul Larson: „*Testing Linux® with the Linux Test Project“*, Ottawa Linux Symposium 2002 - https://www.kernel.org/doc/ols/2002/ols2002-pages-265-273.pdf
[29] http://en.wikipedia.org/wiki/MeeGo#cite_note-1
[30] Grabham, Dan (2010-02-15). "Intel and Nokia merge Moblin and Maemo to form MeeGo". techradar.com.

http://www.techradar.com/news/phone-and-communications/mobile-phones/intel-and-nokia-merge-moblin-and-maemo-to-form-meego-670302
[31] http://merproject.org/
[32] http://www.mentor.com/embedded-software/codesourcery
[33] https://www.debian.org/
[34] http://www.emdebian.org/
[35] http://collab-maint.alioth.debian.org/debtree/
[36] Dominic Sweetman, **See MIPS® Run, Second Edition**
[37] https://gcc.gnu.org/onlinedocs/gcc-4.4.5/gcc.pdf
https://gcc.gnu.org/onlinedocs/gcc-4.7.2/gcc.pdf
[38] https://gcc.gnu.org/wiki/Atomic/GCCMM
[39] https://wiki.yoctoproject.org/wiki/LTP_result

Literaturi pristupljeno 12.2014. godine