

УНИВЕРЗИТЕТ У БЕОГРАДУ  
МАТЕМАТИЧКИ ФАКУЛТЕТ

Стефана Церовина

**ДАРТИНО ВИРТУЕЛНА МАШИНА,  
ИМПЛЕМЕНТАЦИЈА  
ИНТЕРПРЕТАТОРА ЗА ПЛАТФОРМУ  
МИПС**

мастер рад

Београд, 2016.

**Ментор:**

др Милена ВУЛОШЕВИЋ ЈАНИЧИЋ  
Универзитет у Београду, Математички факултет

**Чланови комисије:**

др Саша МАЛКОВ  
Универзитет у Београду, Математички факултет

др Филип МАРИЋ  
Универзитет у Београду, Математички факултет

Датум одбране: \_\_\_\_\_

*Брату, маме и тати*

**Наслов мастер рада:** Дартино виртуелна машина, имплементација интерпретатора за платформу МИПС

**Резиме:**

**Кључне речи:** МИПС, наменски уређаји, интерпретатор,

# Садржај

<b>1</b>	<b>Увод</b>	<b>1</b>
1.1	Кратак увод и мотивација . . . . .	1
<b>2</b>	<b>МИПС</b>	<b>3</b>
2.1	ЦИСК и РИСК . . . . .	3
2.2	Проточни систем (pipeline) . . . . .	3
2.3	Специфичности МИПС платформе у односу на ЦИСК . . . . .	3
2.4	Поређење МИПС платформе и Интел x86-64 . . . . .	3
2.5	Поређење МИПС платформе и АРМ . . . . .	3
2.6	Архитектура МИПС32 . . . . .	3
2.7	Значај и употреба МИПС платформе у наменским уређајима . . .	3
<b>3</b>	<b>Програмски језик Дарт</b>	<b>5</b>
3.1	Уопштено . . . . .	5
3.2	Приватност . . . . .	5
3.3	Конкурентност . . . . .	5
3.4	Грешке и упозорења . . . . .	5
3.5	Променљиве . . . . .	5
3.6	Функције . . . . .	5
3.7	Оператори . . . . .	5
3.8	Класе . . . . .	5
3.9	Интерфејси . . . . .	5
3.10	Енумерације . . . . .	5
3.11	Генерици . . . . .	5
3.12	Уграђени типови . . . . .	5
3.13	Библиотеке . . . . .	5
3.14	Специфичности у односу на јаваскрипт . . . . .	5

3.15 Платформе . . . . .	5
<b>4 Дартино</b>	<b>6</b>
4.1 Опис виртуелне машине . . . . .	6
4.2 Модел програмирања . . . . .	8
4.3 Значајне карактеристике . . . . .	9
4.4 Најзначајније библиотеке . . . . .	14
4.5 Покретање Дарт програма помоћу Дартино ВМ . . . . .	15
<b>5 Имплементација интерпретатора за платформу МИПС</b>	<b>18</b>
5.1 Имплементација у програмском језику Ц++ . . . . .	18
5.2 Нумерички алгоритам у програмском језику Дарт . . . . .	18
<b>6 Закључак</b>	<b>19</b>
<b>Библиографија</b>	<b>20</b>

# Глава 1

## Увод

### 1.1 Кратак увод и мотивација

Појам „Интернет ствари” (енг. IOT- Internet of things) се све чешће среће, и он представља модел повезаности објеката који нас свакодневно окружују у огроман систем у оквиру ког међусобно комуницирају. Дефиниција „ствари” у оквиру појма „Интернет ствари” варира, али се може рећи да је то наменски уређај(наменски систем) који преноси и прима информације путем мреже. Наменски системи су системи специјалне намене, који обављају једну или више функција које тој намени одговарају. Модерни наменски системи су углавном базирани на микроконтролерима, а ређе на микропроцесорима, зато што микроконтролере карактерише ефикасно управљање процесима у реалном времену, масовна производња, ниска цена и мала потрошња електричне енергије. Сматра се да је потенцијал за развој индустрије наменских система велики, и да је будућност у изградњи велики ИоТ система.

Дартино је пројекат компаније Гугл, настао са намером да се програмирање наменских уређаја олакша и приближи што већем броју програмера. Писање апликација за наменске уређаје се обично своди на програмирање „метала”. Микроконтролер садржи малу количину РАМ меморије, малу количину флеш меморије, па се на овако малом уређају не може покренути ниједан регуларни оперативни систем, а и микролинукс често захтева много РАМ меморије да би се покренуо. Често се програмира у асемблерском језику. То све даје тешке околности за развијање апликација. Пошто се Веб апликације много брже развијају од апликација за наменске уређаје, дошло се на идеју да се омогући

програмирање наменских уређаја у вишем програмском језику Дарт, за који постоји подршка за развој Веб, серверских и мобилних апликација. Дартино омогућава писање апликација за мале микроконтролере, на језику који доста личи на С, али је објектно-оријентисан и садржи разне погодности које процес имплементације доста олакшавају, те се апликације могу развијати брже и ефикасније.

МИПС процесори су широко распрострањени у наменским системима, те је подршка МИПС платформе на Дартино виртуелној машини од великог значаја. Пошто су у Дартину биле подржане само Интел и Арм архитектуре процесора, развијен је интерпретатор за платформу МИПС, који је у међувремену званично интегрисан у Дартино пројекат.



## Глава 2

# МИПС

### 2.1 ЦИСК и РИСК

### 2.2 Проточни систем (pipeline)

### 2.3 Специфичности МИПС платформе у односу на ЦИСК

### 2.4 Поређење МИПС платформе и Интел x86-64

### 2.5 Поређење МИПС платформе и АРМ

### 2.6 Архитектура МИПС32

### 2.7 Значај и употреба МИПС платформе у наменским уређајима



## Глава 3

# Програмски језик Дарт

### 3.1 Уопштено

### 3.2 Приватност

### 3.3 Конкурентност

### 3.4 Грешке и упозорења

### 3.5 Променљиве

### 3.6 Функције

### 3.7 Оператори

### 3.8 Класе

### 3.9 Интерфејси

### 3.10 Енумерације

### 3.11 Генерици

### 3.12 Уграђени типови

### 3.13 Библиотеке

### 3.14 Специфичности у односу на јаваскрипт

# Глава 4

## Дартино

### 4.1 Опис виртуелне машине

Дартино виртуелна машина представља виртуелну машину за симулирање језика. Виртуелне машине за симулирање језика извршавају један програм, односно један процес. Имплементирају се са циљем обезбеђивања окружења за програме, независно од платформе, а имплементација се заснива на интерпретатору.

Као што је речено у уводу, Дартино виртуелна машина је настала са циљем да се писање апликација за микроконтролере олакша. Како би се направило боље окружење за рад, потребна је отворена и приступачна платформа, која ће већини програмера бити позната. Требало би омогућити статичку анализу кода, допуњавање кода, библиотеке са разним функционалностима, и тиме развијање апликација учинити ефикаснијим.

Пре Дартина је било могуће програмирати микропроцесоре у Дарту, али не и микроконтролере. Особине микроконтролера постављају велика додатна ограничења при развоју виртуелне машине која ће на њима радити. Обично се под виртуелном машином подразумева нешто што преводи изворни код у некакав међукод, и извршава га. Према томе, виртуелна машина садржи некакву компоненту за извршавање, сакупљач отпадака, омогућује препознавање класа и објеката, и има подршку за дебаговање.

Шта садржи Дартино виртуелна машина?

1. Компоненту за извршавање
2. Објектни модел

### 3. Сакупљач отпадака

Ове три компоненте су неопходне када је у питању виши програмски језик као што је Дарт. Компајлер изворног кода је избачен, и направљен је систем у коме су компајлер и окружење за извршавање раздвојени, као што је случај код гцц-а или ллвм-а. Систем за дебаговање је да поједностављен и смањен.

Како су раздвојени компајлер и окружење за извршавање? Компајлер може да се налази било где, обично на рачунару где се развија апликација, и он комуницира са окружењем за извршавање које се потенцијално налази на другом систему. Интерфејс за комуникацију је једноставан, јер окружење за извршавање мора бити веома једноставно, како би одговарало карактеристикама микτροконтролера. Окружење за извршавање садржи командни АПИ, преко ког му компајлер може затражити да уради одређене ствари. Испод тога се заправо налази стек машина, тако да се може поставити ново стање на стек окружења за извршавање, могу се правити класе, дефинисати објекти и методе путем овог АПИ-ја.



Слика 4.1: Комуникација компајлера и окружења за извршавање

Како би компајлер знао где је која класа завршила у систему, и доделио им симболичка имена, омогућено му је да свакој класи придружи одређени идентификатор. Ови идентификатори окружењу за извршавање не значе ништа, али компајлер при примању објекта назад, зна да је инстанца одређене класе. Сва комуникација се врши преко жичног протокола, тако да је омогућено удаљено извршавање, и обично је базирано на TCP/IP протоколу. На овај начин велики део посла се пребацује на компајлер, а окружење за извршавање се максимално упростићава.

Шта се све може радити помоћу овог протокола? Најпре се може постављати структура програма, што обухвата:

1. Додавање нових класа на стек
2. Додавање нових метода на стек

Може се мењати структура програма:

1. Мењање табела метода

Може се рећи „од овог тренутка желим да ова класа поред метода А има и метод Б”. То значи да је омогућена висока интерактивност са системом.

2. Мењање шема

Мењање шема се односи на мењање класе, нпр. додавањем нових поља. Оно што се дешава у тој ситуацији је да се врши пролаз кроз све инстанце класе, и оне се ажурирају и прилагођавају новом формату класе.

3. Дебаговање

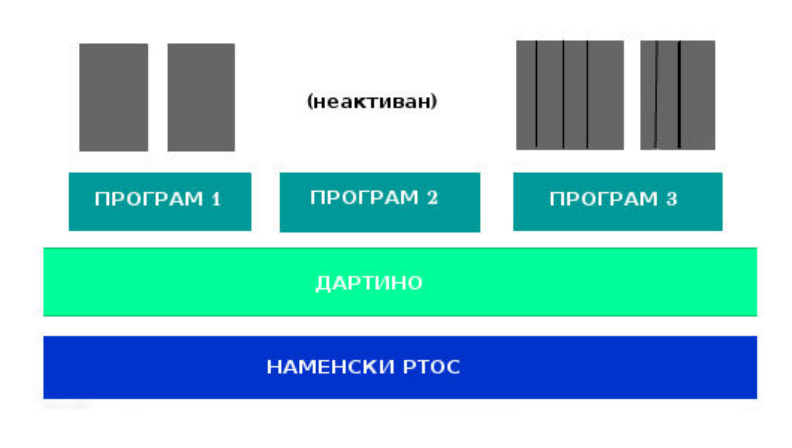
- а) Покретање
- б) Постављање зауставних тачака
- с) Рестартовање

Систем за дебаговање једноставан и ограничен, јер не зна ништа о изворном коду.

## 4.2 Модел програмирања

Доњи слој чини традиционални наменски оперативни систем у реалном времену (нпр. Фри РТОС). Изнад тога се налази Дартино окружење за извршавање, које може извршавати више програма, независно, и они не морају знати ништа једни о другима. Ово се разликује од уобичајених система који раде на наменском оперативном систему, и који обично не могу да извршавају више независних компоненти, на независан начин. За сваки програм можемо имати више процеса, више инстанци програма, или делова програма, који се извршавају конкурентно.

На слици 4.3 је приказана ситуација када имамо 3 програма. Програм2 се тренутно не извршава, за Програм1 су покренуте две инстанце, док се Програм3 извршава у 7 нити.



Слика 4.2: Модел програмирања

### 4.3 Значајне карактеристике

#### 1. Лаки процеси

За сваки процес нам је потребно мало меморије, неколико стотина бајтова. Сав код и подаци у меморији се деле, па меморију заузимају само подаци који представљају процес. На овај начин ради већина оперативних система, али је то новина за оперативне системе на наменским уређајима.

#### 2. Блокирање процеса

Процес се може блокирати, при чему се не заузимају ресурси, већ се главна нит враћа систему. Разлог за то је што у наменским системима обично имамо мали фиксирани број нити, па не желимо да их блокирамо.

#### 3. Паралелизам

Уколико процесор има више језгара, процеси се могу извршавати паралелно.

```
main() {
    for (int i = 0; i < 40000; i++) {
        Process.spawn(factoriel, 12);
    }
}

int factoriel(n) {
```

```
    if (n == 1) return 1;
    return n * factoriel(n-1);
}
```

Могуће је покренути 40000 независних процеса, који ће бити лепо распо-  
ређени на одређени број језгара и извршавати се паралелно.

Како се могу користити процеси за манипулисање надолазећим конекци-  
јама?

```
var server = new ServerSocket("127.0.0.1", 0);
while(true) {
    server.spawnAccept((Socket socket){
        //izvrsava se u novom procesu
        socket.send(UTF.encode("Hello"));
        socket.close();
    })
}
```

Креира се ServerSocket који представља сервер, затим се помоћу метода spawnAccept креира нови процес. У оквиру тог процеса се чека да кли-  
јент успостави конекцију са сервером, и самим тим блокира даље изврша-  
вање док се конекција не успостави. При успостављању конекције, кли-  
јент(socket) шаље серверу(server) поздравну поруку.

### 4. Комуникација

Комуникација између процеса се обавља слањем порука, помоћу канала  
и портова. Канал представља један ред порука. Порт представља мо-  
гућност слања поруке каналу, и без приступа одеђеном порту, не може се  
слати порука каналу који чека на другој страни порта.

Блокирање слањем порука:

```
final channel = new Channel();
final port = new Port(channel);
Process.spawn(() {
    int i=0;
    while(i < 50) {
```



```

        port.send(i++);
    }
});
while(true) {
    print(channel.receive());
}

```

Један процес који шаље поруке, и функција `print` која блокира извршавање све док не прими следећу поруку.

#### 5. Дељено непроменљиво стање између процеса

- a) Сваки непроменљиви објекат се може слати као порука, без копирања
- b) Више процеса могу користити исти објекат истовремено
- c) Нема потребе за експлицитним примитивима за синхронизацију

#### 6. Изолате

Изолате су независни воркери, свака изолата има своју хип меморију, и за разлику од нити нема дељења меморије између изолата. Изолате комуницирају само преко порука. Могу се паузирати, наставити и убити.

```

main() {
    Expect.equals(1597, fib(16));
    Expect.equals(4181, fib(18));
}

fib(n) {
    if (n <= 1) return 1;
    var n1 = Isolate.spawn(() => fib(n - 1));
    var n2 = Isolate.spawn(() => fib(n - 2));
    return n1.join() + n2.join();
}

```

Пример употребе изолата за рачунање елемената Фибоначијевог низа, рекурзивно, покретањем две независне изолате.

#### 7. Влакна

Влакна представљају нити извршавања у оквиру једног процеса, које деле спољну меморију, али не деле непроменљиву меморију. Омогућују да више независних нити једног процеса могу чекати на исту ствар (блокирати) независно једна од друге. На овај начин се имплементира вишеструка одвојена контрола тока.

```
void publishOnChange(Socket socket, String propertyName, Channel input){
    int last = 0;
    while(true){
        int current = input.receive();
        if(current != last)
            socket.send(UTF.encode('("$propertyName": $current)'));
        last = current;
    }
}

Fiber.fork(() => publishOnChange(server, "temperature", temperatureSensor));
Fiber.fork(() => publishOnChange(server, "humidity", humiditySensor));
```

Креирамо два влакна, у оквиру једног се чека на промену вредности у сензору за температуру, док се у оквиру другог чека на промену вредности у сензору за влажност ваздуха. Функција `publishOnChange` блокира извршавање док се у сензору за који је позвана не деси промена вредности. На овај начин је омогућено да се у оквиру једног процеса чека на више ствари, и нема паралелизма, већ се уколико дође до блокирања у једном влакну, прелази на извршавање следећег који је спреман за извршавање.

### 8. Корутине

Дартино има уграђену подршку за корутине. Корутине су објекти налик на функције, које могу давати више вредности, па су веома сличне генераторима. Када корутина врати вредност, зауставља се на тренутној позицији, а њен стек извршавања се чува за касније покретање. Када врати последњу вредност, сматра се завршеном, и више се не може покретати.

Пример употребе корутина:

```
var co = new Coroutine((x) {  
    Expect.isTrue(co.isRunning);  
    Expect.equals(1, x);  
    Expect.equals(2, Coroutine.yield(4));  
    Expect.isTrue(co.isRunning);  
});  
Expect.isTrue(co.isSuspended);  
Expect.equals(4, co(1));  
Expect.isTrue(co.isSuspended);  
Expect.isNull(co(2));  
Expect.isTrue(co.isDone);
```

#### 9. Динамичко отпремање метода

Динамичко отпремање метода се односи на позивање преклопљених метода класа, при чему се проблем преклапања разрешава у фази преводјења, а не у фази извршавања. Табела отпремања се имплементира као низ, у који се сваки метод, сваке класе, смешта на своју позицију. Позиција метода се рачуна помоћу редног броја класе и позиције метода. Оно што је неопходно при добијању одређеног метода из табеле је провера да ли је резултат заправо оно што нам треба. Уколико није, значи да тражени метод одређене класе не постоји. Загарантовано је константно време отпремања, и табела отпремања се израчунава пре извршавања, па представља део апликације. Пример прављења табеле отпремања:

```
class A {  
    metod1();  
}  
  
class B extends A {  
    metod2();  
}  
  
class C extends B {  
    metod1();  
}
```

```
class D extends A {
    metod3();
}
```

Класама редом придружимо бројеве: 0, 1, 2 и 3. Методима придружимо позиције на мало компликованији начин. Метод `metod1`, који је дефинисан у највећем броју класа, придружимо позицију 0, и он се смешта на почетак табеле. Методу `metod2` придружимо позицију 3, и методу `metod3` придружимо 4. Позиције морају бити јединствени бројеви, како не би дошло до преклапања метода. Приметимо да се могу појавити празне позиције у табели.

A	B	C	D	B	C		D
A.metod1	A.metod1	C.metod1	A.metod1	B.metod2	B.metod2		D.metod3
0	1	2	3	4	5	6	7

Слика 4.3: Табела отпремања за дати пример

## 4.4 Најзначајније библиотеке

1. `file` - АПИ за рад са датотекама. Подржано само када се Дартино извршава на ПОСИКС платформи.
2. `ffi` - 'foreign function interface' библиотека која омогућава да се из Дарт кода позивају функције дефинисане у неком другом програмском језику, нпр. функције програмског језика C.
3. `http` - Имплементација HTTP клијента.
4. `mbedtls` - TLS подршка, базирана на `mbedtls`. Ово се може користити на исти начин као нормални сокет (и да се прослеђује HTTP пакету).
5. `mqtt` - MQTT клијентска библиотека за MQTT протокол, IOT протокол за размену порука заснован на објављивању/претплати.

6. `os` - Приступ оперативном систему. Подржано када се Дартино извршава на ПОСИКС платформи.
7. `socket` - Дартино имплементација TCP и UDP сокета
8. `stm32` - Подршка за STM32 плоче.

### 4.5 Покретање Дарт програма помоћу Дартино ВМ

Примарни начин за комуникацију са Дартино виртуелном машином је преко команде `dartino`. Ова команда комуницира са трајним(`persistent`) процесом, који ради сав тежи посао. Помоћу трајног процеса(`dart`) `dartino` компајлира програм, а помоћу Дартино виртуелне машине (`dartino-vm`) извршава и дебагује програме .

Нешто више о командама које се користе при покретању програма:

#### 1. `dartino`

Ова команда је C++ програм. Намера је била да буде што једноставнији програм који једноставно прослеђује стандардне улазно/излазне сигнале трајном процесу преко сокета. Ово је мали извршни фајл за покретање компајлера. Компајлер је написан у Дарту и извршава се у оквиру Дарт виртуелне машине.

#### 2. `dartino-vm`

Самостална виртуелна машина која подржава превођење програма и покретање програма из бајткода.

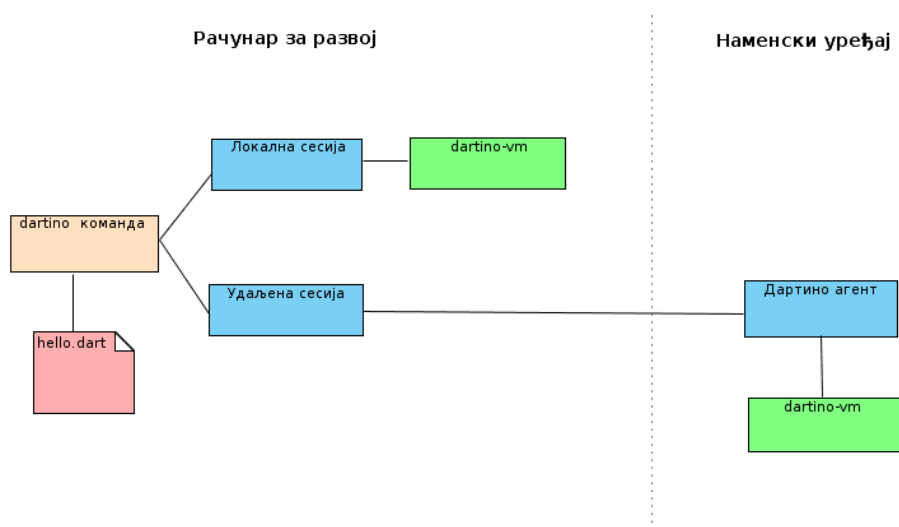
#### 3. `dart`

Трајни процес је Дарт програм: Класична Дарт виртуелна машина за покретање компајлера. Овај извршни фајл се не генерише у оквиру Дартина, већ је део Дарта. Трајни процес се састоји из главне нити, и неколико воркера. Главна нит ослушкује надолazeће конекције из дартино команде, и може да користи воркере да обави тражени задатак.

Шта се заправо дешава када се покрене команда „`dartino run hello.dart`”? Дартино је подразумевано повезан на локалну сесију, која је повезана на ло-

калну виртуелну машину која ради на нашем рачунару. „dartino” преводи код до бајткода, а онда га прослеђује „dartino-vm” која га извршава. Дартино виртуелна машина враћа резултат назад Дартину, и он га приказује.

Дартино подржава више сесија. Свака сесија може бити придружена различитој Дартино виртуелној машини, што омогућује кориснику да тестира више различитих уређаја истовремено. Тренутно је руковање сесијама експлицитно.



Слика 4.4: Процес извршавања програма

Покретање програма у оквиру одређене сесије је омогућено на следећи начин:

```
./out/DebugXMIPS/dartino create session my_session
```

Креирање нове сесије „my\_session”.

Након тога морамо да покренемо дартино виртуелну машину:

```
./out/DebugXMIPS/dartino-vm
```

Након чега се исписује порука типа „Waiting for compiler on 127.0.0.1:61745”. 61745 представља рандом генерисани порт.

У новом терминалу се прикачимо на дати сокет:

```
./out/DebugXMIPS/dartino attach tcp_socket 127.0.0.1:61745 in session my_session
```

А затим покренемо жељени програм у оквиру те сесије:

```
./out/DebugXMIPS/dartino run hello.dart in session my_session
```

Сесија се прекида помоћу следеће команде:

```
./out/DebugXMIPS/dartino x-end session my_session
```

## Глава 5

# Имплементација интерпретатора за платформу МИПС

5.1 Имплементација у програмском језику Ц++

5.2 Нумерички алгоритам у програмском језику  
Дарт



## Глава 6

## Закључак

# Библиографија

- [1] Yuri Gurevich and Saharon Shelah. Expected computation time for Hamiltonian path problem. *SIAM Journal on Computing*, 16:486–502, 1987.
- [2] Petar Petrović and Mika Mikić. Naučni rad. In Miloje Milojević, editor, *Konferencija iz matematike i računarstva*, 2015.