Мај 2014.

ЛиБРЕ!

Часопис о слободном софтверу

број 25

7. maj 2014.
Прво јавно издање LXQt-а доступно је за преузимање.

31. maj, 2014.
Објављен је Linux Mint 17 „Qiana“.

Реч уредника

Медији, *FLOSS*, ЛиБРЕ! и природна катастрофа

Природа, с времена на време, подсети човека колико је мали, колико су његове рукотворине безначајне и подсећа човека да није господар света него смо један његов мали део. Суочен са снагом природе, човек заборавља стечене навике и ослања се на базичне инстинкте. Када се нешто овако катастрофално деси, „ресетовани" на основне инстинкте, показујемо ко смо заправо, колико знамо тј. не знамо, колико смо спретни, организовани, где смо рањиви, како смо припремљени, шта нисмо на време урадили а требало је, шта смо урадили а нисмо смели па сад трпимо последице и тако даље.

Главна опасност од природне непогоде је прошла. Преживелима је сада остало да анализирају шта се десило, да попишу губитке, штету и да сви заједно почнемо да се обнављамо. У ову анализу дешавања треба укључити и употребу медија с циљем кризног информисања. Правовремена, тачна информација је била од кључног значаја за исправно реаговање с циљем преживљавања.

Ово није прва ванредна ситуација у којој је најстарија комуникациона технологија била од кључне важности. Радио техника и радио аматери још једном су показали да су незаобилазан фактор добре цивилне заштите. Просто је невероватно да нико из Министарства одбране никад није укључио те људе у институционални систем одбране, иако су се у много прилика до сад показали као врло поуздани систем информисања у кризним ситуацијама. Они не траже много, никад и нису тражили готово ништа, а увек су на првим линијама одбране.

Радио и телевизија су се углавном добро показали у овим временима. Радио а нарочито телевизија показали су још једном да су најмоћнији медији на нашим просторима. Радио и телевизија били су главни информативни, мотивациони и мобилизациони фактор у кризној ситуацији.

Најновије технологије су подбациле. Ни интернет ни мобилна телефонија нису били искоришћени у мери колико је то могуће у ширењу праве информације. Могло би се рећи да су те технологије биле чак злоупотребљене за ширење лажних и „дестабилишућих" информација. Ова критика иде директно на рачун Владе Србије и државних органа. Државни органи још нису схватили моћ *ICT*-а. Препуштање *ICT*-а стихији, непостојање званичних информација могу само да доведу до ширења дезинформација на интернету, а интернет је комуникациони медиј број један у свету. Дезинформација на интернету ће увек бити. Сетимо се вести о пијаном Србину који је убио ајкулу случајним скоком на њену главу, чиме је спасио купаче у Египту. Међутим, постоје поуздани извори на које се обраћа пажња и остали извори који се занемарују. Поуздани извори морају бити странице државних органа и локалних самоуправа које су овога пута касно реаговале, или нису реаговале уопште, а то је отворило простор незваничним изворима као што су друштвене мреже где нису сви добронамерни.

Један од светлих примера приватне иницијативе је локација http://poplave.rs Ово је пример како приватна иницијатива, добра намера, одлична идеја, *FLOSS* и знање могу заједно да искористе праву снагу интернета. Страница је била толико добро одрађена и прегледна да смо сви у почетку помислили да је то званична страница државних органа. Страница је била важна за координацију акција добровољаца. Једини проблем са том страницом био је превелики промет који је повремено рушио сервер.

Где је ту место *FLOSS*-у? Барем што се тиче интернета, *FLOSS* представља најбољи алат за брзе интервенције. *Wordpress*, бесплатни *CMS* (енг. *Content Manager System* – систем за управљање садржајем) отвореног кода са својим додацима идеални су алати за брзо реаговање и успостављање информационог система у најкраћем времену. У рукама добрих *web* дизајнера представљају моћан алат. То су момци и девојке из Грађанског покрета *poplave.rs* искористили да у најкраћем року, за мање од двадесет и четири сата, успоставе информациони центар са провереним информацијама.

На крају, похвале иду свим добровољцима, радио аматерима, корисницима *twitter*-а, грађанским иницијативама, савезима, државним органима и свима другима на добрим акцијама које су предупредиле још горе последице природне стихије.

За ЛиБРЕ! је у овом тренутку најбитније да је преживео и ову природну непогоду. С обзиром на ниски приоритет рада у овом пројекту, дозволили смо себи кашњење од седам дана да бисмо успели да се прегрупишемо. Још нам се нису јавили сви сарадници. Надамо се да су сви добро. Након кратке релаксације, ускоро ћемо моћи поново да радимо у пуном капацитету. Позивамо лекторе, графичаре, ауторе и све друге који би желели да нам се прикључе, да се јаве на нашу већ познату адресу libre [et] lugons [dot] org.

До читања

ЛиБРЕ! Тим

Моћ слободног софтвера

Број: 25
Периодика излажења: месечник

Извршни уредник: Стефан Ножинић

Главни лектор: Александар Божиновић

Лектура:
Милена Беран
Јелена Мунћан
Маја Панајотовић
Александра Ристовић

Редакција:
Александар Станисављевић
Дејан Чугаљ
Сандрина Димитријевић
Горан Мекић
Александар Тодоровић
Марко Кажић
Недељко Стефановић
Вељко Симић
Михајло Богдановић
Никола Харди
Гаврило Продановић
Жељко Шарић
Александар Брковић
Данило Ђокић
Владимир Цицовић
Петар Симовић
Јоаким Јањатовић
Ромео Млинар
Стефан Стојановић
Дејан Петровић
Марко Новаковић
Златан Васовић
Жељко Попивода

Сарадници:
Велимир Бакса
Милован Кривокапић
Иван Булатовић
Ладислав Урошевић
Тамара Ђорђевић
Бојан Богдановић
Владимир Попадић

Графичка обрада:
Дејан Маглов
Иван Радељић

Дизајн:
Младен Шћекић
Зоран Лојпур

Контакт:
IRC: #floss-magazin на irc.freenode.net
Е-пошта: libre@lugons.org

http://libre.lugons.org

ЛиБРЕ! пријатељи

ubuntu SRBIJA

Slackware Srbija

webserveri.info
info@webserveri:~$ Sve o serverima i programiranju

ubuntu LoCo Montenegro

LinuX User Group of Novi Sad
LUGoNS
Grupa korisnika GNU/Linux operativnog sistema u Novom Sadu

ЛИНог
Линукс блог | вести и мишљења

REGIONALNI
LINUX PORTAL
linuxzasve.com

archlinux.rs

LXQt - окружење радне површи следеће генерације
7. мај, 2014.

Прво јавно издање *LXQt*-а, следећа генерација популарног лаког (енг. *light-weight*) *Linux* окружења радне површи *LXDE*, доступно је за преузимање.

Користан линк: http://j.mp/1kn3lUO

Oracle добио спор против Google-а у вези са коришћењем Јаве на Android-у
12. мај, 2014.

Амерички суд за жалбе је сада преиначио одлуку из 2012. године, по којој структура *Java API*-ја не може бити заштићена ауторским правом. *Oracle* сада може да настави да од *Google*-а тражи отштету у висини од милијарду долара.

Користан линк: http://j.mp/1oLaLV2

Следећа три Linux Mint издања базираће се на Ubuntu 14.04 LTS
14. мај, 2014.

Linux Mint 17, *17.1*, *17.2* и *17.3* користиће *Ubuntu 14.04 LTS* као базу уместо да буду базирани на новијим *Ubuntu* издањима.

Користан линк: http://j.mp/1kn3Iio

Plasma Next
14. мај, 2014.

Објављено је бета издање *Plasma Next*-а. *Plasma Next* се развија са планом да замени актуелну *KDE Plasma* платформу.

Користан линк: http://j.mp/1gUU3jB

Објављен је Deepin 2014 Beta
15. мај, 2014.

Deepin је *Linux* дистрибуција базирана на *Ubuntu*-у. *Deepin 2014* доноси ново графичко окружење под називом *DDE* (*Deepin Desktop Environment*) које је базирано на *HTML5*.

Користан линк: http://j.mp/1n1l7Nz

Open source Novena лаптоп стиже на зиму
19. мај, 2014.

Novena лаптоп успео је да прикупи средства која ће му омогућити производњу. Испоруке се очекују за почетак следеће године.

Користан линк: http://j.mp/1ky9xEY

Telenav пребацује Scaut навигациону апликацију на OpenStreetMap
20. мај, 2014.

Компанија за навигацију *Telenav* најавила је да ће у својим апликацијама намењеним америчком тржишту од сада користити *OpenStreatMap* уместо *TomTom*-а.

Користан линк: http://j.mp/1u61A1g

RawTherapee 4.1
22. мај, 2014.

Објављена је нова 4.1 верзија *RawTherapee*-а. *RawTherapee* је програм за обраду *RAW* слика који долази са великим бројем могућности.

Користан линк: http://j.mp/1jKLIcW

Кина је забранила Windows 8
20. мај, 2014.

Кинеска Влада је забранила коришћење *Windows 8* на званичним Владиним рачунарима. Као разлог се наводи безбедност рачунара. Претпоставља се да ће то бити добра прилика да *Ubuntu Kylin* заузме место сад већ неподржаног *Windows XP*-а.

Користан линк: http://j.mp/1oLb9Ty

The Incredible Adventures of Van Helsing
23. мај, 2014.

Аутори акционе *RPG* игре *The Incredible Adventures of Van Helsing* најавили су верзију за *Linux*.

Користан линк: http://j.mp/1rzFGpp

Званично је објављен Git 2.0
29. мај, 2014.

Нова верзија доноси неколико суптилних новости које старији корисници можда неће ни приметити. Нове *default* функције су сада више „пријатељски" настројене према *Git* почетницима.

Користан линк: http://j.mp/1oLbohm

Објављен је Linux Mint 17 Qiana
31. мај, 2014.

Објављен је *Linux Mint 17 LTS* који ће бити подржан до 2019. године. Нова верзија доноси значајна унапређења корисничког интерфејса и менаџера ажурирања.

Користан линк: http://j.mp/1kwYZei

Уговор Републике Србије са *Microsoft*-ом

(7. део)

FLOSS у предузећима

Аутор: Дејан Маглов

До сада, у овом серијалу, смо анализирали стање примене *FLOSS*-а у Србији и препоручили смо шта би требало урадити да би *FLOSS* имао значајније место у корист свих. Сматрамо да су наши предлози добри и да су реално испуњиви ако би се стекли одређени услови. Услови за остваривање наших препорука јесу политичка воља, боље информисање, већи утицај, боља организованост и већа активност *FLOSS* заједница Србије. Апели нису довољни за популаризацију *FLOSS*-а. Свет покрећу интереси и то углавном економски.

Како то изгледа у свету власничког софтвера?

Интерес власника софтвера јесте да направи добар софтвер и да га рекламирају као најбољег за предвиђену функцију. Циљ компанија које стоје иза власничког софтвера, јесте да заузму монополистички положај како би могли свој производ да изнајмљују, а не да га продају. С тим циљем праве се картели компанија који на бази међусобног договора обезбеђују себи монополско место на тржишту. Постоји договор компанија власничког софтвера као и компанија који производе хардвер о заједничком наступању на тржишту. Компаније које заједно наступају, међусобно не конкуришу једна другој. Корисник је тако принуђен да, уколико жели да користи одређени производ, купи или изнајми производе још пет-шест компанија које су спрегнуте у том ланцу. Картелско повезивање је илегално у западном капитализму, али је тешко је доказати да је ово спрезање намерно. Праћење дешавања у *IT* индустрији јасно наводи на сумњу да картел постоји.

Следећи у низу интереса јесу дилери. Њихово задужење је да контролисано дистрибуирају хардвер и софтвер на одређеној територији. Они се труде да клијентима увек, када је то могуће, понуде пакет хардвера и власничког софтвера. Њихов интерес је да продају што више ових производа и да узму што већу маржу. И они, такође, имају интерес да изнајмљују власнички софтвер уместо да га трајно продају.

Крајњи корисник је последњи у том низу интереса који плаћа све. Овде морамо да станемо и раздвојимо правна од физичких лица. И једни и други имају интерес да користе рачунаре како би повећали своју продуктивност и олакшали себи рад.

Разлика између правних и физичких лица је у томе што физичка лица нико не проверава шта даље раде са својим хардвером и софтвером док се не појаве на тржишту са неким производом који је продукт тог хардвера и софтвера. Ово значи да физичка лица могу некажњено да користе пиратске копије софтвера док их користе у личне сврхе или да користе легални власнички софтвер и након истека периода изнајмљивања. Физичким лицима се нуде и јефтиније лиценце по принципу „ако прође, прошло је" (АПП). Компанијама је јасно да је инсистирање да физичка лица поштују лиценцу контрапродуктивно. За њих је боље да физичка лица користе пиратску копију њиховог производа, него да га уопште не користе. Иако на пиратерији губе велики приход, на овај начин се ствара зависност од њихових производа тако да на страни правних лица имају већу заступљеност и велики проценат наплате.

Правна лица немају избор уколико користе власнички софтвер. Они морају да буду легални јер их на то терају државни органи који су последња кључна карика у ланцу интереса. Не само да правна лица морају да плате све лиценце, него су чак те лиценце скупље за њих у односу на физичка лица. Који су онда интереси правних лица да користе власничке софтвере? Ту постоје објективни и субјективни разлози али и разлози који су плод заблуда и лоше информисаности.

Објективни разлози за коришћење власничког софтвера су:

- Непостојање *FLOSS* алтернатива за специфични софтвер потребан за обављање делатности.
- Не постоје *FLOSS* управљачки програми (енг. *drivers*) за специфични хардвер потребан за обављање делатности.
- Потреба за комуникацијом са сарадницима, клијентима, банком и државним органима који користе власнички софтвер и формате који су везани само за власнички софтвер.

Субјективни разлози су:

- Створена навика да се користи искључиво власнички софтвер.
- Непознавање *FLOSS*-а и његових могућности одбија експериментисање у условима пословног амбијента где је време једнако новац.
- Недовољна понуда обучених радника која би радила на *FLOSS* решењима.

FLOSS још увек прате гомиле заблуда које одбијају кориснике:

- *FLOSS* је претежак за коришћење и одржавање.
- Прелазак на *FLOSS* решења смањује продуктивност.
- *FLOSS* није безбедан јер иза њега не стоји фирма која га одржава.
- Модел развоја *FLOSS*-а не пружа никакве гаранције да ће одређени производ наставити да се развија.

Овакво стање „цементира" држава својом правном регулативом, својим информационим системом и инспекцијским надзором. Једини интерес државе јесте да обезбеди себи порез од продаје власничког софтвера и профита који се на тај начин остварује. Овде се крије привид да држава зарађује на власничком софтверу. Све док Србија користи власнички софтвер страних компанија, држава и њен буџет неће бити на добитку него на чистом губитку. Сви ти приходи који се остварују на основу наплате пореза на власнички софтвер, не подмирују трошкове лиценци власничког софтвера за рачунаре у државним органима ни трошкове инспекцијског надзора.

Анализа интересног ланца власничког софтвера

Анализом интересног ланца власничког софтвера може се закључити да стварни економски интерес за коришћење власничког софтвера имају само компаније које су власници тог софтвера и њихови дилери. Крајњи корисници имају делимични интерес јер обављају успешно своју делатност, али свој профит умањују плаћањем изнајмљивања „алата" за рад.

У условима када компаније власничког софтвера нису на територији државе корисника тих софтвера, држава може само да губи јер не узима порез на укупну добит те компаније. Порез од дистрибуције није довољан, јер он само смањује губитке у буџету. У овом ланцу интереса нешто није у реду. Како је могуће да овај интересни ланац опстаје, а да је једна карика (држава) на губитку, а друга (крајњи корисник), врло важна, дупло плаћа (кроз директно плаћање лиценци и посредно као буџетски обвезник) и тиме значајно умањује свој профит?

Ланац интереса за коришћење *FLOSS*-а

Ми, као бивша социјалистичка држава, потпуно смо се погубили у транзицији. Мало смо позаборављали дефиниције друштвених уређења. Јурећи за бољим животом, изгубили смо и оно што смо имали. Средства за рад су из руку радника уз њихово пређутно одобрење прешла у руке државе да би их и држава за мале паре предала у руке капиталисте

(власника предузећа). Дефиниција капитализма гласи да је то такав тип друштвеног уређења у којем је капиталиста власник средстава за производњу и да на основу тог власништва остварује право на добит. Сад долазимо до апсурда. Модернизацијом средстава за производњу наши капиталисти су увели компјутере у све сегменте пословања. Компјутери пак, користе власнички софтвер који су по закону о интелектуалној својини увек у власништву власника софтвера, према томе, наш капиталиста није у потпуности власник средстава за производњу и зато своју добит мора да дели са правим власником.

Ако наш капиталиста жели да задржи сву добит од свог посла, мора да поврати власништво над средствима за производњу. Требало би да то буде један од основних интереса за коришћење *FLOSS*-а у пословне сврхе. Некад због специфичности посла и окружења није могуће користити *FLOSS* решења, али у принципу у интересу власника јесте да користи, кад год је то могуће, софтвер који је под јавном лиценцом и омогућава му да поврати власништво над средствима за производњу.

Изнајмљивање средстава за производњу (власничког софтвера) би требало да значи да је одржавање истих одговорност правог власника (власника софтвера). Међутим, у пракси то није баш тако. *Enterprise* одржавање власничког софтвера се додатно наплаћује, а лиценце покривају само право на коришћење тј.

чисти намет на власничко право.

Одржавање *FLOSS* решења је одговорност корисника. Постоје гомиле бесплатне документације о начину одржавања *FLOSS* решења тако да корисник може сам да га одржава, али исто тако може и да пита за савет *FLOSS* заједницу која углавном брзо реагује и може да реши већину битних проблема. У интересу *FLOSS* заједница је да се популаризује употреба *FLOSS*-а у пословне сврхе. *FLOSS* заједнице су флексибилне и верујемо да би брзо могле да реагују на измене климе (већи интерес за одржавање пословних система) и да обезбеде легалну *enterprise* подршку.

Српске *FLOSS* заједнице већ показују знање и умеће у развијању српске *Linux* дистрибуције и у случају повећаног интересовања барем једна може да буде понуђена са *enterprise* подршком.

Програмери би могли да се не сложе са нама и да кажу да им концепција власничког софтвера више одговара. Ми мислимо да нису у праву и да ће преласком на *FLOSS* имати више посла. Концепт комерцијалног *OSS*-а па чак и власничког софтвера који може да се портује на *OS* отвореног кôда, неће им смањити приходе, имаће мању конкуренцију великих фирми које производе софтвер, радиће за познатог наручиоца. Концепт власничког софтвера није измишљен за малог програмера, него да заштити интересе великих компанија. Велике компаније немају проблем да за мале паре откупе добру идеју од малог програмера, или му је чак украду, уколико идеју није адекватно заштитио. *Copyright* није прилагођен малом програмеру, превише је скуп и компликован за њега. Трећи интерес програмера да пређу на *OSS*, јесте тај да алат за програмера апликација за власнички софтвер је такође власнички софтвер па према томе и ту мали програмер губи део своје добити.

Масовније прихватање *FLOSS*-а, нарочито у пословном окружењу, отвара могућност даљег развоја *IT* индустрије који је овог пута базиран на *OSS*-у. Сви школовани програмери не могу да се запосле у великим фирмама власничког софтвера. Овим се отвара простор за мање софтверске фирме које ће развијати *OSS* решење или нудити услуге везане за *OSS*. Држава која је била губитник у ланцу интереса око власничког софтвера, делимичним преласком

фирми на *FLOSS* ће изгубити нешто прихода од пореза на промет власничког софтвера. То може да надокнади из пореза на добит нових *OSS* фирми. Са друге стране, држава такође мора да барем делимично пређе на *FLOSS*. То раде земље Европске уније па према томе то је будућност и српске администрације. Ове три године, колико важе лиценце за *Microsoft*-ове производе по уговору који је и повод за овај серијал, треба искористити за почетак промене курса и преусмеравање администрације на *FLOSS*. На руку *FLOSS*-а би требало да иде и чињеница да *Microsoft* више не пружа подршку за *Windows XP*. Наш предлог је да се прелазак на *FLOSS* управо покрене преласком тих рачунара на неку лаганију верзију *Linux*-а и провери колико такво решење може да задовољи потребе корисника тих рачунара. Исправни рачунари свакако нису за рециклажу. У најгорем случају, могу бити донирани некој школи која нема рачунаре. Ми верујемо да ће наставници знати шта ће са њима.

Интерес одговорне државе јесте да има балансирани однос између прикупљеног пореза на промет власничког софтвера и трошкова лиценци за власнички софтвер у својој администрацији. Затим да својим деловањем не намеће непотребни трошак фирмама у виду обавезе да користе власнички софтвер да би комуницирале са државом. На крају, али можда и најважније, да својим деловањем омогући несметани развој свих грана *IT* индустрије па и оне базиране на *OSS*-у.

За крај

Овај наставак серијала „Уговор Републике Србије са *Microsoft*-ом" је природно дошао до теме коришћења *FLOSS*-а у пословном окружењу, али се то одлично уклапа у нови концепт ЛиБРЕ! часописа који има план да се више обраћа пословним корисницима. Овог пута смо акценат ставили на економске интересе коришћења *OSS*-а (комерцијалног и бесплатног софтвера отвореног кôда). Поред економског интереса, ту постоји интерес заштите приватности, интерес веће контроле система, сигурности, независности од вендора (власника софтвера), развоја из сопствених ресурса и други.

О овим и другим интересима неће бити речи унутар овог серијала, али ће бити у осталим чланцима овог часописа. У овом серијалу је остала још једна важна тема – *FLOSS* заједнице, анализа стања и препоруке.

Наставиће се.

Темељ саздан од *FLOSS*-а

Аутор: Марко Кажић

Зашто *Linux* и отворени кôд морају у учионице?

Кинеска пословица каже: Када ветар промене дува, неки дижу зидове, а неки ветрењаче. Размишљајући о свом школовању и о *Linux*-у којег користимо свакодневно, аутор овог текста не успева да споји та два појма у једну складну слику. Рачунари у школи су вазда били стари, исписани, загађени *bloatware*-ом, са тридесет иконица за *Minesweeper* који је ужасно изгледао на измученом *Windows 98 Second Edition*-у. Због тога су наши почеци са *Linux*-ом били непотребно мучни и закаснели (прим.аут.).

Када причамо о едукацији и коришћењу отвореног кода у школама, далеко смо иза актуелних токова, пословично тврдоглави, не прихватамо *Linux* и *FLOSS* као платформу и филозофију добру за нашу децу и развој наше заједнице. Ако *NASA*, *Google*, Међународна свемирска станица, *IBM*, администрација Руске Федерације и многи други[1] користе *Linux*, зашто није добар за нас? Можда јер од њега не профитира нико осим обичног корисника.

О *Linux*-у се често прича као о систему који користе само *geek*-ови, они који „мрзе" *Microsoft* и *Apple* или они који имају стари рачунар за рециклажу. Можемо то тако да гледамо, али нећемо бити у праву. *Ubuntu*, као само једна од безброј *Linux* дистрибуција, у марту 2014. године има око двадесет и два милиона корисника. То нису разлози због којих је *Linux* платформа будућности а слободан софтвер филозофија будуће информатике. Разлози су много битнији и могу се садржати у овоме – Школа и деца би профитирали јер би у *Linux*-у имали јефтину (бесплатну), моћну, поуздану и безбедну платформу која подстиче лични развој и идеје дељења и јачања заједнице. Како?

[1] *List of Linux adopters* – http://en.wikipedia.org/wiki/List_of_Linux_adopters

Слобода, сигурност и избор

Поготово данас када је приватност података и њихова заштита актуелна, треба да размишљамо о својим слободама на интернету и у софтверу

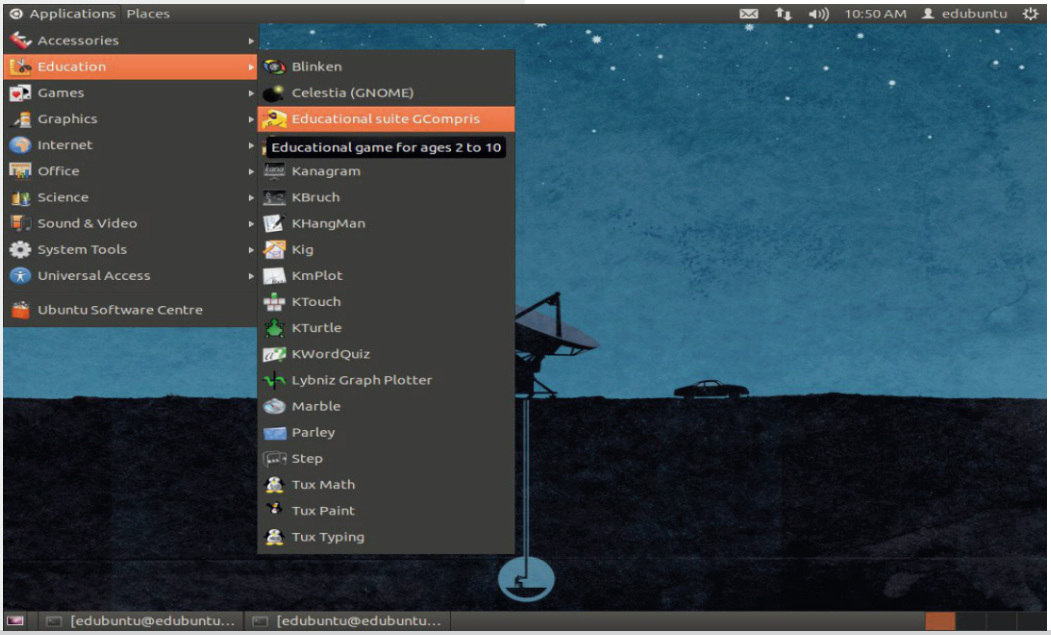
који користимо свакодневно. У школама ћемо често налетати на рачунаре пуне *spyware*-а и *malware*-а, а то једноставно нису рачунари на којима је безбедно радити. Уместо што забрањујемо деци да инсталирају апликације на те исте рачунаре и ограничавамо им права, требало би да користимо систем који подразумева безбедност, који је иоле отпоран на такву врсту злоупотребе корисничког поверења.

Деца треба да науче да преиспитују коме остављају своје податке и како се ти подаци користе. У софтверу отвореног кода, ситуација је много боља него код власничког софтвера. Ако се навикнемо да систем који можемо бесплатно да инсталирамо, променимо радну околину, дистрибуцију, софтвер који користимо, постајемо власници свог избора и својих података, а тако нешто не учимо децу у школама.

Слобода од вендора, слобода од затворених платформи

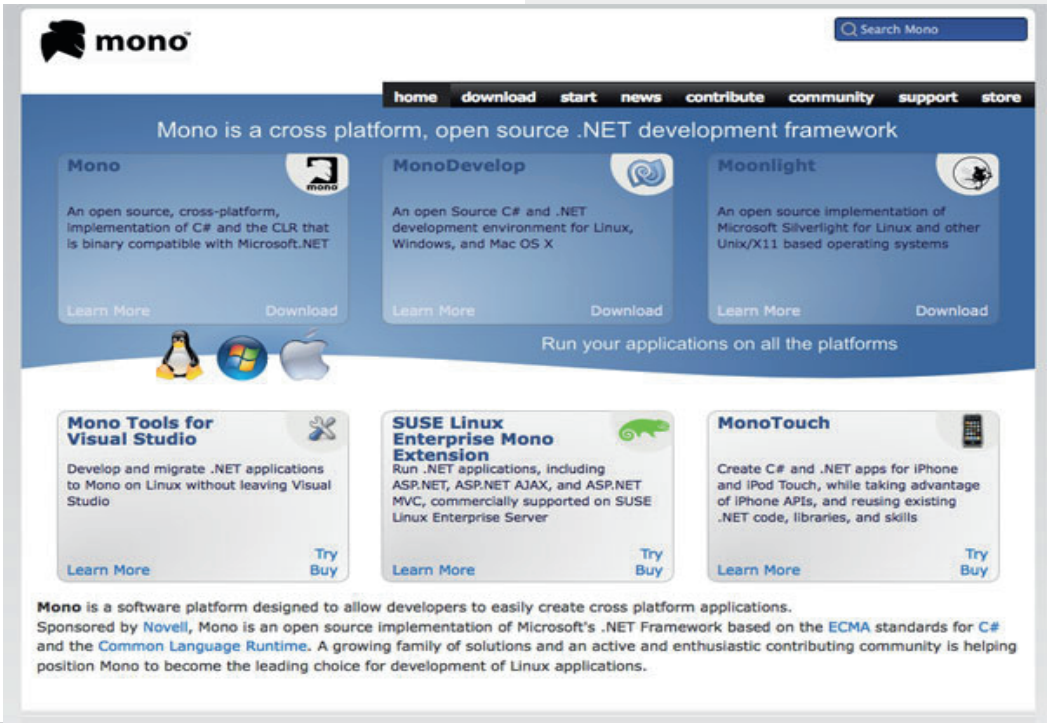
Сви ми који смо имали какав-такав контакт са информатиком у школама, знамо чему смо научени - од *Borland*-а, до *managed* окружења као што су *C#* и *.NET*. Власничке платформе су неминовно зло данашњег *IT* света, али оне нису једине које постоје. Платформа која првенствено служи да унапреди рад и побољша развој, од корисника и програмера врло брзо прави таоце који постају зависни од платформе коју плаћају. Ово је и рекурзивна поента овог чланка. Ако питамо некога зашто користи *Windows*, углавном ћете чути да апликације које користи, нису доступне на другим платформама и системима, што не би био случај да су те апликације отвореног кода.

Деца у Србији данас користе власничке



програме често не знајући да постоји алтернатива. *Office* је одличан пример. Поред затворене платформе која подразумева и затворене формате за чување докумената, у последње време имамо и *OneDrive cloud* где, гле чуда, на два клика можете да сачувате ваше податке на неком серверу хиљадама километара од вас и да га користите на другом рачунару без много муке. Ми више нисмо власници својих података. Парадоксално је али истинито. Власничке платформе улажу много у корисничко искуство и све је доступно на клик или два. Наш је задатак да едукујемо кориснике, да сваку платформу, без обзира на њено сучеље или тему, преиспитају. Они који то не могу, морају системски бити заштићени и мора им се омогућити слободна и сигурна платформа.

Корисници су лишени слободе. Школски програм је написан да подржи власничке платформе и лиценце које нису јефтине. Кажу да би промена програма коштала много. Добар пример да та тврдња није тачна, јесу управо *C#* и *.NET*. Платформе које су замишљене и реализоване тако да буду врхунац власничког кода, портоване су на *Linux*. Корисници (не компаније), направили су Mono [2]. Десило се нешто незамисливо, дешава се да све оно што је урађено у *C#* и *.NET*-у, ради на отвореној платформи. Дешава се да све што млади уче о *C#* и *.NET*-у, могу да примене на отвореној платформи. *Mono* је велика победа *open-source*-а, пројекат на који се треба угледати тражећи пример, првенствено јер не квари тако скуп и деликатно пробран план и програм за информатику који прописује



Министарство, углавном уз помоћ корпоративних спонзора.

[2] *Mono Project* – http://www.mono-project.com/

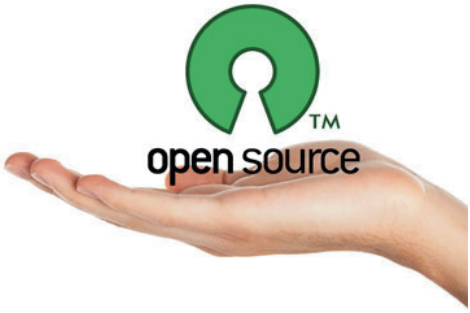
Рачунари трају дуже и доступни су свима

Са дозом скептицизма ћемо узети сопствене речи, али у пракси рачунари у школама су стари, „слаби" и тешко се обнављају. С друге стране, чак и када се нека школа понови, уобичајена је пракса да се стари рачунари баце, јер не подржавају најновију верзију *Windows*-а. То је класична прича једног конзумента. Ни на тренутак не станемо да размислимо, колико школа у Србији нема рачунаре, колико деце из најугроженијих слојева нема рачунаре код куће, колико њих би профитирало знањем и напретком када би само постојао неки начин да те старе рачунаре не бацимо, него да их поново искористимо у неку другу сврху.

Слобода – то је оно што краси *Linux*, а једна од кључних предности те слободе је да можете наћи добре дистрибуције, које сасвим комотно раде на старијим рачунарима. Иако нова технологија има своје предности и то је неоспорно, оно што ми схватамо као смеће, постаје савршено употребљив уређај оног тренутка када му се промени софтвер који га покреће. Расходован рачунар, који би користио *Linux*, могао би да служи у некој другој школи која није имала среће да добије средства за набавку нове рачунарске опреме. Бесплатан хардвер и софтвер, а заједница све јача - То је просто недопустиво.

Знање, знање, знање

Утицај *Linux*-а на развој корисника и развој деце кроз едукацију има толико подставки да их је готово немогуће скоро све навести. Раније смо напоменули да се избегавањем власничких платформи добија слобода. За почетак, слободни смо да софтвер редистрибуишемо и делимо. Младе треба учити да деле и да то није ништа лоше, иако су они годинама успешно делили инсталационе дискове *Windows*-а, сада то могу да раде као део једне поткултуре, легално и без размишљања о природи софтвера.



Могућност мењања софтвера је још једна слобода коју немамо са власничким софтвером. Корисници (они који желе) треба да знају како њихов софтвер функционише и треба да имају право да учествују у његовом развоју уколико то желе. *FLOSS* је непосредна демократија за софтвер. Иако млади данас широм Србије у свом џепу носе *Android* телефон не знајући да је *Android* у суштини платформа заснована делом на *Linux*-у, тај исти *Android* је мање-више производ

горе поменуте слободе. Неке нове платформе које у будућности могу стварати људи из Србије, неће настати од власничких платформи и технологија, већ ће настати од слободног софтвера јер се слободе потребне за добар развој пружају само његовим корисницима. Корисници морају да користе софтвер по својој мери. Код куће, на послу, у учионици, где год да сте, ваш софтвер може бити потпуно прилагођен вама. Многи су примери успешне миграције на слободан софтвер, који је пружио основу за прављење отворене едукативне платформе, у којој сви учествују. Те платформе су потпуно прилагођене потребама школа и ученика или студената, како практично по могућностима, тако и културолошки и језички. Знање и заједница су на првом месту.

Разлога је много и нема места за све, али се надамо да су најбитнији овде. Неки су практични, неки етички и филозофски, али су свакако неоспорни и ту су да остану. *Linux* је платформа коју деца данас морају да упознају, јер су они будућност, а нашу информатичку будућност не смемо градити ни на којим темељима осим на оним сазданим од *FLOSS*-а.

Преглед популарности *GNU/Linux /BSD* дистрибуција за месец мај

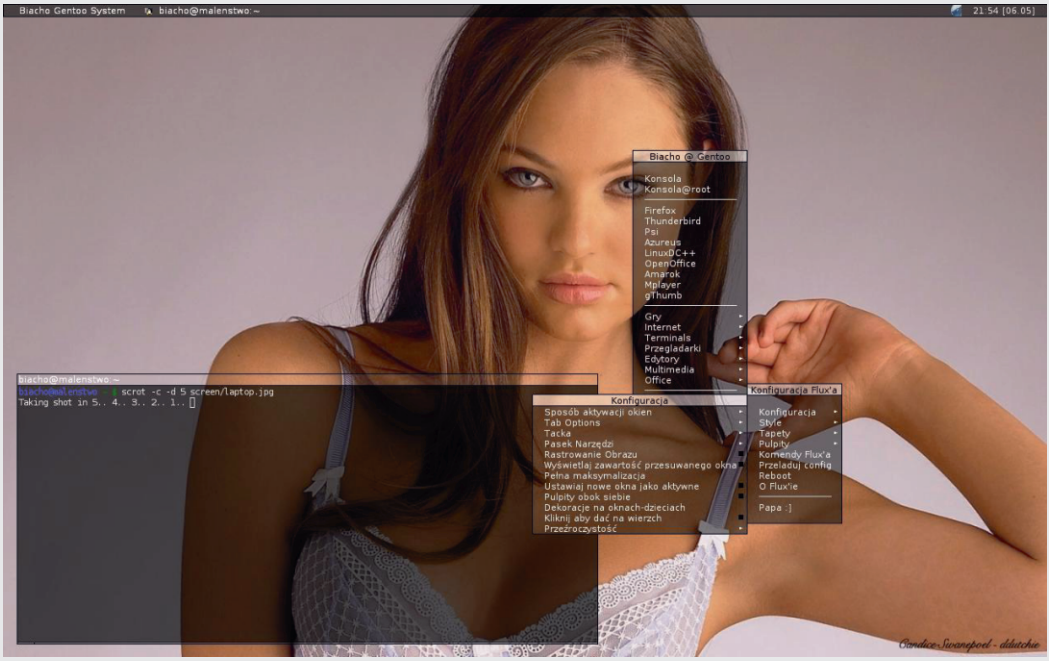
Distrowatch

1	Mint	3391>
2	Ubuntu	1795<
3	Debian	1643<
4	openSUSE	1228<
5	Arch	1172>
6	Fedora	1148<
7	Mageia	1114<
8	elementary	1021<
9	Zorin	968>
10	LXLE	893>
11	Kali	864>
12	Android-x86	801>
13	Lubuntu	781<
14	Puppy	776<
15	SteamOS	751>
16	Lite	740>
17	Pinguy	720<
18	CentOS	706>
19	Chakra	701>
20	wattOS	693>
21	Robolinux	679>
22	Antergos	641>
23	Sabayon	640<
24	Bodhi	608<
25	CrunchBang	599<

Пад <
Пораст >
Исти рејтинг =
(Коришћени подаци са *Distrowatch*-а)

Powered by

Gentoo



Аутор: Стефан Ножинић

Често имамо прилику да пишемо о новим дистрибуцијама које су, пре свега, намењене људима који се први пут срећу са *Linux*-ом. Овог пута ћемо направити изузетак и представити вам *GNU/Linux* дистрибуцију која је управо оно супротно - нипошто није за почетнике. Као што се може већ наслутити, ова дистрибуција није толико једноставна за коришћење, на први поглед. Вреди поставити питање откуд потреба за оваквом дистрибуцијом и како се она пореди са осталим дистрибуцијама у овом рангу?

Gentoo је настао 2002. године и представља једну од дистрибуција која ће вам омогућити да видите *GNU/Linux* баш онаквог какав јесте (стабилан, брз и транспарентан) и из тога да научите нешто ново и занимљиво.

Најпре ћемо покушати да одговоримо откуд потреба за оваквим приступом. *Gentoo* се инсталира тако што се преузме основни систем који у себи садржи потребан софтвер за превођење неких основних програма (енг. *compiling* - превођење изворног кода у извршни) и који служи као основа за даљу надоградњу. Ако би неко желео да ради овако нешто, то је сигурно проистекло из потребе да систем прилагоди својим потребама од самог почетка. Могућност да преведете софтвер, омогућава вам да укључите или искључите неке његове могућности, што вам даје прилику да изаберете само оно што је вама заиста потребно и тиме оптимизујете свој рачунар. На пример, ако не користите *Qt* апликације (*kde*), можете искључити *Qt* подршку за разне библиотеке. Обично кад се помене чување ресурса, одмах се помисли на старе рачунаре. У овом случају се никако не мисли на старе рачунаре, већ на новије рачунаре чији корисници желе да искористе сву њихову снагу само на онај део који је њима потребан. Ако на тренутак заборавимо чињеницу да су нам перформансе битне, овакав приступ има и других предности. Једна од доста битних предности јесте минимализација система. Систем од само неопходног софтвера јесте доста једноставан за одржавање, а и мања је вероватноћа да ће нешто кренути наопако, што омогућава лако одржавање корисницима, који су најчешће и у улози систем администратора на својим личним рачунарима.

Пошто смо видели какве су предности оваквог приступа, вреди запитати се да ли постоје слична решења *Gentoo*-у и која

је суштинска разлика између њих. Многи ће на помен градње сопственог система од нуле прво помислити на *LFS* (енг. *Linux from scratch* - систем од нуле) који даје могућност градње система од самог почетка. Разлика између *Gentoo*-а и *LFS*-а јесте та што се уз *Gentoo* добија мало обимнија основа, па је тиме неки посао који је свакако неопходно урадити на већини система оваквог типа, већ урађен за вас. Уз *Gentoo*-ов основни систем ћете добити преводилац (енг. *compiler*), *shell*, управник пакета о којем ће касније бити више речи и многе друге неопходне алатке. Сигурно нећете добити *kernel* и окружење радне површи.

Системи који користе сличан метод управљања пакетима, јесу *BSD* системи. Они користе портове који су слични *Gentoo*-овом *Portage* стаблу. Овде треба напоменути да *BSD* системи не спадају у *Linux* дистрибуције, па тиме не користе *Linux* језгро.

Управник пакета

На *Gentoo*-у је генерално усвојен принцип да се сав софтвер (са неколико изузетака) директно преводи из изворног кода у крајњи продукт (извршни део и додатни подаци). Ово се постиже помоћу система за управљање пакета који ће компајлирање урадити уместо

корисника, али ће вам опет дати могућност избора који бисте имали и током ручног компајлирања. Овај систем се назива *Portage* и званичан је управник пакета. Избор могућности за сваки софтвер (нпр. да ли ће се укључити подршка за *pulseaudio* у неком софтверу) постиже се тзв. *USE* флаговима (енг. *flags*). Они могу важити глобално за све програме (рецимо ако за све програме желите да укључите подршку за *systemd*), а може бити појединачно постављен за сваки. Постоји могућности и *patch*-овања софтвера, што је још једна предност оваквог приступа. Треба напоменути да поред званичних пакета постоје и тзв. *overlays* који омогућавају коришћење софтвера који није у званичној ризници. Сваки пакет, био он званичан или не, јесте најчешће једна скрипта (*ebuild*) која описује како се тај софтвер преводи и инсталира, тако да управник пакета зна за сваки пакет како да га преведе, подеси и инсталира. Поред ове скрипте, налазе се још и додатне датотеке као што су *dekstop* датотеке које су битне за графичка окружења, а које обично не долазе од аутора самих програма. Као што сте већ могли да претпоставите, пакет не садржи изворни код програма, већ је његова локација забележена у *ebuild*-у и током инсталације се она преузима са интернета ако већ није преузета. *Portage* нуди и аутоматско разрешавање зависности, што доста олакшава читаву процедуру инсталације и надоградње. За похвалу је могућност маскирања пакета која пружа доста једноставан метод да се пакет врати на претходну верзију, или да се прескочи нека верзија одређеног пакета. Ово може бити доста корисно ако желите да имате

стабилан систем. *Gentoo* се једном инсталира и потом је потребно само радити надоградњу пакета, што се зове *rolling release*, па тако можете добити систем који, ако нисте ништа забрљали (прим.аут.), можете користити док ваш рачунар нормално функционише.

Документација

Иако ово све звучи компликовано, тежина инсталације и подешавања се огледа у документацији, а за *Gentoo* можемо слободно да кажемо да је она и више него одлична, али и да подразумева предзнање. Треба похвалити и домаћу заједницу ЛУГоНС која се постарала да *Gentoo*-ов званични приручник, који је обавезно штиво током инсталације, преведе на српски језик и тиме олакша посао многима, којима језик може бити препрека.

Закључак

Можемо рећи да је *Gentoo* одличан систем за све оне који желе апсолутну контролу над својим оперативним системом и који су спремни да одлучују за сваку ситницу, што се на крају исплати. Исто тако, надамо се да ће *Gentoo* испробати и неки системски администратор у фирми јер је овакав приступ заиста погодан за такве услове. Ако сте већ отворили *Gentoo*-ов приручник, нема бољег закључка, него да вам пожелимо срећно компајлирање!

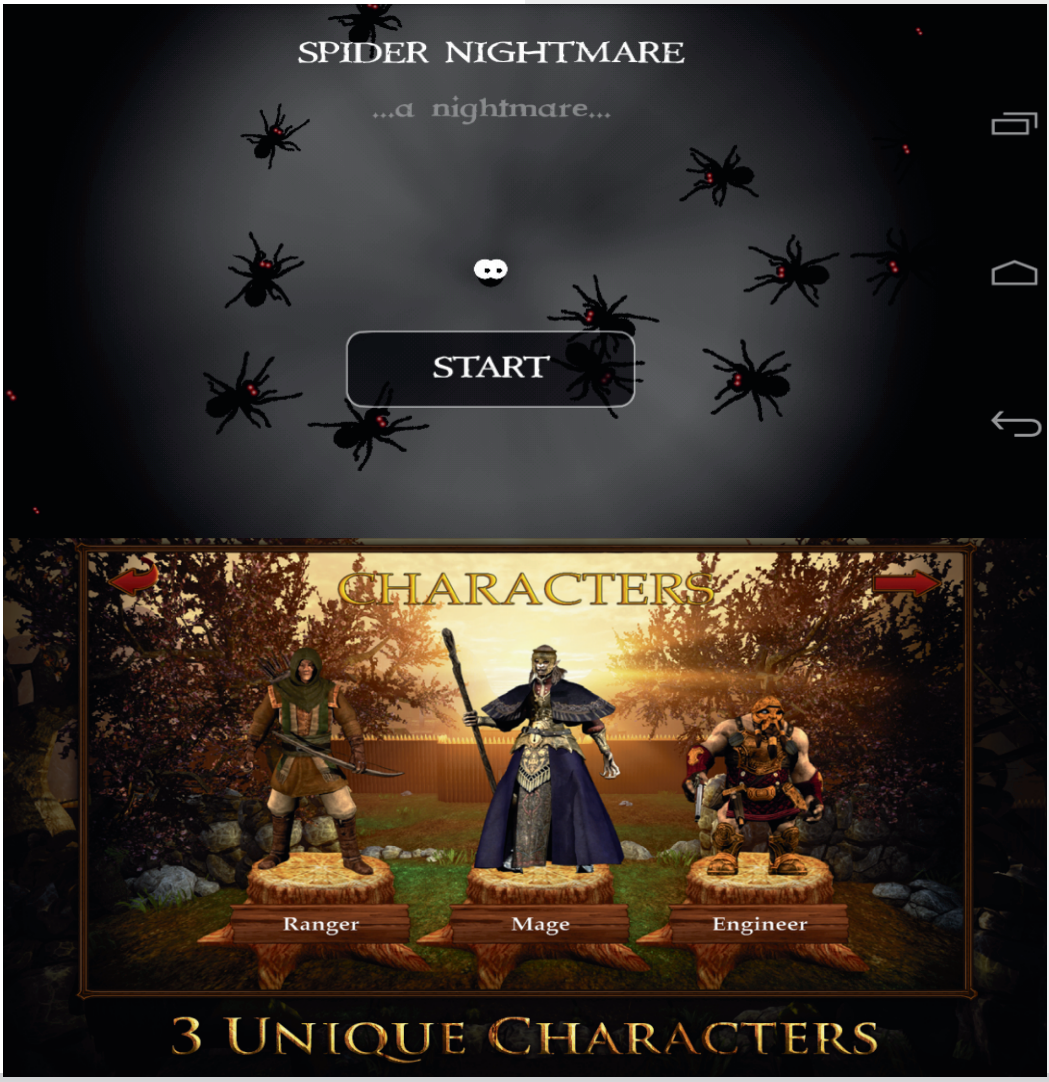
Корисни линкови:

[1] http://www.gentoo.org/
[2] http://www.gentoo.org/doc/en/handbook/
[3] https://gentoo-handbook.lugons.org/doc/sr/handbook/

libGDX

„Java game development framework”

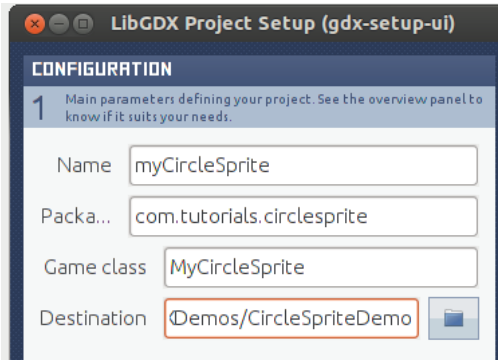
(1. део)



Аутор: Гаврило Продановић

Скоро читавој младој популацији игрице су алатка за опуштање и портал за уживање у некој другој димензији. Многи одрасли често играју игрице са својом дјецом или самостално у слободно вријеме. Да би се саставила игрица, потребне су одређене техничке способности и осјећај за умјетност и креативност да би се игрица дочарала кроз графику, звучне ефекте и кроз сам *gameplay*. Ми ћемо овде представити алат који ће вам олакшати писање кода за игрицу, а име му је *LibGDX* од *badlogicgames*-а.

Некада прије, требало је вјероватно дубоко разумјевање *low-level* ствари да би се написала и нека основна игрица. Да би игрица била доступна на више платформи, обично се прескакао корак. Данас је све много лакше, а подршка за више платформи долази спонтано и подразумјевано, а то су, може се рећи, главни адути *LibGDX*-а. *LibGDX* је *java framework* који подржава следеће платформе: *Desktop (Linux, Mac OSX, Windows), Mobile (Android, iOS)* и *web (HTML5)*. Од *IDE*-а подржани су *Eclipse*, *Intellij IDEA* и *NetBeans*. Могуће је користити само командну линију за развој. *LibGDX* сваки пројекат концептуално дјели у пет дјелова (пројеката): *core*, *desktop*, *android*, *ios* и *html*. *Core* садржи заједнички кôд, а остали пројекти садрже кôд специфичан за платформу. *LibGDX* посједује генератор пројекта који ће подесити све потребне потпројекте и скинути све додатне либове (енг. *libraries*) који су потребни. Можемо одабрати које платформе желимо да користимо, а у понуди су остале библиотеке које ће да прошире функције *libGDX*-а. Користи *gradle*, што аутоматски омогућава импортовање у све познате *IDE*-ове.

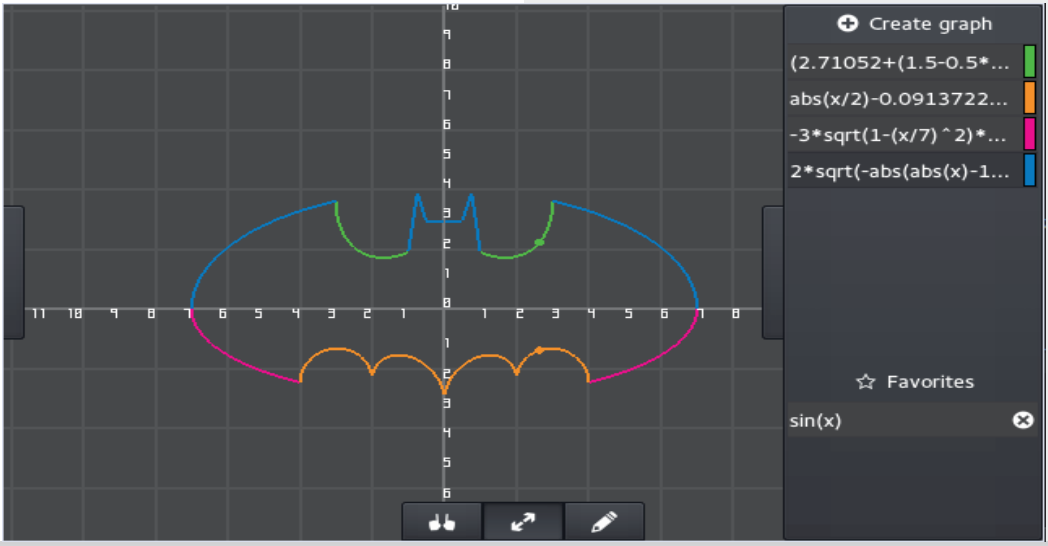


За портабилност *LibGDX*-а вјероватно су највише заслужни *OpenGL* и *Java* који су доступни за различите платформе. Програмер ће кôд логике игре да стави у *core* пројекат, док су остали пројекти генерисани врапери (енг. *wrappers*), а у њима се и не мора много тога мјењати. За *desktop* бисмо поставили подразумјевану резолуцију, за *android* подразумјевану орјентацију, а у *html* пројекат кôд *landing* странице. За све платформе биће вам довољан *Linux OS* за програмирање, осим за *iOS* платформу гдје ће вам бити потребан *Mac* рачунар са *OSX* системом и са *xCode* развојним окружењем. Знајући да се структура система датотека концептуално разликује између *Windows*-а и осталих *UNIX* система, а исто тако се и начини смјештања података на *Android*-у и на *Linux desktop*-у разликују, приказ *LibGDX* почећемо управо о томе како овај *framework* генерише систем датотека. *LibGDX* генерише систем датотека у следећих пет група:

1. *Classpath*: Омогућује приступање датотекама које су упаковане у архиву апликације (нпр. *jar* или *apk*). Приступ овим датотекама је *read-only*.
2. *Internal*: У интерне датотеке спадају оне датотеке чија је путања релативна на *root* путању апликације и радног директоријума за *Desktop* и за датотеке које се налазе унутар *assets*/ директоријума у *Android* пројекту који је укључен у архиву апликације. Интерне датотеке су обично *read-only* ако се односе на датотеке упаковане у архиву.
3. *Local*: Локалне датотеке на *desktop*-у су смјештене у *root* апликације, а за мобилне уређаје унутар приватног директоријума у зависности од система. Препоручено га је користити за чување мањих датотека као што је *game state*.
4. *External*: За *desktop* су смјештене унутар *home* директоријума, док је за мобилне уређаје то *SD* картица. Намјењен је за чување великих датотека, ако има потребе.
5. *Absolute*: Омогућава приступ датотекама на основу апсолутне путање. Овакав начин није препоручљив јер постоји могућност за нарушавање портабилности.

За *html5* доступно је само интерно складиште (енг. *storage*), док се за све остале платформе могу користити сва складишта. Локално и екстерно складиште могу бити физички недоступни на неким платформама, па су ту и функције којима можемо провјерити доступност. Могуће је брисати и мјењати назив датотеке унутар локалног или екстерног складишта или копирати и премјештати датотеку између локалног и екстерног складишта.

Као примјер врапера за систем датотека учитаћемо текстуалну датотеку из интерног складишта која је смјештена унутар *assets* фасцикле у *android* пројекту.



```
FileHandle file =
Gdx.files.internal("myfile.txt");
String text = file.readString();
```

Програм у *LibGDX*, можемо рећи, почиње у класи која наслиједђује *Game*

super класу. Прије тога, покрене се кôд специфичан за платформу, нпр. *main* функција за *desktop* или *onCreate* у *MainActivity*-у за *android*. Најважнија наслиједђена метода је *render()* која се позива у бесконачан *loop*. У њој се позивају функције за рендеровање, а може бити смјештен и кôд логике игрице. У пракси није практично имплементирати читаву игрицу унутар главне *Game* класе, а *libGDX* нам ту помаже јер посједује *interface Screen* који имплементира неку нашу класу која је задужена за одређену уску логику (нпр. један *screen* за главни мени, други *screen* за сам *gameplay*). Интерфејс дефинише већи број метода: *render*, *resize*, *show*, *hide*, *pause*, *resume* и *dispose*. Оне својим именом описују своју намјену. Захваљујући оваквом приступу, лако можемо да смјењујемо логичке цјелине једну за другом. У следећем броју удубићемо се у методе којима можемо да рендерујемо графику у *libgdx*, његове *low-level opengl* враперe и *high-level* класе које су ту да нам олакшају живот. Говорићемо о томе како је *input* класификован кроз платформе и укратко ћемо рећи шта је у понуди за репродуковање звучних ефеката и мелодија. Касније ћемо говорити о могућностима које су интегрисане у *libgdx*, а које нам могу омогућити да направимо што бољи *gameplay*.

Увод у програмски језик *C*

(3. део)

Аутор: Стефан Ножинић

У прошлом делу смо се дотакли типова података, оних променљивих над којима се могу извршити неке операције. У овом делу ћемо дискутовати условном гранању и контролама тока програма и показаћемо неке занимљиве примере.

Условно гранање

Иако смо до сада научили основне ствари о *C*-у, нисмо били у могућности да напишемо неке занимљиве програме. Нисмо могли да напишемопрограм чији ток зависи од вредности неке променљиве, односно од његовог улаза. Прва ствар коју ћемо у овом тексту објаснити, јесте *if*наредба која прима одређени услов и на основу тога да ли је услов испуњен, извршава неки одређен код. Овај услов јесте релацијадве променљиве или променљиве и неке константне вредности. Погледајмо синтаксу *if* наредбе:

```
if (uslov)
{
  /* .... naredbe koje će biti
izvršene ako je uslov ispunjen */
}
```

Као што смо и рекли, услов је релација па су неки примери услова:

- `a == b` - поређење једнакости две променљиве типа ***int***
- `a != b` - тачно у случају да су две променљиве различите вредности
- `a > 0` - тачно ако је променљива `'a'` (типа ***int***) већа од нуле

Исто тако постоје и „додаци” за ову наредбу, а то су *else if* и *else*. Прво се ставља једна *if* наредба, затим неколико *else if* наредби и потом једна *else* наредба. Наредба *else if* задаје други услов у случају да онај пре њега није испуњен, док *else* обухвата све што није обухваћено у горњим случајевима.

Пример: програм који исписује да ли је број позитиван, негативан или нула:

```
#include <stdio.h>

int main()
{
   int n; // broj u koji unosimo
vrednost
   scanf(" %d", &n); // učitavamo
broj
   if (n>0)
   {
     printf("Broj je
```

```
pozitivan\n");
    }
    else if (n < 0)
    {
      printf("Broj je
negativan\n");
    }
    else
    {
      printf("Broj je nula\n");
    }
    return 0;
}
```

Петље

Сада је време да покажемо како се нека одређена секвенца наредби у нашем програму може понављати, односно извршавати у круг.За то служе петље и најчешће се користе *for* и *while* петље.

Петља *for* се користи у случајевима када знамо тачан број понављања и ту имамо једну промењиву као бројач, она добија почетну вредност, услов који треба да буде испуњен како би се петља извршавала у наредбу која се извршава после сваког извршавања петље, односно итерације. Уколико услов потребан за извршавање више није испуњен, петља се зауставља и наставља се даље извршавање програма.

Синтакса *for* петље је:

```
for (inicijalizacija promenljive;
uslov; naredba posle svake
iteracije)
{
   /* ... naredbe u petlji koje
se izvršavaju tokom svake
iteracije ... */
}
```

Ево пример програма који исписује „0 1 2 3 4 5 6 7 8 9”:

```
#include <stdio.h>
int main()
{
   int brojac;
   for (brojac = 0; brojac < 10;
brojac++) // brojac++ je isto što
i brojac = brojac + 1
   {
      printf("%d ", brojac);
   }
   return 0;
}
```

За разлику од *for* петље, *while* петље се извршавају све док је један одређен услов испуњен. Нема бројача па се тако не може унапред знати колико ће се пута одредити, односно то није лако разазнати. Код ове петље се често дешава да се програм „заглави” јер је неки услов стално испуњен па се тако петља непрекидно извршава.

Синтакса:

```
while (uslov)
{
  /* ... naredbe ... */
}
```

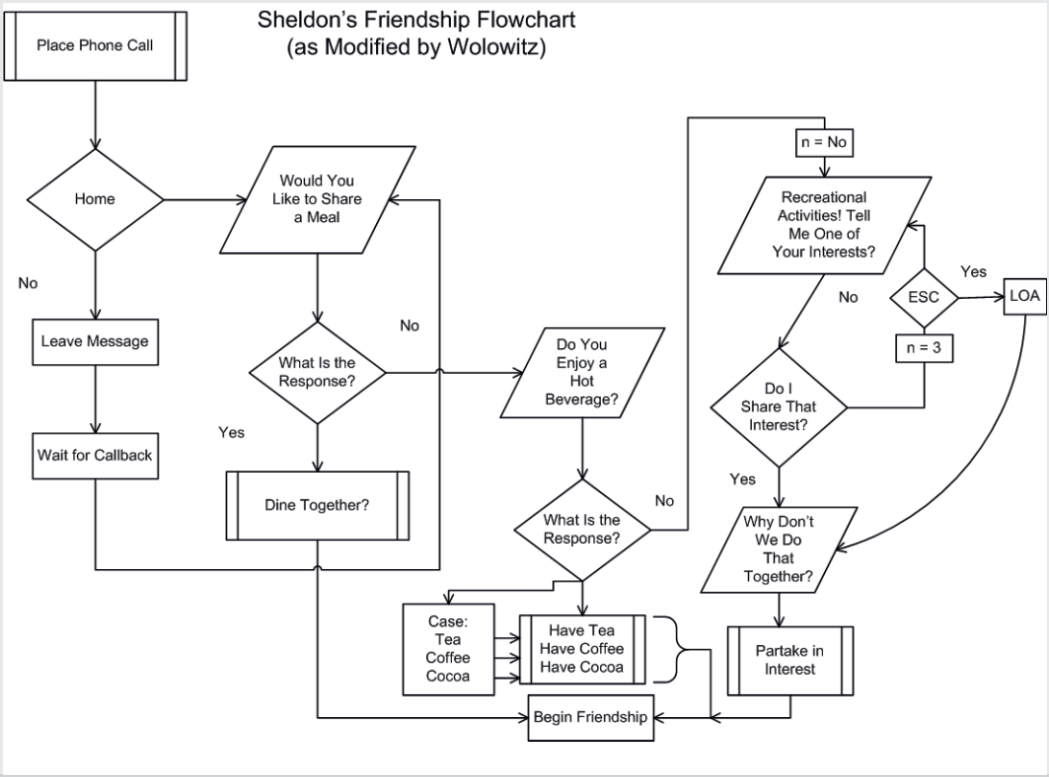
Ево и примера како се горе написани код са *for* петљом могао написати коришћењем *while* петље:

```
#include <stdio.h>

int main()
{
```

```
    int brojac = 0;
    while (brojac < 10)
    {
    printf("%d ", brojac);
    brojac++;
    }
    return 0;
}
```

За крај

За крај овог дела можемо само да вам препоручимо да покушате сами да откуцате нешто и да видите како раде петље јер ћете такоући у „шему”.



Unit tests и JUnit

Аутор: Дејан Чугаљ

Компарацијом индустријске револуције изазване парном машином 18. века (1781, *James Watt*) и инфор-мационих технологија 20. и 21. века примећујемо битну разлику. Неке од основних су неопипљивост и апстракција 21. века - виртуелизација и *cloud*. Иако по питању енергетске скалабилности, употре-бе и искоришћења нисмо далеко одмакли, како каже *Michio Kaku*, „од ТИП*0 цивилизације", (https://www.youtube.com/watch?v=JdILmgJGuvw), некако се може видети та структурна разлика напредака у самој информационој технологији.

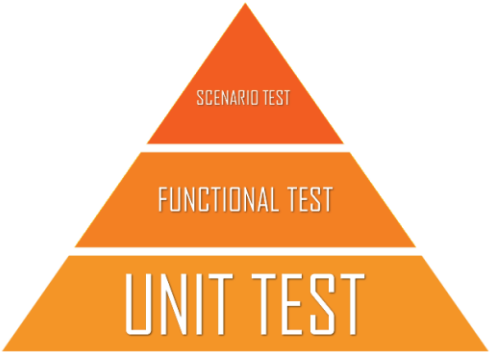
Иако постоји разлика, проблеми испливавају сами по себи. Тренутна ситуација у *IT*-ју је та да влада потпуни раскол, несразмера у развоју хардвера и софтвера који би требало да га прати. Наиме, хардвер је толико напредовао да софтвер никако не може да испрати тај експоненцијални развој хардверских компоненти, тако да су програмери дошли на „мету", и то сагледавајући кроз филозофску призму, стиче се утисак као да људски род још увек није спреман на тако нагли развој.

Да буде још горе, увидели смо још већи проблем који се односи на количину података направљених у јединици времена, глобално гледајући. Некако као да смо остали сами, неприпремљени за све што можемо да створимо. Сви ти подаци који чекају на обраду и који нису процесуирани, морају да прођу кроз неку врсту алгоритма и да буду обрађени, сврстани и постављени зависно од важности у неки *cloud* или медијум за складиштење. Управо ти алгоритми су од изузетне важности јер један погрешан параметар прослеђен алгоритму може да проузрокује тотално неочекиване резултате где би крајњи резултат био складиштење неких медицинских информација у нпр. катастар за земљиште.

Са програмерске тачке гледишта и тачке где човек као индивидуа има удела у целом систему, грешке су неминовне, јер ипак смо само људи, а познато је свима да сваки човек греши док рачунар следи инструкције добијене управо од истих, тих „који греше". Програмерска борба против ових типова грешака води се од самог почетка, почетка комуницирања са рачунаром, од првих програмских језика (1842-1843, *Ada Lovelace*). Једна од данас највише коришћених техника за проверу и најзаступљенијих јесте *Unit testing*, а

пошто ћемо пример имплементирати у *Java* програмском језику, користићемо *JUnit*.

Unit Testing



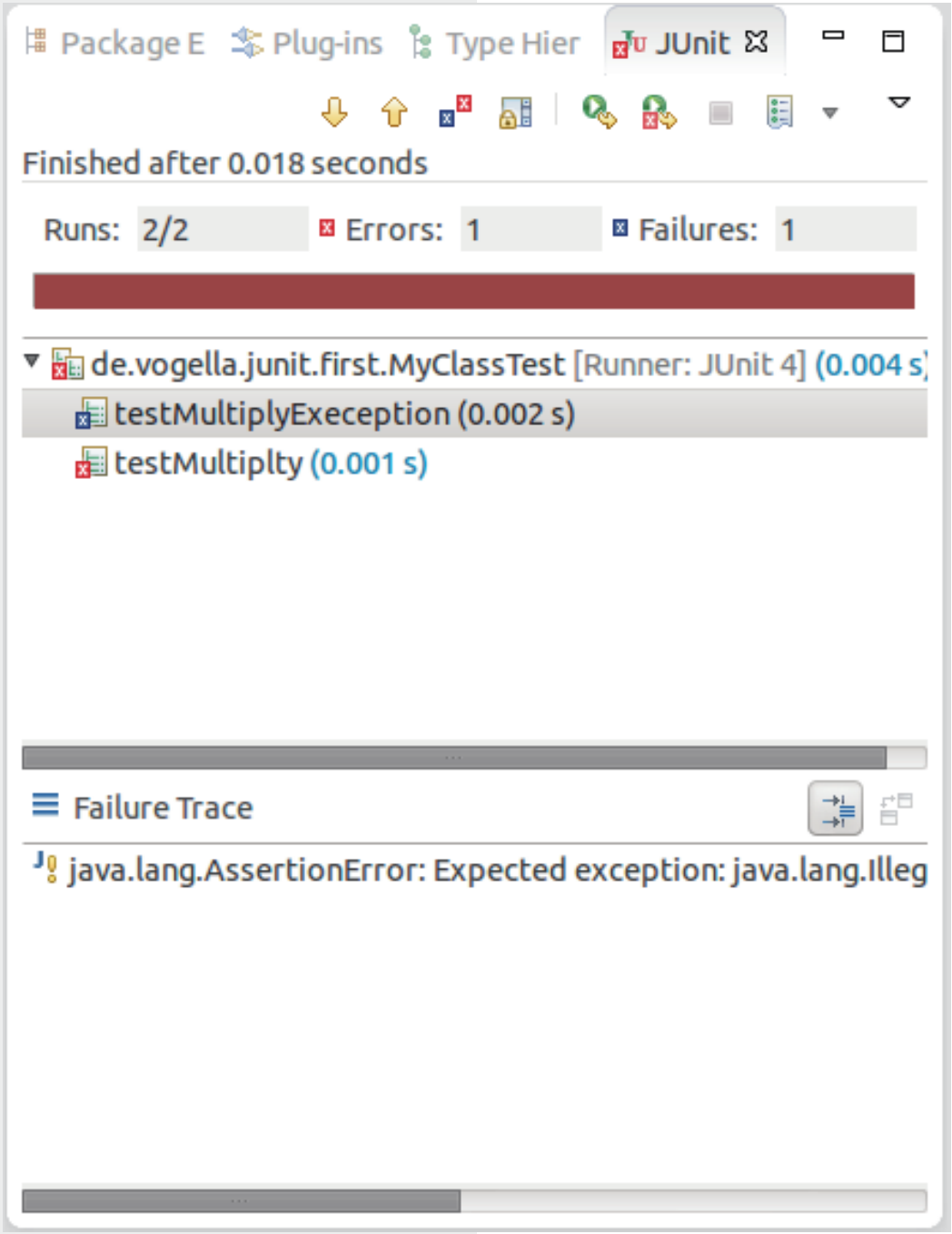
То је део кôда, наменски написан од стране програмера, који извршава специфичан део апликације и проверава добијени (враћени) резултат са сигурно очекиваним подацима или понашањем. (http://www.vogella.com/tutorials/JUnit/article.html)

Лаички речено, током развоја апликације, једна од основних ствари је да се са времена на време написани кôд преводи у машински кôд, извршава и проверава очекиван резултат. Овакво тестирање је углавном и довољно, јер се грешке одмах могу увидети и многи програмери управо то и раде. Иако ова метода узима доста времена, поготово ако се ради о развоју модула неке апликације која има везе са базом података, напишете *query*, покренете, проверите базу и проверите да ли је уписан такозвани „*hard coded*" податак, па наставите. Међутим, замислите ситуацију да свако превођење на машински језик неког модула који развијате, траје и по неколико сати, какав би ваш учинак био у некој компанији (прим.аут.)?

Уколико иоле размишљате на дуже стазе и видите себе као програмера који ради на неким већим пројектима, требало би да нас испратите до краја. У већим софтверским компанијама тестирање је један од круцијалних делова развоја одређеног софтвера. Неки га воле, а неки не, то је чињеница, а док одређени део написане апликације не прође тестове, који опет неки други тим спроводи, написани модул не може никако да уђе у продукционо окружење, ма колико он био добро имплементиран.

JUnit

Kent Beck и *Erich Gamma* увидели су ову важност и написали су *open source framework* 1997. године за *Java*-у познат као *JUnit*. Важност споменутих имена не бисмо овде „проширивали", док више информација о *JUnit framework*-у можете наћи на адреси: (http://www.junit.org).


Package E
Plug-ins
Type Hier
JUnit
Finished after 0.018 seconds
Runs: 2/2
Errors: 1
Failures: 1
de.vogella.junit.first.MyClassTest [Runner: JUnit 4] (0.004 s)
testMultiplyExeception (0.002 s)
testMultiplty (0.001 s)
Failure Trace
java.lang.AssertionError: Expected exception: java.lang.Illeg

JUnit је објављен под *IBM's Common Public License Version 1.0* и постављен је на *GitHub* репозиторијум. Убрзо након објављивања постаје стандард за скоро све језике под називом *xUnit* и подржава: *ASP*, *C++*, *C#*, *Eiffel*, *Delphi*, *Perl*, *PHP*, *Python*, *REBOL*, *Smalltalk* и *Visual Basic*.

Не улазећи у „дубину" могућности овог *framework*-а, пример тестирања ћемо показати малим *Java* примером. Тестирање се имплементира анотацијом одређене методе, класе и потпуно је засебан под-модул-апликативно извршни програм. Замислимо да имамо просту класу *CalculatorTest* и у њој методу сабирања:

```
public class Calculator {
   public double add(double
number1, double number2) {
     return number1 + number2;
   }
}
```

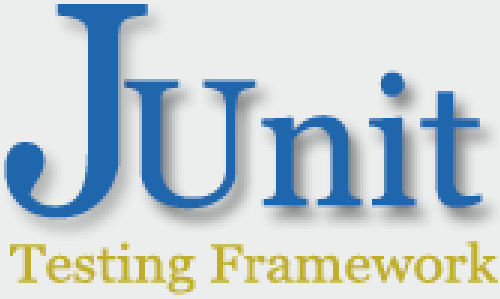
Тестирање *JUnit framework*-ом :

```
import static
org.junit.Assert.*;
import org.junit.Test;

public class CalculatorTest {
   @Test
   public void testAdd() {
     Calculator calculator =
new Calculator();
     double result =
calculator.add(10, 50);
     assertEquals(60, result,
0);
   }
}
```

Можда ово тренутно не изгледа као нека велика помоћ, међутим, развојем апликације и све већим бројем имплементираних модула, ово је од круцијалне важности како за тим, тако и за индивидуалне програмере.

Корисни линкови:

- **Званични сајт:** http://junit.org
- ***GitHub* репозиторијум:** https://github.com/junit-team/



Утицај математике на настанак и темеље рачунарства (1. део)

Увод

Аутор: Недељко Стефановић

Фундаментална математичка истраживања тридесетих година двадесетог века омогућила су настанак рачунара и указала су на границе њихове примењивости, које се ни данас не доводе у питање. Штавише, резултати тих истраживања и данас представљају стуб теоријског рачунарства. Неки од проблема који још увек нису решени, од велике су важности за криптографију и очекује се да њихово решавање доведе до нове епохе у овој науци.

Појам алгоритма и његов развој кроз историју

Наивно схватање појма алгоритма је да је то механички, детерминистички поступак, који полазећи од некаквих улазних података, производи некакав излаз (резултат) у коначном броју корака. Израз „детерминистички" је употребљен у значењу да тај поступак примењен више пута за исте улазне податке, производи увек исти излаз до кога стиже на потпуно исти начин са истим корацима. Међутим, испоставиће се да је употребљено значење израза „механички" много дубље, а упрошћено значи да поступак не захтева размишљање да би се обавио, већ да на основу правила поступка у свакој ситуацији знамо како да наставимо поступак, све док он не буде у потпуности завршен.

Алгоритми се јављају код врло старих народа, који су због потреба премеравања површине земљишта и запремине амбара и судова имали поступке за рачунање површине фигура и запремина тела. Осим тога, у том рачуну су морали некако да представљају бројеве и врше операције са њиховим представљањем, за шта су им опет некакви поступци били неопходни.

Неки поступци који у одређеној мери (мада не потпуно) испуњавају наведене услове, старији су од бројева. Стари пастири нису умели да броје, а имали су потребу да знају да ли су све овце на броју. Било је више поступака сличних овоме. На пример, у празну торбу се

може бацати камење приликом изласка оваца из тора (за сваку овцу по један), а при враћању би се камење вадило из торбе (за сваку овцу по један). Ако у торби остане неки камен, нека овца недостаје.

У старој Грчкој су били откривени алгоритми за решавање квадратне једначине (који имају разгранату структуру, тј. има више случајева који се решавају на различите начине), алгоритам за одређивање простих бројева до неке горње границе (тзв. „Ератостеново сито" које има цикличну структуру, тј. неки кораци се понављају), алгоритми за рачунање највећег заједничког делиоца и најмањег заједничког садржаоца целих бројева или полинома (тзв. „Еуклидов алгоритам"), као и разни други.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Иако су откривани разни алгоритми, прошли су векови док појам алгоритма није уочен као засебан појам и док није добио своје име. То се догодило због једне преводилачке грешке.

Персијски научник ал-Ховаризми (780-850) написао је дело у коме је описао индијски десетични бројни систем (који данас користимо) заједно са поступцима за рачунање у том систему, који се данас уче у нижим разредима основне школе. Као аутор дела се потписао испод наслова, али је његово име збунило преводиоце, који су мислили да се ради о некој њима непознатој арапској речи а не о властитом имену, па су наслов превели као „алгоритми са индијским бројевима". Затим су на основу садржаја покушали да схвате шта су то алгоритми. На тај начин, осим речи „алгоритам", настала је и реч „алгоризам", која означава управо поменуте алгоритме за вршење рачунских операција.

Након тога су у математици откривени многи алгоритми, али је до следећег кључног корака дошло тек тридесетих година двадесетог века. Наиме, савремено схватање појма рачунара је да је то машина која може да изврши програм по било ком алгоритму, тј. да програмски језик, на коме се програмира, обухвата све алгоритме. Математичари такве математичке системе зову потпуним алгоритамским системима. Да би се то остварило, неопходно је свести све алгоритме на коначан број једноставних правила.

Наставиће се.

Ваш посао, *open-source* посао

(2. део)

Аутор: Марко Кажић

Продаја претплате и услуга подршке – продаја стабилности

У прошлом броју смо видели како *open-source* продире у наш лични екосистем софтвера. Видели смо његов утицај на развој данашње *IT* индустрије и обећали смо да ћемо наставити са родитељем свих модела пословања заснованих на *open-source*-у, а то је продаја претплате и услуга. Зачетник саме идеје пословања заснованог на *open-source*-у и један од пионира продаје услуга и едукације, свакако је *Red Hat Inc*. Један од највећих проблема у развоју *open-source*-а је, супротно убеђењу многих, неспособност програмера и фирми да *open-source* наплате. Већи део развоја заснива се на доброчинитељима који развијају бесплатно надајући се понекој донацији.

Продајмо вредност а не производ

Један од најранијих одговора и решења на овај проблем дошао је из *Red Hat*-а и гласио је: „Продајмо вредност а не производ.” *Red Hat* је целокупан свој модел пословања подредио поклањању главног производа - оперативног система, а наплатом свих секундарних услуга, као што су корисничка и техничка подршка, едукација и сертификација. Ипак, сам модел није дошао као последица револуционарног размишљања, већ као потреба да се наплати кôд који није био у власништву *Red Hat*-а. Суштински, кôд је био туђ.

Ипак, генијалност овог модела лежи управо у продаји вредности. *Red Hat* зависи од *Linux*-а, његовог развоја и успеха, и тако *Red Hat* безрезервно помаже његов развој и продорност на тржиште. Тиме омогућава себи приступ новој бази пословних корисника који ће у будућности своје послове мигрирати на сертификоване системе које подржава *Red Hat*, јер пословна архитектура често не оставља простор за експерименте, а често верује само чврстом доказу као што је сертификат. Наравно, уговор о претплати[1] који *Red Hat* склапа са корисницима, уоквирује целу слику система и подршке који добијате као решење. Иако можете и без њега сами да склопите *Red Hat Enterprise Linux* и користите га, концепт је такав да вам се то једноставно не исплати. Претплата такође осигурава дугорочан и сигуран прилив средстава и осигурава сталну везу корисника са *Red Hat*-ом. Окосницу и завршни ударац *Red Hat* задаје помоћу *Red Hat Network*-а, мреже кроз коју пласира ажурирања и безбедносне закрпе и пружа техничку подршку.

[1] *Red Hat Enterprise Agreement EMEA*

Апсолутни владар *Linux Enterprise* света

Око 60% имплементације *Red Hat*-ових производа, уже *RHEL*-а и услуга које из њега произлазе, одлази на сарадњу са великим вендорима и *OEM*-има, као што су *Dell*, *Hewlett-Packard* и *IBM*, док већи део преосталих 40% одлази на велике ентитете који користе *RHEL*. *Red Hat* значајно циља ка пословном свету, сарађује са *OEM*-има на интеграцији својих решења у хардвер и популаризацији своје дистрибуције кроз тај хардвер.

Такође, веома је заступљен и у интерној употреби *OEM*-а, вендора и других *IT* компанија. Сам приступ приватном кориснику је маргинализован јер *Red Hat* управо и циља на *Enterprise* свет. *Red Hat* је овим моделом успео да се удаљи од вендора и *OEM* фирме као што је *Canonical*, који, узгред речено, комбинује готово све моделе пословања са *open-source*-ом. Поуке који оснивачи, предузетници и програмери могу да извуку из овог модела, јесте да продајом вредности и стварањем екосистема око вашег производа можете успешно да наплатите други ниво производа, али и да финансирате даљи развој пројекта. У случају *Red Hat*-а удица за корисника је *Fedora* и управо кроз њу се и гради платформа, повећава њена вредност и пружа полигон за тестирање даљег развоја комерцијалних платформи *Red Hat*-а. Свакако је ово кратка прича и постоји још много примера продаје услуга подршке, едукације и сертификације у *Linux* и *open-source* свету и вероватно ћемо их поменути даље у серијалу. Као што смо већ рекли, поуке овог модела за будући развој су додавање и продавање вредности и коришћење предности отворених платформи на којима се може изградити стабилност коју је релативно лако наплатити у турбулентном свету промена какав је *IT* свет. Надовезујући се на последњу констатацију, у следећем броју причаћемо о моделу који је најпопуларнији данас, а то је *SaaS* модел, и о примени какву има у *open-source* свету. До тада, почните да радите на неком новом *Red Hat*-у, знамо да умете.

OpenSSL:

Сигурност или претња

Аутор: Петар Симовић

Шта је у ствари *OpenSSL* који је проузроковао невиђено зло названо *Heartbleed*?

OpenSSL је *open-source* имплементација криптографског протокола *SSL/TLS* (*SSL - Secure Socket Layer* и његовог наследника *TLS - Transport Layer Security*).Ови протоколи служе за размену симетричног кључа између два рачунара којим се на даље шифрује сва међусобна комуникација, али само током те сесије. За сваку следећу сесију, стари се одбацују и формирају сасвим нови кључеви. Можете видети да ли се користи овакав тип сигурне комуникације по томе што веб-адресе почињу са https:// , уместо са http:// .

Рањивост је откривена првог априла ове године од стране Codenomicon , фирме за веб-сигурност заједно са *Google*-овим сигурносним инжењерима, а публикована је тек седам дана касније, када је изашла и прва закрпа за рањивост. Сама рањивост је у оквиру екстензије зване „*Heartbeat*" за *TLS* протокол која је додата још 2012. године.

Heartbeat, или преведено „откуцај срца", јесте додатак који омогућава одржавање успостављене сигурносне комуникације и у тренуцима када нема активне размене порука између сервера и клијента. Одлика овог додатка је да клијент-корисник шаље поруку са бројем који представља дужину саме поруке серверу на коме је *OpenSSL*, сервер одговара враћањем података из бафера у дужини која му је прослеђена уз саму поруку корисника, што би требало да буде иста порука коју је и добио од истог. Проблем је што нема провере да ли је информација о дужини поруке исправна, тј. да ли је сама порука баш толико дуга.

Овакав проблем се може експлоатисати тако што нападач који се на сервер конектује, може серверу да шаље „откуцај срца” са кратком поруком од неколико бајтова, а да уз то прослеђује информацију да је дужина поруке 64 килобајта. У том случају, сервер прима поруку и информацију о њеној дужини, враћа је нападачу уз произвољне податке до укупне количине од *64kb* (*64kb* - неколико бајтова) из бафера који је резервисан за потребе *OpenSSL*-а. Поставља се питање зашто би нападач желео *64kb* из меморије *OpenSSL*-а. Одговор је зато што баш та меморија може да садржи приватни кључ сервера (*server's private key*), енкрипционе кључеве корисника логованих на сервер, шифре корисника и колачиће корисника (енг. *cookies*). Једном речју, то је све оно што не би смео нападач да зна и због чега и постоје сигурносни енкрипциони протоколи.

Треба напоменути да је катастрофа још већа тиме што се напад може поновити више пута и током сваког напада нападач добија још 64 „златних” килобајта, при томе иза себе не оставља никакве трагове самог упада. Међутим, то није оно најгоре, јер се уз мало среће могао дочепати и вашег приватног кључа и тиме дешифровати сав ваш прошли и будући „шифровани” саобраћај, или је можда могао направити лажни сајт (копију правог) од кога је узео приватни кључ претварајући се на мрежи да је прави помоћу идентификације приватним украденим кључем и тиме је могао да нанесе још већу штету јер може преварити кориснике оригиналног сајта и од њих прикупљати још поверљивих информација. Замислите да је отет идентитет неке банке или неког сервиса за трговину *bitcoin*-ом, или ко зна чега још.

Популарни сервиси угрожени овим *bag*-ом су *LastPass* и *Yahoo mail*, а више о томе на http://goo.gl/59Ct1n и http://goo.gl/og2Pge. Проблем је већи јер се софтвер брзо може закрпити, за разлику од хиљаде других уређаја који су на мрежи и представљају велики проблем јер је неопходна акција корисника истих, а међу њима су и рутери, штампачи, сервери за складиштење (*storage servers*), видео камере, ватрени зидови (*firewalls*) и још многи други.

Међутим, овај проблем се могао избећи омогућавањем *PFS*-а (*Perfect Forward Secrecy*) у оквиру *OpenSSL*-а, што се и препоручује. Он онемогућава злоупотребу добијеног кључа за декриптовање прошлог и будућег шифрованог саобраћаја, а који није подразумевано омогућен, већ мора бити ручно. За оне који желе да омогуће ову опцију на њиховим серверима, препоручујемо да прво прочитају овај документ (http://goo.gl/SkAz5v), а власнике *Apache* и *NGINX* сервера - упутство за омогућавање *PFS*-а који се може наћи на овим странама: http://goo.gl/a5vJVV и http://goo.gl/MGHi50. Проверу да ли је ваш или неки други сајт отпоран на ову рањивост, можете извршити на ова три сајта:

1. http://goo.gl/bOQDdx ,
2. http://goo.gl/LTN5Uz и
3. http://goo.gl/VofjAx.

Добра вест је то што је рањивост само у оквиру *OpenSSL 1.0.1* (*1.0.1f*) и *1.0.2-beta* (*1.0.2-beta1*) верзије и што остале верзије *OpenSSL*-а немају овакав пропуст, па се сматрају безбедним, барем што се *Heartbleed* рањивости тиче, иако је препоручљиво имати увек најновију верзију што је могуће раније, најбоље при самом изласку, као и омогућен *PFS*. У развоју су и алтернативе *OpenSSL*-у, у виду LibreSSL-а и PolarSSL-а, што је одлично јер ће бити већа конкуренција и две трећине интернета неће зависити од једне једине слободне имплементације сигурносног протокола.

LibreSSL

POLARSSL
Straightforward, Secure Communication

Корак до *Google*-а (5. део)

Аутор: Дејан Чугаљ

После паузе која је потрајала, враћамо се имплементацији модула који су нам неопходни за остварење крајњег циља који смо задали себи још у 12. броју ЛиБРЕ! часописа.

Последњи чланак о овој теми објављен је у 15. броју, и због тога ћемо се подсетити шта је све урађено до сада. У 12. броју смо представили неке упоредне податке људске потребе за претраживачима у информационим технологијама и укратко смо споменули *Lucene* библиотеку. Број 13. смо искористили за детаљније упознавање са истом, док смо у броју 14. потпуно ушли у причу и представили студију случаја предстојећег пројекта. У 15. броју ЛиБРЕ! часописа, тј. у четвртом наставку серије чланака „*Lucene* – корак до *Google*-а", почињемо са имплементацијом пројектних модула *Lucene* пројекта. У том издању смо имплементирали прва два модула: „Проналазак свих *PDF* датотека" и „*Tika* екстракција". Такође, овај изузетан *Apache Tika framework* нашао је своје место у засебном чланку и приближио нам своје могућности.

С обзиром на количину кôда који ће бити написан, одлучили смо се за GitHub јавни репозиторијум (Користан линк: https://github.com/libreoss/lucene-moduli у који је постављен целокупан кôд, док оне најбитније делове објављујемо у часопису. Некако, када све саберемо и одузмемо, то је оно што би представило кратку рекапитулацију серијала „Lucene – корак до Google-а".

Пети део серијала нам доноси наставак имплементације модула и то су: „Анализа докумената", „Индексирање *PDF* датотека" и „Индекс (преглед *Lucene* структуре датотека)". Ово је добар моменат да подсетимо да сви модули из студије случаја обојени зеленом бојом јесу тачније они делови где нам *Lucene* пружа своје услуге и довољно је само бити упознат са њеним *API*-јем (енгл. *Application Programming Interface*) да би се могла и користити. Управо тај *API* (http://bit.ly/SBcNKf и документација доступни су на адреси: http://lucene.apache.org/core/4_8_0/index.html (верзија актуелна у току писања

овог чланка је 4.8.0).

Пре него што почнемо, потребно је додати *Lucene* библиотеке у *Eclipse IDE* окружење, које су доступне за преузимање са званичног сајта на адреси: http://lucene.apache.org/core/. Додавање библиотека у пројекат *Eclipse IDE* окружења детаљно је објашњено на адреси: http://bit.ly/1qfFRpb

4. Анализа датотека

Анализа датотека

У првом делу серијала, ако сте пажљиво читали, дотакли смо се токенизације и шта би она укратко требало да представља (12. број ЛиБРЕ! часописа). Управо је ово део где она долази до значаја. Претраживачи не индексирају текст директно, већ се садржај, кандидат за индексирање, растављa на мање делове (*Atomic part*), који се називају *token*-и. Управо се овај процес одвија у овом модулу. Како се ово одвија имплицитно, око овог модула у овом моменту не морамо превише да бринемо, али ако бисмо се удубили у проблематику, видели бисмо да је од великог значаја и да је један од озбиљнијих проблема. Укратко, приликом овог процеса решавају се круцијална питања као што су: да ли је потребно обраћати пажњу на смисао речи (проблем синонима), да ли је потребно приликом претраживања обратити пажњу на семантичку страну природе речи као што су лаптоп и рачунар, да ли је потребно узимати у обзир инфинитив? Проблеми и питања се могу довести до филозофско-филолошког нивоа, тако да бисмо ову тему оставили за неке друге, филолошке науке. Оно што је битно за нас, јесте то да *Lucene* даје низ алата, са којима макар иоле можемо да се приближимо неким реалним резултатима. И коначно, долазимо до најбитнијег дела и сржи овог серијала, а то је „индексирање".

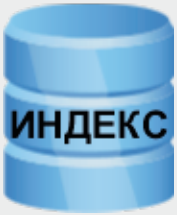
5. Индексирање

Индексирање ПДФ датотека

До сада смо, можда и пречесто, спомињали ову фамозну реч „индексирање", тако да ово заузима централни и најбитнији део целог серијала. Како би хтели да овај део буде респективно, колико је то могуће, приближан квалитету уложеног труда у развијању *Lucene*, овде стајемо и пример дајемо малим „школским примером" имплементације *Lucene* индексирања написаним у *Java* програмском језику. С обзиром да је ово централни (*core*) део серијала, респективно заслужује и место у њему, тако да ћемо га детаљно представити у следећем наставку.

```
import java.io.File;

import org.apache.lucene.analysis .Analyzer;
import org.apache.lucene. analysis.standard.StandardAnalyzer;
import org.apache.lucene. document.Document;
import org.apache.lucene. document.Field;
import org.apache.lucene. document.StringField;
import org.apache.lucene. index.IndexWriter;
import org.apache.lucene.index. IndexWriterConfig;
import org.apache.lucene.store. Directory;
import org.apache.lucene.store. FSDirectory;
import org.apache.lucene.util. Version;

/** * */
public class LuceneIndexExample {
  public static void main(String args[]) throws Exception {
    String text = "Ovo je tekst indeksiran sa Lucene";

    String indexDir = System.getProperty("user.dir")
      + System.getProperty("file. separator") + "index";
    System.out.println(indexDir);
    Directory dir = FSDirectory. open(new File(indexDir));
    Analyzer analyzer = new StandardAnalyzer(Version. LUCENE_48);

    IndexWriterConfig iwc = new
IndexWriterConfig(Version.LUCENE_48, analyzer);

    IndexWriter writer = new IndexWriter(dir, iwc);
    Document document = new Document();

    Field pathField = new StringField("ime_polja", text,
Field.Store.YES);
document.add(pathField);

    writer.addDocument(document);
    writer.close();
  }
}
```

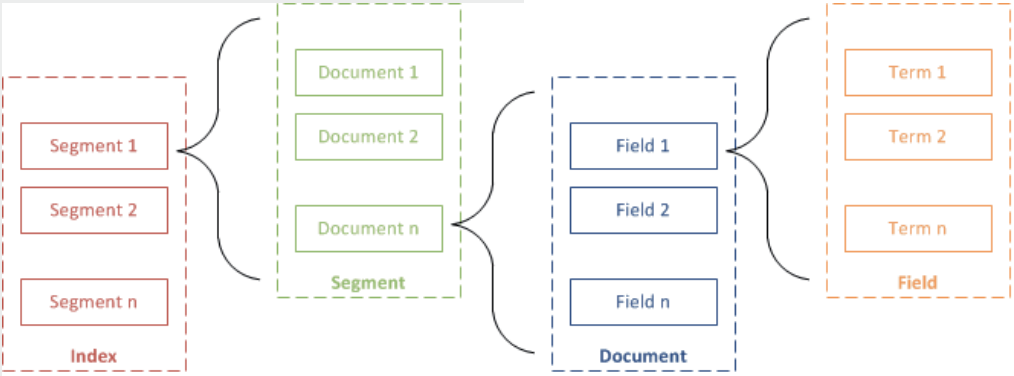
6. Индекс (преглед *Lucene* структуре датотека)



Основни и фундаменталан концепт *Lucene* јесу: индекс (*index*), документ (*document*), поље (*field*) и појам (*term*). Када бисмо то „склопили”, добили бисмо структуру да индекс садржи секвенце документа. Документ је секвенца поља, поље је именована секвенца појма, док је појам секвенца бајтова.

```
-rw-r--r-- 1 root root  1122 Oct  4 20:23 _0.fdt
-rw-r--r-- 1 root root   132 Oct  4 20:23 _0.fdx
-rw-r--r-- 1 root root    32 Oct  4 20:23 _0.fnm
-rw-r--r-- 1 root root 10592 Oct  4 20:23 _0.frq
-rw-r--r-- 1 root root    20 Oct  4 20:23 _0.nrm
-rw-r--r-- 1 root root 32168 Oct  4 20:23 _0.prx
-rw-r--r-- 1 root root   313 Oct  4 20:23 _0.tii
-rw-r--r-- 1 root root 18587 Oct  4 20:23 _0.tis
-rw-r--r-- 1 root root   271 Oct  4 20:23 segments_1
-rw-r--r-- 1 root root    20 Oct  4 20:23 segments.gen
```

Видимо се у следећем броју са нашим „*Lucene* индексима”.



Index Structure

Углавном, цела магија се одвија под „*Lucene* хаубом” и, уколико бисмо желели да уђемо дубље у тематику, сигурно би нам понестало простора. Ово је сасвим довољно за стицање слике и сазнања да после индексирања, *Lucene* прави своју структуру фајлова која се после користи за претрагу.

Корисни линкови:

- ***Званични сајт:*** http://bit.ly/LdDxwN
- **Изворни кôд:** http://bit.ly/1pIFmQo
- **Системски захтеви:** http://bit.ly/1kN0bXP

http://2014.dorscluc.org