Debagovanje sa LLDB-om

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Momir Adžemovic, Miloš Miković, Marko Spasić,
Mladen Dobrašinović

momir.adzemovic@gmail.com, spaskeasm@gmail.com,
milos.mikovicpos@gmail.com, dobrasinovic.mladen@gmail.com

1. april 2020.

Sažetak

Ovaj rad predstavlja grupni projekat u okviru kursa Metologija stručnog i naučnog rada. Ovo je dobra prilika da podelimo sa kolegama naša znanja koja smo stekli ovim istraživanjem koje ima velike primene u praksi. Rad većinom pokriva interesantne informacije o debageru LLDB kao jednom od produžetaka LLVM-a, način korišćenja LLDB i poređenje sa ostalim debagerima.

Sadržaj

1 Uvod **2**

2 Šta je debagovanje **2**
2.1 Bagovi uopšteno 2
2.2 Metode debagovanja 3
2.3 Tehnike za prevenciju bagova 4
2.4 Debager 4

3 Upoznavanje sa LLDB-om **5**
3.1 LLDB interfejs komandne linije 6

4 Gde se on koristi i za koje jezike? **7**

5 Koja razvojna okruženja podržavaju upotrebu ovog debagera i kako? **7**

6 Poredjenje sa drugim popularnim debagerima **9**
6.1 Poređenje: GDB i LLDB 9
6.2 Visual Studio Debugger i LLDB 10

7 Zaključak **10**

Literatura **10**

1 Uvod

U vreme pisanja ovog rada dostupno mnoštvo debagera za jezike C, C++, i Objective-C. Svaki sa svojim specifičnostima koji variraju od platforme do platforme. Najpopularniji izbori debagera za ove programske jezike su LLDB, GDB i Microsoft Visual Studio debager. Nije na prvi pogled očigledno koji je najbolji u zavisnosti od projekta na kom se radi, niti zašto bi neko ko tek počinje svoju karijeru uopšte koristio alat kao što je debager. Nakon pročitanog rada čitalac će biti upoznat sa osnovnim tehnikama debagovanja i specifičnostima debagovanja sa LLDB-om. Za one kojima više odgovara rad u integrisanom razvojnom okruženju poglavlje (5) daje pregled popularnih razvojnih okruženja i na koji način integrišu LLDB. Na kraju su predstavljena poređenja LLDB-a sa drugim debagerima kako bi čitalac lakše mogao da donese odluku da li je LLDB pravi izbor za posao kojim se bavi u zavisnosti od platforme na kojoj radi.

2 Šta je debagovanje

Debagovanje je proces identifikacije pravog problema i njegovo ispravljanje. „Debagovanje je duplo teže od kodiranja, ako napišete kod na najlukaviji (odnosno najkomplikovaniji) način, po definiciji niste dovoljno pametni da ga debagujete." (Brian W. Kernighan) [3]

Koraci pri debagovanju [14]:

1. Uočavanje da postoji greška;
2. Razumevanje greške;
3. Lociranje greške;
4. Ispravljanje greške.

Često je najteži deo ispravno razumevanje i rano otkrivanje greške, kada se greška locira, ispravljanje najčešće nije veliki problem.

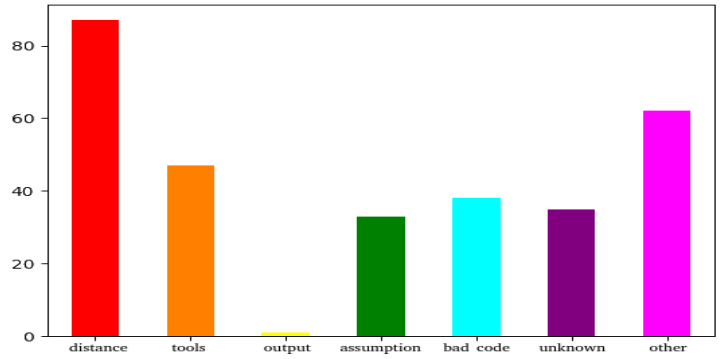
2.1 Bagovi uopšteno

Postojanje grešaka (bagova) se često neopravdano poistovećuje sa propustima u programiranju. U širem kontekstu bag, greška, defekt ili propust se odnosi na bilo koju vrstu problema u bilo kojoj fazi procesa razvoja, kao što su greške u projektovanju, planiranju, arhitekturi, dizajnu. Zato se često termini propust i greška koriste u širem kontekstu razvoja, a termin bag u užem i vezan je za propuste u programiranju.

Jedna od najčešćih klasifikacija bagova prema načinu ispoljavanja obuhvata:

1. **Nekonzistentnosti u korisničkom interfejsu**: često je slučaj da se komanda ctrl+f koristi za pretraživanje dokumenta, Outlook koristi tu komandu za prosleđivanje poruke.
2. **Neispunjena očekivanja**: dobijanje neočekivanog (pogrešnog) rezultata.
3. **Slabe performanse**: stalno ili povremeno čekanje rezultata zbog lošeg odziva sistema, takvi programi su često neupotrebljivi.
4. **Padovi sistema i oštećenja podataka**: predstavljaju najopasniji vid bagova, mogu trajno oštetiti sistem i podatke.

Bagovi su jako neugodni i treba ih sistematski otklanjati čestim refaktorisanjem i planskim građenjem koda. Neke od okolnosti koje pogoduju nastajanju bagova su nedovoljna stručnost razvojnog tima i povećan stres na poslu, a informisanost, sistematičnost i redovnost ih suzbijaju [14].



Slika 1: Glavni razlozi najtežih bagova [2]

Sa slike (1) vidimo da je najčešći razlog za teške bagove upravo rastojanje od izvora greške do neočekivanog ponašanja. U ovim situacijama su veoma bitne metode debagovanja, pa i alati koji se koriste za debagovanje (debageri).

2.2 Metode debagovanja

Programeri iskustvom razviju sopstvene načine kojima intuitivno pristupaju procesu debagovanja koda. U zavisnosti od težine problema, iskustva programera i alata koji mu stoje na raspolaganju metodi debagovanja se mogu grubo podeliti na:

1. **Neformalno debagovanje**: Neformalno debagovanje predstavlja jednostavan i površan pristup i čine ga dva koraka.
   (a) Pokušati sa nekom jednostavnom popravkom;
   (b) Ponavljati korak (a) dok se problem ne reši.

   Ovaj metod se ne preporučuje u praksi i često može da proizvede nove probleme, pogotovo ako vršimo puno sitnih popravki za koje nismo sigurni da će rešiti problem. Ponekad, ako su u pitanju sitne greške, ovaj metod se može oprezno koristiti jer dovodi do brzog rešenja problema [14].

2. **Empirijski naučni metod**: Ovaj postupak je sličan uobičajenom istaživačkom metodu u prirodnim naukama. Čine ga sledeći koraci:
   (a) Posmatrati uočeni problem;
   (b) Postaviti hipoteze o uzroku problema;
   (c) Na osnovu hipoteza napraviti predviđanje ponašanja;
   (d) Eksperimentalno proveriti ispravnost predviđanja;
   (e) Ponavljati prethodne korake uz popravljanje ili menjanje hipoteze, sve dok se ne potvrdi ispravnost hipoteze ili ne ponestanu mogućnosti za njeno dalje unapređivanje.

Uopšteno gledano, ovo je najbolji pristup debagovanju. Često je jako zahtevan i oduzima dosta vremena, ali je sa druge strane je temeljan i koncizan [14].

3. **Heurističko debagovanje**: Ova vrsta debagovanja podrazumeva postojanje heuristike (skup pravila), koja olakšava brže i efikasnije pronalaženje grešaka. Često se za određene skupove problema prave različite heuristike, koje se testiraju u praksi i kasnije koriste kao pravila pri otklanjanju određenih vrsta bagova. Ovakve heuristike odlikuje izbegavanje pravljenja previda pri posmatranju, sužavanje skupa kandidata za iskazivanje hipoteza, usmeravanje posmatranja prema uzroku problema i drugo. Heuristike nisu optimalna rešenja niti egzaktna pravila koja vode rešenju problema, ali često su jako efikasne i daju „dovoljno dobra rešenja" [14].

2.3 Tehnike za prevenciju bagova

Tehnike za prevenciju bagova mogu biti unutrašnje i spoljašnje. Unutrašnje predstavljaju sve ono što se ugrađuje u programski kod samo radi pomoći u prevenciji i otklanjanju grešaka. Neke od njih su pravljenje pretpostavki (eng. asserts), komentarisanje značajnih odluka i mesta u kodu, testiranje jedinica koda. Spoljašnje tehnike i alati se koriste pri razvoju i ne ugrađuju se nužno u programski kod, ali se koriste u čitavom razvojnom ciklusu. Neki od spoljašnjih alata su debager, alati za praćenje verzija programskog koda, alati za podršku i praćenje komunikacije, alati za automatizovanje pravljenja dokumentacije [14].

Jedan od primera unutrašnje prevencije bi bilo korišćenje assert naredbi kao u primeru koda (1). Dereferenciranje NULL pokazivača je nedefinisano ponašanje. To znači da kompajler ima slobodu da uradi bilo šta. Ovakvi bagovi se ne moraju nužno ispoljiti kao momentalno pucanje programa i teško ih je pronaći u velikim projektima. Naredba „assert" osigurava da se program momentalno zaustaviti u slučaju da uđe u nedefinisano stanje i obavestiti korisnika (programera) gde je tačno došlo do greške. Ovo pomaže u ranom uočavanju i lociranju potencionalnog baga što dalje pomaže u njegovom otklanjanju.

```
void example(int* ptr) {
    assert (ptr != NULL);
    printf ("%d\n", (*ptr));
}
```

Kod 1: Primer upotrebe assert naredbe

2.4 Debager

Debager je računarski program koji se koristi za uklanjanje grešaka, testiranje rada i proveru ispravnosti drugih programa. Debageri daju napredne funkcije kao što su pokretanje programa korak po korak (eng. single-stepping), praćenje vrednosti promenljivih kao i stek okvira, praćenje na nivou instrukcija i stanja procesora, zaustavljanje ili pauziranje izvršavanja programa na takozvanim tačkama prekida (eng. breakpoint), a neki čak i omogućavaju menjanje programa tokom izvršavanja.

Većina popularnih debagera daje samo jednostavno okruženje komandne linije (eng. command-line interface), često iz razloga da maksimizuju por-

tabilnost i minimizuju trošenje sistemskih resursa računara. Ipak, popravljanje grešaka u programu preko grafičkog korisničkog okruženja (eng. graphical user interface) debagera se često smatra jednostavnijim, produktivnijim i ugodnijim za rad. Neki debageri pružaju i mogućnosti obrnutog debagovanja (debagovanje unazad) koje omogućava da se vratimo na prethodno stanje programa (step backward). Jedan od debagera koji pruža ovu mogućnost je IntelliTrace koji se koristi u Microsoft-ovom razvojnom okruzenju Visual Studio.

Debagovanje unazad je jako korisno i sve se više koriste debageri koji omogućavaju ovo svojstvo. Mana debagovanja unazad je usporavanje čitavog procesa debagovanja pa čak i do dva puta. Debageri mogu biti zavisni od programskog jezika, ako se mogu koristiti za debagovanje jednog konkretnog jezika, ili mogu biti višejezični i koristiti se za debagovanje više programskih jezika. Neki debageri uključuju i zaštitu memorije kako bi izbegli prekoračenje bafera, ili onemogućili korisniku da pristupa memoriji za koju nema dozvolu i slično [18].

Najčešće korišćeni debageri za C, C++, Objective-C[9][5]:

1. GDB (GNU debager)
2. DDD
3. LLDB
4. Valgrind
5. Nemiver
6. Electric Fence
7. Dbx

3 Upoznavanje sa LLDB-om

LLDB podržava standardne funkcije debagovanja preko komandne linije i može se koristiti kao debager u interaktivnom razvojnom okruženju. Konkretno, sa debagerom pokrenutim nad programom prevedenim sa debag opcijama (eng. debug options) omogućava se [11]:

- Aktiviranje procesa programa sa određenim argumentima komandne linije (eng. command line arguments).
- Korišćenje breakpoint-a (određenog reda ili funkcije u izvornom kodu pri kojima debager zaustavlja izvršavanje programa kada se stigne do odgovarajućeg dela izvršnog koda).
- Korišćenje watchpoint-a (određene promenljive, takve da debager zaustavlja proces ili nit kada se njeno stanje promeni).
- Korišćenje dodatnih uslova nad breakpoint-ovima i watchpoint-ovima.
  - Nastavljanje ili pokretanje programa.
- Pokretanje procesa red po red (sa „ulaženjem” u funkciju ili bez).
- Istraživanje promenljivih ili memorije procesa.
- Izvršavanje proizvoljnog izraza nad stanjem procesa (npr. menjanje neke promenljive na steku).
- Istraživanje steka okvira poziva.
- Izvršavanje drugih naprednih i raznih funkcija.

LLDB omogućava korišćenje eksternih skripti za debagovanje preko javnog API-a za Python, izršavanje proizvoljnog Python koda unutar debagera [20] (preko ugneždenog interpretatora [eng. embedded interpreter]) i omogućavanje REPL (Read-Evaluate-Print-Loop) funkcija za programske jezike zajedno sa mogućnostima debagovanja [6].

3.1 LLDB interfejs komandne linije

LLDB interfejs komandne linije se aktivira pozivom `lldb` komandne linije (eng. shell) sa argumentom koji predstavlja program koji će biti debagovan. Program komandne linije `lldb` se odlikuje struktuisanom sintaksom osnovnih komandi koja je sledećeg oblika [21]:

Primer 3.1

```
<imenica> <glagol> [-opcije [vrednost-opcije]] [argument [argument...]]
```

U ovakvom obliku, imenica se zove i komanda, a glagol potkomanda. Postoje i skraćenice (eng. alias) za komande koje mogu odstupati od ovog oblika. Upravo zato što je ovaj format komandi jako struktuisan mogu biti pogodni skraćeni oblici komandi koji su sličniji onome što je poznato korisnicima drugih debagera [7]. LLDB daje korisnicima više načina da sami definišu komande debagera (najjednostavnija od opcija je komanda `command alias`, koja omogućava jednostavno definisanje sopstvenih skraćenica) [19]. U tabeli (1) su date neke od osnovnih komandi kao primer korišćenja interfejsa i reprezentativni prikaz širokog skupa mogućnosti LLDB-a koji nije naveden u potpunosti u ovom radu. Posebno se ističu komande `help` i `apropos`, koje mogu biti korisne početnicima u korišćenju ovog alata.

Tabela 1: Upotreba interfejsa komandne linije LLDB-a [11][21]

`process launch -- <argumenti>`	Pokreće izabrani program sa datim argumentima.
`thread step-in`	U trenutnoj niti nastavlja izvršavanje sledeće instrukcije izvornog koda programa, ulazeći u pozive funkcija.
`thread step-inst-over`	U trenutnoj niti nastavlja izvršavanje sledeće instrukcije izvršnog koda programa, ne ulazeći u pozive funkcija.
`breakpoint set --file 1.c --line 42`	Postavlja breakpoint na red 42 u izvornom kodu programa 1.c.
`breakpoint list`	Ispisuje postojeće breakpoint-ove debagera.
`breakpoint disable 1`	Deaktivira breakpoint 1.
`apropos <ključna_reč>`	Traži u pomoći za upotrebu komandi (eng. command help) datu ključnu reč.
`help`	Štampa pomoć za komande. (`help` se može koristiti i za nalaženje pomoći za upotrebu potkomandi određene komande [7]).

4 Gde se on koristi i za koje jezike?

LLDB se koristi za debagovanje programa pisanih u programskim jezicima C, Objective-C, i C++. Postoji i verzija za debagovanje programa napisanih u Swift programskom jeziku, tu verziju održava Swift zajednica. Dostupan je na FreeBSD, Linux, macOS, NetBSD, i od 2015 na Windows platformi. Kompletnost skupa funkcionalnosti varira od platforme do platforme[10]:

- FreeBSD - zaostaje za Linux-om, ali brzo napreduje.
- Linux - Približava se kompletnosti funkcionalnosti za debagovanje x86-64, i386, ARM, AArch64, IBM POWER (ppc64), IBM Z (s390x=, i MIPS64 programa.
- macOS - LLDB je sistemski debager na macOS, iOS, tvOS, i watchOS za x86, i386, ARM, i AArch64 debagovanje. Na ovoj platformi ima najbogatiji skup funkcionalnosti koje implementira.
- Windows - I dalje u razvojnoj fazi, ali već koristan za i386 programe.

Skup funkcionalnosti se iz godine u godinu unapređuje i teži se da bude kompletan na svim platformama. Najbolja i najpotpunija podrška je trenutno na Linux i macOS platformama što se može videti iz tabele (2) [10].

Tabela 2: Funkcionalnosti LLDB-a na najpopularnijim platformama

Feature	Linux	macOS	Windows
Backtracking	Yes	Yes	Yes
Breakpoints	Yes	Yes	Yes
C++11	Yes	Yes	Unknown
Commandline tool	Yes	Yes	Yes
Core file debugging	Yes	Yes	Yes
Remote debugging	Yes	Yes	No
Disassembly	Yes	Yes	Yes
Expression evaluation	Yes (Known problems)	Yes	Yes (Known issues)
JIT debugging	Symbolic debugging only	Untested	No
Objective C	N/A	Yes	N/A

Korisnici Windows platformi obično preferiraju alate napravljene od strane Microsoft-a jer su najbolje integrisani sa Windows-om i imaju najbolju podršku na tom operativnom sistemu.

Na Linux i macOS najčešće korišćene funkcionalnosti debagera su implementirane u LLDB-u. Na macOS platformi je LLDB najbolji izbor zato što ima najpotpuniji skup funkcionalnosti u odnosu na druge platforme i održavan je od strane Epla.

5 Koja razvojna okruženja podržavaju upotrebu ovog debagera i kako?

LLDB se može koristiti kao alat komandne linije ili uz neko razvojno okruženje. Neka od popularnih razvojnih okruženja koja imaju mogućnost integracije LLDB-a su Visual Studio Code, Eclipse, CLion, i Xcode 5.

Pošto Epl održava LLDB verziju za svoje operativne sisteme LLDB je podrazumevani debager u Xcode 5 razvojnom okruženju. U daljem tekstu je dat opis načina instalacije na svakom od gore navedenih okruženja i kratak opis koje funkcionalnosti LLDB debagera podržavaju. Za detaljna uputstva pogledati zvanične veb stranice ovih razvojnih okruženja.

Visual Studio Code

Instalacija u Visual Studio Code-u (VSC) se svodi na instaliranje dodataka sa VSC repozitorijuma [17]. Komande se LLDB-u zadaju preko VSC grafičkog korisničkog interfejsa. Podržava:

- Debagovanje na Linux (x64 or ARM), macOS i Windows.
- Uslovni breakpoint-ovi, breakpoint-ovi na funkcijama, watchpoint-ovi.
- Pokretanje iz internog ili eksternog terminala.
- Dissasembly pogled i kretanje instrukciju po instrukciju.
- Python skripte.
- HTML renderovanje za naprednu vizuelizaciju.
- Podrška za Rust programski jezik sa vizuelizacijama za vektor, string i liste.

Eclipse

U Eclipse razvojnom okruženju se korišćenje omogućava instaliranjem Eclipse dodataka [1] koji integriše postojeći LLDB na sistemu sa Eclipse razvojnim okruženjem. Radi na svim platformama koje podržavaju LLDB i Eclipse. Za razliku od VSC-a ima nekoliko ograničenja:

- Debagovanje sa drugog računara nije moguće.
- Core dump debagovanje nije moguće.
- Watchpoint-ovi ne radi.
- Ne može se izmeniti vrednost promenljivih tokom debagovanja.
- Ne može se menjati sadržaj memorije.
- Skoči na liniju, pomeri se na liniju nije implementirano.
- Modules view se ne popuni.

CLion

LLDB dolazi u paketu zajedno sa CLion razvojnim okruženjem na Linux i macOS platformama [8]. Postoji i eksperimentalna verzija LLDB baziranog debagera za MSVC razvojne alate na Windows platformi.

Da bi omogućili korišćenje LLDB-a potrebno je u podešavanjima za dati projekat odabrati postojeći LLDB debager. CLion ne postavlja nikakva ograničenja na LLDB debager kao što je to slučaj kod Eclipse razvojnog okruženja. Dostupne funkcionalnosti variraju od platforme do platforme kao što se može videti u tabeli (2).

Xcode 5

Sa verzijom 5 Xcode razvojnog okruženja LLDB debager je podrazumevani debager u Xcode razvojnom okruženju. LLDB je Eplova zamena za GDB koja je razvijana u koordinaciji sa LLVM-om. Počevši od Xcode 5 svi novi i postojeći projekti se automatski rekonfigurišu tako da koriste LLDB. Dizajniran je tako da korišćenje bude što sličnije GDB debageru kako bi omogućio programerima da se lako prebace sa GDB na LLDB debager. LLDB debager u Xcode 5 ima najbogatiji skup implementiranih funkcionalnosti u odnosu na druge platforme.

6 Poredjenje sa drugim popularnim debagerima

Potrebno je naglasiti da pri poređenju različitih debagera ne može se objektivno odrediti koji je debager najbolji, jer to dosta zavisi od toga koji se operativni sistem koristi, a i od samih preferenci korisnika. Visual Studio Code, jedan od popularnijih editora, koristi LLDB, GDB i VSD za programski jezik C++. Na sledećoj listi se mogu videti debageri za C++ koji se mogu koristiti u okviru Visual Studio Code-a u zavisnosti od toga koji se operativni sistem koristi [15]:

- **Linux**: GDB.
- **macOS**: LLDB or GDB.
- **Windows**: the Visual Studio Windows Debugger or GDB (using Cygwin or MinGW).

Tabela 3: LLDB, GDB, Visual Studio Debugger [4][13][16]

	GBD	LLDB	Visual Studio Debugger
Podrška za programske jezike	C, C++, Objective C	C, C++, Objective C Java, Fortran etc.	C#, C++, Visual Basic, JavaScript etc.
Implementacija	C	C++	C++/C#
Podrška za operativne sisteme	Unix, Windows, macOS	Unix, Windows, macOS	Windows
Razvojni tim	GNU Project	LLVM developer group	Microsoft
Korisnički interfejs	TUI	TUI	GUI

6.1 Poređenje: GDB i LLDB

Debager GDB predstavlja standard za GNU sisteme (ne striktno samo za GNU) [4]. Ako se proverava kvalitet debagera LLDB, onda u potpunosti ima smisla upoređivati ga prvo sa GDB debagerom kao jednim od najpopularnijih debagera. Debager LLDB u debagovanju velikih programa pokazuje bolje performanse od GDB debagera i ima dobar korisnički interfejs [12]. Novije verzije GDB-a podržavaju macOS, ali u proteklih par godina se pretežno koristio LLDB kao glavni debager za macOS. Postoji zvaničan rečnik koji prevodi komande iz GDB u LLDB [11]. Primer:

Tabela (4) predstavlja uzorak iz mape preslikavanja. Vidi se da je način korišćenja ova dva debagera veoma sličan i skup komandi se većinom poklapa.

Tabela 4: Upoređivanje komandi LLDB-a i GDB-a

	GBD	LLDB
Pokretanje procesa:	run	run
Postavljanje argumenata:	set args <args>	settings set target.run-args <args>
Sledeći korak:	step	step
Izlazak iz frejma:	finish	finish

6.2 Visual Studio Debugger i LLDB

Visual Studio Debugger je takođe jedan od poznatijih debagera koji se može uporediti sa LLDB-om. Prednost VSD-a u odnosu na LLDB je u tome što VSD nudi grafički „point-and-click" korisnički interfejs, a prednost LLDB je u broju operativnih sistema za koje ima podršku [16].

7 Zaključak

Jednostavne greške koje se prostiru nekoliko linija se možda i mogu ispraviti bez pomoći spoljnih alata, ali složenije greške koje obuhvataju više komponenti mngo teže, tako da je debager jedan od obaveznih alata kojim bi svaki programer trebalo da vlada. Debagovanje je svakodnevnica svakog programera i dobar debager kao što je LLDB može značajno smanjiti vreme provedeno u debagovanju, ostavljajući više vremena za razvoj. LLDB nastavlja da se unapređuje iz godine u godinu i već u dobrom delu prevazilazi svoje konkurente.

LLDB je podrazumevani debager u XCode5 tako da ovde praktično nema dodatnih koraka podešavanja. Najlakša integracija sa razvojnim okruženjem je trenutno u CLion-u gde LLDB dolazi zajedno u paketu sa ovim razvojnim okruženjem i ne zahteva nikakva dodatna podešavanja, međutim CLion je komercijalni proizvod koji se plaća. Alternativni način koji takođe zahteva minimalno podešavanja je dodatak Visual Studio Code editoru koji pruža GUI za debagovanje LLDB-om. Na Windows platformi je trenutno najbolje koristiti Microsoft-ov debager jer je najbolje integrisan sa Visual Studio-om, ali se očekuje da će u bliskoj budućnosti kvalitet LLDB na Windows platformama dostići nivo da će moći da parira Microsoft-ovom. Ukoliko je projekat za macOS platformu onda LLDB zasigurno pravi izbor jer je podrazumevani debager za ovu platformu i odlično je integrisan sa Xcode 5 razvojnim okruženjem. Za Linux korisnike LLDB nudi mnoštvo funkcionalnosti koje ne postoje u GDB-u kao i poboljšane performanse, tako da ako projekat to dozvoljava LLDB je pravi izbor i na Linux platformi.

Uzimajući u obzir da svaka promena svakodnevnih navika, pa i onih koji se tiču jednog alata koji se koristi svaki dan kao što je debager, LLDB je vredan inicijalnog truda koji treba uložiti da bi se zamenio prethodno korišćeni debager, a ako je LLDB prvi debager sa kojim se programer susretne tim bolje.

Literatura

[1] Eclipse. Eclipse Plugin, 2020. at: https://wiki.eclipse.org/CDT/User/FAQ#How_do_I_get_the_LLDB_debugger.3F.

[2] Michael Perscheid et al. Studying the Advancement in Debugging Practice of Professional Software Developers. page 21, 2016.

[3] Vladimir Filipović. Debagovanje, 2016. at: http://poincare.matf.bg.ac.rs/~vladaf/Courses/Matf RS2/Prezentacije/CC 23 - Debagiranje - Goran Vinterhalter.pdf.

[4] Free Software Foundation. GDB: The GNU Project Debugger, 2020. at: https://www.gnu.org/software/gdb/.

[5] Rubaiat Hossain. Best Linux Debuggers for Modern Software Engineers, 2020. at: https://www.ubuntupit.com/best-linux-debuggers-for-modern-software-engineers/.

[6] Apple Inc. REPL and Debugger, 2020. at: https://swift.org/lldb/#why-combine-the-repl-and-debugger.

[7] Apple Inc. Understanding LLDB Command Syntax, 2020. at: https://developer.apple.com/library/archive/documentation/General/Conceptual/lldb-guide/chapters/C2-Understanding-LLDB-Command-Syntax.html.

[8] JetBrains. Clion LLDB, 2020. at: https://www.jetbrains.com/help/clion/configuring-debugger-options.html.

[9] Linux Links. Best Free Linux Debuggers, 2020. at: https://www.linuxlinks.com/debuggers/.

[10] LLDB. LLDB status, 2020. at: https://lldb.llvm.org/status/status.html.

[11] LLVM. GDB to LLDB command map, 2020. at: https://lldb.llvm.org/use/map.html.

[12] LLVM. New "lldb" debugger, 2020. at: http://blog.llvm.org/2010/06/new-lldb-debugger.html.

[13] LLVM. The LLDB Debugger, 2020. at: https://lldb.llvm.org.

[14] Saša Malkov. Debagovanje, 2019. at: http://poincare.matf.bg.ac.rs/~smalkov/files/rs.r290.2019/public/Predavanja/Razvoj softvera.08.2019 - Debagovanje.p4.pdf.

[15] Microsoft. Visual Studio Code Suport, 2019. at: https://code.visualstudio.com/docs/cpp/cpp-debug.

[16] Microsoft. Visual Studio Debugger Suport, 2019. at: https://docs.microsoft.com/en-us/visualstudio/debugger/debugger-feature-tour?view=vs-2019.

[17] Microsoft. Visual Studio Code LLDB plugin, 2020. at: https://marketplace.visualstudio.com/items?itemName=vadimcn.vscode-lldb.

[18] Margaret Rouse. Debagovanje, 2020. at: https://searchsoftwarequality.techtarget.com/definition/debugging.

[19] Derek Selander. *Advanced Apple Debugging & Reverse Engineering, 2nd edition*. 2017.

[20] The LLDB Team. Python Scripting, 2020. at: https://lldb.llvm.org/use/python.html.

[21] The LLDB Team. Tutorial, 2020. at: https://lldb.llvm.org/use/tutorial.html.