Упоредни преглед дебагера

Семинарски рад у оквиру курса
Верификација софтвера
Математички факултет

Оливера Поповић, 1029/2020
o.popovic@outlook.com

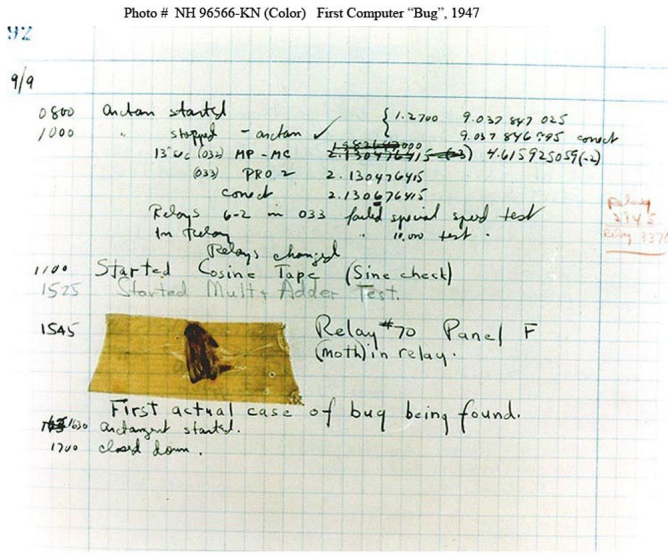
20. јануар 2021.

Сажетак

Како је прављење багова, односно грешака и недостатака система, неизоставни део процеса програмирања, веома је битно постојање алата за њихово проналажење и уклањање. Дебагери су управо то, сложени софтверски алати који пружају помоћ у решавању и анализи оваквих проблема. У овом раду читалац ће имати прилику да се упозна са основим могућностима модерних дебагера, као и са неколико одабраних дебагера за различите системе, језике и платформе. Изложене су специфичности сваког од њих, а приказан је и сумирани преглед њихових функционалности и могућности.

Кључне речи — дебаговање, дебагери, могућности дебагера, поређење дебагера, специфичности дебагера

Садржај



Слика 1: Прва светска буба (eng. *bug*) у рачунару

1 Uvod

9. септембра 1947. године, тим информатичара са универзитета Харвард пријавио је први светски **баг** (eng. *bug*) у рачунару [12]. Термин баг данас се користи да означи грешку или недостатак система. Ипак, овај први баг био је баг у пуном значењу речи, што преведено са енглеског језика значи буба. Наиме, тим са Харварда приметио је да њихов рачунар, који је био познат под називом *Mark II*, константно пријављује необичне грешке. Након што су отворили рачунар, унутра су као узрок грешкама пронашли заглављеног мољца, који је пореметио електронику рачунара. Први светски баг у рачунару приказан је на слици 1. Ипак, ово није био први баг пријављен у смислу техничке грешке. Томас Едисон пријавио је багове у својим дизајнима још у 19. веку. Ово је, дакле, једино био први светски баг у рачунару. Чак је и Грејс Хопер, чланица поменутог тима којој се најчешће приписују заслуге за откривање овог термина, изјавила у једном свом интервјуу да она није створила тај термин, већ да се он уклопио у терминологију која је већ била у употреби [12].

Претходно поменути информатичари били су први који су заправо дебаговали рачунар. **Дебаговање** је процес проналажења и исправљања познатих, али и потенцијалних багова у софтверу ради добијања очекиваних резултата. Да би процес дебаговања могао бити успешан, неопходно је добро познавање пројекта и имплементационог програмског језика, технологија и оперативног система на ком се извршава програм. За отклањање неких посебно дубоко скривених грешака понекад је чак потребно и познавање платформе или процесора који се користи [21].

„Ако је дебаговање процес отклањања софтверских грешака, онда програмирање мора бити процес нихового стварања"

-Едзгер Дејкстра

Програми се могу дебаговати коришћењем за то посебно направљених алата који се зову **дебагери**. Ови алати представљају комплексан софтвер који захтева подршку компајлера, асемблера, линкера, као и оперативног система да би исправно радили [5]. Користе се онда када програм током извршавања не производи очекивано понашање, а разлог за то није одмах јасан. Дебагери се такође некада користе и да се програм интерактивно извршава, како би се лакше разумео код који се посматра.

2 Шта могу дебагери?

Дебагери се у процесу развоја софтвера користе од самог почетка, када је програм још у фази дизајна и када постоји мало написаног кода, па све до тренутка када се софтвер испоручује корисницима [5]. Чак и након тога, дебагери се користе од стране програмера који одржавају и унапређују систем, како би они разумели све специфичности програмског кода. Како је листа употребе веома дугачка, неопходно је да дебагери пруже брз и ефикасан рад, али и једноставну употребу.

Главна функционалност дебагера је контрола тока извршавања програма који се дебагује. Најчешће коришћене опције које нуди дебагер за то су постављање **тачака прекида** (eng. *breakpoints*) и извршавање програма до тих тачака прекида, као и **извршавање корак по корак** (eng. *single stepping*) на нивоу наредбе, функције или инструкције [5].

Тачке прекида подржане су тако што се у део кода који треба да се изврши умеће посебан код (инструкција прекида или нека друга инструкција која није валидна) [13]. Када се приликом извршавања дође до тог дела кода долази до прекида, који се пријављује дебагеру посредством процесора или оперативног система. Како би то било могуће, већина модерних процесора и оперативних система поседује уграђене могућности које дебагери користе. Тачке прекида могу бити и такозване условне тачке прекида (eng. *conditional breakpoints*). Оне омогућавају да се извршавање прекине уколико је дефинисани израз евалуиран као тачан.

Функционалности које се подразумевају на модерним дебагерима су праћење вредности променљивих (бафера, параметара, поља) и стања на стеку приликом извршавања, као и праћење стека позива функција. Још неке занимљиве могућности модерних дебагера су:

- Праћење вредности које се налазе у регистрима
- Евалуација разних израза
- Мењање вредности променљивих у току извршавања - што је корисно јер се након измене извршавање наставља, што штеди време јер је иначе потребан циклус измена-компилација

- Могућност наставка извршавања са друге локације, како би се, на пример, избегла грешка
- Праћење трансакција
- Праћење животног века (креирања и уништавања) динамички алоцираних инстанци
- Праћење изузетака
- Постављање тачке посматрања (eng. *watchpoint*) - праћење вредности на одређеној меморијској локацији уз могућност заустављања при свакој промени
- Дебаговање у вишенитним процесима
- Дебаговање оптимизованог кода

Дебагери подржавају и неколико различитих техника дебаговања, у које спадају [13]:

- Интерактивно дебаговање је техника која обухвата комуникацију са дебагером, односно постављање тачака прекида, извршавање, паузирање и сличне опције за контролу тока програма
- Обрнуто дебаговање је техника која омогућава програмеру да сачува све активности програма (сваки меморијски приступ, израчунавање...) и да онда унатраг испита стање програма
- Удаљено дебаговање је поступак дебаговања програма који се извршава на систему који је удаљен од самог дебагера, те он за повезивање користи мрежу
- Посмртно дебаговање (eng. *post-mortem debugging*) је поступак дебаговања након што се десио прекид програма

3 Преглед дебагера

Модерни алати за развој софтвера попут развојних окружења често нуде могућност да се комуникација са дебагером врши кроз графички кориснички интерфејс развојног окружења или имају дебагер директно интегрисан у окружење. Постоје и варијанте у којима разни текстуални едитори (Visual Studio Code, Vim, Emacs...) који подржавају прикључке[1] (енг. plugin) омогућавају повезивање са разним дебагерима докле год постоји адекватни прикључак који испуњава интерфејс наметнут од стране едитора.

Треба истаћи да се услед овога термин дебагер у рачунарству често користи да означи и комплетан софтверски пакет интегрисан у развојно окружење који укључује графички кориснички интерфејс, систем за комуникацију са дебагером и слично, али користи се и да означи дебагер који ови претходни системи користе да реализују свој рад. Некада и сам тај систем за дебаговање подржава и коришћење из графичког окружења. Дакле, важно је истаћи да постоји вишезначност у употреби овог термина. На пример развојно окружење `QtCreator`

[1] Софтвер који се интегрише у постојећи софтвер са наменом да прошири или обогати његове функционалности.

Слика 2: Пример постављања тачке прекида у развојном окружењу *Visual Studio Code*

користи дебагер `GDB` (eng. GNU Debugger) да би кориснику пружио функционалност дебаговања. У овој ситуацији правилније је рећи да `QtCreator` користи `GDB` дебагер уместо да `QtCreator` има дебагер, али ни једно ни друго нису погрешни.

У даљем тексту ће бити изложени и анализирани неки од познатијих дебагера. У њиховом опису ће такође бити и посвећена пажња претходно споменутој вишезначности поводом употребе термина уколико за тим буде било потребе.

Интегрисани дебагери визуелно су јаснији и једноставнији за употребу из угла корисника. За постављање тачака прекида користи се развојно окружења (пример постављања дат је на слици 2) и у тренутку када програм стане са извршавањем, контрола се враћа у едитор на место које је изазвало прекид рада. Посебно су згодни за кретање корак по корак кроз код, јер корисник може то кретање визуелно да испрати у прозору који садржи изворни код програма. Још једна погодност оваквих дебагера је то што су структуре података компајлера доступне дебагеру, као што је на пример таблица симбола. Такође, уколико је потребна евалуација неких израза то ће бити изведено коришћењем истог компајлера који је и генерисао код, што чини програм конзистентним [5].

Са друге стране, коришћење дебагера путем командне линије такође има своје предности. На пример, уколико је потребно нешто брзо проверити у коду, покретање неког развојног окружења било би исувише захтевно, па је употреба оваквих алата који се покрећу чак и неколико пута брже веома значајна. Још једна веома важна употреба је за дебаговање које се врши на некој удаљеној машини путем командне линије. У таквим ситуацијама графички кориснички интерфејс можда уопште није доступан, или ако јесте може бити веома

спор, па је употреба кроз командну линију неопходна [15]. Дебагери који омогућавају овакав начин рада су такође и погодни за интеграцију у друге скрипте и алате попут *Bash* скрипти.

Данас постоји мноштво дебагера за различите програмске језике и платформе, како комерцијалних тако и бесплатних, при чему неки тврда да могу да препознају на стотине различитих проблема. Неколико одабраних познатих, али и мање познатих дебагера биће изложено у наставку. Редослед излагања дебагера пратиће учесталост појављивања појединачних дебагера у референтним изворима.

3.1 GDB

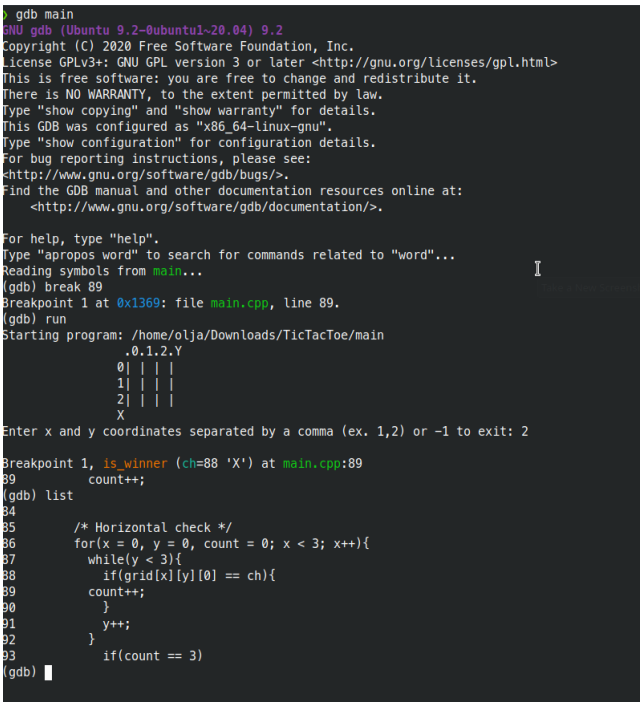
GNU дебагер (GDB) представља један од најчешћих коришћених алата за дебаговање међу програмерима на *Unix* системима [15]. Има дугу традицију и оригинална верзија датира још из 1986. године. *GDB* пружа интерфејс за коришћење из командне линије и подржава језике *C, C++, D, Fortran, Pascal, Modula-2* и *Ada* [11]. Од поменутих језика данас су најчешће у употреби C, C++ и D па је и *GDB* типично најчешће асоциран као дебагер за те језике.

GDB је имплементиран у програмском језику *C* и поседује *GPLv3* лиценцу, тако да припада групи програма отвореног кода. На већини *Unix* система долази или већ инсталиран, или је доступан као системски пакет који се једноставно може преузети и инсталирати. Пошто је у питању софтвер отвореног кода, доступан је и на другим платформама као што су *Windows*, *MacOS* и *Android*. Управо захваљујући томе, програм који се дебагује коришћењем овог алата може се извршавати на истој машини као и сам дебагер, а може се и извршавати и на некој удаљеној машини.

Приликом коришћења *GNU* дебагера постоји неколико начина за паузирање извршавања програма [15]. То су постављање тачака прекида као и условних тачака прекида, постављање тачака посматрања, али и постављање тачака хватања. Тачке хватања (eng. *catchpoint*) користе се да означе да је потребно паузирати извршавање када се деси неки очекивани догађај. Пример коришћења овог дебагера кроз командну линију дат је на слици 3.

Иако овај алат не поседује сопствени графички кориснички интерфејс, за њега је развијено неколико програма који нуде управо то. Један од најпознатијих је ***DDD*** (eng. *Data Display Debugger*) [15]. Поред *GNU* дебагера, подржава још неколико дебагера међу којима су *DBX, WDB, Ladebug, JDB, XDB, bashdb* и *pydb* [9]. Приказ екрана изгледа овог програма дат је на слици 4. Овај алат може се користити примарно на *UNIX* системима. Постоји још много графичких корисничких интерфејса који су развијени за *GDB*, међу којима су ***KDbg*** (намењен системима који користе *KDE* графичко окружење), затим ***Nemiver*** и ***gdbgui***.

Као што је већ поменуто, дебагери се често користе као део већих интегрисаних развојних окружења (eng *IDE*), па је тако *GDB* такође део неколико њих, међу којима су *CLion, Eclipse, Qt Creator, Xcode,*



Слика 3: Пример коришћења *GNU* дебагера

Dev-C++, Lazarus [10].

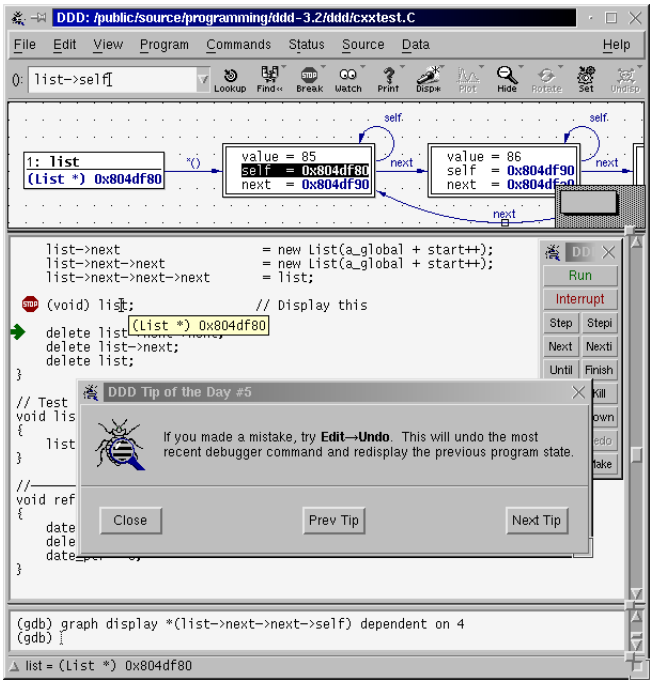
3.2 LLDB

LLVM је пројекат отвореног кода који се развија у циљу пружања алата за развој софтвера као што су компилатори, алати за анализу кода, дебагери, профајлери и линкери и слично [14].

LLDB је модерни дебагер који је изграђен од скупа компоненти које у великој мери користе библиотеке овог пројекта, као што је на пример *Clang* предњи део компајлера за програмске језике из *C* породице језика [22]. *LLDB* има подршку за коришћење из командне линије. Чињеница да користи *Clang* инфраструктуру му омогућава да подржава најновије верзије програмских језика *C, C++, Objective-C* и *Objective-C++*.

LLDB је имплементиран у програмском језику *C++* и поседује *Apache 2.0* лиценцу што га, као и *GDB*, убраја у групу програма отвореног кода. У тренутку писања рада, документација наводи постојање подршке за платформе *Linux, Windows, macOS, iOS, tvOS, watchOS* и *FreeBSD* [22]. Овај алат такође нуди могућност извршавања на удаљеним машинама, односно на машинама које се разликују од оне на којој се извршава сам дебагер.

Неке од занимљивих особина овог алата су могућност дебаговања



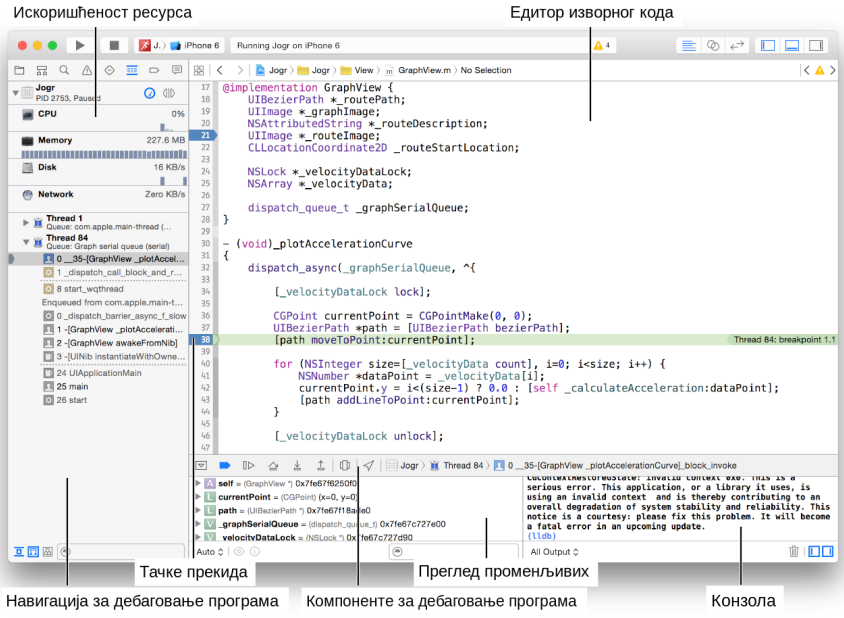
Слика 4: Изглед графичког корисничког окружења *DDD*

уназад, постављање тачака посматрања, инспекција стања и вредности променљивих и још много других. Такође, *LLDB* покушава да користи *JIT* технике за евалуацију израза увек када је то могуће. *JIT* компилација је начин извршавања кода која укључује компилацију током извршавања, а не пре извршавања [8, Chapter 10]. Коришћењем алата *SWIG* [6], који се користи за повезивање програма или библиотека написаних на језицима *C* или *C++* са скрипт језицима, програмском језику *Python* омогућен је приступ и контрола јавног апликационог програмског интерфејса библиотеке за дебаговање [22].

Овај дебагер користи се као подразумевани у званичном развојном окружењу платформе *macOS* које се зове *Xcode*. Пример употребе овог окружења дат је на слици 5. *Android Studio* такође користи *LLDB* за дебаговање [2]. *LLDB* се може користити и кроз многа друга интегрисана развојна окружења и едиторе, међу којима су *Visual Studio Code, Eclipse* и *CLion*. За *Visual Studio Code* постоји прикључак под називом *CodeLLDB*.

3.3 Microsoft Visual Studio Debugger - MVSD

MVSD је дебагер доступан за коришћење из развојног окружења *Visual Studio*. Како је у питању комерцијални софтверски производ, није познато превише детаља око интерне имплементације и архитектуре у односу на развојно окружење. *MVSD* је апстрахован кроз *Visual Studio* и ретко се посматра као одвојени систем или алат из угла кори-



Слика 5: Изглед интегрисаног графичког корисничког окружења *Xcode*

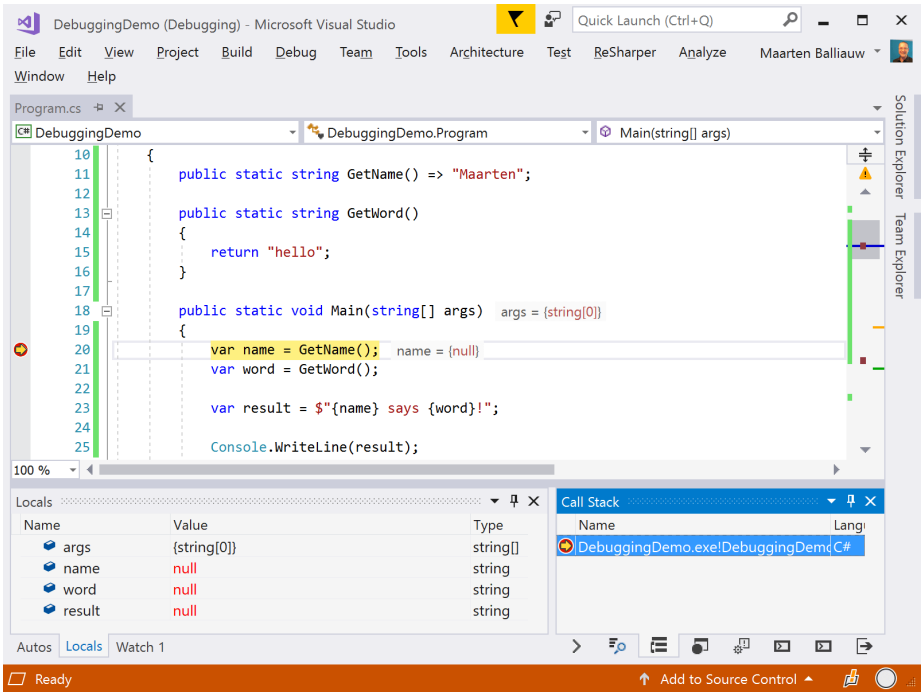
сника. Односно, на њега се више посматра као функционалност коју пружа развојно окружење и није доступан за коришћење из других алата [18]. На слици 6 је приказан пример употребе овог дебагера.

Подржане су практично све најчешће функционалности дебагера као што су условне тачке прекида, праћење тока извршавања програма, приказ садржаја променљивих, извршавање корак по корак, евалуација израза, позивање функција приликом дебаговања, постављање тачака посматрања, повезивање на процес у извршавању и слично [18].

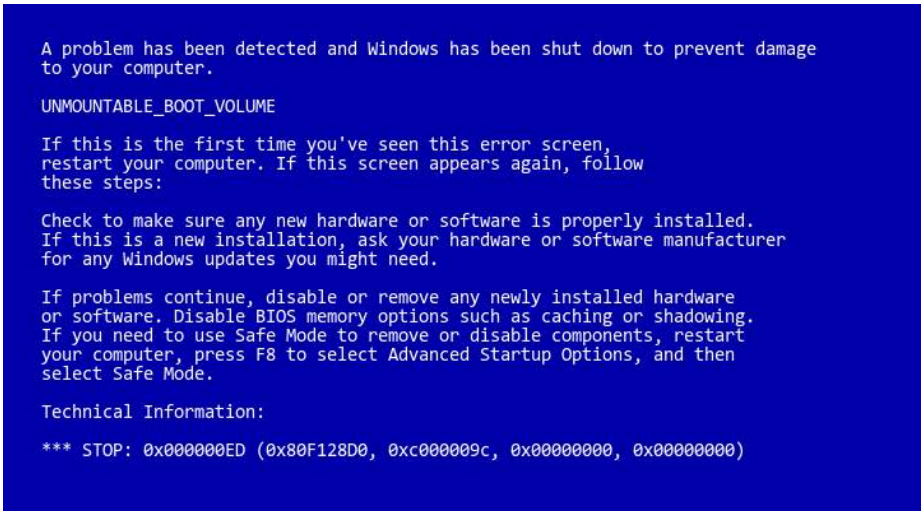
3.4 WinDBG

WinDBG је дебагер који развија компанија *Microsoft* за платформу *Windows*. Његове најчешће намене јесу дебаговање процеса на страни кернела као и драјвера. Постоји и програм са истим именом који пружа графички кориснички интерфејс за коришћење овог дебагера. У пракси је ово дебагер који се користи за дебаговање проблема који се популарно зову плави екран смрти (eng. Blue Screen of Death - BSOD) и представљају ситуације у којима оперативни систем *Windows* престане са радом услед озбиљне грешке. Пример оваквог проблема дат је на слици 7. Постоји и подршка за проширивање функционалности овог дебагера путем прикључака који се повезују са алатом као библиотеке са динамичким повезивањем (eng. Dynamic Linkable Library - DLL) [17].

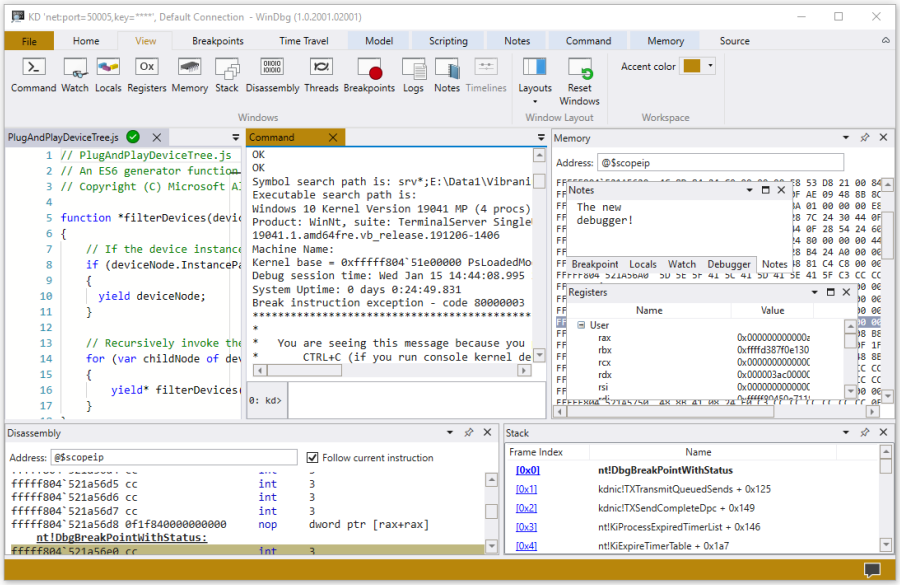
Једна од разлика у односу на *MSVD* јесте у томе што је овај алат



Слика 6: Пример коришћења *Visual Studio* дебагера



Слика 7: *Blue Screen of Death* - екрански приказ након фаталне системске грешке на *Microsoft Windows* оперативним системима



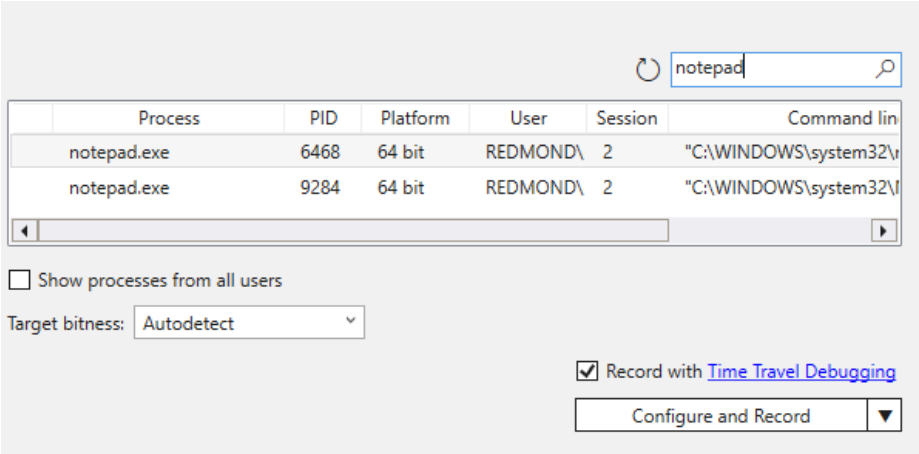
Слика 8: Пример коришћења *WinDBG* дебагера

само дебагер који није директно интегрисан у развојно окружење, иако наравно, постоји подршка за његово коришћење и у оквиру *Visual Studio* алата. Услед тога, алат заузима драстично мање меморије на диску и брже и лакше се покреће у односу на целокупан систем који представља *Visual Studio*. Постоје и функционалности које *WinDBG* поседује које нису доступне кроз *MSVD* дебагер. Више о овим разликама и функционалностима је доступно у табели 2.

Компанија *Microsoft* је 2017. године објавила нову верзију овог дебагера под називом *WinDBG Preview* која је донела побољшања за графички кориснички интерфејс и пре свега подршку за дебаговање са такозваним *путовањем кроз време* (eng. Time Travel Debugging - TDD). Основна идеја овакве врсте дебаговања је у томе да се приликом извршавања процеса сачувају све неопходне информације како би се после тај процес могао дебаговати унапред и уназад [4]. Ово може бити посебно корисно за дебаговање програма пре него што је дошло до његовог пуцања када у класичном дебаговању процес више није доступан за дебаговање [16]. На слици 8 је приказан алат *WinDBG Preview*, а на слици 9 је приказана слика на којој се види како се може покренути програм са укљученом подршком за ТДД дебаговање кроз алат *WinDBG Preview*.

3.5 Java debugger - JDB

Јава debager (eng. *Java debugger*), најчешће скраћено само *JDB*, је алат командне линије за дебаговање програма имплементираних у језику Јава. Еквивалент је *GDB* дебагеру за Јава виртуелну машину



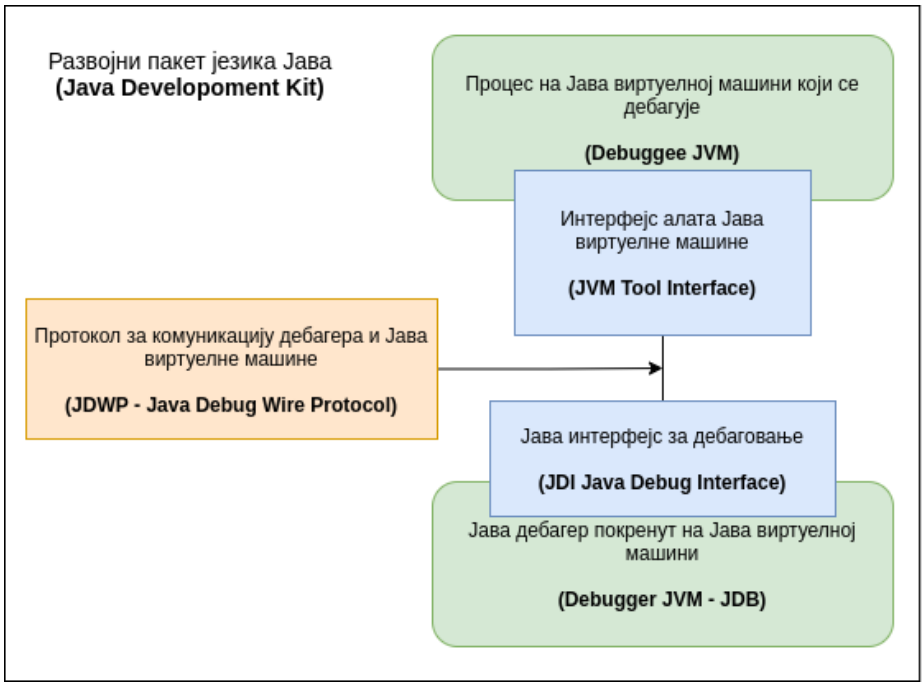
Слика 9: Пример коришћења *TDD* дебаговања у алату *WinDBG Preview*

и долази испоручен у оквиру развојног пакета језика Јава (eng. *Java SE Development Kit*). Овај алат имплементира такозвану архитектуру програма за дебаговање *Java* платформе (eng. *Java Platform Debugger Architecture*) која је приказана на слици 10. Ова архитектура састоји се из три главне компоненте [3]:

- Интерфејс Јава виртуелне машине, која омогућава преглед и дебаговање стања апликације која се извршава на виртуелној машини.
- Интерфејс за дебаговање, који пружа информације корисне дебагерима и сличним системима којима је потребан приступ стању виртуелне машине у току извршавања (често удаљено).
- Протокол који се користи за комуникацију између дебагера и Јава виртуелне машине коју дебагује. Постојање овог протокола може омогућити дебагеру да ради на различитим процесима истог система или на неком удаљеном рачунару.

Када се Јава дебагер покрене и проследи му се име класе, он покреће другу копију Јава интерпретера. *JDB* је, дакле, сам по себи такође Јава програм који се извршава коришћењем додељене копије интерпретера. Овај нови интерпретер учитава фајл који садржи наведену класу и зауставља се ради дебаговања пре извршавања првог бајт кода [7, Chapter 16]. Овај дебагер се такође може покренути тако да се накачи на процес који се већ извршава.

Након што је дебаговање започето, може се извршити велики број команди [7, Chapter 16]. Међу њима је *catch* команда помоћу које се може изазвати прекид извршавања сваки пут када се подигне изузетак. Још неке занимљиве опције које нуди овај дебагер су могућност позивања сакупљача отпадака, приказ искоришћености меморије као и промена тренутне нити извршавања.



Слика 10: Архитектура програма за дебаговања *Java* платформе

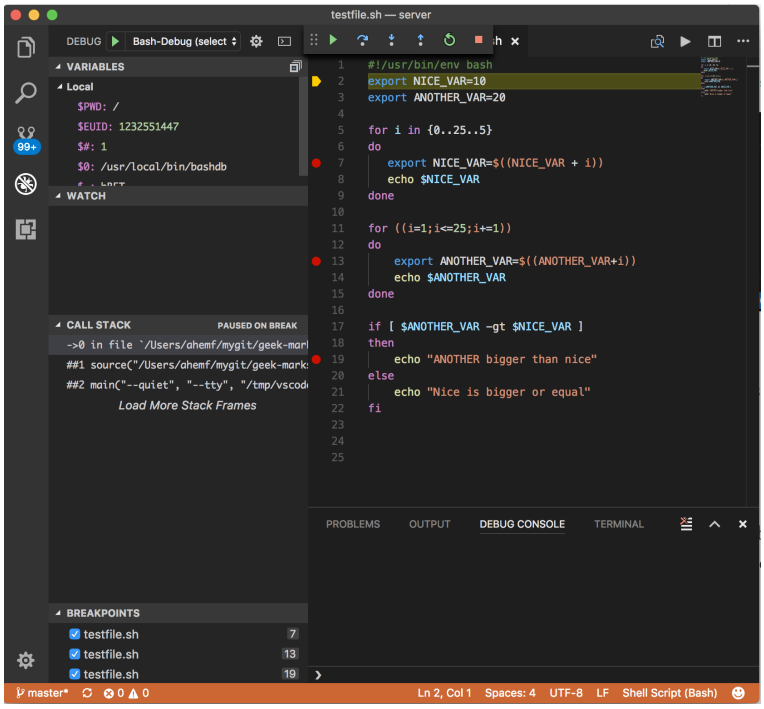
3.6 Bash debugger - bashdb

Основна употреба дебагера за скрипт језик јесте да омогући да се види шта се догађа унутар саме скрипте у току њеног извршавања. Иако је ово дебагер командне линије, за њега постоје прикључци који омогућавају коришћење кроз *Visual Studio Code* и кроз алате компаније *JetBrains*. Такође и алат *DDD* поменут у секцији 3.1, нуди подршку за коришћење овог дебагера кроз графички кориснички интерфејс [1]. На слици 11 приказан је пример коришћења алата *bashdb* у оквиру едитора *Visual Studio Code*.

BashDB има четири главна циља у помоћи за дебаговање *Bash* скрипти. Ти циљеви су:

- Покретање жељене скрипте и дефинисање свега што би могло утицати на понашање
- Заустављање извршавања скрипте уколико је испуњен дефинисани услов
- Испитивање шта се догодило након што је заустављено извршавање скрипте
- Измена саме скрипте, како би се исправили ефекти указаног бага

Функционалности које су подржане како би омогућиле извршавање ових циљева су могућност постављања тачака прекида, инспекција вредности променљивих, кретање уназад али и кретање кроз скрипту



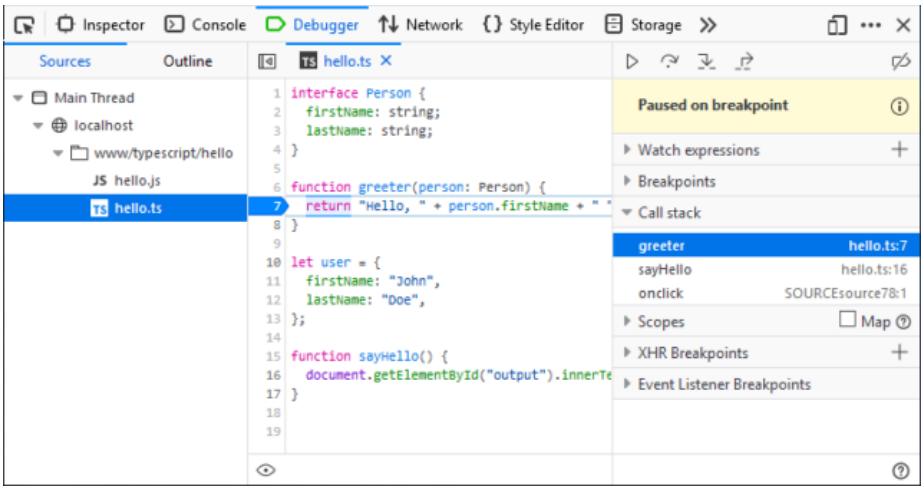
Слика 11: Пример коришћења *bashdb* дебаговања у алату *Visual Studio Code*

корак по корак [1]. Другим речима, пружа неке основне могућности које су за очекивати дебагерима за на пример програмске језике *C* или *C++*. Такође, подржана је и могућност постављања тачки посматрања да би се видело да ли и када се нека променљива мења. Постоји и команда `watche` која омогућава прекид извршавања кад год прослеђени израз буде евалуиран као тачан.

3.7 The Firefox JavaScript Debugger

Firefox алати за развој (eng. *Firefox Developer Tools*) представљају скуп алата који су уграђени у веб прегледач под називом *Firefox*. Састоје се од неколико различитих алата који омогућавају испитивање, мењање и дебаговање *HTML, CSS* и *JavaScript* кода. Међу њима је, поред веб конзоле, алата за мењање поравнања и садржаја, алата за праћење перформанси и других, и дебагер за програмски језик *JavaScript* [19].

Овај дебагер се може користити за дебаговање кода који се извршава локално, на истој машини, али и за дебаговање кода који се налази на некој удаљеној машини. То значи да је могуће користити *Firefox JavaScript* дебагер на локалном рачунару да се дебагује неки веб сајт или веб апликација која се извршава у другим прегледачима или окружењима за извршавање (eng. runtime), на пример на мобилном телефону који је повезан преко УСБ кабла. Пример коришћења



Слика 12: Пример употребе *JavaScript* дебагера у *Firefox* веб прегледачу

дат је на слици 12.

Као и већина претходно изнетих дебагера и *Firefox JavaScript* дебагер нуди мноштво функционалности за олакшавање проналажења узрока грешака. Неке од њих су омогућавање постављања тачака прекида на промене *DOM* стабла, затим паузирање извршавања када се деси неки очекивани догађај и паузирање када се подигне неки изузетак. Више о додатним могућностима може се видети на [20].

4 Сумарни преглед могућности

Сваки од дебагера изложених у поглављу 3 има своје предности, које потичу од специфичности система или програмског језика за који су првенствено намењени. Међутим, већина њих ипак дели одређени подскуп функционалности.

Како би се стекао осећај о старости, одржавању и томе за шта је намењен који дебагер, у табели 1 приказане су неке основне информације о сваком од изложених дебагера. У табели 2 дат је упоредни приказ особина које би могле бити корисне, а опет их дели већи подскуп изложених дебагера. Технике дебаговања које су описане у поглављу 2 издвојене су у посебну табелу 3 и приказане за сваки од дебагера шта од њих подржава.

5 Закључак

Комплекност и количина написаног софтвера расте из године у годину услед све већег броја рачунара који су у оптицају и који се уграђују у уређаје као што су аутомобили, авиони, телевизори, роботи и

Табела 1: Техничке особине изложених дебагера

Особина / Дебагер	Прва верзија	Платформе	Програмски језици	Последња верзија
GDB	1986.	*UNIX* системи, *Windows*	*C, C++, D, Fortran, Pascal, Modula-2, Ada*	Окт 2020.
LLDB	2012.	*Linux Windows macOS, iOS watchOS, FreeBSD*	*C, C++, Objective-C, Objective-C++*	Окт 2020.
MVSD	1995.	*Windows*	*C#, C++, Visual Basic, F#,* Python, JavaScript	Јан 2020.
WinDBG	?	*Windows*	*C, C++*	Апр 2020.
JDB	2000.	*Windows, macOS, Linux, Solaris*	Java	Сеп 2020.
BashDB	2003	*UNIX* системи	*Bash*	Дец 2019.
Firefox JavaScript дебагер	2011	*Windows Linux macOS*	*JavaScript, TypeScript*	Сеп 2020.

Табела 2: Упоредни реглед могућности изложених дебагера

Особина / Дебагер	Условне тачке прекида	Тачке посматрања	Евалуација израза	Праћење изузетака	Вишенитне апликације
GDB	✓	✓	✓	✓	✓
LLDB	✓	✓	✓	✓	✓
MVSD	✓	✓	✓	✓	✓
WinDBG	✓	✓	✓	✓	✓
JDB	✗	✓	✓	✓	✓
BashDB	✓	✓	✓	✗	✗
Firefox JavaScript дебагер	✓	✓	✓	✓	✓

слично. Откривање, анализа и отклањање багова додатно добијају на значају у жељи да се софтверски развојни процес учини максимално ефикасним и квалитетним као и да се минимизују трошкови и ризици. Дебагери су међу најважнијим алатима за остваривање ових циљева и интензивно се користе.

Ови интересантни алати често поседују ефекат који није реткост и за оперативне системе - бивају узети здраво за готово. Очекује се

Табела 3: Упоредни реглед техника које подржавају изложени дебагери

Дебагер \ Техника	Дебаговање уназад	Удаљено дебаговање	*Post-mortem* дебаговање
GDB	✓	✓	✓
LLDB	✓	✓	✓
MVSD	✓	✓	✓
WinDBG	✓	✓	✓
JDB	✗	✓	✗
BashDB	✗	✓	✗
Firefox JavaScript дебагер	✗	✓	✓

да су ту, да раде брзо и ефикасно, а ретко се обрати пажња на њихову комплекност, корисност и начин на који раде. Интересантно је приметити да је дебагер производ рада једног или више програмера, да му је улаз програм, а да је корисник дебагера првенствено и једино програмер. Односно, дебагер као резултат развојног процеса би требао да буде изузетно квалитетан јер сви они који учествују у његовом развијању поседују озбиљно разумевање својих корисника као и њихових потреба и проблема.

Из описаног издвојеног подскупа дебагера, изводи се закључак да иако сваки од њих има своје специфичности, ипак на крају већина ових алата пружа сличан скуп функционалности. То, уз чињеницу да су се аутори сваког од њих трудили да задрже сличан начин употребе, омогућава програмерима да, уз познавање једног од ових алата, могу једноставно и брзо да пређу на коришћење другог алата у тренутку када им је то неопходно. Ово важи како за алате командне линије, тако и за оне који имају и графичко корисничко окружење.

Литература

[1] BASH Debugger. on-line at: http://bashdb.sourceforge.net/.

[2] Debug your app. on-line at: https://developer.android.com/studio/debug.

[3] Documentation. on-line at: https://docs.oracle.com/en/.

[4] Time Travel Debugging - Overview. on-line at: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/time-travel-debugging-overview.

[5] S. K. Aggarwal and M. S. Kumar. Debuggers for programming languages. In Y. N. Srikant and Priti Shankar, editors, *Compiler Design Handbook: Optimizations and Machine Code Generation*, chapter 9. CRC Press, Inc., USA, 2002.

[6] David M. Beazley. Welcome to SWIG. on-line at: http://www.swig.org/.

[7] Benjamin J. Evans and David Flanagan. *Java in a Nutshell*. O'Reilly Media, Inc., 6th edition, 2014.

[8] Benjamin J. Evans, James Gough, and Chris Newland. *Optimizing Java: Practical Techniques for Improving JVM Application Performance*. O'Reilly Media, Inc., 1st edition, 2018.

[9] Free Software Foundation. DDD - Data Display Debugger. on-line at: https://www.gnu.org/software/ddd/.

[10] Free Software Foundation. GDB front ends and other tools. on-line at: https://sourceware.org/gdb/wiki/GDB%20Front%20Ends.

[11] Free Software Foundation. GDB: The GNU Project Debugger. on-line at: http://gcc.gnu.org/.

[12] National Geographic. World's First Computer Bug. on-line at: https://www.nationalgeographic.org/thisday/sep9/worlds-first-computer-bug/.

[13] Milena Vujošević Janičić. *Dinamička analiza*. Matematički fakultet, Univerzitet u Beogradu. on-line at: http://www.verifikacijasoftvera.matf.bg.ac.rs/vs/predavanja/03_dinamicka_analiza/03_dinamicka_analiza.pdf.

[14] llvm-admin team. The LLVM Compiler Infrastructure. on-line at: https://llvm.org/.

[15] Norman Matloff and Peter Jay Salzman. *The Art of Debugging with GDB, DDD, and Eclipse*. No Starch Press, USA, 2008.

[16] Microsoft. Debugging Using WinDbg Preview. on-line at: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-using-windbg-preview.

[17] Microsoft. Getting started with WinDbg. on-line at: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-using-windbg-preview.

[18] Microsoft. Visual Studio Debugger. on-line at: https://docs.microsoft.com/en-us/visualstudio/debugger/?view=vs-2019.

[19] Mozilla and individual contributors. Firefox Developer Tools, 2005-2021. on-line at: https://developer.mozilla.org/en-US/docs/Tools#debugging_the_browser.

[20] Mozilla and individual contributors. The Firefox JavaScript Debugger, 2005-2021. on-line at: https://developer.mozilla.org/en-US/docs/Tools/Debugger.

[21] John Robbins. *Debugging Applications*. Microsoft Press, 2000.

[22] The LLDB Team. The LLDB Debugger, 2007-2022. on-line at: https://lldb.llvm.org/.